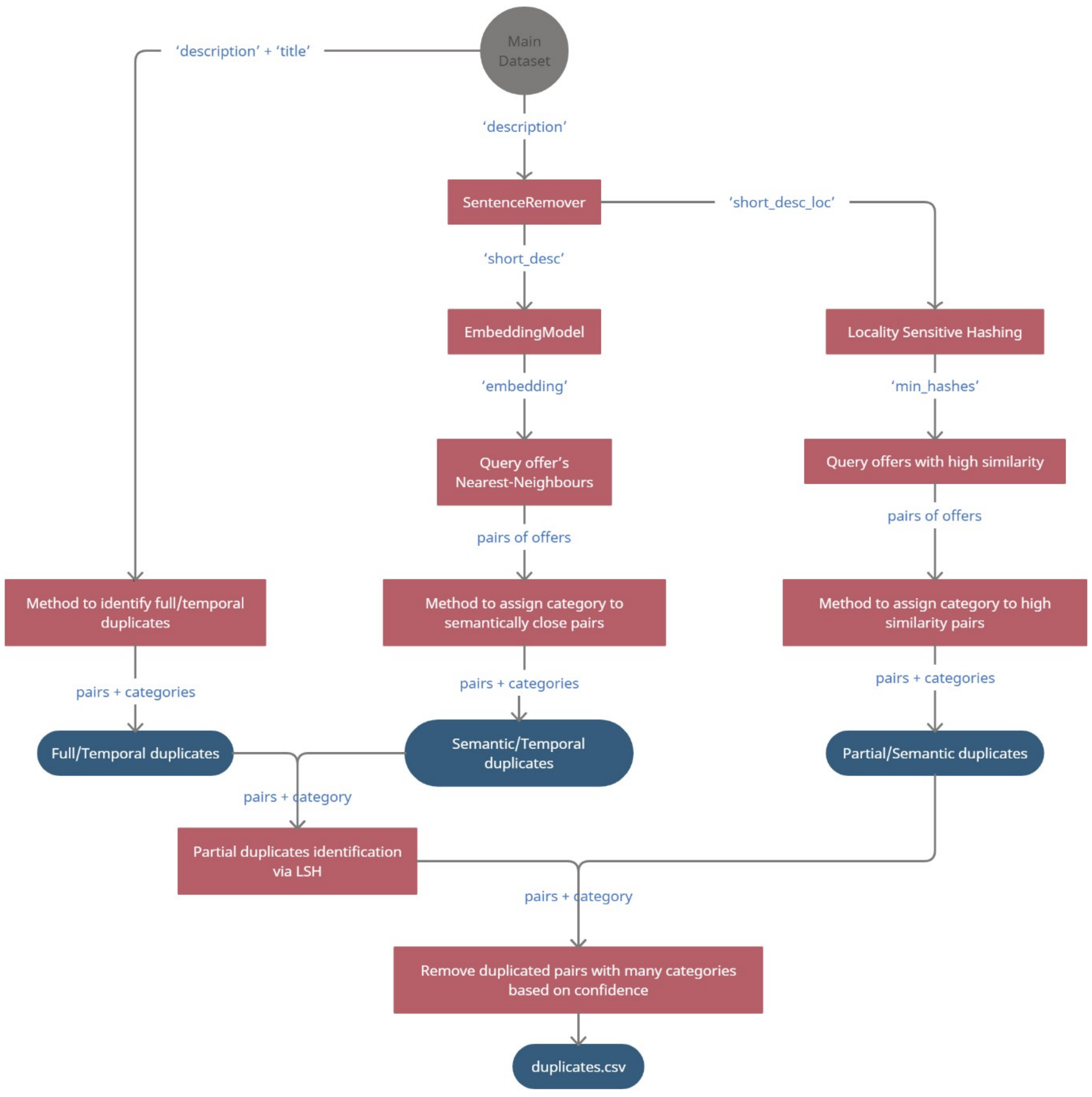
Reproducibility Award Methodology Description

PIPELINE


Main Dataset
'description' + 'title'
'description'
SentenceRemover
'short_desc_loc'
'short_desc'
EmbeddingModel
Locality Sensitive Hashing
'embedding'
'min_hashes'
Query offer's Nearest-Neighbours
Query offers with high similarity
pairs of offers
pairs of offers
Method to identify full/temporal duplicates
Method to assign category to semantically close pairs
Method to assign category to high similarity pairs
pairs + categories
pairs + categories
pairs + categories
Full/Temporal duplicates
Semantic/Temporal duplicates
Partial/Semantic duplicates
pairs + category
Partial duplicates identification via LSH
pairs + category
Remove duplicated pairs with many categories based on confidence
duplicates.csv

Data processing steps:
In the diagram above, the main/output object is the set passed to all teams. The blue descriptions on the arrows describe what information flows between the components. The red rectangles

are separate components that have strictly defined tasks in processing. The blue shapes describe the results we have obtained.
We have 3 main types of data processing. The data processing stage on the far left ('exact approach') is based on the content of the 'description' and 'title' columns. It uses them to identify full and temporal duplicates.
The middle stage of data processing('semantic approach') focuses mainly on obtaining as much information as possible from the 'description' column. Detecting duplicates is based on the semantic similarity between offers.
The data processing stage on the right-hand side ('word similarity approach') is generated based on the 'SentenceRemover' result and is based on word similarity using Locality Sensitive Hashing to find them.
After receiving the results from the first level of processing, some of them are passed to a module that uses Locality Sensitive Hashing to determine which offers are likely to be partial duplicates by comparing them to offers that may extend a given offer.
The final results are passed to the last step, which removes duplicates from identified pairs based on how confident we are about the obtained pairs.

The 'exact approach' is based on an optimized brute-force approach. Instead of comparing each offer with every other offer, we use the title to optimize the process and for a given title, we check all the offers with the same title. If their 'description' content is exactly the same character by character, the pair is marked as a 'FULL' duplicate. In case of a different date, it is marked as a 'TEMPORAL' duplicate.

SentenceRemover

The 'semantic approach' and 'word similarity approach' are based on the result of processing the 'SentenceRemover' component. Many offers, in addition to the job offer text, also included additional text that was displayed on the page and was not related to the job offer. Often it was information about registration, logging in or website regulations. It is a logical assumption that this type of information will appear many times within a given country (the same websites). The use of SentenceRemover has a deep meaning for interpretability - after all, it does not remove random sentences, but sentences that do not actually relate to the offer. This text is removed by the SentenceRemover. 'SentenceRemover' creates a set of 5-grams for each country ('country_id') based on the offers using TfidfVectorizer. Before determining the 5-grams, the text is preprocessed using 'remove_duplicate_sentences' and 'custom_tokenizer' functions.

Function 'remove_duplicate_sentences'(text), takes in the raw input text and applies preprocessing steps to it before tokenizing (in TfidfVectorizer). It removes newline characters, many of the same character (at least 3 in a row), and replaces multiple spaces with a single space. It then calls the find_dup_longest_common_substring function to find the longest duplicated substring and replaces only the first occurrence of the substring with a space if it is longer than 15 words. To avoid excessive influence of repeated text fragments on the analysis results, it was decided to remove duplicated sentences from offer descriptions. This can increase the effectiveness of the model and improve its ability to detect unique characteristics of offer descriptions.

Function 'find_dup_longest_common_substring(text)', finds the longest common substring (sequence of words) inside text. It uses the dynamic programming approach and tokenizes the text by splitting it into words separated by spaces. It initializes a matrix to store the lengths of the longest common suffixes between pairs of words in the text. The function then iterates over all possible pairs of suffixes, checks if they match, updates the matrix entry, and checks if it's the longest match so far. If the suffixes don't match, it sets the matrix entry to 0. Finally, the function uses the end position and length of the longest match to return the matching substring.

Function, 'custom_tokenizer(text)', is used as the tokenizer in

TfidfVectorizer to convert the input text into a sequence of tokens (words or subwords) that will be used as features in the resulting document-term matrix. The function removes digits and punctuation from the text, removes sequences of the same character (at least 3 in a row), replaces multiple spaces with a single space, and finds all word tokens in the text using a regular expression.

After initializing the TfidfVectorizer and passing the data, the next step is to build a dictionary of 5-grams that should be removed from the text. For each language, a different level is determined based on the threshold value of idf. Only a subset of the most frequently repeated phrases is selected for removal and saved in a dictionary.

Function, 'buildNgramDictionary(vectorizer, country)', creates a dictionary that stores the most popular 5-grams to be removed. It takes in a vectorizer object with IDF weights already computed and a string representing the country for which the dictionary is being built. The function gets the feature names and IDF values from the vectorizer, removes duplicate IDF values, and sorts the array in descending order. It then computes the three largest values in the array and sets the initial quantile value to 0.02. If the job offers are from Luxembourg, it sets the threshold manually. Otherwise, it computes the quantile value and checks if the quantile value is different from the three largest values. If it is not, the function sets the quantile value to the third highest value. Finally, the function creates a dictionary with only the terms that have an IDF value smaller than the quantile.

After preparing the 5-grams to be removed, we apply the function 'removeTrashSentencesIDFv2' to the 'description' column to create a new 'short_desc' column, which contains short fragments of the offer after preprocessing.

This function ('removeTrashSentencesIDFv2') takes in a string document and a dictionary vectorizer_dict as arguments. The function returns a cleaned version of the input string with specific 5-grams removed. The vectorizer_dict contains the TfidfVectorizer object, which is used to preprocess the input string, and a dictionary with the 5-grams to be removed. The function applies the preprocessor function to the input string and removes the identified 5-grams using a generator expression and a regular expression. It then replaces all consecutive whitespaces with a single whitespace. If the cleaned_text is empty, the function returns the original input string. The function also saves the cleaned_text in a cache dictionary for reuse.

The preprocessing described above is actually simple in data cleaning. We prepare data in a logical way, not just apply a model.

EmbeddingModel
Based on the shortened job description 'short_desc', we use the model 'sentence-transformers/distiluse-base-multilingual-cased-v2' (https://huggingface.co/sentence-transformers/distiluse-base-multilingual-cased-v2) licensed under the Apache 2.0 license to generate an embedding for each job offer with a maximum input token length of 512. We based on publication by Reimers N. and Gurevych I. (https://arxiv.org/abs/1908.10084). Publication present Sentence-BERT (SBERT), a modification of the pretrained BERT network that use siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity.

Query offer's Nearest-Neighbours
The hnswlib (https://github.com/nmslib/hnswlib) library is used to build a brute-force index with cosine similarity as the distance metric and 512 as the embedding dimensionality. For each offer, the k nearest neighbors are found using brute-force search (k-NN) with the knn_query function of the hnswlib library. The embedding and short_desc of the current offer are retrieved, and a nested loop iterates over each of the 43 nearest neighbors. For each pair cosine similarity is computed and is stored with offers id in separate dataframe.

Method to assign category to semantically close pairs
All the methods 'get_class...' use the same sentence transformer model to compare the 'title' text, but with only 50 input tokens.

For the subset of offers with a cosine similarity score equal to 1, we select the offers that have been labeled as 'SEMANTIC' duplicates using the 'get_class_cos_1' method.

Function 'get_class_cos_1'. This function takes in a row containing information about two offers and compares their titles to classify them as either "FULL", "SEMANTIC", "TEMPORAL", or "NON-DUPLICATE" duplicates.

It first extracts the titles of the two offers from the row and checks if they have the same values for three features: "location", "country_id", and "company_name". If they do, the function calculates the cosine similarity between the embeddings of the two titles using a pre-trained sentence transformer model with a cache to speed up the computation.

If the offers have the same retrieval_date, the function first checks if the descriptions and titles are the same. If they are, the function classifies them as "FULL" duplicates. If not, it checks if the cosine similarity between the embeddings of the two titles is greater than or equal to a pre-defined threshold for semantic similarity (0.58). If it is, the function classifies them as "SEMANTIC" duplicates. If not, it classifies them as "NON-DUPLICATE" duplicates.

If the offers have different retrieval_dates, the function checks if the cosine similarity between the embeddings of the two titles is greater than or equal to a pre-defined threshold for temporal similarity(0.65). If it is, the function classifies them as "TEMPORAL" duplicates. If not, it classifies them as "NON-DUPLICATE" duplicates.

If the offers have different values for any of the three features ("location", "country_id", and "company_name"), the function classifies them as "NON-DUPLICATE" duplicates.

The remaining set of previously identified pairs based on embeddings is divided. Offers with cosine similarity above 0.35 and below 1, and the same date are considered as possible 'SEMANTIC' or 'PARTIAL'. Offers with cosine similarity from 0.48 to 1 with different dates are checked if they can be 'TEMPORAL'. Checking the classes is done using the 'get_class' function. From this set pairs with category ['SEMANTIC', 'TEMPORAL', 'PARTIAL'] are only selected.

Function 'get_class'. This function takes a row from the DataFrame of potential duplicate pairs as input and returns the appropriate class for that pair. The function first extracts the left and right offers from the DataFrame and their titles. It then counts the number of features that are the same or both are NaN. If all three features (location, country_id, and company_name) are the same or both are NaN, the function proceeds to compute the embeddings for the left and right offer titles using the model_title.

If the retrieval dates are the same, the function checks if the descriptions and titles are the same, and if they are, it returns 'FULL'. Otherwise, if the cosine similarity between the embeddings of the titles is above the semantic threshold(0.56), the function checks if the pair contains an id that was identified as a potential partial duplicate. If it does, the function assigns the 'PARTIAL' category, otherwise, it assigns the 'SEMANTIC' category. If the cosine similarity between the embeddings of the titles is below the semantic threshold, the function assigns the 'NON-DUPLICATE' category.

If the retrieval dates are different, the function checks if the cosine similarity between the embeddings of the titles is above the temporal threshold(0.65). If it is, the function assigns the 'TEMPORAL' category, otherwise, it assigns the 'NON-DUPLICATE' category. If any of the features are different, the function assigns the 'NON-DUPLICATE' category.

Partial duplicates identification via LSH
Offers from both 'exact approach' and 'semantic approach', which have been marked as 'SEMANITC' or 'FULL', are checked in the context of being 'PARTIAL'. If a pair contains an index that

has been identified as a probable 'PARTIAL' duplicate, their category is changed to 'PARTIAL'.
The process of identifying 'PARTIAL' duplicates involves reviewing each offer from the main set and comparing whether there is an offer that contains all the information that the checked offer has and additional information that it lacks. To optimize the search for offers, we use Locality Sensitive Hashing based on words separated by spaces. It searches for offers that contain a lot of similar text. For the closest offers returned by Locality Sensitive Hashing Forest implementation - http://ekzhu.com/datasketch/lshforest.html – based on paper - http://ilpubs.stanford.edu:8090/678/1/2005-14.pdf. Jaccard is calculated on words in the 'description' column content to discard less important offers that have a Jaccard similarity less than 0.2. For the remaining offers, it is checked how many additional words the offer that was deemed the most similar contains. If the offer that we were checking for 'PARTIAL' has all the words that appear in the closest offer (most similar words) and additionally, the closest offer has at least 2 more words up to 20, the checked offer is considered 'PARTIAL'.

Locality Sensitive Hashing ('word similarity approach')
The 'word similarity approach' is based on the 'short_desc_loc' column, which is a combination of the text from the 'location' column with a previously generated short job description called "short_desc". The location was added to increase the likelihood of similarity, as location text is rarely translated. To find job offers with a lot of similar text, we used an implementation of LSHForest and a query method, which returns offers with the closest similarity.

A separate LSHForest was built for each language ('country_id'). For the closest offers, Jaccard similarity was calculated on words to compare how close they are based on the text in 'short_desc_loc'. Offers that were in the range <0.6,0.8) and had the same date were marked as 'SEMANTIC'.

Offers between (0.8,1) with the same date are labeled as 'PARTIAL'. Additionally, for each pair identified as 'PARTIAL' or 'SEMANTIC', the 'get_class_lsh' method assigns its category.

The function first retrieves the two offers with IDs IDLeft and IDRight from the DataFrame, and extracts their titles. It then checks if the offers have the same values for the location, country ID, and company name columns or if both have null values in these columns. If they do, it calculates the cosine similarity between the title embeddings of the two offers using a pre-trained encoder model. If the retrieval dates of the two offers are the same, it checks if the descriptions of the two offers are the same and if their titles are the same, in which case it classifies the offers as full duplicates. If the descriptions or titles are different, it compares the cosine similarity of the title embeddings to the cos_title_threshold_sem (0.58) value, and if it is above this threshold, it classifies the offers as semantic duplicates. If the retrieval dates of the two offers are different, it compares the cosine similarity of the title embeddings to the cos_title_threshold_temp (0.65) value, and if it is above this threshold, it classifies the offers as temporal duplicates. If the cosine similarity is below either threshold or the offers have different values for the location, country ID, or company name columns, it classifies the offers as non-duplicates.

It works as a filter, where pairs that receive the 'SEMANTIC' label are kept with their original category assigned earlier.

Remove duplicated pairs with many categories based on confidence

For each previously generated dataset, we are able to determine how well this method worked through previous analyses and their pairs are more important to us. The order of importance is: 'exact approach', 'word similarity approach', 'semantic approach', 'semantic approach' (pairs with cosine equal to 1). For the combined dataset, duplicates are removed while retaining the first pair that appeared.

It took 17 hours and 20 minutes to generate duplicates.csv file. All computations were done on a single laptop CPU (not all cores were used during computation). Thanks to the use of the cloud and greater computing power, the methods described here have great potential for scalability.

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.

Similarities:

1. **Using text processing techniques, such as tokenization, vectorization, and similarity measures (very popular similarity measure - cosinus), to compare text data.**
2. **Using improved BERT - SBERT model based on transformers architecture. The essence of the state-of-the-art techniques nowadays.**

Differences:

1. **The used methodology involves a combination of different approaches for identifying duplicates. Cleaning the data, removing meaningless sentences, filter data by get_class methods, using embeddings for cosine similarity and so on. These approaches are filling each other. Analogy to ensemble learning – many weak classifier can create one strong classifier.**
2. **The approach for 'partial' identification. For each job description(child offer) we search for parent offer (job description that contains base information) to see whether child offer contains subset of the parent information without additional text.**
3. **Dealing with long sequences of text by reliable data cleaning. Not just relying on a model.**

Lessons Learned (optional)

Please state any lessons learned during the competition.

1. **The values in the 'description' column do not contain information only about job offers, additional**

information such as text from advertised offers or menu text is often included. There are also cases where parsing errors occurred and instead of the offer description, an error description was saved.

2. **Since we are dealing with the problem of deduplication, it is advisable to save the results of calculations to a cache wherever possible, as it is highly likely that we may encounter the same input again in the future and speed up the calculations.**

3. **'Partial' duplicates can be identified by searching the parent offer. It is common that one job offer (parent) contains all the information, while another offer (child) with missing words from the parent offer's text is placed on another website.**

   The child offer contains subset of information from parent without any additional text. This assumption can be changed to allow offers with subset of information from parent to have some additional text to enhance the 'partial' duplication detection.

4. **Since identifying duplicates requires checking each pair with each other, it is worth using approximate nearest neighbour search methods from the hnswlib library, which are not as accurate as brute force but provide over 90% recall compared to brute force approach. They reduce neighbour search time by around 50%.**

5. **How to deploy AI in real problem. How to incrementally increasing accuracy for pipeline. First experience on working with big multiclass classification problem.**

Hardware Specifications

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

Machine 1

CPUs	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz (only part of the cores were used in processing)
GPUs	Intel(R) Iris(R) Xe Graphics (Not used)
TPUs	(Not used)
Disk space	16 GB RAM (available on the laptop, not fully used) Model used for embeddings creation – around 1.1 GB, other things may sum up to few GBs.