1. Introduction and objective

For this project I explored different machine learning classification models to predict four salary categories for Data Science job posts using publications from Indeed.co.uk.

The goal was to obtain an accuracy of 0.8 or higher on both the train and test group, which implies predicting correctly at least 80% of the total population.

2. First stage: scraping and cleaning

In every machine learning project, the first part of it consists of an extensive cleaning work to obtain a solid and complete dataset to make predictions.

As not every publication in Indeed.co.uk has the salary information, it was necessary to scrap thousands of pages in order to have at least 1k publications after dropping those without salary. This first scraping was done considering City using a list of all the cities in the UK. I scraped 1000 pages for the biggest cities and only 50 for the smallest ones, since those have very few job publications and scraping more would have ended up in a lot of duplicate information.

Once I collected all the data from Indeed, two different datasets were created from it:

- **Base**: it contains the core information scraped and cleaned from all the job posts and some other fields created from it
- **Summary Vectorized**: it contains each word that appeared in any publication, transformed into a column, where the values are weighted according to the number of times each word appeared in total (mainly used as a back up)

In order to put together the Base DataFrame, I created a few new variables to represent interesting features of the job title or summary:

- **Job_Kind**: Indeed has default filter options used to scrap the webpage by layers, but also to store as a feature of my dataset. Job_Kind makes reference to the type of contract, which can be ‘Full-Time’, ‘Permanent’, ‘Contract’, ‘Temporary’, ‘Internship’, ‘Part-Time’ or ‘Apprenticeship’
- **Seniority**: created from the Job Title and Summary, it was designed from two different lists, with words belonging to Senior o Junior levels of jobs. If any word of each level was present, either on the Job Title, on in the Summary, the corresponding seniority level was assigned. If none of the words was in those features, the job post was assigned as a Middle level job.

Also from the scraping, I stored Job_Type already clean from the filters used.

Besides this, I also incorporated into my dataset the following information:

- **Company**: scraped and cleaned from each job post
- **City**: as "Job_Kind", this feature was also stored from Indeed filters
- **Industry**: scraped from the About page of each Company on Indeed, using a filtered and clean list of unique names of companies scraped

Finally, I also created a third dataset, that is just the combination of the base_df and the vectorized information from the summary feature of each publication.

3. Analyzing the data

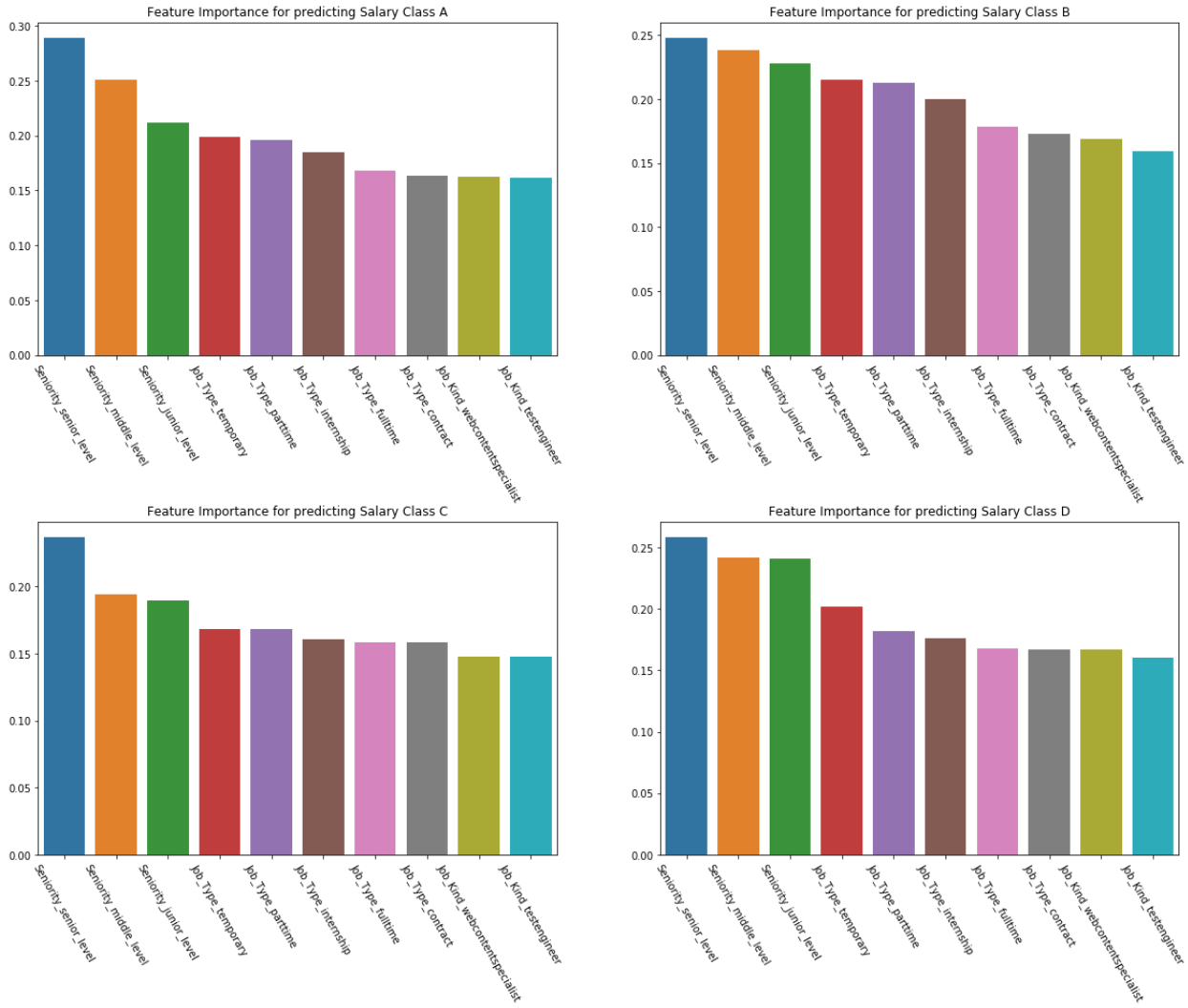
I started the analysis running one model per dataset to compare their initial performance and decide with which I would continue exploring in deep. Unfortunately, neither this combined dataset or the one with only the summary performed well. However, the base_df worked very good since the first model created, using only Logistic Regression with cross-validation so I went deeper in its analysis running also the following models:

- **A KNN model with bagging**: KNN stands for a k-nearest neighbours model, which works by checking the class of the nearest points to the one being predicted in order to classify it. Combining it with bagging, we can create new samples from our original dataset, to run several times our model and get improve our results by learning from each time.
- **A Decision Tree model with boosting**: in this case a decision tree works as a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test and each leaf node represents a class label and a decision taken. The paths from the root to leaf represent classification rules. In this model, although boosting is a very different method than bagging, it also allows us to obtain better results by running several models and predict from it.

However, beyond all these models, the first model created using Logistic Regression with cross-validation ended up being the one with the better accuracy, obtaining an overall score of 0.86, and a precision of at least 0.80 for all classes, which means that at least 80% of all the predicted values for each category was correct.

4. Feature importance

There are some clear features that determine if one job post is in one category or another:



Even though maybe the size of the labels doesn't help, we can conclude from these plots that the following features of our dataset are relevant when predicting the salary category:

- Seniority: as we can see, the tree levels impact very strongly in all categories, being the first coefficients in terms of absolute size.

- In the second place, comes the Job_Type.

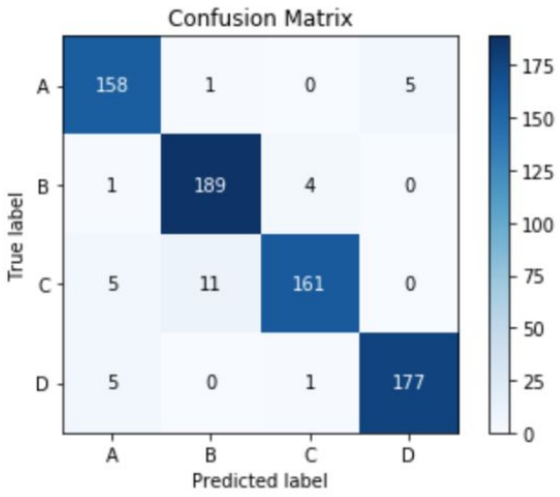
- And finally, we've for all the salary categories, 2 kinds of jobs, that are the titles scraped and cleaned from indeed: Web Content Specialist and Test Engineer.

This dataset contains hundreds of features, but it's nice to see there's a clear trend throughout the categories, at least regarding the most important features.

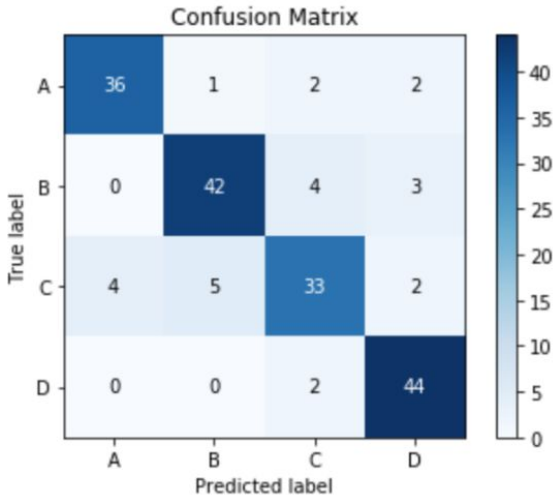
5. Conclusions and trustability

In the end, our Base Dataset ended up with a pretty trustable model. We can see in the following Confusion Matrices, that both the train and test groups, were predicted with a solid accuracy throughout the four salary categories:

TRAIN GROUP



TEST GROUP



One important last clarification is that, although our final model seems to be very accurate, it works very good to predict categories when the importance of them is equal and we don't have the need for pondering any class or classes. For example, is we were creating this model for a company, for which could be more important rather tell a client incorrectly that they would get a lower salary job than tell a client incorrectly that they would get a high salary job), our model would struggle. I wouldn't be able to predicting all the positive values as positive, without predicting also a lot of negative values incorrectly. So in that case, we should work on a new model with weighted categories.