**Reproducibility Award Methodology Description – Team *Nins***

# **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

(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.

|  |
| --- |
| **Our code was written using the open source Python framework for data science Kedro (**[**https://kedro.org/**](https://kedro.org/)**). To observe the articulation of the different steps, you can:**  **- Clone the repository (antoine-palazz/deduplication) or open the submitted code**  **- Run «./setup.sh» to install the dependencies**  **- Run «kedro viz» to visualize the whole pipeline**  **- Filter to the «final\_models» tag to keep only what was used for the final submission**  **(1) Data processing steps**  *See deduplication/src/deduplication/pipelines/data\_processing for the code*  *See deduplication/conf/base/parameters/data\_processing.yaml for the parameters*  **Generation of new information:**  **- Use Named Entity Recognition (NER) to find all the locations and organizations present in the offers (all fields considered). The model selected from the ’transformers’ package is «Davlan/distilbert-base-multilingual-cased\_ner-hrl». Others were tested out, such as «Babelscape/wikineural-multilingual-ner», but not kept.**  **- Use the ’detect’ method from the ’ftlangdetect’ package to find the language of each offer**  **2 levels of data preprocessing:**  **- Basic preprocessing: Replace NaNs by empty strings + Remove HTML tags and residuals from the scraping + Lower texts**  **- Extended preprocessing: Basic preprocessing + Remove non-alphabetic characters + Remove stopwords + Lemmatize texts + Remove most frequent words**  **Computation of new columns:**  **- Beginning and end of (extensively preprocessed or not) descriptions**  **- Concatenation of all of the (extensively preprocessed or not) textual columns**  **Filtering of the offers depending on the situation:**  **- very\_preprocessed\_described\_offers: Extensively preprocessed rows that have a non-empty description**  **- very\_preprocessed\_detailed\_offers: Extensively preprocessed rows that have non-empty title, company name and location**  **- well\_preprocessed\_and\_described\_offers: Extensively preprocessed rows that have a non-generic description, ie a description that is unique to the title**  **- well\_preprocessed\_and\_described\_international\_offers: Well preprocessed and described international offers for which the company published offers in several countries or languages**  **(2) Methods and models used**  **(2.a) Find the ’gross’ pairs of duplicates**  **The idea of our approach is to stack several restrictive methods to find as many ’gross’ or ’potential’ duplicates as possible while limiting the amount of false positives. As of today, it is still possible to add as many methods as desired (while keeping them all quite restrictive) in order to improve the recall of our approach. To do that, one can use the «kedro pipeline create» command to create the files for a new method and copy the methodology used for the other approaches.**  **During the time of this challenge, 5 approaches were tested out and 3 were selected to generate the final submission.**  **(2.a.i) The ’easy’ duplicates**  *See deduplication/src/deduplication/pipelines/easy\_duplicates for the code*  *See deduplication/conf/base/parameters/easy\_duplicates.yaml for the parameters*  **Testing the equality of specific tuples of fields, without using any model, can easily lead to the discovery of duplicated pairs. So our first method was to go through all the pairs of offers in our dataset and test if certain columns were the same:**  **- «All the columns» to find the full duplicates**  **- «Title + extract of filtered description» to find some less specific duplicates**  **- «Title + company name + location » to find some less specific duplicates**  **- «Company name + location + retrieval\_date» with different languages to find some semantic/partial duplicates**  **(2.a.ii) The ’subtle’ duplicates**  *See deduplication/src/deduplication/extras/utils/py for the cosine similarity code*  **Several embedding methods were used to tokenize the offers (concatenating all textual columns). These tokens were then compared to all of the others through a cosine similarity matrix. Depending on the model, the time difference between the offers and their languages, a threshold for the cosine similarity of the pair is given to determine if the pair is a potential duplicate or not. The cosine similarity being too big to store, it was computed and treated by chunks of size 5000.**  **4 embedding approaches were tested out:**  **(2.a.ii.I) TF-IDF**  *See deduplication/src/deduplication/pipelines/subtle\_duplicates\_tf\_idf for the code*  **TF-IDF method on the extensively preprocessed offers, using ’tfidfTokenizer’ from ’sklearn.feature\_extraction’.**  **Mainly used to capture monolingual duplicates. Kept for the final submission.**  **(2.a.ii.II) Multilingual BERT**  *See deduplication/src/deduplication/pipelines/subtle\_duplicates\_multilingual\_bert for the code*  **Multilingual (pre-trained) BERT method on the basically preprocessed international offers, using ’bert-base-multilingual-uncased’ from ’BertModel’ and ’BertTokenizer’, within the ’transformers’ package.**  **Mainly used to capture multilingual duplicates. Not kept for the final submission (but still available ans stackable to the rest), as the embeddings were all too similar given the similar context, and the network was too complex to refine through extra training. The method below replaced it.**  **(2.a.ii.III) Distiluse Multilingual**  *See deduplication/src/deduplication/pipelines/subtle\_duplicates\_distiluse\_multilingual for the code*  **Distiled version of multilingual (pre-trained) BERT method on the basically preprocessed international offers, using ’distiluse-base-multilingual-cased-v2’ from ’SentenceTransformer’.**  **Mainly used to capture multilingual duplicates. Kept for the final submission, as it has proven to be more restrictive than multilingual BERT while identifying quite well multilingual true positives.**  **(2.a.ii.IV) XLM Roberta**  *See deduplication/src/deduplication/pipelines/subtle\_duplicates\_xlm\_roberta for the code*  **Multilingual ROBERTA method on the basically preprocessed international offers, using ’xlm-roberta-base from ’AutoModelForMaskedLM’ and ’AutoTokenizer’, within the ’transformers’ package. Last layer re-trained using the corpus from the offers.**  **Mainly used to capture multilingual duplicates. Not kept for the final submission (but still available), as the re-training of the last layer of the network is very long to compute and the fine-tuning was not possible given the time of the challenge, but it could be looked into for the future.**  **(2.b) From «gross duplicates» to «true duplicates»**  *See deduplication/src/deduplication/extras/utils/py for the code*  *See deduplication/conf/base/parameters.yaml for the parameters and thresholds*  **While the previous methods are able to identify most of the duplicates (leading to a high recall), they may also identify a lot of false positives. Moreover, the previous approaches do not differentiate the duplicates between the different types.**  **That is why all the potential (or gross) duplicated pairs identified previously go through an ultimate test to send them to the correct ’box’. This function is called ’differentiate\_duplicates’ and can be decomposed in following tests:**  **- Are all the columns of the pairs equal? If yes, FULL duplicate**  **- Are all the columns but the retrieval\_date equal? If yes, TEMPORAL duplicate**  **- Are the countries different or the dates too far away? Are the descriptions lengths too different (in both absolute and relative difference) ? Is there a field (title, company, location, … ?) for which the two offers have a Jaro-Winkler difference below a certain threshold? If yes, NOT a duplicate.**  **To determine if the pair is a PARTIAL duplicate, two methods were tested out:**  **- The rejected one, based on NER: If the offers have different dates, this is a TEMPORAL. Otherwise, if one offer has a missing field that the other one has filled, then is the number of identified named entities (through NER in all the fields) the same? If no and if the offers are in the same language, then this is a PARTIAL duplicate. Otherwise it is a SEMANTIC. In practice, this method returns too many PARTIALS.**  **- The selected one, based on description lengths: If one offer has a missing field that the other one has filled, then we apply a series of tests based on descriptions lengths and number of missing fields to determine what kind of duplicate (or not duplicate) the pair is.**  **After all these tests, the pair is necessarily a duplicate. If the dates are the same, SEMANTIC duplicates, otherwise TEMPORAL duplicates.**  **(2.c) Aggregation and transitivity properties**  *See deduplication/src/deduplication/pipelines/union\_all\_duplicates for the code*  **Once all the ’true’ duplicates are computed, they get aggregated into one dataframe (each unique pair being kept only once). From this table of duplicates, we can add some more using transitivity properties:**  **(2.c.i) Add semantic duplicates (implemented)**  **Being a semantic duplicate is a transitive property. If each offer is a node in a non oriented graph, with edges representing the "semantic duplicate" property, then this function transforms all of the connected components into cliques, ie applying this transitive property in order to find more semantic duplicates not caught earlier.**  **(2.c.ii) Add partial duplicates (not kept)**  **If an offer A is a partial duplicate to an offer B, then B is a partial duplicate to all of the semantic duplicates of A and A is a partial duplicate to all of the semantic duplicates of B. Going even further, all of the semantic duplicates of A are partial duplicates to all of the semantic duplicates of B. Using this property, we can catch many new partial duplicates that were not found before. However, this method requires to have a high level of confidence into the duplicates (whether they are partial or semantic) we already have, which we never had during the time of the challenge. If this level of confidence is reached, it would be interesting to uncomment this part in the code.**  **(2.d) Adjust final output based on past approaches**  **Use past submissions or approaches that we know are inefficient (from logic or manual labellings) to filter out pairs that are most likely not duplicates. This will not increase the recall but may improve the precision from a few percent, even if not significantly. This part is not concerned by the Reproducibility part.**  **(3) References and inspirations**  **(1) The data processing is very standard. The original part comes from the filtering of the rows (for instance removing too generic descriptions) not to add false duplicates.**  **NER: https://huggingface.co/Davlan/distilbert-base-multilingual-cased-ner-hrl**  **Language detection: https://pypi.org/project/fasttext-langdetect/**  **(2.a.i) Original method based on intuition**  **(2.a.ii) Methods inspired from thorough discussions with ChatGPT, aligned with state-of-the-art practices**  **(2.a.ii.I)** [**https://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.TfidfVectorizer.html**](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)  **(2.a.ii.II)** [**https://huggingface.co/bert-base-multilingual-uncased**](https://huggingface.co/bert-base-multilingual-uncased)  **(2.a.ii.III) https://huggingface.co/sentence-transformers/distiluse-base-multilingual-cased-v2**  **(2.a.ii.IV) https://huggingface.co/docs/transformers/model\_doc/xlm-roberta**  **(2.b) Original approach coming from trial-and-error**  **(2.c) Original method based on intuition**  **(2.d) Original approach coming from trial-and-error**  **(4) Times of execution for the final submission**  ***Please note that with Kedro, most of the code can be run faster with a ParallelRunner, using «Kedro run --runner=ParallelRunner», and has been made so that a maximum of steps can be run at the same time, so the total duration of execution is way lower than the sum of the times of execution.***  **(1) Data processing**  **- Data preprocessing: ~15min**  **- NER: ~45min** *(not kept in final version)*  **(2) Methods and models used**  **(a)(i) + (b) ~45min** *(using the ParallelRunner)*  **(a)(ii.{I, II, IV}) + (b) ~1h** *(using the ParallelRunner)*+2h if re-training last layer of ROBERTA and using it for encoding, but not kept in the final model  **(a)(ii.III) + (b) ~1h**  **(c) ~45min**, the longest part being the ’transitivity’ part  **(d) ~1min**  **Total time of execution: 3 hours**, when optimizing the use of the resources using «Kedro run» with:  **--tags=final\_models\_parallel --runner=ParallelRunner**  **--tags=final\_models\_sequential --runner=SequentialRunner**  Without optimizing the use of the resources, ie using:  kedro run --tags=final\_models,  the time of execution climbs **up to 7 hours**.  Also add **10min** for the very first execution to download packages and models.  See <https://github.com/antoine-palazz/deduplication>for the code. |

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

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

|  |
| --- |
| The statistical methods themselves used for this challenge are not revolutionary, nothing to challenge state-of-the-art techniques. The embedding algorithms (*multilingual BERT, XLM Roberta*), the language detection (*with Fasttext*) or the NER (*with Distilbert*) mentioned above are aligned with standard up-to-date practices though.  However, the plus of our approach that can be used for OJA deduplication is **the use of the right tests, comparisons and filters on the right objects**. This includes:  - **Not using description-based tests for generic descriptions**, ie descriptions that are re-used by a company for all of its offers, independently of the title  - Having approaches that work well for monolingual duplicates (equalities of fields and TF-IDF) and **keeping the more complex approaches only for companies that publish offers in multiple languages** (thus focusing on multilingual duplicates)  - Playing with alphabetical sortings in order to **reduce the complexity of the searches from a O(n²) to a O(n\*log(n)) when doing it by hand** instead of using Pandas-like groupby approaches. Overall, a lot of precautions were taken to reduce the complexity of the methods to a minimum, whether it is **by parallelizing requests or by simplifying the searches**.  - **Looking into pairs of offers published by the same company in the same location on the same day, but in different languages**, most likely multilingual semantic duplicates  - Trying to **match only extracts (beginning or end) of descriptions**, thus reducing the number of false negatives due to punctual typing mistakes  - **Lemmatizing offers and removing stopwords only for approaches not based on the semantic** of the sentences, such as TF IDF  - **Adding a last layer of tests on all potential pairs** to exclude as many false positives as possible. These tests can go from **comparing the descriptions lengths to the difference in time between the offers, by way of comparing the syntactic difference of the fields** (using Jaro-Winkler difference). Each difference must be below a certain threshold, that will depend on if the offers are in the same language, their date difference, etc. We optimized these thresholds using the time and resources we had for this challenge, but they are probably still far from their optimal value. **Using a grid search on the training set could have a huge impact** on the amount of false positives that can be prevented.  Overall, our approach consists in many simple stacked tests, which can lead to a high number of intermediary tables. However, the results we obtained show that **high performances can be obtained through simply implemented and fast to execute tests that can be combined with each other**. This means this approach is **easy to maintain, easy to apprehend and to understand for a new comer on the project, easily stackable with other methods** as new ideas will arrive. Moreover, **once optimal thresholds are determined, we believe our approach will lead to excellent performances**.  The innovative part in our work is our use of **Kedro** (<https://kedro.org/>), an open sourced Python framework hosted by the Linux Foundation. **Kedro standardizes how data science code is created to ensure it is reproducible, maintainable, and modular; it uses software engineering best practices to help build production-ready data science code**.  Kedro is perfectly adapted for our approach, as the use of nodes and pipelines fits very well to our concept of stacking and combining methods on differently processed dataframes. It allows for a **very clearly structured code, visualization tools, and has shown many benefits for both exploration and production phases**. We believe our code **can be very easily re-used for OJA production purposes** thanks to this framework, that is **aligned with the most state-of-the-art techniques** in the field. **Detailed information about Kedro and its benefits is available on the framework website.**  To conclude, we at Insee Innovation Team place a paramount value on scientific reproducibility, particularly in the realm of data science, and as such, we have made **all our codes easily accessible to the public through Github** (<https://github.com/antoine-palazz/deduplication>). **The comprehensive collection of codes has been available to the public since the commencement of the competition**, and we take great pride in our unwavering commitment to transparency and openness.  For more information about the Datalab used for our computations, please visit the **Onyxia** project, developed by **INSEE**: <https://github.com/InseeFrLab/onyxia>. |

## **Lessons Learned (optional)**

Please state any lessons learned during the competition.

|  |
| --- |
| Several difficulties we met during the challenge came from our **understanding of the definitions**, particularly about the different types of duplicates. We learnt that spending more time early in the challenge defining precisely the borders between each category, for instance between semantic and partial duplicates, could have offered us more time to focus on more subtle tasks, instead of wasting submissions leveraging poor performances. We believe that **any work on the deduplication of job offers requires an important amount of time spent on clarifying the definitions and the subtleties between each type of duplicate**. We insist on that as **the definitions themselves are often the key to attaining very high performances, as simple tests and checks of the properties of the pairs can identify quite well if there is a duplicate or not**.  We also learnt that **gross conditions and thresholds can be very useful, even if they lead to the loss of some true duplicates**. For instance, not allowing duplicated pairs with a time difference above 100 days or not allowing partial duplicates with a different date was not specified within the rules and made us lose some true positives. However, the amount of false positives we did not return thanks to this additional conditions was so big in comparison that we chose to keep the tests for our final models. In practice, our recall lowered a bit, but thanks to the important increase in precision it generated, the F1-score increased as well. We believe such conditions can be re-used for OJA purposes.  Finally, we learned that **the most complex models were not always the best performing ones** for a task such as deduplicating job offers. For instance, the distiled version of multilingual BERT performed way better than its more complete version, that tended to embed most offers in very similar directions. **Our best performances were attained through simpler models and tests** that did not try to find relationships where there were not any. |

# **Hardware Specifications**

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

**Machine 1:** [***https://datalab.sspcloud.fr/***](https://datalab.sspcloud.fr/)

|  |  |
| --- | --- |
| CPUs | **25** CPUs |
| GPUs | **1** GPU |
| TPUs | **0** TPU |
| Disk space | **50** Go |

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

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

|  |
| --- |
| **Antoine Palazzolo** – Data Scientist at INSEE, within the innovation team in data science.  Mail: [antoine.palazzolo@insee.fr](mailto:antoine.palazzolo@insee.fr) / Tel: +33 7 60 94 99 72  LinkedIn: <https://www.linkedin.com/in/antoine-palazzolo/> |

**Appendix**

Example of a possible pipeline visualization with Kedro

