**Document Parser**

**Name: Shashank Saumya**

This document will explain all the steps I have performed in detail.

**Table of Contents:**

1. Introduction
2. Exploratory Data Analysis and Feature Engineering
3. The model Pipeline
4. Hyperparameter Tuning
5. Results

Introduction

The dataset we have consists of multiple dataframes, with each dataframe corresponding to the image of a form. Every info from the form is extracted as token and takes up a row in the dataframe. The features available to us are,

start\_index and end\_index provide the location of the token, had the entire image been flattened into a line. Four columns correspond to the x and y coordinates of the top left and bottom right point of the token respectively. Then there is a transcript column which contains the information present in the token and finally, the field column (our label) which tells us what type of information the token contains.

Exploratory Data Analysis and Feature Engineering

This section is heavily inspired by the Exploratory Data Analysis done by Luis Fernando Torres’s [Wine Quality: EDA, Prediction and Deploy](https://medium.com/latinxinai/how-i-deployed-a-machine-learning-model-for-the-first-time-b82b9ea831e0) project.

The very first observation that we come across is that our data is very skewed. The “OTHER” label is much more in number than all the other labels combined (Fig. 1). This suggests that we should definitely go for either an undersampling of “OTHER” or an oversampling of the minority labels. Excluding the “OTHER’ label, we still have a slight skewness in our labels (Fig. 2), but it is not as drastic as compared to before.

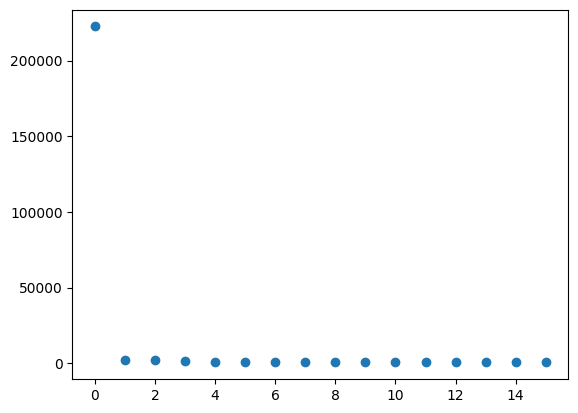


Figure : Count of each label

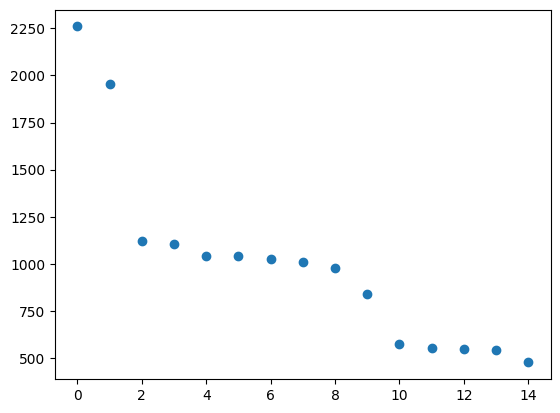


Figure : Count of all other labels

Now, we create 3 more features, namely, File No. which tells us which dataframe our row came from, index\_len which is obtained by subtracting the start and end index columns, x\_center and y\_center which correspond to the coordinates of the center point of the token. Furthermore, we get rid of the transcript feature. The reason behind this is that this feature contains all sorts of info, from numbers to codes (ex:72-8989-56) and strings. Now, this feature can possibly help us build a better model using some NLP techniques. However, there is a lack of time and experience to do so on my end. Albeit given more time, I could possibly learn more and do so. So, for our current situation, I have decided to get rid of it entirely.

Furthermore, we look at the correlation heatmap of the features to see which features are most correlated to our labels. The heatmap for the entire dataset is given in Fig,3 and the heatmap for the dataset excluding the “OTHER” label (we will call it **filtered dataset** from now on) is given in Fig. 4.

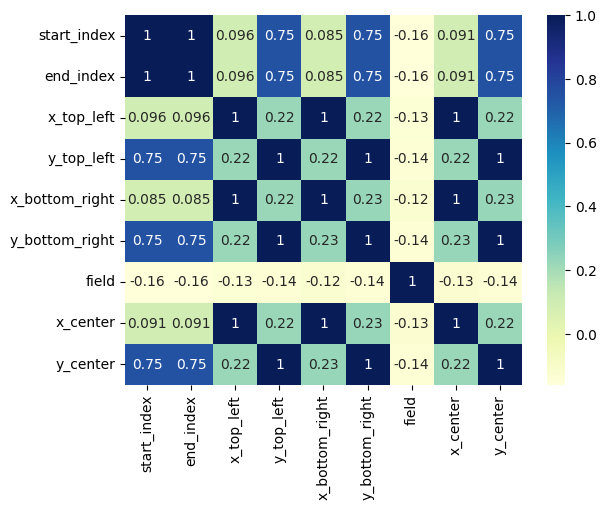


Figure : Heatmap of entire dataset



Figure : Heatmap of filtered datatset

Now, from these heatmaps, we see that x\_center and y\_center almost perfectly incorporate the information contained in the top left and bottom right coordinates. So, we shall get rid of these 4 features. Also, in the entire dataset, we see that the field is almost equally correlated to all the features. While in Fig. 4, this is not the case. This means, that overwhelming number of “OTHER” labelled data, (which is a placeholder label for any irrelevant information that we do not need) is the cause. So, I decide to use create two classification models. The first one classifies wether the data corresponds to an “OTHER’ label or not. If it does not belong to “OTHER” category then it is sent to a more specialised model whose task is to then further classify into the relevant labels. We shall discuss more on this later in the next section.

Now, we also look at the mean, std dev, etc of the data and find out that our data isn’t standardised or normalised. However, since we are going for a classification problem (specifically Random Forest for this case) we do not need to perform either of those. Furthermore, we also look at the violinplot and boxplot of the datasets as well.

Now, out of all the features left, we perform a RFECV (Recursive Feature Elimination, Cross Validated) to get the most important features that we will use to train our model. The features that we get are **start\_index, end\_index, x\_center, y\_center and index\_len.** All of this analysis was done in two jupyter notebooks “**Data Analysis full.ipynb**” and “**Data Analysis filtered.ipynb**” for the entire and the filtered dataset respectively.

The Model Pipeline

Now, let’s talk about our model. Based on the analysis above, I have decided to perform the classification at two levels. The first model is fed the entire dataset, but it the labels are set as “OTHER” and “not other”. Meaning its job is to simply classify wether the entry provided to us is of relevance to us or not. Then the second model is trained on the filtered dataset which means given that the token entry is relevant to us, its job is to identify what is the actual label of the entry. I have made a rough sketch of the training and testing processes by hand (Fig 5.). The entire process is as follows, while training, we begin by merging the entire dataset as one dataframe. And then feature engineer the columns. Then we make a copy of this dataframe as we need to perform separate changes for the two models. For the 1st model, we convert all the relevant labels to “not other” and then Undersample the data in order to take care of the skewed ratio. For the second model, we simply drop all entries with “OTHER” as the label. Then we train both the models on their respective datasets. As for the testing data, after feature engineering, it goes into model 1 first. If the outcome is “OTHER” it simply reported as it is. Otherwise, the data is sent to model 2 for further prediction.

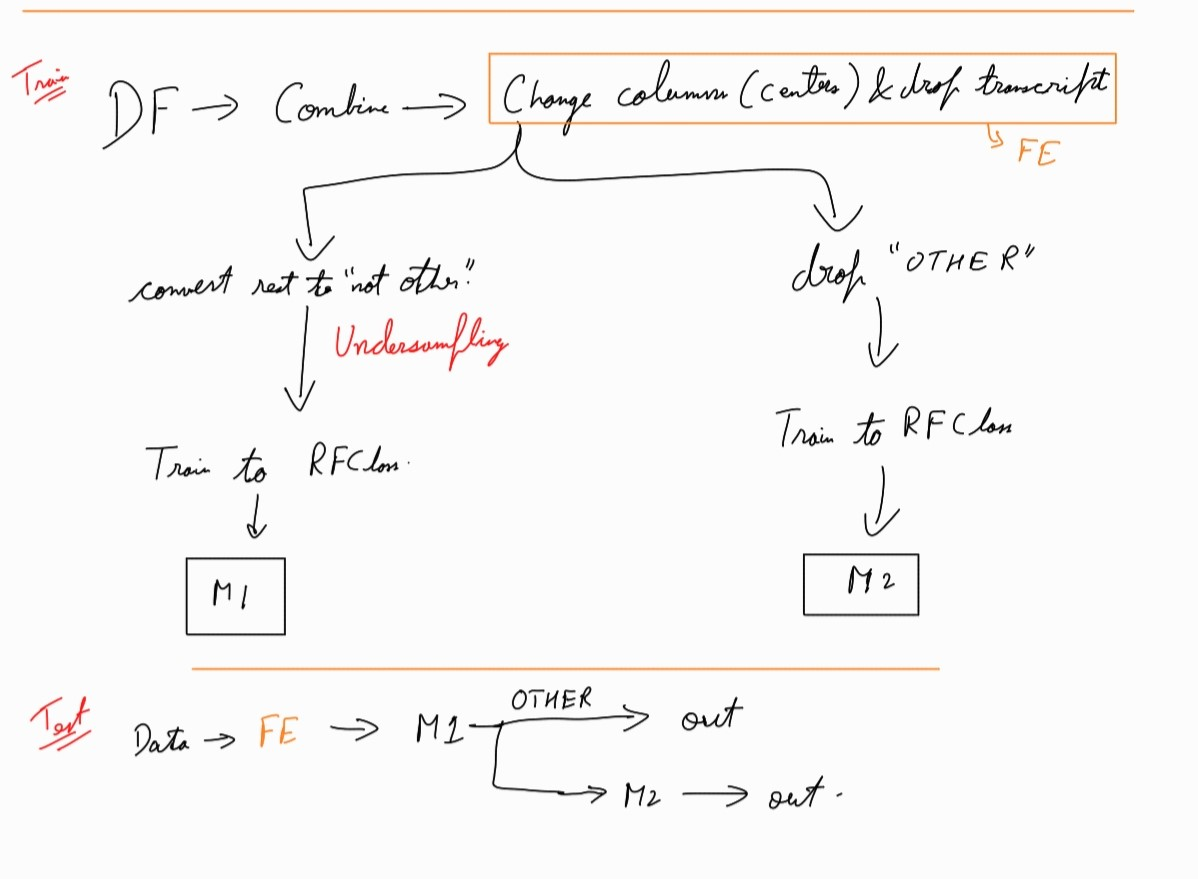


Figure : Rough sketch of Pipeline

Furthermore, for this project, I decided to go with Random Forest Classifiers as a baseline model. There are definitely more models that might perform better for this data, but in order to implement and check, I shall need more time.

**Note:** For the Under Sampling, I initially went for Clustering Centroid which uses Cluster Centres to replace the data which has huge count. This preserves the pattern it has while reducing its number. However, it was computationally very expensive, on my personal device, the code ran for hours before I had to manually interrupt. So, due to lack of time, I had to go for RandomUnderSampling instead.

Hyperparameter Tuning

In order to decide the n\_estimators parameter, I decided to go with Stratified K fold Cross Validation which retains the distribution of labels in each fold (further helps deal with our skewed data). After trying various ranges, I found out that the ideal value of ‘n\_estimators’ turns out to be 250 for both the models with an average accuracy of around 96.57% for the 1st and an average accuracy score of around 94.96% for the 2nd. So, we know that our model should work really well for this problem.

Then, we went ahead and separately trained both the models on their respective dataframes with ‘n\_estimators’ value range close to 250 (we got this from CV). The accuracy obtained by the 1st model is shown in Fig 6. And the accuracy for the second model is shown in Fig. 7. The accuracy achieved is quite high (94-96%) and the variation in accuracy is also quite small. This is why I can simply choose 250 estimators as my final tuned value and then go for training the model with the complete data.

All of the above analysis has been done in the jupyter notebook named **“Model Analysis.ipynb”**

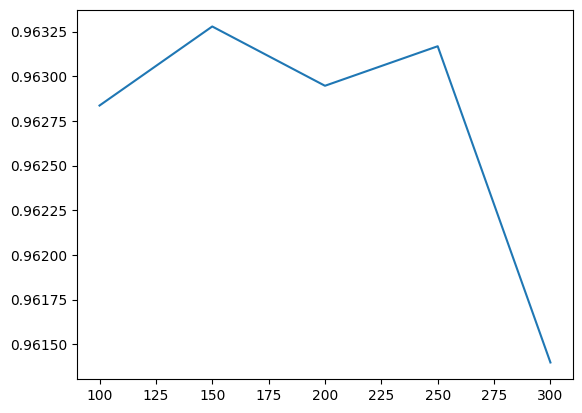


Figure : Accuracy vs n\_estimators for 1st model

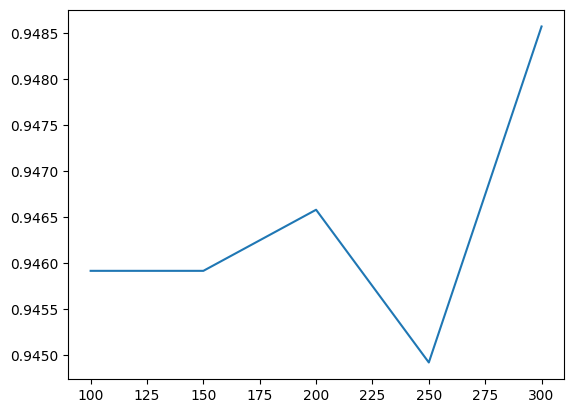


Figure :Accuracy vs n\_estimators for 2nd model

Results

So, finally I decide to go with ‘n\_estimators’ = 200 and then train the entire training dataset over it. Then, just to test our model a little, I use just one single file from the validation dataset. Surprisingly, the model gives me a 100% accuracy.

The entire code for this section is present in ‘main.ipynb”.