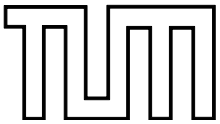
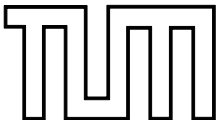


Department of Informatics
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Design, Prototypical Implementation and Evaluation of an Active Machine Learning Service in the Context of Legal Text Classification

Johannes Hermann Muhr



Department of Informatics
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Design, Prototypical Implementation and Evaluation of an Active Machine Learning Service in the Context of Legal Text Classification

Design, Prototypische Implementierung und Evaluation eines Active Machine Learning Dienstes im Kontext der Klassifizierung von Rechtstexten

Author:	Johannes Hermann Muhr
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Bernhard Waltl, M.Sc.
Submission Date:	15.07.2017

I confirm that this master's thesis is my own work and I have documented all sources and materials used.

Ich versichere, dass ich diese Masterarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, 10.07.2017 Johannes Muhr

Acknowledgments

First and foremost, I would like to thank my advisor Bernhard Waltl for the supervision of this thesis as well as for his great support and the interesting discussions during the last months. Additionally, I would like to thank Elena Scepankova for her support.

Furthermore, I would like to express my gratitude to Prof. Dr. Florian Matthes for his time, feedback and for providing me the opportunity to write this thesis at the chair Software Engineering for Business Information Systems (SEBIS).

Most of all, I would like to thank my family, my girlfriend Simone and all my friends for their absolute support during my master thesis and the last years in general.

Abstract

In the contemporary era, great quantities of legal texts are produced, stored digitally, and retrieved for work later, to the extent that manual classification of these documents, and the manual processing of the content, has become unfeasible. This study provides support for this business need by implementing a microservice (LexML) for legal document and norm classification, which applies the concept of active machine learning. Following the evaluation of possible solutions for (legal) text classification and (active) machine learning in the existing literature, LexML was implemented using Apache Spark MLlib as the machine learning framework. Within the scope of this study, the existing functionality of the legal data-science environment called Lexia was utilized. Various classifiers and query strategies were implemented and evaluated using German legal data. Overall, active learning strategies outperform traditional machine learning in terms of the speed of learning and maximum accuracy. The results of the document and norm classification experiments vary greatly: while for document classification, Naïve Bayes and Multilayer Perceptron outperform Logistic Regression, the latter is undoubtedly superior to the other two for norm classification.

Keywords: Active Learning, Machine Learning, Apache Spark MLlib, Legal Text Classification

Content

List of Figures

List of Tables

Listings

Abbreviations

AI	Artifical Intelligence
AL	Active Learning
ALE	Active Learning Engine
ASF	Apache Software Foundation
API	Application Programming Interface
ANN	Artificial Neural Network
BGB	Bürgerliches Gesetzbuch
BP	Back Propagation
BSD	Berkeley Software Distribution
DR	Dimensionality Reduction
DC	Document Classification
EGL	Expected Gradient Length
FS	Feature Selection
FR	Functional Requirement
HDFS	Hadoop Distributed File System
IDF	Inverse Document Frequency
JDBC	Java Database Connectivity
kNN	k-Nearest Neighbor
LR	Logistic Regression
ML	Machine Learning
MLP	Multilayer Perceptron
MNB	Multinomial Naïve Bayes
NB	Naïve Bayes
NN	Neural Network
NFR	Non-Functional Requirement
NC	Norm Classification
PL	Passive Learning
QBC	Query by Committee
RL	Random Learning
ROC	Receiver Operator Characteristic
REST	Representational State Transfer
RDD	Resilient Distributed Datasets
SVM	Support Vector Machine
TC	Text Classification
TF	Term Frequency

UIMA	Unstructured Information Management Architecture
US	Uncertainty Sampling
URI	Uniform Resource Identifier
UI	User Interface

1 Introduction

1.1 Motivation

> *"There seems to be little question that machine learning will be applied to the legal sector. It is likely to fundamentally change the legal landscape." [1]*

Digitization has already brought disruptive transformation to numerous business sectors, and it is likely that it will also substantially transform the legal domain, as the citation above predicts. The amount of data currently produced per day is immense, and it will increase considerably in the future. At present, 2.5 quintillion bytes of data are created daily from various sources, such as audio, video or text files. Furthermore, in 2015, 90% of the world's data had been created in the preceding two years[1]. This trend is reflected in the legal domain, where the amount of written information has increased exponentially, resulting in stresses to the legal system [2]. Thousands of legal documents such as judgments, contracts, patents, among others are produced by governments, firms, law offices and the like every day. In the era of globalization and digitization, these documents are usually made available online via the internet or stored in an internal database so that they can be shared, retrieved and easily utilized again later. For instance, many people working in the legal environment use online legal portals such as "juris"[2] and "beck-online"[3] in Germany, or "westlaw"[4] in the United States for their daily work. They all provide quick access to a vast selection of online legal texts. Additionally, these databases improve the public's accessibility to legislation.

Sharing legal data on a particular platform (e.g. on a website on the internet) is an important collective task. To retain the document overview and to retrieve these legal documents in a proper manner, it is essential that their categorical classification is correct. The task of analyzing and organizing legal texts can be understood as a *document classification* task. This document-level classification is in itself a complex task, and the amount of data that requires classification is considerable. Manual classification of such documents is time-consuming and costly, especially as legal knowledge is required. For instance, in the study of [3], attorneys engaged in litigation proceedings spent in excess of four months reviewing and classifying documents as either responsive or nonresponsive, resulting in a total cost of US$13.5 million.

Both the number of legal documents and the volume of their content is increasing. For example, in Germany alone in the 2009-2013 period, more than 500 new laws were introduced or updated[5]. Additionally, laws become increasingly complex in order to correspond to economic and social complexities [4]. Such complexity manifests in long sentences with complex clauses, cross referencing, and amendments such as changes in definitions over time [5]. Thus, while document classification enables people such as attorneys to find the relevant documents

[1] http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/
[2] https://www.juris.de
[3] https://beck-online.beck.de
[4] http://legalsolutions.thomsonreuters.com/law-products/westlaw-legal-research/
[5] https://www.bundestag.de/dokumente/textarchiv/2013/46598866_kw37_statistik/213446

relatively quickly via online databases, they may have problems finding the pertinent passages within the document or understanding it. To simplify and speed up the search for the relevant provisions within a legal text, it is advantageous to assign individual sentences of a document to particular classes (*norm classification*). For example, an attorney would then be able to rapidly ascertain the norms or paragraphs of the document in which a legal definition is to be found. Again, manual classification of a legal text's content is not practically achievable. For instance, the German Civil Code (Bürgerliches Gesetzbuch (BGB)) alone has 2,385 paragraphs and hence, a considerably greater number of sentences. Additionally, the person who would undertake such a classification of the text would be required to possess a robust knowledge of the domain. Consequently, labeling the legal document's content manually is unfeasible. To overcome the problem of having to manually classify documents and their content, one can use computer-aided support.

One way to achieve this is to use rule-based text annotation software like Apache UIMA Ruta[6] to classify documents or sentences. These methods analyze the text and check whether defined rules can be applied to the document or single sentences. For instance, one can define a rule to the effect that if "im Sinne des Gesetzes" occurs, the sentence is always classified into the class "legal definition". These rule-based classifiers perform very well in many cases. However, the problem is that minor variations (which are not defined in a rule) may result in misclassification. Furthermore, this approach requires comprehensive prior semantic analysis of the legal text to calculate linguistic patterns for each rule.

Another method of classification supported by computers is the application of various machine learning techniques. The drawback of traditional (supervised) machine learning techniques is that they often require a relatively large set of already labeled training instances to train the classification algorithm appropriately. As already indicated, for norm classification especially, this is very time-consuming and costly to achieve which makes it still hardly viable. Therefore, *active machine learning*, a technique aimed at creating a good classification algorithm with fewer training instances, is the focus of this research. Active machine learning in the text domain is a promising approach that has already been successfully applied to a number of tasks, including classification [6, 7], named entity recognition [8], and spam filtering [9], as well as in the legal domain [5, 10, 11] .

The objective of this study is to attain better insight into text classification and machine learning, especially active machine learning. Having gained deeper knowledge about these subjects, this understanding should be applied to develop an active machine learning microservice for legal-text classification using a carefully selected machine learning framework. In the final step, the quality of the service in the classification of legal documents and its norms are evaluated.

[6] https://uima.apache.org/ruta.html

1.2 Research Questions

In order to achieve these objectives, the thesis orients itself by means of the following research questions:

I. What are common concepts, strategies and technologies used for text classification?

Text classification is a comprehensive and complex task that includes several steps, such as preprocessing, transforming the text into a suitable format, feature selection, and the classification itself. With regard to classification, many classifiers with different advantages and drawbacks, depending on various factors, exist. The goal is to understand the text classification process and gain insights into existing approaches. Later in the work (see Chapter 3), machine-learning technologies used for text classification are introduced and analyzed.

II. How can (active) machine learning support the classification of legal documents and their content (norms)?

Once a basic understanding of text classification has been achieved, the next step is to introduce active machine learning. As with text classification, active machine learning is a complex method that requires the consideration of many different influencing factors. Achieving an overview of the concept of active learning and how these influencing factors can be managed are within the scope of this question. In the literature review, the term *textual data* is often used to refer to both entire documents as well as individual norms.

III. What form does the concept and design of an active machine learning service take?

The comprehensive literature review answers research questions one and two. The next step is to calculate the requirements for developing an independent active machine-learning prototype (microservice). This prototype should be able to conduct text classification and active machine learning using imported legal textual data. Following this, a suitable concept and design for a technical solution implementing these requirements is developed.

IV. How well does the active machine learning service perform in classifying legal documents and their content (norms)?

In the final step, the performance of the built prototype is evaluated in terms of the quality of classification of documents and their content. Common evaluation measures discovered in literature review are applied to compare different classifiers and active-learning strategies. That is, not only the final classification results are compared, but also the “learning”.

1.3 Research Method

The objective of this thesis is to report on the building of a prototypical implementation of an active machine-learning microservice. In order to be able to develop an appropriate concept for the implementation, this study is primarily oriented to the *design science* approach depicted by [12]. They describe the two concepts, *behavioral science* and *design science.* Figure 1 provides a conceptual overview that combines *design science* and *behavioral science* paradigms to understand and evaluate information systems (IS) research. In contrast to *behavioral science*, where the objective is the development and justification of theories explaining or predicting issues related to identified business needs, the aim of *design science* approach is to develop and evaluate artifacts created to meet these needs. Although no own research was conducted to identify specific business needs, they were calculated in the course of the literature research. As already indicated in Chapter 1.1, these business needs for the improvement of the current state of legal-text classification have been identified. To improve this situation, this work focuses on the design artifact. In doing so, a prototypical implementation of an active machine-learning microservice (i.e. the artifact) is developed and evaluated. The development is based on the theories, frameworks and methods discussed in a literature study. The built prototype is evaluated by means of data-analysis techniques and specific evaluation measures. The results obtained principally provide additions to the knowledge base, but may eventually also be applied in an appropriate environment to address the business needs identified.

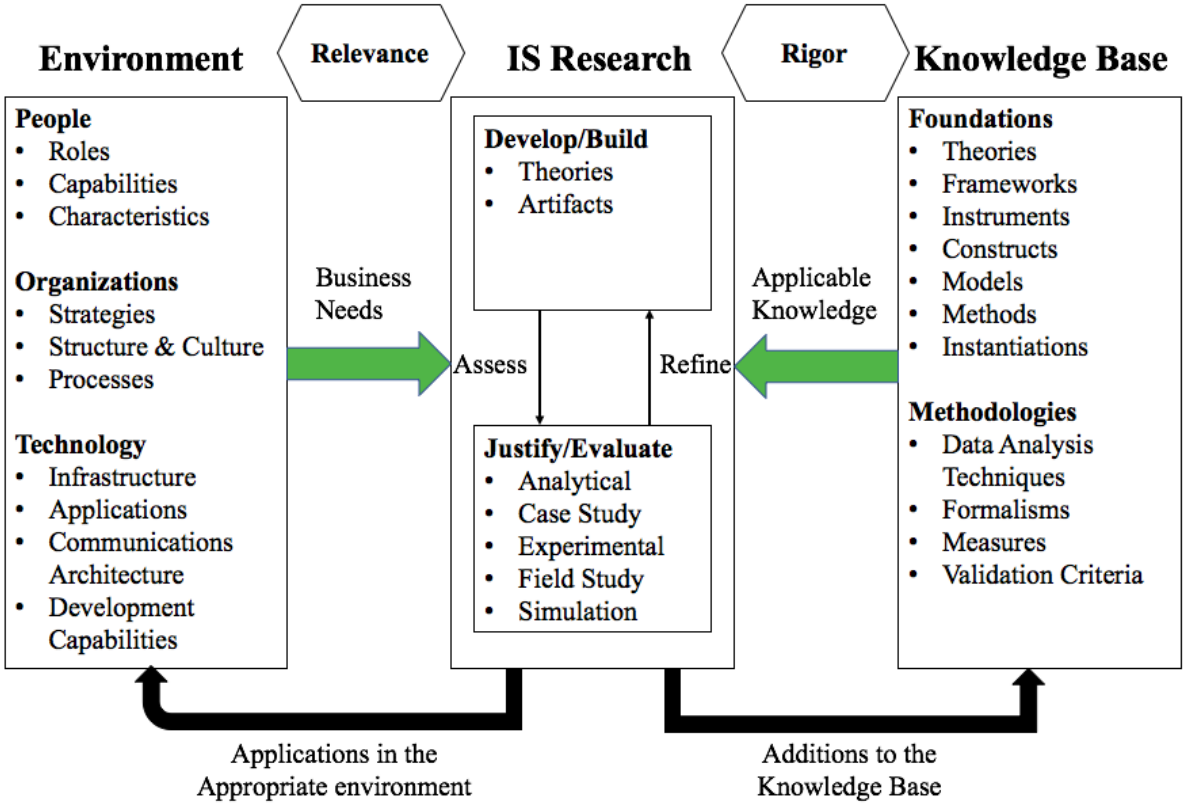


Figure 1: Information Systems Research Framework
Source: Own illustration based on [12]

To attain the basic knowledge required to develop the appropriate concept for the microservice, as a first step a comprehensive literature study was performed [13]. The results of the literature study have been the following:

- Develop an overview of machine learning, its various forms and processes, and the terminologies used.
- Learn how (legal) text classification can be applied and which factors must be considered.
- Understand how active machine learning works and which influencing factors must be taken into account.
- Discover and assess machine learning frameworks that can be used for text classification and active machine learning.

Several sources, such as conference papers, journals, blogs and books were consulted in the course of the literature review. To access the pertinent literature, online platforms such as the following were utilized:

- Google Scholar,
- Web of Science,
- Institute of Electrical and Electronics Engineers (IEEE),
- Online Public Access Catalogue (OPAC),
- and Google Books.

A variety of search queries were conducted using these online platforms, with the searches dependent on the platform's respective thematic areas (text classification, (active) machine learning, etc.). Additionally, relevant work within academic papers germane to the subject matter was investigated in greater detail (backwards search). The objective was to find relevant work from the past as well as current research studies. The identified papers were managed by means of a reference management system.

The results of the literature study were used as starting point to develop a suitable concept for an active machine learning prototype for the classification of legal documents and norms, implement the prototype, and evaluate it.

1.4 Outline

This thesis comprises three main parts: in the first, a comprehensive literature review on topics pertinent to the classification of legal texts is conducted. Based on this literature study, the second part deals with the development of a machine-learning prototype that is suitable for legal-text classification. In the third part, both the prototype itself and the classification results obtained are evaluated. The work is structured as indicated below.

First, in Chapter 2.1 the fundamental terminology and concepts of machine learning and the text-classification process are explained. These understandings form the basis for the discussion on active machine learning and the factors that influence it in Chapter 2.2. Both concepts are brought together and transferred to the legal domain in Chapter 2.3. As the basis for the practical part, the suitability of selected machine-learning frameworks for the prototype is assessed in Chapter 3. The most suitable one is used for the building of the prototype.

The development of the prototype is a component of the chapters that follow, the second main part. First, the concept and design are described in Chapter 4, addressing subjects like the

environment of the prototype, its requirements, and the resulting architecture. Then, important details about the implementation of the prototype are presented in Chapter 5.

Thereafter, the implementation of the prototype and the results obtained in classifying legal texts applying active machine learning are evaluated in Chapter 6.

Finally, the results relevant to the research questions from all three parts are discussed in Chapter 7.

2 Knowledge Base

2.1 Background

In this chapter, the terminology and concepts relevant to this study are explained and discussed.

2.1.1 Machine Learning

Machine learning (ML), a research area related to artificial intelligence (AI), is concerned with answering the question of how one can build computer systems that automatically improve their performance in a certain task through experience, or training data [14, 15, 16]. That is, ML uses a set of methods to detect patterns in data and then employs the patterns discovered to make predictions about future data [17]. ML was developed on the basis of the ability to use computers to analyze the structure of data, even if the manner in which the data is presented is not clear. ML could therefore be regarded as a "natural outgrowth of the intersection of Computer Science and Statistics"; it also incorporates approaches from neuroscience and related fields [14, p. 1].

Traditionally, ML is divided into two main categories:

- predictive or supervised learning
- descriptive or unsupervised learning

In predictive or *supervised learning*, a system learns a mapping of inputs x to outputs y based on given (labeled) input-output pairs $D = \{(\mathrm{X}, \Upsilon)\}$, known as a *training set*. This *training data* comprises instances of the input vectors along with their corresponding target vectors. In this manner, the *learning algorithm*[7] uses the input data along with the received correct output to teach itself a matching classification function (*classifier*) by comparing its actual output with the correct outputs in order to find errors. That means there is a function $f: \mathrm{X} \mapsto \Upsilon$ that, given the input data $x \in X$, returns a certain prediction $y \in Y$, a so-called *label* or *class*. The *label* y is usually a categorical or continuous value. Generally, there exist the following three different kinds of classification settings [18]:

- *binary classification*: y is associated with a single label l from a set of disjoint labels L and $| L | = 2$
- *multiclass classification*: y is associated with a single label l from a set of disjoint labels L and $| L | > 2$
- *multilabel classification*: y is associated with a set of labels $l \in L$ from a set of disjoint labels L and $| L | > 2$

In contrast to a label, each input variable x is a multidimensional vector representing a certain *feature* or *attribute* of the *instance* to be classified (*feature vector*).

[7] The term "learning algorithm" is often also used for the resulting classifier, depending on the context.

In descriptive, *unsupervised learning*, also known as knowledge discovery, the training data consists of input data vectors without any corresponding target value. The objective is to find patterns of interest in the data based on a given input set $D = \{X\}$ in which the input examples are not labeled. Here, the system does not know the correct answer, and a fully correct answer may not exist. Typical applications of *supervised learning* include classification and regression problems; however, the goal of *unsupervised learning* is to discover clusters and group the data into different categories based on the similarities discovered (*clustering*), or to determine the distribution of data (*density estimation*). In comparison to supervised learning, the problem is less clearly defined, the process not predictive but descriptive, and there is no obvious error metric [14, 15, 19].

The following two figures (Figure 2 and Figure 3) present and compare the basic workflow of the two ML approaches, first the supervised approach, followed by the unsupervised learning approach.

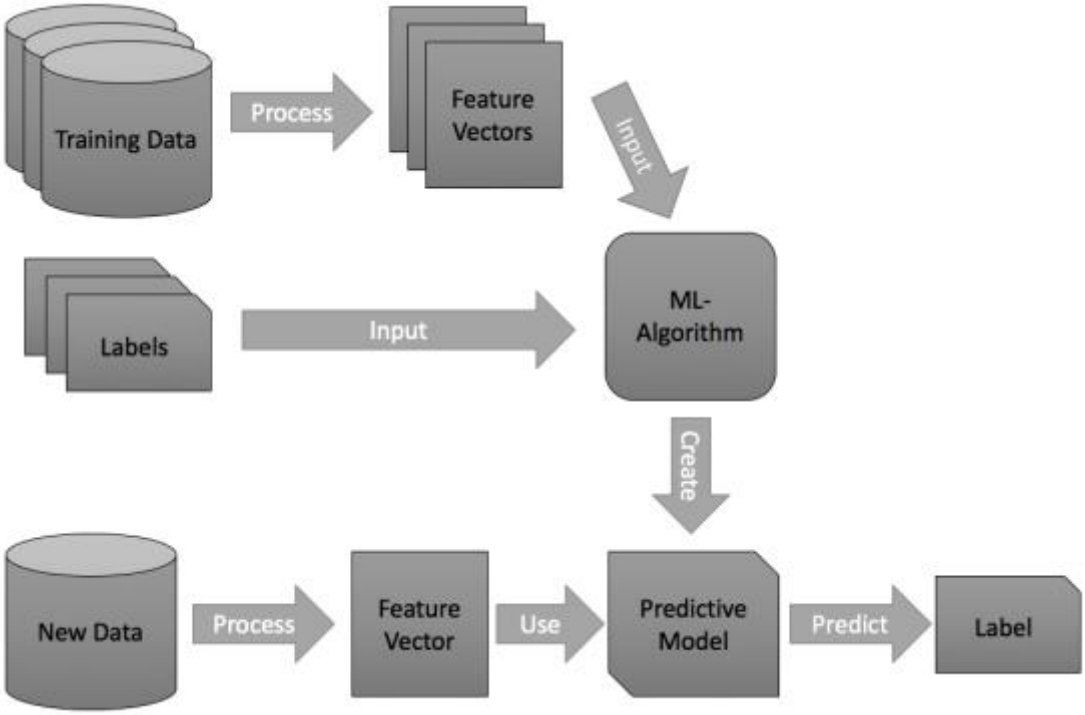


Figure 2: Illustration of the supvervised ML process
Source: Own illustration based on [20]

In both approaches, existing training data is first processed into several *feature vectors*. As these vectors serve as input for the ML algorithm, they are decisive for the resulting model quality. It is for this reason that appropriate preprocessing steps are often conducted in order to improve the quality of the feature vector (see Chapter 2.1.2.4). Depending on the context, feature vectors typically consist of a set of real numbers, integers or Boolean values (see Chapter 2.1.2.3).

In a next step, the feature vectors (and the corresponding *labels*, in case of supervised learning) are used by a ML algorithm to create a *predictive* (supervised) or *descriptive* (unsupervised) *model*. The selection of a suitable ML algorithm is another crucial yet difficult decision and remains a major research topic [14]. In addition to the strengths and weaknesses of each algorithm, influencing factors like the context of classification, the dependency of features, and

the distribution and noise of data must be considered in reaching this decision (see Chapter 2.1.2.6).

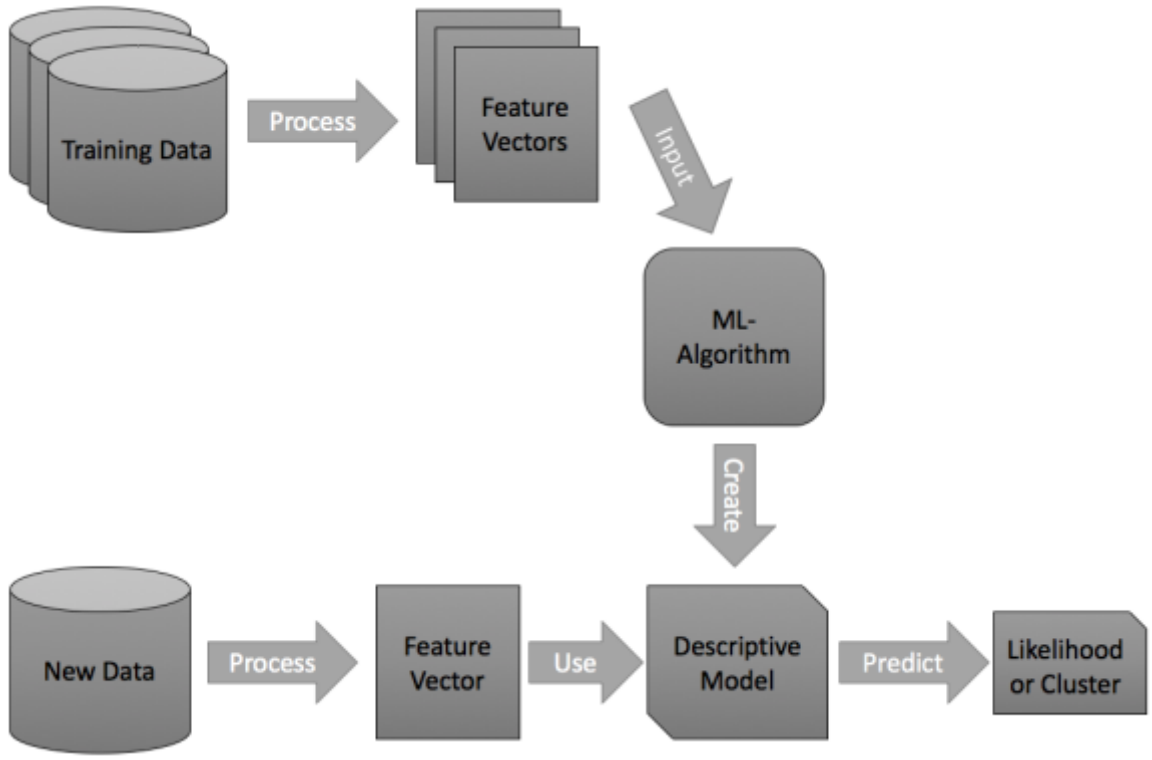


Figure 3: Illustration of the unsupervised ML process
Source: Own illustration based on [20]

The output of the ML algorithm generates the two models mentioned. They are the result of what is being learned from the data. Optimally, when adding new data, this model should then be able to either label instances of the new data correctly (supervised learning) or to categorize them in the appropriate cluster (unsupervised learning). This means that the re-usability of the model is a concern. As the model is mostly trained on very specific training instances (e.g. using a set of emails to train a model to recognize spam emails), it can only classify new instances in the same, or at least a similar, context (emails). Applying the model in another context, for example, to classify news into the categories "sport" or "no-sport" (similar to "spam" and "no-spam"), would result in failure [21].

A third form of ML that has garnered attention in recent years is *semi-supervised learning*. It is midway between *supervised* and *unsupervised* learning and uses both labeled and unlabeled data for training. In this case, the training set $D = \{(X, Y), (X)\}$ can be divided into two components: one for which labels are provided: $D \coloneqq (X, Y)$; and the other for which the labels unknown: $D \coloneqq (X)$. Semi-supervised learning is often useful if the costs of labeling data are excessive, or if the amount of data that needs to be labeled is unduly large [22, 23].

Important ML terminology and artifacts discussed in this chapter are summarized in Table 1.

Term	Definition
Data Set	A schema and a set of instances matching the schema
Training Set	A larger part of the data set used for training the classifier
Test Set	A smaller part of the data set used for checking the quality of the learned classifier
Instance/Example	A single object of the data, mostly described as a feature vector from which the model will be learned, or by means of which a model will be used for prediction.
Attribute/Field/ Variable	A quantity describing an instance and its value, depending on its domain type (categorical or continuous)
Feature	The specification of an attribute and its value (sometimes used as synonym for attribute)
Learning Algorithm/Learner	An algorithm that takes instances as input and produces a predictive model
Classifier	Mapping from unlabeled instances to (discrete) classes
Label/Class	A target class
Predictive Model	A model generated by learning algorithms that can be used as classifier. It has a structure and corresponding interpretation that summarizes a set of data
Unsupervised Learning	A learning technique used to learn relationships between independent attributes and labels
Supervised Learning	A learning technique that groups instances without pre-defined labels (e.g. clustering)
Semi-supervised Learning	A hybrid form of supervised and unsupervised learning using labeled and unlabeled instances

Table 1: Machine Learning Glossary
Own illustration based on definitions from [24]

Following this presentation of the basic workings of ML and the relevant terminology, the subsequent section demonstrates how ML is applied in practice in the context of text classification.

2.1.2 Text Classification

Text classification (TC) (also called text categorization; see Chapter 2.1.2.1) describes the problem of automatically assigning predefined categories to textual data based on their content [25, 26]. TC is a subfield of natural language processing (NLP) that has gained considerable importance as a result of the availability of the vast amounts of textual data available in electronic form, such as in digital libraries or from news articles. Typically, TC problems can be solved by applying supervised learning methods [26] in which a text classifier is built automatically by learning from a set of predefined (i.e. labeled) textual data [27, 28]. This learned model should then be able to classify unlabeled textual data with the correct label. Before addressing the TC process more thoroughly, the terms "classification" and "categorization" are defined and differentiated.

2.1.2.1 Categorization vs. Classification

While in the ML literature the terms *categorization* and *classification* are principally used as synonyms (e.g. [27, 29, 30]), Jacob [31] postulates a significant difference between the two terms.

Categorization is defined as "the process of dividing the world into groups of entities whose members are in some way similar to each other" [31, p. 518].

Classification describes three related concepts: (1) "a system of classes, ordered according to a predetermined set or principles and used to organize a set of entities;" (2) "a group or a class in a classification system;" (3) "and the process of assigning entities to classes in a classification system" [31, p. 522].

In principle, both *categorization* and *classification* pursue the objective of establishing order by grouping related appearances; the difference lies in how that order is achieved. Classification arranges entities into an arbitrary system and creates a taxonomy with non-overlapping classes. The result is that an entity can only be member of a single class. On the other hand, categorization divides entities into groups having some similarity within a given immediate context. This means the latter does not require predetermined definitions, but rather uses the existing context to cluster the entities into categories [31, 32].

Some of the major differences between the two terms are summarized in Table 2.

	Categorization	**Classification**
Process	Creative synthesis of entities based on context or perceived similarity	Systematic arrangement of entities based on analysis of necessary and sufficient characteristics
Boundaries	Fuzzy	Fixed
Membership	Based on generalized knowledge and context	Based on the intension of the class (either entity is member or not)
Criteria for Assignment	Context dependent and context independent	Criteria are predetermined guidelines or principles

Table 2: Differences between Categorization and Classification
Source: Own illustration according to [31]

The table illustrates that categorization is a more flexible process that draws non-binding associations between entities based on recognized similarities, while traditional classification strictly differentiates based on predefined rules.

In the context of ML, categorization may be understood as a type of unsupervised learning technique, such as clustering; whereas classification is related to supervised learning.

Strictly speaking, only binary- and multiclass classifications (but not multilabeling) can therefore be understood as traditional classification according to [31]. However, as the intention of this work is to conduct multiclass classification with clear boundaries generated by humans, the term classification is used in this work henceforth.

2.1.2.2 The Text Classification Process

Text classification is an extensive research topic that has been explored in combination with ML techniques in a number of fields, including *spam filtering* [33, 34], in which emails are assessed as to whether they are inappropriate and unrequested. Other applications include

sentiment analysis [35], by means of which the overall attitude (e.g. positive or negative) of a text is examined; and *named entity recognition*, where certain entities (e.g. the names of people) are extracted from a document [36, 37]. The basic classification process is similar for all applications.

Typically, the TC process consists of several steps (see Figure 4): (1) the text is preprocessed; (2) unnecessary features are removed and relevant features are selected by means of a feature vector; (3) the vector is used to train the classifier (learning algorithm) so that (4) new textual data can be classified accordingly [26].

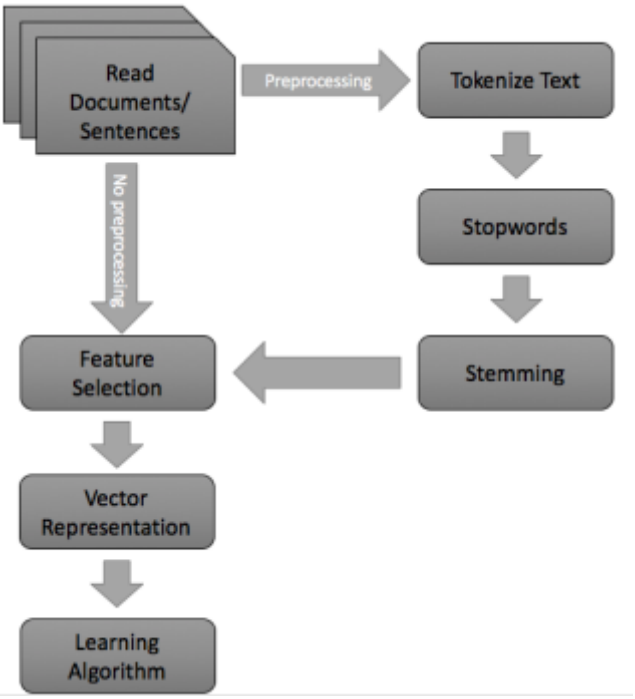


Figure 4: Illustration of the TC process
Source: Own illustration based on [38]

The individual elements of this process are explained in the following sections.

2.1.2.3 Text representation

When classifying textual data (e.g. a document or norms), it must be represented in a form that the ML algorithm can process [26]. A text is defined as a sequence of words that is usually represented as a vector of words or terms (*feature set*) [26]. This vector-space model assumes that each word is a dimension of the feature space and the dimension of the word (term) space is the number of words in the corpus (the *bag of words* representation). In this regard, the significance of a certain word in a document has to be considered and must be reflected by a word weighting to manage the following antagonism: on the one hand, the greater the frequency of a word, the greater its relevance to the topic; however, on the other hand, the word discriminates more poorly between documents [39]. To overcome this problem, several approaches have been developed (see Chapter 2.1.2.5) [40].

The most common approach to illustrate the weighting of a word in a document is the *term frequency*/*inverse document frequency* (TF-IDF). Here, the term (word) frequency (TF) states how often a certain term occurs in a text. The rationale behind the inverse document frequency (IDF) is that features that appear rarely in a collection of texts are particularly valuable. That is, the IDF of a feature is inversely proportional to the number of documents in which it occurs [41]. Hence, the TF-IDF approach weighs infrequent terms higher, as they are probably more informative for the learning algorithm. A disadvantage of TD-IDF is that the length of the text is not considered [26, 38]. The *TFC-weighting* is a similar approach that takes length into account and performs a length normalization as part of the word-weighting formula. Another method called *LTC-weighting* aims to reduce the effects of large differences in word frequencies by using the logarithm of the word frequency instead of counting raw word occurrences [39]. A completely different approach is used by [42] who used only a binary feature vector for expressing whether a term is present in the text or not.

An essential problem in the field of TC is the high dimensionality of the feature space (word vector) that cannot easily be managed by computational systems [39]. Hence, *feature reduction* is applied with the intention of removing irrelevant features and decreasing the dimensionality of the feature space. This usually compromises two methods: *feature extraction (preprocessing)* and *feature selection* [26, 38].

2.1.2.4 Preprocessing

The process of *preprocessing*, also known as *dimensionality reduction* or *feature reduction*, is applied to "make clear the border of each language structure and to eliminate as much as possible the language dependent factors" [43, p. 3814]. *Preprocessing* is an important step in the TC process, as redundant or irrelevant attributes can reduce both the speed and classification accuracy [38]. Uysal and Gunal [40] confirm this theory in their study, in which they examine the effect of *preprocessing* in TC tasks in two different text domains (news and emails) and languages (English and Turkish), although the actual necessity for *preprocessing* is controversial in the literature [44, 45, 46, 47]. They [40] also found that there is not one unique, perfect combination of *preprocessing* steps and tasks that provides promising results for every domain or language. To achieve a constant performance when creating a low-dimension vector, the setup must be analyzed and tested accordingly for each domain and language.

Typically, *preprocessing* consists of several steps (although it is not necessary to conduct all of them): (1) tokenizing the text, (2) removing stop words, (3) converting words into their root form (known as stemming) and (4) transforming words into lowercase characters [26, 40, 42, 43].

(1) *Tokenizing* is a form of text segmentation that describes the procedure of splitting a text (documents, sentences, etc.) into words, certain phrases, or context-dependent meaningful parts. Typically, non-alphanumeric characters like white spaces are used to separate these components in the text [26, 40].

(2) The objective of *stop-word removal* is to eliminate words that are assumed to be irrelevant for the classification process. Stop words are language-specific terms that are highly frequent, for example, articles or conjunctions such as "the" and "or" in English,

and “der” and “oder” in German [26, 38]. While [44, 45, 47] see no or only an insignificant improvement when eliminating stop words, [40, 46] enhanced their classification accuracy when eliminating stop words.

(3) In *stemming*, different forms of the same word (e.g. singular, plural or different tenses) are converted into a similar canonical form. This is, words are returned to their root form (e.g. connection to connect, computing to compute) [26, 38]. The significance of stemming has been questioned in various papers, for instance, [44, 45, 46, 47] recognized no significant improvement when using *stemming*. Furthermore, the significance of stemming can also be correlated to the language analyzed. For instance, in German, one word can have more forms than a comparable English word.

(4) It is assumed that whether words are uppercase or lowercase has no bearing on the results of text processing. Therefore, all uppercase words are usually converted to lowercase form. This supports both achieving eventually a better accuracy or at least a reduced feature size (see Chapter 2.1.2.5)[40].

2.1.2.5 Feature Selection

Feature selection (FS) describes the “process of selecting a subset of the features occurring in the training set and using those selected features in the text classification” [48, p. 283]. Typically, the main benefits are to (1) reduce measurement and storage cost, (2) avoid overfitting, and reduce noise and redundancy, (3) reduce training and utilization time, and (4) facilitate data understanding [42, 49, 50]. FS methods usually involve one of two different approaches – *wrappers* and *filters* – depending on the manner in which they select features from the vector”.

With the *wrapper* approach, various subsets of dependent features are built, evaluated and compared to a specific classifier (e.g. the recognition rate for a specified classifier). Wrappers are usually not suitable for TC as their application to large datasets is excessively time consuming [26, 38].

In the *filter* approach, features are treated independently. The core idea is to rank, and then select top-ranked, features based on a particular criterion calculated by a function [43]. The low-ranked features are removed from the vector. This approach is more suitable for TC: it is faster and simpler as the FS step is performed once and then the reduced feature vector may be used with any classifier [26]. Studies have demonstrated that none metric consistently performs better than others, as the result of the score (rank) is highly dependent on the type of classifier, and the sparsity and balance of the training data, among other matters [42, 50, 51, 52]. Some commonly used FS methods in TC are summarized in Table 3.

Method	Explanation
TD-IDF	Words occurring less than a defined threshold are removed from the word vector (see *Text representation*). As an unsupervised method, it does not require class labels
	$idf_i = \log \frac{\lvert D \rvert}{\lvert \#(f_i) \rvert}$
Information Gain (IG)	IG measures the decrease in entropy when the feature value is given. In other words, it measures the number of bits of information obtained for category prediction by calculating the presence or absence of a term in a document. Terms that have an IG below a predefined threshold are removed from the vector
	$IG(t) = H(C) - H(C\lvert T)$ $\sum_{c \in \{c_i, \bar{c}_i\}} \sum_{t \in \{t_k, \bar{t}_k\}} P(t,c) * \log \frac{P(t,c)}{P(t) * P(c)}$
Chi Square (χ^2)	By measuring the independence of a term in relation to a class, chi square identifies the ones that are most dependent on a certain class. Chi square is a normalized value and therefore comparable across terms for the same category
	$\chi^2(c_i, f_j) =$ $\frac{\lvert D \rvert \times \left(\#(c_i, f_j)\#(\bar{c}_i, \bar{f}_j) - \#(c_i, \bar{f}_j) \cdot \#(\bar{c}_i, f_j)\right)^2}{\left(\#(c_i, f_j) + \#(c_i, \bar{f}_j)\right) \times \left(\#(\bar{c}_i, f_j) + \#(\bar{c}_i, \bar{f}_j)\right) \times \left(\#(c_i, f_j) + \#(\bar{c}_i, f_j)\right) \times \left(\#(c_i, \bar{f}_j) + \#(\bar{c}_i, \bar{f}_j)\right)}$
Odds Ratio (OR)	OR is a measure of association between an exposure and an outcome. It is based on the assumption that the distribution of features is different in relevant and non-relevant documents. OR is the proportion of the word's occurrence in the positive class normalized by that of the negative class. As a result, features that occur rarely in a positive document but never in a negative obtain a relatively high score
	$OddsRatio\ f_i c_j = \log \frac{P(f_i\lvert c_j)\ (1 - P(f_i\lvert \neg c_j))}{(1 - P(f_i\lvert c_j))\ (P(f_i\lvert \neg c_j))}$
c	A class of the training set
C	The set of classes of the training set
D	The set of documents of the training set
t	A term or word
P(c) / P(c_i)	The probability of class c or c_i appearing in the training set
$P(\bar{c})$ $/P(\neg c)$	The probability of the class not occurring in the training set
P(c | t)	The probability of class c given that the term t occurs.
$P(\neg c)\lvert t$	Denoting the probability of class c not occurring given the term t occurs
P(c, t)	The probability of class c and term t occurring simultaneously
H(C)	The entropy of set C
#(c) #(t)	The number of documents which belong to class c or contain term t, respectively
#(c, t)	The number of documents containing term t and belonging to class c

Table 3: Filter Feature-Selection Methods
Source: Own illustration based on [25, 39, 42, 48, 52, 53]

Rogati and Yang [54] compared in their study these filter FS methods using four different classifiers; they conclude that a combination of metrics should be used. That is, Chi Square especially, in combination with IG or TF should be selected as FS methods as these had

constantly performed better across classifiers. Several common classifiers are described in the following section.

2.1.2.6 Classifiers for TC

Automatic TC in the area of ML is a comprehensive research field that has been extensively studied. As described in Chapter 2.1.1, there are two main methods – supervised and unsupervised – to classify textual data. For both approaches, various classifiers have been researched. Several surveys exist that provide a good overview of important learning algorithms [27, 38, 55, 56]. As mentioned in the previous chapter, there is no single best classifier. As a first step, for each application, factors like context, objectives and other influences must be analyzed. Thereafter suitable classifiers can be selected and tested.

Figure 5 provides a short overview or taxonomy of often-used TC classifiers. Generally, a clear distinction between classification algorithms is not simple, as they may have been modified or combined in various ways [38], or there may be special conditions in which classifiers behave differently. For example, the three linear classifiers presented in Figure 5 may also be implemented in a non-linear manner [55] (e.g. the Support Vector Machine (SVM) [57, 58]), or the Naïve Bayes may adopt a linear form. Those in blue are introduced in greater detail.

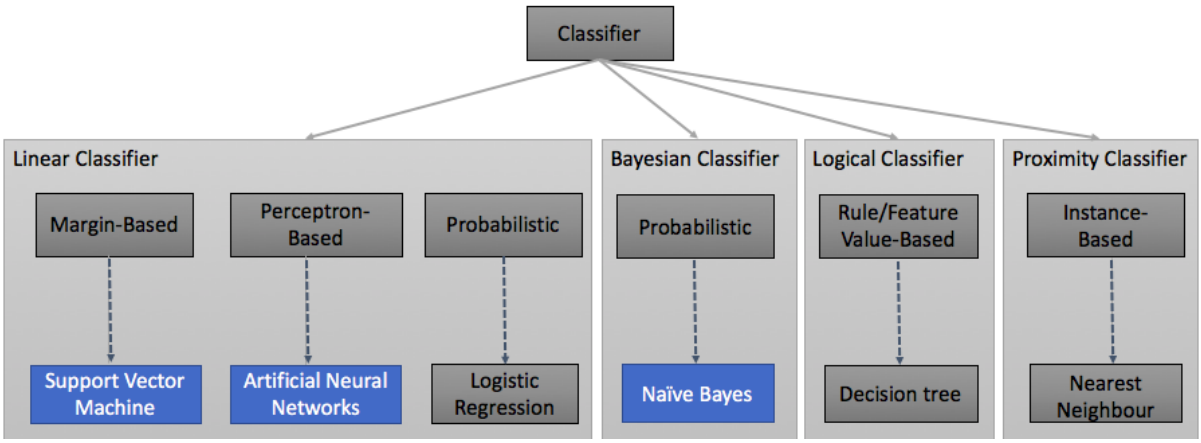


Figure 5: Classification of ML Algorithms
Source: Own illustration based on [55, 59, 60]

Generally, one can distinguish between linear and nonlinear classification problems. A classifier is linear if the feature space can be separated into different classes by a linear margin. Typical examples of originally linear classifiers are SVMs or perceptron-based techniques such as *artificial neural networks* (ANNs). The perceptron-based algorithm (single layer perceptron) is originally a binary classifier using a linear prediction function that multiplies a word vector (feature values) with a weight vector to yield the class label. Linear classifiers may also have a probabilistic basis, as does *Logistic Regression* (LR). LR uses the characteristic that the probability of observing the true label (logit transformation) can be modelled as a linear combination of the attributes [55, 59]. *Bayesian* Classifiers, such as the *Naïve Bayes* (NB), make use of the Bayes' theorem which assumes independence between the features. Although typically having a nonlinear decision boundary, it can also be modeled in a linear way.

Another statistical learning method is instance-based learning using distance (proximity) measures. A popular form of an instance-based algorithm is the k-*nearest neighbor algorithm* (kNN). It assumes that similar instances are close to each other as they have related properties. Several measures exist to calculate this relatedness, like the *Euclidean* or *Minkowsky Distance* [38].

Kotsiantis [59] compared existing theoretical and empirical studies and illustrated the strengths and weaknesses of the respective classifiers (see Figure 6). However, it bears reiterating that the performance of each classifier can vary significantly, depending on various factors. For instance, classifiers like the NB can yield excellent performance better than other classifiers [61].

	Decision Trees	Neural Networks	Naïve Bayes	kNN	SVM	Rule-learners
Accuracy in general	**	***	*	**	****	**
Speed of learning with respect to number of attributes and the number of instances	***	*	****	****	*	**
Speed of classification	****	****	****	*	****	****
Tolerance to missing values	***	*	****	*	**	**
Tolerance to irrelevant attributes	***	*	**	**	****	**
Tolerance to redundant attributes	**	**	*	**	***	**
Tolerance to highly interdependent attributes (e.g. parity problems)	**	***	*	*	***	**
Dealing with discrete/binary/continuous attributes	****	***(not discrete)	***(not continuous)	***(not directly discrete)	**(not discrete)	***(not directly continuous)
Tolerance to noise	**	**	***	*	**	*
Dealing with danger of overfitting	**	*	***	***	**	**
Attempts for incremental learning	**	***	****	****	**	*
Explanation ability/transparency of knowledge/classifications	****	*	****	**	*	****
Model parameter handling	***	*	****	***	*	***

Figure 6: Comparing learning Algorithms (**best, *worst)**
Source: [59]

Below, the algorithms (1) NB, (2) SVM, and (3) ANN are introduced in greater detail.

(1) Naïve Bayes

Naïve Bayes is a simple and commonly used probabilistic generative[8] classifier applying the Bayes' theorem. The algorithm is naïve as it assumes strong independence between the single

[8] Generative models are full probabilistic models of all variables. In other words, they learn the joint probability distribution (e.g. p(x,y)) [62].

features. This means that the occurrence of a certain feature (e.g. a word) does not influence the appearance of another word.

The probability that feature d belongs to class c can mathematically be illustrated by

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

whereby P(d) is a constant, as each feature has the same probability of being in the dataset [63]. The difficulty lies in estimating P(d|c), as there are a vast number of possibilities [60].

For the classification itself, usually two models, the *Multivariate Bernoulli Model* and the *Multinomial Model* (Multinomial Naïve Bayes (MNB)) are used in practice. Both calculate the posterior probability of a feature (word) belonging to a class based on the distribution of the word in the text, while ignoring the position of the word. The feature is then assigned to the class having the highest posterior probability [55, 60].

The *Multinomial Model* incorporates the frequency of a word by means of a vector containing a count of the words in the textual data (the *bag of words* representation). Then, for each class, each text is modeled by means of a multinomial distribution of words. Thus, the conditional probability of textual data given a certain class, is the product of all probabilities of the observed words in that class [55, 60].

In contrast, the *Multivariate Bernoulli Model* does not quantify word frequencies, but uses a binary vector to ascertain the presence or absence of a word. The probability of textual data given a certain class, is then the product of the probability of the feature values for all words [55, 60].

McCallum and Nigam [64] compared these two models and asserts that the *Multinomial Model* regularly outperforms the *Multivariate Bernoulli Model* and is therefore used more frequently in research. For this reason, it is also used in this thesis. Hence, in this study the abbreviation NB refers to the Multinomial Model of the Naïve Bayes.

Although feature independence is often not given in textual data and NB has the lowest accuracy according to [27, 59], [63] show that even in the presence of strong feature dependencies NB performs reasonably well, and outperforms other classifiers such as decision trees or kNN. Various TC studies confirm that NB classifier performs reasonably well despite its simplicity [61, 63]. Rennie, Shih, Teevan [65] introduced a modified *Transformed Weight-normalized Complement Naïve Bayes* (TWCNB) in their work and achieved the same accuracy as SVMs. A further advantage of the NB classifier is its short computational training time and ease of implementation.

For further reading, refer to [64] and [60] where more extensive overviews of the Bayesian classifier are provided.

(2) Support Vector Machine

An SVM is a frequently used discriminative[9] TC classifier. SVMs have sound theoretical foundations, and have achieved good results in empirical research.

Originally, an SVM is a linear classifier that assumes that the training data is linearly separable. The basic proposition is to create a hyperplane through a multidimensional *input space* **Z** given by the feature representation $\vec{x}$ of an instance x. The optimal hyperplane is the one having the maximum distance (margin) from the feature vectors. In other words, the goal of the algorithm is to maximize the distance between the hyperplane and the instances on each side [59]. Given a labeled training data set $(y_1, x_1),\ldots,(y_n, x_n)$, this optimal hyperplane can be defined as

$$(w_0, x_1) + b_0 = 0$$

whereby w is a weight vector, and b the offset [66].

The feature vectors can then be classified according to the following two equations [59]:

$$x_i \cdot w + b \geq +1 \quad for\ y_i = +1$$

$$x_i \cdot w + b \leq -1 \quad for\ y_i = -1$$

resulting in the following general SVM classifier:

$$f_{\vec{w},b}(x) = sgn(<\vec{w}, \vec{x}> + b)$$

Hence, the optimization problem is to minimize the squared norm of the separating hyperplane. This can be seen as a convex quadratic programming problem [59]:

$$Minimize\ \frac{1}{2}\ ||w||^2 \qquad by\ adapting\ w\ and\ b$$

Figure 7: Possible Hyperplanes of an SVM illustrates a binary classification setting (possible labels $y_i \in \{-1,1\}$) with three possible hyperplanes: A, B and C. All instances lying below the hyperplane (blue dots) belong to class -1; the others (green crosses) to class 1. The instances lying closest to this hyperplane are called *support vectors* (framed with black circles in Figure 7). Only these data points are used for optimization (minimization problem). The other instances are ignored. In this example, although all three hyperplanes split the two classes, A is the best separator as the distance from the support vectors to the hyperplane is the greatest.

[9] Discriminative models directly model the conditional probability distribution, without assuming anything about the input distribution (e.g. p(y|x)) [62].

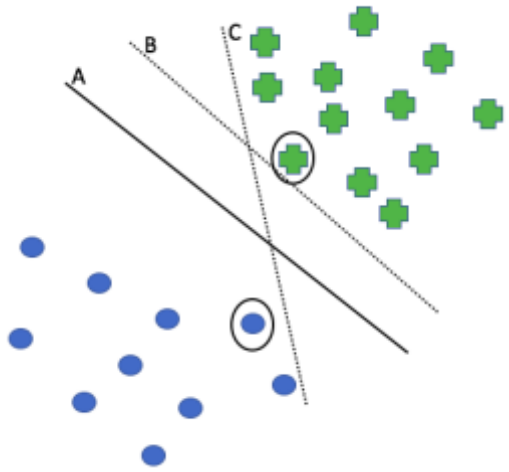


Figure 7: Possible Hyperplanes of an SVM
Source: Own illustration based on [55, 67]

As previously stated, SVM classifiers can also handle nonlinearly separable problems by mapping the data onto a higher dimensional space (*transformed feature space* or *Hilbert space* ***H***), using a converting function Φ [59, 68]:

$$\Phi : Z \rightarrow H$$

The result is that a nonlinear problem in the original *input space* is now equivalent to a linear separation in the *Hilbert space* [59]. Hence, the nature of the training algorithm would depend on the data points through dot products, for example, functions such as $\Phi(x_i) * \Phi(x_j)$ [59, 68]. Thus, by using a *kernel function* **K** in the form of

$$\mathrm{K}(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j),$$

an explicit determination of Φ by the training algorithm is not necessary, leading to the following nonlinear general decision function:

$$f(\mathrm{x}) = \mathrm{sgn}\left(\sum_{i=1}^{n} y_i \alpha_i \cdot \mathrm{k}(\mathrm{x}, x_i) + b\right),$$

where x represents support vectors with non-zero α_i values [69].

Kernels can be classified into the two main classes: *stationary* kernels (e.g. a Gaussian Kernel as an example of a radial basis function), and the most general form, *nonstationary* kernels (e.g. a Polynomial Kernel) [70]. Souza [71] provides a good overview of many possible kernel functions, such as the often-used *linear* kernel or *sigmoid* kernel.

Although SVMs are defined for binary problems, there are approaches for multiclass classification. The basic proposition is to reduce the multiclass problem into a set of binary problems. One way to do this is the *one-against-all* method in which one class is trained with positive labels, and all others with negative labels. Alternative methods are, for example, the *one-against-one* method, or the *directed acyclic graph* SVM [72].

One advantage of SVMs is their robustness when confronted with high feature-dimensionality, as they search for a suitable combination of features and only support vectors are included in the learning algorithm. This the reason they mostly yield high accuracy and good results in TC. One drawback is that, as they include an optimization problem, they come with time-consuming computations as a quadratic programming problem has to be solved [55, 59].

For further information, refer, for example, to Vapnik's book (e.g. [67]) or to Burges [68], who provide an extensive tutorial on SVMs.

(3) Artifical Neural Networks

An *Artificial Neural Network* is a discriminative linear classifier related to SVMs [55]. The basis of the network is a so-called *neuron* or *unit* designated as either input or output. When an input unit receives a set of inputs denoted by a vector ($\overrightarrow{\boldsymbol{X_i}}$), it is propagated through the network using the edges between the nodes (units). Additionally, there are weights, **A**, associated with each neuron, or rather on the edges between the individual neurons.

In order to predict the class label, various decision functions can be applied. The simplest form of an NN is based on a linear function as follows [55]:

$$p_i = A \cdot \overrightarrow{X_i}$$

$$\overrightarrow{X_i} = (t_1, f_1) \dots (t_n, f_n)$$

where $f_1, \dots, f_n$ are normalized frequencies of the word's occurrence in the text [73]. The *perceptron* learning algorithm, proposed by [74] [75], is a simple binary algorithm making use of this linear function. At the beginning, all weights $(w_o, \dots, w_n) \in A$ have the same positive or a random value. During the learning process, the weights are modified (increased or decreased by the *learning rate* α) in case a misclassification had taken place (*additive weight-updating*) [27]. Hence, the goal of a perceptron is to find a set of weights so that

$$\sum_i^n w_i * f_{ij} \geq \Theta \quad \equiv \quad x_i \cdot w + b \geq 0 \qquad for\ y = +1$$

$$\sum_i^n w_i * f_{ij} \leq \Theta \quad \equiv \quad x_i \cdot w + b \leq 0 \qquad for\ y = -1$$

where y represents the respective class labels, Θ is a defined threshold, and b the bias (Θ = -b) [73, 76]. This definition is similar to the one for SVMs.

The *Positive-* and *Balanced Winnow* [77] are other linear algorithms similar to the perceptron. However, in contrast to perceptrons they use *multiplicative weight-updating algorithms*. As the name indicates, this algorithm does not add or subtract the weights by α, but multiplies them [78]. All three learning algorithms have shown good results in practice, especially when noise and irrelevant features are present [27].

Instead of having a step function with the output {0,1}, it is possible to use a linear transfer function like the *logistic function* σ[10] to get continuous output. Typically, real numbers between [-1,1] are used, resulting in the following output: $\sigma(x_i \cdot w + b)$. This has the positive effect that weight modifications do not have such grave effects. The logistic function σ can be defined as

$$\sigma = \frac{1}{1 + e^{-n}}$$

where $n = x_i \cdot w + b$ is the linear combination of input features [79]. For that step, ANN are the same as a LR [80]. The main difference between the two algorithms lies in the optimization [81].

Figure 8 illustrates two possible variants of ANN: on the left, a single-layer perceptron consisting of two layers, input and output; and on the right, a feed-forward (where no cycles are allowed) multilayer perceptron (MLP). In additional to the input and output layers, MLPs have one or more hidden layers. The nodes of the "upper layer" are usually connected to all the nodes of the "lower layer" so that numerous combinations are possible. Though weights of the multlayer perceptron are not illustrated here, each edge has a certain weight.

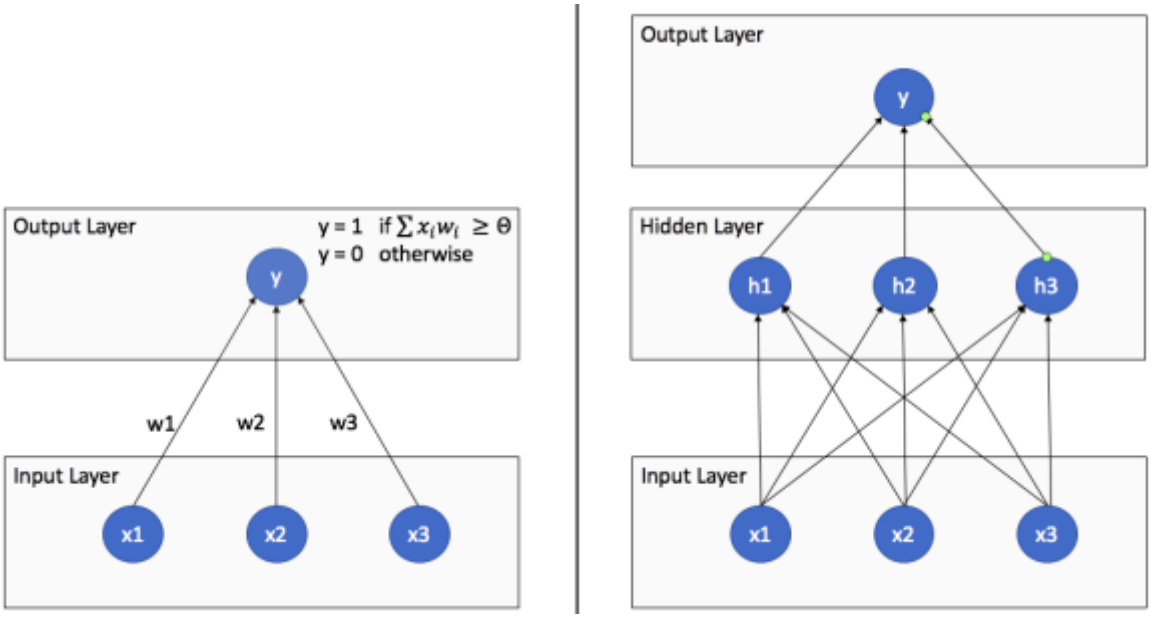


Figure 8: Illustration of a Single-layer Perceptron (left) and a Multilayer Perceptron (right)
Source: Own illustration based on [55]

In a manner similar to SVM, the use of these hidden layers allows the modeling of nonlinear relationships between the input and output variables. Each higher level represents input based on the feature combinations of the previous levels [79]. The number of nodes in such a hidden layer is not easy to define as on the one hand, insufficient neurons can result in poor

[10] This is sometimes also called a sigmoid function.

generalization and approximation capabilities. On the other hand, too many nodes lead to overfitting and can make the search for the global optimum more difficult. Among other reasons, this is why neural networks usually implement some type of stopping criterion [59].

The most widely used learning algorithm for MLPs is *Back Propagation*. This is an iterative process, that, at a basic level, functions in the following manner: in the forward phase, weights are fixed and training instances are propagated through the network. In the backward phase, the desired output is compared to the actual output at each output neuron to calculate the error rate. Following this, the local error and a scaling factor are calculated for each node on the hidden layer. In a next step, for each neuron in the network, all weights are adjusted so that the actual output better approximates the desired output. This is especially challenging for the hidden nodes. Hence, as the process continues, neurons connected with a stronger weight gain more responsibility [59, 76].

Weights are updated based on this general rule [82]:

$$\Delta W_{ji} = \alpha \delta_j O_i$$

where O_i is the output computed by neuron i and

$$\delta_j = O_j(1 - O_j)(T_j - O_j) \qquad for\ output\ neurons$$

$$\delta_j = O_j(1 - O_j)\left(\sum_k \delta_j W_{kj}\right) \qquad for\ internal\ (hidden) neurons.$$

As with SVMs, neural networks have the advantage of robustness when confronted with high feature-dimensionality: they can also manage a wide range of distributions accurately. However, this characteristic can also lead to overfitting and a lack of generalization capabilities in the trained model [81]. Another disadvantage is that, due to the back propagation algorithm, the learning algorithm may be time consuming to complete.

In summary, although, according to [27, 59], ANNs cannot achieve the same accuracy as SVMs, they still function well in the context of TC.

For a more detailed introduction, refer to Haykin [76].

Several performance measures have been proposed to test the performance of classifiers. The most common ones are introduced in the following section.

2.1.2.7 Performance Measures

The evaluation of a textual classifier's performance is typically conducted experimentally, rather than analytically. The problem with an analytical performance evaluation is that a formal specification of the problem is not given in most cases. Hence, usually the *effectiveness* (not efficiency), meaning the ability to classify the text into the correct category, is evaluated [27].

To determine the *effectiveness*, *precision*, *recall*, as well as the harmonic mean of recall and precision, called F_1, as well as *accuracy* are generally used. The basis of these measures is a

contingency table (also called *confusion matrix*). In this matrix, the four possible classification results are illustrated. These four terms are used to compare the classification assigned to an item (e.g. the predicted label of a word or document) with the desired correct classification (the class the item actually belongs to). Table 4 shows an example of a confusion matrix for a binary classification scenario with the two categories "A" and "B". To the left of the table, the four alternative classification results are explained.

If the outcome of a prediction is positive and the actual label is positive, then it is called a *true positive* (TP); however, if the actual label is negative, it is said to be a *false positive* (FP). A *true negative* (TN) prediction occurs when negative examples are correctly predicted to be negative, and a *false negative* (FN) when positive examples are incorrectly labeled negative.

		True Class	
		A	B
Predicted class	A	TP	FP
	B	FN	TN

Table 4: A Confusion Matrix
Source: Own illustration

Table 5 explains this evaluation technique when transferred to the context of TC.

Class	Definition
TP	Textual data that is correctly assigned to a category by the system
FP	Textual data that a system incorrectly assigned to a category
FN	Textual data that is not assigned to a category but should be
TN	Textual data that should not be marked as being in a particular category and is not

Table 5: Evaluation of the Classification of a Document
Own illustration based on [39, 42, 48]

Based on these definitions, the following evaluation criteria can be defined:

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$F_1 \text{ (F-score)} = \frac{2*Precision*Recall}{Precision+Recall}$$

$$\text{Accuracy} = \frac{TP+TN}{P+TN+FP+FN}$$

Additionally, in a non-binary classification problem, micro- and macro-averaging are used to calculate *precision* and *recall* across a set of categories [39]. In *macro-averaging*, *precision* and *recall* (or other performance measures) are first calculated for each category, and then the results are averaged across all different categories to compute the global means. In *micro-averaging*, the total number of correct and incorrect predictions (TP, FP, FN, TN), including all categories, are calculated first and then used to compute performance measures such as *precision* and *recall*. Hence, while *micro-averaging* gives equal weight to every document,

macro-averaging gives equal weight to each category, and therefore has the ability to perform adequately on categories with low generality [27, 39] .

Another way to evaluate and visualize the performance of a classifier is *Receiver Operator Characteristic* (ROC) curves. A ROC curve is a two-dimensional depiction of classifier performance. It plots the TP-rate on the y-axis against the FP-rate on the x-axis to confront the correctly classified positive examples with the incorrectly classified negative instances. One advantage compared to other evaluation measures such as *precision* and *recall* is that ROC curves decouple classifier performance from class skew and error costs [83]. Another common method to compare the classifier performance is to calculate the *Area Under* the ROC curve (AUC). This measures the quality of the classification averaged across all possible probability thresholds, expressing the performance in a single scalar value between 0 and 1 [84, 85].

When there are *n* classes (multiclass classification) – that is, more than two – the situation becomes more complex as the *n* * *n* confusion matrix then has *n* correct classifications (diagonal entries), and n^2-n possible errors (off-diagonal entries). One solution for this issue is to produce *n* different ROC graphs (one for each class). To compare the performance of classes, one class is as assumed to the positive class (*reference class*) and all other as the negative class [83]. The same problem applies for AUC for multiple classes and has been addressed by, for example [86, 87]. The latter used an approach similar to ROC and computed the AUC by using the weighted average of the AUCs obtained when taking each class as a *reference class*. The weight of a *reference class* AUC is their frequency in the data.

2.1.2.8 Summary

This chapter aimed to answer the first part of research question one: What are common concepts, strategies and technologies used for text classification? To this end, the basic concepts, approaches and terms of TC in the context of ML have been discussed (see Table 6).

TC is one of the central research fields in ML as it facilitates various applications, such as spam filtering and sentiment analysis. Typically, based on a labeled training set, classifiers like SVM or Naïve Bayes attempt to create a suitable mapping function from textual data to preexisting classes or labels. For this purpose, the textual data must be processed and developed into an adequate format for further processing.

While there is a vast amount of textual data available online, the problem in supervised TC is that the classifiers need large amounts of labeled training data to perform well. This labeling process incurs much time and effort as it involves reading the text and determining the correct category it should be assigned to. While the goal of ML typically is to learn automatically without human assistance or intervention, *Active Learning* or sometimes also called *Active Machine Learning* (AL), is an ML strategy that utilizes human support to reduce the number of training examples needed for classifier training and hence, for the labeling effort. The following chapter provides a more detailed overview of AL, and aims to answer research question two.

Keyword	Summary

Text Classification	Describes the problem using supervised ML methods to automatically assign predefined categories to text documents based on their content.
Text Representation	A text is usually represented as a vector of word terms. Often, the TF-IDF form is used as the representation form to include the weight of a certain word.
Preprocessing	Different methods, mostly tokenizing, stop-word removal, stemming and lowercase conversion are applied to remove redundant or irrelevant attributes.
Feature Selection	Based on certain feature-selection methods (e.g. IG, TF-IDF), features that are below a defined threshold are removed from the vector and not used for classification.
Classifier	A variety of classifiers, including modified, have been proposed in theoretical and practical research. Although there is evidence that some have, in principle, a higher accuracy than others, there is no one best classifier that can handle all influencing factors. In certain settings, even simple classifiers can perform better than complex ones.
Performance Measures	The performance of a classifier is usually evaluated in terms of its effectiveness for precision and recall, or by ROC curves and the AUC.

Table 6: Text Classification Concepts
Source: Own illustration

2.2 Active Machine Learning

This section provides a general review of the core ideas, concepts, methods and issues in AL based on existing literature. For a more comprehensive review, refer to Settles [88, 89], and Olsson [90].

2.2.1 Basic Concept

The key hypothesis of AL is that if the learning algorithm (in AL often called a learner) can select the data from which it learns (i.e. the most informative data), it will perform well with a small training set. As we saw in the previous chapter, supervised ML algorithms require large sets of labeled data which, especially in (legal) TC, is very expensive and time-consuming to produce. In contrast, AL does not necessarily assume a pre-existing training set (a *Seed Set*)[11], but creates the training data in an interactive loop with the support of an "oracle" (e.g. human expert). More precisely, the system asks queries in the form of unlabeled instances to be labeled by the human annotator. Those labeled instances are then used to train the predictive model and the loop recommences (see Figure 9). As a result, it is likelier that the classifier will perform well although it uses a minimum of labeled instances, thereby reducing the cost of the classification process [88].

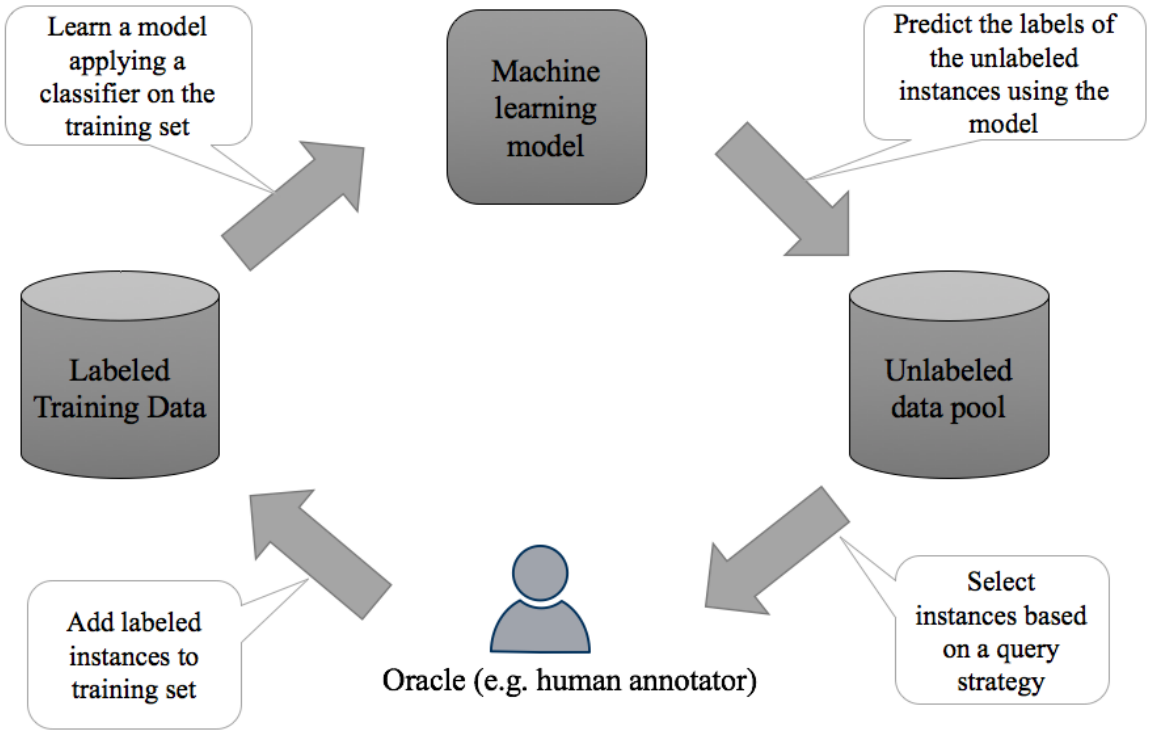


Figure 9: The Pool-based Active Learning Process
Source: Own illustration based on [88]

[11] It is also possible to use an existing labeled training set as seed set, and assist the model with the unlabeled instances in cases where it errs in assigning a label to a class.

Based on this concept, several AL scenarios and frameworks have been considered in literature. The main scenarios and query frameworks, as well as further terminology, are introduced in the following sections.

2.2.2 AL Scenarios

In the literature, three main scenarios have been proposed for how the learner can access the data (see Figure 10).

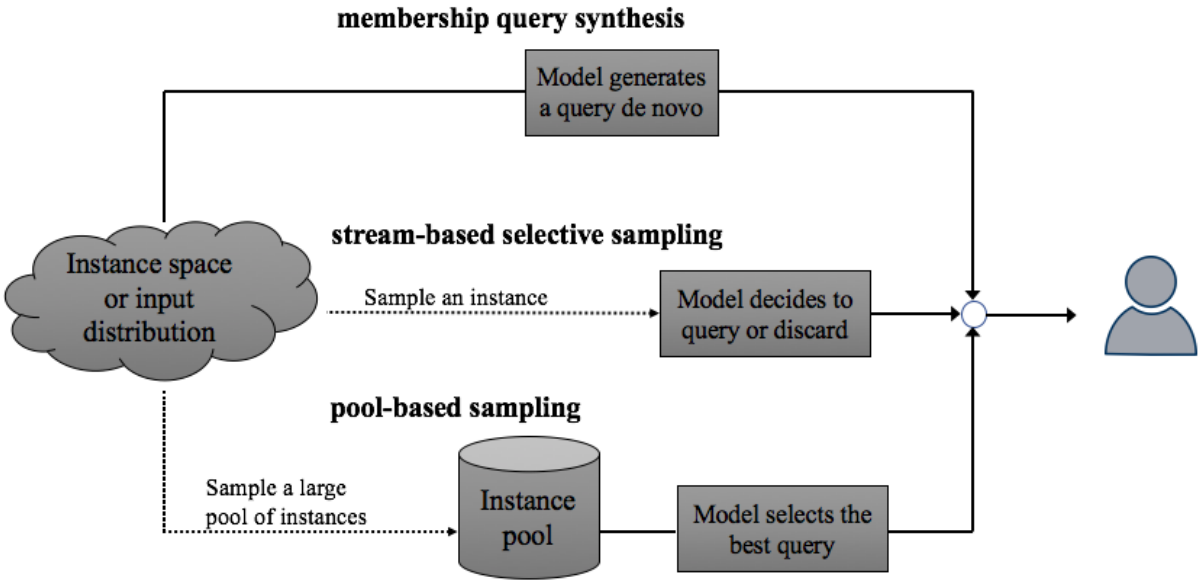


Figure 10: Main Active Learning Scenarios
Source: Own illustration based on [88]

(1) (Membership) Query Synthesis

The first approach investigated is active learning with *membership queries* [91]. Here, the learner is allowed to request labels for any unlabeled instance in the input space. In this case, queries are not sampled from some natural underlying distribution, but generated *de novo*. This approach is not particularly suitable for TC as the text produced by the algorithm may not be readable for human annotators as the learner may select arbitrary values [88, 89].

(2) Stream-Based Selective Sampling

In *stream-based* (or sequential) active learning, the learner is given access to an input stream underlying a natural distribution. The learner can then decide whether to request the instance to label it or discard it depending on the strategy used (see Chapter 2.2.5)[92]. The disadvantage here is that the learner does not have the option of selecting the most informative instances from the set of examples as a whole. On the other hand, this scenario may be suitable for settings where storage or processing power is limited (e.g. a smartphone) as there are no comprehensive calculation processes required [89]. Usually, however, a large amount of unlabeled data is available at once, and instances vary in value (information quality). It is for this that the *pool-*

based approach is more commonly used in applied AL research, especially for TC (e.g. [7, 93, 94]).

(3) Pool-Based Sampling

In *pool-based* active learning, used for the first time in the study of [95], the learner has access to the whole pool $\mathcal{U}$ of unlabeled instances from the beginning. The learner can analyze those and evaluate all instances in the pool according to some defined *informativeness measure* (see Chapter 2.2.5). The most "informative" instance(s) (depending on the *Batch Size*) are then queried by the oracle so that they may be labeled. The labeled examples are removed from the pool and added to the labeled training set $\mathcal{L}$. This labeled training set is then used to train the learner and the process begins anew (see Figure 11).

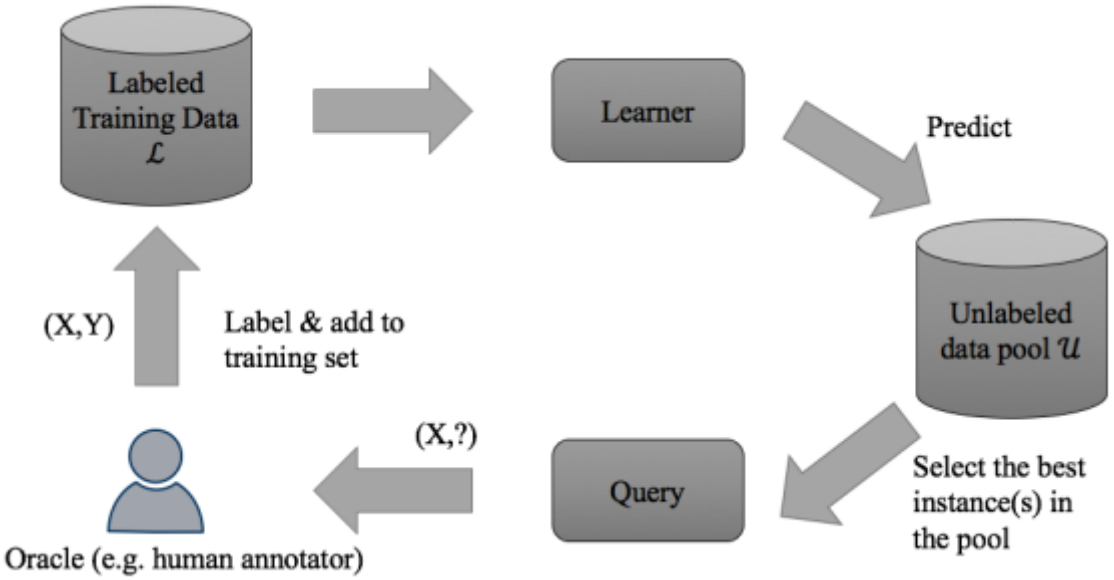


Figure 11: Pool-based Sampling
Source: Own illustration based on [89]

2.2.3 Batch Size

The *batch size*, the number of instances queried for subsequent classification per round, would in an ideal situation be one. This would allow future queries to be based on decisions from an optimally trained model. In order to increase efficiency, it is usually set higher than one, also depending on the data size [96]. Using a larger batch and querying several instances at once is known as *batch-mode* AL. The challenge here is the manner in which to arrange the query set in order to avoid generating an overlap in information [88]. In their study using "greedy" heuristics to ensure that instances in batch are both diverse and informative, Hoi, Jin and Lyu [28] demonstrate that large batch sizes can also lead to a good performance.

2.2.4 Seed Set

Another issue in AL is how to pick the first instances, also called *seed set*, for the initial training data. In many studies, the seed set is generated by random sampling from the whole unlabeled pool of data [11, 97, 98]. The problem, however, is that when a seed set does not contain a certain class, the learner may grow overconfident about class membership. This can be especially problematic if the class distribution in data is skewed or imbalanced. In such a scenario, a typical random example will possibly not contain rare classes, which can result in the *missed class effect* [99]. As a solution, Nguyen and Smeulders [100] suggest performing pre-clustering aimed at selecting informative examples from each class (cluster). Dligach and Palmer [98] recommend enriching the data with rare class examples. In their study, they achieve better *accuracy* at the beginning of the AL process; however, as the number of examples from the predominant class increases, the *accuracy* worsens. Nevertheless, the overall classification of *accuracy* was still higher than for random sampling.

Once an initial training set has been established, the subsequent question is which instances should be queried next to improve the model. Some of those possible query strategies are introduced in the following section.

2.2.5 Query Strategy Frameworks

Generally, there are two main learning strategies and approaches to combining them. The *exploitation*-based strategy reaches a decision predicated on the output generated by the classifier, focusing on examples near the current decision boundary in order to refine the decision boundary. On the other hand, *exploration*-based strategies search for regions in the version space that the algorithm might classify incorrectly [101]. This thesis focuses on exploitation-based learning strategies, as described below.

(1) Uncertainty Sampling

Due to the relative ease of implementation and intuitive appeal, *uncertainty sampling* (US) is probably the most widely used AL strategy in practice [8, 89, 95]. The core idea is that the instances about which the learner is most uncertain, or least confident, are queried. US has been realized with both probabilistic-base learners such as Naïve Bayes and non-probabilistic-base learners such as SVMs. These classifiers produce usually output scores or probabilities, indicating class membership (an *informativeness measure*). In case of output scores, studies have demonstrated that these scores can successfully be converted to probabilities [102, 103, 104]. These probabilities can then be utilized as a measure of uncertainty to find examples for the classifications that are the least certain [88]. For example, in a binary probabilistic classification, the algorithm queries the example which has a *posterior probability* nearest 0.5 [95]. Generally, the following three main approaches have been developed for measuring the uncertainty (see [89]):

Least Confident: The instances for the predicted output of which there is least confidence are queried:

$$x_{LC}^{*} = \underset{x}{argmax} 1 - P\emptyset\,(\hat{y}|x),$$

is the prediction with the highest *posterior probability* under the model ø. Therefore, it selects instances of the label about which it is most unsure. One disadvantage of this approach is that it considers only the best prediction and discards the rest of the information. For this reason, this approach is only suitable for binary classification problems.

Margin: Another AL strategy based on the output margin:

$$x_M^* = \underset{x}{argmax}[\, P\text{ø}\,(\hat{y}_2|x) - P\text{ø}\,(\hat{y}_1|x)],$$

as $\hat{y}_1$ and $\hat{y}_2$ are the two most likely predictions in terms of the model, it addresses the drawback of the LC method by also including the second-best labeling. Ambiguous instances with small margins should help the model to discriminate between them. However, for many classes this approach still does not include much of the output distribution.

Entropy: A more general approach based on Shannon entropy, measuring the variable's average information content:

$$x_H^* = \underset{x}{argmax} - \sum_y P\text{ø}\,(y|x) \log P\text{ø}\,(y|x),$$

where y incorporates all possible labels of x, and ($P\text{ø}\,(y|x)$) is the *posterior probability*. Based on this classifier distribution, $x_H^*$is the uncertainty measurement function.

While all three strategies constantly outperform passive classifiers, there is no generally superior strategy and performance quality is application-dependent. Nevertheless, in a study of [105], the margin-based approach was most promising.

Another method is not to use the direct output of the classifiers, but to convert the output into a confidence value [102]. This can be challenging in case of non-probabilistic classifiers.

An interesting approach was conducted in the study by [11], who executed exactly the opposite of the common US process. They selected instances having the highest certainty (i.e. the lowest entropy) to avoid incorrect assumptions made by the model having a small dataset.

Although US provides simplicity and speed, the fact that probabilities are based on the output of a single classifier is a disadvantage, especially if the classifier has only been trained with minimal data [89].

(2) Query by Committee

Query by committee (QBC) is a second widely used selection strategy proposed by [106], grounded in a committee of several classifiers c. In this original form they used a selective sampling algorithm with a committee of |C| = 2 that sequentially gets an instance x at each iteration. Then, two random hypotheses are created in the version space containing all unlabeled data and the output, and the respective labels of x are compared. If there is a disagreement between the two hypotheses, the oracle must label the input x and it is added to the training data. This method is inefficient for real-world applications with an outsized version space and a high-dimensional feature space.

To overcome this problem, different modifications of this QBC have been developed, which relate especially to efficient means of creating committees, and how disagreement of these committees may be measured accurately.

Committee Creation

Freund and Schapire [107], [108] develop a strategy called *AdaBoost* that uses an additional "WeakLearner" classifier that aims to find examples that are hard to classify and assign higher weight to them.

Mamitsuka [109] proposes a new query learning approach called *Query-by-Boosting* and *Query-by-bagging*, combining the idea of QBC and "boosting" [110] or "bagging" [111]. "Boosting" and "bagging" are both techniques to improve classification by running the learning algorithm repeatedly on a set of re-sampled data. The resulting output hypotheses are combined to create predictions of greater accuracy. *Query-by-bagging* is equivalent to the original QBC, except that it utilizes "bagging" on the input sample (uniform distribution) and selects subsamples at which the hypotheses of the predictions are most evenly split. *Query-by-Boosting* is founded on "boosting", and *AdaBoost* focuses on misclassified instances. In this manner, hypotheses are created iteratively whereby for each subsample the distribution is adapted according to a misclassification rate. These subsamples are then drawn based on the re-weighted distribution [109].

Further ensemble-based methods have been proposed, for example, the *Decorate* and *Active-Decorate* strategies [112, 113]. The *Decorate* algorithm focuses explicitly on creating diverse ensembles by adding artificially generated examples to the training set. The *Active Decorate* is a variant of the original QBC using the *Decorate* algorithm to create the committees resulting in more effective sample selection than *Query-by-Boosting* or *Query-by-bagging*.

Disagreement Measures

Once a committee of classifiers has been created, the next question is the manner in which the disagreement of the committee's member can be measured and quantified. As with committee creation, many heuristics have been introduced. Some of the widely used measures are described below:

The *vote entropy* strategy proposed by [114] is similar to the one for US applied to QBC. It can be defined as:

$$x_{VE}^{*} = \underset{x}{argmax} - \sum_{y} \frac{V(y,x)}{|C|} \; log \frac{V(y,x)}{|C|},$$

where y involves all possible labeling, V(y,x) is the number of committees that "voted" for label y for instance x, and |C| is the committee size. This formulation can also be rewritten in a "soft form" taking the confidence of the decision into account:

$$x_{SVE}^{*} = \underset{x}{argmax} - P_C(y|x) \; log P_C(y|x),$$

where $P_C(y|x) = \frac{1}{|C|}\sum_{c \in C} P_c\,(y|x)$ is the average ("consensus") probability that y is correctly labeled according to the committee. Further adaptions of these uncertainty measures for QBC on a probabilistic basis have been proposed by [115] who combines margin-based disagreement with the maximal class probability normalized by the number of class values.

Another approach called *f-complement* is used by [116], who apply the F-score to measure the disagreement between the committees. It can be calculated as:

$$x_F^* = \frac{1}{2} \sum_{M_i M_j \in C} (1 - F_1\left(M_i(x), M_j(x)\right)),$$

where $F_1\left(M_i(x), M_j(x)\right)$ is the F-score of the predictions of instance x, by a committee i relative to the prediction of a committee j.

Kullback-Leibler (KL) divergence D [117] is a third variant to measure the disagreement by quantifying the difference of two probability distributions P and Q:

$$D(P \parallel Q) = KL\,(P, Q) = \sum_{y \in Y} P(y) \log \frac{P(y)}{Q(y)},$$

This definition is used by [93] in a modified version called *KL divergence to the mean* (KLM) which is the KL divergence between each distribution $P_{c \in C}(y|x)$ and the average of all distributions $P_{avg}(y|x)$ defined as

$$x_{KLM}^* = \underset{x}{argmax} \frac{1}{|C|} \sum_{c \in C} D\,(P_{c \in C}(y|x) \parallel P_{avg}(y|x)).$$

A high KLM score indicates diverging distributions.

Melville and Mooney [113] successfully use a similar score to KLM called Jensen-Shannon divergence in their *Active-Decorate* algorithm.

(3) Alternative Learning Strategies

While US and QBC are the learning strategies that are used mostly in practice, other approaches like *Expected Model Change*, *Expected Error Reduction* or *Variance Reduction* have been proposed as well. As these methods require a relatively high computational effort or bring restrictions on the learning algorithms that can be applied, they are only summarized briefly.

Expected Model Change

The *expected model-change* framework aims to select the instances that would cause the greatest model change if we knew its label [88]. Settles, Craven and Ray [118] propose a way to implement this by defining the *expected gradient length* (EGL) using gradient-based optimization. In so doing, those instances are selected where the model change measured by the length of the training gradient is the greatest. As the algorithm does not know the correct label in advance, the length is calculated as an expectation over all possible labeling. Although

the results in this work are promising, it is computationally expensive in case of large datasets. Another precondition is that the features are properly scaled [88].

Expected Error Reduction

AL with *expected error reduction* [119] considers the unlabeled data pool in its entirety to estimate the expected error of the current learning algorithm. Furthermore, the effects on the expected error of all potential labeling requests are determined. This is done by considering, for each unlabeled instance x, all possible labels y and adding these pairs (x, y) to the existing training set. Following this, the classifier is retrained on the remaining unlabeled data set, the expected loss is estimated (e.g. by log loss or 0/1 loss), and the average expected losses for each possible labeling y are assigned to x. The instance x with the lowest expected error is selected to be labeled by the oracle.

According to [119], this approach performs better than US or QBC. Nevertheless, they also state that it is computationally more complex compared to other QBC algorithms and totally inefficient when implemented naively. They propose some solution strategies, for instance, using heuristics such as Monte Carlo sampling for error estimation and the choice of query, or using only a subsample of the pool.

Variance Reduction

Variance Reduction tries to improve the previous method by reducing the generalization error indirectly by minimizing the output variance. It assumes that learner's expected error can be decomposed into the three factors: *noise*, *bias* and *output variance*. As the first two factors are invariant given a fixed model-class, the objective is to reduce the model's output variance for the unlabeled instances [89]. Although it is more efficient than expected error reduction and several heuristics have been proposed [28, 105, 120], there are still practical drawbacks in terms of computational complexity. Additionally, it can't be used with classifiers like k-nearest-neighbor or decision trees [89].

2.2.6 Classifiers in the Context of Active Learning

In Chapter 2.1.2.6, possible classifiers for TC were introduced. The basic principle is that a set of labeled training examples is used by the classifier to predict the class of unlabeled examples. In this chapter, those classifiers are further investigated in the context of AL and some related studies are introduced.

Additionally, in contrast to traditional supervised machine-learning, in AL people are involved in the classification loop. This is the reason for both the pure accuracy being decisive, and for the efficiency of the classifiers having an even more important role. As it is often necessary that people conducting such research have a domain knowledge, it may become very expensive should there be long waiting times in front of the computer waiting for the classifier to finish. That is the reason that the choice of a specific classifier must be carefully balanced: to ensure having both an accurate and a user-friendly application.

In this section, the three classifiers introduced are examined in the context of AL.

(1) Naïve Bayes in the Context of Active Learning

Naïve Bayes is a probabilistic classifier that assumes independence between the instances of the training data. It is often used in AL and TC environments due to its simplicity and speed, in addition to its frequently good *accuracy*.

Generally, NB can be used with several query selection strategies like US [9, 11, 121] or *QBC* [93]. The uncertainty of an example in a binary classification problem could be determined as follows:

$$uncertainty(c) = 1 - |p(x \in class1) - p(x \in class2)|$$

where $p(x \in class1)$ and $p(x \in class2)$ are the outputs of the NB classifier [122]. When applying QBC one can use the classification scores of NB to calculate the differences between the committees (e.g. Kullback-Leibler divergence) [93].

Segal, Markowitz and Arnold [9] use US with NB as a classifier in their work on the grounds that it is efficient on larger datasets and still yields good results. They apply a modified variant utilizing the geometric mean instead of the arithmetic mean to join the conditional probabilities of the words for labeling the content of a large email corpus as spam or not spam utilizing AL.

Cardellino, Villata, Alemany [11] use *reversed* US together with the NB when applying AL to classify legal licenses, opposed to their regular ML setting where they use the SVM classifier. Applying feature selection and using the one vs. all strategy, they achieved better results using NB.

Settles [121] created the open-source AL application, "Dualist", using entropy-based US and NB that allows the classification of textual data into categories (e.g. newsfeed into the categories hockey and baseball). He also relies on NB due to its abovementioned advantages.

Often, NB is combined with Expectation-Maximization (EM) algorithms to improve the estimates of the classifier [93, 121, 122, 123].

(2) SVM in the Context of Active Learning

The maximum margin-based classifier SVM is, according to the web survey of [124], the most widely used classifier in AL. That is probably due to the fact that SVMs generally achieve the highest accuracy.

A widely used approach for classification using the version space was proposed by [125]. The fundamental idea is that uncertainty can be defined as the distance from the separating hyperplane, which means that the examples with the larger distance to the hyperplane belong to the class with greater confidence:

$$uncertainty(c) = d(x) = < w, x > + b$$

Alternatively, the instances closest to the hyperplane are the most informative examples as they halve the version space. [7], Tong [125] [7], proposed three approaches for achieving this.

1) *Simple Margin:* select the unlabeled instance in the pool closest to the current decision hyperplane (smallest margin).

2) *MaxMin Margin:* for all unlabeled instances, compute the margins m^- and m^+ of the SVMs obtained when x is labeled -1 and +1, respectively. Then, the unlabeled instance for which the quantity $min(m^-, m^+)$ is greatest should be chosen.

3) *Ratio Margin:* in a manner similar to the MaxMin margin, compute the two margins m^- and m^+, but rather than considering the relative sizes, use the instance for which the proportion $min\left(\frac{m^-}{m^+}, \frac{m^+}{m^-}\right)$ is largest.

The *Simple Margin* strategy for AL is essentially the same as the US method that has already been introduced as the instance about which the SVM is most uncertain is chosen [7].

The *Simple Margin* has efficiency advantages as it needs only one SVM compared to *MaxMin-* and *Ratio Margin*. On the other hand, as it is only an approximation, it has the lowest accuracy [7, 125].

Variants of the *Simple Margin* have been proposed in the context of AL, for example, by [51, 126, 127]. Bordes, Ertekin, Weston [126] introduce *LASVM*, an online algorithm that randomly selects a small subset of examples from the whole pool. Within this subset, the instance closest to the hyperplane is used to train the classifier. The results are comparable to those for which the whole dataset was queried to find the closest instance. In the experiment of [127], a setup was created in which the oracle had to label both instances (e.g. documents) and features (e.g. words).

Nevertheless, Schohn and Cohn [128] use the *Simple Margin* strategy in their experiment utilizing AL having a small and actively selected set of training instances, and accomplished a better result than those in which the whole dataset was labeled. Although they remark on the time problem with SVMs, the running times of the AL experiment on large datasets were comparable or even faster than training all available data.

Guo and Wang [129] presents a *multiclass-classification approach based on SVM active learning* (MC_SVMA) which first computes possible pattern classes using the discrepancy between unlabeled and labeled instances. Thenceforth, their algorithm does not only use uncertainty as informativeness measure, but also includes rejection and compatibility to search for the most valuable instances.

Yang, Sun, Wang [130] introduce another selection strategy called *Maximum loss reduction with Maximal Confidence* (MMC). To solve a multiclass problem, they use binary SVMs classifiers (a one vs. all strategy) and the sigmoid function to convert the SVM output into probabilities in order to assign classification probabilities to all training instances. Thereafter, they use LR in addition to this to train the predictive model. Using this method, they achieved good results; however, they state that their approach would be expensive for large datasets due to the many iterations in AL.

(3) Artificial Neural Networks in the Context of Active Learning

An ANN is a classifier related to SVM, and is based on a net of connected and weighted neurons.

For multilayered NN, the classification of some regions of the input space may sometimes be determined implicitly by several ANNs. Nevertheless, there may be other regions where the ANNs disagree – the so-called region of uncertainty [131, 132]:

$$R(S^m) = \{x : \exists c_1, c_2 \in C \;\; c_1, c_2 \; are \; consistent \; with \; all \; s \in S^m, and \; c_1(x) \neq c_2(x)\}$$

whereby C is a class and S^m a set of examples.

Freund, Seung, Shamir [133] proves in his work that the QBC strategy is an efficient algorithm for the perceptron concept in cases where the distributions of prior examples are close to uniform.

Cohn, Atlas and Ladner [131] introduce a selective sampling query strategy similar to the QBC strategy. They used a strategy called SG-net, a type of version-space search algorithm. They define the two concepts "most specific" network and "most general" network as a type of committee in their AL implementation. Those examples where the two networks disagree the most are then selected [89].

Lu, Rughani, Tranmer [134] use an informative-sampling algorithm (an evolutionary algorithm), a combination of ANN and EM, in order to locate the most informative data points in an explorative way in the context of large unbalanced datasets. The algorithm also incorporates the data points that result in the largest disagreement among the models into the training set.

Wang, Kwong, Jiang [135] describe an uncertainty-based AL method for single hidden-layer networks using an extreme learning machine algorithm (the weights connecting input and the hidden layer are never updated). To calculate the uncertainty, the outputs of the ANN are transformed into a probabilistic form using the entropy strategy.

Another AL algorithm in the context of ANN is introduced by [132]. They create the algorithms called "active learning based on existing examples" (ALBETE) and ALBETE/NI (where NI is "network inversion"), that aim to refine the classification boundary by taking reliability into account. They want to focus the training data on the more reliable classification boundary (e.g. near the decision boundary) in order to avoid misclassification.

2.2.7 Stopping criterion

As the goal of AL is to minimize labeling costs, an important part of an AL application is the consideration regarding when to discontinue the labeling process. A stopping criterion is used to end the learning process automatically [90]. Most of the approaches proposed are based on the principle of stopping once the *accuracy* of the learner will only increase with excessively high costs or not at all go beyond a certain level (target *accuracy*) [89]. A very simple approach is to stop when no more uncertain instances are inside the pool. As the amount of textual data is generally extremely large, this is rarely achievable. Another option is to use cross-validation or a hold-out set, and stop when the classifier performance reaches some target threshold in a

randomly selected validation set, separated from the training set [102]. Schohn and Cohn [128] developed a margin-exhaustion stopping criterion (only applicable for margin based classifiers like SVM) that stops when all remaining unlabeled instances are outside the model's margin (soft-margin SVMs [66]). Vlachos [136] employs a confidence-based stopping criterion that uses the average uncertainty of the unlabeled set and stops when the model's confidence starts to decrease. Other confidence-based ideas are the minimum expected error strategy [137], overall-uncertainty [138], and performance convergence/uncertainty convergence using gradients [8]. An additional concept is *stabilizing predictions* introduced by [139], which tests whether the predictions of a so-called stop set have stabilized.

In general, there is no single best stopping criterion. The question of when to discontinue the labeling process is difficult to answer and depends on a number of factors, including the kind of classifier used, and the natural distribution of the data, among others.

2.2.8 Evaluation

In addition to classical TC performance measures such as *precision* and *recall*, or the ROC curve evaluating the fully labeled data set, an AL system requires a more detailed analysis of the labeling process. For this reason, *learning curves* are typically used to monitor and visualize the progress of the AL system according to some classifier performance [140]. The learning curve is usually plotted with the number or percentage of instances labeled by the annotator on the x-axis, and a performance measure like *accuracy*, *precision* or F_1 on the y-axis (see Figure 12) [125, 128, 140, 141]. This curve can then be utilized to calculate a global assessment score like the *Area under the Learning Curve* [140, 142, 143].

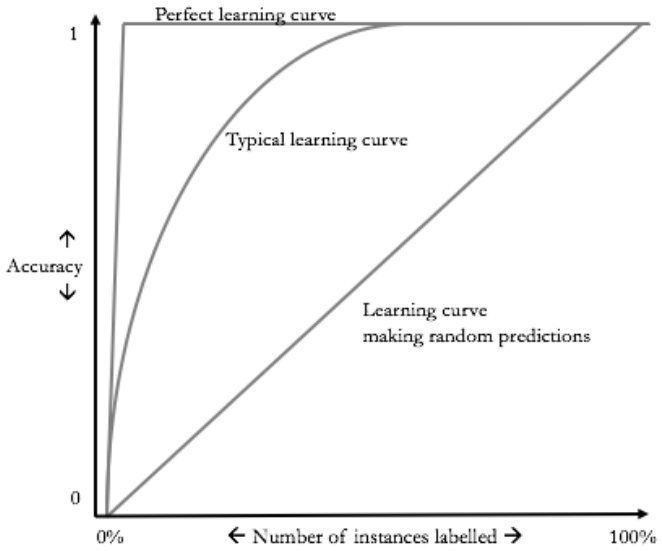


Figure 12: Learning Curves
Source: Own illustration

Another approach is called *deficiency*, a measure using *accuracy* to evaluate the learning speed of an AL algorithm [144]. Raghavan, Madani and Jones [127] modified this definition and use the F_1 score to define *efficiency* as "1- *deficiency*" to indicate the learning rate as a value ranging between 0 and 1.

A further issue in the context of AL is to keep the simplicity and duration of the annotation process in mind, as the costs of labeling are often quite high. For example, [145] examined the impact of user interfaces in AL (e.g. showing features or instances), while [123] analyzed the waiting time of the user during processing.

2.2.9 Summary

In this chapter, research question two – "How can (active) machine learning support the classification of legal documents and their content (norms)?" – has been partially answered. To that end, the basic concept, classifiers, query strategies and other terms in the context of AL and TC have been discussed, but the focus has not been on the legal domain.

AL is a subfield of machine learning that involves people in the learning process to reduce the need for having a large set of labeled data to create an accurate classifier. The objective is to find and label those instances that best assist the learning algorithm to differentiate between the labels. Therefore, only a small amount of labeled data is needed. In this learning process, many different influencing factors must be considered. The most important factors are summarized in Table 7.

Keyword	Summary
AL Process	Depending on various factors, such as the scenario used and query framework, unlabeled instances are queried and labeled with the support of an oracle (e.g. human expert) in an interactive loop.
Scenarios	Instances are either queried one-by-one in a sequential manner (stream-based), or batch-wise (pool-based).
Batch Size	The number of instances that are queried in the first learning round. Usually, these instances are taken by a random sampling of the whole pool of unlabeled data.
Seed Set	The number of instances that are queried simultaneously is, in an ideal situation, one. Due to performance considerations, usually two or more instances are queried at once.
Query Framework	To select those instances that should be queried, several strategies have been proposed depending on different factors like the choice of classifier. The most commons ones used in practice are uncertainty sampling (US) and Query by Committee (QBC).
Classifier	Many common TC classifiers have also been part of AL research. As people are in the loop in AL, efficiency becomes more important. For each classifier, different ways exist to find those instances about which the classifier is unsure.
Stopping Criterion	The goal of a stopping criterion is to stop the learning process to avoid waste of effort. Several approaches have been developed to find the right point to stop learning.
Performance Measure	In addition to the original TC measures, additional measures such as learning curves or deficiency are used to visualize the learning performance of the system.

Table 7: Active Learning Concepts
Source: Own illustration

AL has already been successfully applied to various topics in TC. In the chapter that follows, TC in the legal domain generally, and in the context of AL, is studied more closely.

2.3 Text Classification in the Legal Domain

While the last chapter gave a general overview how AL can support the classification of textual data, in this chapter the focus lies on *legal* textual data. First, the motivation is briefly recapitulated. Then, the two use cases, *document classification* and *norm classification*, are investigated.

2.3.1 Rationale for Classification

As indicated in Chapter 1.1, the legal landscape is likely to undergo major changes in the near future. For instance, current research deals with, *inter alia*, predicting the legal outcomes of cases at court, and automated legal document classification [146].

The latter is an important task with regard to digitization as a vast number of legal documents, such as judgments, contracts, patents and other legal texts, in a relatively unstructured format (e.g. emails) are produced daily by different types of firms and organizations. For a variety of reasons (e.g. legal reasons), these legal documents must be stored internally or made publicly available via online portals like "juris". Hence, to retain the document overview and provide fast access in order to enable working with them (see norm classification), it is necessary to organize in a proper manner all documents produced. This organization process, referred to as *document classification (DC)*, is a crucial daily task. For instance, Gruner [147] analyzes a lawsuit and states that much time and cost are involved in document discovery and research tasks. Appropriate DC could reduce the time and cost required to complete these tasks. In the classification process, the most important problems are, on the one hand, the sheer number of legal documents that require labeling. On the other hand, correct classification requires knowledge and understanding of the content of the document and of the taxonomy of legal texts. Additionally, as the person who undertakes the classifying might not be the producer of the document (e.g. receiving an email with a legal document), the document must be read first in order to be classified correctly. This is in itself considerably time-consuming and requires legal-domain knowledge. Hence, the (partially) automatic classification of legal documents would generate big relief in many use cases. Fortunately, legal texts have some common characteristics that can facilitate such a classification (see Chapter 2.3.2).

Most legal documents must not only be classified into certain categories as a matter of form, but are also required for further work. Merely reading the many types of legal documents already mentioned to find relevant norms and provisions would require a substantial amount of time. However, it is usually not sufficient to skim a text; rather, it must be read carefully and the details understood. Legal documents are often very comprehensive and contain complex expressions that had become more complex over time [4]. Such complexity manifests in long sentences with complex clauses, cross referencing and amendments, such as changes in definitions over time [5]. The result is that people working with legal content require much time to find and understand core statutes and provisions. This problem should be addressed by *norm classification* (NC). The objective of NC is to organize the individual sentences into classes so that important sections of the text are already annotated for the user, and fast access to the information sought can be provided. As this classification is even more complex, time-consuming, and requires strong legal-domain knowledge, in practice it is not typically undertaken. Hence, a (partially) automatic classification of the sentences in a legal document

would be an appropriate solution that could enhance work with legal content in terms of time and work quality. A more detailed introduction is given in Chapter 2.3.3.

Although legal texts may appear complex to the ordinary human eye, compared to other texts like newsfeeds or social media posts, they have the advantage that the language is very formal so that the variations normally observed in written texts are relatively limited. In addition, legal texts such as laws are unlikely to contain grammatical or spelling errors. Legal texts therefore appear to be a very promising data source for applying ML algorithms. The theoretical and practical study of the classification algorithms used in ML, as discussed in the previous chapters, provide a convenient starting point for such an endeavor.

2.3.2 Classification of Legal Documents

In the previous section, the rationale for DC in the legal domain was presented. In this section, an overview of the structure and various kinds of legal texts is provided and current research projects in the context of legal DC are discussed.

At the macro-level, legal documents are usually highly structured documents. While there is a multitude of different legal texts, such as a variety of laws, different kinds of contracts, patents, and the like, they are all structured in a similar manner: they are organized into chapters and sections, often having a meaningful title that reveals much information about the content of the document [148].

The classification of legal documents into categories has not been investigated intensively. For an overview of the most important legal text types (document classes), a heuristic classification is presented in Figure 13 (in the figure, the text-type names are kept in the original language, German, in order to avoid translation errors). Busse [149] distinguishes the following nine super classes of legal-text types, emphasizing that this taxonomy is not final, and that several of these classes may overlap at some point:

1. *Textsorten mit Normativer Kraft (förmlich verabschiedete Normtexte)* (“text types having a normative power”): legal texts in this super class are text types possessing the force of law, such as like laws or international contracts.
2. *Textsorten der Normtext-Auslegung* (“texts having a standard text interpretation form”): texts in this super class have usually many quotations or references to other texts, for example, legal commentaries or judgmental commentaries.
3. *Textsorten der Rechtsprechung* (“jurisdiction”): legal texts aiming to craft legislation such as court decisions or orders.
4. *Textsorten des Rechtsfindungsverfahrens* (“texts arising in the findings of justice process”): all texts and documents that are produced during a trial, for instance, legal opinions and judicial transcripts.
5. *Textsorten der Rechtsbeanspruchung und Rechtsbehauptung* (“texts in the context of legal claims”): legal texts in which the producer of the text is not an agent of a legal institution (e.g. petition).
6. *Textsorten des Rechtsvollzugs und der* Rechtsdurchsetzung (“texts in the context of legal enforcement”): texts of this type are typically produced by an institution and directed at non-institutions (e.g. warrants).

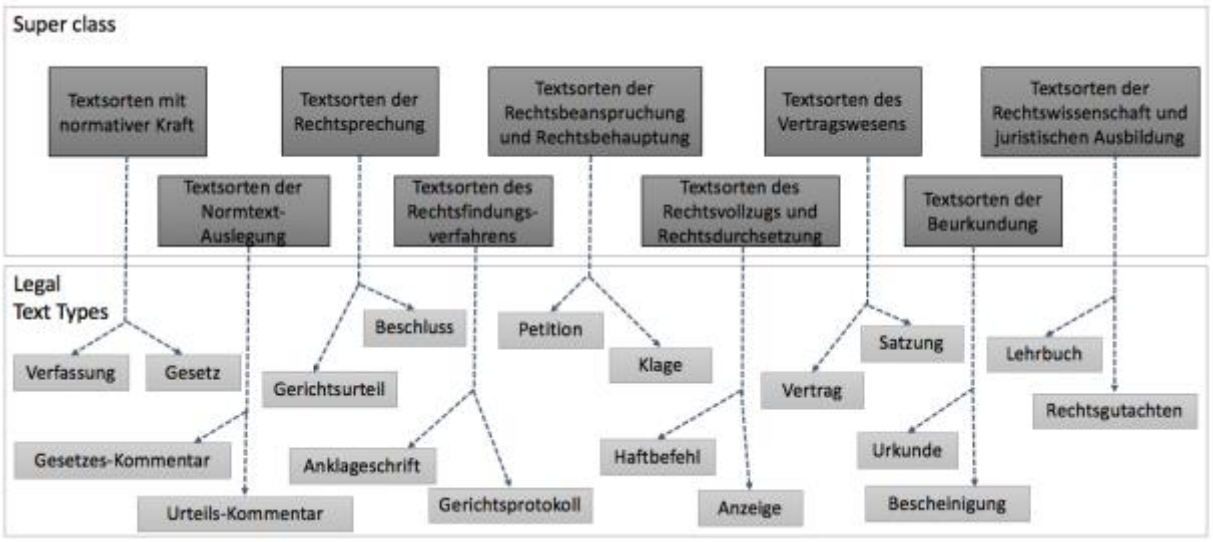


Figure 13: Classification of Legal Texts
Source: Own illustration based on [149]

7. *Textsorten des Vertragswesens* ("contracts"): this super class compromises all types of contracts, such as those from civil law, or notarial contracts.
8. *Textsorten der Beurkundung* (notarielle und amtliche Textsorten) ("certificates"): texts having a notarial and official certifying character issued in an independent institutional context (e.g. certificate, testament, land register entry)
9. *Textsorten der Rechtswissenschaft und juristischen Ausbildung* ("texts in the context of legal sciences and legal education"): legal texts used in university science, such as like educational books, case collections, and judgment commentaries.

This classification of legal text types into super classes provides valuable insight in the variety of legal texts. In practice and in this work, the classification within the text-type level, for example, the classification of a document as a certain judgment or a certain law, is of greater significance than the simple allocation of a text type to a super class.

In the following, some attempts at legal DC in research projects are described:

In their study, Gonçalves and Quaresma [150] used a training set of 8,151 legal documents representing the court decisions of the Portuguese Attorney General's Office. These documents were categorized into one or multiple classes (multilabeling), such as "military" or "army injured". For the classification, they applied several preprocessing steps, including feature reduction and stemming, and use a linear SVM as a classifier, executing it in the ML tool Weka. They performed two experiments: one using all words, and one using only specific word types or combinations (e.g. only nouns and adjectives). They observed better results with the latter than with the initial base setup which included all words.

Ratner [151] compared several classifiers (i.a. LR, SVM and MNB) in the classification of 10,000 legal contract documents having in total 77 categories, such as "Arbitration Agreements" or "Manufacturing Contract". He also applied preprocessing to the training data and used the TF-IDF text representation. He achieved a test *accuracy* of 82% with the LR and the linear SVM, while the SVM with the radial basis function kernel accomplished only 25%.

Although the training *accuracy* of MNB is comparable to the one of Logistic Regression and SVM, it reaches a test *accuracy* of "only" 70%.

Roitblat, Kershaw and Oot [3] compared the classification *accuracy* of computers relative to traditional human manual review. All documents that are potentially relevant for the litigation were labeled responsive, and the irrelevant documents, non-responsive (i.e. a kind of binary classification problem). For this study, they used a random sample of 5,000 documents from the original review for a second review. This was accomplished by two human teams and two commercial electronic discovery systems. The result was that the two computer systems used were in every measure at least as accurate (measured against the original review) as the human re-review.

2.3.3 Classification of Legal Norms

We have observed that legal documents may be comprehensive, and statutes and their provisions may often be difficult to find and to understand. Hence, it would be beneficial to classify the content of legal texts, the sentences, into predefined categories in order to enhance organization within a certain law text and to find similar components among laws.

De Maat and Winkels [152] provides a taxonomy of possible sentence types based on the analysis of Dutch laws. This taxonomy, illustrated in Table 8, may also be used as a common reference; it was used in a similar manner by, for instance, [153] or [154].

Sentence Type	Description	Typical Pattern
Definitions and Type Extensions	Definitions are used to describe terms that are used in the legal source. Type extensions are additional definitions that expand or limit earlier ones.	"By x is understood y"
Deeming Provisions	Sentences that declare one situation equal to another situation within a certain context. Often like definitions, but usually involve some kind of legal fiction.	"Is deemed to"
Norms	Norms form the actual content of legal texts. Norms can by grouped into rights, permissions, obligations and duties. Rights and permissions are difficult to distinguish, as are duties and obligations.	"May" (rights) "must" (obligation) Duties infrequently follow a pattern
Delegation	Delegations confer the power to create additional rules to some legal entity.	"May create rules," "may adopt provisions"
Application Provisions	Application provisions specify situations in which other legislation (e.g. an article) does or does not apply.	"does (not) apply"
Penalization	In case of a violation of norms, the law specifies the penalties.	"will be punished with"
Value Assignments and Changes	A value assignment is a sentence that provides an initial value for a concept. Changes are modifications of these values in a later step.	Contain mathematical operations (e.g. reduce, increase) and comparisons (at most)
Rule Management	Sentences that deal with the maintenance of a legal text.	?

Table 8: Sentence Types in Legal Texts
Source: [152]

De Maat and Winkels [152] used these developed patterns associated with a certain sentence type to classify the content of 18 different Dutch regulations. They achieved an *accuracy* of 91% using this pattern-based classifier. One year later, de Maat, Krabben and Winkels [155] performed a similar study using ML techniques to classify Dutch sentences. Sentences with mixed types were discarded so that they used 584 sentences for the experiment in total applying multiclass classification. They executed the experiment in various settings, using different combinations of preprocessing steps and text representations applying a linear SVM as classifier within the Weka framework. With the optimal setup, they achieved enhanced *accuracy* of almost 95%. In a second experiment, they analyzed the generalization capabilities of a certain classifier across law texts. In other words, they examined whether a classifier trained with certain law(s) could perform as well on different legal texts. However, they found that, in this case, the ML approach performed worse than in the original experiment in which the same data had been used for training and testing.

A similar study was conducted by Šavelka, Trivedi and Ashley [10] who examined the relevance and applicability of the individual statutory provisions. Their objective was to identify relevant provisions within a specific medical context (a binary classification problem). They conducted two experiments using an interactive ML framework similar to AL. In total, they utilized 403 statutory documents with 4,022 provisions for the "Kansas" experiment, and 135 documents with 1,564 provisions for the "Alaska" experiment. As a classifier, they chose a linear SVM. In the "Kansas" experiment, they created a new predictive model having a final classifier *accuracy* of 82%. In the "Alaska" experiment, in which they re-used this model, the "cold start problem" could be skipped (discovery of version space). They achieved from the beginning on improved outcomes, resulting in a final *accuracy* of 83%. They also state that AL techniques are very helpful and outperform the traditional manual assessment.

Francesconi and Passerini [153] used ML techniques to detect different provision types (e.g. Repeal, Delegation, Prohibition) in 582 Italian legislative texts (provisions). They applied preprocessing and attempted several methods of document representation (e.g. TF-IDF) using NB and SVM as classification algorithms. With the best combination of document representation and feature selection strategies, they obtained an *accuracy* of 88% with NB and 92% with the SVM. Di Silvestro, Spampinato and Torrisi [154] conducted a similar study with what was probably the same data set. They used the NB as classifier and obtained an *accuracy* of 82%.

Cardellino, Villata, Alemany [11] created a multiclass AL test setting to classify 433 licenses. While they achieved only an average *accuracy* of 76% using an SVM in their default supervised ML setting, with AL they attained an improved *accuracy* of 83% using NB as the classifier, and reverse US to query the next instance. For the interactive classification process, they built a hybrid application with Perl, Scala and Java, using libraries within Weka like LibSVM for classification.

Within the scope of the ACILA project, [156, 157] studied automatic recognition of an argumentation structure in legal texts and the subsequent classification of these arguments in the relevant category (e.g. counter argument, rebuttal). In one of their first studies, they examined sentences from different kinds of texts (i.a. newspapers, magazines, court reports or

parliamentary reports) to search for arguments. They attempted several means of representing the text, and used the two classifiers MNB and the Maximum Entropy model. They obtained similar results with both classifiers. In terms of text types, the most arguments were found in non-legal texts (newspapers, with an *accuracy* of 76%), and the fewest in legal judgments (68% *accuracy*). However, they had a small number of legal texts in the training data.

3 Assessment of Machine Learning Frameworks

In the previous chapters, research question two – "How can (active) machine learning support the classification of legal documents and their content (norms)?" – was answered. Regarding research question one – "What are common concepts, strategies and technologies used for text classification?" – common concepts and strategies in the context of TC have been discussed. This chapter presents the overview of technologies normally used within this field, and introduces several machine learning frameworks to answer the rest of research question one.

As there is a multitude of machine learning frameworks[12] with advantages, drawbacks and overlapping issues, a pre-selection was conducted. The problem is that many ML libraries are aligned to a very specific use case and lack broader functionality. Hence, decisive criteria in the pre-selection were, *inter alia*, whether the framework is applicable to TC (e.g., the popular ML framework H_2O is not aligned for TC), the application programming interface (API), the quality of the documentation, the license conditions and the stability of the framework. Furthermore, frameworks providing a Java API are preferred.

3.1 Machine Learning within the Hadoop Ecosystem

Apache Hadoop[13] is an open-source ever-present big-data framework for reliable, scalable and distributed computing released by the Apache Software Foundation. It aims to distribute large datasets across clusters of computers to increase the processing efficiency. The project includes the following four main modules[13] [158]:

- *Hadoop Common:* a set of common utilities like I/O methods and error detection.
- *Hadoop Distributed File System (HDFS):* a file system (not a database) aligned to store large amounts of data at several computer nodes having a master-slave architecture.
- *Hadoop YARN (Yet Another Resource Negotiator):* a framework for job scheduling and cluster resource management.
- *MapReduce:* a parallel data-processing engine.

Besides these main components, there are many other related Apache projects like the databases *HBase* or *Cassandra*, the processing engine *Spark*, and ML libraries like MLlib or Mahout. A section of the architecture of the Hadoop ecosystem is shown in Figure 14.

[12] A valuable listing of existing machine learning frameworks separated by language and context can be found here: https://github.com/josephmisiti/awesome-machine-learning
[13] Apache Hadoop: https://hadoop.apache.org/

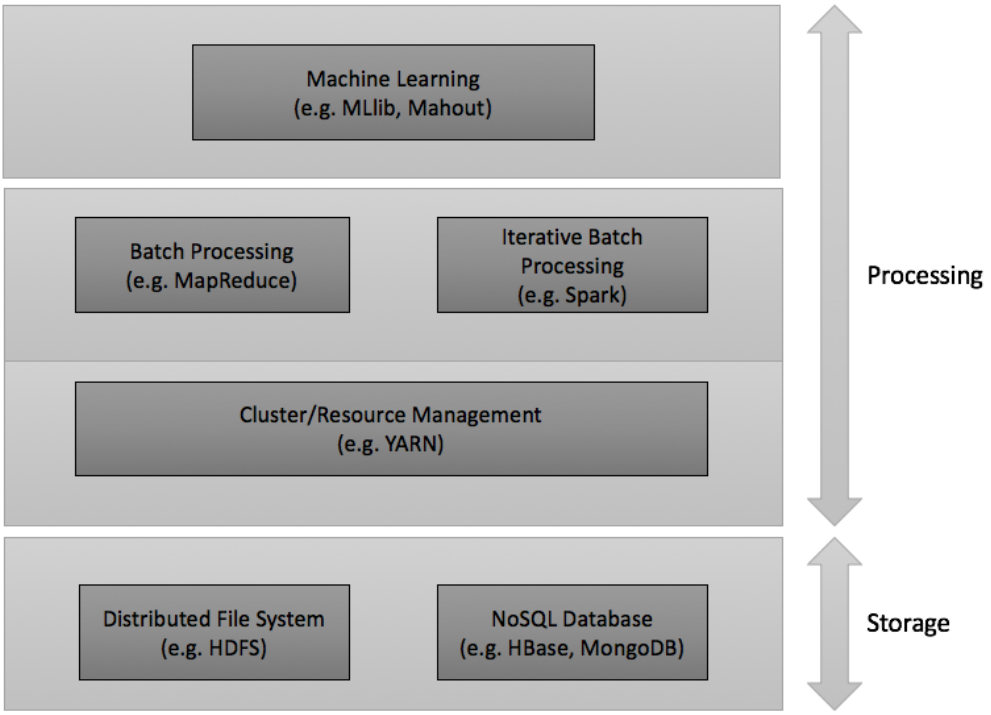


Figure 14: The Hadoop Ecosystem
Source: Own illustration based on [158]

The *storage layer* is the lowest level of the ecosystem. The ecosystem can work with several databases that run on top of HDFS or work as standalone systems. Further, support for non-relational databases is also given. The latter are especially suitable for ML, as databases like MongoDB also support unstructured data [158].

The *processing layer* is the area where the data analysis takes place. Within Hadoop, this layer is founded on the YARN framework. This in turn enables the running of data processing engines like *MapReduce* or *Spark*, and additional tools like ML libraries (e.g. MLlib, Mahout) on top of it. Examples for other processing engines are *H_2O*, *Flink* or *Storm* [158]. Within the scope of this work, the processing engine *Spark* is described more precisely as it can be used with the two ML libraries, MLlib and Mahout.

Apache Spark

Apache Spark[14] is a fast and general open-source cluster-computing framework for large-scale data processing currently available under version 2.1[15]. Spark began as a research project at the University of California, Berkeley, in response to the limitations of MapReduce, and was designed to be fast for interactive queries and interactive algorithms. It was open-sourced in March 2010 and was released under the Apache Software Foundation (ASF) license in June 2013 [159].

[14] Apache Spark: http://spark.apache.org/docs/latest/index.html
[15] Date: January 25th, 2017

Spark provides an ecosystem consisting of several components that can be utilized in the same application. It is written primarily in Scala but also provides APIs in R, Python and Java (see Figure 15). It can run in various ways, such us in a standalone cluster mode, on Hadoop YARN, or just locally, supporting iterative computations.

The *Spark core component* is the foundation of the overall project and contains the basic functionality, including components for memory management, task scheduling, fault recovery and for interaction with storage systems [159].

Another component called *Spark SQL* enables work with structured data within the Spark ecosystem. Spark is designed to process the data directly in the memory of the individual cluster nodes (in-memory processing) and is fault-tolerant. Only if the data size becomes too large is it stored on external sources. In this manner, the Spark SQL component is compatible with HDFS and various other data storages such as Apache Hive, Cassandra, Java Database Connectivity (JDBC) databases, or any other Hadoop data source. By means of the SQL or DataFrame interface, it supports operations on a variety of data formats, for example, JSON, Parquet, LibSVM, CSV and TXT. A DataFrame can be understood as a collection of data (i.e. the Dataset) distributed into named columns conceptually equivalent to a table in a relational database.

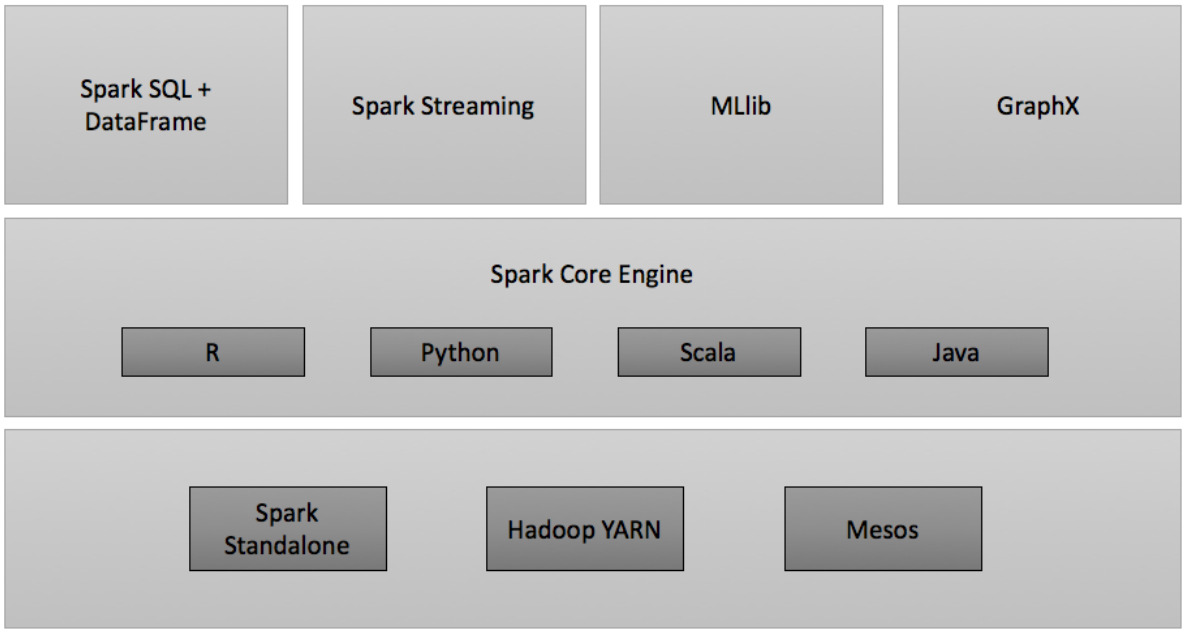


Figure 15: Spark Ecosystem
Source: Own illustration based on [160]

Spark Streaming allows the processing of data streams during run time, and *GrapX* provides a library for the treatment of graphs. In the following section, two machine-learning libraries utilizing the Spark environment are evaluated.

3.1.1 MLlib

MLlib, a Machine Learning Library currently available in version 2.1[16], is a component on top of the Spark core that can be easily combined with its other modules. As it is founded on Spark's iterative batch and streaming approach and in-memory capabilities, it is well suited for large datasets. Through integration with Spark SQL and the DataFrame API, these tools can be used for reading the data and for further processing.

The MLlib library consists of a variety of efficient and scalable implementations of common ML settings. It incorporates various learning algorithms for classification (e.g. NB, SVM, ANN, LR) or clustering (e.g. k-means). However, not all classifiers are able to perform multiclass classification out of the box, but Spark implements the one vs. all strategy in order to conduct multiclass and multilabel classification.

In additional, the library offers methods to transform the unstructured text data into structured data, such as the common bag of words representation and additionally, FS methods (e.g. TF-IDF). Furthermore, there are possibilities to apply preprocessing (e.g. stop-word removal).

The library also includes common evaluation measures like precision, recall and accuracy, and visualization measures like *ROC* and *AUC* curves (but only for binary classification). As it is based on Spark, it provides APIs in R, Python, Scala and Java. However, not every tool is available for all languages.

With the update to MLlib 2.0 end of July 2016, fundamental changes were made to the API. These were the result of the switch from the Resilient Distributed Datasets (RDD)-based API (spark.mllib package) to the DataFrame-based API (spark.ml package). This resulted, among other things, in a more user-friendly and uniform API, and better storage management. Additionally, the ML Pipeline concept was further improved. It allows the execution typical steps of TC (preprocessing, feature selection, classification) in one ML Pipeline[17]. The resulting pipeline models can be persisted subsequently. One disadvantage of ML Pipelines is that not all algorithms can be used within a ML Pipeline. For instance, the SVM classifier, commonly used in TC, cannot be utilized within this environment and is only available via the old RDD-API. However, according to Spark, feature-parity between the two APIs should be reached with the next, larger, update to version 2.2.

In terms of efficiency, Spark and MLlib have set new standards. In 2014, in a benchmark test to assess the speed at which a system could sort 100TB of data, Spark was three times faster using ten times fewer machines than the Hadoop MapReduce implementation (see Chapter 3.1.2). This is even more surprising as all the sorting took place on the HDFS disk without using Spark's in-memory capabilities [161]. When executing an algorithm in-memory (e.g. LR), it is up to one hundred times faster than Hadoop. Considering only the execution of algorithms, Spark MLlib shows excellent performance and is being advanced even further. In the experiment run by [162] comparing the machine-learning frameworks Apache Mahout, running

[16] Date: March 15th, 2017
[17] http://spark.apache.org/docs/latest/ml-pipeline.html (last accessed: May 12th, 2017)

on Hadoop MapReduce, and MLlib, both the efficiency advantage of MLlib and improvement over time are clearly observable (see Figure 16).

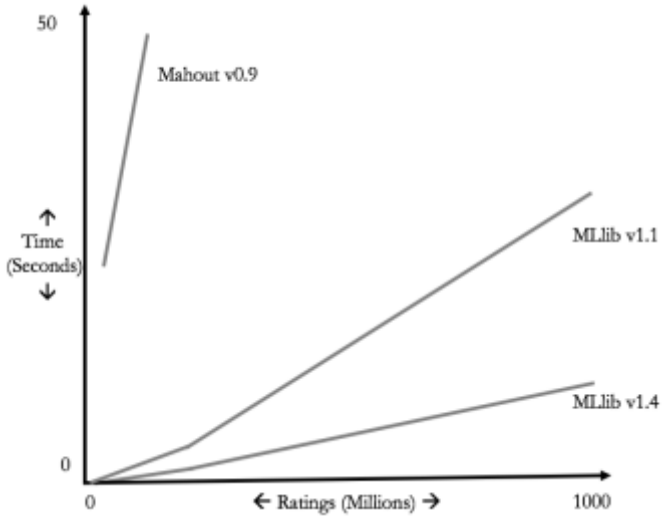


Figure 16: Benchmarking Results for Mahout vs. MLlib Using Alternating Least Squares
Source: Own illustration based on [162]

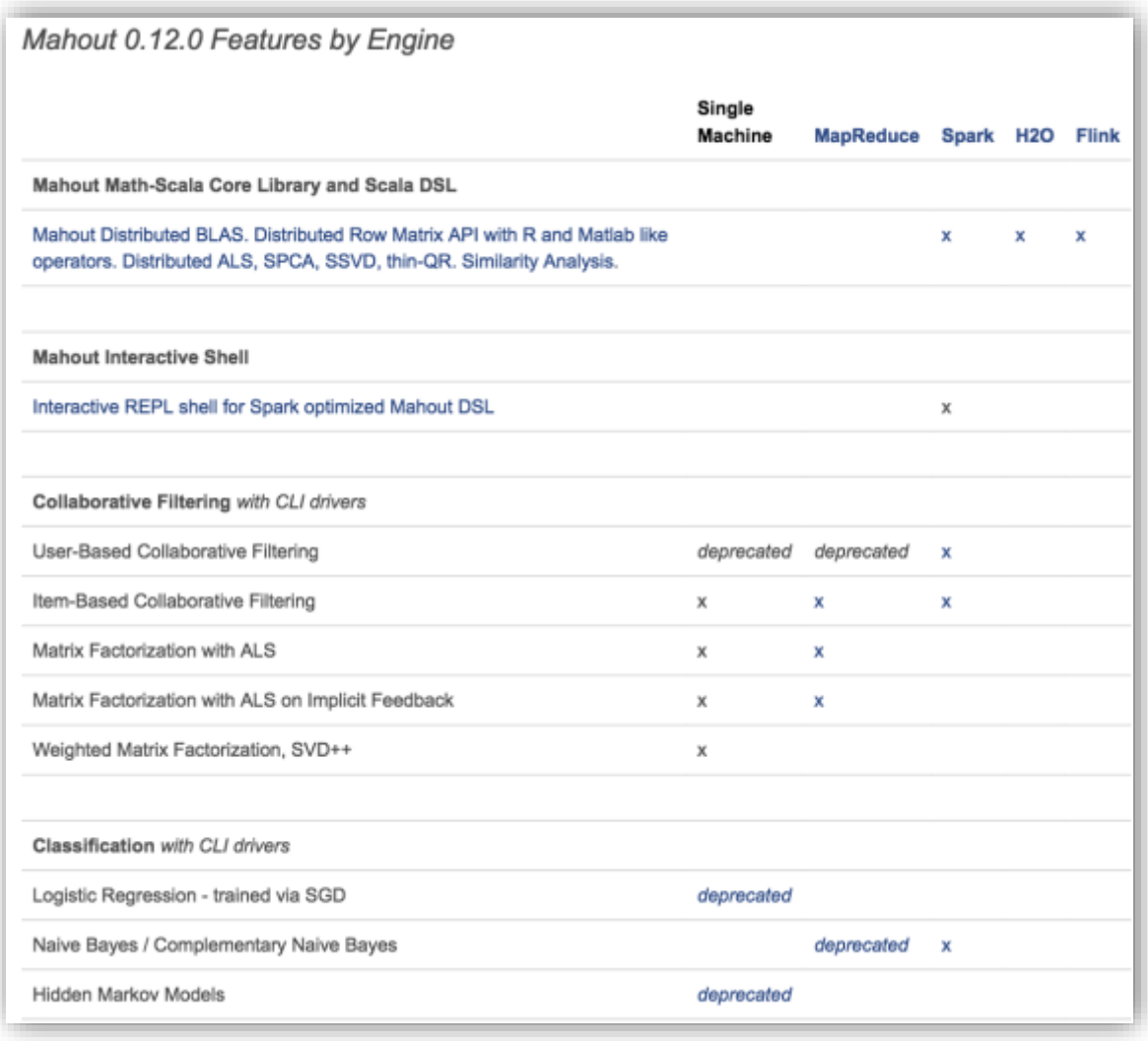
Spark's rapid development was pushed by its strong community and the large number of contributors. It is used in a wide range of organizations, such as Opentable, in production, and more than 1,000 developers have contributed to Spark since 2009.

In addition to the JavaDoc, an extensive documentation is provided, including code examples and details for implementing certain algorithms, on their website for all four APIs. However, in the community forums, the most discussed language is Scala.

We have not found research studies that use Spark MLlib for legal TC, or for AL. However, according to [163], the framework has the potential to develop own machine learning algorithms.

3.1.2 Mahout

Apache Mahout[18] is, like Apache Spark, an open-source project by the ASF. It started as a subproject of Lucene[19] in 2008. Initially, it was driven by the MapReduce distributed-computing framework and implemented with the Hadoop framework focusing on the key-areas recommender engines, clustering and classification [164]. As shown in Figure 16 in the previous section, the MapReduce implementation is no longer up-to-date in terms of efficiency.



Mahout 0.12.0 Features by Engine

	Single Machine	MapReduce	Spark	H2O	Flink
Mahout Math-Scala Core Library and Scala DSL					
Mahout Distributed BLAS. Distributed Row Matrix API with R and Matlab like operators. Distributed ALS, SPCA, SSVD, thin-QR. Similarity Analysis.			x	x	x
Mahout Interactive Shell					
Interactive REPL shell for Spark optimized Mahout DSL			x		
Collaborative Filtering *with CLI drivers*					
User-Based Collaborative Filtering	*deprecated*	*deprecated*	x		
Item-Based Collaborative Filtering	x	x	x		
Matrix Factorization with ALS	x	x			
Matrix Factorization with ALS on Implicit Feedback	x	x			
Weighted Matrix Factorization, SVD++	x				
Classification *with CLI drivers*					
Logistic Regression - trained via SGD	*deprecated*				
Naive Bayes / Complementary Naive Bayes		*deprecated*	x		
Hidden Markov Models	*deprecated*				

Figure 17: Part of Mahout's Ecosystem
Source: http://mahout.apache.org/users/basics/algorithms.html (Accessed 30.1.2017)

[18] Apache Mahout: http://mahout.apache.org/
[19] Lucene: http://lucene.apache.org/core/

For this reason, with the release 0.10 in April 2015, the Mahout project shifted away from using MapReduce[20]. The focus is now on a math environment called Samsara, which provides statistical operations, linear algebra, and data structures. Unlike most of the other ML libraries, the objective of Mahout-Samsara is to provide an extensible programming environment for Scala so that users can develop their own distributed algorithms, instead of providing a machine-learning library with existing algorithms. Furthermore, different distributed engines such as Spark or H_2O are supported.

Mahout can, therefore, be seen as an "add-on" to Spark and its technologies. It is possible to utilize Spark's components, such as the DataFrame API, or even combine MLlib methods with Mahout algebra[20].

Currently[21], Mahout is available in version 0.12, released in June 2016. Figure 17 gives an impression of how the concept of Mahout has changed away from being a machine-learning library for classification to being an "add-on" for other processing engines. All three classification algorithms of the original library (before v0.10) are now marked as deprecated. Only an implementation of MNB is provided by the Mahout ML library, which is further optimized in the Spark module[22].

The environment now consists of an algebraic backend-independent optimizer (Mahout math-Scala core library), and a Scala domain specific language (DSL) consolidating the distributed and in-memory algebraic operators.

Hence, Mahout is an ML engine that requires deeper knowledge of both mathematical and programming skills. Moreover, due to the many dependencies, the initial configuration of Mahout may be difficult. The sparse documentation makes this even more challenging. Especially with regard the method for applying TC with Mahout and Lucene, hardly any documentation is available.

For evaluation of NB, Mahout provides common metrics, such as the confusion matrix or the ROC curve.

Compared to Spark, and thus also MLlib, community support is low with only 24 contributors. The result is that the development of the project has moved relatively slowly; for instance, the current stable version is only 0.12, although the project is more than 9 years old.

[20] http://www.weatheringthroughtechdays.com/2015/04/mahout-010x-first-mahout-release-as.html
[21] Date: March 15th, 2017
[22] https://mahout.apache.org/users/algorithms/spark-naive-bayes.html

3.2 Weka

Waikato Environment for Knowledge Analysis (Weka)[23] is a general-purpose open-source software workbench incorporating several ML techniques for data-mining tasks. It was originally developed as an internal project written in C at the university of Waikato in New Zealand. It was made publically available in 1996 and rewritten in Java for the 3.0 release in 1999 [165]. Today, it is available as version 3.8 under the GNU General Public License[24].

Weka provides a comprehensive collection of data-preprocessing tools and ML algorithms that are widely used in combination with related projects for research purposes[25]. One reason for this prevalence is the graphical user interfaces (UI) (e.g. Explorer, Workbench) that facilitate easy access to the underlying functionality for data analysts who are not particularly familiar with programming (see Figure 18). Using the Weka Explorer can result in problems when training large datasets as the Explorer always loads the entire dataset into the computer's main memory.

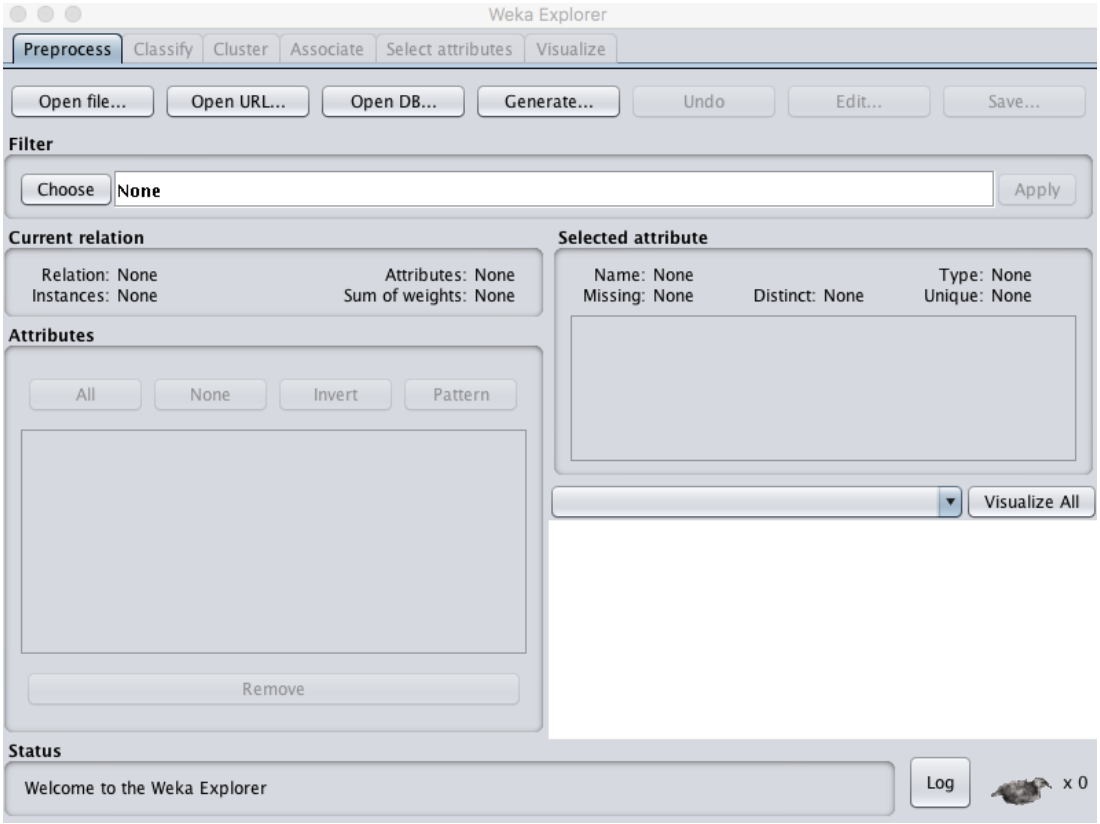


Figure 18: Weka Explorer
Source: Own screenshot

[23] Weka: http://www.cs.waikato.ac.nz/ml/index.html
[24] Date: March 15th, 2017
[25] For a list of publications in which WEKA or a related project like MOA, MEKA or Mulan are used or at least mentioned see: http://www.cs.waikato.ac.nz/ml/publications.html + http://weka.wikispaces.com/Related+Projects

However, Weka also offers a Java interface to integrate it in one's own application[26]. The problem with large datasets is addressed by providing an interface called "UpdateableClassifier". Classifiers implementing this interface can be trained incrementally. That way the data does not have to be loaded into memory all at once. Unfortunately, only a small selection of classifiers is implementing this interface (e.g. "NaiveBayesMultinomialText", a MNB implementation for text data). Alternatively, one can make use of the library accessing the related MOA data stream software to handle large datasets.

In principle, Weka on its own provides implementations of all relevant preprocessing steps (stemmer, stop-word remover, etc.), as well as unsupervised and supervised ML algorithms, such as NB, Perceptron or SVMs. For the evaluation of the classifier's performance, common measures such as precision, recall and F_1-Measure and the ROC-curve for visualization are available.

Additionally, there are many software projects that are related to Weka, which they use in some form[27]. Most of these use regular Weka classifiers and enhance the functionality in some manner. On its own, Weka provides only algorithms for binary and multiclass classification. However, Mulan[28] and Meka[29] extend the original functionality of Weka and allow multilabel classification. As these projects are often relatively small, the documentation is frequently sparse and a full stability is not always given.

Weka can access files, URLs and both SQL data sources like MySQL or Oracle, and NoSQL databases like MongoDB using JDBC. Furthermore, since release 3.8, there have been attempts to apply distributed data mining using, for instance, Hadoop- or Spark-specific wrappers in order to improve the efficiency by offering access to distributed computing. For classification, Weka usually uses its own Attribute-Relation File Format (ARFF) consisting of a header section (e.g. information about the attributes and their types) and a data section. Additionally, LibSVM, CSV, C4.5, TXT or JSON formats are supported directly or indirectly. Although Weka was originally not designed for TC, it can handle textual data[30]. The textual data ("String") must first be processed using suitable filters such as the "StringToWordVector" that converts the text into a word vector. This enables further processing, for example, applying FS using the common TF-IDF method and the subsequent use of classification algorithms.

A suitable benchmark for classification efficiency with Weka could not be found. However, as computations with datasets that are not excessively large are performed in the computer's main memory, it should be quite fast. For large datasets, the efficiency decreases as the data must be retrieved iteratively from external storages. These shortcomings are addressed by the recent addition of the aforementioned wrappers for distributed computing or the related MOA project.

---

[26] https://weka.wikispaces.com/Use+WEKA+in+your+Java+code
[27] http://weka.wikispaces.com/Related+Projects
[28] Mulan: http://mulan.sourceforge.net/
[29] Meka: http://meka.sourceforge.net/
[30] http://weka.wikispaces.com/Text+categorization+with+WEKA

Weka provides extensive documentation about ML in general, and about the implementations of these algorithms in Weka [166]. Apart from the Javadoc documentation, there is also some documentation that includes Java code examples on how to integrate Weka in one's own application. However, this documentation is quite sparse and could be more detailed. As it is a part of many research projects, Weka has active communities, such as pentaho[31] behind it and is still under active development.

The Weka toolkit has already been used successfully in the context of legal TC [150, 155] and in an AL environment [11].

[31] Pentaho: http://community.pentaho.com/

3.3 scikit-learn

The *scikit-learn* framework[32] is an open-source general-purpose ML library for the Python programming language (there is no API for Java). It is currently available under version 0.18.1 released under the Berkeley Software Distribution (BSD) license[33]. The project started as a Google Summer of Code project in 2007, aiming to provide non-ML experts with access to an efficient tool that is reusable in different scientific areas and various contexts. As it is built upon *NumPy*[34] and *SciPy*[35] (scientific Python), these libraries must be installed in advance.

Scikit-learn incorporates with several options for data mining, data analysis and visualizations (see Figure 19). They have a vast choice of both supervised and unsupervised algorithms for classification (e.g. SVM, NB), regression algorithms (e.g. LR, Bayesian Regression) and clustering (e.g. kNN). A further advantage of scikit-learn is that all classifiers do multiclass classification out-of-the-box. To conduct multilabel classification, the library offers an implementation of the one vs. all strategy.

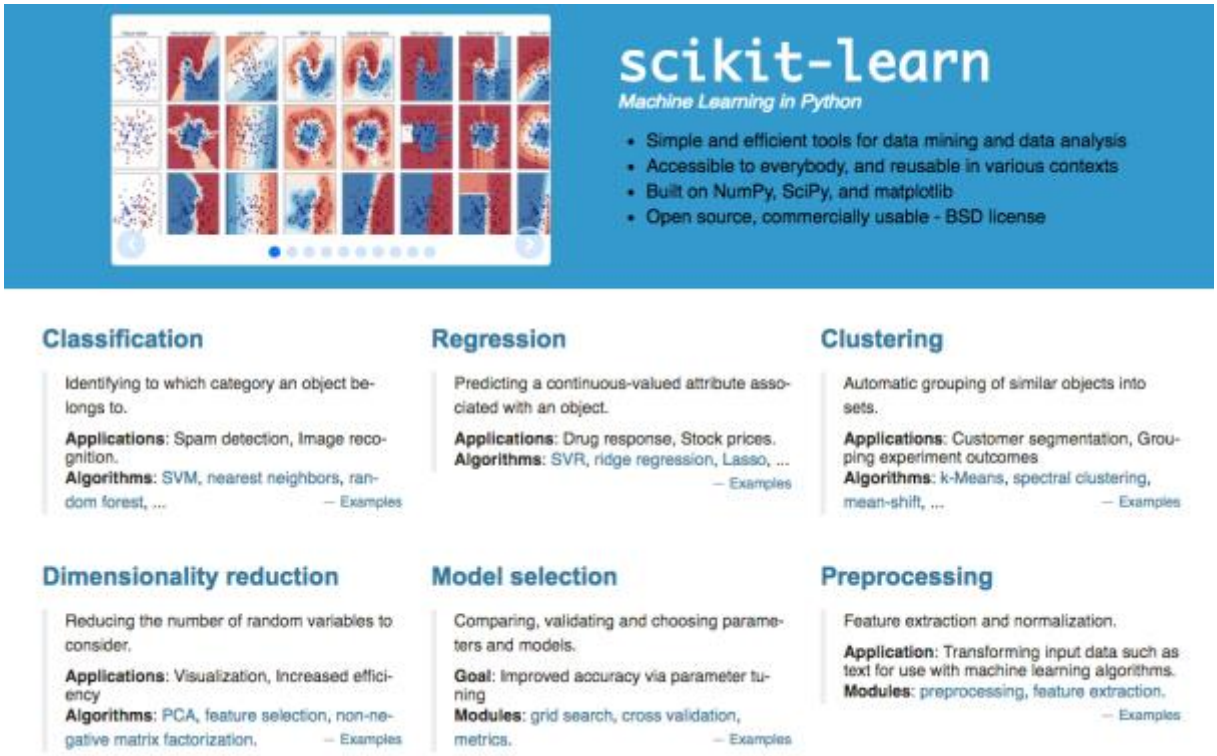


Figure 19: Overview of scikit-learn
Source: Screenshot of the homepage[32]

For large datasets, additional incremental implementations of classifiers (e.g. NB, Perceptron) are provided, for which incrementally only a small number of instances is loaded into the main memory.

---

[32] Scikit-learn: http://scikit-learn.org/
[33] Date: March 15th, 2017
[34] NumPy: http://www.numpy.org/
[35] SciPy: https://www.scipy.org/

Although it is a general-purpose ML framework, scikit-learn offers appropriate instruments for working with textual data[36]. The text is transformed into a feature vector (bag of words representation) in order to generate a suitable text representation for further processing. Besides this simple word vector, FS techniques, such as the TF-IDF methods, are also supported.

In order to improve the quality of the training set, scikit-learn offers a few (but not all) common tools for the preprocessing of text data, such as the filtering of stop-words and stemming. Generally, the library comes along with a useful selection of FS methods[37].

Like MLlib, scikit-learn also supports the pipeline concept, in which several steps (e.g. conversion into feature vector, preprocessing, classification) can be combined.

For evaluation of the model[38], scikit-learn includes all common classification evaluation metrics such as accuracy, F_1, precision and recall, and the AUC.

Besides text data, scikit-learn also provides utility functions to load datasets in the SVMLIGHT/LibSVM format. Furthermore, it is possible to convert external datasets, such as pandas DataFrame, that support common formals including CSV, JSON or SQL.

To provide reusability, it is possible to save and load created models by using Python's built-in persistence model for object serialization, called *pickle* or *joblib*, especially suitable for large datasets.

The documentation provided by the scikit-learn developers is quite extensive and covers all important parts of the API, including code examples, explanations and tutorials. Additionally, it offers a 2007-page pdf-documentation text with a very detailed description of scikit-learn, as well as of common ML topics and algorithms.

With 100 contributors[39] since 2010, scikit-learn has a remarkable and very active community that is also used for commercial purposes, e.g. by Spotify.

[36] http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
[37] http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction
[38] http://scikit-learn.org/stable/modules/model_evaluation.html
[39] https://github.com/scikit-learn/scikit-learn/graphs/contributors

3.4 Mallet

The MAchine Learning for LanguagE Toolkit (Mallet)[40] is an open-source ML package written in Java, aligned for statistical NLP, especially for DC and other NLP-related methods, such as topic modeling. It was developed at the University of Massachusetts, Amherst, by Andrew McCallum and his team, and released in 2002 [167]. Currently, it is available in version 2.0 under the Common Public License[41].

Unlike the other general-purpose ML frameworks, Mallet is specifically designed for NLP and thus textual data-like documents. It incorporates several useful tools for TC. To embed Mallet's ML tools into one's own application, a Java API is provided. However, compared to the other frameworks, Mallet is a relatively small project having no extensive functionality.

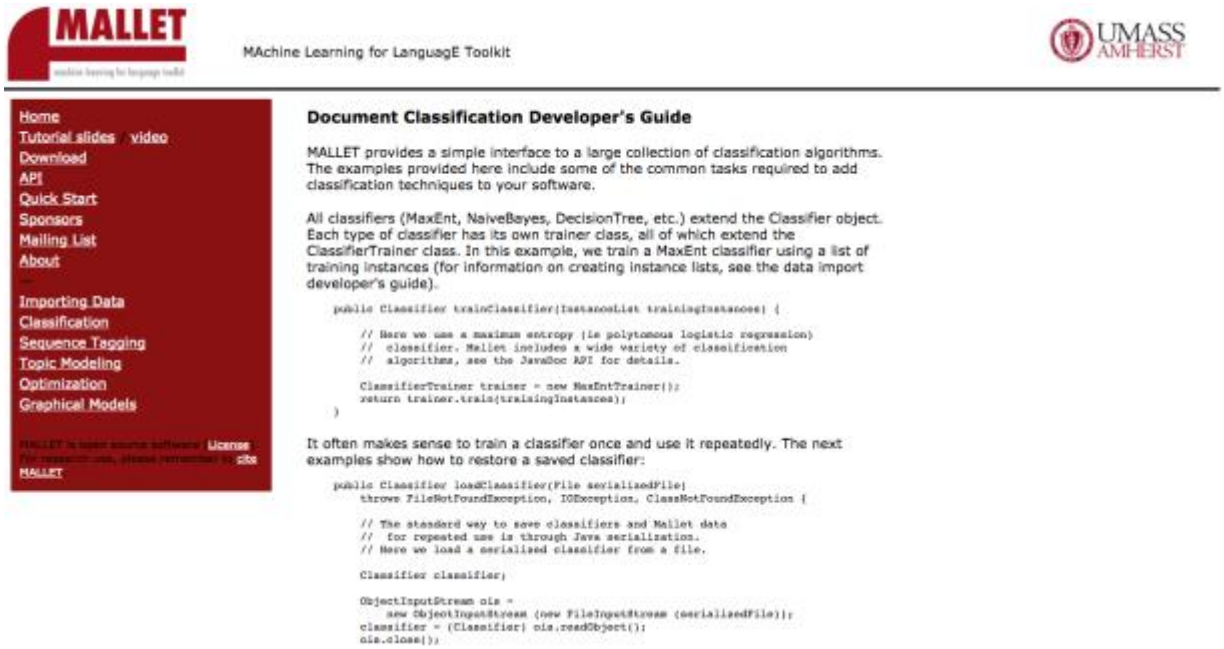


Figure 20: Part of Mallet's Developer Documentation
Source: Own screenshot

Similar to the other libraries, Mallet offers routines to transform text documents into numerical representations (feature vectors) so that the classifiers can process them efficiently. Commonly used FS methods, such as the TF-IDF method, are not possible with Mallet.

To work with textual content, the data must be transformed in a specific Mallet format called an "instance"[42]. An instance represents a training example and has the four fields: "name" (identifier), "label", "data" (feature vector) and "source" (original file or string). These instances are stored in a "InstanceList," utilizing the pipeline implementation in a manner

[40] Mallet: http://mallet.cs.umass.edu/
[41] Date: March 15th, 2017
[42] http://mallet.cs.umass.edu/import.php

similar to Spark. Within this pipeline it is possible to apply several preprocessing steps, for example, conversion into lowercase letters or the removal of stop-words.

The import of data is only possible via the local file system. Most Mallet classes therefore use plain Java serialization to store and load models and data. Mallet only directly supports the data formats text and SVMLight. In addition, it is possible to store the trained classifier on the local file system and use it later for prediction.

With respect to classification algorithms, Mallet does not have that large range of options compared to other frameworks. Of the classifiers described in detail in the previous chapters, only NB is offered. Multi-label classification is not supported. However, for the evaluation of classifiers, common measures like accuracy, precision and recall, and the F_1 are available.

In contrast to Spark or Mahout, and similarly to Weka, Mallet is generally not aligned to import or process extremely large amounts of data as this can lead to problems with the computer's main memory. Nevertheless, the library itself contains cleverly optimized code and algorithms.

On the homepage one can find tutorial presentations about the concept Mallet implements. Additionally, there is a brief developer guide demonstrating the classification process, including some code snippets (see Figure 20). However, most of the documentation is only available in form of JavaDoc.

When looking at the project at github[43], one can observe that Mallet is still undergoing active development, having more than 20 contributors in the last years.

Compared to the other ML frameworks described, Mallet has the advantage that it provides an additional API for AL. By implementing the interface "ClassifierTrainer.ByActiveLearning"[44], the classifier trainer will select certain instances and request that they be labeled. In the context of AL, Mallet was used, for instance, as ML library by Settles for the implementation of the aforementioned AL application "Dualist" [121]. However, he did not use the AL interface.

[43] https://github.com/mimno/Mallet/graphs/contributors
[44] JavaDoc: http://mallet.cs.umass.edu/api/cc/mallet/classify/ClassifierTrainer.ByActiveLearning.html

3.5 Summary and Conclusion

The results of the previous evaluation chapters are summarized in Table 9. This table is not complete and does not show all features of the respective ML framework. For instance, although the processing engine H_2O has also a powerful ML engine, it is not listed as it is not relevant for this work.

In general, an evaluation and comparison of the ML frameworks discussed (as well as those not mentioned) is quite difficult. Each ML framework has its advantages and disadvantages. The larger libraries especially, such as MLlib and Mahout, that are part of a vast ecosystem have various configuration possibilities. For instance, the efficiency certainly depends on the implementation of the algorithm, but also on the setup used (standalone, as a cluster, etc.). Therefore, although MLlib is, when used as a cluster-computing framework, superior to the other libraries in terms of efficiency and suitability for big data, it is not particularly relevant for this study as the classification is performed on a single computer. For this reason, terms such as efficiency or scalability are not listed in this table.

Table 9 presents a general overview of the respective ML frameworks discussed in the previous chapters.

	MLlib	**Mahout**	**Weka**	**Scikit-learn**	**Mallet**
Current Version (2nd Feb. 2016)	2.1	0.12.2	3.8	0.18	2.0.8
License	Apache Software Foundation (AFS)	Apache Software Foundation (AFS)	General Public License (GPL)	Berkeley Software Distribution (BSD)	Common Public License (CPL)
Open Source	yes	yes	yes	yes	yes
Popular Users	OpenTable, Verizon	?	Pentaho	Evernote, Spotify	?
Processing Platform	MapReduce (deprecated), Spark	Spark	wrapper for Spark available	none	none
Interface Language	Java, Scala, Python, R	Mainly Scala, Java (for older versions)	Java, R	Python	Java
Suitable for large datasets	yes	yes	partially	yes	partially
Community Support	good	moderate	good	good	moderate
Documentation	very good	moderate	good	very good	good
Support for NLP/Textual Data /AL	good	moderate (via Lucene)	good	good	very good
Configuration	simple [c]	difficult [c]	simple [c]	simple	very simple

	MLlib	Mahout	Weka	Scikit-learn	Mallet
Classification Algorithms					
NB	✓ [a]	✓ (Spark optimized)	✓ (batch, incremental)	✓	✓
SVM	✓ [b]	only via MLlib	✓ (batch)	✓	/
MLP	✓ [a]	only via MLlib	✓ (batch)	✓	/
Multiclass Classification	one vs. all	only via MLlib	one vs. all, one vs. one	one vs. all, one vs. one	/
Multilabel Classification	one vs. all	/	by extensions (e.g. Mulan, Meka)	one vs. all	/
Pipeline	✓ [a]	/	/	✓	✓
Total Algorithm Coverage	very high	high	very high	very high	moderate
Preprocessing					
Stemming	(by extensions (Snowball))	/	✓	✓	✓
Stop Words	✓	/	✓	✓	✓
Coverage of Feature Selection Methods	very high	low	very high	very high	moderate
Text Representation					
Word Vector	✓	✓	✓	✓	✓
TF-IDF	✓		✓	✓	
Evaluation					
Confusion Matrix	✓	✓	✓	✓	✓
ROC/AUC Curve	✓	/	✓	✓	✓

Table 9: Comparison of Machine Learning Frameworks
**a: available via the DataFrame-API (new API) *b: available via the RDD-API (old API)*
**c: complexity depends on the configuration (e.g. combination with other tools)*

Mahout has changed significantly in the last years to adapt to Spark. With the new concept, it does not fit very well for this use case.

While *Mallet* has the advantage that it can be used very easily and is aligned for TC, having even an AL interface, it lacks the choice of classification algorithms and other TC techniques (e.g. FS methods).

Weka is also a widely used and useful tool for ML that has already been used in an TC experiment applying AL. However, the documentation for TC is very sparse and for larger data sets it also lacks suitable classifiers.

MLlib and *scikit-learn* are both libraries having a vast choice of tools for text processing, classification, good documentation and an active community. Thus, both would be a good option for implementing a ML application. But as noted, libraries having a Java API are favored for this study. Hence, MLlib version 2.1 is used as the ML framework for building a prototypical implementation of an AL microservice for the classification of legal documents and norms.

4 Concept and Design

Following the discussions of the TC process, the theory of AL in the context of legal TC, and having analyzed and chosen a ML framework, these understandings are applied to the prototypical development of an independent AL-microservice using Spark MLlib. For the work with legal content, use will be made of an existing legal data science environment called Lexia [168].

In the following chapters, both Lexia and LexML, the AL-prototype developed within the scope of this thesis, are described in greater detail, as is the interaction between the two services. Based on the existing Lexia framework described in this chapter (see Chapter 4.1) and the findings in the theoretical component of this thesis, requirements for the LexML service are derived (see Chapter 4.2). Thereafter, the fundamental architecture of LexML and its communication interface (see Chapter 4.3) are discussed.

4.1 Lexia Framework

Lexia, developed within the scope of the interdisciplinary research program *Lexalyze*[45] at the Technical University of Munich (TUM), is a "data science environment for the semantic analysis of German legal texts" [168]. In what follows, a short overview of the system and the components relevant to this thesis are provided. For a more detailed review, please refer to the papers of Waltl et al. (e.g. [168, 169]).

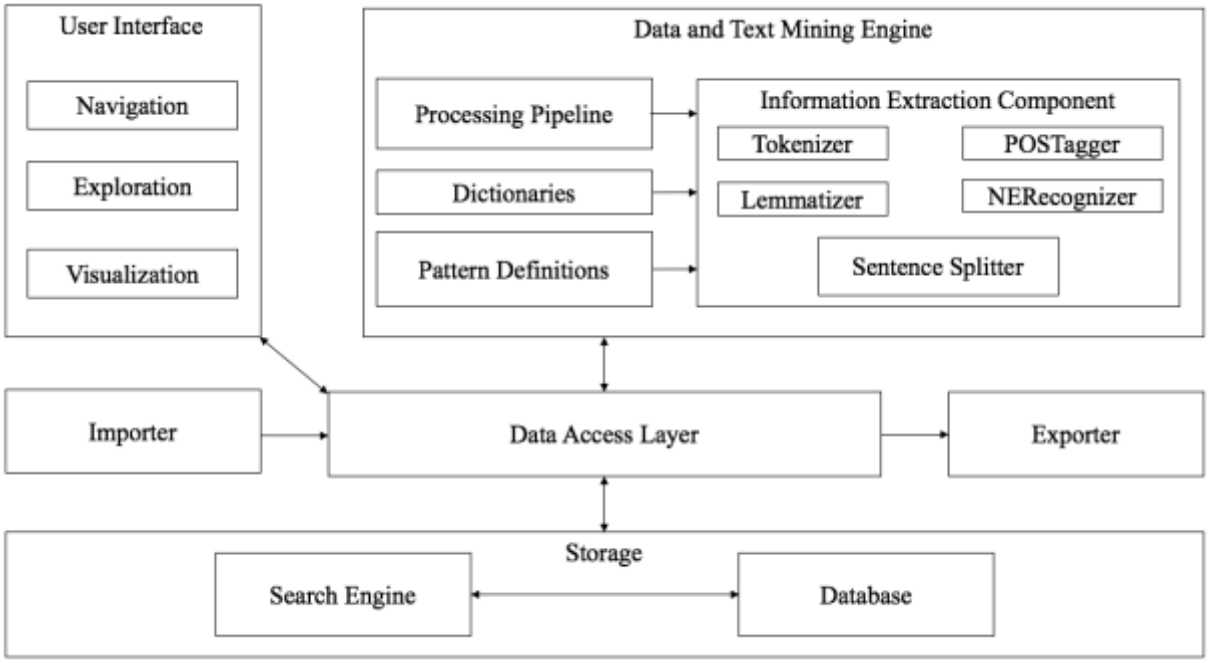


Figure 21: Main Components of Lexia
Source: Own illustration based on [168]

[45] http://www.lexalyze.de/

Lexia is a powerful collaborative web application consisting of the following main components: an *importer*, an *exporter*, a *data storage* and *access layer*, a *text-mining engine*, and a *user interface*. The project was implemented with a Java back-end using the web application framework *Play*[46] and an Elasticsearch database to guarantee efficient access to the text data, including a full-text search. An overview of major components is presented in Figure 21.

Through the *Importer*, legal texts in various formats such as HTML, XML or PDF can be imported into the Elasticsearch database of Lexia. The system differentiates between several legal document types such as *laws, judgments, contracts, patents* and *miscellaneous* (generic legal documents). All types are derived from the abstract superclass *LegalDocument*. The *LegalDocument* class, in turn, contains the *document content* stored in *Section Containers* and then in *Sections*, or directly in *Sections*, to handle the nested structure of legal texts (e.g. in laws). *Sections* represent the actual textual content (e.g. norms). Each of these *Sections* can have *Annotation* objects of a certain type reflecting the outcome of the *Data* and *Text-Mining Engine*.

The *Data* and *Text-Mining Engine* is the heart of Lexia, built primarily on the Apache Unstructured Information Management Architecture (UIMA) Java framework[47]. This component contains an *Information Extractor Component* which has the capability to extract and annotate semantic information in legal texts. This process is supported by, for instance, dictionaries and by two kind of pattern definitions: (1) regular expressions (regex) and (2) ruta scripts (rule-based text annotation). The latter enables means of specifying more elaborate rules and also recognizes complex linguistic structures.

Further, the UIMA architecture utilizes the use of pipelines to process legal texts, meaning that various tasks (e.g. Sentence Splitter, Tokenizer, POSTagger) of the aforementioned *Information Extractor Component* can be concatenated and executed in "one step" (see Figure 22). This allows efficient processing of legal texts into both *linguistic Annotations* (e.g. annotating a sentence as a sentence, or annotating the verbs of a specific sentence), and *semantic/legal Annotations* (e.g. annotating a specific sentence as a legal definition). These different *Annotation* types are illustrated in Figure 23 (Linguistic Entities and Legal Entities).

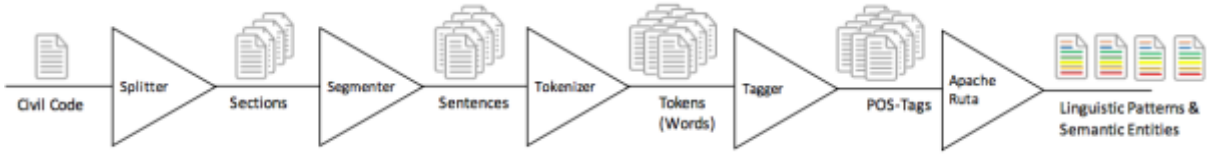


Figure 22: Processing Pipeline for Determining Linguistic Patterns with Apache UIMA and Ruta
Source: [168]

[46] https://www.playframework.com/
[47] https://uima.apache.org/

To enable simple access for legal experts, Lexia incorporates with a simple UI. Here, a user can import legal texts, select a certain pipeline and one or more ruta scripts to be processed, or see the result of a processed pipeline. There are many additional routes of navigation that are not relevant for this thesis and consequently not described here. An example of the UI showing an annotated/processed law can be seen in Figure 23.

Figure 23: Lexia User Interface Showing Different Annotation Types
Source: Own screenshot

In summary, Lexia is a powerful environment for analyzing legal textual data linguistically and semantically. However, in its current state, this knowledge is only derived via rules. For instance, a sentence is only annotated as a legal definition if a certain pattern defined in a ruta script occurs. As stated previously, many patterns containing all the linguistic variations must be defined to find most of the semantic information in a legal text. Furthermore, defining these patterns requires strong legal-domain knowledge and a time-consuming analysis of the text.

Another issue is that the type of document (e.g. law, judgement) must be known before the import, or it has to be noted in the document, so that Lexia can assign it to the correct class. If a legal document could be assigned by a computer without its type having been stating explicitly, the import could be simplified and speeded up.

To further enhance this functionality, it is part of this work to analyze whether AL is a promising alternative for the classification of legal documents and their content (norms). To this purpose, the infrastructure and features of Lexia's existing legal data science environment are used to develop an independent AL-prototype (microservice) in which the ML takes place. The objectives of, and the resulting requirements for, this microservice are derived in the following section.

4.2 Objectives and Requirements

As described in the previous section, the objective of this work is to develop an independent prototype in order to classify legal documents (DC) and their content (NC) by using a classifier. The following essential objectives and requirements were derived from both the literature review, and from the current state of Lexia and the technical prerequisites of Spark MLlib.

To clarify the relevance, the keywords *shall*, *should* and *may* specified by the ISO[48] are used: *shall* indicates a requirement, *should* a recommendation, and *may* is not mandatory, but useful to have. Requirements that are essential, for instance for the evaluation of this work, receive higher priority.

Furthermore, a distinction is made between functional requirements (FR) and non-functional requirements (NFR).

4.2.1 Functional Requirements

First, the FR affecting LexML's behavior, and hence design, are described in the following.

FR01 Perform Legal Text Classification

LexML *shall* have the capabilities to perform legal TC on imported legal data (see FR02). The imported legal data can be either a set of documents or a set of sentences from one document.

FR02 Import of Legal Texts

Legal texts that are already imported in Lexia *shall* be used for classification in LexML. This leverages the existing functionality of Lexia and complements it with the idea of performing and evaluating the potential of AL for German legal texts. These legal texts *shall* be assigned to a specific AL pipeline and saved in LexML's database (see FR06).

Further, it *may* be possible to limit the number of legal documents to import into LexML.

FR03 Utilize Lexia's Existing Environment

As described in Chapter 4.1, the existing pipeline architecture of Lexia *shall* be utilized in three ways.

First, it *shall* be possible to import the sentences of a legal document into LexML (see FR02). For this purpose, Lexia's sentence splitter is utilized to save each sentence of a legal document as a single annotation object.

[48] https://www.iso.org/foreword-supplementary-information.html (last accessed: May 7th, 2017)

Second, it, *should* be possible to import only specific linguistic structures of a legal document or sentence (e.g. to import only adverbs of a sentence). Ruta scripts allow one to find different types of words, such as verbs, nouns and adverbs.

Both methods require that the legal document has been preprocessed by one of Lexia's pipelines.

Additionally, Lexia's existing UI *should* be complemented in a way to fulfil all FR and NFR in the best possible manner.

FR04 Multiclass Classification

As seen in Chapter 2.1.1, there are three types of classification problems (binary, multiclass, and multilabel). LexML *shall* be designed to perform and evaluate multiclass classification.

FR05 Use Spark's ML Pipeline Concept

The Spark ML Pipeline concept discussed in Chapter 3.1.1, in which several tasks are concatenated (e.g. tokenizer, stop-word removal, classifier) *shall* be used in LexML to perform efficient AL and TC. The resulting pipeline model should be persisted in order to be reusable later (see FR10, FR11).

FR06 Use of AL Pipelines

It *shall* be possible to create and remove an AL pipeline in LexML from the UI of Lexia. An AL pipeline implements Spark's ML concept for TC (see FR05). Each of these AL pipelines *shall* have a unique identifier (name), be assigned legal textual data for training (see FR08) and testing (see FR10, FR11), and contain options to configure specific AL settings (see FR07).

FR07 Configure AL Pipelines

In Chapter 2.2, the influencing factors of AL were discussed. To create a realistic AL environment, it *shall* be possible to configure the following basic settings of an AL pipeline:

- type of pipeline (Document Classification or Norm Classification),
- labels,
- classifier (Naïve Bayes, Logistic Regression, Perceptron),
- query strategy (Uncertainty Sampling, Query by Committee),
- size of seed set,
- batch size,
- minimum and maximum learning rounds (stopping criteria), and
- Lexia annotations used (see FR03).

FR08 Train Classifier Using AL Pipeline

Only after the import and the configuration of the AL pipeline, *shall* it be possible to start this AL pipeline to begin the iterative AL process and train the classifier. The first labeling round *shall* be done with a set of randomly queried instances (the seed set). Following this, the actual learning by the model and the labeling *shall* take place iteratively, based on the specified configurations until one stopping criterion is met.

FR09 Save Classifier/AL Pipeline Model

At the end of a learning process, the resulting classifier/pipeline model *shall* be persisted due to FR10 and FR11. Furthermore, this is also a precondition to proceed with the learning at a later point of time without having to restart the process with random instances.

FR10 Predict with AL Pipeline

For testing the performance of a classifier, it *may* be possible to predict the labels of the remaining unlabeled textual data. The difference from FR11 is that in this case the imported documents or sentences do not have any test set to evaluate the performance. This implies that the correctness of the predictions must be checked manually.

FR11 Evaluate AL Pipeline

It *shall* be possible to evaluate the trained model with a test set independent of the training set in order to obtain evaluation metrics discussed in Chapter 2.1.2.7. In contrast to FR10, for the evaluation of an AL pipeline a sufficiently large test must be available. For this purpose, the following multiclass evaluation measures offered by Spark MLlib[49] should be incorporated:

- Accuracy,
- Weighted precision (macro-averaging),
- Weighted recall (macro-averaging),
- Weighted F-measure (macro-averaging),
- Precision by label,
- Recall by label,
- F-measure by label, and
- Confusion matrix.

Additionally, it *shall* be possible to indicate whether the data necessary to create learning curves (see 2.2.8 Evaluation) should be exported during learning (see FR13). This implies that the system *shall* be able to evaluate the model obtained from each learning round against the test set.

[49] http://spark.apache.org/docs/latest/mllib-evaluation-metrics.html (last accessed: May 7th, 2017)

The evaluation is more highly prioritized than the prediction (FR10), as the evaluation of DC and NC is an important part of this study.

FR12 Transparency

The system *should* give feedback during the learning process, for example, showing the prediction (label) of the model for each instance and its confidence about it. Further, it *should* be possible for the user to see the evaluation results in detail, for instance, which document or sentence was predicted or evaluated correctly.

FR13 Export of Evaluation Results

To evaluate the learning process, it *shall* be possible to export the evaluation metrics described in FR11 in an xlsx file. This export *shall* be possible after each learning round and at the end of learning process.

4.2.2 Non-Functional Requirements

There are also several non-functional requirements (NFR) that influence the system's architecture primarily that are defined below.

NFR01 Simple User Interface

It *shall* be easy for the user to create, configure and process (train, evaluate and predict) an AL pipeline. Therefore, there *should* be a clear separation between these three steps at the UI. The user *shall* only navigate in the UI of Lexia as LexML does not provide an own UI.

NFR02 Maintainability of Software Architecture

The software architecture *should* support the reuse of the components.

NFR03 Extensibility of Software Architecture

The adding of new functionality like additional classifiers or query strategies *should* be possible without considerable refactoring.

NFR04 Data Exchange

Lexia and LexML *shall* communicate via Rest API using the standardized data-exchange format JSON.

NFR05 Use Spark MLlib as ML Framework

As analyzed in Chapter 3, Spark MLlib is an efficient ML framework, also suitable for TC. Hence, it *shall* be used in LexML.

NFR06 Simulation of AL

It *shall* be possible to simulate the AL process (when having a sufficiently large training and test set) in order to speed up the evaluation process for this study.

4.2.3 Summary and Prioritization

Table 10 provides an overview of the FR and NFR requirements described and their priority.

The priority column reflects the ISO-keywords as a number scaled from 1 to 5, where 5 indicates highest priority:

- may ≅ priority 1 – 2
- should ≅ priority 3
- shall ≅ priority 4 – 5

ID	Requirement	Priority
FR01	Perform Legal Text Classification	5
FR02	Import of Legal Texts	5
FR03	Utilize Lexia's Existing Environment	4
FR04	Multiclass Classification	5
FR05	Use Spark's ML Pipeline Concept	4
FR06	Use of AL Pipelines	5
FR07	Configure AL Pipelines	5
FR08	Train Classifier Using AL Pipeline	5
FR09	Save Classifier/AL Pipeline Model	5
FR10	Predict with AL Pipeline	2
FR11	Evaluate AL Pipeline	5
FR12	Transparency	3
FR13	Export of Evaluation Results	5
NFR01	Simple User Interface	4
NFR02	Maintainability of Software Architecture	3
NFR03	Extensibility of Software Architecture	3
NFR04	Data Exchange	3
NFR05	User Spark MLlib as ML Framework	5
NFR06	Simulation of AL	5

Table 10: Overview of LexML's Requirements and their Priority

This concludes the analysis and specification of the requirements for the AL-microservice LexML, and some of the implications for the existing legal data science environment Lexia. The next step is to develop a suitable architecture that can handle these requirements.

4.3Architecture

In this section, the architecture of LexML is developed and presented from different perspectives. First, a conceptual overview of the appearance of the two systems is presented. Then the components and the data-model of LexML are described in greater detail. Finally, a typical AL workflow is presented.

4.3.1 Conceptual Overview

Figure 24 provides a high-level overview of Lexia and LexML and how they interact.

On the left, the main components of Lexia are shown in a manner similar to Figure 21. Amongst other features, Lexia has existing importers for legal data, an efficient database and a data-access layer, a data- and text-mining engine based on the UIMA framework, a UI, and a Rest API. All of these components are re-used and complemented, considering the following: (1) the *UI* becomes an additional ML interface, where the user can execute all features specified in the FRs and perform AL while fulfilling NFR01. (2) The *export* functionalities are supplemented by an additional xlsx exporter which exports the evaluation results (see FR13). (3) The existing *Rest API* is extended to communicate with LexML. (4) Minor adaptions are made to the *Data Access Layer* in order to transfer the required textual data (documents and sentences) to LexML (see FR03).

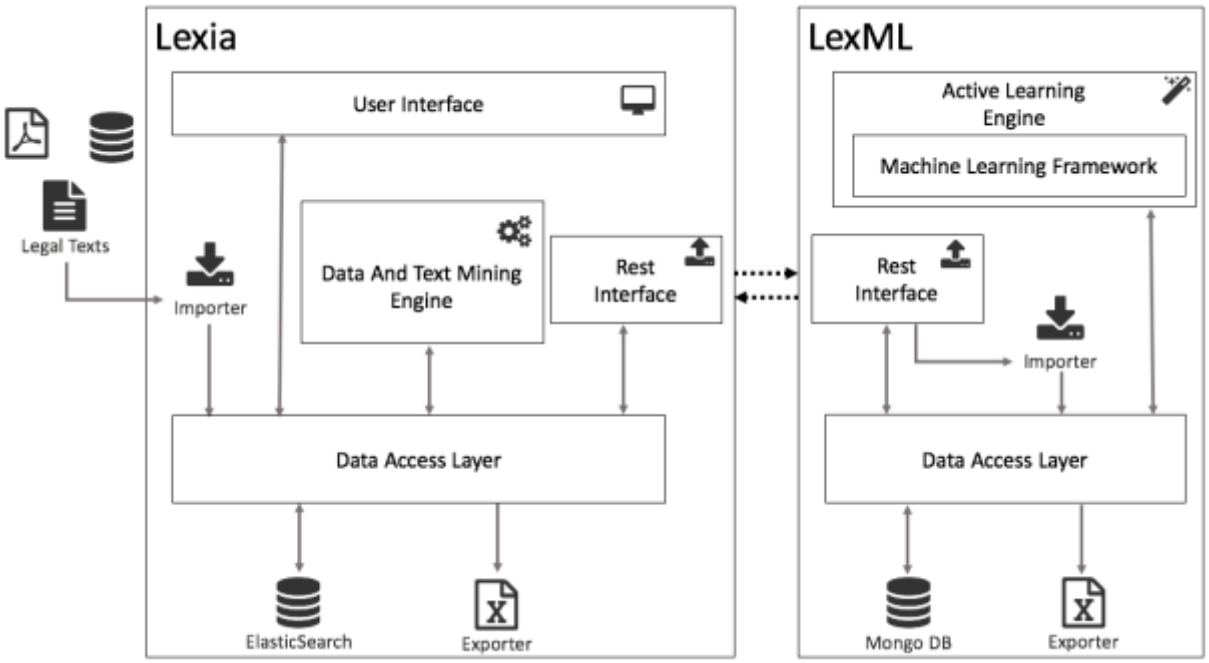


Figure 24: Conceptual Overview of Lexia and LexML
Source: Own illustration

On the right, the conceptual architecture of LexML is demonstrated. To interact with Lexia, LexML must also have a *Rest API*. The central component is the Active Learning Engine (ALE) that contains the logic of AL. To perform the actual ML in LexML's ALE, Spark MLlib is used as *Machine Learning Framework*. Further, a *Data Access Layer* provides access to *Mongo DB* in which the AL pipelines, including the legal textual data, are stored. The legal textual data is transformed into a suitable format for ML with Spark by means of a specific *importer*.

As a constant communication with Lexia is not necessary during simulation (NFR06), LexML should have its own *exporter* (see FR13) to save bandwidth and time.

4.3.2 Modular Component Overview

Following this impression of the general concept, some of the aforementioned components and the technologies used are described more detailed.

Rest API

A detailed overview about the Rest API of Lexia and LexML is given in Chapter 4.3.4.

Importer

To fulfil FR02, LexML must have an importer that is able to receive the legal textual data as a JSON request from Lexia and, create the training and test data that is suitable for TC with Spark (using the Spark SQL *Row* format). To retrieve the document on Lexia side again at a later point of time, not only the text itself but also the identifier (id) of the Lexia annotation must be part of the request. Furthermore, in cases where a test set for the evaluation of the model should be created during the import, the JSON request must also contain the type (class) of the legal text (e.g. law or judgement in the context of DC).

Active Learning Engine

The *ALE*, conceptually illustrated in Figure 25, is the central component of LexML using Apache Spark's MLlib as ML framework. The components framed in grey highlight the subcomponents of the MLlib framework that are used in the AL process in particular.

The ALE is based on the typical pipeline architecture of TC processes and, has a type of pipeline architecture consisting of the three subcomponents, similar to that of Lexia:

The *configuration* subcomponent contains most of the logic for creating (FR06), configuring (FR07) and editing AL pipelines, such as specifying the query strategy, creating the test set and training set, defining the stopping criteria, and selecting the classifier. Possible classifiers are part of the MLlib framework and should be applicable together with the ML Pipeline concept, as well as suitable for TC (see Chapter 2.1.2.6). NB, LR and MLP fulfil this requirement, in contrast to SVMs which cannot yet be used as pipeline stage of an ML Pipeline. All mandatory configuration tasks of this component must be finished in order to commence the learning process.

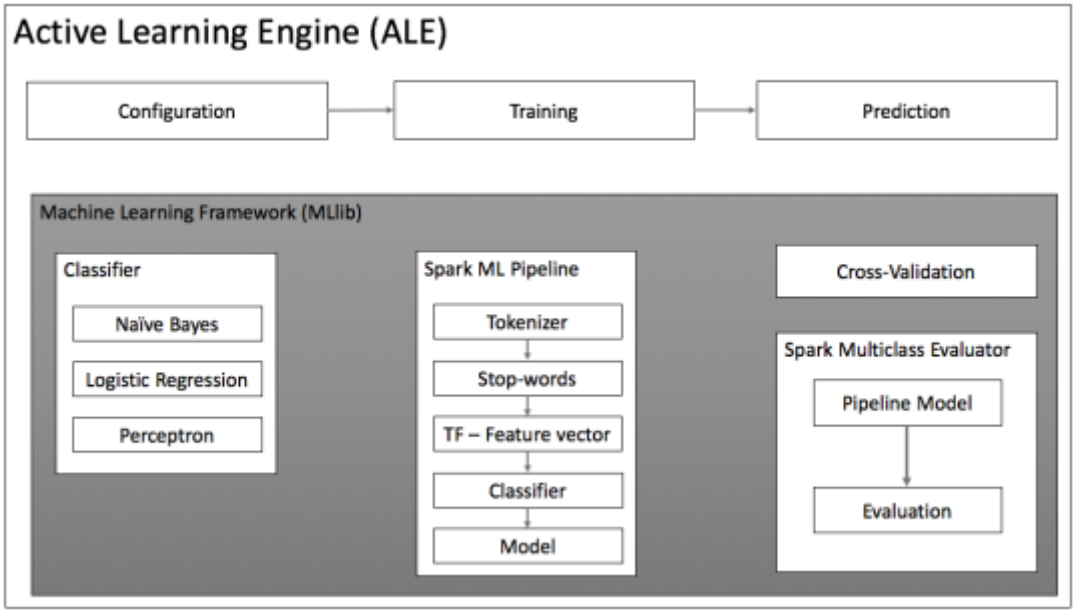


Figure 25: Detailed Conceptual Overview of LexML's AL Engine
Source: Own illustration

As discussed, preparing the datasets is a prerequisite during the configuration of an AL pipeline. Hence, although the importer subcomponent is depicted outside of the ALE in Figure 24 to allow better intelligibility, it may also be considered as part of the configuration subcomponent, and thus of the ALE component.

In the *training* subcomponent, the actual learning process and the TC pipeline are implemented (FR01). There, the model is trained with the labeled training data set (FR08) and applied to the remaining unlabeled training data to predict their label and to find the most informative instances for the classifier based on a selected query strategy. These instances are then labeled in the following round. Cross-validation ensures that these predictions are made with the best model found. At the end of each round and at the end of the learning process, the resulting pipeline model is persisted so that it can be used by the prediction component (FR09).

The *prediction* component requires a persisted pipeline model either to predict the labels of the rest of the training data (FR10), or to evaluate the model against the test set (FR11) created at import time. For evaluation purposes, the multiclass evaluator of Spark is used.

For both training and evaluation, Spark's ML Pipeline concept (FR05) is employed in order that several TC steps can be concatenated and executed at once.

4.3.3 Workflow Overview

The previous chapters gave an overview of the architecture of LexML and Lexia and illustrated how these two services interact at a higher-level. This chapter presents a more profound insight into this interaction and demonstrates a typical AL process. The results of this workflow are additional valuable information for the design of the required methods of the Rest API presented in Chapter 4.3.4.

To perform interactive AL, three actors are required: (1) the user utilizing the UI and the imported texts of the (2) legal data-science environment Lexia, and (3) the ML service LexML. Figure 26 depicts the interaction between these three actors; the arrows illustrate the communication messages and their direction. As described in Chapter 4.3.2, the AL process consists of the following three main stages or components: (1) the configuration of the AL pipeline (framed with blue), (2) the labeling and training (learning) of the classifier (bordered with red), and (3) the evaluation of the classifier (surrounded with green).

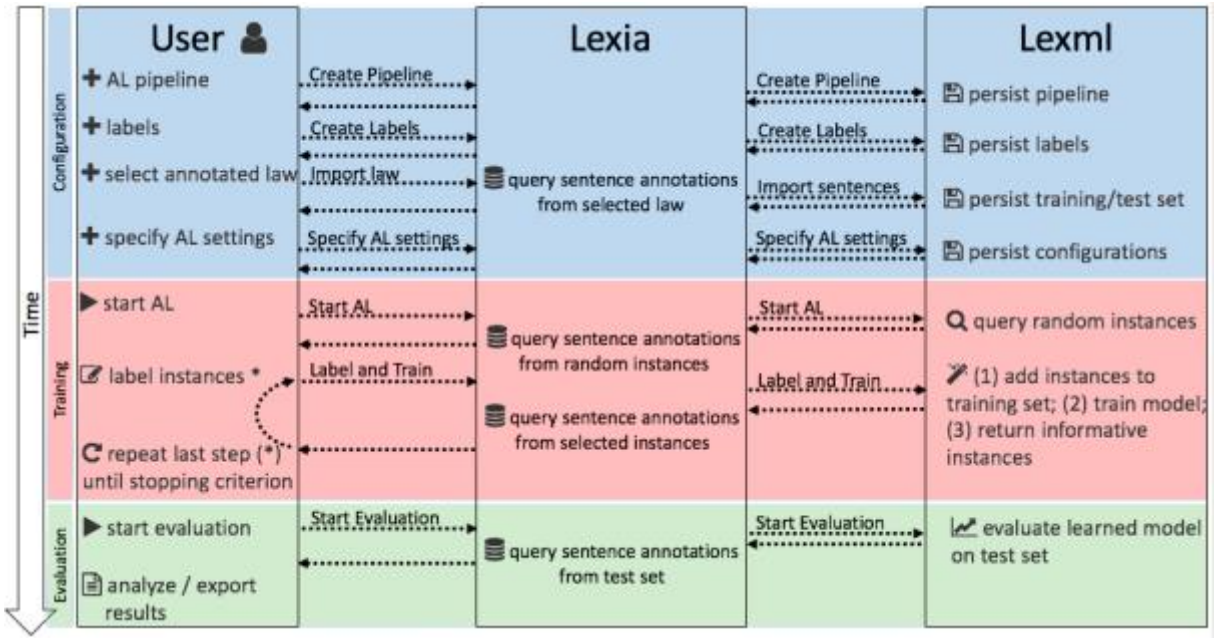


Figure 26: Conceptual Workflow Model of Norm Classification with LexML
Source: Own illustration

In what follows, a detailed description of the three workflow stages exemplified for NC is given, starting at the upper left corner of Figure 26.

Configuration Stage

The configuration consists primarily of the following four process steps:

1) In order to perform AL with legal texts, an AL pipeline must be created in a first step by stating its name and specifying the pipeline type (DC or NC). The request is forwarded to LexML via Lexia.

2) Then, all possible labels that the norms of the law that will be imported can have, must be created. A selection of possible labels for NC is given in Chapter 2.3.2. These labels

are assigned to the created AL pipeline and persisted. In this manner, a mapping is created to map the label name to a Spark specific value.

3) The next step is to select a law that is (1) already imported in Lexia and (2) has already been annotated by the sentence splitter. In cases where only specific linguistic types (e.g. verbs) should be imported, one can specify this here, too. This requires the prior execution of additional ruta scripts. Having selected the law, Lexia queries the relevant annotations (e.g. sentence or verb) from its database, and sends a suitable request containing the texts to import, their identifier and if it exists, the sentence type to create a test set. On the LexML side, this data is added to the unlabeled data set of the training data and when labeled instances are available, a test set is created.

4) In the last configuration step, specific AL settings, such as the classifier, the query strategy, the size of the seed- and query set, and the stopping criterion must be configured. These configurations are also assigned to the AL pipeline and persisted in LexML.

Following this, the actual interactive and iterative AL process begins.

Training Stage

1) After the user has started the learning process for the configured AL pipeline, random instances are queried and removed from the unlabeled training set at LexML. As the Lexia id of the imported sentences is persisted in LexML as well, more information about the origin of this sentence can be queried in Lexia and presented to the user in a proper manner.

2) The user labels these presented instances with one of the defined labels. This information is then sent back to LexML via Lexia. These examples are added to the labeled training set which is then used to train the pipeline model with the selected classifier. This model is used to make predictions about the unlabeled training set. By applying the selected query strategy to the predictions, the instances that are especially informative for the classifier are selected to be labeled next and sent back to Lexia. There, the chosen instances are again enriched with information and presented to the user.

3) The interactive process described in step 2 is repeated until a defined stopping criterion is met.

At the end of this training process, the resulting model is persisted so that it can be used by the prediction stage. In cases where the *learning* of the pipeline should be evaluated, the model is persisted after each learning round so that the prediction pipeline stage can also be executed each round.

Evaluation and Prediction Stage

This stage has two possible execution options: (1) evaluation, and (2) prediction. As previously stated, the evaluation of the classifier is prioritized over pure prediction using this classifier. Hence, the evaluation process is described as follows:

1) In this step, the model received from the training stage is applied on the test set created in the configuration stage. The predictions are evaluated with a multiclass evaluator of Spark to obtain the stated evaluation metrics.

2) These evaluation metrics together with the respective prediction for each sentence are sent back to Lexia and enriched with further information. Then, these common evaluation measures (e.g. total *accuracy*, *precision* & *recall*) along with a comparison of the prediction and correct label for each sentence are presented to the user.

The AL process having been illustrated in detail, LexML's and Lexia's necessary communication interface can be designed.

4.3.4 Rest API

The Representational State Transfer (REST) API is a software architectural style that allows distributed systems to communicate with each other in a stateless manner. It is the state-of-the-art technology for web applications. Essentially, the transfer message consists of three fundamental components: a constrained http-method (e.g. Get, Post), a resource identifier (URI), and some constrained content type (e.g. JSON, XML) [170].

In what follows, Lexia's and LexML's communication APIs, necessary to allow ML as defined in the previous chapters, are described in Table 11 and Table 12. The architecture and the intention of each request are described, and the request parameters and the JSON nodes that must be part of the request body are mentioned. The description of the JSON nodes does not reflect the actual correct JSON architecture of this request, but provides a useful indication what the actual request does and what it requires.

The requirements defined in Chapter 4.2 and the workflow described above in Chapter 4.3.3 Workflow Overview are reflected in the architecture of both APIs. They are composed of several requests to create and configure an AL pipeline, to manage the labels, to import the textual data, to conduct training and (simulated) evaluation. As it is the id of an AL pipeline, the pipeline name must always be part of the request.

Except for the import, the APIs are fundamentally very similar and often a user request is only forwarded from Lexia to LexML. For the import, Lexia has some additional APIs to guarantee the flexible retrieval of legal documents and sentences persisted in Lexia's Elasticsearch database. These documents are then transmitted to LexML, which has only one standardized API for the import of textual data. Additionally, Lexia has an interface to export the evaluation results after learning.

HTTP Method	URI	Intention	Mandatory content of request body or request parameter
GET	/pipeline/getAll	Get all created AL pipelines	/
GET	/pipeline/get	Get specific AL pipeline by name	pipeline name
# Pipeline and Settings			
POST	/pipeline/create	Creation of a new AL pipeline	pipeline name, pipeline type
GET	/pipeline/remove	Removal of AL pipeline	pipeline name
POST	/pipeline/settings/save	Save AL pipeline settings	classifier, query strategy, stopping criteria, seed set, query size
POST	/pipeline/trainingsdata/clear	Reset the training set	pipeline name
# Label			
POST	/label/create	Creation of one label	pipeline name, label name
POST	/label/remove	Removal of label	pipeline name, label name
POST	/label/save	Creation of label mapping	pipeline name
POST	/labels/create	Creation of several labels	pipeline name, node with labels containing label name
# Import			
POST	/pipeline/import/legalDocuments	Import of documents or sentences	pipeline name, node with legal texts containing their Lexia id, actual text that is used for learning and eventually their labels
POST	/pipeline/import/labelled/ legalDocuments	Import of documents or sentence of which some should be added to the labeled training set	pipeline name, node with legal texts containing their Lexia id, actual text that is used for learning and their labels
POST	/pipeline/import/clear	Removal of imported data	pipeline name
# Training			
POST	/pipeline/startLabeling	Start labeling and query random instances	pipeline name, boolean indicating whether learning should be evaluated
POST	/pipeline/label	Label queried instances	pipeline name, Node with ids and selected labels of the instances
GET	/pipeline/train	Train the classifier	pipeline name
GET	/pipeline/getInstancesToLabel	Get instances that are most informative based on a certain query strategy	pipeline name
# Evaluation			
POST	/pipeline/evaluate	Evaluate the model based on the test set	pipeline name
# Prediction			
POST	/pipeline/predict	Predict the label of the unlabeled data set	pipeline name
# Simulation			
POST	/pipeline/simulate	Simulate training and evaluation	pipeline name, boolean indicating whether learning should be random or based on a defined query strategy

Table 11: LexML's REST API

HTTP Method	URI	Intention	Mandatory content of request body or request parameter
# Pipeline and Settings			
forward to Lexia			
# Label			
forward to Lexia			
# Import			
POST	/pipeline/laws	Get all laws that have been processed by the sentence splitter	/
POST	/pipeline/import/legalSentences	Import of Sentences	selected law, selected annotations (e.g. verb)
POST	/pipeline/import/csvSentences	Import of Lexia sentences mapped to a label obtained from a csv file (sentence evaluation)	selected annotations
POST	/pipeline/import/legalDocuments	Import of documents	number of documents that should be imported, selected annotations, selected feature (e.g. title, text, date)
POST	/pipeline/import/clear	Removal of the imported data	pipeline name
# Training			
forward to Lexia			
# Evaluation			
forward to Lexia			
# Prediction			
forward to Lexia			
# Simulation			
forward to Lexia			
# Export			
POST	/pipeline/export	Export of evaluation results	pipeline name, evaluation measures, labels, percentage of labeled instances

Table 12: Lexia's REST API

4.3.5 Data Model

With this presentation of the requirements and the workflow having been concluded, the next step is to create a suitable data model for LexML. An overview of LexML's data model is depicted in Figure 27. It does not illustrate all attributes and methods, but gives a useful indication of how the software is built.

The central component is the class *ALPipeline*, which reflects one specific AL process for either DC or NC (ActiveClassificationType). It derives from *PersistentEntiy* where the functionality for saving and deleting an entity in the Mongo DB is implemented.

The *ALPipeline* has an embedded list of at least three class *labels* (multiclass) created by the user. These labels are mapped to a specific number that is used by Spark for learning. The class *ALPipelineConfigurations* contains all possible configuration options that are specific to AL, such as the size of the *seed set*, the number of instances that should be queried in each round (*query size*), and whether certain annotations (e.g. verb, adverb) have been selected on the Lexia side at the import (see FR07).

Furthermore, the *ALPipelineConfigurations* class has an *IClassifier* attribute representing either a probabilistic classifier (*Logistic Regression*, *Naïve Bayes*) or a *MultiLayerPerceptronClassifier*.

Additionally, the *ALPipelineConfigurations* class contains a mapping to the *IStoppingCriterion* interface to implement the strategies for stopping only after a minimum number of rounds (*MinRound*), but at the latest after a certain number of rounds (*MaxRound*).

To register the selected query strategy, the *ALPipelineConfigurations* class has an *IQueryStrategy* attribute. Both US methods *MarginSampling* and Entropy (*AbstractEntropy*) as well as two QBC (*AbstractQueryByCommittee*) strategies implement this interface. The *AbstractQueryByCommittee* is further abstracted to allow the simple adding of additional QBC strategies that are not based on vote entropy (*AbstractVoteEntropy*). There is an extra class for the processing of MLPs as this classifier does not provide a posterior probability.

The use of an interface for the classifier, stopping criterion and query strategy ensures that the maintainability (NFR02) and especially the extensibility (NFR03) of the ML service is given.

To assign the training and test data to an *ALPipeline*, each instance has a relation to *TrainingData* and *TestData*. The *TrainingData* class has the following three attributes:

- *labelledData* – part of the imported textual data that is labeled and used for training,
- *unlabelledData* – part of the imported textual data that is not yet labeled but used for prediction to find the most informative ("helpful") ones,
- *dataToLabelNext* – part of the imported data that is based on the query strategy particularly informative and sent to the user in order to be labeled.

The *TestData* class contains an independent labeled *testData* set for evaluation as well as the labeled *labelledTrainingsData* that is necessary to simulate AL (see NFR06).

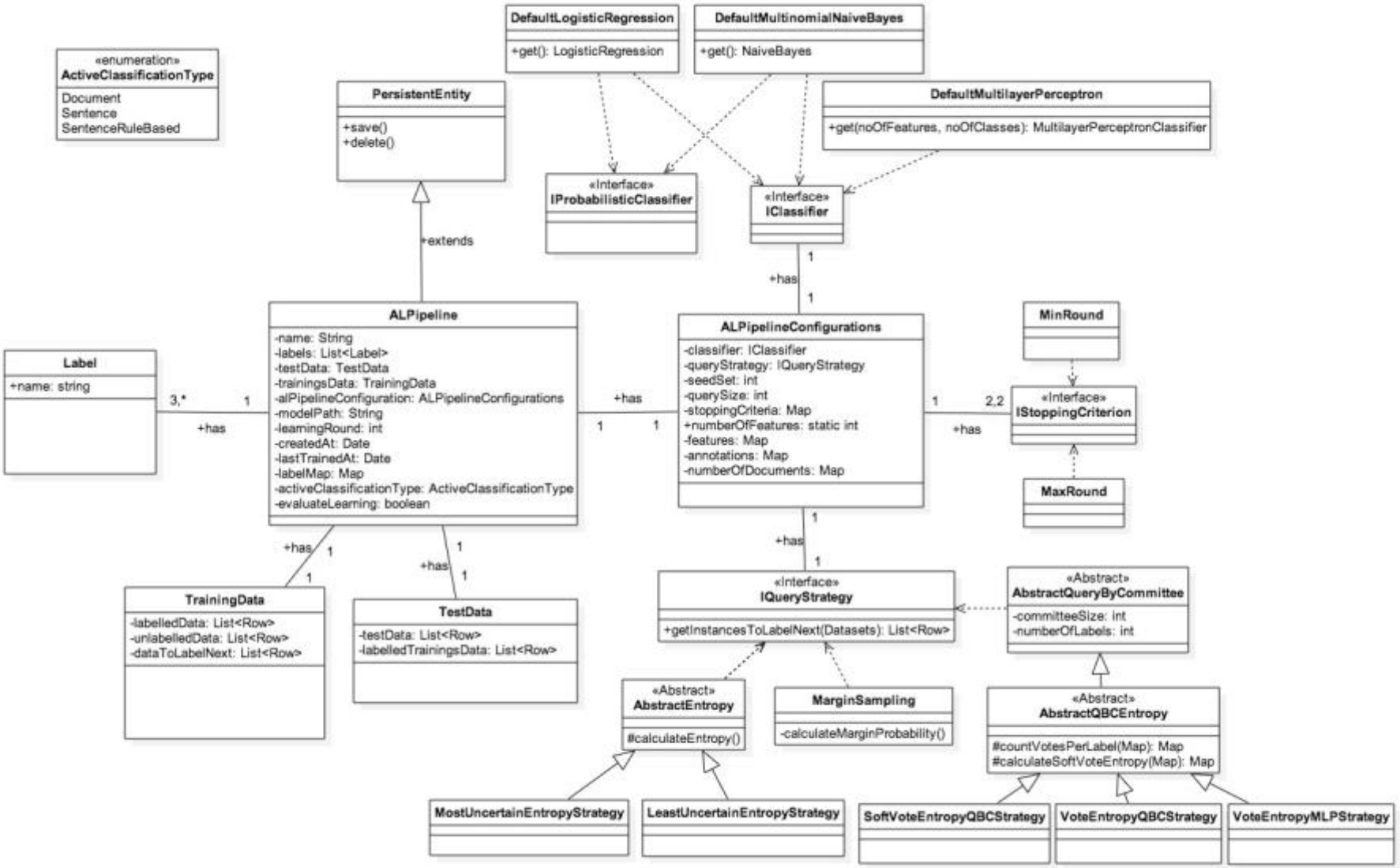


Figure 27: Overview of LexML's Data Model
Source: Own illustration

5 Implementation

The previous chapter discussed LexML's objectives and requirements, the effects on Lexia, and presented the development of a suitable strategy; the next step is to implement it. How this implementation is realized is described in this chapter.

5.1 Basics

Like Lexia, LexML is developed with the web application framework Play[50], which has a Java back-end. To perform ML, the Spark MLlib framework introduced in Chapter 3.1.1 is used.

The database used to persist the AL pipeline, its configurations and the textual data is MongoDB version 3.2.10[51]. As a schema-free document database with high scalability and flexibility, it is eminently suitable for this use case.

In order to map Java objects to the MongoDB documents and back, use is made of the Morphia framework built on the MongoDB Java Driver[52]. It is simple to use and as the configurations can be made with annotations, the readability of the code is better. Furthermore, it is type safe and incorporates an intuitive query interface. Morphia's basic functionality is depicted in Listing 1 using the class *ALPipeline* as reference, the central component in LexML's data model (see Figure 27). References to other classes are made with the *@Embedded* annotation.

```
1.  @Entity("pipelines")
2.  @Indexes(@Index(value = "name", fields = @Field("name")))
3.  public class ALPipeline extends PersistentEntity {
4.
5.      @Id
6.      private ObjectId _id;
7.
8.      @Indexed
9.      private String name;
10.
11.     @Embedded
12.     List<Label> labels;
13.
14.     @Embedded
15.     TestData testData;
16.
17.     @Embedded
18.     ALPipelineConfigurations alPipelineConfigurations;
19.
20.     @Embedded
21.     TrainingData trainingsData;
22.       //further code
23. }
```

Listing 1: Functionality of Morphia

[50] https://www.playframework.com/ (last accessed: May, 11th 2017)
[51] https://www.mongodb.com/blog/post/mongodb-3210-is-released (last accessed: May, 8th 2017)
[52] http://mongodb.github.io/morphia/ (last accessed: May, 8th 2017)

5.2 Active Learning Engine

An outline of the implementation of the main components of LexML's ALE, introduced in Chapter 4.3.2, is provided in below. The implementation of the logic is similarly arranged as shown in this chapter and as described in the workflow model (Figure 26). The ALE is compromised by the four classes *Importer*, *ConfigurationPipeline*, *TrainingPipeline* and *PredictionPipeline.* A more detailed insight to the implementation of the classifiers and query strategies is provided in Chapter 5.3 and 5.4.

Importer

The task of the *Importer* class is to transform the JSON-request containing the legal texts into, respectively, training and test data of an AL Pipeline which has the Spark SQL *Row* format that is needed for ML with MLlib. A *Row* is a generic object with an ordered collection of fields. The schema of a Row for attributes of the class *TestData* and for the attribute *labelledTrainingData* of the class *TrainingData* consists of the following three fields: (1) a document or a sentence's annotation id in Lexia (String), (2) the label of this annotation (double), and (3) the actual text of the annotation that is used for classification (String). A Row of the *TrainingData's unlabelledData* and *dataToLabelNext* attributes is built in a similar manner, except that it does not have the *label* field.

Listing 2 shows the code for the creation of the *Rows* for the attributes of the classes *TestData* and *TrainingData*. Most of the textual data is added to the *unlabelledData* attribute from which the instances are queried during training, to be labeled by the user. Additionally, if desired, with a certain percentage of the dataset (depending on size of the import), a labeled test set is created. This test set is used for evaluation. In cases where no class for an instance is provided, a default label "-1.0" is created. Additionally, TestData's *labelledTrainingsData* attribute is filled with the labeled training data. This is effected to create a labeled dataset that is not used for learning but to find the correct label of an instance to make the simulation of AL possible.

```
1.  if(testSet){
2.                  if(label != -1){
3.                      pipeline.getTestData().getTestData().add(
4.                         RowFactory.create(id, label.doubleValue() , text));
5.                  } else {
6.                      pipeline.getTestData().getLabelledTrainingsData().add(
7.                         RowFactory.create(id, Double.valueOf("-1.0"), text));
8.                  }
9.  } else {
10.                 pipeline.getTrainingsData().getUnlabelledData().add(
11.                        RowFactory.create(id, text));
12.                 if(label != -1){
13.                     pipeline.getTestData().getLabelledTrainingsData().add(
14.                        RowFactory.create(id, label.doubleValue() , text));
15.                 } else {
16.                     pipeline.getTestData().getLabelledTrainingsData().add(
17.                        RowFactory.create(id, Double.valueOf("-1.0"), text));
18.                 }
19. }
```

Listing 2: Schema for a Row for Test and Labeled Training Data

Configuration

The *ConfigurationPipeline* class contains most of the methods for the creation, configuration, and editing of the AL pipeline, and affects especially the *ALPipelineConfigurations* class. While the presence of most of the methods is very self-explanatory, the intention of the *createDummyEntriesInLabelledDataSet()* method, depicted in Listing 3, should be described.

```
1.  public void createDummyEntriesInLabelledDataSet(ALPipeline pipeline){
2.          Integer i = 0;
3.          for(Map.Entry<Integer,String> entry: pipeline.getLabelMap().entrySet()){
4.            pipeline.getTrainingsData().getLabelledData().add(
5.            RowFactory.create(String.valueOf(i),(double) entry.getKey(),dummyWord));
6.              i++;
7.          }
8.          pipeline.save();
9.      }
```

Listing 3: Configuration of Dummy Entries

The *raison d'être* for this method is based on the characteristic of AL that involves the fact that often in the first learning rounds, not all defined labels are part of the labeled training set. This means that the classifier does not know that these labels exist. During classifier training and prediction, MLlib normalizes the label values starting from 0.0. The result is that the mapping from the label name to its value (persisted in the attribute *labelMap*) is no longer correct. To solve this problem, for each defined label, a *Row* instance in the labeled training dataset is created containing the same dummy textual content. This overcomes the problem that involves the fact that the classifier does not know all the possible labels from the start. Furthermore, as the same word is used for all labels, the classifier is not distorted by it.

Training

The actual logic of the AL process is implemented in the *TrainingPipeline* class. It possesses all the methods that empower the execution of the learning process described in the red-framed section of the workflow model (Figure 26). Its attributes *trainingData* and *unlabelledData* are the pendants to *TrainingData*'s *labelledData* and *unlabelledData* attributes transformed into a *DataFrame* (*Dataset* of *Rows)*. Such *DataFrames* can then be used for conducting ML with Spark. The *model* attribute having the type *PipelineModel* is the resulting model that is persisted (*saveModel()*) after training and used for prediction and evaluation.

To perform TC with it, a Spark ML pipeline must be prepared accordingly. This is effected by means of the *preparePipeline()* method in which the stages (tokenizer, stop words, feature vector, classifier) of the pipeline are set (see Listing 4). Additionally, the aforementioned *DataFrame* objects for training are created. Hence, this method must be called for each learning round as the training data requires updating.

```
1.  public void setStages(ALPipeline pipeline) {
2.          Tokenizer tokenizer = DefaultTokenizer.get();
3.          StopWordsRemover stopWordsRemover = new StopWordsRemover()
4.                .setInputCol(tokenizer.getOutputCol())
5.                .setOutputCol("filteredWords")
6.                .setStopWords(StopWordsRemover.loadDefaultStopWords("german"));
7.          hashingTF = DefaultHashingTF.get(stopWordsRemover.getOutputCol(),
8.                numberOfFeatures);
9.          PipelineStage classifier = ClassifierFactory.getClassifier(pipeline);
10.         SparkSingleton.getInstance().getPipelineSpark()
11.               .setStages(new PipelineStage[]{tokenizer, stopWordsRemover,
12.               hashingTF, classifier});
13. }
```

Listing 4: Set Stages of a Spark ML Pipeline

At the beginning of the workflow, random instances must be queried from the unlabeled training set. This is implemented by the method *getRandomInstances()* in which a number of instances (e.g. typically the size of the seed set) are added to the *dataToLabelNext* attribute of the *TrainingsData* class and removed from its *unlabelledDataSet* attribute.

The labeling is handled in the *label()* method which has a Key-Value-Map as the parameter containing the id and the label created by the user. The previously filled *dataToLabelNext* attribute is used to quickly find these instances and add them labeled to the *labelledDataSet*.

Before the actual processing/training is started, the *stopLearning()* method checks whether a stopping criterion applies. If this is the case, the learning process stops. Otherwise, the *prepareDataToLabelNext()* method depicted in Listing 5 is called. Through the method call on *makePredictionsOnUnlabelledData()*, the labeled training data is used to create a list of predictions (*DataFrames*). With this method, the training data is used to train the ML pipeline. For probabilistic classifiers, Spark offers cross-validation to tune the ML pipeline [53]. This concept implemented as five-fold cross validation in the *getCrossValidatorModel()* method ensures that the predictions for the unlabeled data made with the ML pipeline are based on the best model found. Depending on the classifier and query strategy chosen, either one prediction *DataFrame* (for US strategies) or three prediction *DataFrames* (for QBC strategies) are created. If the unlabeled dataset contains more than 400 legal documents, only 70% of randomly selected instances are used to improve efficiency. Returning to the *prepareDataToLabelNext()* method, the selected query strategy is applied to the obtained predictions to calculate an uncertainty measure. The resulting list is sorted in ascending order based on the defined output measure of the respective query strategy and added to the *dataToLabelNext* attribute. For the perceptron, the predictions do not have any posterior probability. Hence, the field on which sorting is performed (illustrated in Listing 5, lines 17-26) varies. Generally, as there are some further minor differences between the both probabilistic classifiers and MLP, there are sometimes minor specialized implementations for MLPs (e.g. *getModelForMLP()*).

[53] http://spark.apache.org/docs/latest/ml-tuning.html (last accessed: May 12th, 2017)

```
1.  public boolean prepareDataToLabelNext(ALPipeline pipeline){
2.
3.          //make predictions on unlabelled data set with the best model found
4.          List<Dataset<Row>> predictionsList = makePredictionsOnUnlabelledData(
5.            pipeline);
6.
7.          IQueryStrategy queryStrategy = pipeline.
8.                getAlPipelineConfigurations().getQueryStrategy();
9.          List<Row> uncertainRows = queryStrategy.getInstancesToLabelNext(
10.             predictionsList);
11.
12.         if(uncertainRows.size() == 0) {
13.             return false;
14.         }
15.
16.             //sort ascending
17.         if (pipeline.getAlPipelineConfigurations().getClassifier()
18.            instanceof IProbabilisticClassifier){
19.               uncertainRows.sort(Comparator.comparingDouble(
20.                   o -> ( double) o.get(3)));
21.
22.         } else if(pipeline.getAlPipelineConfigurations().getClassifier()
23.            instanceof DefaultMultilayerPerceptron){
24.             uncertainRows.sort(Comparator.comparingDouble(
25.                   o -> (double) o.get(2)));
26.         }
27.
28.         pipeline.getTrainingsData().setDataToLabelNext(uncertainRows);
29.         pipeline.save();
30.         return true;
31.     }
```

Listing 5: Implementation of PrepareDataToLabelNext Method

To obtain the most informative instances within the *getMostUncertainInstancesFromPrediction()* method, examples are selected from the *dataToLabelNext* attribute. Depending on the query strategy, these instances are either the first or the last elements of this list. A String-Array containing interesting information for the user (e.g. classifier confidence) is prepared. The number of instances that are queried is based on the configured query size.

Evaluation and Prediction

In the class *PredictionPipeline* the methods for both (1) evaluating the obtained pipeline model, and for (2) using this model to only make predictions without an evaluation, are implemented.

For the first case, the method *executeEvaluation()* is called. There, the persisted model is loaded and, in a manner similar to the *TrainingPipeline*, the test data is transformed into a *DataFrame*. The loaded model is then used to make predictions on this test data. These predictions are conveyed to the evaluator which uses Spark's multiclass evaluator *MulticlassMetrics*[54] to

[54] https://spark.apache.org/docs/2.1.0/mllib-evaluation-metrics.html (last accessed: May 13th, 2017)

compare the predictions with their actual label. The output of the specified evaluation measures (see FR11) is written to a JSON object so that it can be exported or sent back to the user.

The operation of *executePrediction()* is essentially the same, except that the evaluator is not used as no reference label exists to evaluate the predictions.

5.3 Classifier

An overview about possible TC classifiers was provided in Chapter 2.1.2.6. Additionally, in Chapter 2.2.6, these classifiers were discussed in the context of AL. Several classifiers were introduced and the three classifiers NB, SVM and MLP were described in detail. As the use of Spark's ML Pipeline is required, only NB, MLP, and additionally the LR were implemented. In the current MLlib version, SVMs cannot be used as pipeline stage of an ML Pipeline. However, as described in the data model (see Figure 27), the implementation was realized in a manner that allows additional classifiers to be added without difficulty. This flexibility is created through the use of the Interface *IClassifier* that must be implemented by all classifiers.

Naïve Bayes

The implementation of Spark's NBs classifier is effected in the class *DefaultMultinomialNaiveBayes* (see Listing 6). As a probabilistic classifier, it additionally implements the *IProbabilisticClassifier* interface. As mentioned above, there are differences between these probabilistic classifiers (generative) and classifiers like MLP (discriminative). Hence, the use of the interface is a further means to ensure readable code.

```
1.  public class DefaultMultinomialNaiveBayes implements IClassifier, IProbabilisticClas
    sifier {
2.
3.      public static NaiveBayes get() {
4.          return new NaiveBayes().setFeaturesCol("features");
5.      }
6.
7.  }
```

Listing 6: Implementation of Naïve Bayes

For the implementation itself, Spark's default implementation of the Naïve Bayes is used[55]. The defined feature input column is the output column of the feature vector column named *features*.

Logistic Regression

As a probabilistic classifier, LR also implements the respective *IProbabilisticClassifier* interface (see Listing 7).

```
1.  public class DefaultLogisticRegression implements IClassifier, IProbabilisticClassif
    ier {
2.
3.      public static LogisticRegression get() {
4.          return new LogisticRegression().setMaxIter(10)
5.                      .setElasticNetParam(0.8).setRegParam(0.001);
6.      }
7.  }
```

Listing 7: Implementation of Logistic Regression

[55] https://spark.apache.org/docs/2.1.0/ml-classification-regression.html#naive-bayes (last accessed: May 13th, 2017)

Spark provides a multiclass implementation of LR[56] using multinomial logistics. The conditional probabilities are modeled using the softmax function while a weighted negative log-likelihood having an elastic-net penalty is applied to control for overfitting.

LexML's default implementation of LR reduces the number of default iterations from 100 to 10 to increase the efficiency of the training process. The *elasticNetParam* is set to 0.8, which leads to a combination of the L1 and L2 penalties, favoring the L1 penalty. The L1 penalty should be especially helpful in cases of a high feature-dimensionality [171]. The regularization parameter is set to 0.3.

Multilayer Perceptron

In contrast to both other classifiers, MLP is not a probabilistic classifier and does thus only implement the *IClassifier* interface (see Listing 8).

```
1.  public class DefaultMultilayerPerceptron implements IClassifier {
2.
3.      public static MultilayerPerceptronClassifier get(int noOfFeatures, int numberOfC
    lasses){
4.          noOfFeatures = ((noOfFeatures == 0) ? (int) Math.pow(2,12): noOfFeatures);
5.          int[] layers = new int[] {noOfFeatures, 20, 10, numberOfClasses};
6.          return new MultilayerPerceptronClassifier()
7.                  .setFeaturesCol("features")
8.                  .setLayers(layers)
9.                  .setBlockSize(128)
10.                 .setSeed(1234L)
11.                 .setTol(1E-6)
12.                 .setMaxIter(100);
13.     }
14. }
```

Listing 8: Implementation of Multilayer Perceptron

Spark's MLP is an implementation of the described feedforward ANNs applying the back-propagation algorithm. While nodes in the intermedia layers use the logistic function, nodes in the output layer use the softmax function.

LexML's default implementation of the MLP has four layers: the size of the input layer is specified in the *ALPipelineConfigurations* class by the *numberOfFeatures* attribute. Its default value is 2^{13}. The number of nodes of the two intermediate layers is by default 20 and 10, respectively, and the size of the output layer is equivalent to the number of defined labels. For the other configurations, Spark's default settings are used.

[56] https://spark.apache.org/docs/2.1.0/ml-classification-regression.html#multinomial-logistic-regression (last accessed: May 13th, 2017)

5.4 Query strategies

As described in Chapter 2.2.5, a variety of possible query-strategy frameworks for searching for the most informative instances for the classifier exist. Each of those frameworks has different advantages and disadvantages, which were discussed in that chapter. As indicated in the illustration of the data model (see Figure 27), query strategies from both the US framework and the QBC framework are implemented in LexML. All strategies have in common that they implement the interface *IQueryStrategy* to improve the flexibility, extensibility and readability of the code. In this interface, the method *getInstancesToLabelNext()* that must thus be implemented by all query strategy classes is defined. The purpose of this method is to apply a query strategy in which a measure (the informativeness measure) is calculated that should express the utility of each instance.

Uncertainty Sampling

Due to its simple implementation and its efficiency benefits compared to QBC strategies, US is in practice a very common AL query strategy. Two query strategies for this approach introduced in Chapter 2.2.5 are implemented in LexML. As a discriminative classifier, the predictions of MLP do not have any probability that can be used as an informativeness measure to calculate a score. Hence, the two US strategies detailed below can only be used with classes implementing the *IProbabilisticClassifier* interface. Classes implementing this interface receive, via the *getInstancesToLabelNext()* method, a list containing one DataFrame with predictions. Each instance of this DataFrame has several columns (fields) like the id, the text that was used for learning, the predicted label, and a probability vector in which the probability for each defined label is indicated. This vector is used for these learning strategies. The highest probability in this vector reflects the actual predicted label for this instance.

Uncertainty Sampling – Margin Sampling Strategy

The *margin sampling* strategy is implemented in LexML as depicted in Listing 9. The basic concept is that the classifier has difficulty in differentiating between instances for which the margin between the highest and second highest probability in the vector is exceedingly small.

```
1.  private double calculateMarginProbability(Vector probabilities){
2.
3.          // get highest probability
4.          int maxIndex = probabilities.argmax();
5.          double max1 = probabilities.apply(maxIndex);
6.
7.          // get second highest probability
8.          double[] probabilitiesArray = probabilities.toArray();
9.          probabilitiesArray[maxIndex] = 0.0;
10.
11.         int maxIndex2 = probabilities.argmax();
12.         double max2 = probabilities.apply(maxIndex2);
13.
14.         return max1 - max2;
15.
16.     }
```

Listing 9: Implementation of Margin Sampling

Hence, the margin is calculated using the two highest probabilities in the probability vector. The difference between these probabilities is then used as a informativeness measure in this query strategy. The instances having the smallest distance are the most helpful ones.

Uncertainty Sampling – Entropy Strategy

In contrast to the margin sampling, the *entropy strategy* has the advantage that all probabilities are used to calculate the informativeness measure, not only the two highest ones. This more general approach is illustrated in Listing 10, in which the Shannon entropy is calculated.

```
1.  protected static double calculateEntropy(double[] probabilities) {
2.
3.          double entropy = 0;
4.          for (int i = 0; I < probabilities.length; i++) {
5.              entropy -= (probabilities[i] > 1e-7)
6.                      ? probabilities[i] * (Math.log(probabilities[i]))
7.                      : 0;
8.          }
9.
10.         return entropy;
11.     }
```

Listing 10: Implementation of Entropy Strategy

To improve efficiency, only probabilities above a certain threshold are included in this calculation. The entropy is then calculated by iterating across the probabilities and adding the product of each probability to the logarithm of this probability. The instances with the highest resulting sum have the largest information content. Hence, knowing their label would best assist the classifier in differentiating between the classes. The instances having the highest entropy are selected to be labeled in a next step.

Query by Committee (QBC)

Compared to US methods, *QBC strategies* do not use only one classifier, but rather a committee of classifiers (at least two). The idea is that by using several classifiers, the version space can be better covered and the search for informative instances can be further narrowed. As this method is time-consuming, its eligibility for AL and TC with a huge version/feature space is limited. Nevertheless, as discussed in Chapter 2.2.5, it is a promising approach, and is therefore implemented in LexML. All classes that use a QBC strategy have the *AbstractQueryByCommittee* class as parent and implement the *IQueryStrategy* interface. As several classifiers are used in this strategy, the list parameter of the *getInstancesToLabelNext()* method here contains more DataFrames with predictions, depending on the size of the committee.

For the *committee creation*, no enhanced strategy like *AdaBoost* or *Query-by-Bagging/Boosting* is currently implemented for the creation of the individual classifier. Instead, each classifier utilizes a minor variation of the training set. This is enough to create a set of different classifiers.

To *measure* the *disagreements* between this created committee, the two strategies below are implemented in LexML. As both strategies refer to the vote entropy strategy, both methods are implemented in the class *AbstractQBCEntropy*.

Query by Committee – Vote Entropy Strategy

The simplest QBC strategy discussed in the theoretical part of this work was *vote entropy*, in which the fraction of the number of votes divided by committee size is multiplied by the logarithm of this fraction and accumulated across all labels.

The implementation of this algorithm in LexML is depicted in Listing 11. In contrast to both implementations of the US strategy, this QBC strategy does not have a probability vector as a parameter, but a Key-Value Map called *idVoteMap*. The *id* of this map is equivalent to the id of this instance in the training set. The *int-Array* has the same length as labels were created by the user. In this Array, each Array-index represents a certain label based on the mapping defined in the *labelMap*. For instance, assuming the *idVoteMap* is {"id_1", [2,0,0,1]}, there are four defined labels and three committees. Two committees have predicted the label 0.0 for this instance and one the label 3.0. This number can then be mapped to a label name via the *labelMap*.

```
1.  protected Map<String[],Double> countVotesPerLabel(Map<String,int[]> idVoteMap){
2.
3.
4.        Map<String[],Double> idFinalVoteMap = new HashMap<>();
5.
6.        for(Map.Entry<String,int[]> entry : idVoteMap.entrySet()){
7.            int[] votes = entry.getValue();
8.
9.            double fraction;
10.
11.           double sum = 0;
12.           int counter=0;
13.           int max = 0;
14.           String[] key = new String[2];
15.
16.           for(int vote : votes){
17.               fraction = vote / (double) getComitteeSize();
18.
19.               if(vote != 0){
20.                   fraction *= Math.log(fraction);
21.               }
22.               if(vote > max){
23.                   key[1] = String.valueOf(counter);
24.                   max = vote;
25.               }
26.
27.               sum += fraction;
28.               counter++;
29.           }
30.           key[0] = entry.getKey();
31.           idFinalVoteMap.put(key, -sum);
32.       }
33.
34.       return idFinalVoteMap;
35.   }
```

Listing 11: Implementation of Vote Entropy Strategy

The map returned consists of a String Array of the length two as key, and the calculated vote entropy as value. The first field of the key is the id received from the *idVoteMap*. The second field is the index of the label that was selected most frequently – this is displayed to the user. The algorithm itself is implemented as described above: it is iterated over each instance Array containing the votes of the classifier. The entropy is calculated by adding up all the products. The instances having the highest entropy are the most useful ones.

Query by Committee – Soft Vote Entropy Strategy

This strategy is similar to the previous one but also includes the output probability of the classifiers for the predicted labels in the algorithm. Instead of taking the absolute number of votes for a label, the average probability that an instance is labeled correctly is calculated.

```
1.  protected Map<String[],Double> calculateSoftVoteEntropy(Map<String,
2.              double [][]> idVoteMap){
3.
4.        Map<String[],Double> idFinalVoteMap = new HashMap<>();
5.        double [] averageMaximumProbabilities = new double[idVoteMap.size()];
6.
7.        int counterEntrySet = 0;
8.        for(Map.Entry<String, double[][]> entry : idVoteMap.entrySet()){
9.            double [][] probabilityArray = entry.getValue();
10.           double probabilitySum = 0;
11.
12.           // get average maximum probability of probability vector
13.           double [] averageProbabilities = new double[getNumberOfLabels()];
14.
15.           for(int i = 0; i < getNumberOfLabels(); i++){
16.
17.               for(double[] probabilities : probabilityArray){
18.                   probabilitySum += probabilities[i];
19.               }
20.
21.               double averageProbability = probabilitySum / (double)
22.                   getComitteeSize();
23.
24.               if(averageProbability != 0){
25.                   averageProbabilities[i] = -averageProbability *
26.                           Math.log(averageProbability);
27.               }
28.           }
29.
30.           double max = 0.0;
31.           int counterSelectedLabel = 0;
32.           String[] key = new String[2];
33.           for(double maxAverageProbability : averageProbabilities){
34.               if(maxAverageProbability >= max){
35.                   averageMaximumProbabilities[counterEntrySet] =
36.                           maxAverageProbability;
37.                   max = maxAverageProbability;
38.                   key[1] = String.valueOf(counterSelectedLabel);
39.               }
40.               counterSelectedLabel++;
41.           }
42.           counterEntrySet++;
43.
44.           key[0] = entry.getKey();
45.           idFinalVoteMap.put(key, max);
```

```
46.         }
47.         return idFinalVoteMap;
48.     }
```

Listing 12: Implementation of Soft Vote Entropy Strategy

The manner in which this algorithm is implemented in LexML in the *calculateSoftVoteEntropy()* method is indicated in Listing 12. As with the vote entropy strategy, the method receives an *idVoteMap* with an id as key. The value is a two-dimensional array having the same number of rows as the number of committees created. The probabilities for each predicted label are stored in the columns.

By iterating over these probability arrays and the labels, the average probability is calculated as a first step. This averaged probability is then multiplied by the logarithm of this averaged probability, as described above. The average probability of each label is then stored in the appropriate index of the *averageProbabilities* Array. In the next step, this array is scanned to find the maximum value. This value is then used as an informativeness measure – particularly high values are assumed to be more supportive. The map returned is built in the same manner as the previous strategy. Again, the instances having the highest entropy are selected to be labeled in a later step.

5.5 Frontend

To deliver an intuitive operation of AL service, Lexia was complemented with an additional ML UI, as indicated in Figure 24.

In Figure 28, the *landing page* of this additional UI, the overall starting point, is depicted. From there, the user can create a new pipeline or select an existing pipeline either to configure it or use it for training. The latter is only feasible after configuration. Additionally, the user is presented some information about the type of pipeline, the creation date, and the date of the last training.

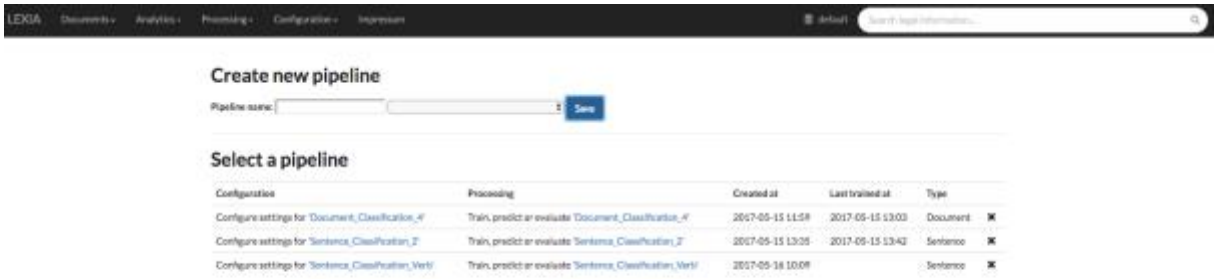


Figure 28: Lexia's ML Landing Page
Source: Own illustration

To ensure the simple configuration of an AL pipeline, it is completed in a step-by-step process on a supplementary *configuration page*. The first step in this process is to create the labels that will later be possible classification classes. The next step is to import the textual data that is used for learning. Figure 29 displays this step for NC. The user sees a list of laws for which Lexia's sentence segmenter has already created at least one sentence annotation. After selecting one or more laws, the user has the option of narrowing the import to specific word types. For instance, where the flag is set to "Verb", then only the verbs of sentences, rather than entire sentences, are imported into LexML. The button "Import BGB Mietrecht" refers to the evaluation of NC (see chapter 6.1.3).

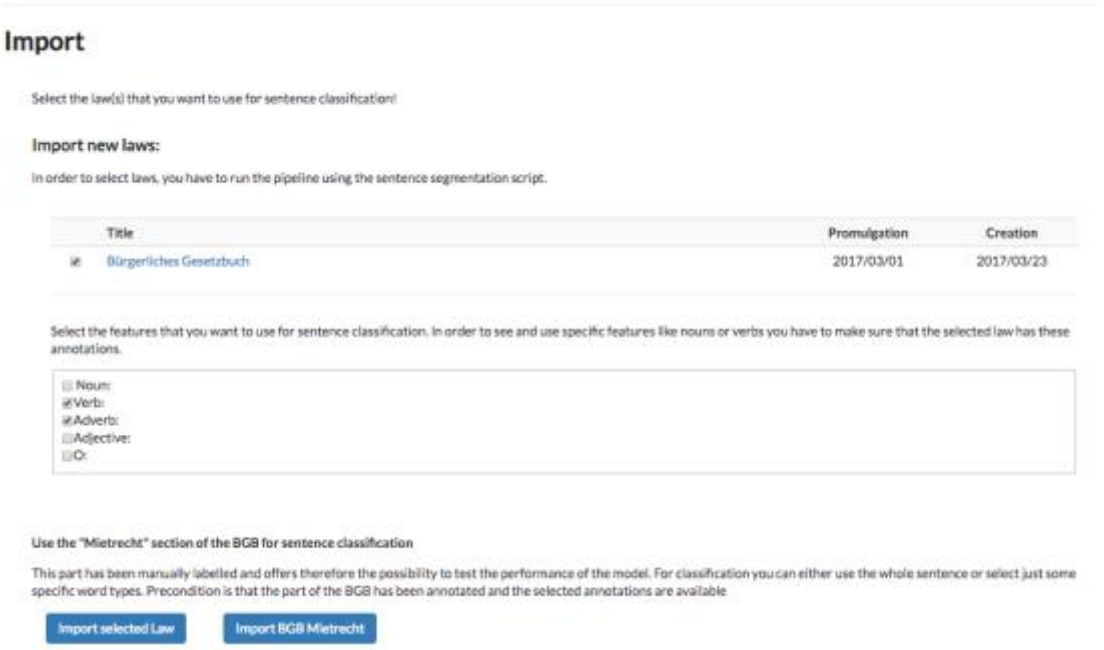


Figure 29: Importing of Legal Texts
Source: Own Illustration

The importing of documents works slightly differently. Here, specific documents cannot be selected, but the maximum number of each document type that should be imported into LexML can be specified.

Once the legal data has been imported from Lexia into LexML, the number of imported sentences or documents is visible to the user.

The last configuration step is to configure all necessary AL settings, such as the classifier, the query strategy, stopping criterions, and the size of the seed set and query set (see Figure 30).

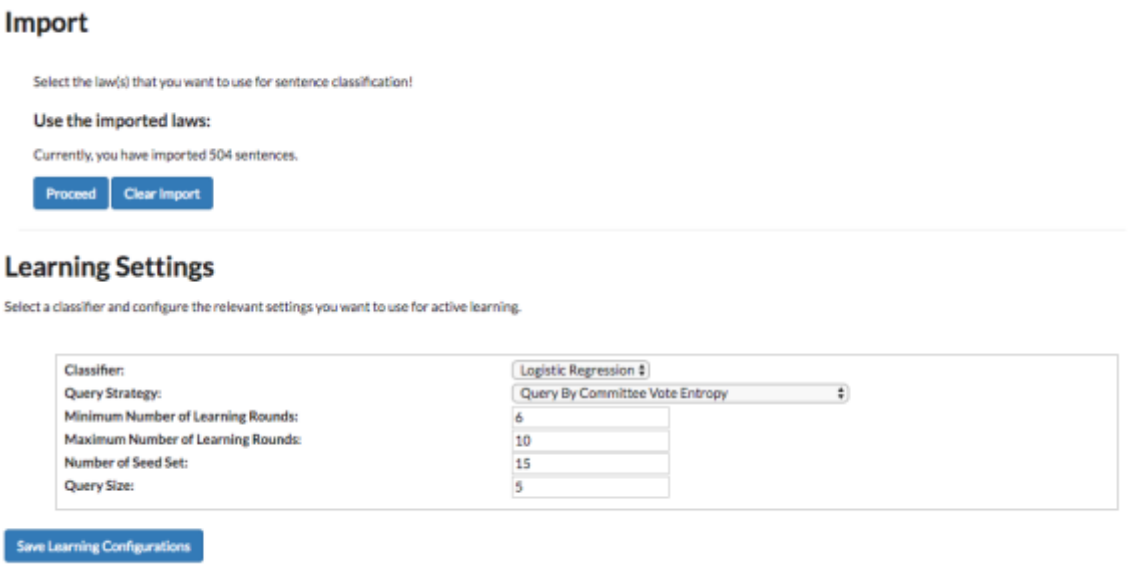


Figure 30: Configuring AL settings
Source: Own illustration

Once an AL pipeline has been configured, it is ready to be processed. The starting point for all activities that refer to the training or the evaluation or prediction of an AL pipeline is the *processing page*, which can be reached from the *landing page*. From here, the user can commence the AL process either with or without evaluation of the learning progress. In cases where this training process has been executed once so that a pipeline model has been created, evaluation of this trained pipeline model is possible. Furthermore, this model can be used to make predictions about the remaining unlabeled instances.

In Figure 31, the labeling of instances, exemplified for DC, is illustrated. As the id of a Lexia annotation is persisted in LexML, more information about the legal document, such as its title and the promulgation date, can be retrieved from Lexia and displayed to the user. As the user, in this example, is already on the second learning round, the prediction made by the classifier and its confidence are also visible to the user. To engender a degree of understanding of the label of the legal document, the beginning of the content of the document is also made available. The label must then be marked in the dropdown menu in the rightmost column so that it can be used for learning in the next round.

Figure 31: User View When Labeling Legal Documents
Source: Own illustration

Once the training process is complete, the resulting model can be used either for evaluation if a test set was created, or for predicting unlabeled instances. On the *evaluation page*, all general important evaluation metrics (see Figure 32) are visible to the user. In addition, these evaluation measures can be exported in xlsx format for further analysis.

Figure 32: Visualization of General Evaluation Measures
Source: Own illustration

6 Evaluation

The evaluation chapter comprises two key sections. The first deals with the assessment of the classification results obtained in DC and NC using Lexia and LexML. The second verifies the that the built prototype achieves the objectives and requirements defined in Chapter 4.2.

6.1 Evaluation of Legal Text Classification

As LexML was constructed to be particularly flexible and customizable, both classification objectives can be conducted with only minimal adjustments. The basic AL setting are the same for both tests (see Chapter 6.1.1). Following a description of this basic experimental design, the particularities and results of each type are detailed.

6.1.1 Experimental Design

This section provides an overview of the experimental design used for DC and NC. The objective is to generate insight into how well DC and NC work utilizing Spark MLlib and German legal texts. Furthermore, the effectiveness of each classifier is analyzed using different query strategies. In LexML, three classifiers (NB, LR, and MLP) and four query strategies (US: margin sampling, vote entropy; and QBC: vote entropy, soft vote entropy) were implemented. As MLP does not produce an output score, only the QBC vote entropy approach could be used with this classifier. At the end, nine different AL combinations are evaluated (see Table 13). Additionally, for each classifier, the performance of random learning (RL), also called passive learning (PL), is tested. Thereby, in each learning round random instances are queried from the dataset without applying any query strategy.

Classifier	Query Strategy
Multinomial Naïve Bayes	Entropy, Margin Sampling, QBC Vote Entropy, QBC Soft Vote Entropy
Logistic Regression	Entropy, Margin Sampling, QBC Vote Entropy, QBC Soft Vote Entropy
Multilayer Perceptron	QBC Vote Entropy

Table 13: Combination of All Evaluation Settings Used

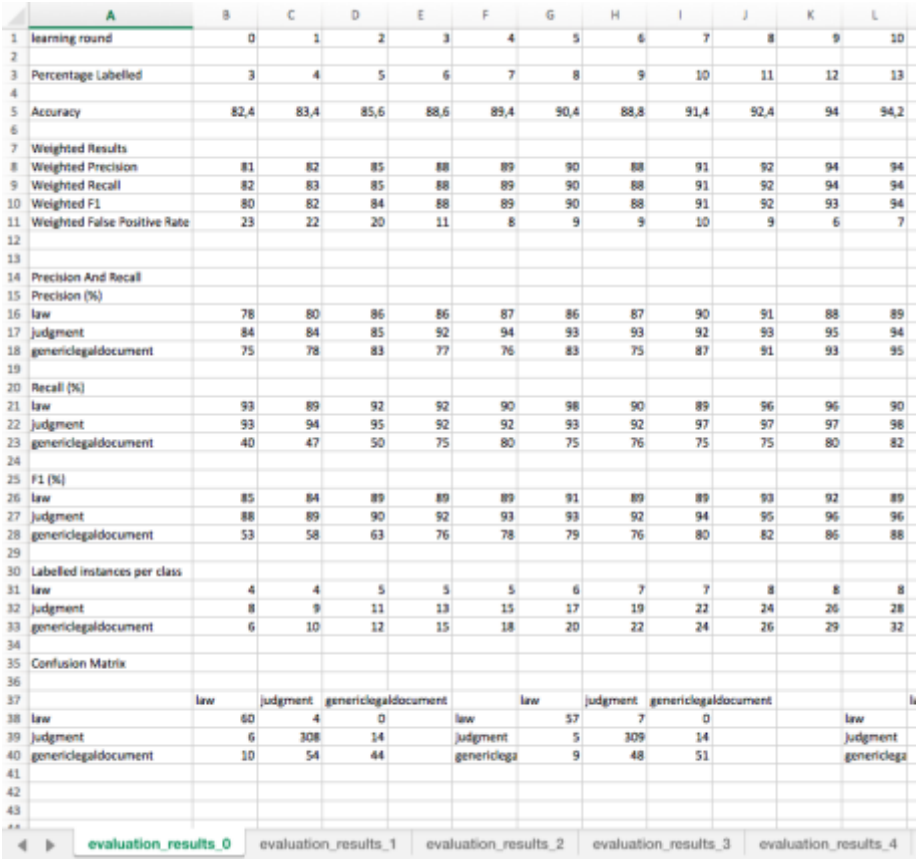
To improve the significance of the classification, each of the 12 combinations was executed five times. The average result of the five rounds was then used as reference value.

As LexML was designed to handle different kinds of TC, the basic process was the same for both DC and NC and was conducted as described in Chapter 4.3.3. The only difference from this chapter was that the labeling was not executed by a user, but by a computer simulating the labeling process to shorten the duration of the experiment. This was possible as a large labeled dataset for both classification types was available. In order to highlight a number of the specific characteristics of this process, it is summarized below.

Essentially, for both experiments, the existing labeled dataset was divided into a training set and an independent test set. The training set was composed of a dataset containing the unlabeled data, and one containing the already labeled data. At the beginning, all training data was added

to the unlabeled dataset. The labeled dataset consisted only of the dummy variables created (see Listing 3).

In the first round alone, several random instances were selected from the unlabeled training set, labeled by the computer and added to the labeled training set. This labeled training set was used to train a pipeline model using a Spark ML Pipeline (see Listing 4). The ML pipeline consisted of four (DC) or rather three (NC) of the following stages. First, a tokenizer split up the textual content into single words. In case of DC, these words were then searched for occurring German stop words, which were removed. This action was not performed for NC, as certain word sequences containing stop words there are meant to aid in distinguishing between classes. For both experiments, the (remaining) sequence of terms was mapped to their term frequencies (TF vector). In a final step, this feature mapping was used by the classifier for training. The resulting pipeline was subsequently utilized to make predictions about the unlabeled dataset. For both generative classifiers used, five-fold cross validation was applied to ensure that the predictions were made with the best model found. Based on the selected query strategy, the most informative instances were removed from the unlabeled dataset, labeled by the computer, and added to the labeled dataset. This learning process was repeated until all instances from the unlabeled training set had been labeled. After each round, the resulting pipeline model was applied to the test data to evaluate the performance of the model used. The evaluation measures obtained were exported to an xlsx file so that the learning could be analyzed in a suitable manner (see Figure 33).



	A	B	C	D	E	F	G	H	I	J	K	L
1	learning round	0	1	2	3	4	5	6	7	8	9	10
2												
3	Percentage Labelled	3	4	5	6	7	8	9	10	11	12	13
4												
5	Accuracy	82,4	83,4	85,6	88,6	89,4	90,4	88,8	91,4	92,4	94	94,2
6												
7	Weighted Results											
8	Weighted Precision	81	82	85	88	89	90	88	91	92	94	94
9	Weighted Recall	82	83	85	88	89	90	88	91	92	94	94
10	Weighted F1	80	82	84	88	89	90	88	91	92	93	94
11	Weighted False Positive Rate	23	22	20	11	8	9	9	10	9	6	7
12												
13												
14	Precision And Recall											
15	Precision (%)											
16	law	78	80	86	86	87	86	87	90	91	88	89
17	judgment	84	84	85	92	94	93	93	92	93	95	94
18	genericlegaldocument	75	78	83	77	76	83	75	87	91	93	95
19												
20	Recall (%)											
21	law	93	89	92	92	90	98	90	89	96	96	90
22	judgment	93	94	95	92	92	93	92	97	97	97	98
23	genericlegaldocument	40	47	50	75	80	75	76	75	75	80	82
24												
25	F1 (%)											
26	law	85	84	89	89	89	91	89	89	93	92	89
27	judgment	88	89	90	92	93	93	92	94	95	96	96
28	genericlegaldocument	53	58	63	76	78	79	76	80	82	86	88
29												
30	Labelled instances per class											
31	law	4	4	5	5	5	6	7	7	8	8	8
32	judgment	8	9	11	13	15	17	19	22	24	26	28
33	genericlegaldocument	6	10	12	15	18	20	22	24	26	29	32
34												
35	Confusion Matrix											
36												
37		law	judgment	genericlegaldocument			law	judgment	genericlegaldocument			
38	law	60	4	0		law	57	7	0		law	
39	judgment	6	308	14		judgment	5	309	14		judgment	
40	genericlegaldocument	10	54	44		genericlega	9	48	51		genericlega	
41												
42												
43												

evaluation_results_0 | evaluation_results_1 | evaluation_results_2 | evaluation_results_3 | evaluation_results_4

Figure 33: Example of Exported Evaluation Results in DC

6.1.2 Document Classification

The first classification objective was to classify multiple legal documents into created categories (multiclass classification). An introduction about this topic is provided in Chapter 2.3.2. In what follows, the data used and AL settings, as well as the results of the evaluation are described.

6.1.2.1 Data Used

The dataset for conducting DC was provided by DATEV[57]. It consists of more than 132,000 xml documents. From this set, 1,000 documents were randomly selected and imported into Lexia using its existent import functionalities. As these documents had been manually annotated by DATEV, Lexia could already assign them to the correct class. While DATEV's classification is finely grained, Lexia consolidates this classification into three main document types. This mapping is depicted in Table 14 where the column "Class" complies with Lexia's document classes, and the column "Sub-classes" with the classes used by DATEV. In this experiment, the three labels provided by Lexia are utilized.

Class	Sub-classes
Law ("Gesetz")	Gesetzestext (Gesamttext), Richtlinie (Gesamttext), EU-Richtlinie (Gesamttext),
Judgment ("Urteil")	Urteil, Beschluss, Gerichtsbescheid
Generic legal document ("Sonstige Dokumente")	Aufsatz, Anmerkung, Verfügung, Verfügung (koordinierter Ländererlass), Kurzbeitrag, Erlass, Erlass (koordinierter Ländererlass), Schreiben, Schreiben (koordinierter Ländererlass), Übersicht, Mitteilung

Table 14: Lexia's Legal-Document Mapping

The imported dataset comprised:

- 128 laws (12.8%),
- 655 judgments (65.5%) and
- 216 generic legal documents (21.6%).

This unbalanced distribution reflects common real-world scenarios and provides a realistic sample. Although the number of labels (three) is low for a multiclass classification problem, the unequal distribution makes the classification more complex.

As legal documents, especially laws, can be very long, the imported textual content that is used for ML was limited to the first 4,096 characters to increase the efficiency of AL. Assuming an average word length of 10 letters, approximately the first 410 words of each document were imported into LexML and used for AL. After the stop-words had been removed, the textual content was transformed into the TF vector described (bag of words representation) and used by the classifier.

[57] https://www.datev.de/

6.1.2.2 Data preparation

As the dataset available for DC is extremely large, a large test set with 500 randomly selected documents was created in LexML. The remaining 500 documents were used for iterative training. Consequently, the *training dataset* consisted of:

- 65 laws (13%),
- 327 judgments (65.4%) and
- 108 generic legal documents (21.6%),

and the *test dataset* of:

- 64 laws (12.8%),
- 328 judgments (65.6%) and
- 108 generic legal documents (21.6%).

6.1.2.3 AL Pipeline Preparation

While the selection of the classifier and query strategy changed every five rounds, the basic AL pipeline configurations remained the same for each DC experiment. These basic configurations are presented in Table 15.

Name	Configuration
Number of learning rounds (stopping criterion)	97
Size of seed set	15
Size of query set	5

Table 15: Basic AL Pipeline Configurations Used in the DC experiment

In the first round, 15 random instances were queried from the unlabeled training set, labeled and used for learning (seed set). In the next rounds, either the five most informative instances, in the case of a query strategy, were used or otherwise, five random instances were removed from the unlabeled training set, labeled and added to the labeled training set. As all documents of the training set should have been labeled, 97 learning rounds were conducted (15 + 97*5 = 500).

6.1.2.4 Evaluation

As described in Chapter 6.1.1, each combination of classifier and query strategy was executed five times (see A.1 Examples for Averaging the Learning Rounds). The average value for five rounds was then used to answer the following four main questions:

1) Which implemented classifier performs best?
2) Is AL superior to PL?
3) Which implemented query strategy performs best?
4) Which documents are best recognized by the classifier(s)?

To answer these questions, some of the common evaluation measures described in Chapter 2.1.2.7, and the learning curves described in Chapter 2.2.8 are used.

1) Which Implemented Classifier Performs best?

Figure 34 compares the three implemented classifiers performing PL in which the labeling sequence is not considered and only random instances are queried (without using any query strategy). While all classifiers have a good *accuracy* of between 70%-80% from the start (15 labeled instances), NB's and MLP's slope is steeper than LR's one. Once about 50% of the instances (250) have been labeled, the *accuracy* of all classifiers increases only marginally. Both the NB and MLP yield continuously more or less the same *accuracy*, having a final maximum *accuracy* of about 95,5%. By contrast, LR achieves only a final average *accuracy* of ≈92%.

Hence, in case of DC, NB and MLP are undoubtedly the better choice, as they learn faster and achieve a higher overall maximum *accuracy*.

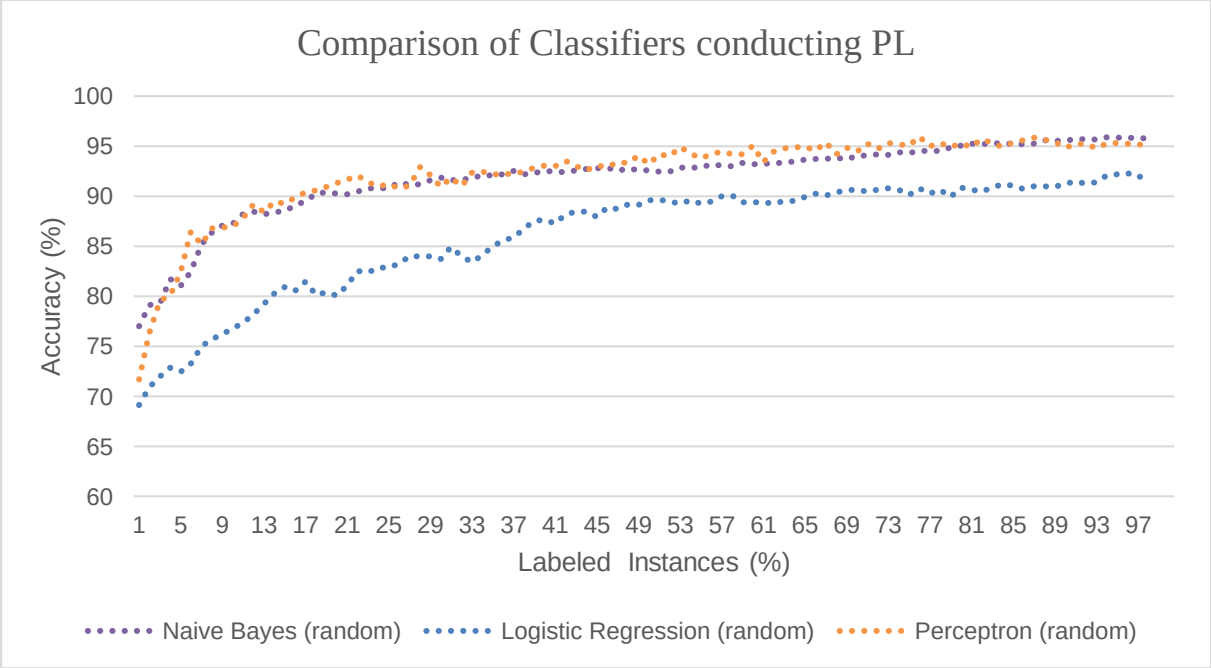


Figure 34: Comparison of Classifiers Conducting PL
Source: Own illustration

2) Is AL Superior to PL?

Figure 35 compares the PL curves, known from Figure 34, to the average *accuracy* of each classifier consolidating the result of all query strategies used (*consolidated accuracy*). For instance, in case of NB, the average *accuracy* obtained from the four query procedures – margin sampling, vote entropy, QBC vote entropy, and QBC soft vote entropy – was again averaged (see Table 13).

It becomes evident that AL is clearly superior to PL as the average *accuracy* of all classifiers using AL is above the *accuracy* of PL until a certain point. Regardless of which classifier is

employed, AL methods result in a much faster learning. NB and MLP achieve their maximum *accuracy* at already about 35%-40% of the labeled instances (175-200). Labeling the remaining 300 instances does not result in a better *accuracy*. Moreover, LR performs far better when using AL. It attains its maximum at about 55% of the labeled instances (275), reaching an even higher *accuracy* (≈94,5%) than when labeling all instances.

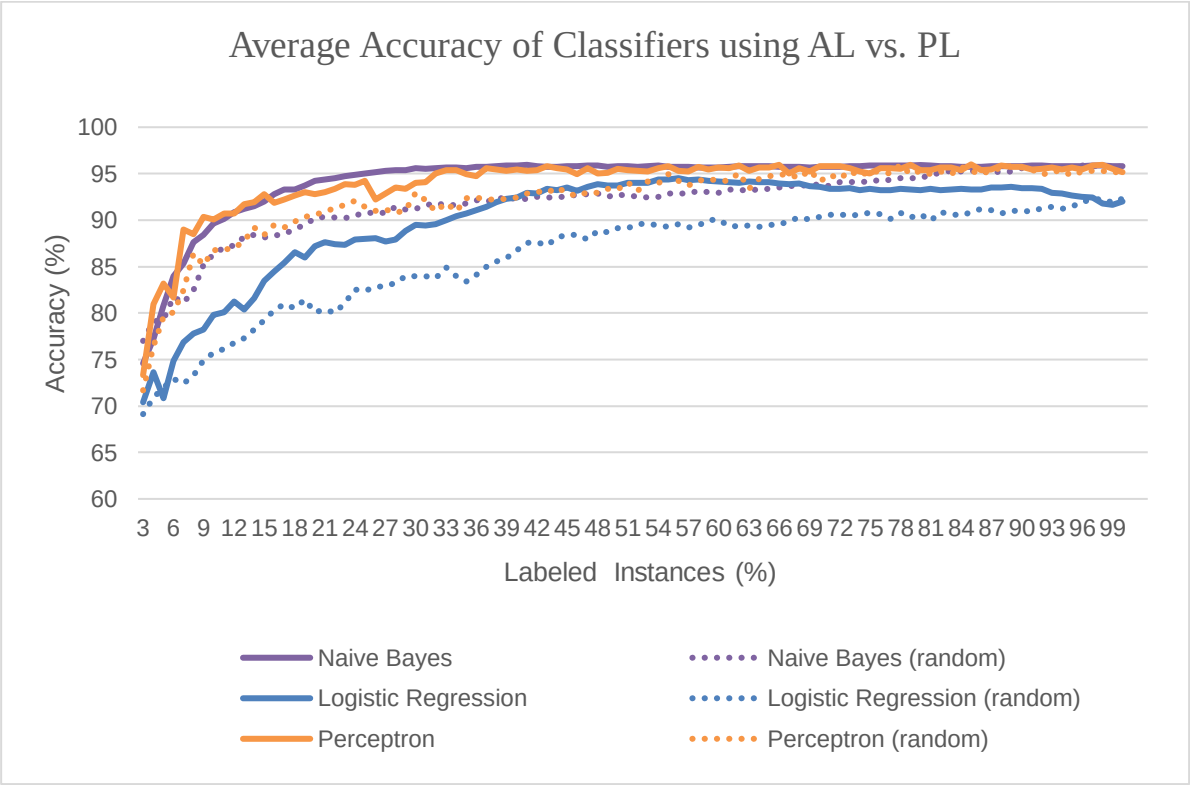


Figure 35: Average Accuracy of Classifiers Using AL vs. PL
Source: Own illustration

As the number of labeled instances increases, both curves of one classifier converge and intersect at a certain point. Until this point, AL is superior to PL.

Hence, one can conclude that AL is markedly superior to supervised learning in the case of DC, not only due to faster learning, but also because of the achievement of a higher maximum *accuracy* while using more than 40% fewer instances. This is especially valid for LR. Other evaluation measures depicted in appendix A.2 Overall evaluation of DC confirm these findings.

3) Which Implemented Query Strategy Performs best?

Figure 36 shows the learning curves of the NB classifier together with each implemented learning strategy, as well as the one conducting PL. It appears that it does not matter which query strategy is used: AL is always superior. It is also observable that all four query strategies show almost the same learning curve. In this case, the use of more elaborate and time-consuming QBC procedures does not result in better learning for the classifier.

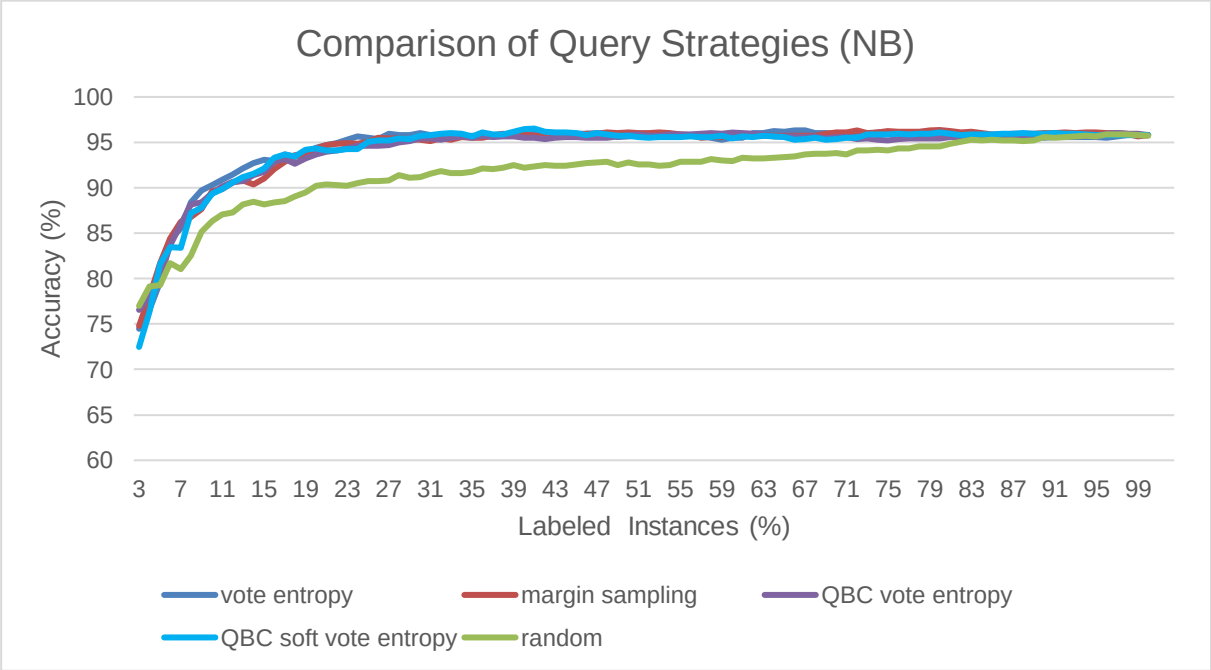


Figure 36: Comparison of Query Strategies (NB)
Source: Own illustration

Figure 37, in which the several learning curves of LR are compared, shows similar findings. Apart from the vote entropy strategy, the other query strategies have the same curve progressions, all superior to PL. The vote entropy strategy exhibits some problems at the beginning, but, at around 30% of the labeled instances, it adapts to the other strategies and from that point on is superior to PL.

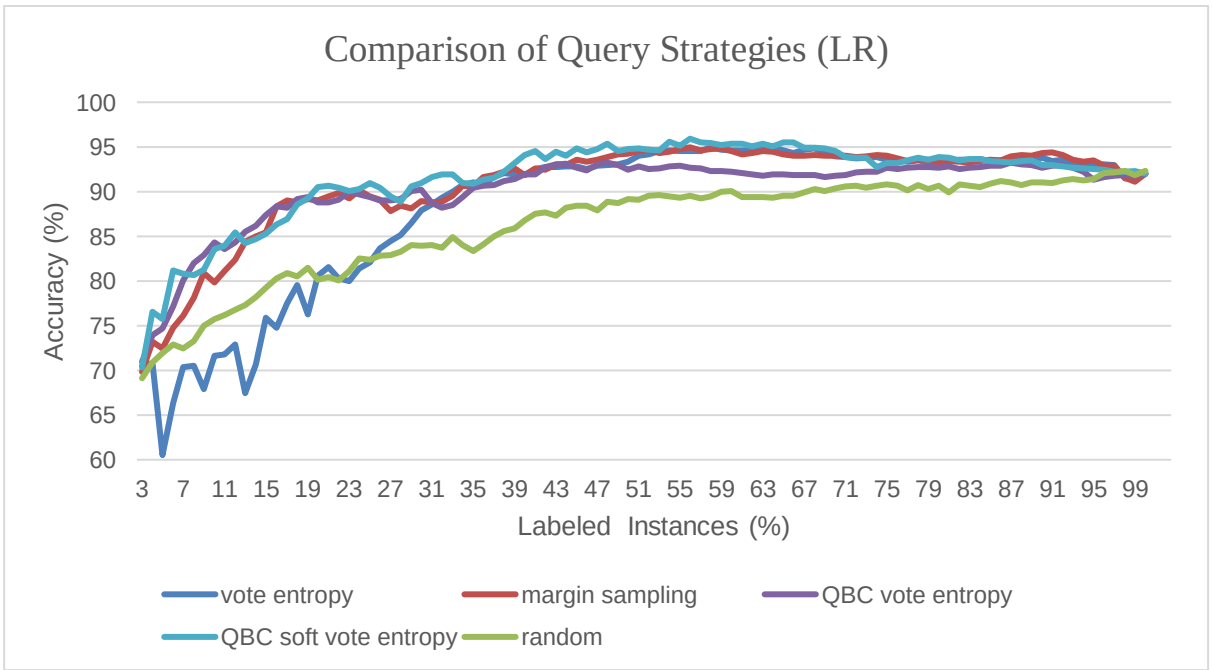


Figure 37: Comparison of Query Strategies (LR)
Source: Own illustration

It is difficult to say whether the vote entropy strategy fits LR in case of DC. As the *accuracy* increases sharply after ≈12%, it is more likely that the randomly queried instances at the beginning misled the classifier to query in the "wrong direction". The viewing of the individual learning rounds (see Figure 49) strengthens this theory as the results from two rounds are very good, and another two rounds yield results comparable to the other query strategies. Only in one round does it initially show a poor performance, which, however, decreases the average value considerably.

In summary, in the DC experiment it could not be concluded that one query strategy constantly outperforms the others. Rather, it appears that all the query strategies work equally well.

4) Which Documents are best Recognized by the Classifier(s)?

Figure 38 illustrates the classification performance per document class, utilizing the average F_1 of each classifier consolidating the result of all query strategies (*consolidated* F_1 for NB, LR and MLP). This consolidation can be accomplished without falsifying the outcomes, as the previous evaluation analysis has demonstrated that all query strategies work equally well. *Judgments* are recognized very well from the start. It may be that this the case because judgments are by far the most frequent documents in the training and test sets. Hence, the probability that *judgments* are initially queried is high. After 10%-20% of the labeled instances, *laws* are recognized as well as *judgments* are, having an average F_1 of more than 96%. In contrast, all classifiers have problems recognizing *generic legal documents*. Although the curve increases strongly in a manner similar to the one for laws, the average maximum F_1 value never exceeds the 90%. This might be on the result of the great variety of types of legal document that are consolidated in this class (see Table 14). It is possible that some subtypes of these documents in the test set might not have been part of the training set.

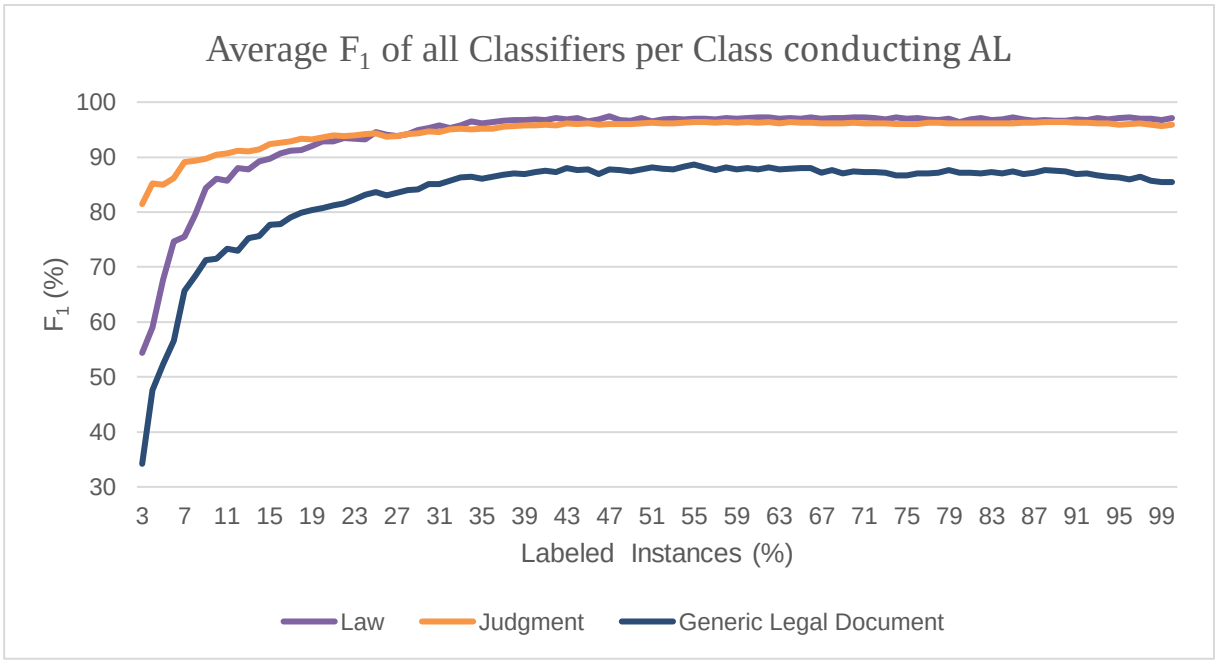


Figure 38: Average Accuracy of All Classifiers per Class
Source: Own illustration

In the preceding pages, it was demonstrated that NB is most stable and most precise classifier. Viewing only the averaged results of this classifier per document class, very similar curve progressions are observable, except that the slope is slightly steeper and the maximum F_1 value is minimally higher for all three document classes. Nevertheless, the class *generic legal document* (*generic*) does not achieve 90%.

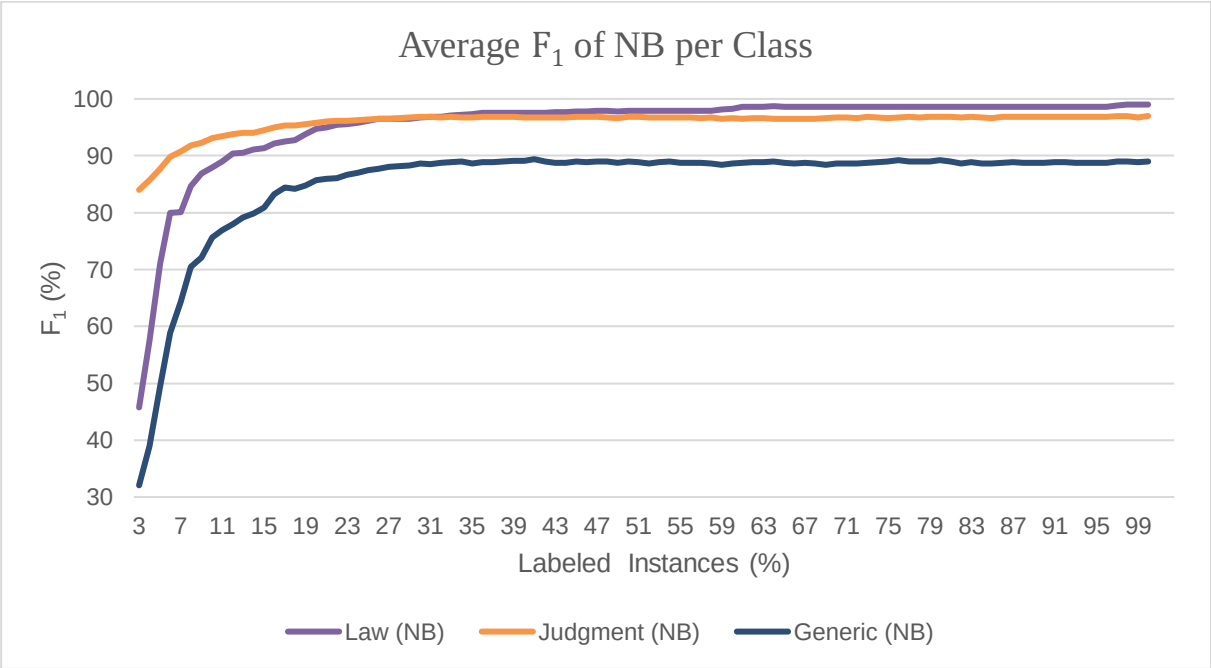


Figure 39: Average F_1 of NB per class
Source: Own illustration

These curves become clearer when the final confusion matrix of the NB classifier is viewed (see Table 16). The predictions are illustrated in the columns while the rows reflect the correct label. Hence, 63 *laws* were correctly classified, resulting in an average *precision* of 100% and a *recall* of 98%. Only one *law* was misclassified as a *generic legal document*. All 328 *judgments* in the test set were classified correctly, achieving a *recall* of 100%. However, as 20 *generic legal documents* were also classified as *judgment*, the *precision* is only 94%. All other *generic legal documents* were recognized as such, reaching a *precision* of 98% and a *recall* of 81%. The distribution of the *precision* and *recall* curves for NB can be found in the appendix (see A.3 F_1, Precision and Recall per Class for DC).

	Law	**Judgment**	**Generic Legal Document**
Law	63	0	1
Judgment	0	328	0
Generic Legal Document	0	20	88

Table 16: Final Confusion Matrix for DC Using NB as Classifier
Source: Own illustration

In summary, it can be concluded that the document types *law* and *judgment* are recognized very well by the classifier, especially by NB and MLP. In contrast, although almost 90% is still a very good F_1 score, the correct classification of *generic legal documents* was more difficult for all classifiers.

6.1.3 Norm Classification

The second classification objective was to classify the content of legal documents, as presented in Chapter 2.3.3. As a result of the flexible design of LexML, the basic experimental settings described in Chapter 6.1.1 differ only in one respect: the pipeline stage stop-word removal is eliminated for the norm classification experiment. As in the previous section, the data used, the adjusted AL settings, and the evaluation results obtained are described.

6.1.3.1 Data Used

In order to prepare a suitable dataset, a legal expert assigned to a specific category each sentence in the law of tenancy section ("Mietrecht") in the German civil code (BGB, §535-§595) published on March 1st, 2017. The result was 532 labeled sentences using 16 different labels. As eight of the 16 labels had a support lower than 1.2%, they were removed from the dataset. The 504 remaining sentences were composed of the eight classes presented in Table 17.

Class	Sub-classes	Occurrences	Support
Recht	Gebot (positiv/negativ/Soll-Pflicht (H)), Verbot (/Duldung (U))	126	25.00%
Pflicht	Erlaubnis (/Erlaubnis beschränkt), Ermächtigung	109	21.63%
Einwendung (Einw rh)	Unwirksamkeit (/Wirksamkeit beschränkt (sachlich)/(persönlich)), Unzulässigkeit (/Zulässigkeit beschränkt), Ausschluss (/Ausschluss beschränkt), Nichtigkeit	92	18.25%
Rechtsfolge (RF vAw)	Rechtzuweisung (/Rechtsübergang), Pflichtzuweisung (/Pflichterweiterung), Rechtseigenschaft, Freistellung	50	9.92%
Verfahren	Form, Frist (/Fälligkeit), Maßstab (inkl. Berücksichtigung), Entscheidungskompetenz	49	9.72%
Verweisung	Direkt § /direkt Vorschriftenkomplex, Analog § /analog Vorschriftenkomplex, Negativ	46	9.13%
Fortführungsnorm	Ausnahme, Erweiterung, Einschränkung, Gleichstellung	19	3.77%
Definition	Direkt, Indirekt, Negativ	13	2.58%
		504	**≈100%**

Table 17: Taxonomy of Dataset Used for NC

The "class" column represents the label that was used as the classification class in this experiment. As described for DC, all classes referring to one of the sub-classes are consolidated to the parent class.

The data distribution was similar to that for DC – a high imbalance, with a range from only ≈2.5% support to 25% support. Additionally, the number of labels increased from three to eight. As additionally the textual content available to differentiate between the individual classes is much less, this classification problem is far more complex than the DC experiment.

The 504 sentences listed were prepared in csv format and imported into LexML. An extract of this csv file is provided in appendix B.1 Prepared csv-file for NC. The preparation of this dataset for training and evaluation in LexML is described in the section that follows.

6.1.3.2 Data preparation

As the available dataset for NC is smaller than the one used for DC, only 126 sentences of the dataset (25%) were randomly added to the test set. The remaining 378 sentences (75%) were used for iterative training. The detailed allocation of training and test data for the NC experiment is presented in Table 18.

Class	Training	Support	Test	Support
Recht	96	25.40%	30	23.81%
Pflicht	83	21.96%	26	20.63%
Einwendung rh	61	16.14%	31	24.60%
RF vAw	40	10.58%	10	7.94%
Verfahren	39	10.32%	10	7.94%
Verweisung	39	10.32%	7	5.56%
Fortführungsnorm	11	2.91%	8	6.35%
Definition	9	2.38%	4	3.17%
	378 (75%)	**≈100%**	**126 (25%)**	**≈100%**

Table 18: Allocation of Training- and Test Data
Source: Own illustration

6.1.3.3 AL Pipeline preparation

While the possible combinations of the classifiers and query strategies remained the same, the basic AL pipeline configurations were adapted to the modified training set. The configurations used for NC are presented in Table 19.

Name	Configuration
Number of learning rounds (stopping criterion)	72
Size of seed set	18
Size of query set	5

Table 19: Basic AL Pipeline Configurations Used in the NC Experiment

Due to the more complex situation, the size of the seed set was increased from 15 to 18 to increase the probability of discovering more class types. These 18 random instances were queried from the unlabeled training set, labeled and used for learning in the first round (seed set). In the subsequent rounds, again either the five most informative instances in the case of a query strategy were used; or five random instances were removed from the unlabeled training set, labeled and added to the labeled training set. As all sentences of the training set should labeled, 72 learning rounds were conducted (18 + 72*5 = 378).

6.1.3.4 Evaluation

The evaluation process and the evaluation measures are the same as described for DC. Each combination of classifier and query strategy was executed five times. The average value of five rounds was then used to answer the same four questions:

1) Which implemented classifier performs best?
2) Is AL superior to PL?
3) Which implemented query strategy performs best?
4) Which norms are best recognized by the classifier(s)?

1) Which Implemented Classifier Performs Best?

Surprisingly, in this case is the performance of the classifiers in reverse order to that for DC. From the start, LR achieves the highest *accuracy*, obtaining an average maximum *accuracy* of ≈73%. The MLP attains about 66%, and NB only about 55%.

However, that the classification task is more complex than DC is clearly observable. Both the slope of the learning curve is much flatter and the maximum *accuracy* of the best classifier is more than 20% lower than the one attained with the best classifier in DC.

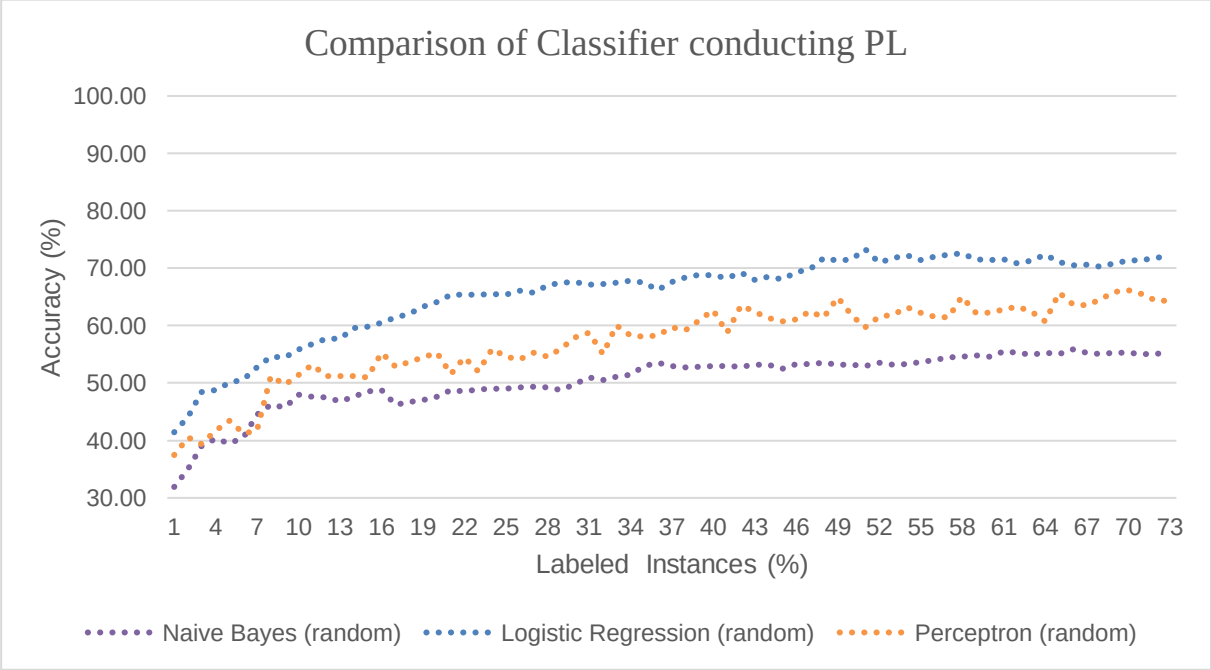


Figure 40: Comparison of Classifiers Conducting PL
Source: Own illustration

Nevertheless, in case of NC, LR is clearly the best choice as it consistently achieves the highest *accuracy*.

2) Is AL Superior to PL?

As with DC, the *consolidated accuracy* (using the average *accuracy* of all four combinations of LR and NB) is opposed to the *accuracy* obtained from PL (random approach) to evaluate this question (see Figure 41).

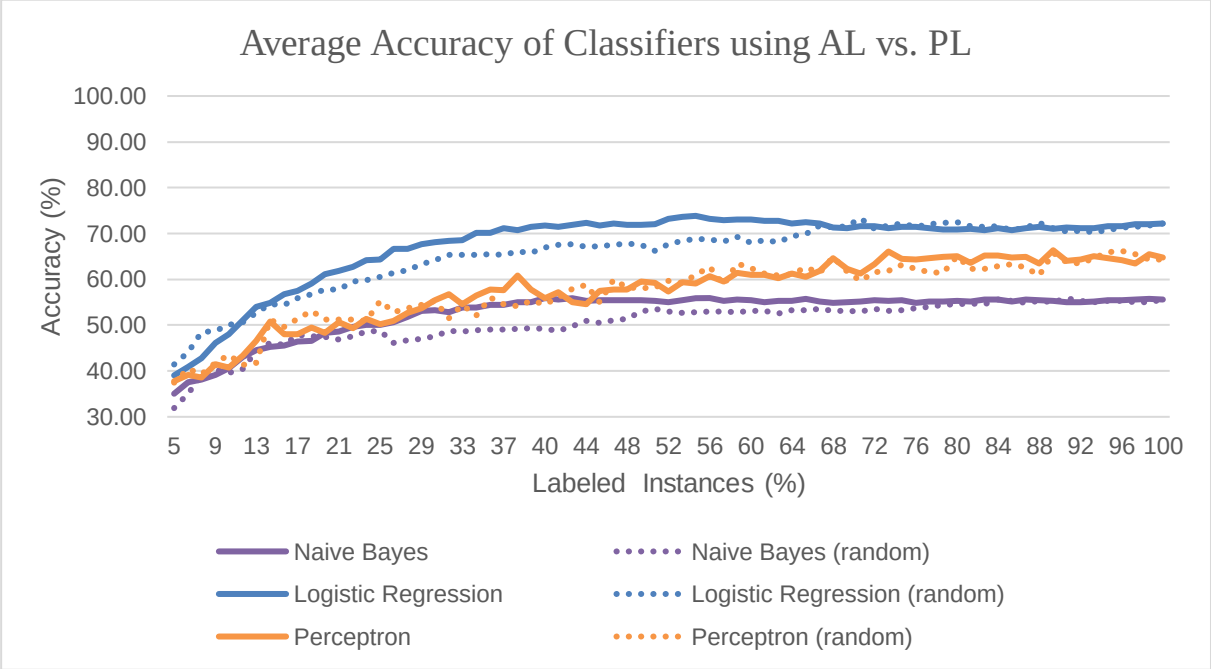


Figure 41: Average Accuracy of Classifiers Using AL vs. PL
Source: Own illustration

The evaluation again emphasizes that AL is superior to PL, although the differences are not as strong as they were for DC. Particularly for MLP, there is hardly any difference between PL and AL learning using the QBC vote entropy strategy.

However, the use of AL results in faster learning and a higher overall maximum *accuracy* for the classifiers LR and NB. The impact of AL first becomes visible after labeling about 10%-20% of the instances (≈37-75). This is probably caused by the large version space that must be discovered by the classifier before it can be searched for useful instances to delimit it in a meaningful way. After that point, the AL procedures consistently achieve a higher *accuracy* until 70%-80% of the labeled instances. The *accuracy* at the point of having labeled about 40% (NB) and 55% (LR) is even minimally higher than the final *accuracy* using the whole training set. Labeling the remaining instances is not useful for the classifier: it is unhelpful due to overfitting of the classifier.

Hence, the results of the NC show that AL is preferable to PL, even if the results are not significant for MLP. The classifiers NB and LR yield a better result having labeled only about half of the instances in the training set. Other evaluation measures are depicted in appendix B.2 Overall evaluation of NC.

At this point, the importance of having a seed set of a high quality should be reconsidered. Figure 42 shows the curve progressions of the five AL rounds using LR as classifier and US vote entropy as query strategy. It is observable how essential an early discovery of the version space is. As the seed set was created randomly, these progression curves exhibit major differences in the beginning. A balanced seed set having higher coverage of the version space results in an almost 20% higher *accuracy* having labeled only 17% of the instances. Further, a maximum *accuracy* of almost 80% can be achieved having labeled only 35% of the instances. This represents an increase of more than 6% compared to PL while using 65% fewer instances.

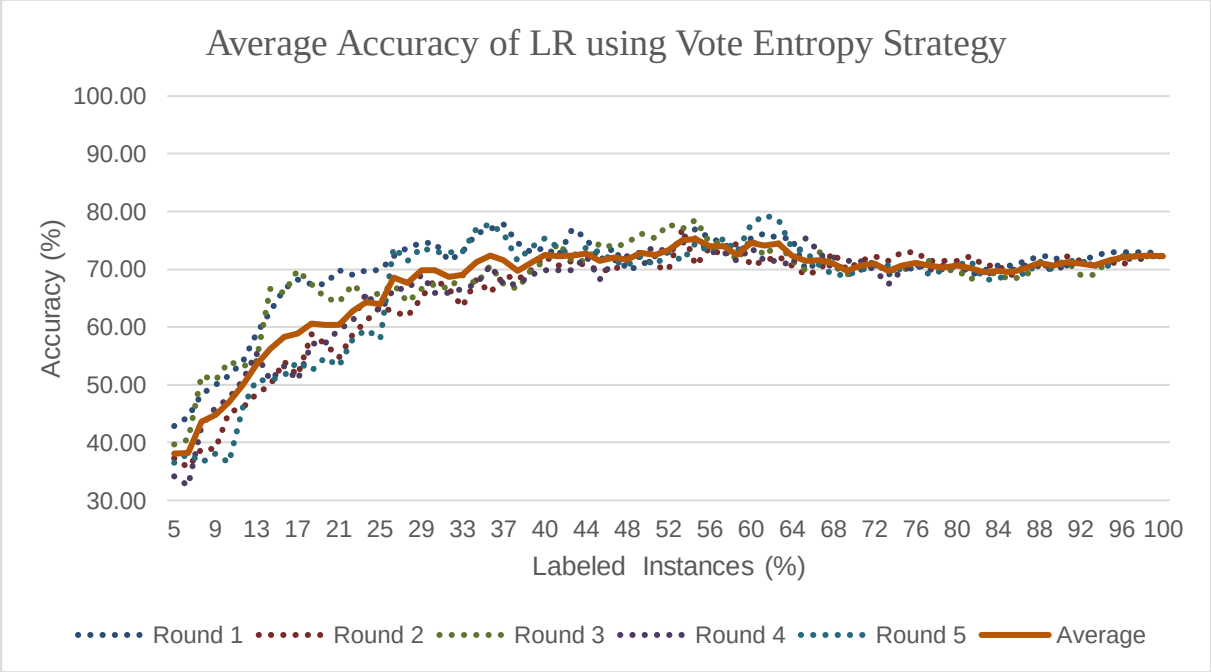


Figure 42: Average Accuracy of LR Using Vote Entropy Strategy
Source: Own illustration

3) Which Implemented Query Strategy Performs best?

Figure 43 and Figure 44 provide an overview of the performance of the individual query strategies applicable for NB and LR. The outcome of the previous questions is again confirmed. After a certain discovery phase, compared to random learning, all query strategies are almost always predominant until a specific point.

When using NB, the vote entropy strategy showed the best performance, reaching an average maximum *accuracy* of 60% in the five rounds. All other strategies exhibit a very similar curve so that no noticeable difference is observed at the beginning. But after approximately 50% of the labeled instances, the margin sampling strategy intersects with the random learning curve, whereas the other learning curves intersect much later (at 80%-90% of the labeled instances).

Looking at the curve progressions of LR, again no noticeable difference is observable. After the discovery phase, all classifiers have a very similar form and intersect with the random learning curve roughly at the same point (≈65% of the labeled instances). Only the QBC vote-entropy strategy shows a short downturn at ≈30%, but adjusts in relation to the other strategies quite soon.

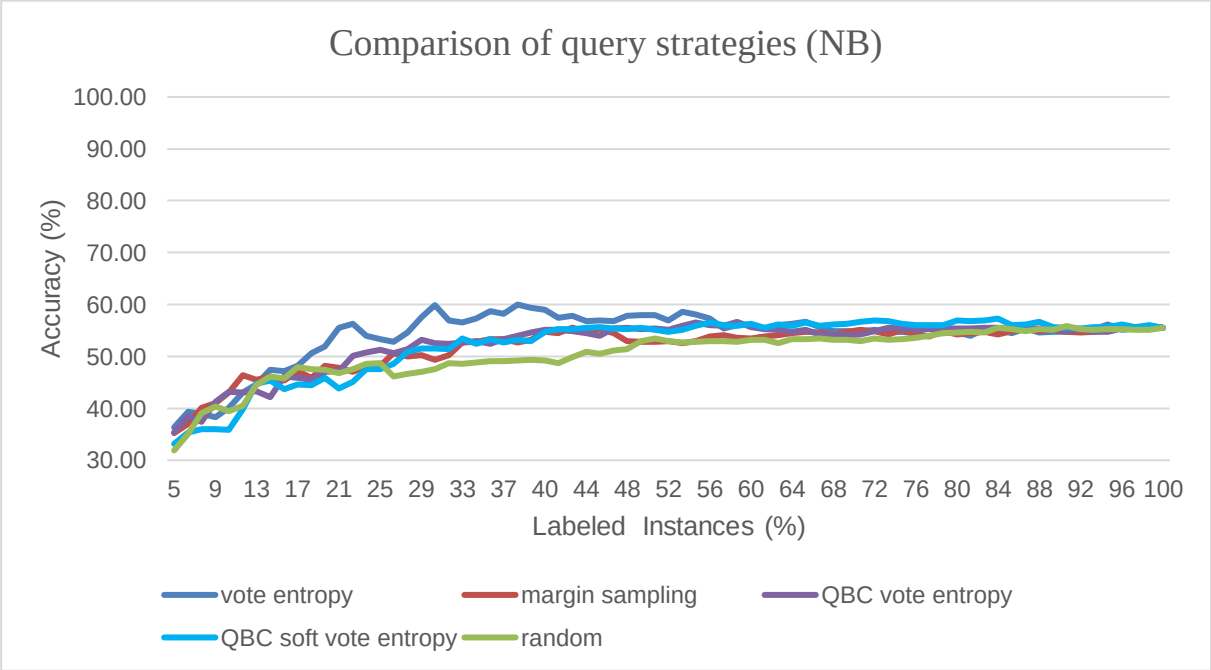


Figure 43: Comparison of Query Strategies (NB)
Source: Own illustration

In summary, in the evaluation of NC, no query strategy has been shown to be predominant relative to the others. However, the superior results of the vote-entropy strategy when using NB as classifier again demonstrate that the more elaborated QBC strategies are not necessarily better.

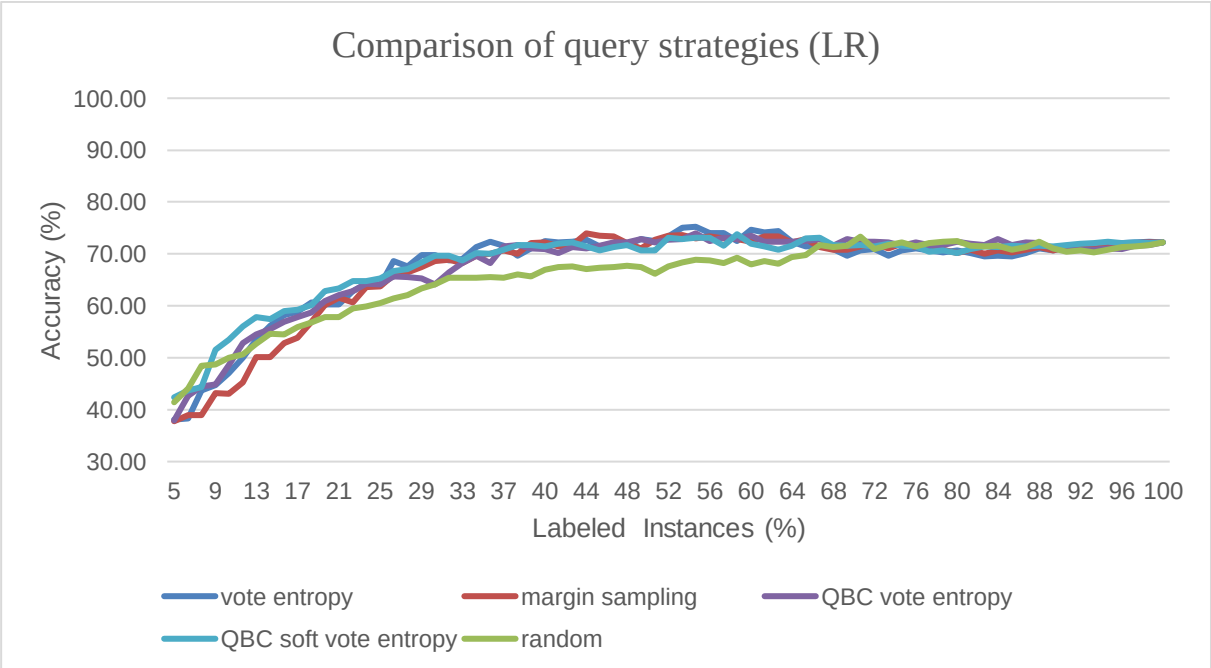


Figure 44: Comparison of Query Strategies (LR)
Source: Own illustration

4) Which Norms are best Recognized by the Classifier(s)?

To answer this question, the results of the best classifier, LR, are discussed below. A short overview about the results of the other classifiers is given in appendix B.3 F_1, Precision and Recall per class for NC.

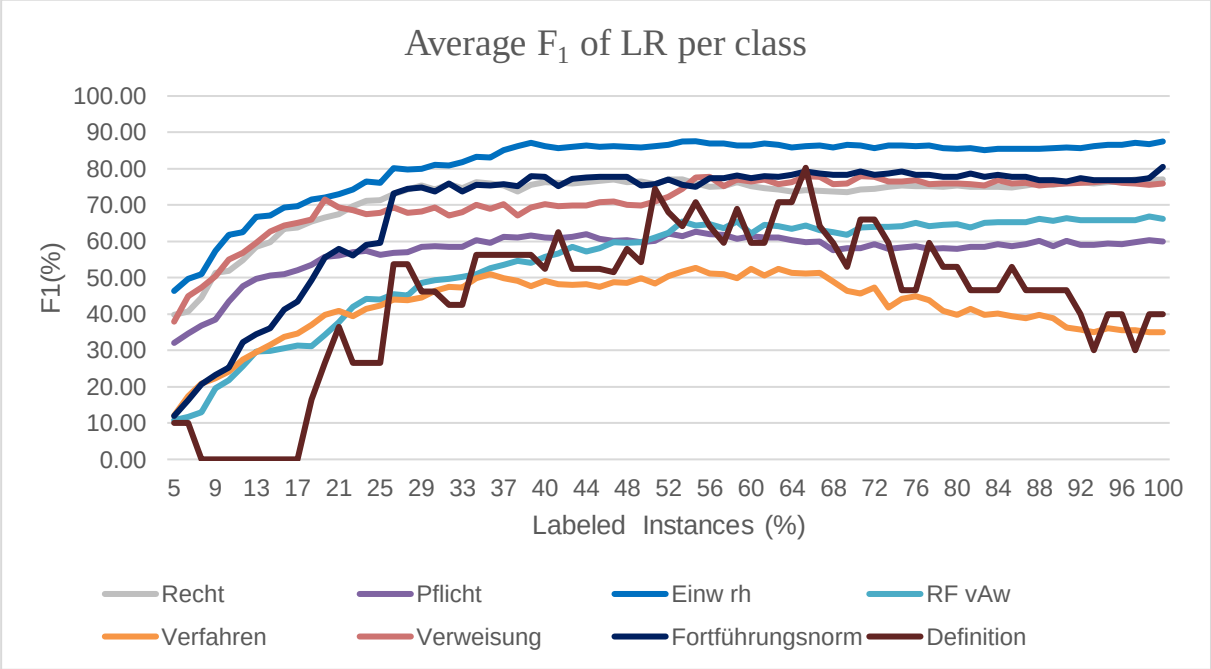


Figure 45: Average F_1 of LR per Class
Source: Own illustration

Figure 45 illustrates the classification performance per class (norm) using the average F_1 of LR consolidating the result of all query strategies (*consolidated F_1*). This consolidation can also be performed for NC again without falsifying the outcomes as all query strategies work equally well.

The different results of the individual labels are very noticeable. While norms belonging to the class *Einwendung rh* (Einw rh) are well recognized, soon having an F_1 of almost 90%, towards the end, norms referring to the class *Definition* or *Verfahren* cannot be classified easily by a classifier. The reason for the low end-value for the class *Definition* might be their low support – less than 3% – resulting in only a very small training set. However, the training set for the class *Fortführungsnorm* contains only two more instances and this has an F_1 value of more than 80%. Hence, the classifier might have problems distinguishing a *Definition* from another class, or the kind of *Definitions* in the training set vary from those of the test set (e.g. a different sub-class). As the class *Definition* has 100% *precision* but a low *recall* (see Figure 46 and Figure 47), the reason is probably the latter. As the argument that the training set is too small does not apply for the class *Verfahren* (which comprises almost 10% of the instances in the dataset), and both the *precision* and *recall* are low, the reason for the poor performance is probably caused by a lack of distinction. Although the number of training instances is high for the class *Pflicht*, the classifier has problems recognizing them. As both *precision* and *recall* are low, possible inaccuracies might be based on both the lack of distinction between other classes and diverse formulations within this class. The class *Recht* had the highest *recall* towards the end, but a

rather low *precision*. This could be a result of the fact that the class *Recht* is the most frequent class in the data set.

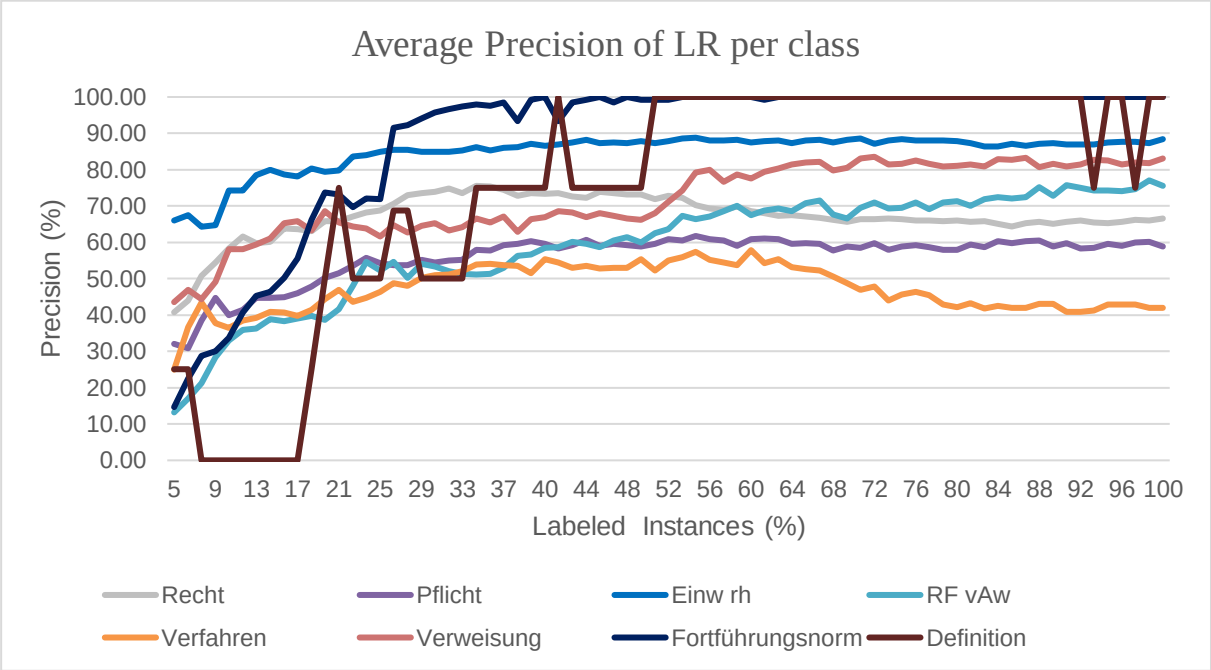


Figure 46: Average Precision of LR per Class
Source: Own illustration

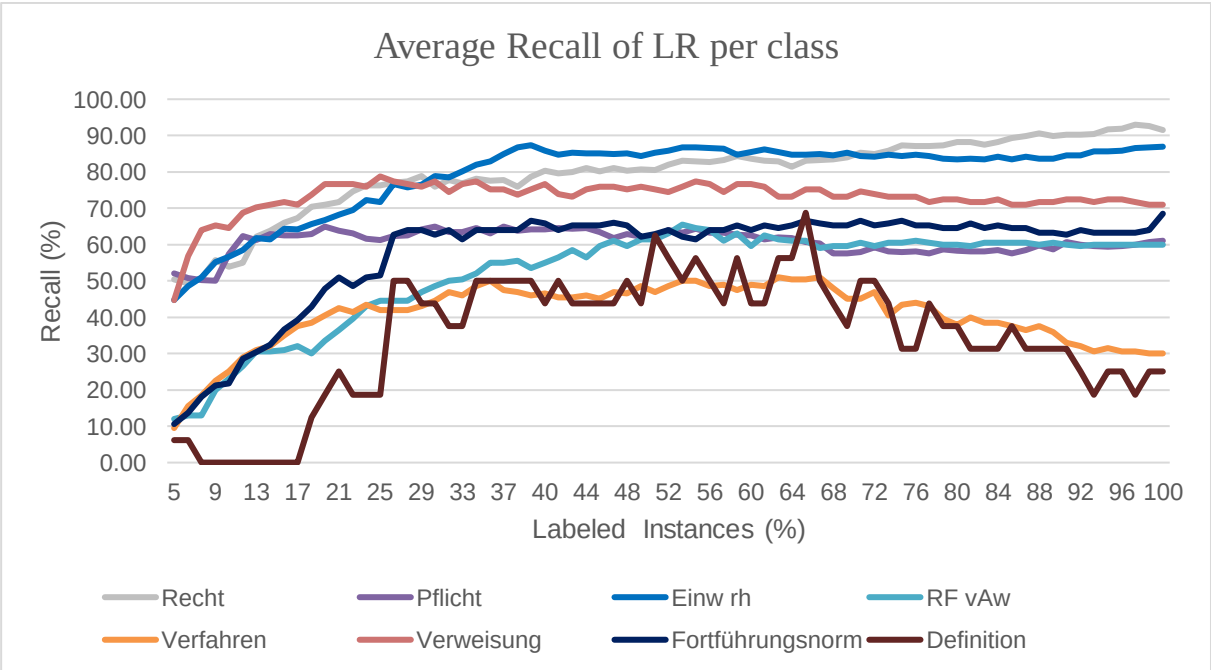


Figure 47: Average Recall of LR per Class
Source: Own illustration

The previous illustrations again highlight the advantages of AL, as well as the fact that all progression curves have their maximum at between 30%-70% of the labeled instances. After that, hardly any class attains a better value; indeed, some fare worse.

As discussion above has focused on especially the end-values, Table 20 presents a confusion matrix at a point where, due to the use of AL, an *accuracy* of about 77% was achieved. It refers to the third round using LR as classifier, and US vote entropy as query strategy having labeled 52% of the instances (see Figure 42). Round three is slightly superior compared to the other rounds, but close enough to the average that it provides valuable insight.

	Recht	Pflicht	Einw rh	RF vAw	Verfahren	Verweisung	Fortführungsnorm	Definition
Recht	27	1	0	1	1	0	0	0
Pflicht	2	19	2	0	1	2	0	0
Einw rh	1	2	28	0	0	0	0	0
RF vAw	2	1	0	7	0	0	0	0
Verfahren	2	3	0	0	4	1	0	0
Verweisung	0	0	1	1	0	5	0	0
Fortführungsnorm	0	1	1	0	0	1	5	0
Definition	0	1	0	0	0	0	0	3

Table 20: Confusion Matrix Using LR having labeled ≈52% of the Training Data in the NX Experiment
Source: Own illustration

The predictions are illustrated in the columns while the rows reflect the correct label. The table shows that though the classes *Fortführungsnorm* and *Definition* have only low support in the dataset, both have very high *precision* as well as good *recall*. This high level of *Definition*-detection is in disagreement with what has been discussed above for this class. The reason is that, at this point, the classifier was very well trained with the existing *Definition* classes in the training dataset. Thereafter, the frequent occurrences of the other classes resulted in overfitting of the classifier to the other classes. Thus, at the end, only one *Definition* is found (see Table 21). Both confusion matrixes show that the class *Verfahren* is generally difficult to distinguish from the others. Nevertheless, the outcomes of the matrix above are marginally better. The *precision* of the class *Recht*, with 72% at this point, is also higher than at the end. However, due to overfitting, the *recall* of this class increases towards the end. The result is that seven instances of the class *Pflicht* are labeled as a *Recht* at the end. The results of the classes *Einw rh* and *RF vAw* remaining relatively constant, apart from minor deteriorations.

	Recht	Pflicht	Einw rh	RF vAw	Verfahren	Verweisung	Fortführungsnorm	Definition
Recht	28	0	0	0	2	0	0	0
Pflicht	7	16	1	1	1	0	0	0
Einw rh	2	2	27	0	0	0	0	0
RF vAw	2	2	0	6	0	0	0	0
Verfahren	2	5	0	0	3	0	0	0
Verweisung	0	0	1	1	0	5	0	0
Fortführungsnorm	0	0	2	0	0	1	5	0
Definition	1	1	0	0	1	0	0	1

Table 21: Final Confusion Matrix using LR in the NC Experiment
Source: Own illustration

It can be concluded that, due to increased granularity, the classification of legal norms is much more difficult. However, despite this complexity, a notable *accuracy* of almost 80% can be achieved using AL procedures. The five classes *Recht*, *Einw rh*, *Verweisung*, *Fortführungsnorm* and *Definition* are recognized best, attaining an F_1 of above 70%.

6.2 Requirement Verification

Below, the extent of the achievement of the objectives and requirements defined in chapter 4.2 are presented. The defined requirements can be said to be either completely fulfilled (✓), partially fulfilled, (☑) or unfulfilled (☒).

ID	Requirement	Priority	Fulfillment
FR01	Perform Legal Text Classification	5	✓
FR02	Import of Legal Texts	5	✓
FR03	Utilize Lexia's Existing Environment	4	✓
FR04	Multiclass Classification	5	✓
FR05	Use Spark's ML Pipeline Concept	4	✓
FR06	Use of AL Pipelines	5	✓
FR07	Configure AL Pipelines	5	✓
FR08	Train Classifier Using AL Pipeline	5	✓
FR09	Save Classifier/AL Pipeline Model	5	✓
FR10	Predict with AL Pipeline	2	✓
FR11	Evaluate AL Pipeline	5	✓
FR12	Transparency	3	✓
FR13	Export of Evaluation Results	5	✓
NFR01	Simple User Interface	4	✓
NFR02	Maintainability of Software Architecture	3	✓
NFR03	Extensibility of Software Architecture	3	✓
NFR04	Data Exchange	3	✓
NFR05	User Spark MLlib as ML framework	5	✓
NFR06	Simulation of AL	5	✓

Table 22: Verification of LexML's Requirements

Table 22 provides an overview of the defined requirements and their degree of fulfillment. As indicated by the number of ticks, all the requirements were successfully implemented. In LexML, an environment for conducting multiclass legal TC (FR01 and FR04) with Spark MLlib (NFR05) was created, utilizing the existing Lexia environment (FR02 and FR03) and a REST API (NFR04). A Spark ML Pipeline containing various steps like a tokenizer, a stop-word remover and a classifier was employed (FR05) and the obtained model can be persisted (FR09), in order to apply it for later prediction and evaluation purposes (FR10 and FR11). The multiclass TC process using MLlib's ML Pipeline was implemented using some kind of AL pipeline in LexML. In doing so, the three use cases of the (1) creation of AL pipelines (FR06), (2) configuration of AL pipelines (FR07), and (3) processing of AL pipelines (FR08, FR10 and FR11) have been fulfilled. To provide a simple UI for users, these specific applications are built on individual pages in Lexia (NFR01). Furthermore, the user has not only the possibility to

export and analyze the evaluation and learning results (FR13), but also obtains feedback about it during the labeling process (e.g. confidence about prediction) (FR12). As it is possible, for instance, to use only the ML component when not conducting AL without significant changes to the code, the maintainability of LexML is ensured (NFR02). Additionally, the use of interfaces provides a good starting point to complement the functionality, for example, by adding more classifiers or query strategies (NFR03).

7 Discussion

The possibility of the automatic classification of legal textual content is driven by the impact of the rapid development of digitization. This trend contributes to the need of improving the current way of working with legal data. A vast amount of legal content is created daily, which must be properly classified so that it can be located, retrieved and worked with again at a later point in time.

In the age of digitization, content is usually stored in a computational medium. While the uploading time and data storage of the legal content is no obstacle, the subsequent manual classification process is very time consuming and expensive, mostly due to the number of legal documents produced.

However, even in cases where the documents are classified correctly and can be recovered easily, reading and understanding the content of a legal document remains an elaborate process, especially as such content has tended to become more complex in recent times. To support the search for relevant details, the document content (norms) can be assigned to certain classes. Having a document classified by its legal norms allows users to rapidly filter out irrelevant subjects.

Several computer-aided support approaches exist for both methods of classification. One common way to do this is to use rule-based text-annotation software to identify specific linguistic patterns. Due to improved technical capacities, machine learning (ML) techniques have gained more attention in the context of (legal) text classification (TC). In this work, active (machine) learning (AL), a promising semi-supervised machine learning approach was introduced and applied to German legal data. Thereby, the research questions discussed below were investigated and answered.

7.1 Reflection on the Research Questions

In the paragraphs that follow, a brief summary of the results of this work, referring to the individual research questions defined in Chapter 1.2, is provided.

I. What are common concepts, strategies and technologies used for text classification?

TC is a use case for ML with the objective of assigning textual content to predefined categories. Typically, a TC process consists of the three principal steps: (1) preprocessing, including tokenizing, stop-word removal, and occasionally stemming; (2) transforming the textual content into a feature vector (e.g. TF format), and eventually narrowing the feature space by applying FS methods (e.g. TF-IDF format); and (3) using this feature vector to train a classifier. For the latter, several generative and discriminative classifiers exist. Each of them has certain advantages and disadvantages in terms of *accuracy*, capabilities of dealing with unbalanced data, and speed of learning, among other issues. The classifiers NB, SVM and MLP were introduced in greater detail. In theory, and often in practice, it is claimed that LR, MLP and especially SVMs achieve a higher *accuracy* in TC than does NB. However, NB has other

benefits, such as its speed of learning, and its tolerance against noise. In addition, practical experiments have shown that by using NB, a suitably high *accuracy* can be achieved. Hence, there is not one classifier that is always markedly better than the others. Their suitability to a task depends on numerous factors that are difficult to estimate in advance. To evaluate the performance of these classifiers, common evaluation measures like *accuracy*, F_1, *precision*, and *recall* were introduced in this work.

It was also determined that a variety of ML frameworks exist that enable the classification of textual content. The wide choice of possible frameworks makes the selection of a suitable one difficult. Thus, after a preselection, five common ML frameworks (Weka, scikit-learn, Mallet, Mahout, and MLlib) were compared in terms of suitability as implemented classifiers for TC, means of representing the textual content, general applicability for TC, community support, and quality and extent of available documentation. It was concluded that Spark MLlib is a promising framework, and this was used for the development of the AL prototype (microservice).

II. How can (active) machine learning support the classification of legal documents and their content (norms)?

TC was originally effected using traditional supervised ML, which is expensive and time consuming as it requires many labeled instances. AL is another subfield of (semi-)supervised ML that provides a more efficient way of conducting TC. AL is an interactive and iterative process during which, based on small amount of labeled data (a seed set), a classifier is trained. This trained classifier is used to make predictions regarding the remaining unlabeled dataset. From these predictions, the instances that aid the classifier the most in distinguishing between the classes are selected to be labeled next. A specific algorithm (a query strategy) is used to find those instances. They are then labeled and added to the training dataset. This growing labeled training set is used to improve the model and the process is repeated. Hence, by labeling only "informative" instances, it is more likely that the classifier will perform well while using fewer labeled instances to achieve its aim, thereby minimizing the cost for the classification process.

To find those most informative instances, several query strategies have been proposed. Many of the strategies utilize the output scores of the classifiers (e.g. posterior probability) to calculate these uncertain measures. The most prominent versions refer either to uncertainty sampling (US) or to the more elaborated query by committee (QBC) methods. While the former uses only one classifier model, the latter creates a committee of classifiers with the intention of covering a larger area of the version space. Hence, QBC strategies are more expensive than US methods.

Further issues that must be considered when applying AL are the number of instances that should be queried every round, the choice of the classifier, and when the learning process should be stopped (by means of a stopping criterion). For all these matters, several common theoretical and practical solutions have been discussed in this work. Additionally, learning curves, a way to visualize the progress of the classifier, were introduced and discussed.

How helpful AL can be in the legal domain is assessed later. Although the taxonomy of legal textual content can be complex, legal texts have the advantage that they are written in extremely formal language. To provide an overview of the legal classification task, common legal

document types and a typical taxonomy of norms were introduced. In this context, results of current studies referring to legal TC using ML techniques were presented.

III. What form does the concept and design of an active machine learning service take?

Based on the literature review and the objectives of this work, a set of requirements were defined to develop a suitable AL-prototype named LexML. LexML was implemented as an independent ML microservice in a very flexible and extendible way so that it can handle all kind of (legal) textual content using Spark MLlib as ML framework. LexML has a REST API to initiate necessary tasks, such as the importing of the textual data, the creation and configuration of an AL Pipeline, the iterative training and improvement of the ML pipeline (classifier), and the evaluation of the trained ML pipeline (classifier). To assess the learning levels of the classifier, suitable evaluation measures such as the F_1 or *accuracy* were implemented. The evaluation results can be exported as a xlsx file so that the learning can be readily analyzed.

To provide an extensive ML service, the classifiers MLP and NB were introduced in detail, and additionally LR was implemented and can be selected by the user. Furthermore, two query strategies of each of the US (vote entropy, margin sampling) and QBC (vote entropy, soft vote entropy) methods were implemented. As MLP does not provide a posterior probability, only the QBC vote-entropy strategy is provided for selection.

LexML does not have an own UI. Instead, it was integrated into Lexia, an already existing data science environment for the analysis of German legal texts. The communication between Lexia and LexML is managed by the REST API. LexML employs Lexia's available functionality, for example, its importers and sentence splitter, in order to receive the respective sentences for norm classification.

IV. How well does the active machine learning service perform in the classifying legal documents and their content (norms)?

To evaluate LexML and AL for the purpose of legal TC, the three questions that follow have been discussed for both document classification (DC) and norm classification (NC). For both experiments, each combination of classifier and query strategy was executed five times to get a significant and comparable result.

1) Which implemented classifier performs best?

As discussed in the theoretical part of this study, it was determined that there is not a single best classifier for all use cases. In the DC experiment, documents were classified into one of three classes by using approximately the first 400 words of the document content. In this case, NB and MLP perform most effectively, having a final *accuracy* of 95%, while LR achieves only 92%. In contrast, for the more complex NC experiment, with eight classes with a very unbalanced distribution, LR is undoubtedly the best performing classifier, achieving a final *accuracy* of 72%, while NB obtains only 55%. Apart from the fact that different data used, the only difference between the two experiments was that stop-words were removed in the DC experiment.

An explanation for the varying performance of NB may be, first, NB's assumption of strong feature independency and secondly, its low tolerance to redundant features. Hence, as in the DC experiment frequent stop-words were removed, the number of unnecessary redundant features was reduced. Further, when classifying whole documents, word order is generally not relevant; rather, whether a certain word occurs in the text at all is of major importance. Thus, the independence assumption of NB should also be no problem. Consequently, despite its simplicity, NB achieved good results in the first experiment.

In contrast, in the NC experiment, stop words were not removed, and certain word sequences are important for recognizing the class of a norm. Hence, both weaknesses take effect and NB performed worse than the other classifiers.

2) Is AL superior to PL?

Both classification experiments have shown that AL is markedly superior to PL, similar to traditional supervised learning, where the instances are queried randomly each round. The use of AL increased not only the speed of learning, but also resulted in a higher maximum *accuracy* obtained during the classification process. In both experiments, the average *accuracy* was after a short "discovery phase" between 5%-10% higher once 20%-70% of the instances had been labeled, compared to the PL (random) approach. Additionally, the obtained (intermediate) *accuracy* was often higher than when all instances had been labeled (traditional supervised learning). With an increasing number of AL rounds, the problem of overfitting become observable, so that after a certain proportion of instances have been labeled (70%-95%), both approaches align to the same final *accuracy*.

When viewing the results of the individual learning rounds for a specific combination, the importance of having a "high quality seed set" becomes clear. The term quality refers, on the one hand, to having a balanced distribution of instances to prevent the missed class effect. On the other hand, it is useful to have meaningful instances that support the classifier in distinguishing between classes and to find the truly most informative ones in the next rounds.

As the seed set used in the experiments was composed of randomly queried instances, the learning differs especially in the initial phases. Hence, it was after a discovery of the version space (discovery phase) that AL was significantly superior to PL.

3) Which implemented query strategy performs best?

Although it is claimed that the more elaborated QBC strategies are superior to US methods, no significant difference could be observed. With only slight fluctuations, all four query strategies worked similarly well and were superior to PL. A reason for the lack of superiority of the QBC strategies could be that the attention was on the committee's disagreement measures rather than on committee creation. No specific algorithm discussed in the theoretical section of this work to create an optimal committee was implemented. Instead, only the composition of the trainings data was adapted for each committee.

4) Which norms and documents are best recognized by the classifier(s)?

A simple summary of the classes found most frequently is not reasonable in this case. Instead, some general results that were noticeable when this question was discussed are described.

In general, it is claimed that traditional supervised learning has problems with unbalanced data so that classes that occur infrequently are rarely classified correctly. This study has shown that under suitable conditions, AL can handle unbalanced data very well. In the NC experiment, the two classes with the lowest support (*Definition*, *Fortführungsnorm*) temporarily had the highest F_1 score. This advantage disappeared when the frequent classes led to overfitting of the classifier.

A second point addresses the reusability of a trained model for which the results of the document classification experiment might provide some insights. Although two of the three classes (judgment and law) obtain a very high F_1 very soon, the class of generic legal document never exceeds never a certain threshold (≈90%). The supposed reason is that this generic class covers too many different document types, only some of which are in the test dataset. Hence, the classifier cannot learn that this kind of document belongs to the generic document class. This outcome can be transferred to the concept of the reusability of a model. It means that a model can only be reused if the textual content is very similar to the one the model it was trained with. For instance, the model trained with the law of tenancy of the German civil code could hardly be used to classify norms from other civil code sections.

7.2 Conclusion

In this work, an independent service for conducting (legal) multiclass TC using AL was designed and implemented utilizing Apache Spark MLlib as ML framework. Common concepts and strategies found in the literature study form the basis for the developed requirements and solutions. Three classifiers (NB, LR and MLP) as well as four query strategies (US vote entropy, US margin sampling, QBC vote entropy, and QBC soft-vote entropy), and other configuration options create a comprehensive microservice to perform legal TC. To provide the groundwork for further development, LexML is built in a very maintainable and extendible way so that further functionality can be easily added (e.g. additional classifiers, query strategies). Additionally, LexML has an integrated evaluator using common TC measures such as *accuracy* and F_1 to evaluate the trained model and pursue the learning progress. These measures can be exported to an xlsx file so that they can be analyzed further in a proper manner.

While LexML is an independent ML component, within the scope of this research it made use of Lexia, an existing data-science environment for the analysis of German legal. Lexia provides functionality such as available importers for legal documents, possibilities to split and export the sentences of an imported law using Apache ruta, and an uncomplicated UI [168, 169]. This existing UI was complemented by a UI for conducting AL with the legal texts imported in Lexia. Nevertheless, LexML is an independent microservice that can be used for other multiclass TC problems and AL without much adaption.

Within the scope of this work, two TC experiments using German legal content were conducted to evaluate (1) the potential of AL compared to PL (similar to traditional supervised ML), and (2) the quality of legal TC when using ML/AL techniques. The objective of the first experiment was to classify three different kinds of documents as one of three possible classes (DC). In a more complex second experiment, norms of the law of tenancy section of the German civil code were classified as one of eight possible classes (NC).

(1) Both experiments show that AL is clearly superior to PL (random approach). The use of AL increases not only the speed of learning, but also results in a higher maximum *accuracy* obtained during the classification process. Classifying too many instances frequently results in overfitting, and a lower final *accuracy*. Consequently, no single best classifier for all use cases exists. While NB and MLP were the superior classifiers in the document classification experiment, LR was significantly better than both other classifiers in the norm classification experiment. Although QBC strategies purportedly work better due to their greater coverage of the version space, no significant difference could have been found between the US and QBC query strategies implemented. However, LexML still has the potential to improve the committee creation process.

(2) As discussed in literature review, both experiments show that the quality of legal TC depends strongly on numerous influencing factors that cannot always be detected in advance. Nevertheless, in the DC experiment, NB and MLP achieve a very high average *accuracy* of 96%. In contrast, for the NC experiment, a maximum average *accuracy* of only 74% was attained with LR. However, by using AL, an *accuracy* of almost 80% was temporarily achieved in a few rounds. This is a result comparable to other studies using a

similar experimental setup (see Chapter 2.3.3), especially due to the complex setup that included an unbalanced dataset with some instances having support of less than 5%.

The LexML artifact and the outcomes of this work are an addition to the knowledge base and provide a suitable starting point for future research in the fields of AL, TC in general, and especially TC of German legal texts.

7.3 Limitations and Future Work

Although this work provides a good starting point for future work, some limitations described below must be kept in mind.

Even though each possible combination of an experiment was conducted five times in order to obtain a significant result, and even though the resulting learning curve is certainly a meaningful indicator for the superiority of AL, the experiments require further replication to attain a statistically significant value. This is reinforced by the fact that the classifiers used for AL are black-box classifiers that do not provide any appropriate explanation of how the classification results are obtained.

Furthermore, the power of MLlib was not completely utilized as primarily only one default setting was used for the evaluation of the experiments. As MLlib provides many levers to adjust classifier settings, FS techniques (e.g. TF-IDF), and the like, further experiments using different configurations should be conducted and evaluated. For instance, no specific FS method was evaluated within the scope of this study. Nevertheless, as MLlib was built in a maintainable and extendible way, there is much opportunity for future work, such as integrating additional query strategies, classifiers, further pre-processing steps, and so on.

It was asserted that a "high quality seed set" leads to a faster and better learning. As the seed set in this study was created randomly for each experiment, a certain "discovery phase" was needed until AL became superior to PL. One component of future work could be to apply unsupervised learning methods before starting the learning process, in order to identify clusters in the dataset and allocate a number of instances from each cluster to the seed set. Further options are to integrate in LexML exploration-based strategies that search for regions in the version space that the algorithm would classify incorrectly [101].

A known problem of ML that has also partially occurred in this study concerns the reusability of a trained model. Once a classifier has been trained with a certain kind of content or domain, the resulting model only works well with content that is similar.

Additionally, in the document classification experiment only three different classes are used. As shown in Chapter 2.3.2, the taxonomy of legal documents is usually more complex and more accurate.

Nevertheless, using AL for the classification of legal content is a promising approach to supporting the work in the legal domain and must be investigated further. An interesting approach could be to combine the AL with other classification techniques, such as the use of rules applying a hybrid concept. The idea is that the outcome of the rules provides the input of the training set (the seed set) to conduct AL. In this manner, the discovery problem mentioned could be resolved.

Appendix

A Document Classification

This appendix describes further evaluation measures obtained in DC.

A.1 Examples for Averaging the Learning Rounds

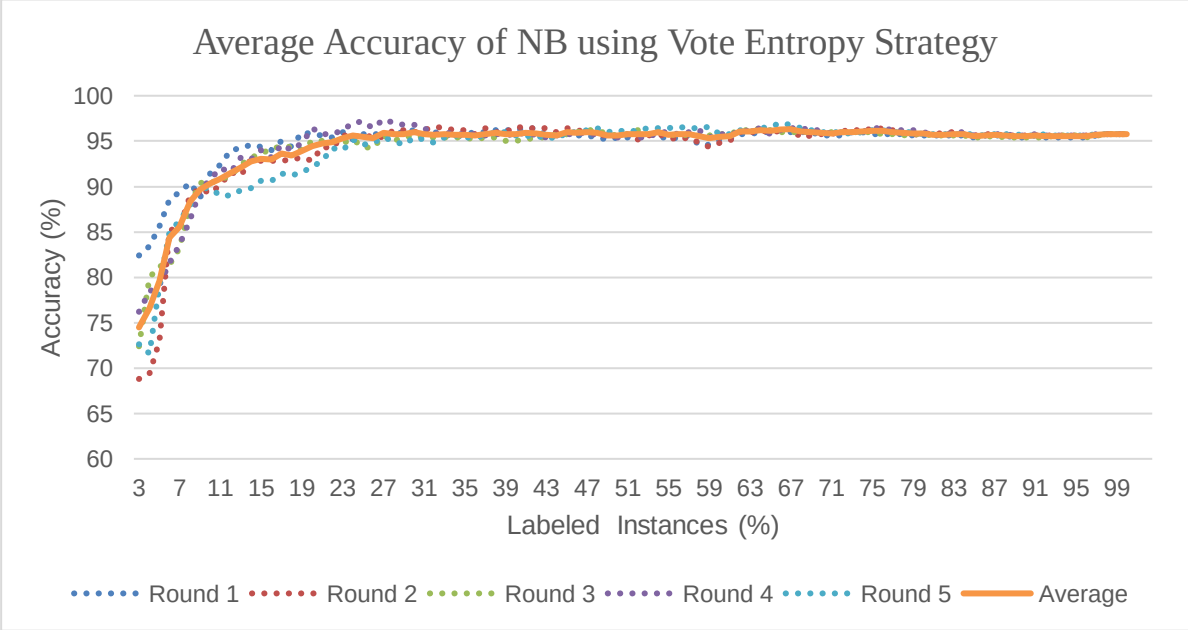


Figure 48: Average Accuracy of NB Using Vote Entropy Strategy
Source: Own illustration

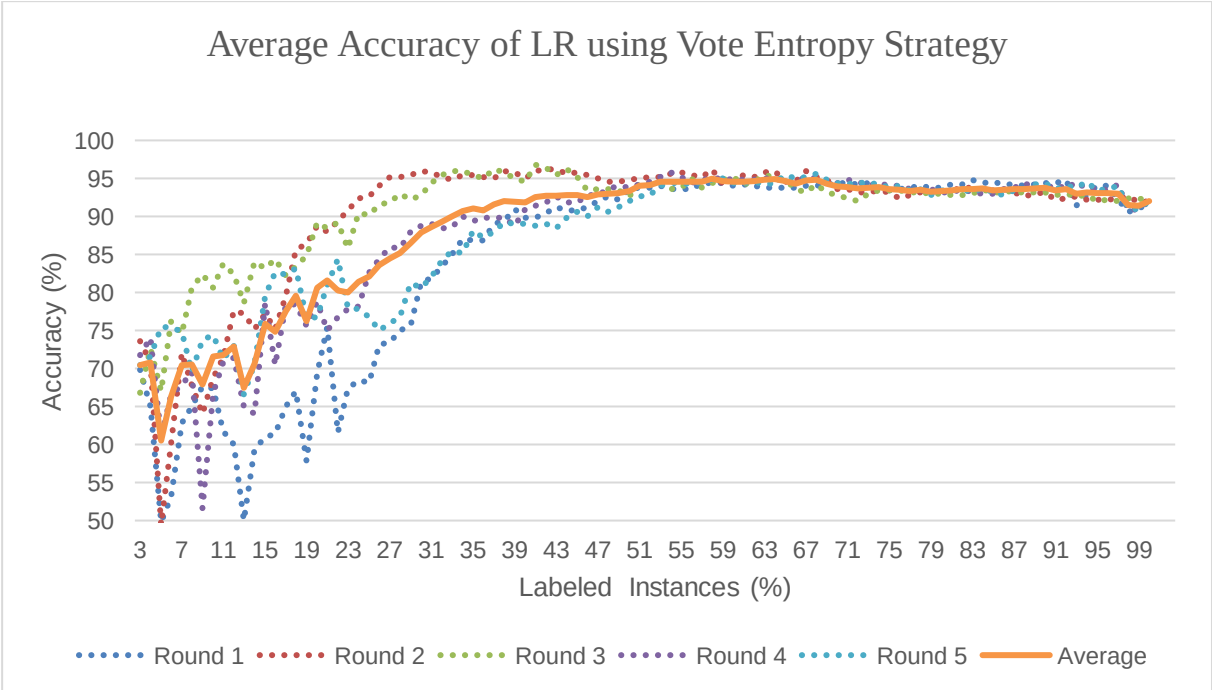


Figure 49: Average Accuracy of LR Using Vote Entropy Strategy
Source: Own illustration

When using NB, the distribution of the curve progressions for all five learning rounds is nearly the same; however, the ones from LR exhibit greater difference. As the query strategy works in two of the five rounds well, this is probably mainly caused by a "bad" seed set that misleads the classifier. Although the variations between the individual learning rounds using LR and another query strategy are not as strong as in the case presented, it becomes generally evident that NB operates in a more stable manner than LR in the context of DC. In addition, MLP, which is not shown here, performs better than LR, but slightly worse than NB.

A.2 Overall evaluation of DC

As with Figure 35, Figure 50, Figure 51 and Figure 52 compare AL with PL, similar to traditional supervised learning, in which the labeling sequence is not considered and instances are queried randomly. Again, the average outcome of all query-strategy possibilities for one classifier is averaged.

On this occasion, instead of using *accuracy* as evaluation measure, the weighted metrics F_1, *precision* and *recall* are utilized. In this case, all progression curves exhibit a similar course analog to the one for *accuracy*. Hence, the statements made in Chapter 6.1.2 for the question "2) Is AL Superior to PL" are confirmed by these evaluation measures.

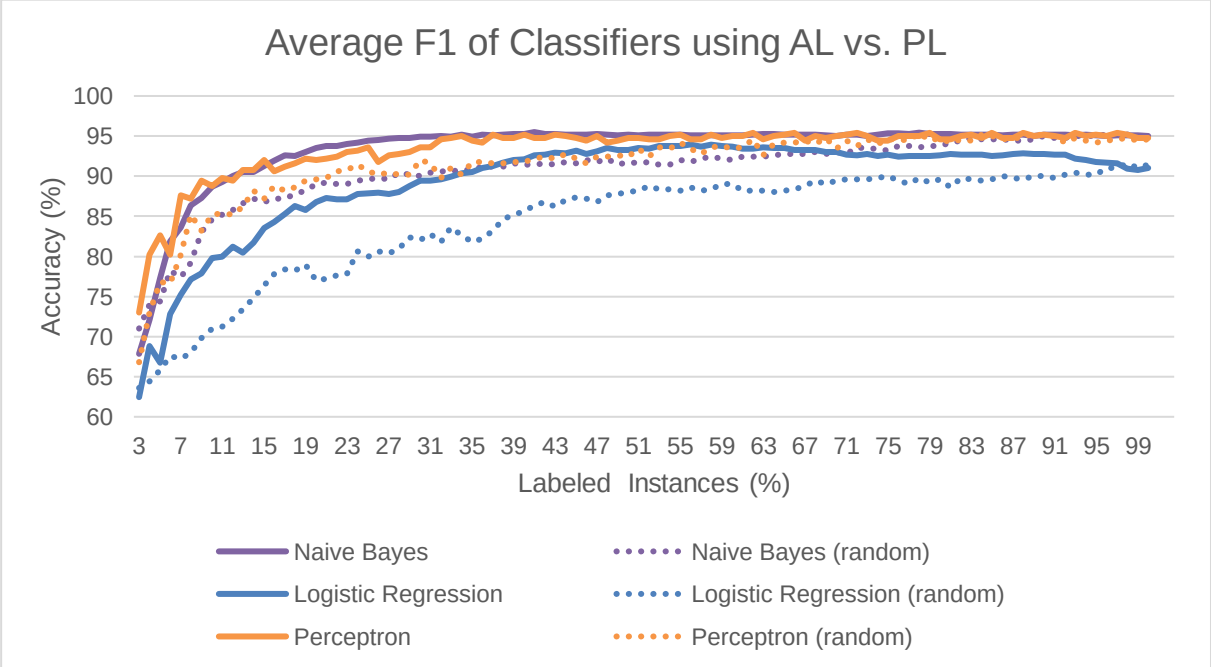


Figure 50: Average F1 of Classifiers Using AL vs. PL
Source: Own illustration

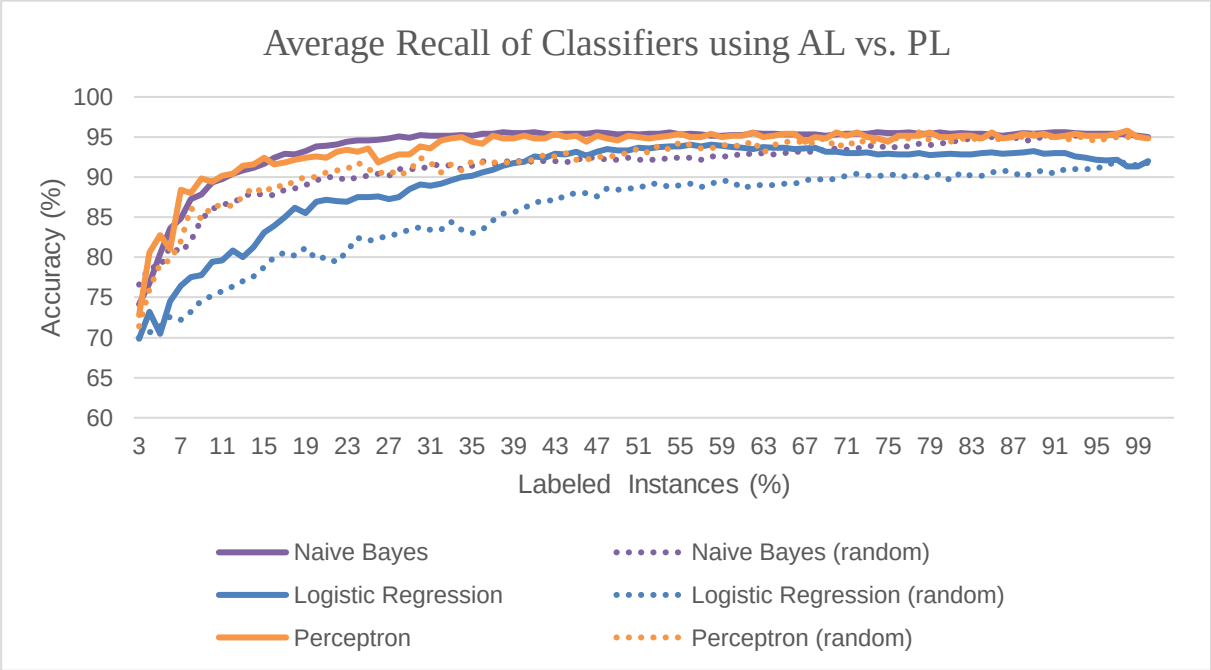


Figure 51: Average Recall of Classifiers Using AL vs. PL
Source: own illustration

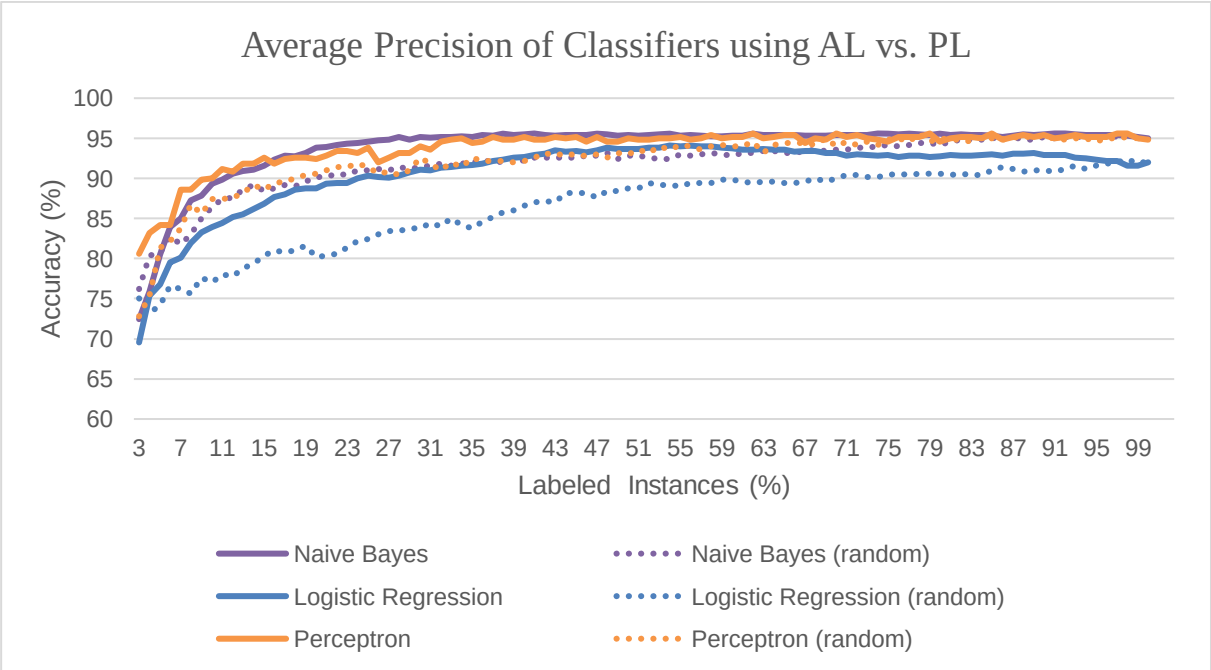


Figure 52: Average Precision of Classifiers Using AL vs. PL
Source: Own illustration

A.3 F_1, Precision and Recall per Class for DC

As the evaluation results have shown that all query strategies work equally well, the four results obtained from the combinations of LR and NB when conducting AL have been consolidated in the following illustrations.

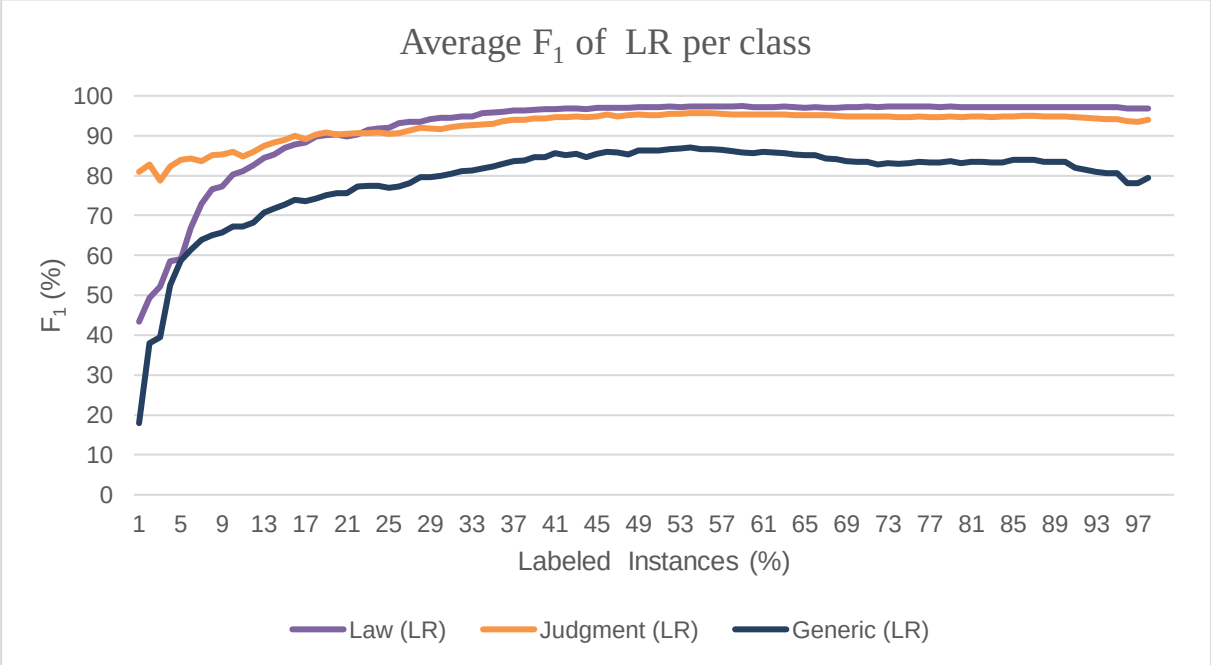


Figure 53: Average F_1 of LR per Class
Source: Own illustration

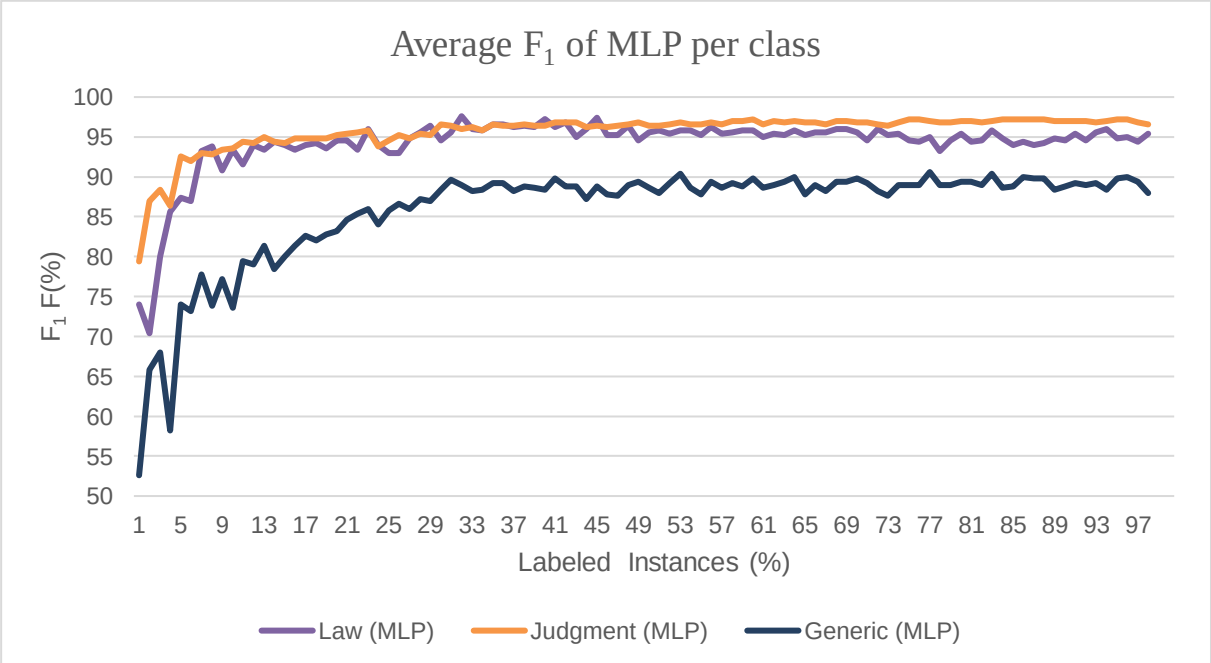


Figure 54: Average F_1 of MLP per Class
Source: Own illustration

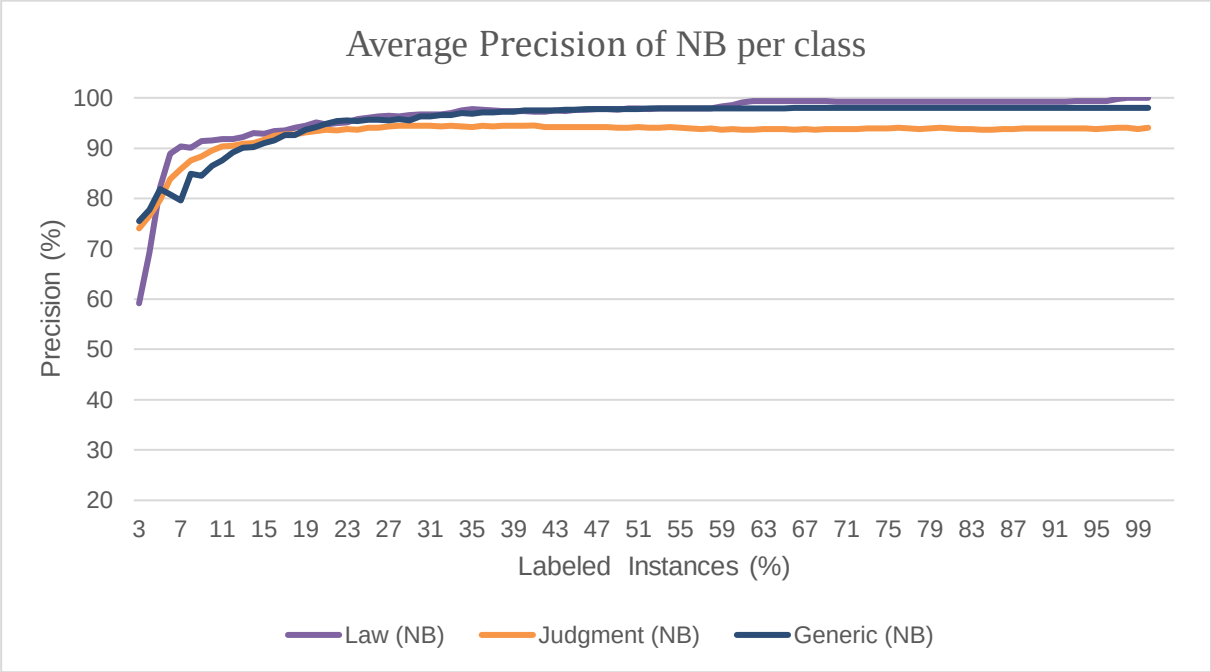


Figure 55: Average Precision of NB per Class
Source: Own illustration

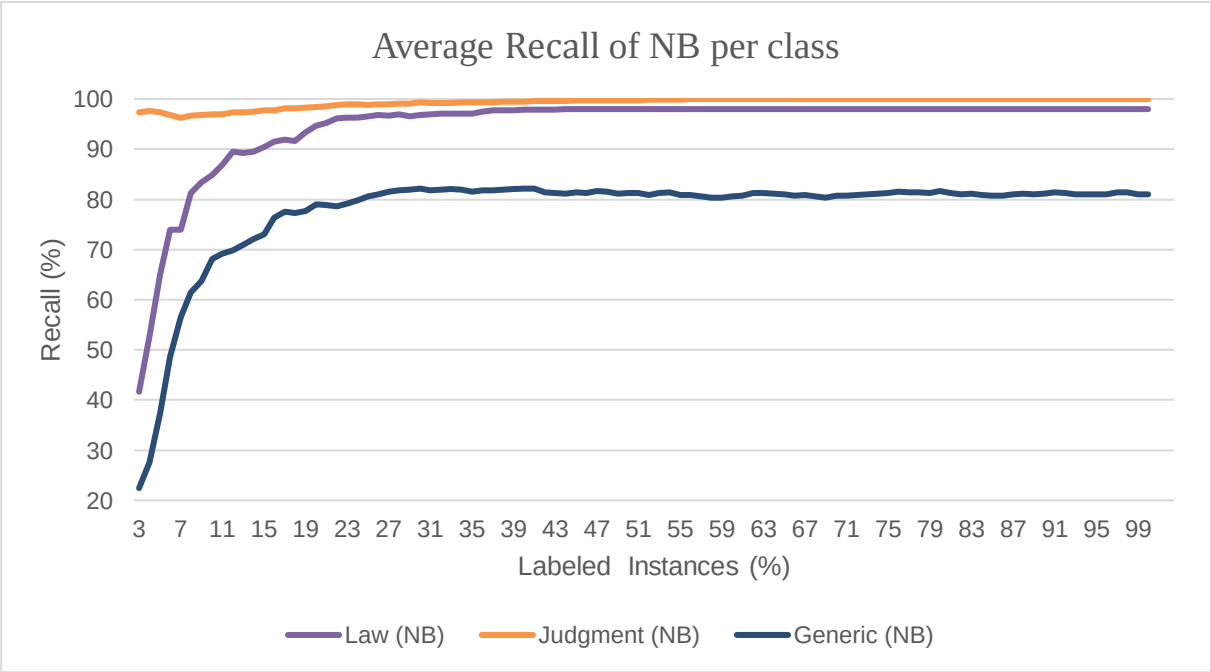


Figure 56: Average Recall of NB per Class
Source: Own illustration

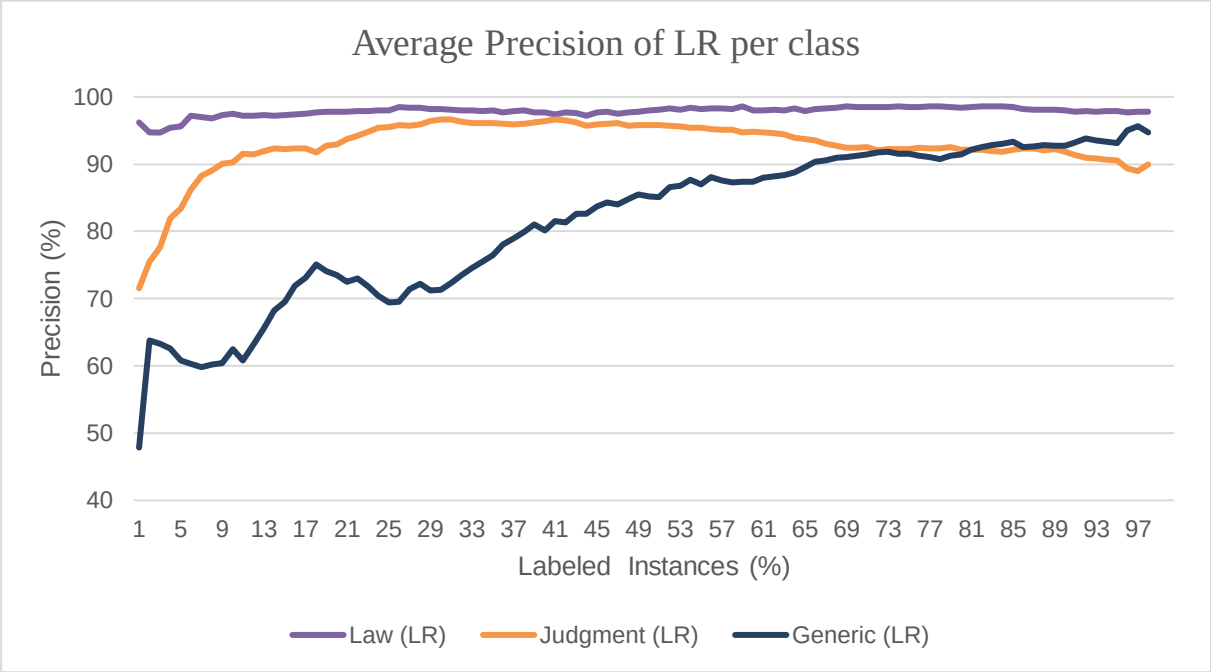


Figure 57: Average Precision of LR per Class
Source: Own illustration

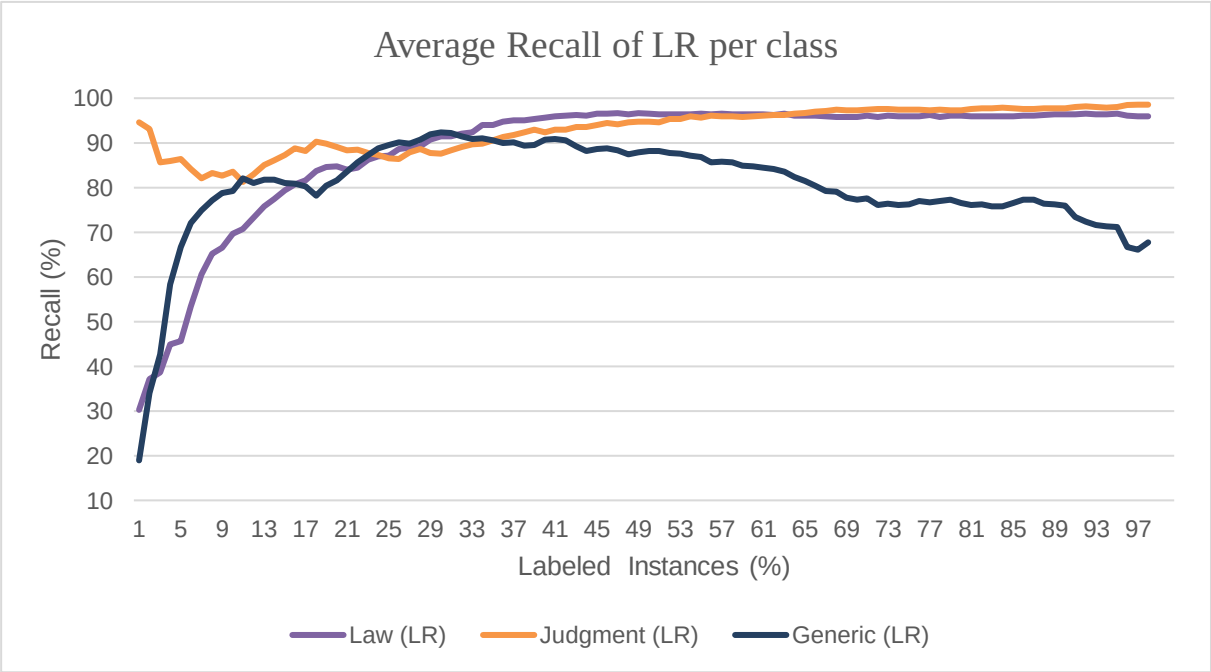


Figure 58: Average Recall of LR per Class
Source: Own illustration

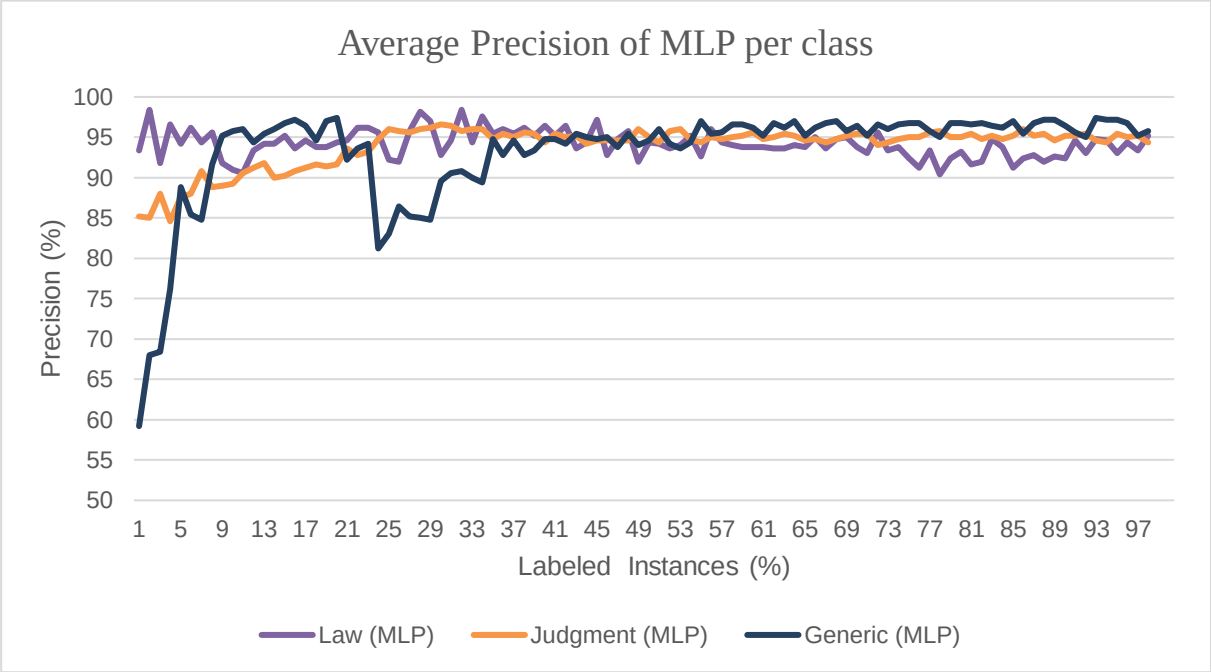


Figure 59: Average Precision of MLP per Class
Source: Own illustration

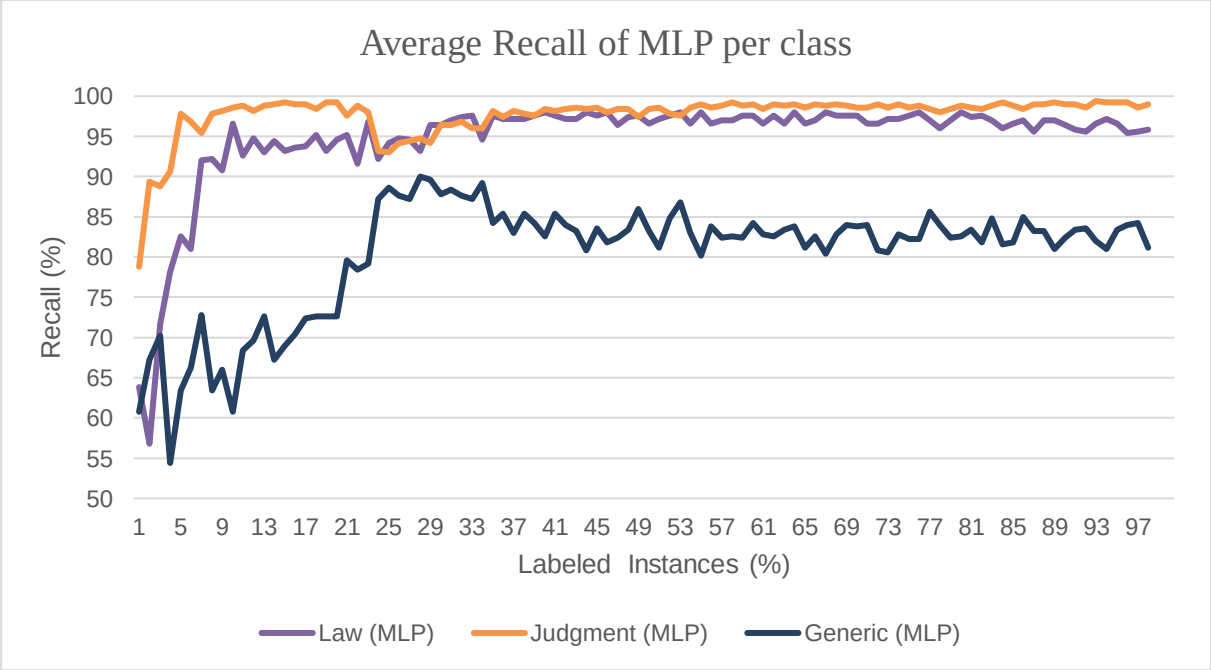
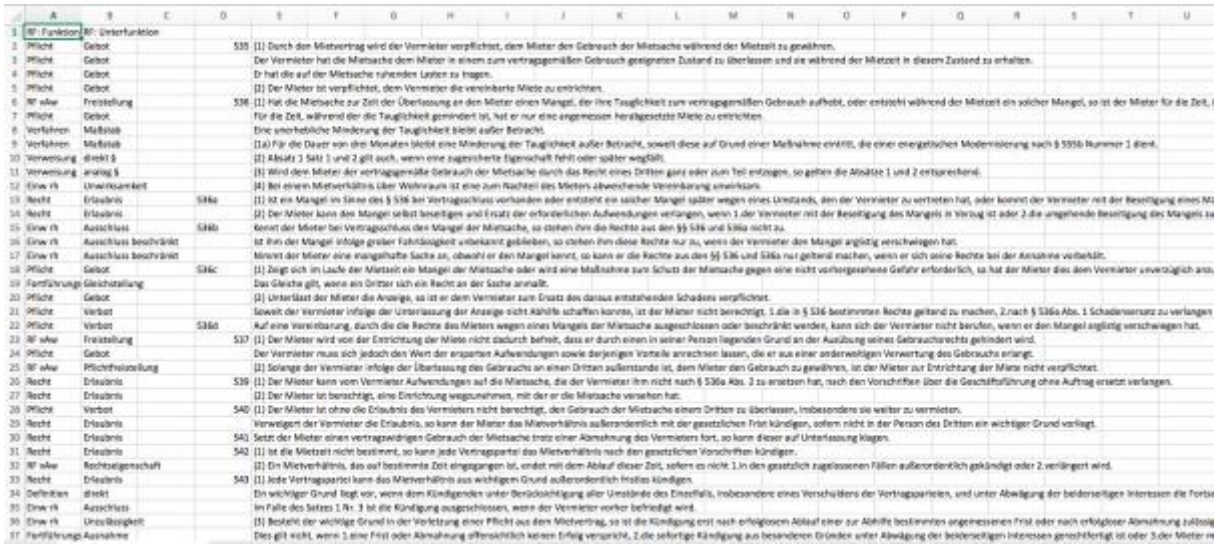


Figure 60: Average Recall of MLP per Class
Source: Own illustration

B Norm Classification

This component of the appendix provides further insights about NC.

B.1 Prepared csv-file for NC



	A	B	C	D	E
1	RF: Funktion	RF: Unterfunktion			
2	Pflicht	Gebot		535	(1) Durch den Mietvertrag wird der Vermieter verpflichtet, dem Mieter den Gebrauch der Mietsache während der Mietzeit zu gewähren.
3	Pflicht	Gebot			Der Vermieter hat die Mietsache dem Mieter in einem zum vertragsgemäßen Gebrauch geeigneten Zustand zu überlassen und sie während der Mietzeit in diesem Zustand zu erhalten.
4	Pflicht	Gebot			Er hat die auf der Mietsache ruhenden Lasten zu tragen.
5	Pflicht	Gebot			(2) Der Mieter ist verpflichtet, dem Vermieter die vereinbarte Miete zu entrichten.
6	RF eAw	Freistellung		536	(1) Hat die Mietsache zur Zeit der Überlassung an den Mieter einen Mangel, der ihre Tauglichkeit zum vertragsgemäßen Gebrauch aufhebt, oder entsteht während der Mietzeit ein solcher Mangel, so ist der Mieter für die Zeit,
7	Pflicht	Gebot			für die Zeit, während der die Tauglichkeit gemindert ist, hat er nur eine angemessen herabgesetzte Miete zu entrichten.
8	Verfahren	Maßstab			Eine unerhebliche Minderung der Tauglichkeit bleibt außer Betracht.
9	Verfahren	Maßstab			(1a) Für die Dauer von drei Monaten bleibt eine Minderung der Tauglichkeit außer Betracht, soweit diese auf Grund einer Maßnahme eintritt, die einer energetischen Modernisierung nach § 555b Nummer 1 dient.
10	Verweisung	direkt §			(2) Absatz 1 Satz 1 und 2 gilt auch, wenn eine zugesicherte Eigenschaft fehlt oder später wegfällt.
11	Verweisung	analog §			(3) Wird dem Mieter der vertragsgemäße Gebrauch der Mietsache durch das Recht eines Dritten ganz oder zum Teil entzogen, so gelten die Absätze 1 und 2 entsprechend.
12	Einw rh	Unwirksamkeit			(4) Bei einem Mietverhältnis über Wohnraum ist eine zum Nachteil des Mieters abweichende Vereinbarung unwirksam.
13	Recht	Erlaubnis		536a	(1) Ist ein Mangel im Sinne des § 536 bei Vertragsschluss vorhanden oder entsteht ein solcher Mangel später wegen eines Umstands, den der Vermieter zu vertreten hat, oder kommt der Vermieter mit der Beseitigung eines M
14	Recht	Erlaubnis			(2) Der Mieter kann den Mangel selbst beseitigen und Ersatz der erforderlichen Aufwendungen verlangen, wenn 1.der Vermieter mit der Beseitigung des Mangels in Verzug ist oder 2.die ungehende Beseitigung des Mangels z
15	Einw rh	Ausschluss		536b	Kennt der Mieter bei Vertragsschluss den Mangel der Mietsache, so stehen ihm die Rechte aus den §§ 536 und 536a nicht zu.
16	Einw rh	Ausschluss beschränkt			Ist ihm der Mangel infolge grober Fahrlässigkeit unbekannt geblieben, so stehen ihm diese Rechte nur zu, wenn der Vermieter den Mangel arglistig verschwiegen hat.
17	Einw rh	Ausschluss beschränkt			Mindert der Mieter eine mangelhafte Sache an, obwohl er den Mangel kennt, so kann er die Rechte aus den §§ 536 und 536a nur geltend machen, wenn er sich seine Rechte bei der Annahme vorbehält.
18	Pflicht	Gebot		536c	(1) Zeigt sich im Laufe der Mietzeit ein Mangel der Mietsache oder wird eine Maßnahme zum Schutz der Mietsache gegen eine nicht vorhergesehene Gefahr erforderlich, so hat der Mieter dies dem Vermieter unverzüglich anz
19	Fortführung	Gleichstellung			Das Gleiche gilt, wenn ein Dritter sich ein Recht an der Sache anmaßt.
20	Pflicht	Gebot			(2) Unterlässt der Mieter die Anzeige, so ist er dem Vermieter zum Ersatz des daraus entstehenden Schadens verpflichtet.
21	Pflicht	Verbot			Soweit der Vermieter infolge der Unterlassung der Anzeige nicht Abhilfe schaffen konnte, ist der Mieter nicht berechtigt. 1.die in § 536 bestimmten Rechte geltend zu machen, 2.nach § 536a Abs. 1 Schadensersatz zu verlangen
22	Pflicht	Verbot		536d	Auf eine Vereinbarung, durch die die Rechte des Mieters wegen eines Mangels der Mietsache ausgeschlossen oder beschränkt werden, kann sich der Vermieter nicht berufen, wenn er den Mangel arglistig verschwiegen hat.
23	RF eAw	Freistellung		537	(1) Der Mieter wird von der Entrichtung der Miete nicht dadurch befreit, dass er durch einen in seiner Person liegenden Grund an der Ausübung seines Gebrauchsrechts gehindert wird.
24	Pflicht	Gebot			Der Vermieter muss sich jedoch den Wert der ersparten Aufwendungen sowie derjenigen Vorteile anrechnen lassen, die er aus einer anderweitigen Verwertung des Gebrauchs erlangt.
25	RF eAw	Pflichtheiszellung			(2) Solange der Vermieter infolge der Überlassung des Gebrauchs an einen Dritten außerstande ist, dem Mieter den Gebrauch zu gewähren, ist der Mieter zur Entrichtung der Miete nicht verpflichtet.
26	Recht	Erlaubnis		539	(1) Der Mieter kann vom Vermieter Aufwendungen auf die Mietsache, die der Vermieter ihm nicht nach § 536a Abs. 2 zu ersetzen hat, nach den Vorschriften über die Geschäftsführung ohne Auftrag ersetzt verlangen.
27	Recht	Erlaubnis			(2) Der Mieter ist berechtigt, eine Einrichtung wegzunehmen, mit der er die Mietsache versehen hat.
28	Pflicht	Verbot		540	(1) Der Mieter ist ohne die Erlaubnis des Vermieters nicht berechtigt, den Gebrauch der Mietsache einem Dritten zu überlassen, insbesondere sie weiter zu vermieten.
29	Recht	Erlaubnis			Verweigert der Vermieter die Erlaubnis, so kann der Mieter das Mietverhältnis außerordentlich mit der gesetzlichen Frist kündigen, sofern nicht in der Person des Dritten ein wichtiger Grund vorliegt.
30	Recht	Erlaubnis		541	Setzt der Mieter einen vertragswidrigen Gebrauch der Mietsache trotz einer Abmahnung des Vermieters fort, so kann dieser auf Unterlassung klagen.
31	Recht	Erlaubnis		542	(1) Ist die Mietzeit nicht bestimmt, so kann jede Vertragspartei das Mietverhältnis nach den gesetzlichen Vorschriften kündigen.
32	RF eAw	Rechtseigenschaft			(2) Ein Mietverhältnis, das auf bestimmte Zeit eingegangen ist, endet mit dem Ablauf dieser Zeit, sofern es nicht 1.in den gesetzlich zugelassenen Fällen außerordentlich gekündigt oder 2.verlängert wird.
33	Recht	Erlaubnis		543	(1) Jede Vertragspartei kann das Mietverhältnis aus wichtigem Grund außerordentlich fristlos kündigen.
34	Definition	direkt			Ein wichtiger Grund liegt vor, wenn dem Kündigenden unter Berücksichtigung aller Umstände des Einzelfalls, insbesondere eines Verschuldens der Vertragsparteien, und unter Abwägung der beiderseitigen Interessen die Fortse
35	Einw rh	Ausschluss			Im Falle des Satzes 1 Nr. 3 ist die Kündigung ausgeschlossen, wenn der Vermieter vorher befriedigt wird.
36	Einw rh	Unzulässigkeit			(3) Besteht der wichtige Grund in der Verletzung einer Pflicht aus dem Mietvertrag, so ist die Kündigung erst nach erfolglosem Ablauf einer zur Abhilfe bestimmten angemessenen Frist oder nach erfolgloser Abmahnung zulässig
37	Fortführung	Ausnahme			Dies gilt nicht, wenn 1.eine Frist oder Abmahnung offensichtlich keinen Erfolg verspricht, 2.die sofortige Kündigung aus besonderen Gründen unter Abwägung der beiderseitigen Interessen gerechtfertigt ist oder 3.der Mieter m

Figure 61: Prepared csv file for the NC experiment
Source: Own illustration

Column A corresponds to the eight labels, while column E contains the textual content that was used for training the classifier.

B.2 Overall evaluation of NC

In a manner similar to Figure 41, the following illustrations compare AL with PL. The consolidated outcome is used for the weighted metrics F_1, *precision* and *recall* in these illustrations. All progression curves exhibit a very similar course analog to the one of *accuracy*. Hence, the statements made in the chapter 6.1.3 for the question "2) Is AL Superior to PL" are confirmed by these evaluation measures.

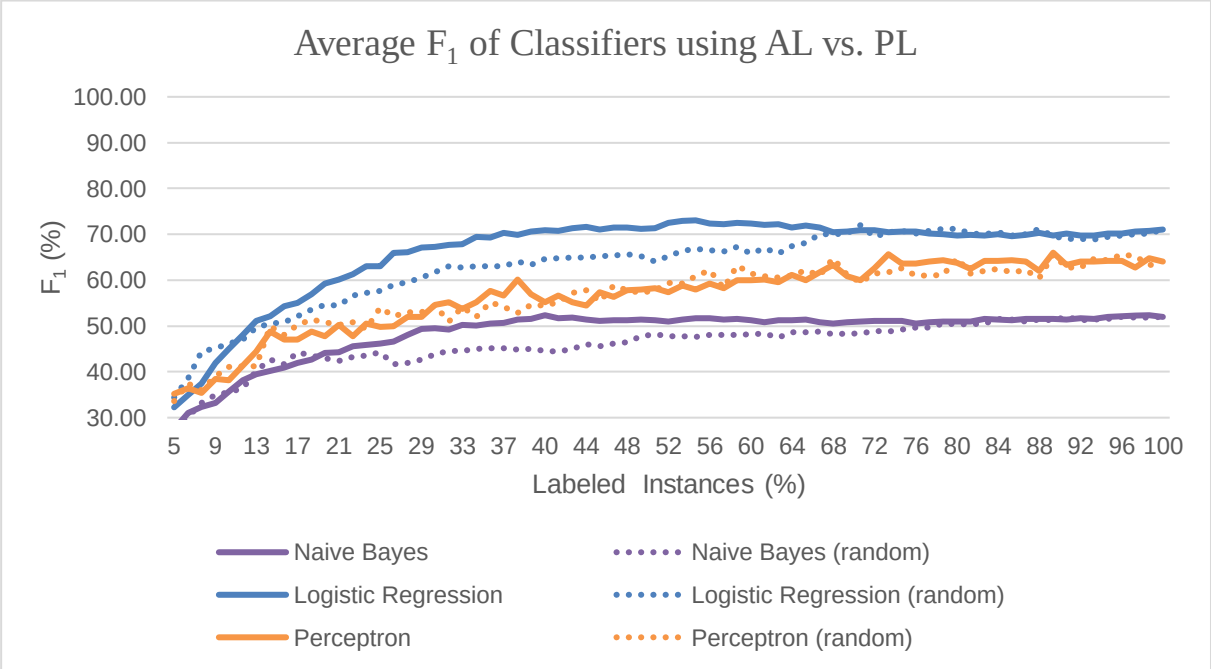


Figure 62: Average F_1 of Classifiers Using AL vs. PL
Source: Own illustration

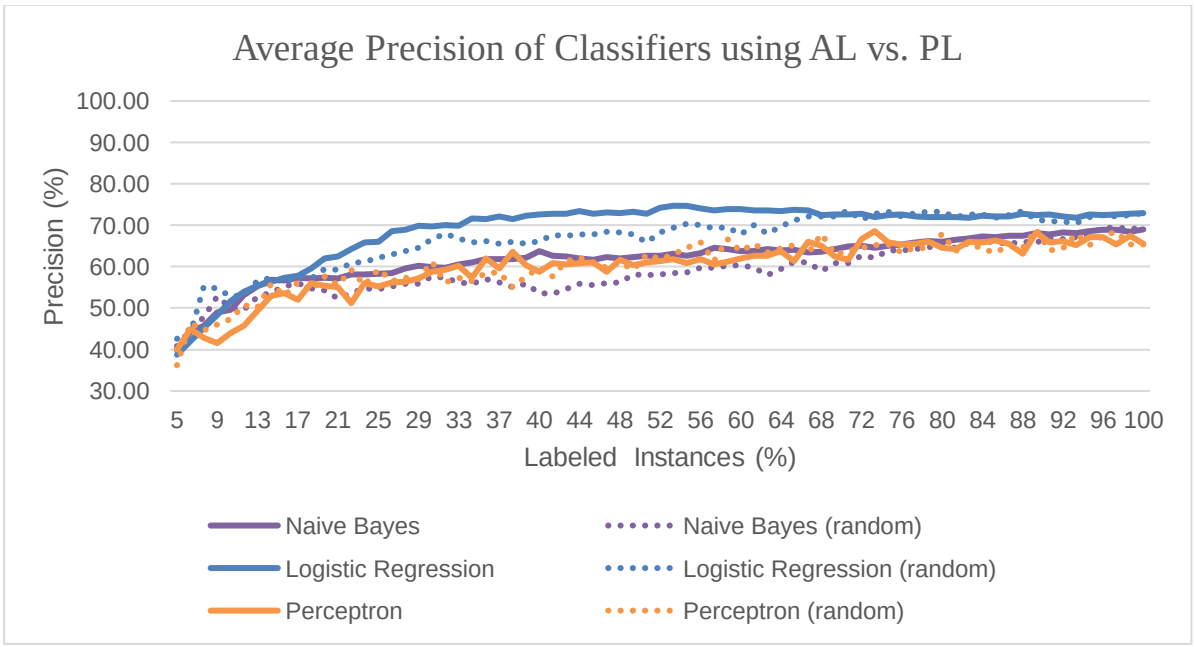


Figure 63: Average Precision of Classifiers Using AL vs. PL
Source: Own illustration

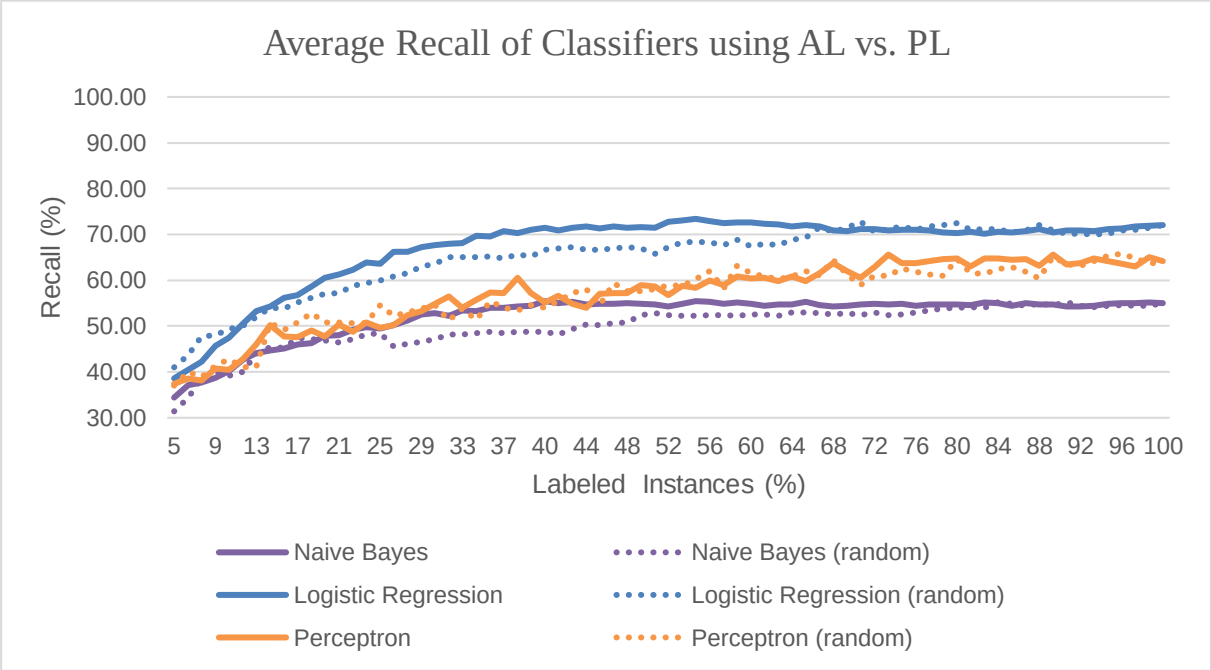


Figure 64: Average Recall of Classifiers Using AL vs. PL
Source: Own illustration

B.3 F_1, Precision and Recall per class for NC

Naïve Bayes:

It is noticeable that the classes *Fortführungsnorm* and *Definition* that temporarily obtain the highest F_1 when using LR have the lowest recognition, with an F_1 of more or less zero throughout. Although the average recognition rate is lower than the one with LR, the "order" of the other classes is similar to the one described for LR.

Multilayer Perceptron:

The various "jumps" show that MLP reacts the most sensitively to a modified training set. Nonetheless, the "order" of the other classes is similar to the one described for LR when neglecting the average recognition rate.

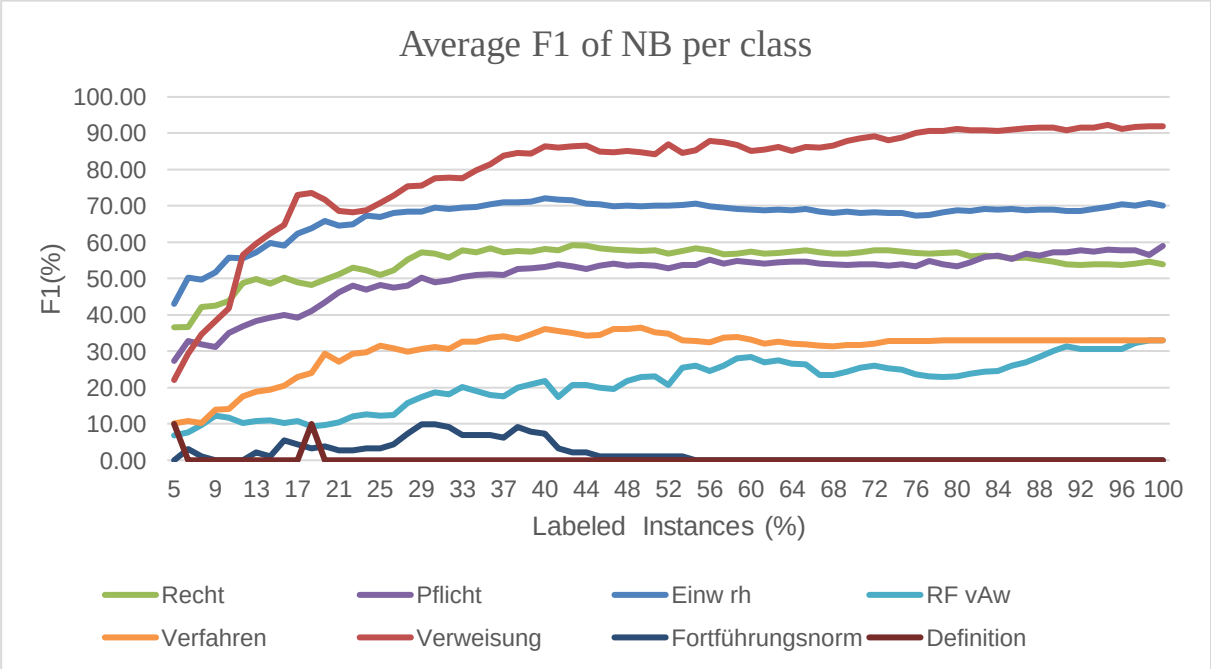


Figure 65: Average F_1 of NB per Class
Source: Own illustration

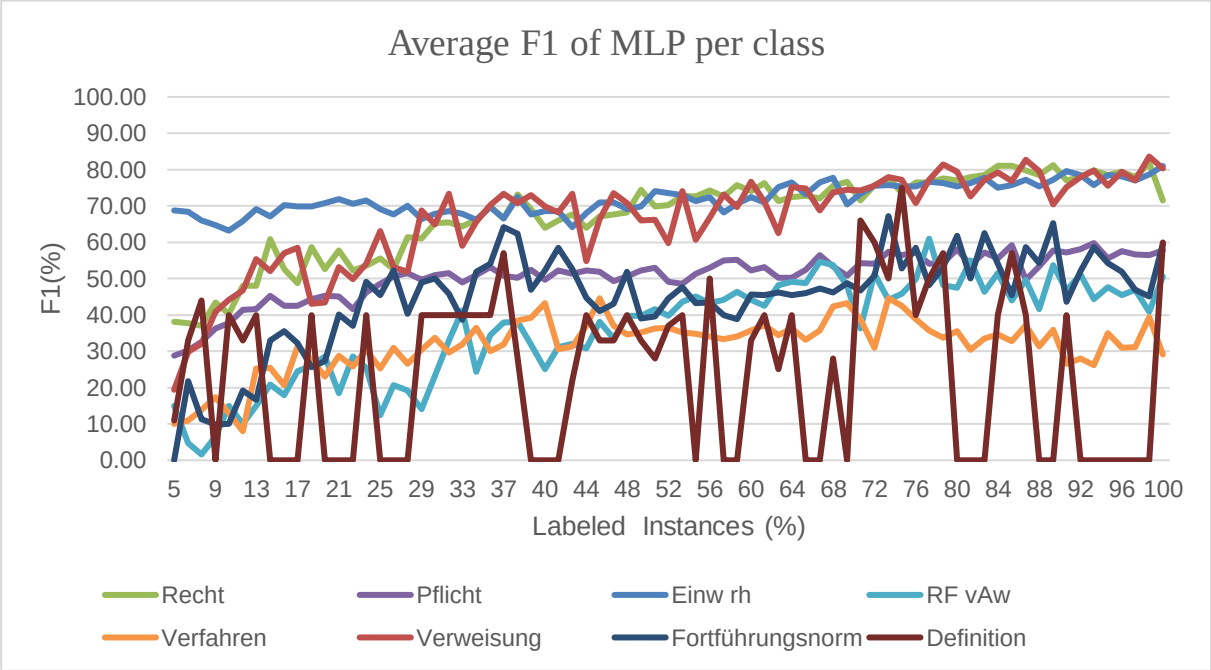


Figure 66: Average F_1 of MLP per Class
Source: Own illustration

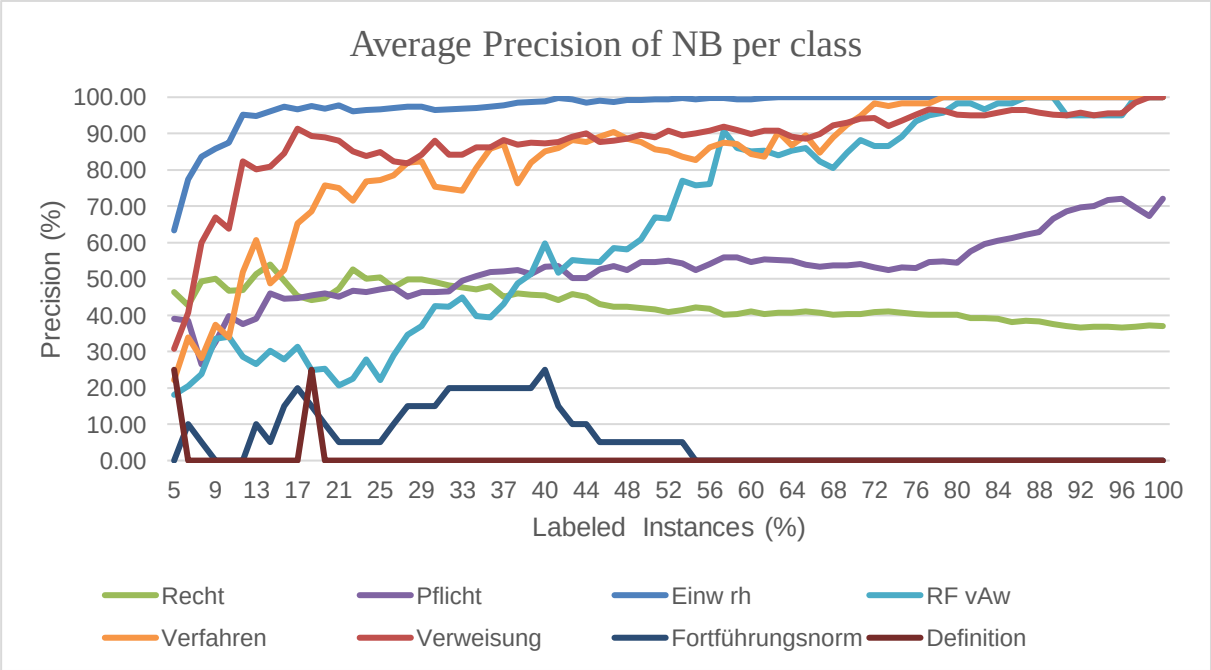


Figure 67: Average Precision of NB per Class
Source: Own illustration

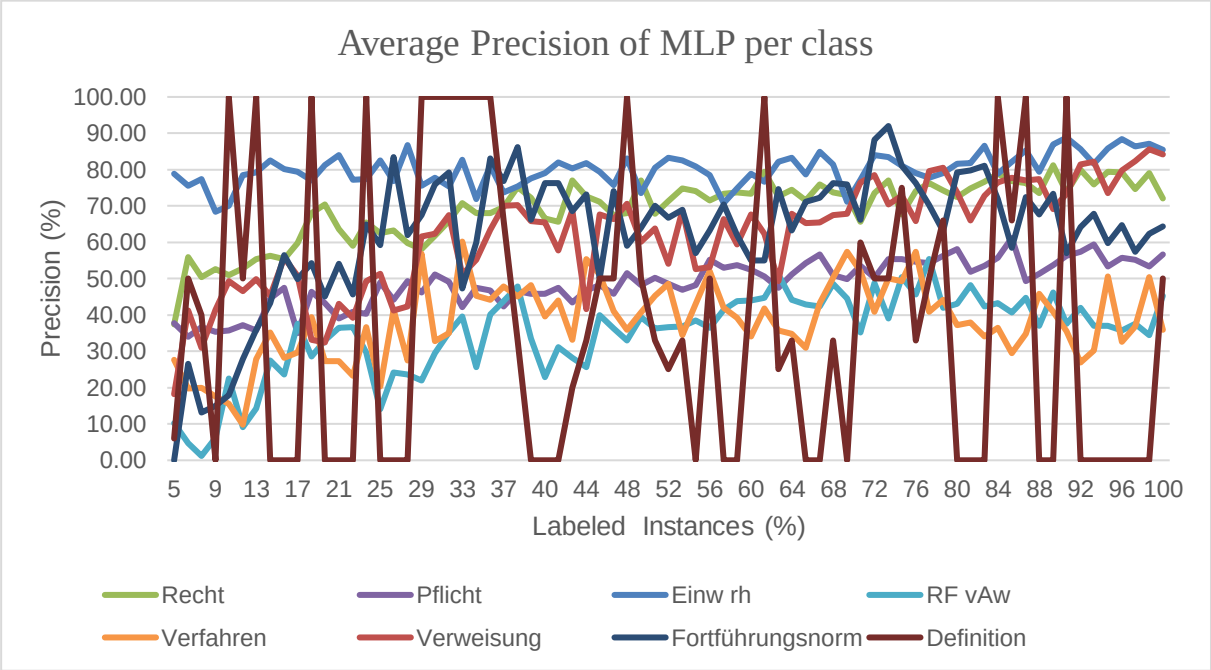


Figure 68: Average Precision of MLP per Class
Source: Own illustration

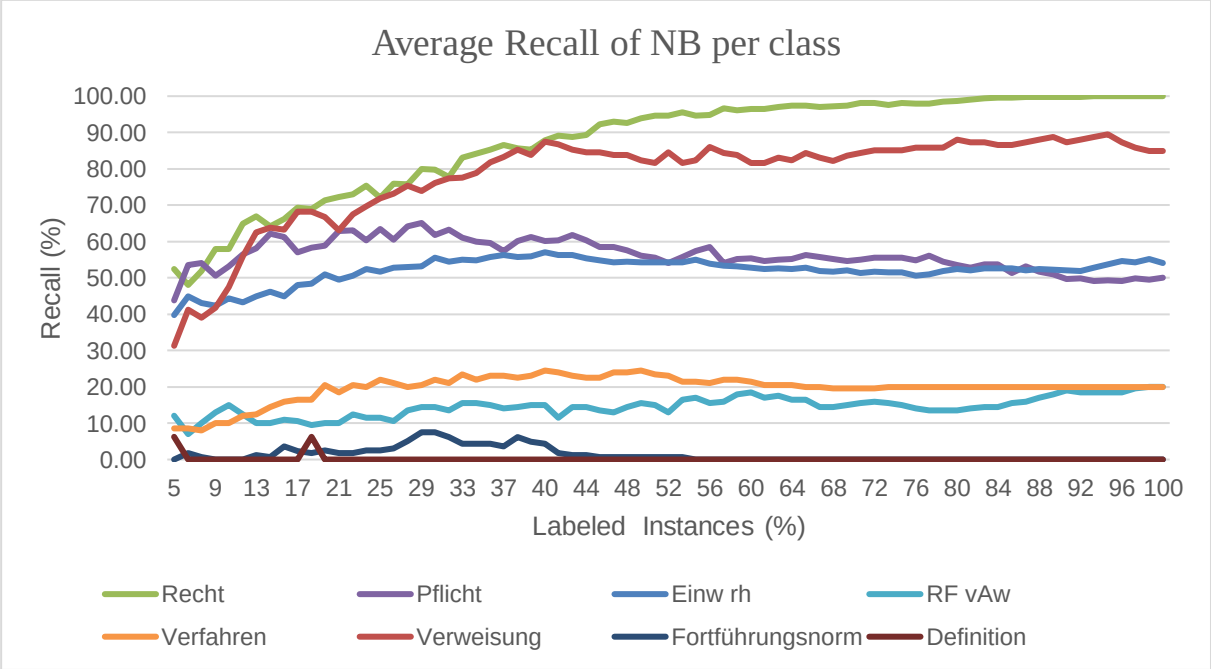


Figure 69: Average Recall of NB per Class
Source: Own illustration

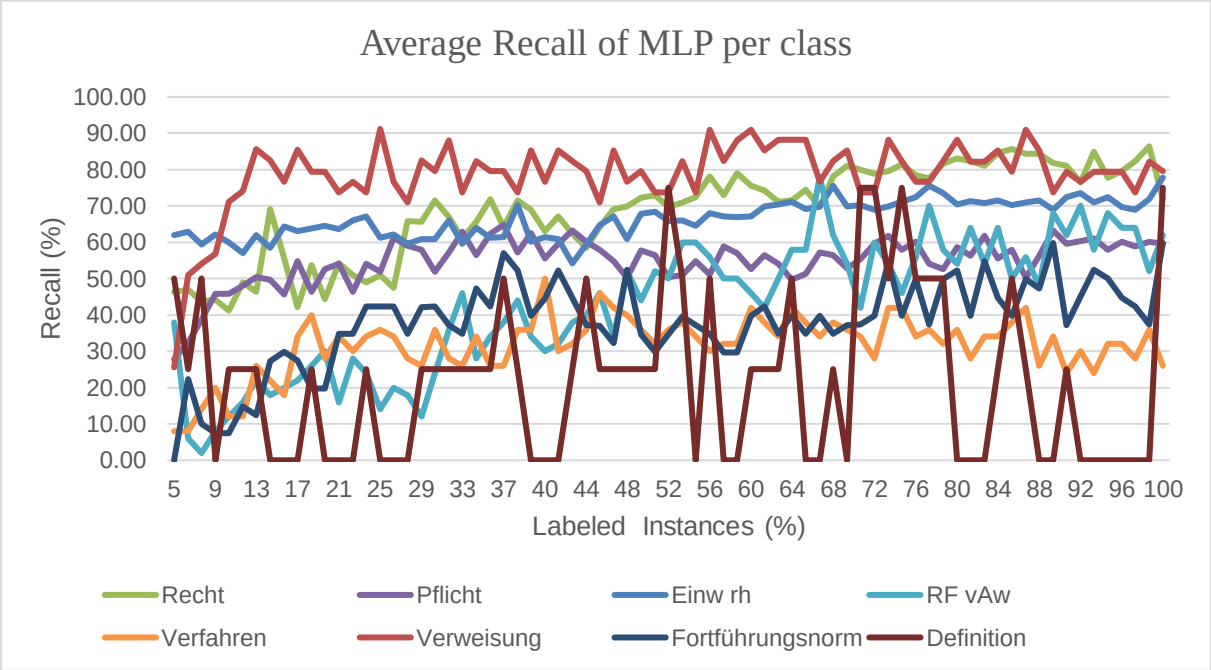


Figure 70: Average Recall of MLP per Class
Source: Own illustration

Literature

1. Liu, B., *Machine Learning and the Future of Law*, L.T. Blog, Editor. 2015.

2. Paul, G.L. and J.R. Baron, *Information inflation: Can the legal system adapt.* Rich. JL & Tech., 2006. **13**: p. 1.

3. Roitblat, H.L., A. Kershaw, and P. Oot, *Document categorization in legal electronic discovery: computer classification vs. manual review.* Journal of the American Society for Information Science and Technology, 2010. **61**(1): p. 70-80.

4. Waltl, B.M., F., *Towards Measures of Complexity: Applying Structural and Linguistic Metrics to German Laws*, in *Jurix: International Conference on Legal Knowledge and Information Systems*. 2014: Krakau, Poland.

5. Sunkle, S., D. Kholkar, and V. Kulkarni. *Informed Active Learning to Aid Domain Experts in Modeling Compliance*. in *2016 IEEE 20th International Enterprise Distributed Object Computing Conference (EDOC)*. 2016.

6. Novak, B., D. Mladenič, and M. Grobelnik, *Text Classification with Active Learning*, in *From Data and Information Analysis to Knowledge Engineering: Proceedings of the 29th Annual Conference of the Gesellschaft für Klassifikation e.V. University of Magdeburg, March 9–11, 2005*, M. Spiliopoulou, et al., Editors. 2006, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 398-405.

7. Tong, S. and D. Koller, *Support vector machine active learning with applications to text classification.* Journal of Machine Learning Research, 2002. **2**(1): p. 45-66.

8. Laws, F. and H. Schätze. *Stopping criteria for active learning of named entity recognition*. in *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*. 2008. Association for Computational Linguistics.

9. Segal, R., T. Markowitz, and W. Arnold. *Fast Uncertainty Sampling for Labeling Large E-mail Corpora*. in *CEAS*. 2006. Citeseer.

10. Šavelka, J., G. Trivedi, and K.D. Ashley, *Applying an Interactive Machine Learning Approach to Statutory Analysis.* 2015.

11. Cardellino, C., et al. *Information Extraction with Active Learning: A Case Study in Legal Text*. in *International Conference on Intelligent Text Processing and Computational Linguistics*. 2015. Springer.

12. Von Alan, R.H., et al., *Design science in information systems research.* MIS quarterly, 2004. **28**(1): p. 75-105.

13. Webster, J. and R.T. Watson, *Analyzing the past to prepare for the future: Writing a literature review.* MIS quarterly, 2002: p. xiii-xxiii.

14. Mitchell, T.M., *The discipline of machine learning*. Vol. 9. 2006.

15. Mitchell, T.M., *Machine learning*. McGraw-Hill series in computer science. 1997, Boston, Mass. [u.a.]: WCB/McGraw-Hill. XVII, 414 S.

16. Alpaydın, E., *Introduction to machine learning*. 3. ed. ed. Adaptive computation and machine learning. 2014, Cambridge, Mass. [u.a.]: MIT Press. XXII, 613 S.

17. Murphy, K.P., *Machine learning : a probabilistic perspective*. Fourth printing: August 2013 (fixed many typos) ed. Adaptive computation and machine learning series. 2013, Cambridge, Massachusetts ; London, England: The MIT Press. xxix, 1071 Seiten.

18. Tsoumakas, G. and I. Katakis, *Multi-label classification: An overview.* International Journal of Data Warehousing and Mining, 2006. **3**(3).

19. Bishop, C.M., *Pattern recognition and machine learning*. Information science and statistics. 2006, New York, NY: Springer. XX, 738 S.

20. AstroML. *Machine Learning and Data Mining for Astronomy*. 2017 [cited 2017 16.01]; Available from: http://www.astroml.org/.

21. Flach, P., *Machine learning : the art and science of algorithms that make sense of data*. 1. publ. ed. 2012, Cambridge: Cambridge Univ. Press. XVII, 396 S.

22. Chapelle, O., B. Scholkopf, and A. Zien, *Semi-Supervised Learning (Chapelle, O. et al., Eds.; 2006)[Book reviews]*. IEEE Transactions on Neural Networks, 2009. **20**(3): p. 542-542.

23. Zhu, X. and A.B. Goldberg, *Introduction to semi-supervised learning.* Synthesis lectures on artificial intelligence and machine learning, 2009. **3**(1): p. 1-130.

24. *Glossary of Terms.* Mach. Learn., 1998. **30**(2-3): p. 271-274.

25. Yang, Y. and J.O. Pedersen. *A comparative study on feature selection in text categorization*. in *ICML*. 1997.

26. Agarwal, B. and N. Mittal, *Text Classification Using Machine Learning Methods-A Survey*, in *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012), December 28-30, 2012*, B.V. Babu, et al., Editors. 2014, Springer India: New Delhi. p. 701-709.

27. Sebastiani, F., *Machine learning in automated text categorization.* ACM computing surveys (CSUR), 2002. **34**(1): p. 1-47.

28. Hoi, S., R. Jin, and M. Lyu. *Large-scale text categorization by batch mode active learning*. in *Proceedings of the 15th international conference on World Wide Web*. 2006. ACM.

29. Yang, Y., *An evaluation of statistical approaches to text categorization.* Information retrieval, 1999. **1**(1-2): p. 69-90.

30. Soucy, P. and G.W. Mineau. *A simple KNN algorithm for text categorization*. in *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. 2001. IEEE.

31. Jacob, E., *Classification and categorization: a difference that makes a difference.* Library trends, 2004. **52**(3).

32. Jacob, E., *Classification and Categorization: Drawing the Line.* 1991, 1991: p. 18.

33. Drucker, H., D. Wu, and V.N. Vapnik, *Support vector machines for spam categorization.* Trans. Neur. Netw., 1999. **10**(5): p. 1048-1054.

34. Guzella, T.S. and W.M. Caminhas, *A review of machine learning approaches to Spam filtering.* Expert Systems with Applications, 2009. **36**(7): p. 10206-10222.

35. Pang, B., L. Lee, and S. Vaithyanathan. *Thumbs up?: sentiment classification using machine learning techniques*. in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. 2002. Association for Computational Linguistics.

36. Nadeau, D. and S. Sekine, *A survey of named entity recognition and classification.* Lingvisticae Investigationes, 2007. **30**(1): p. 3-26.

37. Zhou, G., et al., *Recognizing names in biomedical texts: a machine learning approach.* Bioinformatics, 2004. **20**(7): p. 1178-1190.

38. Baharudin, B., L.H. Lee, and K. Khan, *A review of machine learning algorithms for text-documents classification.* Journal of advances in information technology, 2010. **1**(1): p. 4-20.

39. Aas, K. and L. Eikvil, *Text categorisation: A survey*. 1999, Technical report, Norwegian Computing Center.

40. Uysal, A.K. and S. Gunal, *The impact of preprocessing on text classification.* Information Processing & Management, 2014. **50**(1): p. 104-112.

41. Salton, G. and C. Buckley, *Term-weighting approaches in automatic text retrieval.* Information processing & management, 1988. **24**(5): p. 513-523.

42. Ikonomakis, M., S. Kotsiantis, and V. Tampakas, *Text classification using machine learning techniques.* WSEAS transactions on computers, 2005. **4**(8): p. 966-974.

43. Yi, W. and W. Xiao-Jing. *A new approach to feature selection in text classification*. in *2005 International Conference on Machine Learning and Cybernetics*. 2005.

44. Méndez, J.R., et al. *Tokenising, stemming and stopword removal on anti-spam filtering domain*. in *Conference of the Spanish Association for Artificial Intelligence*. 2005. Springer.

45. Pomikálek, J. and R. Rehurek, *The Influence of preprocessing parameters on text categorization.* International Journal of Applied Science, Engineering and Technology, 2007. **1**: p. 430-434.

46. Toman, M., R. Tesar, and K. Jezek, *Influence of word normalization on text classification.* Proceedings of InSciT, 2006. **4**: p. 354-358.

47. Song, F., S. Liu, and J. Yang, *A comparative study on text representation schemes in text categorization.* Pattern analysis and applications, 2005. **8**(1-2): p. 199-209.

48. Feng, G., et al., *A Bayesian feature selection paradigm for text classification.* Information Processing & Management, 2012. **48**(2): p. 283-302.

49. Abe, N. and M. Kudo, *Non-parametric classifier-independent feature selection.* Pattern recognition, 2006. **39**(5): p. 737-746.

50. Makrehchi, M., *Evaluating feature ranking methods in text classifiers.* Intelligent Data Analysis, 2015. **19**(5): p. 1151-1170.

51. Ertekin, S., et al. *Learning on the border: active learning in imbalanced data classification*. in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. 2007. ACM.

52. Mladenić, D., et al. *Feature selection using linear classifier weights: interaction with classification models*. in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. 2004. ACM.

53. Montanes, E., et al., *Scoring and selecting terms for text categorization.* IEEE Intelligent Systems, 2005. **20**(3): p. 40-47.

54. Rogati, M. and Y. Yang. *High-performing feature selection for text classification*. in *Proceedings of the eleventh international conference on Information and knowledge management*. 2002. ACM.

55. Aggarwal, C.C. and C. Zhai, *A survey of text classification algorithms*, in *Mining text data*. 2012, Springer. p. 163-222.

56. Brindha, S., K. Prabha, and S. Sukumaran. *A survey on classification techniques for text mining*. in *Advanced Computing and Communication Systems (ICACCS), 2016 3rd International Conference on*. 2016. IEEE.

57. Boser, B.E., I.M. Guyon, and V.N. Vapnik. *A training algorithm for optimal margin classifiers*. in *Proceedings of the fifth annual workshop on Computational learning theory*. 1992. ACM.

58. Vapnik, V.N. and S. Kotz, *Estimation of dependences based on empirical data*. Vol. 40. 1982: Springer-Verlag New York.

59. Kotsiantis, S.B., *Supervised machine learning: A review of classification techniques*. 2007.

60. Aghila, G., *A Survey of Naiive Bayes Machine Learning approach in Text Document Classification.* arXiv preprint arXiv:1003.1795, 2010.

61. Colas, F. and P. Brazdil. *Comparison of SVM and some older classification algorithms in text classification tasks*. in *IFIP International Conference on Artificial Intelligence in Theory and Practice*. 2006. Springer.

62. Bouchard, G. and B. Triggs. *The tradeoff between generative and discriminative classifiers*. in *16th IASC International Symposium on Computational Statistics (COMPSTAT'04)*. 2004.

63. Domingos, P. and M. Pazzani, *On the optimality of the simple Bayesian classifier under zero-one loss.* Machine learning, 1997. **29**(2-3): p. 103-130.

64. McCallum, A. and K. Nigam. *A comparison of event models for naive bayes text classification*. in *AAAI-98 workshop on learning for text categorization*. 1998. Citeseer.

65. Rennie, J.D., et al. *Tackling the poor assumptions of naive bayes text classifiers*. in *ICML*. 2003. Washington DC).

66. Cortes, C. and V. Vapnik, *Support-vector networks.* Machine learning, 1995. **20**(3): p. 273-297.

67. Vapnik, V.N., ¬*The nature of statistical learning theory*. 2. ed. ed. Statistics for engineering and information science. 2000, New York [u.a.]: Springer. XIX, 314 S. : graph. Darst.

68. Burges, C.J., *A tutorial on support vector machines for pattern recognition.* Data mining and knowledge discovery, 1998. **2**(2): p. 121-167.

69. Schölkopf, B. and C.J. Burges, *Advances in kernel methods: support vector learning*. 1999: MIT press.

70. Genton, M.G., *Classes of kernels for machine learning: a statistics perspective.* Journal of machine learning research, 2001. **2**(Dec): p. 299-312.

71. Souza, C.R., *Kernel functions for machine learning applications.* Creative Commons Attribution-Noncommercial-Share Alike, 2010. **3**.

72. Hsu, C.-W. and C.-J. Lin, *A comparison of methods for multiclass support vector machines.* IEEE transactions on Neural Networks, 2002. **13**(2): p. 415-425.

73. Ng, H.T., W.B. Goh, and K.L. Low. *Feature selection, perceptron learning, and a usability case study for text categorization*. in *ACM SIGIR Forum*. 1997. ACM.

74. Rosenblatt, F., *The perceptron: a probabilistic model for information storage and organization in the brain.* Psychological review, 1958. **65**(6): p. 386.

75. Rosenblatt, F., *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. 1961, DTIC Document.

76. Haykin, S.S., *Neural networks and learning machines*. 3. ed. ed. Pearson International Edition. 2009, Upper Saddle River [u.a.]: Pearson. 934 S.

77. Littlestone, N., *Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm.* Machine learning, 1988. **2**(4): p. 285-318.

78. Dagan, I., Y. Karov, and D. Roth, *Mistake-driven learning in text categorization.* arXiv preprint cmp-lg/9706006, 1997.

79. Wiener, E., J.O. Pedersen, and A.S. Weigend. *A neural network approach to topic spotting*. in *Proceedings of SDAIR-95, 4th annual symposium on document analysis and information retrieval*. 1995. Citeseer.

80. Zhang, G.P., *Neural networks for classification: a survey*. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2000. **30**(4): p. 451-462.

81. Schütze, H., D.A. Hull, and J.O. Pedersen. *A comparison of classifiers and document representations for the routing problem*. in *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*. 1995. ACM.

82. Rumelhart, D.E., G.E. Hinton, and R.J. Williams, *Learning internal representations by error propagation*. 1985, DTIC Document.

83. Fawcett, T., *An introduction to ROC analysis.* Pattern recognition letters, 2006. **27**(8): p. 861-874.

84. Bradley, A.P., *The use of the area under the ROC curve in the evaluation of machine learning algorithms.* Pattern recognition, 1997. **30**(7): p. 1145-1159.

85. Hanley, J.A. and B.J. McNeil, *The meaning and use of the area under a receiver operating characteristic (ROC) curve.* Radiology, 1982. **143**(1): p. 29-36.

86. Hand, D.J. and R.J. Till, *A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems.* Mach. Learn., 2001. **45**(2): p. 171-186.

87. Provost, F. and P. Domingos, *Well-trained PETs: Improving probability estimation trees.* 2000.

88. Settles, B., *Active learning literature survey.* University of Wisconsin, Madison, 2010. **52**(55-66): p. 11.

89. Settles, B., *Active Learning.* Synthesis Lectures on Artificial Intelligence and Machine Learning, 2012. **6**(1): p. 1-114.

90. Olsson, F., *A literature survey of active machine learning in the context of natural language processing.* 2009.

91. Angluin, D., *Queries and concept learning.* Machine learning, 1988. **2**(4): p. 319-342.

92. Atlas, L., et al., *Training connectionist networks with queries and selective sampling*, in *Advances in neural information processing systems 2*, S.T. David, Editor. 1990, Morgan Kaufmann Publishers Inc. p. 566-573.

93. McCallum, A. and K. Nigam. *Employing EM and pool-based active learning for text classification*. in *Proc. International Conference on Machine Learning (ICML)*. 1998. Citeseer.

94. Sculley, D. *Online Active Learning Methods for Fast Label-Efficient Spam Filtering*. in *CEAS*. 2007.

95. Lewis, D.D. and W.A. Gale. *A sequential algorithm for training text classifiers*. in *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. 1994. Springer-Verlag New York, Inc.

96. Thompson, C.A., M.E. Califf, and R.J. Mooney. *Active learning for natural language parsing and information extraction*. in *ICML*. 1999. Citeseer.

97. Zhu, J., et al. *Active learning with sampling by uncertainty and density for word sense disambiguation and text classification*. in *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*. 2008. Association for Computational Linguistics.

98. Dligach, D. and M. Palmer. *Good seed makes a good crop: accelerating active learning using language modeling*. in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*. 2011. Association for Computational Linguistics.

99. Tomanek, K., et al. *On proper unit selection in active learning: co-selection effects for named entity recognition*. in *Proceedings of the NAACL HLT 2009 Workshop on Active Learning for Natural Language Processing*. 2009. Association for Computational Linguistics.

100. Nguyen, H.T. and A. Smeulders. *Active learning using pre-clustering*. in *Proceedings of the twenty-first international conference on Machine learning*. 2004. ACM.

101. Osugi, T., D. Kim, and S. Scott. *Balancing exploration and exploitation: A new algorithm for active machine learning*. in *Fifth IEEE International Conference on Data Mining (ICDM'05)*. 2005. IEEE.

102. Mingkun, L. and I.K. Sethi, *Confidence-based active learning.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2006. **28**(8): p. 1251-1261.

103. Zadrozny, B. and C. Elkan. *Transforming classifier scores into accurate multiclass probability estimates*. in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2002. ACM.

104. Platt, J., *Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods.* Advances in large margin classifiers, 1999. **10**(3): p. 61-74.

105. Schein, A.I. and L.H. Ungar, *Active learning for logistic regression: an evaluation.* Machine Learning, 2007. **68**(3): p. 235-265.

106. Seung, H.S., M. Opper, and H. Sompolinsky. *Query by committee*. in *Proceedings of the fifth annual workshop on Computational learning theory*. 1992. ACM.

107. Freund, Y. and R.E. Schapire. *A desicion-theoretic generalization of on-line learning and an application to boosting*. in *European conference on computational learning theory*. 1995. Springer.

108. Freund, Y. and R.E. Schapire. *Experiments with a new boosting algorithm*. in *Icml*. 1996.

109. Mamitsuka, N.A.H. *Query learning strategies using boosting and bagging*. in *Machine Learning: Proceedings of the Fifteenth International Conference (ICML'98)*. 1998. Morgan Kaufmann Pub.

110. Freund, Y., *Boosting a Weak Learning Algorithm by Majority.* Information and Computation, 1995. **121**(2): p. 256-285.

111. Breiman, L., *Bagging predictors.* Machine learning, 1996. **24**(2): p. 123-140.

112. Melville, P. and R.J. Mooney. *Constructing diverse classifier ensembles using artificial training examples*. in *IJCAI*. 2003. Citeseer.

113. Melville, P. and R.J. Mooney. *Diverse ensembles for active learning*. in *Proceedings of the twenty-first international conference on Machine learning*. 2004. ACM.

114. Dagan, I. and S.P. Engelson. *Committee-based sampling for training probabilistic classifiers*. in *Proceedings of the Twelfth International Conference on Machine Learning*. 1995. The Morgan Kaufmann series in machine learning,(San Francisco, CA, USA).

115. Körner, C. and S. Wrobel, *Multi-class Ensemble-Based Active Learning*, in *Machine Learning: ECML 2006: 17th European Conference on Machine Learning Berlin, Germany, September 18-22, 2006 Proceedings*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Editors. 2006, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 687-694.

116. Ngai, G. and D. Yarowsky. *Rule writing or annotation: Cost-efficient resource usage for base noun phrase chunking*. in *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*. 2000. Association for Computational Linguistics.

117. Kullback, S. and R.A. Leibler, *On information and sufficiency.* The annals of mathematical statistics, 1951. **22**(1): p. 79-86.

118. Settles, B., M. Craven, and S. Ray. *Multiple-instance active learning*. in *Advances in neural information processing systems*. 2008.

119. Roy, N. and A. McCallum, *Toward optimal active learning through monte carlo estimation of error reduction.* ICML, Williamstown, 2001: p. 441-448.

120. Settles, B. and M. Craven. *An analysis of active learning strategies for sequence labeling tasks*. in *Proceedings of the conference on empirical methods in natural language processing*. 2008. Association for Computational Linguistics.

121. Settles, B. *Closing the loop: Fast, interactive semi-supervised annotation with queries on features and instances*. in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. 2011. Association for Computational Linguistics.

122. Tomanek, K. and K. Morik, *Inspecting Sample Reusability for Active Learning.* Active Learning and Experimental Design@ AISTATS, 2011. **2011**: p. 169-181.

123. Probst, K. and R. Ghani. *Towards 'interactive'active learning in multi-view feature sets for information extraction*. in *European Conference on Machine Learning*. 2007. Springer.

124. Tomanek, K. and F. Olsson. *A web survey on the use of active learning to support annotation of text data*. in *Proceedings of the NAACL HLT 2009 Workshop on Active Learning for Natural Language Processing*. 2009. Association for Computational Linguistics.

125. Tong, S., *Active learning: theory and applications*. 2001, Citeseer.

126. Bordes, A., et al., *Fast kernel classifiers with online and active learning.* Journal of Machine Learning Research, 2005. **6**(Sep): p. 1579-1619.

127. Raghavan, H., O. Madani, and R. Jones, *Active learning with feedback on features and instances.* Journal of Machine Learning Research, 2006. **7**(Aug): p. 1655-1686.

128. Schohn, G. and D. Cohn. *Less is more: Active learning with support vector machines*. in *ICML*. 2000. Citeseer.

129. Guo, H. and W. Wang, *An active learning-based SVM multi-class classification model.* Pattern recognition, 2015. **48**(5): p. 1577-1597.

130. Yang, B.S., et al., *Effective Multi-Label Active Learning for Text Classification*. Kdd-09: 15th Acm Sigkdd Conference on Knowledge Discovery and Data Mining. 2009, New York: Assoc Computing Machinery. 917-925.

131. Cohn, D., L. Atlas, and R. Ladner, *Improving generalization with active learning.* Machine learning, 1994. **15**(2): p. 201-221.

132. Takizawa, H., et al., *An active learning algorithm based on existing training data.* IEICE TRANSACTIONS on Information and Systems, 2000. **83**(1): p. 90-99.

133. Freund, Y., et al., *Selective sampling using the query by committee algorithm.* Machine learning, 1997. **28**(2-3): p. 133-168.

134. Lu, Z., et al. *Informative sampling for large unbalanced data sets*. in *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*. 2008. ACM.

135. Wang, R., et al. *Active Learning Based on Single-Hidden Layer Feed-Forward Neural Network.* in *2015 IEEE International Conference on Systems, Man, and Cybernetics*. 2015.

136. Vlachos, A., *A stopping criterion for active learning.* Computer Speech & Language, 2008. **22**(3): p. 295-312.

137. Wang, J.Z.H. and E. Hovy. *Learning a stopping criterion for active learning for word sense disambiguation and text classification*. in *Third International Joint Conference on Natural Language Processing*. 2008.

138. Zhu, J., H. Wang, and E. Hovy. *Multi-criteria-based strategy to stop active learning for data annotation*. in *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*. 2008. Association for Computational Linguistics.

139. Bloodgood, M. and K. Vijay-Shanker. *A method for stopping active learning based on stabilizing predictions and the need for user-adjustable stopping*. in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*. 2009. Association for Computational Linguistics.

140. Hu, R., B. Mac Namee, and S.J. Delany, *Active learning for text classification with reusability.* Expert Systems with Applications, 2016. **45**: p. 438-449.

141. Zhu, J. and E.H. Hovy. *Active Learning for Word Sense Disambiguation with Methods for Addressing the Class Imbalance Problem*. in *EMNLP-CoNLL*. 2007.

142. Guyon, I., et al. *Design and analysis of the WCCI 2010 active learning challenge*. in *The 2010 International Joint Conference on Neural Networks (IJCNN)*. 2010. IEEE.

143. Guyon, I., et al., *Results of the Active Learning Challenge.* Active Learning and Experimental Design@ AISTATS, 2011. **16**: p. 19-45.

144. Baram, Y., R.E. Yaniv, and K. Luz, *Online choice of active learning algorithms.* Journal of Machine Learning Research, 2004. **5**(Mar): p. 255-291.

145. Druck, G., B. Settles, and A. McCallum. *Active learning by labeling features*. in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*. 2009. Association for Computational Linguistics.

146. Surden, H., *Machine learning and law.* Wash. L. Rev., 2014. **89**: p. 87.

147. Gruner, R.H., *Anatomy of a Lawsuit - A Client's Analysis and Discussion of a Multi-Million Dollar Federal Lawsuit*. 2008.

148. Opsomer, R., et al. *Exploiting properties of legislative texts to improve classification accuracy*. in *Legal Knowledge and Information Systems: JURIX 2009, the Twenty-second Annual Conference*. 2009. IOS Press.

149. Busse, D., *Textsorten des Bereichs Rechtswesen und Justiz,* in *Text- und Gesprächslinguistik. Ein internationales Handbuch zeitgenössischer Forschung. (Handbücher zur Sprach- und Kommunikationswissenschaft)*, G. Antos, et al., Editors. 2000, de Gruyter: Berlin/New York. p. 658-675.

150. Gonçalves, T. and P. Quaresma. *Is linguistic information relevant for the classification of legal texts?* in *Proceedings of the 10th international conference on Artificial intelligence and law*. 2005. ACM.

151. Ratner, A., *Leveraging Document Structure for Better Classification of Complex Legal Documents.*

152. de Maat, E. and R. Winkels, *A next step towards automated modelling of sources of law,* in *Proceedings of the 12th International Conference on Artificial Intelligence and Law*. 2009, ACM: Barcelona, Spain. p. 31-39.

153. Francesconi, E. and A. Passerini, *Automatic classification of provisions in legislative texts.* Artificial Intelligence and Law, 2007. **15**(1): p. 1-17.

154. Di Silvestro, L., D. Spampinato, and A. Torrisi, *Automatic Classification of Legal Textual Documents using C4. 5.*

155. de Maat, E., K. Krabben, and R. Winkels. *Machine Learning versus Knowledge Based Classification of Legal Texts*. in *JURIX*. 2010.

156. Moens, M.-F., et al. *Automatic detection of arguments in legal texts*. in *Proceedings of the 11th international conference on Artificial intelligence and law*. 2007. ACM.

157. Palau, R.M. and M.-F. Moens. *Argumentation mining: the detection, classification and structure of arguments in text*. in *Proceedings of the 12th international conference on artificial intelligence and law*. 2009. ACM.

158. Landset, S., et al., *A survey of open source tools for machine learning with big data in the Hadoop ecosystem.* Journal of Big Data, 2015. **2**(1): p. 24.

159. Karau, H., et al., *Learning spark: lightning-fast big data analysis*. 2015: " O'Reilly Media, Inc.".

160. Laskowski, J. *Mastering Apache Spark 2.0*. 2017; Available from: https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-overview.html.

161. Xin, R., *Apache Spark officially sets a new record in large-scale sorting,* E. Blog, Editor. 2014.

162. Meng, X., et al., *Mllib: Machine learning in apache spark.* JMLR, 2016. **17**(34): p. 1-7.

163. Zheng, J. and A. Dagnino. *An initial study of predictive machine learning analytics on large volumes of historical data for power system applications*. in *2014 IEEE International Conference on Big Data (Big Data)*. 2014.

164. Owen, S., et al., *Mahout in action*. 2011, USA: Manning Publications Co.

165. Hall, M., et al., *The WEKA data mining software: an update.* ACM SIGKDD explorations newsletter, 2009. **11**(1): p. 10-18.

166. Witten, I.H., et al., *Data Mining: Practical machine learning tools and techniques*. 2016: Morgan Kaufmann.

167. McCallum, A.K. *MALLET: A Machine Learning for Language Toolkit.* 2002; Available from: http://mallet.cs.umass.edu/.

168. Waltl, B., et al., *LEXIA - A Data Science Environment for Semantic Analysis of German Legal Texts*, in *IRIS: Internationales Rechtsinformatik Symposium*. 2016: Salzburg, Austria.

169. Waltl, B., M. Zec, and F. Matthes, *LEXIA: A Data Science Environment for Legal Texts*, in *Jurix: International Conference on Legal Knowledge and Information Systems*. 2015: Braga, Portugal.

170. Masse, M., *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. 2011: " O'Reilly Media, Inc.".

171. Ng, A.Y. *Feature selection, L 1 vs. L 2 regularization, and rotational invariance*. in *Proceedings of the twenty-first international conference on Machine learning*. 2004. ACM.