Text Similarity Techniques for Matching Employee Objectives

Ahmed Elnaggar[*‡] ahmed.elnaggar@tum.de
Mohab Ghanem[*] ge39god@mytum.de
Florian Matthes matthes@in.tum.de
Lehrstuhl für Informatik 19 - sebis
Fakultät für Informatik der TU München
Boltzmannstrasse 3
85748 Garching, Germany

Adam David Mckinnon[*] adam-david.mckinnon@merckgroup.com
new company address

Christian Debes christian.debes@merckgroup.com
Merck
Frankfurter Str. 250
64293 Darmstadt, Germany

Editor:

Abstract

The application of matching employees based on their personal objectives is made difficult in multinational companies that have many offices spread worldwide and thousands of employees. This is due to the fact that manually comparing the personal objectives of thousands of employees, and extracting similar candidates is time and effort consuming. The problem becomes more difficult when employees change their personal objectives on a yearly basis. In this paper, we present a system to automate the process of matching employees based on their personal objectives. The system extracts similar employees based on the semantic similarity of their personal objectives encoded in word embeddings. Usability evaluations show that the system improves the process of matching employees based on their similar objectives in terms of reducing the time consumed and finding results that are difficult to be obtained by human matching.

Keywords: Text Similarity, Word Embedding, Document Embedding

1. Introduction

Merck is a multinational company with 57,000 employees and multiple offices in 66 countries around the globe. Employees in Merck have to write a personal objective for their year. The personal objective is a piece of text which states a goal that the employee tries to develop throughout the year, like acquiring better presentation skills or improving their document editing skills. Employees with similar personal objectives are matched together so that they can cooperate and encourage one another to work on their goals. The HR

∗. These authors contributed equally to this work.
‡. The official GitHub repository: https://github.com/agemagician/

department in Merck is responsible for the process of matching employees with similar objectives, which is a very difficult task to achieve because of the huge number of employees in Merck, the separation across different countries, and also the variability in the length of the objective for each user. The process is even more complicated because it has to be repeated every year, which motivates the desire to automate this recurrent and effort consuming task.

Encoding textual data into numerical vectors can be achieved by a number of methods. Notably among these methods are those based on the distributional hypothesis, which states that words with similar meaning should have similar vectors, where similarity is indicated by the closeness in vector space (Landauer and Dumais, 1997). Word embedding algorithms are a class of algorithms inspired by the distributional hypothesis and learn numerical representations for words based on contexts in which those words appear. A generalization of the concept of word embeddings are sentence or document embeddings, which learn a numerical representation for a piece of text based on the words it contains.

Word Embeddings have shown to be of great help in various applications like document search, language translation, as well as sentiment analysis (Socher et al., 2013a,b). The task of finding personal objectives that are semantically similar to other personal objectives can be rendered as a document search problem with two steps, First, each textual personal objective is encoded into its numerical vector using a word embedding algorithm. Second, the semantic similarity is computed (represented in distance) between all possible pairs of personal objectives vectors.

The main contributions of this work include: A comparison of pretrained word and sentence embeddings for the task of semantic similarity of sentences; A system for suggesting work partners for a given employee based on semantic similarity of their personal objectives; A usability evaluation of the implemented system.

The rest of this paper is organized as follows. Sections 2 discusses background information about word embedding models and related works that use word embeddings in solving business problems. Section 3 discusses the employee objectives data obtained from Merck and gives a concise overview of the implementations of word embedding models and distance metrics used in our system. Section 4 discusses the components of the system and their implementation details. Section 5 presents a usability evaluation on various aspects of the implemented system and discusses the results.

2. Background

In Natural Language Processing, methods based on word embeddings have proven to outperform other methods for various tasks. Among those tasks is the task of estimating the semantic similarity of text (Socher et al., 2013b). Many problems in research and industry require finding the semantic similarity of text as part of the solution. This section starts by discussing some related works that use word embeddings to solve different business use cases, then presents a quick summary of word embedding models used in our system.

2.1 Related Words

In recent works, word embeddings were used in job board websites to automate the process of recommending jobs to job seekers, and finding the degree to which an applicant's profile fits a job description (Yuan et al., 2016; Pombo, 2019; Schnitzer et al., 2019; Fernández-Reyes and Shinde, 2019). Similarly and inspired by the idea of embedding words in a vector space, Other works extract skills from job descriptions and applicant's profiles and map these skills to their own vector space called skill2vec (Van-Duyet et al., 2017; Wong et al., 2017). A different application of the task of semantic text similarity is information retrieval in software engineering, where queries are expressed in natural language, while results are a mixture of natural language and programming languages (code). Recent work by (Ye et al., 2016) tries to improve information retrieval in software engineering websites by using word embedding algorithms to create a vector space of software engineering documents like API documentation, bug reports, and code snippets. The distance between the embedded representations of these documents in the vector space serves as an indicator of similarity between their content. This is used in an information retrieval system, for example, a community question-answering platform to suggest answers containing code snippets to a question in natural language.

2.2 Text Embedding Models

Thanks to the advancements in deep neural networks, many word embedding algorithms have been developed and their number keeps growing rapidly. Word embedding algorithms are mainly based on the distributional hypothesis, which states that words that appear in similar contexts have a similar meaning, and motivates the idea of using the surrounding context of a target word to identify its features (Firth, 1957). Different algorithms have different ways of defining the context of the target word, and the target word itself, in some cases the target is even a sentence or a full document, not just a single word. Therefore, we choose to divide embedding algorithms roughly into two categories: (1) word-based and (2) sentence-based. Word-based embedding algorithms are those which learn representation for a single word during their training process. While sentence-based algorithms are those which learn representation for multiple words during their training process. This section gives a brief description of the embedding algorithms supported by our system.

2.2.1 Word-Based

We start with an overview of word-based embedding algorithms. The general flow of the upcoming algorithms is that they convert a single word to its numerical representation in vector space. To get the representation of a full document in vector space, we embed each word in the document into vector space and take their average as the document embedding.

One of the earliest algorithms for embedding words in a vector space is The Neural Probabilistic Language Model (Bengio et al., 2003). The model uses a neural network to estimate the probability of a sequence of words. A language model is created in such a way that likely word sequences get high probabilities, whereas unlikely word sequences get low probabilities

(Shi, 2017). The probability of a sequence of words is represented as the product of the conditional probability of each word in the sequence given its previous words.

$$P(w_1^T) = \prod_{t=1}^{T} P(w_t | w_1^{t-1}) \tag{1}$$

Language models can also be used to predict the next word in a sentence, and get a numerical representation of words.

Word2vec (Mikolov et al., 2013b) is a breakthrough in the area of word embeddings which paved the way for many other advancements. The word2vec algorithm comes in two flavors: it can either use a neural network that learns to predict a word given its context (called Continuous Bag of Words model) or use the network to predict the context given a word (called Skip-gram model). In both cases, it uses the parameters learned by the network as a vector representation for the target word.

ELMo (Peters et al., 2018) is the next breakthrough in the area of word embeddings as it addresses one of the big drawbacks of previous word embedding models. In previous models, a word (syntax) has only one numerical representation, regardless of its usage in context (semantics). ELMo tackles this problem by introducing the concept of contextualized word embeddings, that is a word does not have a single numerical representation that is independent of its context, but rather multiple numerical representations depending on the context of this word. ELMo also uses a deep bidirectional language model represented in a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) that uses previous and also future contexts to model the probability of a given word.

The LSTM used in ELMo passes through the text in a sequential manner, which means that ELMo can not be trained on multiple processors in parallel. The research community then switched to using Transformers (Vaswani et al., 2017) instead of LSTMs, because transformers are based on the attention model, which does not require the input to be fed sequentially. BERT (Devlin et al., 2018) is a popular transformer architecture that takes into account left and right contexts, it is trained using masked language modeling, and next sentence prediction.

BERT suffers from some problems regarding allocated memory during training and quick growth of training time. ALBERT (Lan et al., 2019) is an improvement over BERT that tries to address these problems by implementing parameter reduction techniques and allows for better scalability. DistilBERT (Sanh, 2019) is another attempt to overcome the memory consumption and training time problems introduced in BERT. DistilBERT uses knowledge distillation to reduce the number of parameters required in BERT by 40% while retaining 97% of BERT's performance, in addition to being 60% faster. RoBERTa (Liu et al., 2019) is again another attempt to overcome the memory consumption and training time problems introduced in BERT. The authors of RoBERTa examine the usefulness of many of the design choices and hyperparameters used in BERT. They remove training on next

sentence prediction and train RoBERTa only on masked language models, they also change the learning rate and use larger mini-batches. Furthermore, they augment the masked language model of BERT with the idea of dynamic masking, where the masked tokens change during the training epochs. Finally, they report better performance than BERT on multiple downstream tasks.

OpenAI GPT (Radford et al., 2018) is another language model based on the transformer architecture. However, the transformer architecture for GPT is unidirectional, unlike BERT which employed a bidirectional transformer architecture. GPT2 (Radford et al., 2019) is a direct scale-up of GPT. It is also based on a unidirectional transformer architecture, but trained with more than 10 times the number of parameters in GPT, and trained on more than 10 times the size of data.

XLNet (Yang et al., 2019) is another transformer model. It aims to overcome the drawbacks of BERT's masked language model training, and introduces the idea of permutation language modeling where all tokens are masked and predicted in random order, this contrasts BERT which only masks 15% of the tokens in sequential order. The authors of XLNet report that their work improves upon BERT in 20 NLP tasks.

XLM (Lample and Conneau, 2019) is another improvement over BERT. It also trains by masked language modeling, however, it uses corresponding sentences from different languages in order to learn relations between words in those languages. This allows the model to use the context from one language to predict the word in the other language. This also allows XLM to share vocabulary between languages and learn new inherent relations.

The Reformer (Kitaev et al., 2020) is yet another transformer architecture that focuses on modeling long sequences of text while remaining memory efficient. One of the issues with BERT is that it limits the number of tokens in its input text to 512 tokens. However, the Reformer relaxes this threshold and can support up to 1M tokens in the input. The way Reformer takes in this huge number of tokens while remaining memory efficient is by approximating the full attention computation using Locality Sensitive Hashing attention, which results in a drop of space complexity from $O(L^2)$ to $O(L \log(L))$ where L is the length of the sequence.

To improve the performance of transformer models that we discussed previously, most of the attempts relied on having either a larger model, training on a bigger dataset, or training for a longer duration. ELECTRA (Clark et al., 2020) - short for Efficiently Learn an Encoder that Classifies Token Replacements Accurately - is a recent transformer model that tries to improve performance otherwise. The idea behind ELECTRA is that all these resources and computations are not yet exploited in their entirety and that it is still possible to improve the performance using the same model size and the same amount of data by changing the way by which the model learns, which is masked language modeling. The way ELECTRA works is that it adds a discriminator layer after the masked language model generator, the goal of this discriminator is to check for each token in the output of the generator whether it was replaced by the generator or not. In non-technical terms, the

generator cares about replacing the masked tokens with the most likely token values, the discriminator cares about whether those replaced values by the generator are actually good enough. The authors of ELECTRA reports that its contextual representations outperform those of BERT trained using the same model size, amount of data, and training duration, they also report outperforming GPT, RoBERTa, and XLNet.

The T5 model (Raffel et al., 2019) - short for Text-To-Text Transfer Transformer - is another transformer model that differs from all the previously discussed transformer models in the way it learns. The model treats a wide variety of NLP tasks as simply taking input text and predicting output text. For example, for the task of sentiment analysis, the input text is a sentence, and the output text is the semantic polarity expressed as text. Another example is machine translation, the input text is a sentence in the source language, and the output text is the translated sentence in the target language. This approach enables the T5 model to perform many NLP tasks. It also enables the model to combine the information it learns from all the NLP tasks it is trained on.

2.2.2 Sentence-Based

This section explains sentence-based embedding algorithms. For these models, no averaging over individual word embeddings is required as these models output a full-sentence embedding. Sentence-based models are designed to produce embeddings that encode the semantics of a complete sentence or document.

Contrary to the word embedding algorithms described in the previous section, the Universal Sentence Encoder (Cer et al., 2018) is trained to model word sequences rather than individual words. The Universal Sentence Encoder comes in one of two implementations, either as a Transformer Encoder or as a Deep Averaging Network Encoder.

The most recent work at the time of writing this report is Sentence transformers (Reimers and Gurevych, 2019, 2020; **?**) which augments some of the previously mentioned transformer models like BERT and DistilBERT to model word sequences rather than single words. The models are fine-tuned to produce sentence embeddings such that sentences of similar meanings get close embeddings in vector space. The authors of Sentence Transformers report achieving state of the art performance in multiple NLP tasks including semantic text similarity.

3. Methods

This section presents an analysis of the job description data obtained from Merck, and discusses the implementations of the pretrained embedding models used in our system, and the distance metrics used to estimate the pairwise similarity between embedded objectives.

3.1 Merck Dataset

Merck provided an anonymized subset of historical data of employee's objectives from the years 2018 and 2019. All of the entries of the data are from employees in the HR depart-

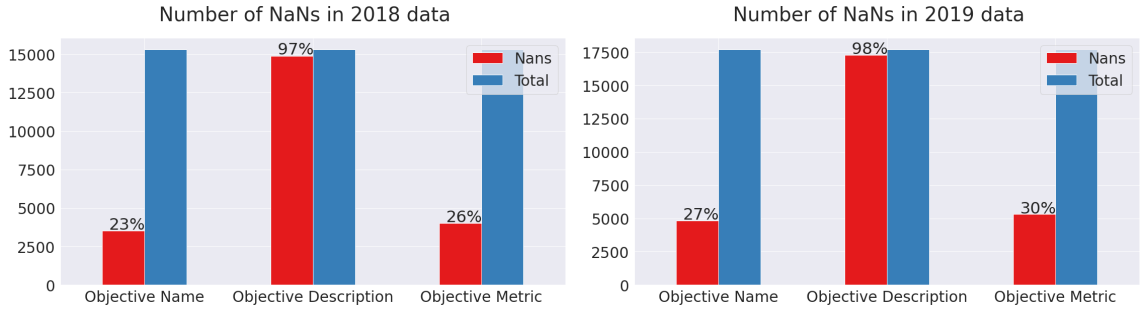


Figure 1: Number of NaNs (missing values) in (a) 2018 and (b) 2019 data. A full employee objective is spread along the three columns on the x-axis: *Objective Name, Objective Description*, and *Objective Metric*. The distribution of missing values is almost the same in both years. The amount of missing values in *Objective Description* is large.

ment. The format of the data is a *csv* file with 30K rows and 9 columns. The useful columns for our use case are: *User ID, Objective Name, Objective Description*, and *Objective Metric*. The last three columns constitute the complete personal objective of a user. We divide the data by year and discuss some properties of the data:

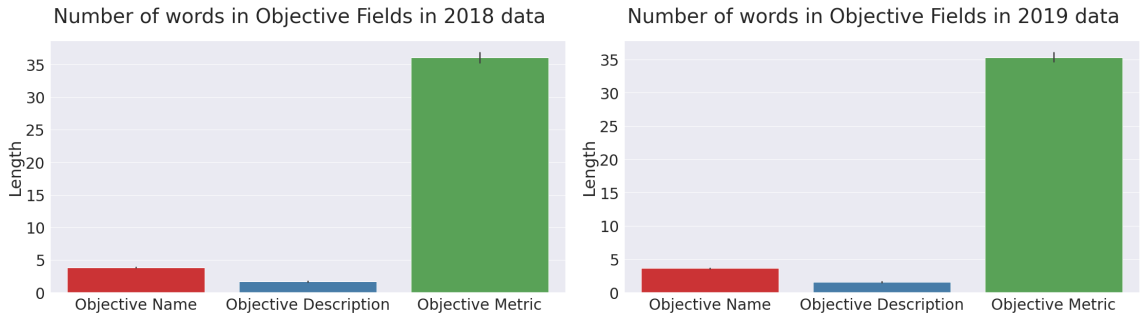
1. Missing Values
For the three columns constituting an objective, we look at the number and percentage of missing values as shown in Figure 1. We notice that for the 2019 data, 98% of the *Objective Description* column is missing which is a huge amount of missing data. The same distribution appears in 2018 data with slightly different numbers.

2. Length of Objective Text
For the three columns constituting an objective, we look at the average number of words in each column as shown in Figure 2. We notice that the *Objective Metric* column has the longest text. But from the previous point, we notice that it also has 30% of missing values

We solve the above two issues (1 and 2) by concatenating the text in the three columns to form one *Combined Objective* column, which will serve as our main text for generating numerical representation.

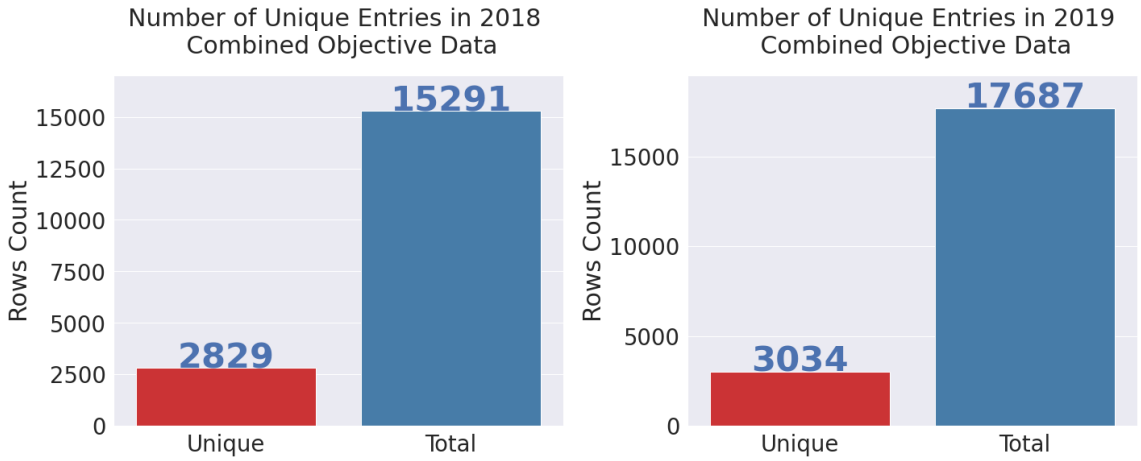
3. Uniqueness of Combined Objective
After creating the *Combined Objective* column, we investigate the uniqueness of the values as shown in Figure 3. Duplicate rows can be caused by the same employee writing the same objective more than once, or multiple employees copying each other's objectives. We notice that only about 20% of the data is unique.

4. Number of Objectives Per User
We further look at the number of personal objectives that can exist for a user, these include



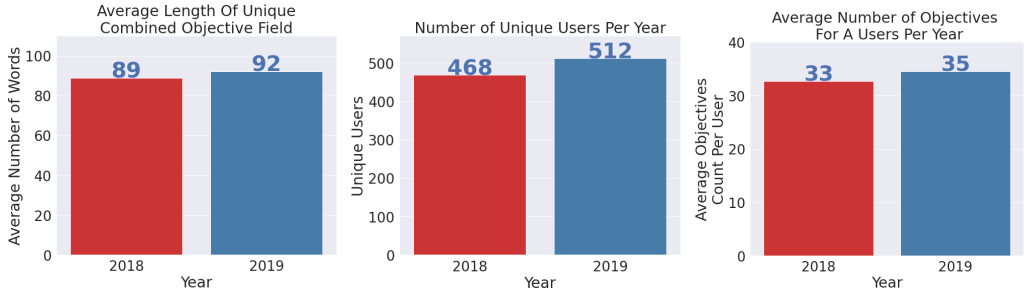
(a) (b)

Figure 2: Average number of words in (a) 2018 and (b) 2019 data. A full employee objective is spread along the three columns on the x-axis: *Objective Name*, *Objective Description*, and *Objective Metric*. The distribution of the number of words is almost the same in both years. Most of the objective text is contained in *Objective Metric*.



(a) (b)

Figure 3: Number of unique entries in the *Combined Objective* column in (a) 2018 and (b) 2019 data. The distribution is almost the same in both years which indicates a high number of duplicate values in the data.



(a) (b) (c)

Figure 4: (a) The average length of the final *Unique Combined Objective* field obtained after combining different objective fields, removing duplicates, and choosing the longest combined objective for each user. (b) The number of unique users in each year counted as the number of unique *User ID* fields in the original data. (c) The average number of objectives for a user. This high number is caused by duplicates, near-duplicates, and multiple objectives for the same user.

exact and near-duplicates, objectives spread across multiple columns, and multiple personal objectives as users are free to write multiple personal objectives. We notice that the average number of objectives per user is high in both years as shown in Figure 4b and 4c. For the purpose of our system, we assume a user can only have one objective, and we choose the objective with the maximum number of words to be the unique employee objective.

5. Final Unique Entries

After combining the different objective columns into the *Combined Objective* column, and selecting a unique objective per user, the final cleaned data set has 512 records for 2019, and 468 for 2018 as shown in Figure 4b. The average length of the *Unique Combined Objective* column is shown in Figure 4a.

The number of unique objectives is much less than the original size of the provided data, but that's due to the high number of duplication and missing values in the original data. We pick the 512 records of 2019 and proceed with it in building the system. The system can also be run with 2018 or any other data that follows the same format, that is a single objective per column. We also test the system on open source job description data[1] from Kaggle after putting it into the required format and discuss the results in section 5.1.

3.2 Pretrained Embedding Models

TensorFlow Hub[2] and Hugging Face Transformers[3] are two portals that provide pretrained text embedding models with various configurations. In this section, we discuss the imple-

1. https://www.kaggle.com/bman93/dataset
2. https://www.tensorflow.org/hub
3. https://huggingface.co/transformers/

mentations of the text embedding models mentioned previously in section 2.2 that we used in our system.

3.2.1 Word-Based

To get the representation of a complete employee objective in vector space using a word-based embedding model, we embed each word in the objective in vector space and take their average as the objective embedding.

- **Neural Network Language model:** Implementation from TensorFlow Hub[4]. The model is pretrained on the English Google News 200B corpus, and it maps words to a 128-dimensional numerical vectors.

- **word2vec:** Implementation as skipgram model from TensorFlow Hub[5]. The model is pretrained on the English Wikipedia corpus[6](Mikolov et al., 2013a), and maps words to a 500-dimensional numerical vectors.

- **ELMo:** Implementation from the TensorFlow Hub[7]. The model is pretrained on the 1 Billion Word Benchmark and maps words to a 1024-dimensional numerical vectors.

- **BERT:** Implementation from Hugging Face[8]. The model is pretrained on English Wikipedia[6] and the BookCorpus dataset[9] which consists of 11,038 unpublished books, and maps words to a 1024-dimensional numerical vectors.

- **ALBERT:** Implementation from Hugging Face[10]. Maps words to a 4096-dimensional numerical vectors.

- **DistilBERT:** Implementation from Hugging Face[11]. The model is pretrained on the same data as Bert which is English Wikipedia[6] and the BookCorpus dataset[9] which consists of 11,038 unpublished books, and maps words to a 768-dimensional numerical vectors.

- **RoBERTa:** Implementation from Hugging Face[12]. The model is pretrained on a union of five datasets comprising 160GB of text: BookCorpus[9], English Wikipedia[6], CC-News[13], OpenWebText[14], and Stories[15]. Maps words to a 1024-dimensional numerical vectors.

4. https://tfhub.dev/google/nnlm-en-dim128/2
5. https://tfhub.dev/google/Wiki-words-500/2
6. https://en.wikipedia.org/wiki/English$_W ikipedia/?oldformat = true$
7. https://tfhub.dev/google/elmo/3
8. https://huggingface.co/bert-large-uncased
9. https://yknzhu.wixsite.com/mbweb
10. https://huggingface.co/albert-xxlarge-v2
11. https://huggingface.co/distilbert-base-uncased
12. https://huggingface.co/roberta-base
13. https://commoncrawl.org/2016/10/news-dataset-available/
14. https://github.com/jcpeterson/openwebtext
15. https://arxiv.org/abs/1806.02847

- **GPT:** Implementation from Hugging Face[16]. The model is pretrained on the BookCorpus[9] and maps words to a 768-dimensional numerical vector.
- **GPT2:** Implementation from Hugging Face[17]. Maps words to a 768-dimensional numerical vectors.
- **XLNet:** Implementation from Hugging Face[18]. Maps words to a 1024-dimensional numerical vectors.
- **XLM:** Implementation from Hugging Face[19]. Maps words to a 2048-dimensional numerical vectors.
- **Reformer:** Implementation from Hugging Face[20]. Maps words to a 256-dimensional numerical vectors.
- **ELECTRA:** Implementation from Hugging Face[21]. Maps words to a 256-dimensional numerical vectors.
- **T5:** Implementation of T5 from Hugging Face[22]. Maps words to a 512-dimensional numerical vector.

3.2.2 Sentence Based

To get the representation of an employee objective in vector space using a sentence based embedding model, we simply pass the whole objective into a sentence based embedding model

- **Universal Sentence Encoder:** Implementation as Deep Averaging Network Encoder from TensorFlow Hub[23]. Maps a full sentence to a 512-dimensional numerical vector.
- **Sentence Transformers:** We use two models in our system, namely Sentence BERT and Sentence DistilBERT titled 'bert-large-nli-stsb-mean-tokens' and 'distilbert-base-nli-stsb-mean-tokens' respectively. The implementation comes from the sentence-transformers library[24]. Sentence BERT and Sentence DistilBERT map a sentence to a 1024-dimensional and a 768-dimensional numerical vector respectively.

3.3 Distance Metrics

In the previous section we showed how to represent employee objectives as numerical vectors in the embedding space. Motivated by the distributional hypothesis, the semantic relatedness between two employee objectives is the distance between their vector representations.

16. https://huggingface.co/openai-gpt
17. https://huggingface.co/gpt2
18. https://huggingface.co/xlnet-large-cased
19. https://huggingface.co/xlm-mlm-en-2048
20. https://huggingface.co/google/reformer-crime-and-punishment
21. https://huggingface.co/google/electra-small-discriminator
22. https://huggingface.co/t5-base
23. https://tfhub.dev/google/universal-sentence-encoder/4
24. https://github.com/UKPLab/sentence-transformers

	Distance Metrics	
1	Cosine Distance	$1 - \frac{u \cdot v}{\|\|u\|\|_2 \|\|v\|\|_2}$
2	Minkowski Distance($p = 2, w_i = 1$)	$(\sum \|u_i - v_i\|^p)^{\frac{1}{p}} \cdot (\sum w_i(\|(u_i - v_i)\|^p))^{\frac{1}{p}}$
3	Correlation	$1 - \frac{(u-\bar{u}) \cdot (v-\bar{v})}{\|\|(u-\bar{u})\|\|_2 \|\|(v-\bar{v})\|\|_2}$
4	City Block Distance	$\sum_i \|u_i - v_i\|$
5	Euclidean Distance	$\|\|u - v\|\|_2$
6	Square Euclidean Distance	$\|\|u - v\|\|_2^2$
7	Chebyshev Distance	$\max_i \|u_i - v_i\|$
8	Canberra Distance	$\sum_i \frac{\|u_i - v_i\|}{\|u_i\| + \|v_i\|}$
9	Braycurtis Distance	$\frac{\sum \|u_i - v_i\|}{\sum u_i + v_i}$

Table 1: Distance metrics supported by our system.

The distance metrics that are supported by our system are shown in Table 1. The reason we support many variations of embedding algorithms and distance metrics is to provide the data science team in the HR department of Merck (Which will be using the system) with a flexible system that they can use for running multiple experiments using multiple configurations.

4. Implementation

This section discusses the different components of the system, and presents a high-level workflow of how the system is used by the HR department of Merck. The system consists mainly of three components: the embeddings core, the back-end server, and the front-end server. The three components were packaged in a docker container and delivered to Merck as a docker image through a DockerHub private repository. Figure 5 shows the general flow of the system. In the next subsections, each component is explained in detail.

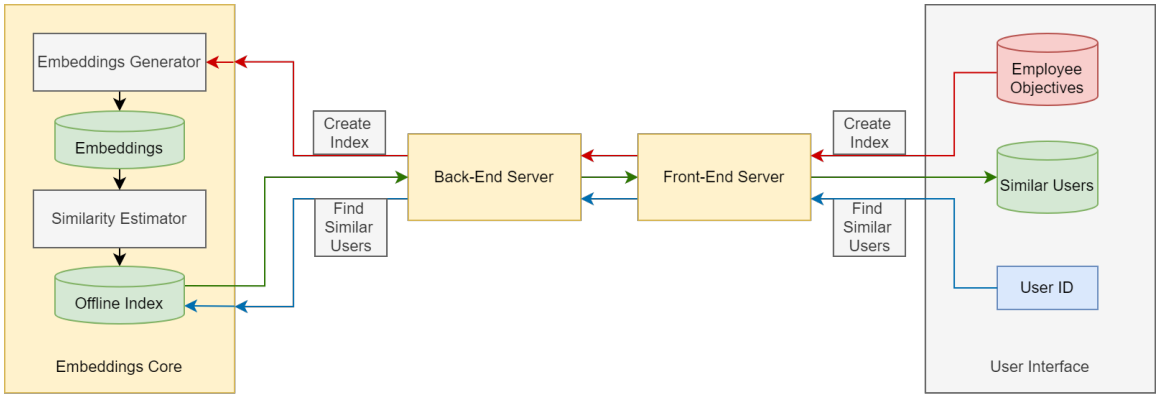


Figure 5: General system flow. **The red path:** represents system flow at indexing time. The user uploads employee objectives (red object) through the user interface. The front-end server sends the employee objectives data to the back-end server. The back-end server communicates with embeddings core to create embeddings, calculate pairwise distance, and save data offline for later querying. **The blue Path:** represents system flow at querying time. The user enters an employee-id in the user interface. The front-end sends a request to the back-end which queries the offline index for similar users. **The green path:** represents system-generated objects at index time (embeddings, and offline index) and returned results at querying time. For a given employee-id and a number N, the system returns the nearest N employees according to the semantic similarity of their objectives.

4.1 Embeddings Core

The embeddings core contains the main functionality of the system. It is technically a wrapper around two modules: Embeddings Generator, and Similarity Estimator. The embeddings core is responsible for the whole process of generating embeddings from textual objectives and finding the users with the most similar objectives to a given user. The constituents of embeddings core are explained below:

4.1.1 Embeddings Generator

The embeddings generator has instances of all the embedding algorithms mentioned in section 3.2. The main purpose of the embeddings generator simply is to read a file containing employee objectives, generate numerical objectives for each objective using the desired embedding algorithm, and saves these embeddings to desk. We use HDFstore to handle saving and accessing embeddings data from and to desk. The HDFstore is partitioned into chunks, each chunk contains a list of employee objective embeddings, along with the corresponding employee ID. This process is explained in Figure 6.

4.1.2 Similarity Estimator

The similarity estimator implements all the distance functions discussed previously in section 3.3. The main purpose of the similarity estimator is to calculate the pairwise distance between all the employee objective embeddings, and store this information in an offline index so that it can be easily queried later on. The similarity estimator maintains a max

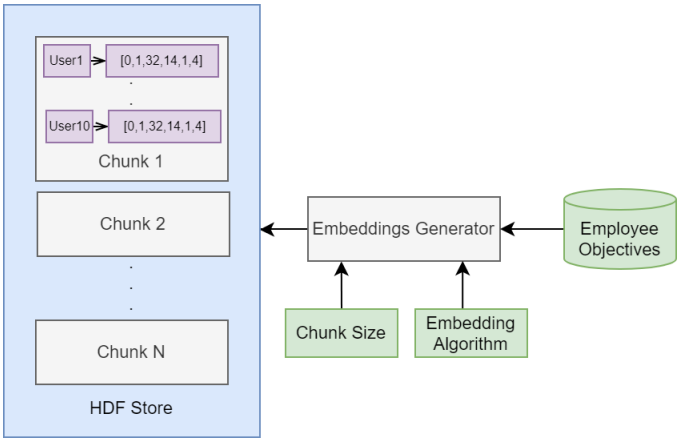


Figure 6: Embeddinga Generator. Takes employee objectives data as input along with embedding algorithm and chunk size. Writes embedded objectives into HDF Store.

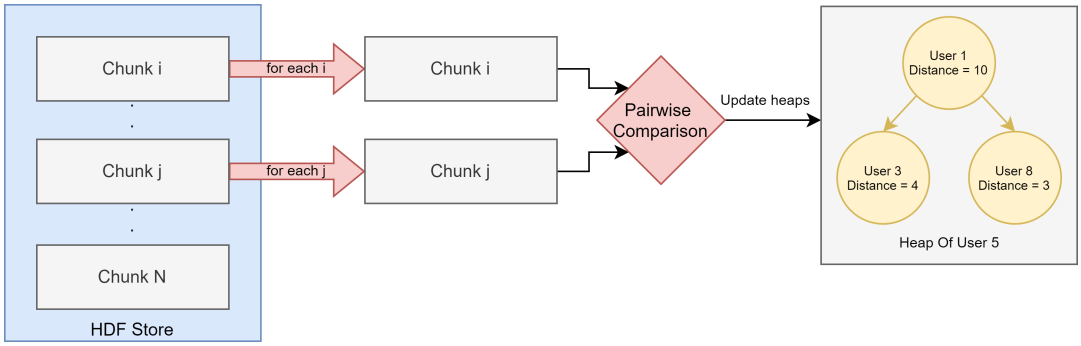


Figure 7: Similarity Estimator. Performs a double loop over the chunks to compute the pairwise similarity between embedded employee objectives in those chunks, and updates employee-specific heaps. The employee-specific heap will contain the nearest K employees after the double loops are over.

heap of size K for each target user indexed by the target user ID. The nodes in this will contain distance between the embeddings of the target user's objective, and the embeddings of other user's objectives. The similarity estimator then proceeds in a double loop over all chunks, and for $chunk_i$ and $chunk_j$ the pairwise distance is calculated for all employee objective embeddings in those two chunks, and the max heaps are updated accordingly, this is illustrated in Figure 7. After the double loop pass, each heap will contain only K user ids which are the user ids with the smallest distance to the target user. The heaps are persisted on disk, and the K user ids inside the heaps will be returned when similar users to the target user are requested.

4.2 Back-End Server

The back-end server is a wrapper around the functionality provided by the embeddings core. The back-end server is a flask server and provides the following endpoints:

1. **login:** Authentication functionality. Accepts *username* and *password* as request parameters. returns *200* and *403* for the cases of a successful login and incorrect credentials respectively. in case of a successful login, the user is redirected to their homepage by the front-end server.

2. **signup:** Registration functionality. Allows the creation of elevated and regular users. Returns *200* and *403* in cases of successful new user creation and 'username' exists respectively. in case of a successful new user creation, the user is redirected to their homepage by the front-end server, else the front-end displays a dialog box with error information.

3. **generate_models:** Indexing Functionality. Creates the offline index which is technically a map from each user id in the dataset to their corresponding nearest K user ids (represented as a map of the heaps described in section 4.1.2). The offline index is used during the querying phase for quicker response time (as compared to generating and comparing embeddings on the fly). This call also fits a TF-IDF model on the provided dataset, the TF-IDF model is used during querying time to identify important keywords in the target user's objective, and in the resulting employee objectives which serves as one of the explainability features in our system. The indexing functionality is performed according to the following request parameters:

   - *file*: A csv file containing the employee objectives along with employee IDs. The file should be comprised of two columns, one for the employee ID, and the other for the employee Objective.
   - *id_key*: The name of the *ID column* in the employee objectives csv file.
   - *sentence_key*: The name of the *objective column* in the employee objectives csv file.
   - *max_k*: The indexed data is a map from each target user id to a number of nearest users ids. This parameter controls the number of nearest users to index for each target user. At query time, a different number of K nearest users can be sent in the request as long as it is less than *max_k*.
   - *models*: A list of embedding algorithms. A separate index for each (embedding algorithm, distance metric) will be created.
   - *metrics*: A list of distance functions. A separate index for each (embedding algorithm, distance metric) will be created.
   - *chunk_size*: If the provided data is large. Computing the embeddings for each user and comparing the similarity between these users will be memory exhausting. By providing the *chunk_size* attribute, the index can be created in chunks.

4. **get_nearest:** Querying Functionality. Finds K users with the most similar objectives to a target user given the *user id*. Search is performed according to the request parameters listed below:

   - *id*: The id of the target user.
   - K: The number of similar users to retrieve.

- *method*: The embedding algorithm to use when comparing the employee objectives.
- *metric*: The similarity metric to use when comparing the employee objectives.

in case of a successful request, the back-end server replies with a JSON object with the following attributes:

- *userObjectives*: The personal objective of the target user.
- *objectives*: A list of tuples containing the nearest k users to the target user. Each element in the list represents a single result and consists of: user id, employee objective, and percentage of similarity.
- *importantWords*: Important words found in the target employee objective and the resulting users objectives. The TF-IDF model which was fitted during the *generate_models* call is used to identify important words. Important words are obtained by querying the TF-IDF model for unigrams, bigrams and trigrams of the highest frequencies.
- *commonWords*: Common words are bigrams and trigrams that exist in both the target user's objective and the results users' objectives.

4.3 Front-End Server

The front-end is a React.js server which implements the user interface for interacting with the system. It provides the following functionalities:

1. **Login and Registration:** Users of the system can create new accounts and login to their created accounts using a regular login/sign up module shown in Figure 13. Accounts can either be admin or non admin accounts. This functionality calls the **login** and **signup** endpoints.

2. **Generating Indexes:** An admin account has the ability to generate indexes. The front-end provides a dialog box shown in Figure 20, where the admin user can upload the file which contains employees embeddings, choose which embedding algorithms to generate embeddings, choose which distance metrics to use when calculating similarity, and provide the other parameters required by the **generate_models** endpoints. This functionality is hidden from non-admin users.

3. **Querying:** Both admin users and non-admin users can use the querying functionality. This works by typing the target user id, choosing the embedding algorithm, the distance function, the number of desired users to retrieve, and hitting the search button as shown in Figure 14. This calls the **get_nearest** endpoint. The front-end then presents the retrieved information in a table which shows the retrieved users IDs, the percentage of similarity with the target user, important and common keywords for each result, and the resulting users' objectives as shown in Figure 15 and Figure 17. The resulting objectives are also highlighted with their common and important words accordingly. The admin user can see all the previously mentioned information, while the non-admin user can only see the IDs, percentage of similarity, and keywords

as shown in Figure 16, and Figure 18. Non-admin users can not see employees' objectives so as not to violate confidentiality of the employees. However, a non-admin user can still get an idea of why the system suggests a specific user by observing the percentage of similarity and highlighted keywords.

4.4 System Workflow

The components of the system are containerized into a docker container which is delivered to Merck. The system is simply started by running the container locally or hosting it on a server. When a user starts the system, they are faced with the home screen shown in 13. The system supports two modes of users: Admin and Non-Admin users. Admin users have access to all the capabilities of the system, while non-admin users have limited access to ensure privacy of employee objectives stored in the system.

After login, a user is faced with the search screen shown in 14. Before performing search, data has to be indexed offline. Only admin users can index employee objectives data because non-admin users do not have access to employee objectives. Indexing is done by clicking on the left button in the top right corner. The embedding generation options dialog box appears as shown in 20. The dialog box has two tabs. The left tab required uploading the file containing the employee objectives, the file has to be a csv file with two columns: The first column should contain employees IDs. The second column should contain employees objectives. The default settings of the system is to use sentence DistilBERT as the embedding model, and cosine distance as the similarity metric. The right tab in the dialog box is for advanced users (Data scientists in the HR department). This tab enables users to experiment with different variations of embedding models and similarity metrics.

After data is generated. Both admin and non-admin users can search using the search box shown in 14. Search is performed by typing the target employee-id in the search box, choosing the embedding algorithm from the models drop down list, choosing the similarity metric from the metrics drop down list, entering the desired the number of similar employees to return, and finally clicking on the search button.

The search results for admins users appear as shown in 15 and 17. The admin user can see the objectives of the target and resulting users, as well as common and important words highlighted. For a non-admin user, results appear as shown in 16 and 18. The non-admin user can only see the keywords in the objectives, not the full text. The color coding of important and common words is an attempt to provide explainability for the results of the system. Common words are simply some of the common words between the target employee's objective and the resulting employees objectives. While important words are words which have high Tf-IDF scores.

5. Results and Discussion

The evaluation of the system was conducted as a survey. Eight employees from the HR department of Merck -the end users of the system- were asked to use the system and experiment with the different capabilities of the system, then take a questionnaire in which

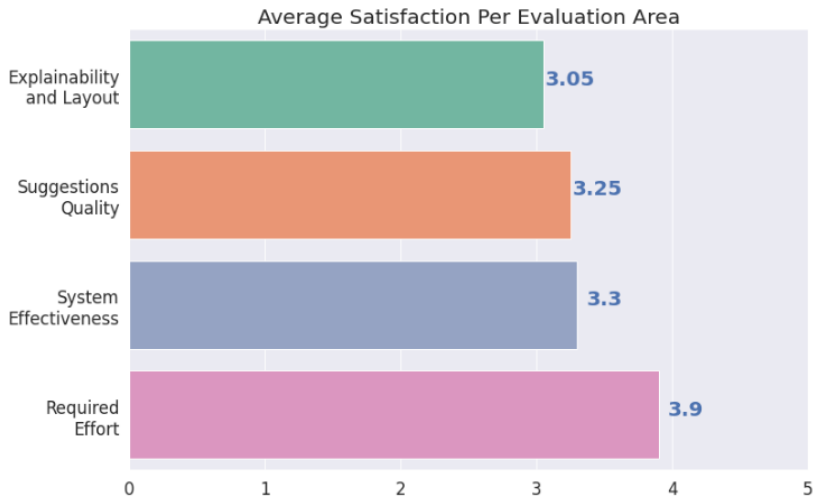


Figure 8: Scores of the evaluation survey averaged over evaluation area. The feedback states that the system requires little effort to use, while explainability features need to be improved.

they express their feedback. The questionnaire is composed of 20 statements evaluating four areas of interest: (1) Quality of suggestions, (2) Effectiveness of the system, (3) Effort to use the system, (4) Layout and explainability. Participants were asked to read each statement in the questionnaire and express their agreement/disagreement on a five point scale ranging from *strongly disagree* to *strongly agree*. The statements are shown in Figures 9, 10, 11, and 12.

The bar-plot in Figure 8 below summarizes the evaluation results for the different areas and the feedback we received from Merck. The evaluation area **Required Effort** received the highest average score from the users taking the questionnaire, the feedback we received was that the system was easy to use and navigate. The following second and third scores are for **System Effectiveness** and **Suggestions Quality**. The general feedback for these two areas is also good. The final evaluation area is **Explainability and Layout**, the evaluation for which was not as good as the previous areas, and the reason for this was that the explainability features were not clear, and the color coding for the important and common words was also confusing for some people.

5.1 Job Description Dataset

A quick sanity check was performed to ensure that the results of the system will generalize to different data from other domains. Since the goal of the system is to find semantically related pieces of text, we used the system to find similar job descriptions, for that we used the job description dataset from kaggle[25]. The dataset contains 72,292 entries, each entry contains a job description. There are 30 unique jobs in total in the dataset. To test the

25. https://www.kaggle.com/bman93/dataset

system, we used the dataset as input, created offline indices, then queried the system for various job description and manually observed the results. The results retrieved by the system were shown to be relevant to the query. An example of search results using the job descriptions datasets is shown in Figure 19.

6. Conclusion

In this work we implemented an end-to-end system to assist the HR department of Merck find and match employees with similar objectives in order for them to cooperate and assist one another. The system computes the pairwise semantic similarity of the employees objectives and saves the results in an offline index. Then, the system can be queried for the most similar N employees for a given employee. The semantic similarity between two employee objectives is computed as the distance between their numerical vector representation (embeddings). The system supports multiple embedding algorithms to compute the numerical representation of objectives. It also supports multiple distance metrics to compute the similarity between two embedded objectives.

The system consists mainly of three parts: (1) Embedding Core, (2) Back-End wrapper around embeddings Core, (3) Front-End for the user interface. The workflow of the system goes as follows: (1) login, (2) upload data and create index, (3) query for similar users. For a regular user, the system can be used with the default settings (embedding algorithm= sentenceDistilBert, distance metric=cosine), and the user in this case doesn't have to worry about choosing an embedding model or a similarity metric. However, there are data scientists in the HR department of Merck, and for these advanced users we provide the capability of playing around with different combination of embedding algorithms and distance metrics.

Furthermore, the back-end server and the front-end server of the system are completely decoupled, which means that the data scientists of Merck can integrate the back-end server as an external python module in their systems. When using the system through our Front-End, we provide two types of users (1) Admins and (2) Non-Admins. Admin users can exploit all the capabilities of the system, while non admin users have limited access so as not violate the confidentiality of employee objectives. For this we introduced the ct features, by which we use two color codes to highlight common and important words in objectives, this gives users of the system an idea of why the system made a specific suggestions, and for the case of non admin users we only show the keywords without showing the whole objective.

The system was delivered to Merck as a docker image through a private docker-hub repository. The evaluation of the system was conducted as a questionnaire. Employees from the HR department of Merck were asked to use the system and take the questionnaire. The main feedback for the system can be summarized as (1) The system is easy to use, (2) The suggestions of the system are fairly good. (3) The system improves the process of matching employees with similar objectives. (4) The explainability features of the system need to be improved. Finally, a future suggestion we received from Merck is to add the option to filter the results of the system based on key-words or concepts.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Francis C Fernández-Reyes and Suraj Shinde. Cv retrieval system based on job description matching using hybrid word embeddings. *Computer Speech & Language*, 56:73–79, 2019.

John R Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, 1957.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*, 2019.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Thomas K Landauer and Susan T Dumais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2):211, 1997.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

José Luís Fava de Matos Pombo. *Landing on the right job: a machine learning approach to match candidates with jobs applying semantic embeddings.* PhD thesis, 2019.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL `https://arxiv.org/abs/1908.10084`.

Nils Reimers and Iryna Gurevych. Making monolingual sentence embeddings multilingual using knowledge distillation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2020. URL `https://arxiv.org/abs/2004.09813`.

Victor Sanh. Smaller, faster, cheaper, lighter: Introducing distilbert, a distilled version of bert. *Medium (blog). August*, 28:2019, 2019.

Steffen Schnitzer, Dominik Reis, Wael Alkhatib, Christoph Rensing, and Ralf Steinmetz. Preselection of documents for personalized recommendations of job postings based on word embeddings. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 1683–1686, 2019.

Dengliang Shi. A study on neural network language modeling. *arXiv preprint arXiv:1708.07252*, 2017.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465, 2013a.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013b.

Le Van-Duyet, Vo Minh Quan, and Dang Quang An. Skill2vec: Machine learning approach for determining the relevant skills from job description. *arXiv preprint arXiv:1707.09751*, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30:5998–6008, 2017.

Tak-Lam Wong, Haoran Xie, Fu Lee Wang, Chung Keung Poon, and Di Zou. An automatic approach for discovering skill relationship from learning data. In *Proceedings of the Seventh International Learning Analytics & Knowledge Conference*, pages 608–609, 2017.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763, 2019.

Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th international conference on software engineering*, pages 404–415, 2016.

Jianbo Yuan, Walid Shalaby, Mohammed Korayem, David Lin, Khalifeh AlJadda, and Jiebo Luo. Solving cold-start problem in large-scale recommendation engines: A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1901–1910. IEEE, 2016.

Supplementary Material

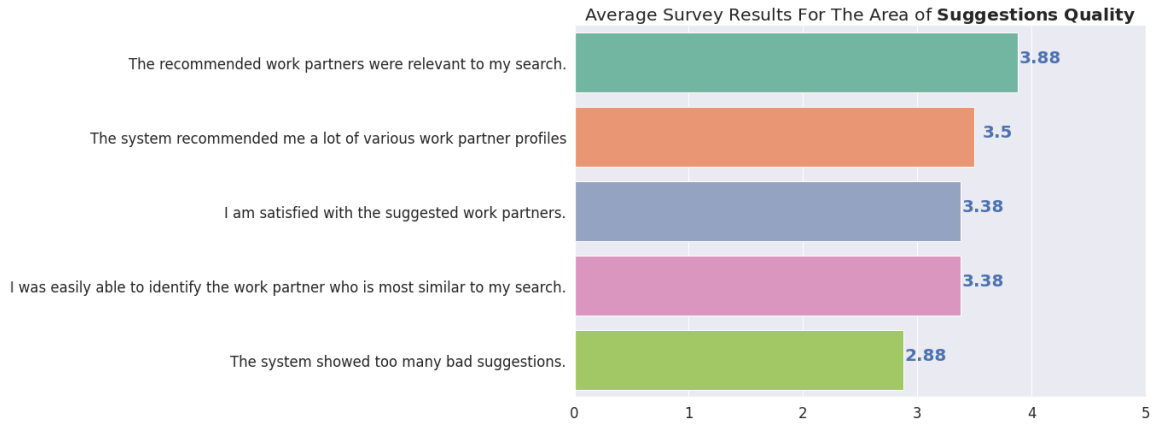


Figure 9: Average scores obtained on questions in the **Suggestions Quality** area.

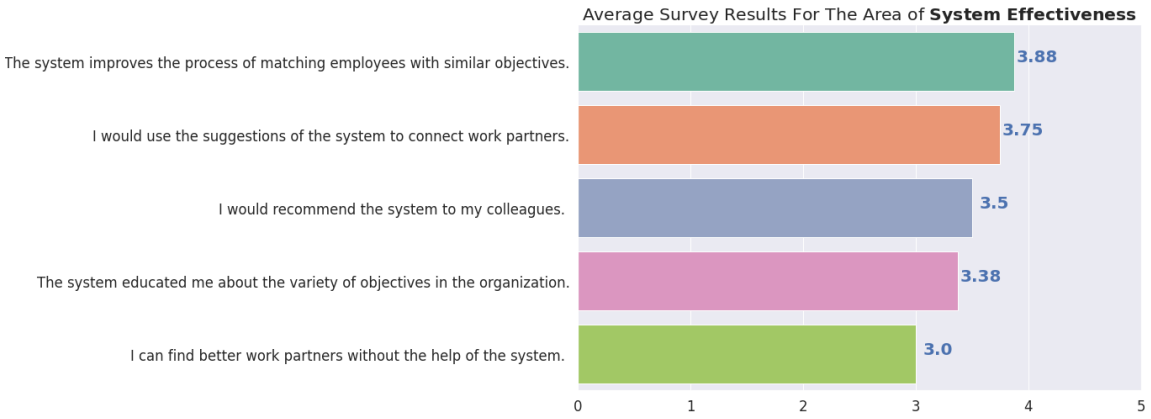


Figure 10: Average scores obtained on questions in the **System Effectiveness** area.

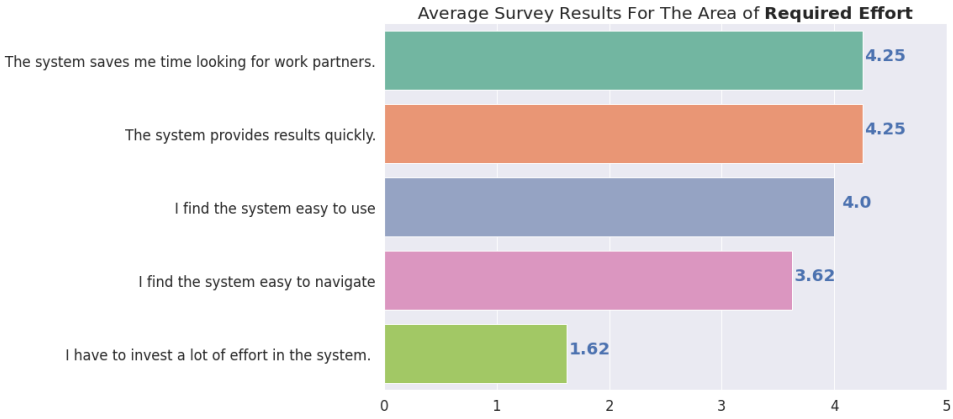


Figure 11: Average scores obtained on questions in the **Required Effort** area.

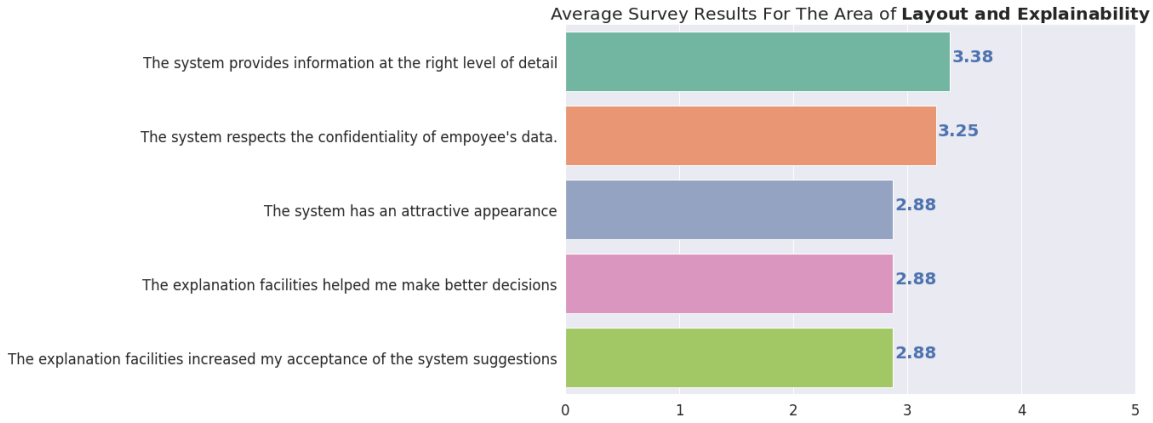


Figure 12: Average scores obtained on questions in the **Layout and Explainability** area.

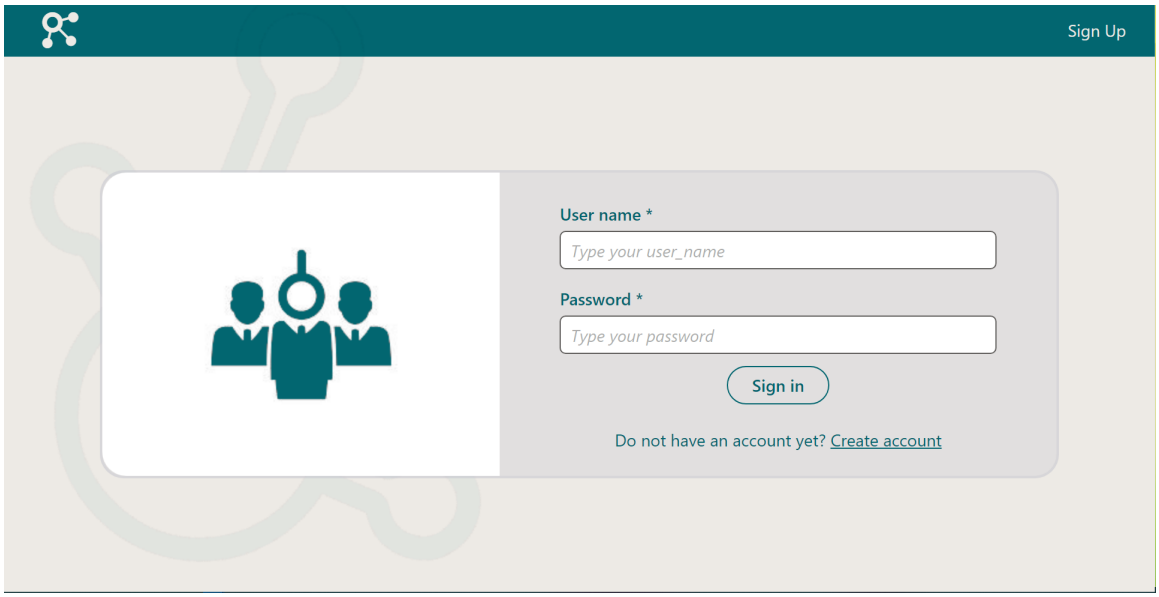


Figure 13: Login Screen.

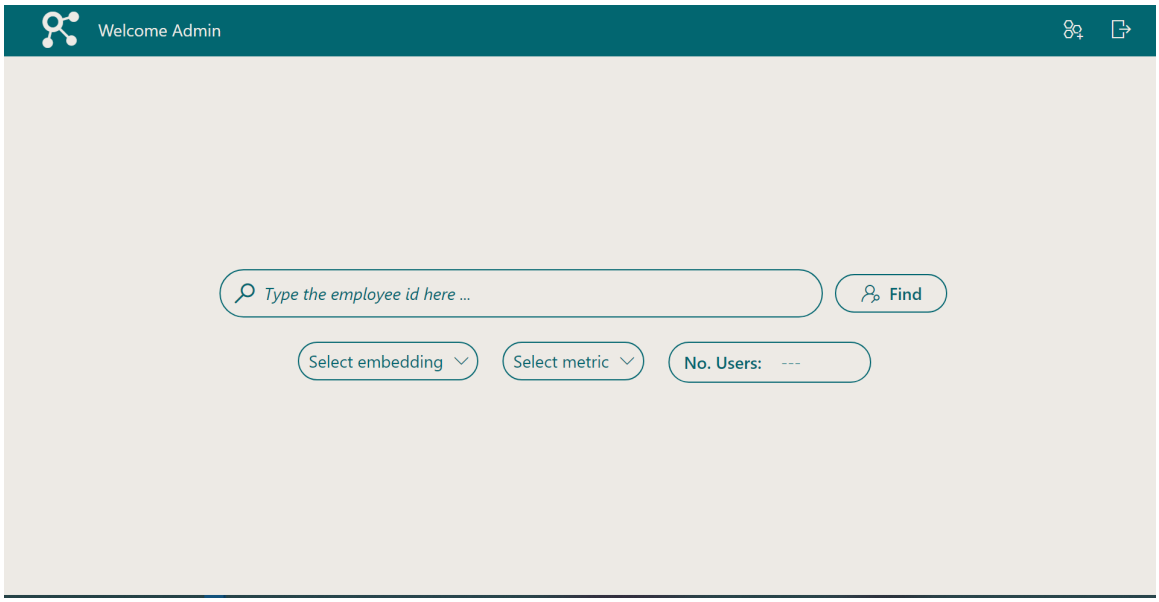


Figure 14: Search Screen. User types target employee-id in the search box, and chooses the desired combination of embedding model, similarity metric, and number of users to retrieve.

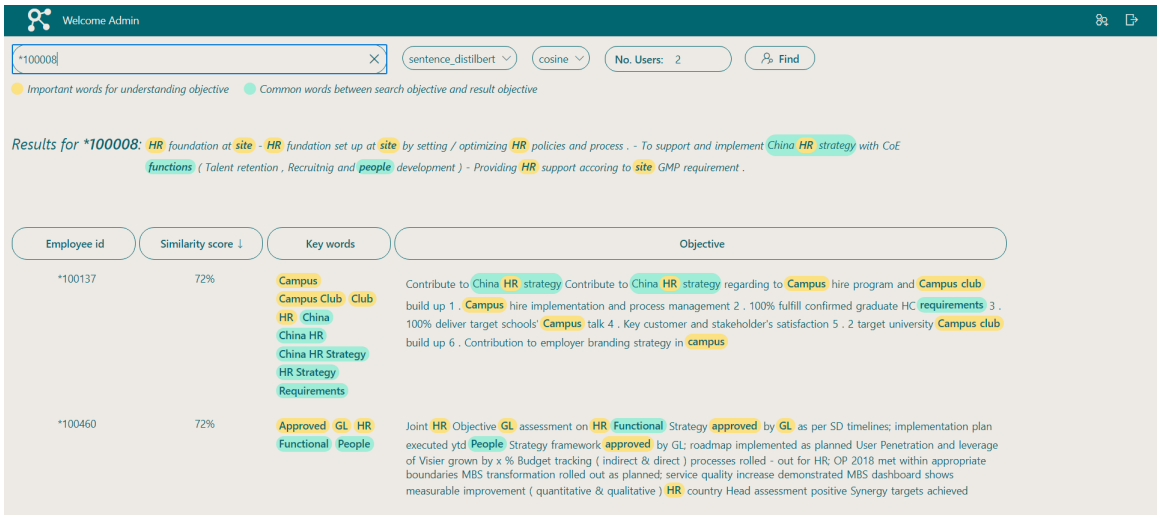


Figure 15: Search results as seen by an admin user. The target employee is *100008. Two resulting employees are shown. Important and common keywords are highlighted in yellow and green respectively

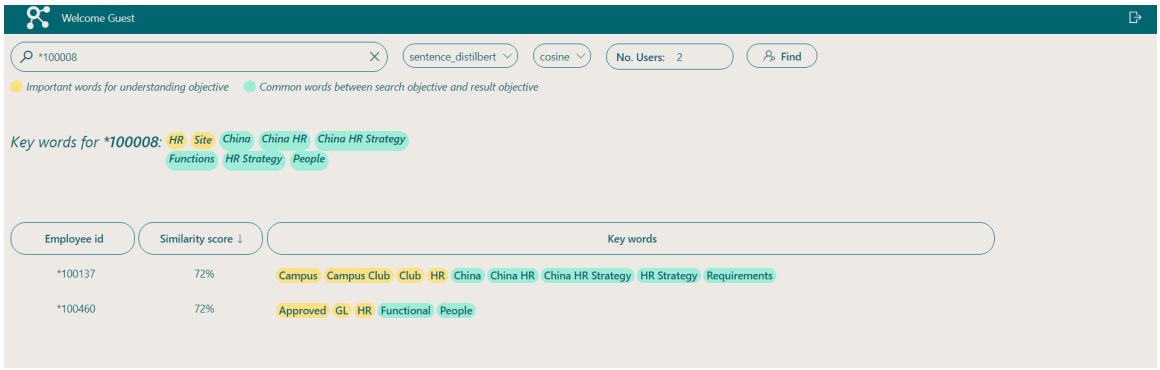


Figure 16: Search results as seen by a non-admin user. The target employee is *100008. Two resulting employees are shown. A non-admin user can only see the important and common keywords highlighted in yellow and green respectively

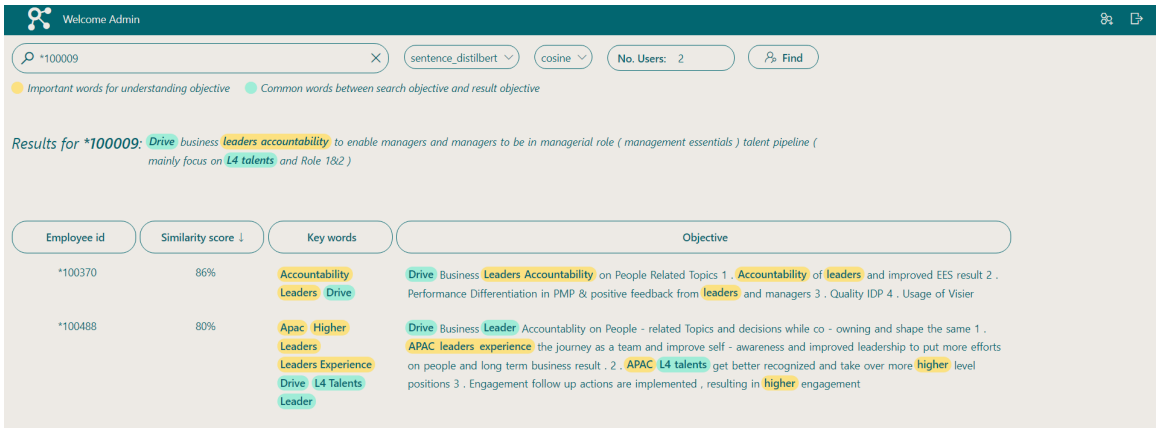


Figure 17: Search results as seen by an admin user. The target employee is *100009. Two resulting employees are shown. Important and common keywords are highlighted in yellow and green respectively.

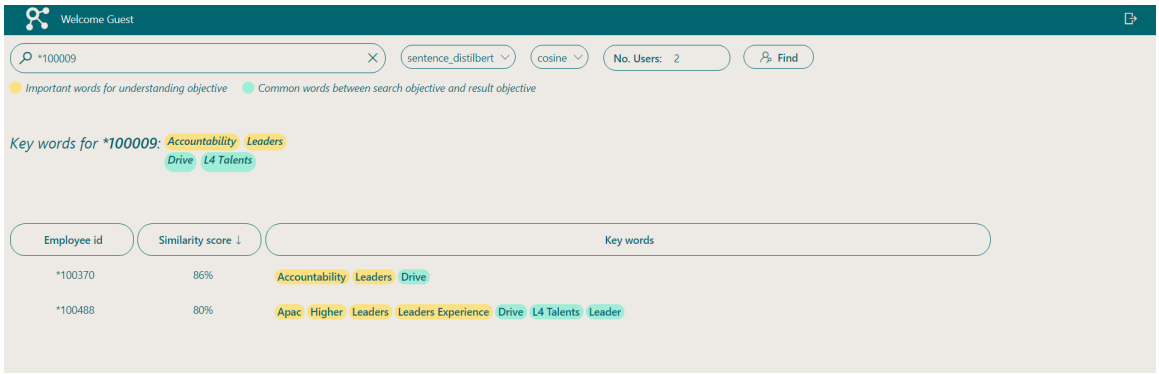


Figure 18: Search results as seen by a non-admin user. The target employee is *100009. Two resulting employees are shown. A non-admin user can only see the important and common keywords highlighted in yellow and green respectively.

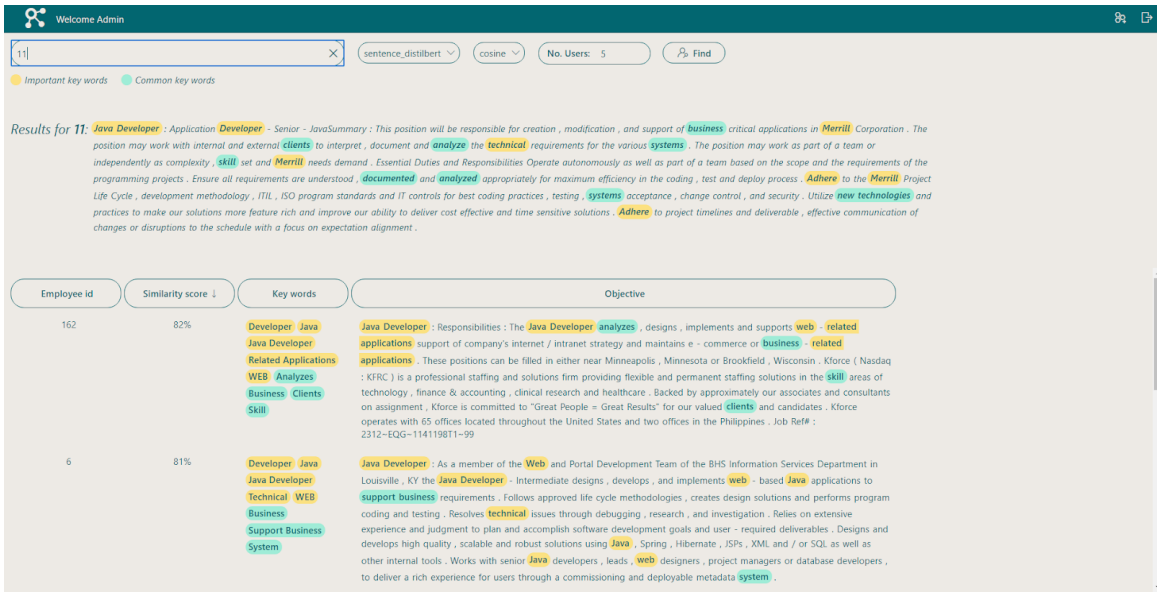


Figure 19: A demonstration of how the system performs on the Job Descriptions dataset from Kaggle. The target query is about a Java Developer. The results are semantically similar with high similarity score.

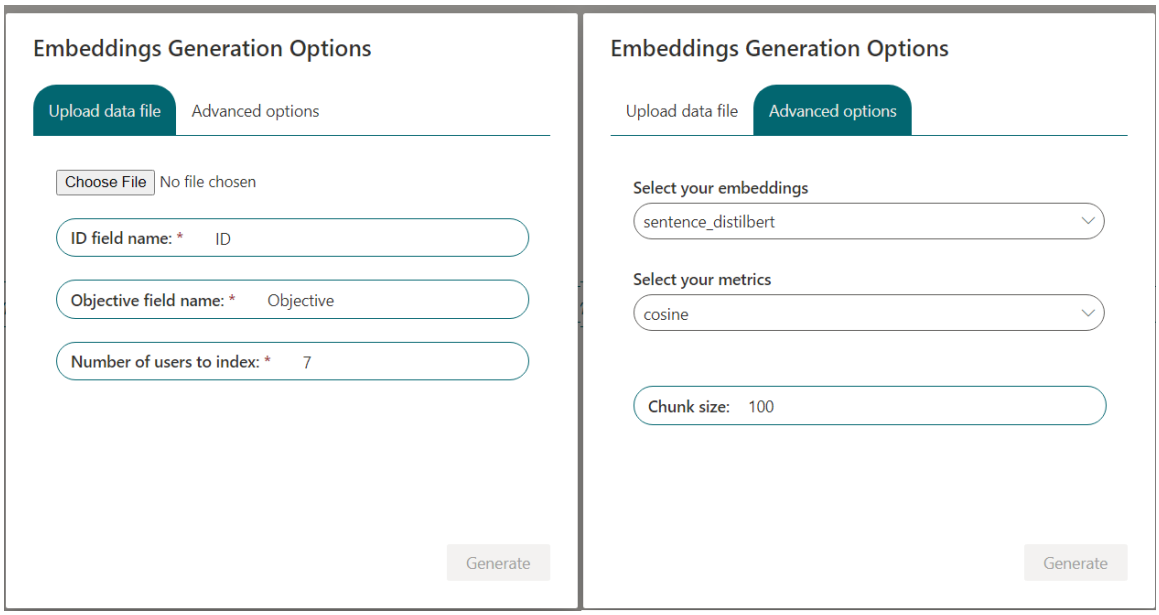


Figure 20: Embedding Generation Options. **Left**: regular users tab, where a user is required to upload the csv file which contains the employee objectives. **Right**: Advanced users tab, where a user can choose different combinations of embedding models and distance metrics.