Reproducibility Award Methodology Description

Methodology

Please provide a detailed description of the methodology used for identifying different types of duplicates in the online job advertisement data set. The description should contain (1) the data processing steps, (2) the methods and models used, (3) references to the scientific papers/sources that present the methods and models used, and (4) the time it took to process the data set and identify the duplicates.

Bear in mind that the workflow will be also evaluated by its originality, interpretability, and simplicity of the methodology.

Overview

The deduplication challenge required us to identify duplicated job advertisements in a dataset of web-scraped postings. A traditional way of utilizing Natural Language Processing (NLP) for problems of economic relevance, was the so-called dictionary-method that considers text inputs as a bag-of-words *(Loughran & McDonald, 2011)*. For our deduplication task, this for instance, could mean that one measures the overlap of words between two job advertisements. Yet, a major challenge for the dataset at hand, is that many duplicates were not identical, but instead were even written in different languages or had slightly different wording. For example, even though being in the same language, the titles “datastore manager” and “database administrator” may describe the same job but none of their words overlap.

However, in recent years, there have been remarkable advancements in the field of NLP. With a shift away from rule-based language analysis to larger neural networks, modern language models can now capture a more comprehensive understanding of language. A standout development in this area is Google's neural network, BERT (Bidirectional Encoder Representations from Transformers), which was published in 2018 *(Devlin et al. 2018)* and relies on the transformers architecture *(Vaswani et al. 2017)*. BERT and other similar transformer-based language models have made significant progress in various NLP tasks and applications, including question-answering, fact-checking, and hate-speech detection, setting new high scores. *(González-Carvajal & Garrido-Merchán, 2020)*.

Therefore, for our solution, we utilize transformer-based language models to create embeddings for our text input, that is, complex vector representations of texts *(Reimers & Gurevych 2019)*, and in turn, measure the similarity between those text embeddings. This allows us to capture postings that describe the same job position but do not use the exact same words.

However, applying transformer-based models is fairly computationally intense, and with approximately 120,000 job advertisements, comparing each posting against every other possible match would be computationally infeasible. Therefore, we utilize FAISS (Fast AI Similarity Search) index search, a scalable vector search engine *(Johnson, Douze & Jégou, 2019)*, and we efficiently search for the 100 closest matches for each job description among all others, reducing the complexity of the problem by over 99%. With this pre-filtered dataset that we generate, we classify observation tuples as semantic, temporal, or partial duplicates using some more detailed rules.

In our final solution, before we utilize transformer models and FAISS, we translate all text inputs into English, and apply an English language model *(Song et al. 2020)*, but we also present an alternative approach that does not use translations and instead applies a multilingual text model instead *(Reimers*

& Gurevych 2019). We discuss the advantages and disadvantages of these two approaches in the section “Lessons Learned”.

The entire procedure is written purely in Python and utilizes some of the latest machine learning libraries available. This procedure has benefited greatly from being executed on GPU machines.

Below you find a more detailed description of our methodology and the different steps that we apply.

Steps and procedures

- ***1. Find perfect matches***
  Identifying all FULL and (full) TEMPORAL duplicates proved to be straightforward, as translations or advanced NLP techniques were unnecessary. To separate these easily identifiable matches from the rest of our procedure, we defined a function named *find_perfect_duplicates()*. After minimal cleaning, the function simply detects all job postings that have identical descriptions and titles (as well as retrieval dates for the FULL duplicates). Checking whether two strings are identical (rather than just similar) is a simple operation in terms of computational cost, and it is done in under two minutes even on retail-grade computers. The final output of this function is csv file with the detected FULL and (full) TEMPORAL duplicates’ ID combinations.

  Execution time: **Less than 1 minute**

- ***2. Clean raw data***
  To identify the less trivial cases of TEMPORAL, PARTIAL and SEMANTIC duplicates, the raw dataset has to be cleaned in order to reduce noise. This is done with a text pre-processing function named *clean_input_dataset().* This function again, applies a first layer of minimal cleaning like above, and later proceeds with some more heavy cleaning operations:
  - Removes HTML tags coming from the web-scraping process (e.g,
 ,).
  - Converts all HTML character references to the equivalent ASCII characters.
  - Keeps only ASCII characters and specific punctuations.
  - Removes unnecessary whitespaces.
  - Splits lowercase characters that come before uppercase characters.
  - Removes repeated punctuations.

  Finally, the function converts the data types to the optimal ones. The output is a parquet file with the cleaned data.This cleaning step is important as just due to simply cleaning steps we are only left with around 61,500 (compared to the 112,000 in the original dataset).

  Execution time: **Less than 2 minutes**

- ***3. Prepare input for embedding (with optional translations)***
  Translating the dataset is an optional step, but highly recommended in order to achieve a higher accuracy. We define a function named *prepare_input_data_for_embedding()* which takes care of preparing the data for the embedding process that follows next. If the translation parameter is set to *True*, the function translates the cleaned job descriptions and titles into English. For that, the Google API Translator is used. Using the Google API Translator is regarded as a suitable intermediate step for NLP tasks, even for languages that are semantically different to English (Ramadasa et al. 2022). To handle the translation process, a helper function named *translate()* is defined. It takes a string as input and returns the corresponding string in English, allowing for a maximum of 5 attempts in case the API fails. The main function reads the cleaned data, and translates only unique descriptions and titles in order to optimize the process.

Lastly, it merges title and description into a new column, assigning a weight to the title, since titles are usually shorter, but relatively more important in order to identify duplicated job postings. For that, a helper function named *merge_title_and_description_with_weights()* is defined. It takes the title and description strings and a desired weight on the title as parameters and returns the merged string. In case the translation parameter is set to *False*, this helper function is applied directly on the cleaned data maintaining the original languages. The output of this function is a parquet file that also includes the new column (created based on the translated or the original cleaned strings).

Execution time with Method 1 (Translating): **Approximately 8 hours**
Execution time with Method 2 (Not translating): **Less than 1 minute**

- ***4. Use Transformer models to create overall embeddings***

To tackle the challenge of identifying duplicates within a dataset of job postings, it is necessary to represent each posting numerically. This is achieved through the use of an embedding model, which is capable of capturing the semantic meaning of job titles and descriptions. In our implementation, we utilize the SentenceTransformer library and specifically the *all-mpnet-base-v2* model. The all-* models have been trained on over one billion training pairs and are designed as general-purpose models. Among these models, *all-mpnet-base-v2* stands out with the highest quality, making it the ideal choice for our utilization (*Reimers, 2023*). In case of skipping the translations step, *distiluse-base-multilingual-cased* and *paraphrase-multilingual-MiniLM-L12-v2* seem to be fairly good model choices to partially solve the multilinguality issue.

The *create_overall_embedding()* function is responsible for generating an overall embedding for each job posting using the chosen embedding model (mpnet preferably). Initially, it reads the parquet file from the previous step. The next step is to remove any actual fully duplicated job postings, since there is no point in computing embeddings for identical postings. To do this, the function creates a mapping between each job posting's title and description accompanied by a unique hash value, and then drops any duplicates based on this hash. The function then creates a corpus of all unique job descriptions, and applies the specified embedding model to this corpus in order to generate overall embeddings for each description. The resulting embeddings are saved to the disk as a Tensor, along with a mapping between each job posting's title and description and its corresponding hash value. These embeddings are finally saved in a parquet file, along with additional information to be used later.

This step is vital in identifying duplicates, as it enables us to compare the similarity between job postings in a high-dimensional numerical space. A potential shortcoming of transformer-based models is their scalability, as calculating complex vector similarities between over 6 billion pairs of job postings would be computationally infeasible. Therefore, we rely on FAISS, a scalable open-source search engine, developed by *Facebook Research* that greatly improves the performance for complex similarity searches *(Johnson, Douze & Jégou, 2019).*

We apply FAISS to generate a pre-filtered set of potential duplicate-pairs. With the over 6 billion potential duplicate pairs, finding duplicates appears like searching a needle in a haystack. FAISS allows us to find similar job postings for all advertisements in a reasonable amount of time while still referring to the complex embedding representations generated by our transformer-model.

In order to achieve this type of search, a new function named *perform_index_search()* is defined, which takes the embedding corpus from the previous step, the mapping of text descriptions to their corresponding hash values, the name of the language embedding model, and the desired number of nearest neighbors as parameters. It converts the embedding corpus to a NumPy array and creates an index object in FAISS. Then, it adds the corpus embeddings to the index with their corresponding hash values. The function initializes an empty dictionary

to store the results and iterates over each text description hash in the mapping. For each hash value, it retrieves the corresponding embedding from the index and retrieves the k-nearest neighbors (kNN) in the index using FAISS. By increasing the value of kNN, more duplicates can potentially be captured, however since time efficiency is also an important factor, the kNN parameter was found to be balanced at *k = 100* after testing different values. Finally, the function stores the results in the dictionary.

This step is implemented by defining a function named *use_embedding()*, which reads the mapping of text descriptions to their corresponding hash values and the pre-computed embeddings. It calls the *perform_index_search()* function to find the *k = 100* nearest neighbors of each text description hash in the embedding corpus. Then, it stores the results in a dictionary using the pickle library.

Finally, a function named *merge_results_to_dataframe()* is defined, which merges the pre-filtered set of potential duplicate pairs with the full set of job postings to create a single dataframe containing all potential duplicates. It uses the Pandas merge function to join the pre-filtered set on both the left and right sides with the full set of job postings using their unique hash values. The resulting dataframe is saved to disk as a pickle file.

Overall, these functions play a crucial role in the pipeline by consolidating the results from the pre-filtering step and providing a single source of potential duplicates for further processing.

Execution time for overall embedding creation (a): **Approximately 18 minutes**
Execution time for overall kNN search (b): **Approximately 18 minutes**
Execution time for overall pre-filtered dataset reassembling (c): **Less than 1 minute**

Execution time for step 4 (total): **Approximately 37 minutes**

- ***5. Apply expert rules***

  In this last step, the final filtering and distinction between SEMANTIC, TEMPORAL and PARTIAL duplicates is performed. A function named *apply_expert_rules()* is defined for that purpose, but also for the creation of the final output file. Essentially, this is a wrapper for two other functions.

  The first one is the *prepare_input_for_expert_rules()* function, which prepares the dataset for the expert rules that follow. It reads the pre-filtered ID combinations and brings back all relevant fields like titles and descriptions for each ID pair, as well as the L2 distance from the previous step. Next, it creates some additional columns like descriptions string lengths and same title flags, and more importantly, it generates new cosine similarities between the translated or original titles (based on the chosen method from the previous steps).

  Continuing, the *apply_expert_rules_on_data()* function takes over. Initially, it performs some additional cleaning operations (e.g. VisiDarbi.lv) and applies the pre-defined expert rules. First, it detects all TEMPORAL duplicates. For that, it selects the pairs that have the same but not missing company names, and a title similarity above the *temporal_title_similarity_unrestricted* threshold, which is set to 0.8. On top of that, it also selects the pairs with a title similarity above the *temporal_title_similarity_restricted* threshold which is set to 0.92, which also have a L2 distance below *temporal_l2_threshold*, which is set to 0.1. After that, the function detects the SEMANTIC duplicates. For that, we select all the pairs with the same retrieval dates, excluding combinations with different countries, locations, or company names. Lastly, the function detects the PARTIAL duplicates, which are basically semantic duplicates with missing characteristics. In order to do that, three types of methods can be defined. The best one seemed to be the *tag* method which is set by default and it is explained below. Finally, the FULL and (full) TEMPORAL duplicates from step 1 are read, and all the dataframes are put together generating the final output file which exported as csv, with the desired format.

All these parameters can be set in the code (we have left our defaults included for reproducibility):

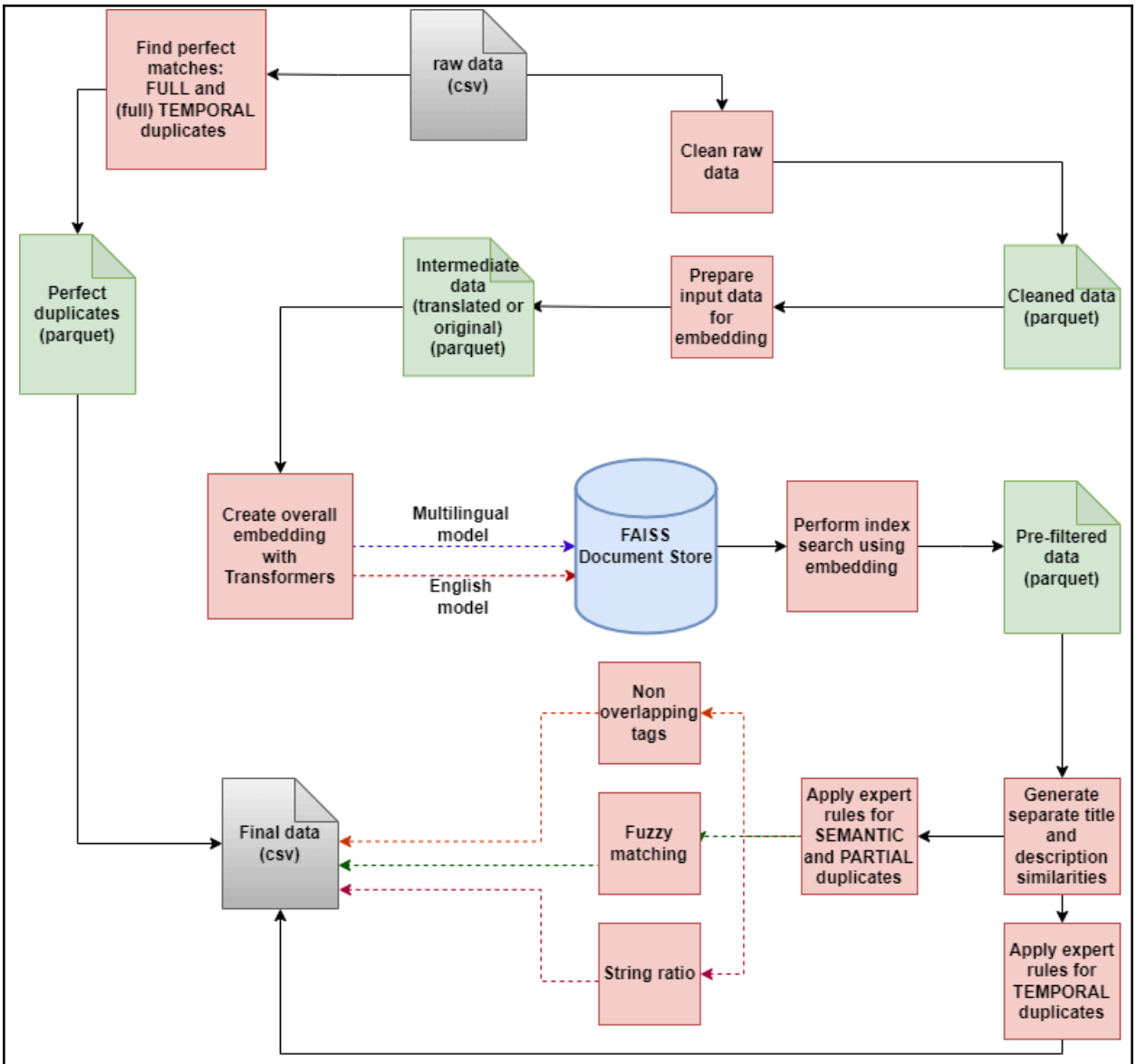
- **`language_embedding_model:`** previously selected model (keep the same)
- **`description_similarities`**: keep it equal to False (not used)
- **`partial_rule`**: Either **`tag`** (option 1), **`fuzzy`** (option 2) or **`string_ratio`** (option 3):
  - *Option 1*: We check if a group of word is present in one job descriptions but not in the other e.g. 'bachelor', 'master', 'degree' to identify partials. Tag is the best method so far.
  - *Option 2*: We measure the fuzzy ratio between the two descriptions.
  - *Option 3*: We measure the string ratio difference between the two descriptions

- **`temporal_title_similarity_unrestricted`**: lowest similarity two titles can have if the company is the same
- **`temporal_title_similarity_restricted`**: lowest similarity two titles can have if the company is different or missing
- **`temporal_l2_threshold`**: maximum l2 distance between two job advertisements if the company is different or missing

Execution time with Method tag: **Approximately 10 minutes**
Execution time with Method fuzzy: **Less than 10 minutes**
Execution time with Method string_ratio: **Less than 10 minutes**

Overall Execution Time

Method	**Runtime**	**Overall F1 Score**
Translate and tag	~ 9 hours	~ 82%
Multilanguage embedding and string ratios	~ 1 hour	~ 60%

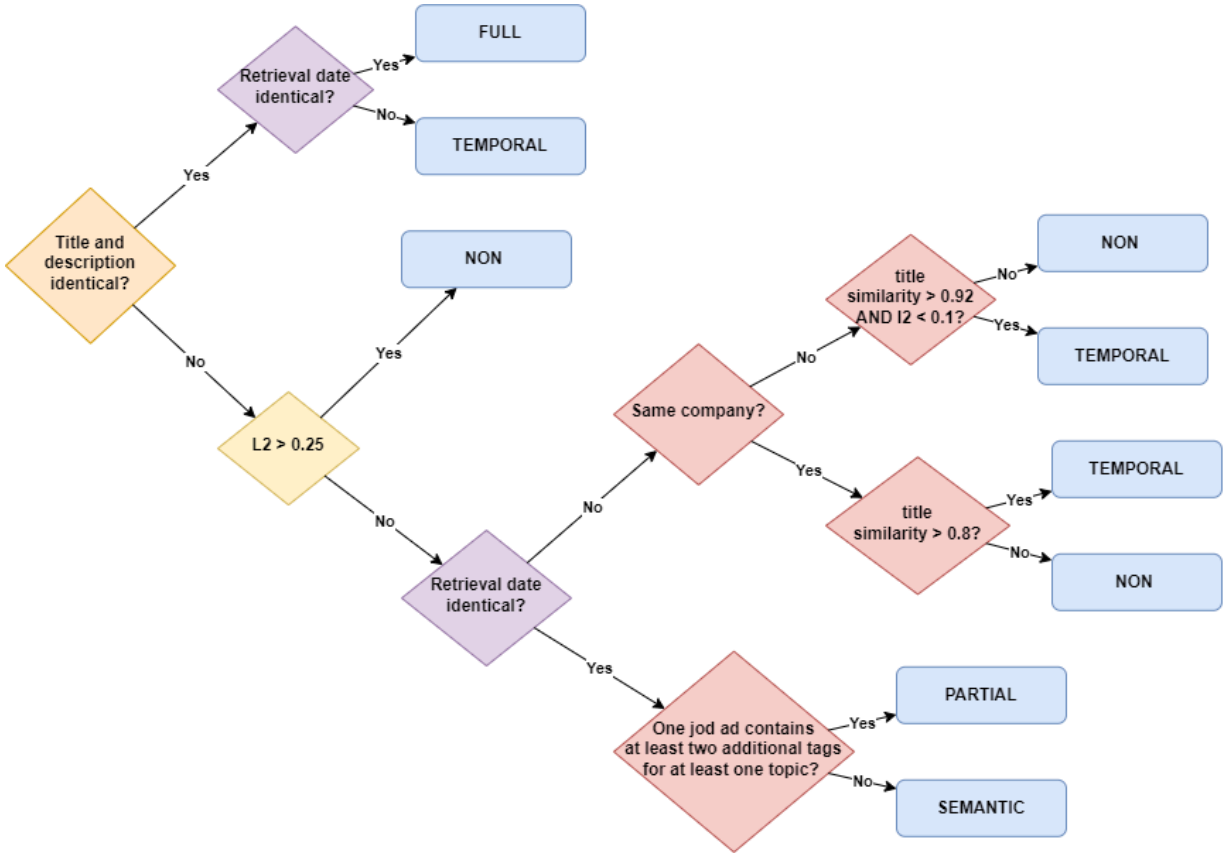


Interpretability of results

Interpretability is one of the key challenges in machine learning applications. One advantage of our approach is that we combine pre-trained language models with expert rules based systems that allows to preserve interpretability. Consider two job ids, *id1* and *id2*. Which features will make them more or less likely to be classified as semantic, temporal or partial matches? Lets go through our predictors one by one:

- **L2 distance:** The embedding model creates a complex vector representation for each text input of length 768. The L2-distance measures the Euclidian distance between the vector representations of two text inputs. Hence, the lower the L2-distance the higher the semantic similarity between two text inputs. Two identical text inputs would show an L2 distance of 0.
  If the L2 distance between title+description is larger than 0.25, then the observations will not be considered duplicates. While this creates some false negatives, it also massively reduces the complexity of the problem
- **Retrieval dates**: If the retrieval dates are not identical, then the two duplicates can only be temporal duplicates.

- **Company name:** Intuitively, if two similar worded job advertisements are posted by the same company, we consider it more likely that those are duplicates.
- **Title similarity:** The title similarity is computed as the cosine similarity between the embedding representation of the two (translated) job titles. A higher title similarity
- **Topics and Tags:** We identify topics (education, contract type, schedule, skills,...) and tags for each topic (e.g. for education: bachelor, master, …) and check to which degree these overlap between two jod advertisements



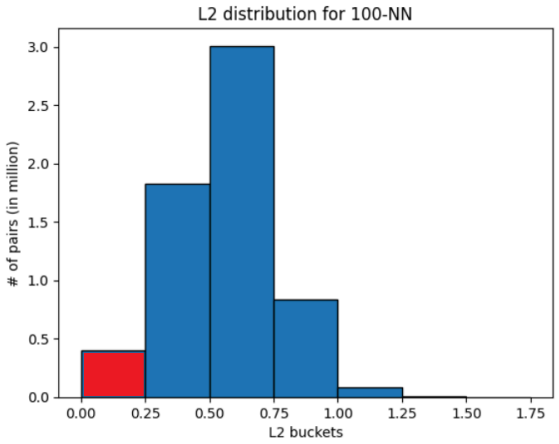
Parametrization of Thresholds		
Predictor	**Parameter**	**Relationship with outcomes**
L2 distance	`pre_filtering_l2_threshold`	The higher the L2-threshold the more potential duplicate candidates will be pre-selected
	`temporal_l2_threshold`	The higher the L2-threshold the more pairs will be put as temporal duplicates (conditional on other conditions being fulfilled)
Title similarity	`temporal_title_similarity_unrestricted`	The higher the parameter the less pairs that have the same company name will be declared as temporal duplicates

	`temporal_title_similarity_restricted`	The higher the parameter the less pairs that have a different company name will be declared as temporal duplicates

Full duplicates and full temporal duplicates: As described above, in step 1, we gather the full duplicates and full temporal duplicates using the function named *find_perfect_duplicates().* These duplicate pairs are very straightforward to understand: Full duplicates are identical in all columns of the initial raw data, while the "full temporal" duplicates, have a different retrieval date, and, hence, move into the temporal category.

Pre-Filtering of semantic, partials, and temporals based on L2-distance: To filter out other potential duplicate pairs, we only extract duplicate-pairs with an L2 distance between their merged title+description texts lower or equal to 0.25. In our code implementation, however, this L2-threshold for pre-filtering can be adjusted, with a higher L2-threshold indicating that less duplicate pairs would be "thrown out" in the pre-filtering phase, or, vice versa, with a lower L2-threshold one would gather a smaller, more restrictive pre-filtered dataset.

Our approach of 100-Nearest Neighbours produces for each unique text hash, roughly 61,000 texts,100 potential duplicate candidates, i.e. 6.1 million candidates. The L2 threshold of 0.25 filters out the 397,000 most semantically similar ones among those, i.e., the 6% most similar. When looking at the distribution of L2-distances, we, in fact, find that there is a dense distribution in the L2-buckets 0.25-0.50, suggesting that this would give us too many semantically dissimilar non-duplicates.



Temporals: Separating Temporals from Semantics and Partials, again is fairly straightforward, as we just extract those pairs from the pre-filtered duplicates with a different retrieval date. Hence, these pairs can be interpreted as semantically similar but with different retrieval date.

Moreover, our *expert rules for temporals* allow for a more fine-grained extraction and interpretability of the final temporal duplicates. While the L2-distance measures the combined similarity of a merged title+description text, our implementation allows to measure the title and description similarities separately. In our final solution, we omit the generation of description similarities, but in principle it could be included with the parameter and only set additional rules for the title similarity.

Current limitations and avenues for future development

Overall our model performs well given the constraints at hand. However, there are some limitations that are worth discussing. *TODO: List all problems and potential ramifications if we had more time to develop / stronger gpus etc*

Transformer models input length cap
Transformer models currently have a limit on the length of texts they can support. For our main model, MPNET, this limit is 384 "tokens," where a token roughly corresponds to a single word, although longer words may span two or even three tokens. In practice, this means that roughly one-third of our observations have texts that exceed 384 tokens/words, which results in the text being truncated and all information beyond the 384th token being lost.

Choice K nearest neighbors
Another limitation is that we currently only use the 100 nearest neighbors for each observation before evaluating the L2 distance. Ideally, instead of selecting the 100 nearest matches first and subsequently keeping only those that have an L2 distance below a certain threshold, we should have only considered the threshold itself. This would be low-hanging fruit in terms of improvement. However, it would only affect 881 job advertisements. For these job advertisements, all 100 nearest matches have an L2 distance below 0.25, and it is likely that the 101st match would have been selected as well. In other words, the red group of matched tuples would have been more numerous if we had not capped the potential matches to 100 before considering the L2 distance.

Transformer models require graphical processing units (GPUs)
Language embedding is a computationally costly operation which, like most machine learning models benefits greatly from being executed on a GPU rather than a CPU. This is makes running the code more expensive as GPU compute time is typically much more expensive compare to CPU.

Google Translate API is a bottleneck
The current bottleneck in our script is the Google Translate API, as translating the entire dataset is estimated to take approximately eight hours. This is because each translation requires a separate API call. However, by using asynchronous API calls, we could potentially improve the runtime significantly. However, it's worth noting that without a paid account, Google throttles concurrent requests to prevent excessive usage of their free service (thereby limiting our ability to invoke asynchronous API calls). Thus, a paid subscription to Google Translate (or DeepL, for that matter) may be a worthwhile endeavor for a production-grade solution.

Difficulties in identifying non-overlapping features between different languages
The task of identifying partial duplicates among job titles and descriptions in a multilingual dataset can be a challenging endeavor, fraught with difficulties that stem from differences in language and semantics. One of the “expert rules” we used involves translating all texts into a common language, such as English, and applying stemming to identify non-overlapping features. However, this approach is not without its challenges. One of the most notable difficulties is that certain features may be lost in translation, as they are expressed differently across languages. This can lead to a loss of granularity and nuance in the comparison process, which in turn can make it difficult to accurately identify partial duplicates. Indeed, even synonymous terms may have subtle differences in meaning that can escape notice when translated into another language. As such, identifying non-overlapping features

between different languages requires a careful balancing act between accuracy and flexibility, one that demands expertise and an appreciation for the complexity of language and meaning.

Similarities/differences to State-of-the-Art techniques (optional)

Please provide a list of similarities and differences between the used methodology and to the state-of-the-art techniques.

Transformer-Based Models

While in the early and mid 2010s bag-of-words methods and LDA (topic modeling) approaches have been widely dominant in economic research (Loughran & McDonald, 2011; Hansen & McMahon, 2016), transformer-based models are increasingly used in the industry, AI research (Acheampong et al., 2021) and, in recent years, also in economic research (Zhao et al. 2021).

For applications involving textual similarity, sentence transformers are regarded as the state-of-the-art method (Thakur et al, 2021). Different from Word-2-Vec (Rong 2014), sentence transformers allow the generation of vector representation for longer text passages and not just single words. While different sentence transformer models exist, we rely on an model comparison from Reimers (2023) to apply the most performant English model, all-mpnet-base-v2 (Song et al. 2020):

Sentence-Transformers: Model Comparison

Model	Avg. Performance	Speed	Model Size
all-mpnet-base-v2	63.30	2800	420 MB
multi-qa-mpnet-base-dot-v1	62.18	2800	420 MB
all-distilroberta-v1	59.84	4000	290 MB
all-MiniLM-L12-v2	59.76	7500	120 MB
multi-qa-distilbert-cos-v1	59.41	4000	250 MB
all-MiniLM-L6-v2	58.80	14200	80 MB

Comparison from Reimers (2023). Avg. Performance refers to the performance in 20 different tasks involving sentence encoding and semantic search. Speed refers to the sentence encoding speed per second on a V100 (higher being better).

As demonstrated by benchmark comparisons, mpnet showed the highest performance. However, with a model size of 420MB, it is one of the largest models evaluated. Since we prioritized high prediction accuracy for the Accuracy and Accuracy+ awards, we chose mpnet as our English model. However, if storage size or speed are more important metrics, one could consider a smaller model, such as *all-MiniLM-L6-v2*.

For our multilingual approach, we used distiluse-base-multilingual-cased-v2, which is the default multilingual sentence-transformer model that includes 50 languages, and *paraphrase-MiniLM-L12-v2*, which is also trained on 50 languages.

Dense Passage Retrieval and Faiss for scalable search

Our approach to representing text with complex vector representations and then searching/retrieving similar texts is known as Dense Passage Retrieval (DPR). In contrast to this, there is Sparse Passage Retrieval that only considers direct overlap of words, such as BM25, and therefore ignores similarities between synonyms.

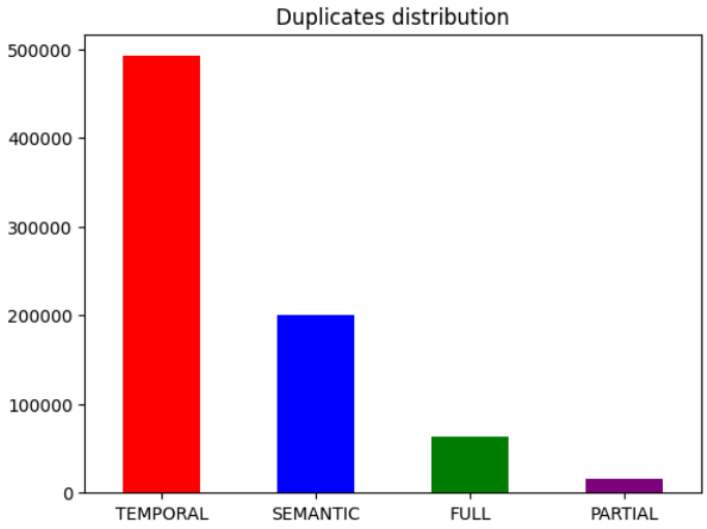
Using FAISS as a component for Dense Passage Retrieval is a common solution in state-of-the-art NLP industry solutions, such as Haystack (Möller, Risch, & Pietsch, 2021) and Pinecone (Krishnan & Liberty, 2021), as well as in academic research (Karpukhin et al., 2021).The choice between dense or sparse vectors for passage retrieval involves a trade-off between time and performance, with dense retrievers usually showing higher accuracy but also higher duration (*Ma et el. 2021*).

Combining Titles and Descriptions

In our dataset, each job advertisement has two text fields: title and description. While the natural solution is to merge the two fields, we found that the job title is more important when comparing potential duplicates. Hence, we constructed a new text field that is a weighted combination of the title and description fields, with a title weight of 30% for pre-filtering. This ensures that the weighting of the title remains relevant in longer texts. We tested different title-weightings for pre-filtering, and found a weight of 30% to be the most effective. Hence, we differ from SotA to adjust our methodology to the specificities of the use case: Having two text fields per observation with different relative importance.

Using expert rules

While many state-of-the-art industry applications combine transformer models with scalable document stores like FAISS for the final output, we only use these tools for a rough pre-selection of potential duplicates. We use more fine-grained rules for the final output, as detecting partials requires another set of rules beyond simple semantic similarity.

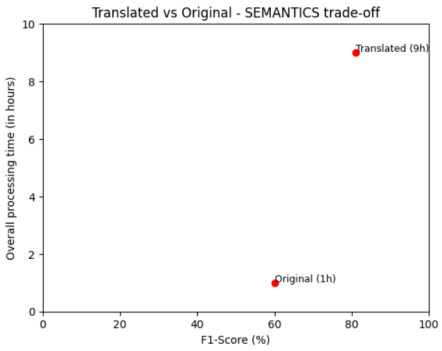


Lessons Learned (optional)

Please state any lessons learned during the competition.

i) Translated texts & mono-lingual model versus original texts & multi-lingual model:
When we first approached this challenge, one of the primary questions we had to ask ourselves was whether translating all texts into English and applying a mono-lingual model would be better compared to using the original texts and applying a multi-lingual model. Since we had a number of submissions, we decided to test both approaches separately.

Overall, the multi-lingual model (distiluse) worked reasonably well and generated similar word embeddings for texts in different languages. However, we found that our accuracy was significantly higher when we included the translation step in our process and applied an "English-only" language model (mpnet). We confirmed that, when we manually checked a sub-sample of our pre-filtered dataset using a multi-lingual model, and discovered that the multi-lingual approach did not perform well for less widely used languages. For example, we falsely matched the Lithuanian job titles PAMAINOS VADOVAS (Shift manager) and GAMYBOS DARBUOTOJAS - KOKYBĖS KONTROLIERIUS (Production worker - quality controller) as duplicates because the multi-lingual model could not detect the dissimilarity between the titles.



However, the translation step is the most time-consuming process in our final solution, taking around eight hours, while the remaining steps take less than one hour in total (using a GPU). Although the limited number of submissions did not allow for a clean comparison between the two approaches, we can use parts of our submissions to obtain a rough benchmark for the accuracy-time consumption trade-off between the two approaches. In submission 1, we used the multi-lingual model approach and achieved a 60% F1 score in the SEMANTICS with roughly 300k pairs submitted as semantics. Similarly, in submission 5, we used the translations & English model approach and submitted a similar number of pairs.

We ultimately opted for the translation approach due to its higher accuracy and the identification of potential problematic classifications. However, we want to emphasize that if minimizing run time is the top priority, the multi-lingual model approach may be more appropriate.

ii) Understanding the data matters
Understanding the data is crucial when it comes to building machine learning models. Although these models are often marketed as end-to-end solutions that eliminate the need for developers to spend time exploring and analyzing the data, we found that this is not entirely true. While we are satisfied with the performance of our model, we recognize that we could not have achieved such results without investing many hours in carefully scrutinizing the data and gaining a deep understanding of it. This allowed us to create expert rules that ultimately contributed to the success of our solution. Therefore, it is essential to emphasize the importance of data investigation and understanding in the process of developing machine learning models.

iii) Diversity matters
Being part of a team composed of individuals with diverse academic and demographic backgrounds was an extraordinary privilege. Each member brought their unique experiences to the table, enabling us to tackle challenges from multiple perspectives and generate innovative solutions that would have been impossible for any individual to conceive on their own. Our team consisted of two Greek and two German members, and we were fortunate enough to share a common European spirit, which added a valuable dimension to our collaboration.

Hardware Specifications

Please describe the hardware specifications of the machines that were used to run the methodology.

Machine 1: AWS sagermaker studio lab using EC2 instance g4dn.xlarge

CPUs	4
GPUs	1 NVIDIA T4 Tensor Core GPU
TPUs	-
Disk space	15GB

See:

- https://aws.amazon.com/de/blogs/aws/now-available-ec2-instances-g4-with-nvidia-t4-tensor-core-gpus/
- https://docs.aws.amazon.com/sagemaker/latest/dg/studio-lab-overview.html

Machine 2

CPUs	-
GPUs	AWS ml.g5.4xlarge
TPUs	-
Disk space	50GB

Short description of the Team – area of expertise (optional)

Please provide a description of the team, your area of expertise and contact information.

Dimitrios Petridis, MSc Economics, Data Analyst

Stefan Pasch, PhD Economics, Data Scientist

Jannic Cutura, PhD Economics, Data & Software Engineer

Charalampos Lagonidis, MSc student (AI and Data Analytics), Research Assistant (NLP)

REFERENCES

Acheampong, F. A., Nunoo-Mensah, H., & Chen, W. (2021). Transformer models for text-based emotion detection: a review of BERT-based approaches. *Artificial Intelligence Review*, 1-41.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

González-Carvajal, S., & Garrido-Merchán, E. C. (2020). Comparing BERT against traditional machine learning text classification. *arXiv preprint arXiv:2005.13012*.

Hansen, S., & McMahon, M. (2016). Shocking language: Understanding the macroeconomic effects of central bank communication. *Journal of International Economics*, *99*, S114-S133.

Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, *7*(3), 535-547.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., ... & Yih, W. T. (2020, November). Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 6769-6781).

Krishnan, A., & Liberty, E. (2021). Projective Clustering Product Quantization. *arXiv preprint arXiv:2112.02179*.

Loughran, T., & McDonald, B. (2011). When is a liability not a liability? Textual analysis, dictionaries, and 10-Ks. *The Journal of finance*, *66*(1), 35-65.

Ma, X., Li, M., Sun, K., Xin, J., & Lin, J. (2021). Simple and effective unsupervised redundancy elimination to compress dense vectors for passage retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (pp. 2854-2859).

Möller, T., Risch, J., & Pietsch, M. (2021). GermanQuAD and GermanDPR: Improving Non-English Question Answering and Passage Retrieval. In *Proceedings of the 3rd Workshop on Machine Reading for Question Answering* (pp. 42-50).

Ramadasa, I., Liyanage, L., Asanka, D., & Dilanka, T. (2022). Analysis of the effectiveness of using Google Translations API for NLP of Sinhalese.

Reimers, N. (2023). Sentence-transformers. Pretrained-Models. URL: https://www.sbert.net/docs/pretrained_models.html. Retrieved: 2023, April, 9th

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 3982-3992).

Reimers, N., & Gurevych, I. (2020). Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 4512-4525).

Rong, X. (2014). word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Song, K., Tan, X., Qin, T., Lu, J., & Liu, T. Y. (2020). Mpnet: Masked and permuted pre-training for language understanding. *Advances in Neural Information Processing Systems*, *33*, 16857-16867.

Thakur, N., Reimers, N., Daxenberger, J., & Gurevych, I. (2021, June). Augmented SBERT: Data Augmentation Method for Improving Bi-Encoders for Pairwise Sentence Scoring Tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 296-310).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Zhao, L., Li, L., Zheng, X., & Zhang, J. (2021). A BERT based sentiment analysis and key entity detection approach for online financial texts. In *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (pp. 1233-1238). IEEE.