Capstone Project Report

1. Definition of the problem

Certain Bank in Colombia is trying to strength its Credit Risk Management models using client's tweets, after getting the consent of the consumers for use that information. The Bank will predict the level of income (lower, higher) based on client's tweets.

In order of achieving this, the bank collected user's tweets in certain places of the country where lived according with Colombian government high-income or low-income people. Depending on the place where the tweet is collected, the tweet was labeled in one of the categories.

The information that is available for the bank is a csv file with information of the tweet, the place where the tweet was collected, the date of the tweet, the user that made the tweet and the labeled tweet (high-income or low-income)

1.1. Possible Solutions

In order of solving bank's problem there were analyzed three possible solutions:

a. Make a rule based on the number of times people used certain words in tweets for each category. Based on this rule each word in the Spanish dictionary will be labeled as high-income word or low income-word, and depending on the times a high or low word were used in a tweet, the tweet will be labeled as high-income or low-income tweet.
b. Extract variables "correct spelling" and "word sentiment" variables to create a Machine Learning model which predict the label for each tweet.
c. Transform the people tweets in structured data using an algorithms of Natural Language Processing (NLP) like "Bag of Words", "Words embedding" or "TF-IF", extract additionally other type of variables as tweets sentiment, emojis sentiment or correct spelling of the tweet. Once the structured data is created, develop a Machine Learning algorithm of classification to create a model that based on the tweet classifies it in one of the labels.

1.2. Chosen Solution

After considering the three options, the solution that was selected was c. The solution a, can be a good approximation for solving the problem in a fast way, however it omits certain information models can capture, like order of words in a particular tweet, stemming of words, or correlation between certain words, and that can make the difference between a high or low income tweet. The solution b, extract certain features from tweets however it assumes that sentiment and spelling can predict correctly the income of people, and this is a very hard assumption, taking into account that tweets are short texts which can't predict correctly spelling of people, and both, high income or low income people can tweet in any mood.

The option c considers the relation of all possible variables, create a model that can predict based on statistical methods, without making hard assumptions. And is easily deployable for the bank consumption, so it was the solution selected.

1.3. Performance metrics

In order of evaluating the performance of the implemented model it was considered the next metrics:

a. Accuracy
b. F1 metric
c. AUC

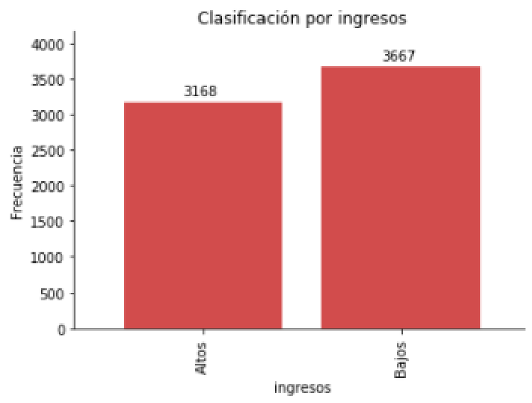
The selected metric was the AUC because it does not have problems when the classes are unbalanced, in addition it does not depend on threshold levels when it is selecting the best model and it depends on the classifier itself. Furthermore, in the literature AUC is traditionally selected to compare and select classification model.

2. Analyze the problem

The first variable that is convenient to explore to decide how to focus the algorithms, or if it is necessary to make extra processing with the data is, the objective variable, in this case the variable which define if a tweet corresponds to high-income or low income-people.

First according with the collected data, it will be defined if the classes for the problem of classification are balanced or not, so in Figure 1 it can be seen the total frequencies of data for each of the classes

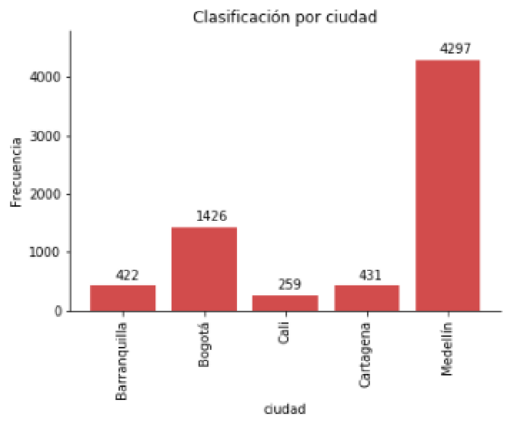
Figure 1

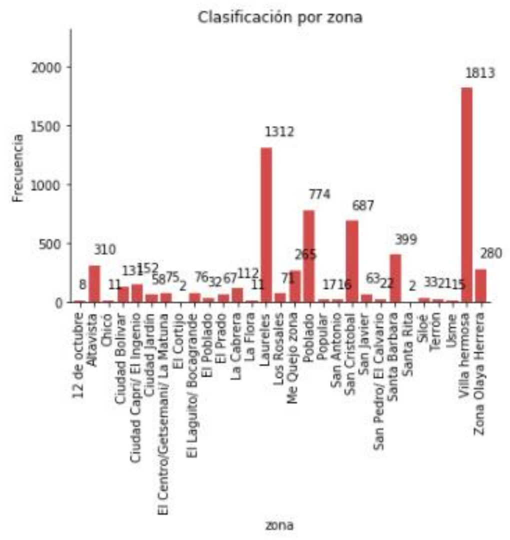


It can be seen in the figure that both classes have almost the same number of observations, 3168 observation for high-income tweets and 3667 for low-income tweets, consequently there was not used a method or algorithm to balance classes.

Another interesting thing in data that is a good idea to explore is if there's a particular concentration of tweets in certain zone of Colombia or in certain city, because this ca made that the final solution of the problem will be biased to certain region.

The number of observations for each city and zone can be seen in the figures 2 and 3.





It can be seen in the figures that the majority of data was collected in Medellin with more than 50% of total tweets, however each of the mayor cities is represented with more than 100 data points , and taking into account that there's no a big difference in the written language of any part of the country, it can be concluded that the final model will be representative for the mayor cities in the country.

In order of exploring the words used in each category of tweets, it was created a cloud of words for both high-income and low-income tweets, the word cloud is presented I the next figures:

Figure 4. High-Income words

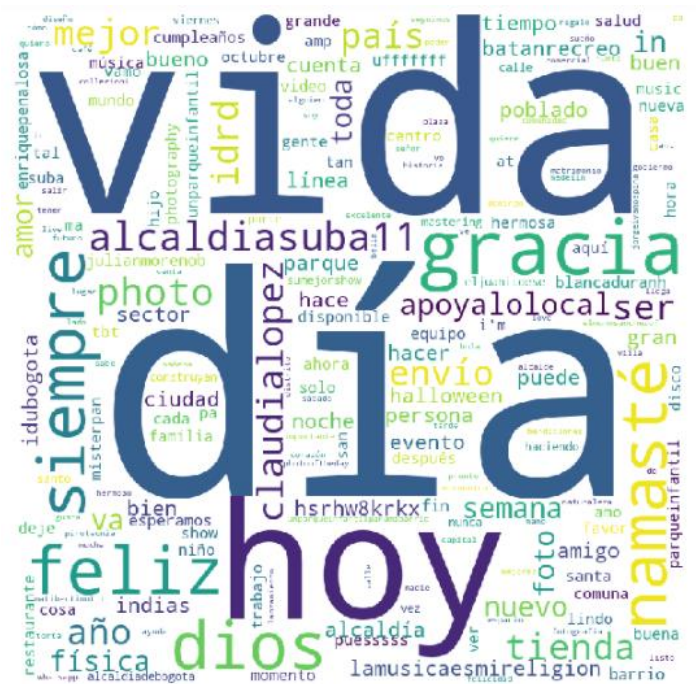
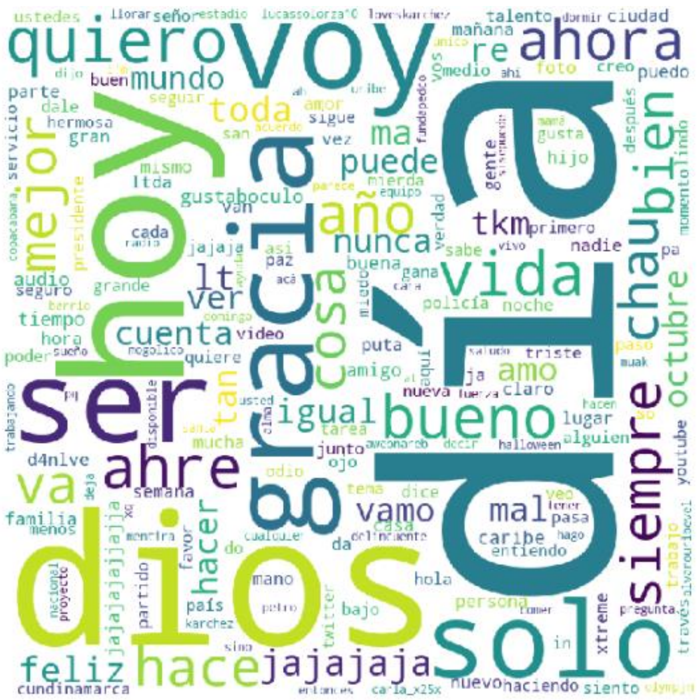


Figure 5 Low-income words



There are certain words that were used in both categories very much and which possibly can not help to predict the label of the tweet, as “día”, “gracia” or “hoy”.

However, there are particular words that can be associated to high-income tweet which does not appear in the low-income cloud as “vida”, there are words in low-income class too, like “tkm” or “ser” than can nor be seen in the high-income class, this words possibly will be used by the model to correctly classified each of the tweets.

Finally, there are words that even are used for both classes are more common in one of the classes than in the other. For example, “Dios” is a word that is much more common in low-income tweets, and “feliz” is much more common in high income tweets. This words with the presence or absence of other words in the tweet can help to make a separation of the classes.

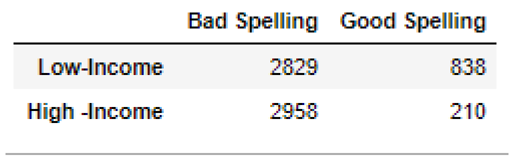
Once analyzed the information that we can extract directly from data, it was decided to create the variables of spelling, sentiment, and emoji-sentiment. The process of creation of these variables and their visualization relating with the objective variable are shown in the following sections:

a) Spelling variable: It was not possible to find a library that directly check the spelling in Spanish, however it was created a dictionary in Spanish, considering the genre and the number of the words. Based on this dictionary it was able to check if a tweet has good or bad spelling, when a tweet is written correctly. It is good to clarify that the method created to check the spelling does not check things like punctuation, grammar or use of lower and upper cases, it just checks that the word is correctly written.

   At the end, each tweet in the data was checked to mark with 1 if it has good spelling and with 0 if not. The range of the variable is binary.

   After creating the variable, it was made a confusion matrix to see if this variable can be a good candidate for predicting the category of the tweet. The result is shown in the next picture:

Figure 6



	Bad Spelling	Good Spelling
Low-Income	2829	838
High -Income	2958	210

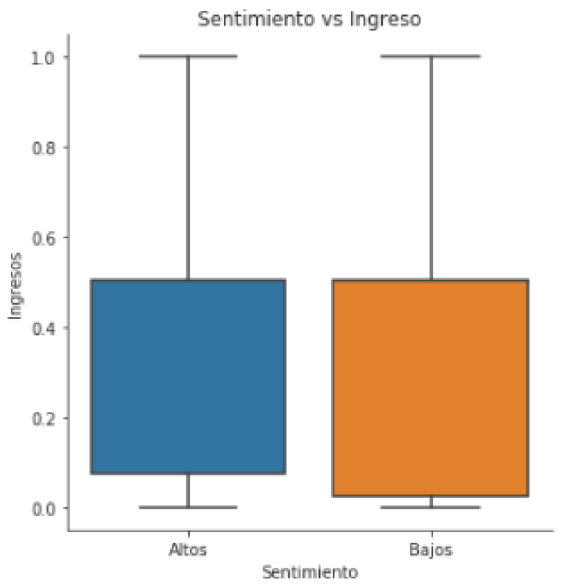
   It seems that there is no difference in spelling between high-income and low-income people. In fact, most people have bad spelling.

b) Sentiment variables: Based on a Pyton library “Sentiment_analysis_spanish” made by Hugo Bella[1] with permission of use, it was able to predict the polarity of sentiment of each tweet.

[1] https://pypi.org/project/sentiment-analysis-spanish/

This tweet generates a number between 0 and 1, where 1 represents a positive sentiment and 0 a negative one.
Applying the sentiment analysis for the tweets and relating the results with the objective variable, it was made the next figure:

Figure 7



It does not seem to have difference in the tweets' sentiment between high- and low-income people, so it is probable that this variable will not be significant in the final model.

c) Emoji variables: To capture if emojis can predict the income of people, it was created two new variables. The first one which helps to identify when a tweet have an emoji, and the second one a sentiment analysis based only in emojis.

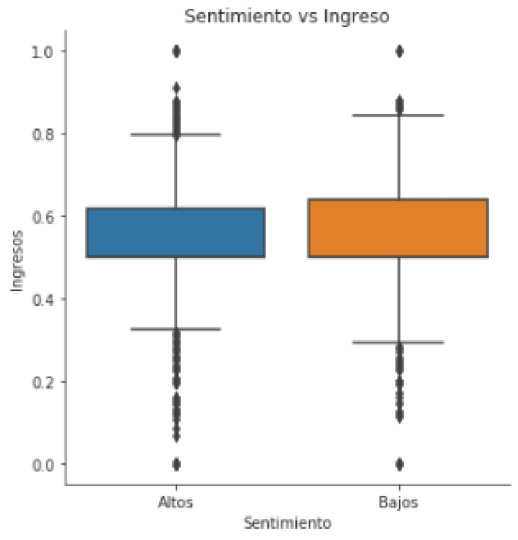
   The first variable was captured with the "emoji" python library which help to extract emojis and transform them to their correspondent word in English.

   For the second variable it was made the following steps:

   i) Extract the emojis for each tweet and delete the rest of the tweet.
   ii) Convert each emoji to their correspondent word in English.
   iii) Transform the results to words (eliminate special characters)
   iv) Apply a sentiment analysis library ("nltk.sentimet.vader") to extract the sentiment.
   v) Create a new polarity variable between 0 and 1 with the sentiment of the emoji, where 0 is a negative sentiment and 1 a positive one.

   After finding the sentiment of emojis, it was made a boxplot to check for the power prediction of variable.

Figure 8



Once again, it seems that the polarity of the emojis does not make a good prediction in this case.

Finally, to see how emojis are seen depending on the label class of the tweets, it was made a word cloud for high and low-income tweets:

Figure 9 High-income emojis

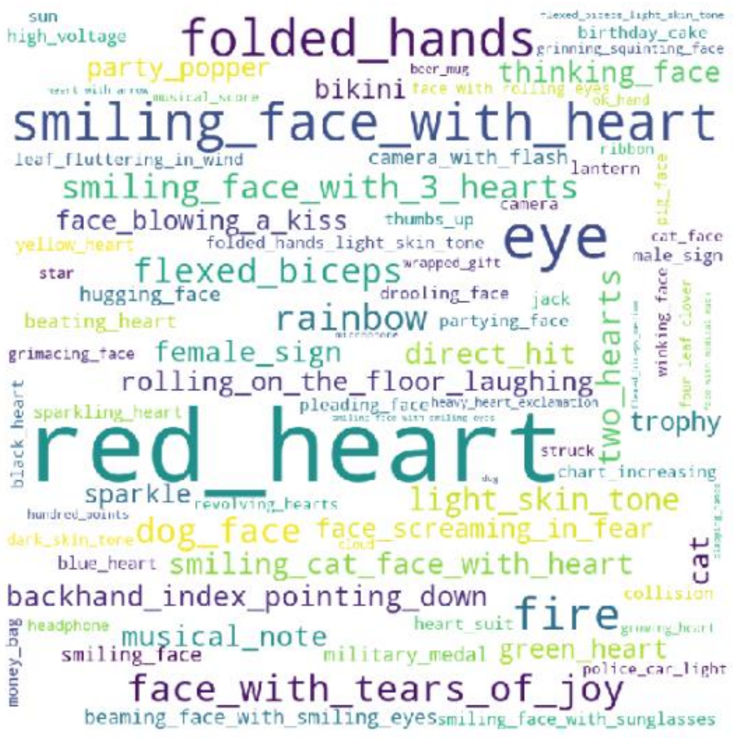
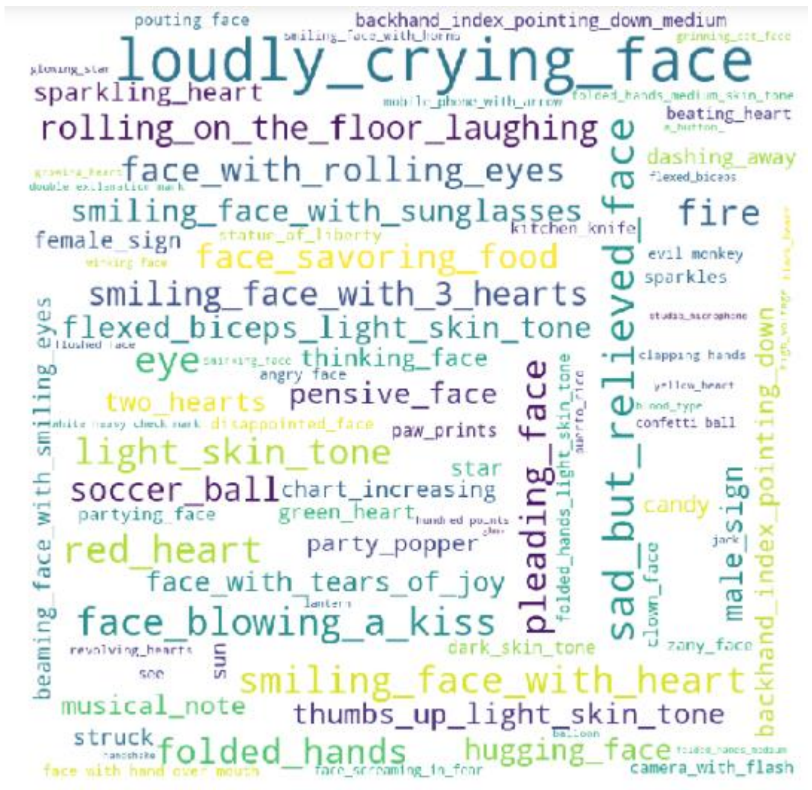


Figure 10 Low-income emojis



It can be seen by the images, that high-income people tends to use as “red heart” or “smiling face with heart”, and people with low-income income uses emojis as “loudly crying face” or “sad but relieved face”.

3. Implementation of algorithms

After visualizing and creating new variables useful for the problem it was preprossed tweets text according with “words embedding” and “bag of word” method, to be able to structure the unstructured data.

3.1. Preprocessing

3.1.1. Train-Test Split

Before transforming the data and considering, that the dictionaries made for structuring text data cannot use test data. It was splitting in this point the dataset in train and test data. The train data in this case corresponds to 80% of data, and the test corresponds to the 20% left.

It is not used a cross-validation approach because the construction and extraction of features with new dictionaries can be time consuming, and it is not considered that a train-test approach performs very different.

3.1.2. Transform text data

To be able to structure the text data it was made the next steps:

- Eliminate all special characters like "html" characters.
- Eliminate the spanish stopwords.
- Eliminate common words in Colombia characteristic of certain region or that does not tell anything of the tweet as Antioquía, Bogotá, Colombia, etc-
- Eliminate the special words that are common in tweets and that are difficult to interpret as tags or hashtags.
- Split each sentence in words and save the words in a list of words for each tweet.
- Stem each word with the spanish stemmer
- Save the list in a new variable

Depending on the method that will be used it was made extra steps to the list of words previously created.

Words embedding and *Bag of Words* methods to structure the data, for limitations of time, and taking into account that the tweets are made with few words, so a simple method like Bag of words could work better than TF-IF methods because of overfitting.

3.1.2.1. Words Embedding

In order of apply words embedding method the following additional steps were made to the previously created list of words:

- Create a dictionary of 5000 words with the most common words in the train data. It was chosen 5000 words because it is a number used for other similar applications.
- Assign a number to each word of the dictionary. And keep the 0 number to represent no word, and 1 to represent a word outside the dictionary.
- Convert tweets to a sequence of numbers each number representing a word. The maximum length of a tweet will be 35 considering the limitation of text in Twitter.
- Extract the total length of the tweet and use it as another variable to facilitate the embedding.

The final dataset will be the words embedding data with length concatenated with spelling and sentiment variables.

For the test data set the same steps will be made, but in this case a dictionary will not be created, but rather the same data-train dictionary will be used.

3.1.2.2. Bag of Words

To be able to apply the bag of words method, the following steps were made:

- Create a dictionary of 3000 words, based on training set tweets texts. It was used 3000 words in this case because with 5000 words there are many variables bearing in mind the dimension of the train dataset.
- Create a binary matrix, where the columns correspond to the 3000 words and the rows to each tweet. The value of the matrix is 1 if the word is in the tweet and 0 in any ither case.

The final dataset will be the matrix of bag of words concatenated with spelling and sentiment variables.

For the test data set the same steps will be made, but in this case a dictionary will not be created, but rather the same data-train dictionary will be used

3.2. Benchmark

In this case the benchmark chosen was a linear learner for classification (logistic regression) made by the technique of Bag of Words and a dictionary of 3000 words.

The model will be trained and tested in Sagemaker.

3.3. Models

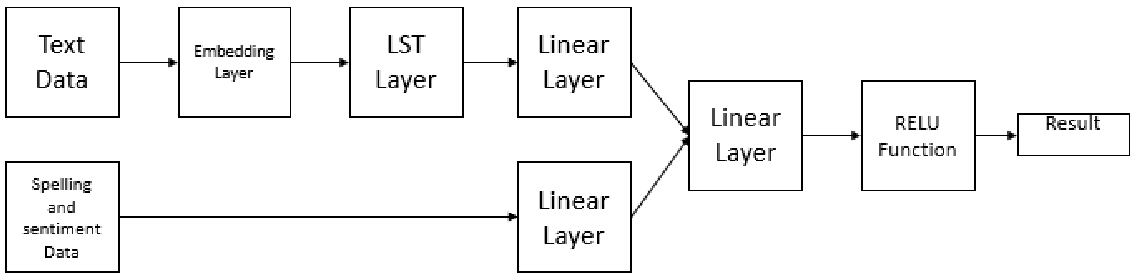
To be able to find the best model, the best performing models in similar applications were tested

3.3.1. Neural Networks on Pytorch

The first model to be tested was a neural network, in this case it was used the word embedding model with 5000 words dictionary, using the embedding layer in pytorch that is very useful for this kind of problems.

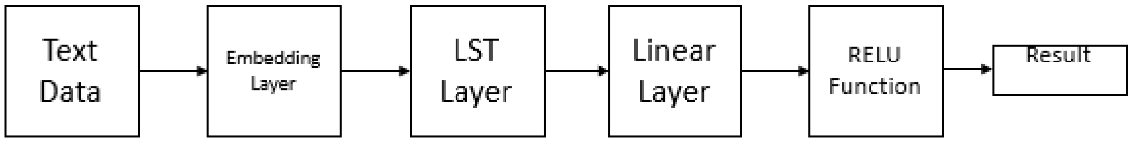
The architecture of the first model created was the next:

Figure 11



However, this architecture had problems when the model was trained, and did not converge easily, so considering that spelling and data sentiment seems to be not significant in the model and considering that simpler models are better. The final used model was:

Figure 12



This model only uses the data extracted for the word embedding method.

To find the best hyperparameters when training the model, a grid search optimization for hyperparameters with cross validation was made.

The hyperparameters that were calibrated in this case were the hidden dimension of the LSTM layer, that according with the literature is one of the most sensible hyperparameters, and the total of epochs.

There was made a total of 3 CV tests to find the best combination of epochs and LSTM dimension, epochs could be 15 or 50, and LSTM dimension could be 10, 100 or 200. So, a total of 18 model were trained and tested.

Embedding dimension was fixed to 35 and Linear layer dimension was 1.

3.3.2. XGBoost model

The XGBoost algorithm will be tested using the bag of words information and considering a word vocabulary of 3000.

In this case a 3CV method will be made, for tuning the max_depth and the subsample parameters, other XGBoost parameters were set as default. In this case max_depth could take the values 5, 10 and 20, and subsample could be 0.8, 0.5 and 1.

The best model in the CV average was trained and deployed in Sagemaker.

3.3.3. SVM with SKlearn

The last proven model was a Support Vector Classifier with *rbf* kernel. The C and gamma hyperparameters were tested with a 3CV method, and the metric to compare algorithms was AUC.

For the Cross-Validation optimization with grid search, both the regularization parameter C and the parameter gamma took values of 0.01, 0.32 and 10. The others hyperparamters were set as default.

For all the models the high-income tweets were labeled as 1, and the low income tweets were labeled as 0.

4. Results

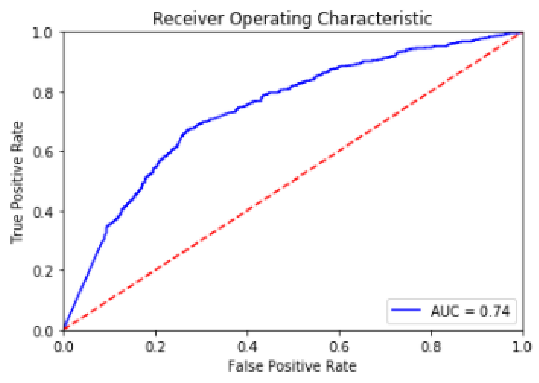
The results of the CV models and the chosen models in the test data are presented in this section

4.1. Performance of models

- **Benchmark**

The ROC curve for the Benchmark in the test data can be see in the next picture:

Figure 13



The AUC for this model was 0.74, which according with the assumptions made when collecting the data, is a very good AUC. This value is the metric which the other models will compare and will try to beat.

- **Pytorch Model**

After making the Cross Validation to select the best pair of hyperparameters, the following results was obtained:

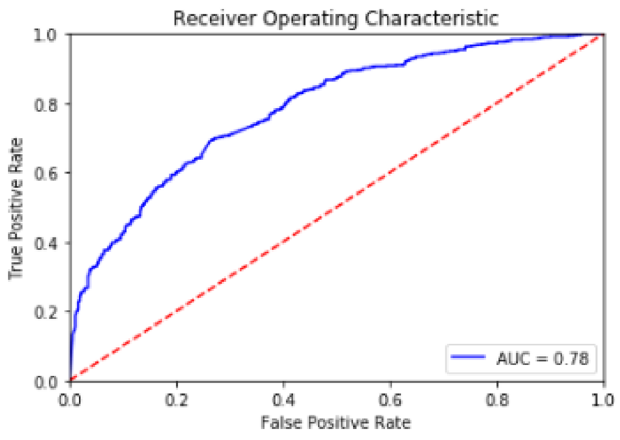
Figure 14

	Epochs	LSTM Dim	Avg AUC
0	15	10	0.998711
1	15	100	0.998805
2	15	200	0.998543
3	50	10	0.998332
4	50	100	0.998808
5	50	200	0.998788

The best combination of hyperparameters in this case was 50 epochs and 100 LSTM hidden dimensions. For this case AUC is very high for each of the combination of parameters, that can be explained with the fact that the vocabulary used is the same in all the cross-validation tests.

Using the best epochs and hidden dimension to create a model in Sagemaker and proving it in the test data generate the following ROC curve:

Figure 15

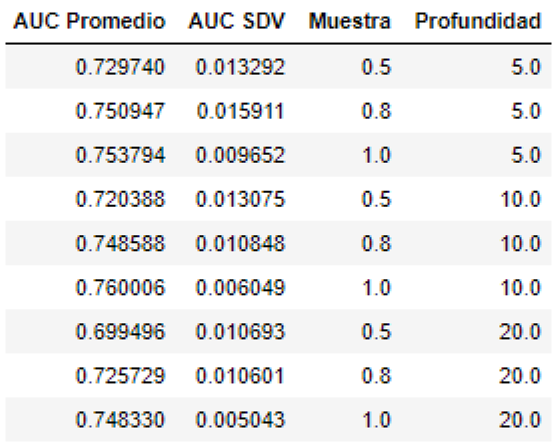


```
AUC: 0.78
```

The AUC for this case is 0.78, that is better that the one obtained with the linear classifier. Once again, considering the nature of the data, is a very goof value which can be enough for the purposes of the exercise.

- **XGBoost model**

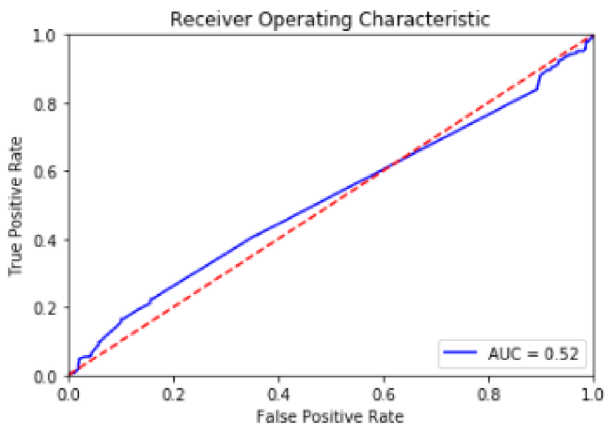
The result, for hyperparameter selection and Cross validation are shown in the next picture:



AUC Promedio	AUC SDV	Muestra	Profundidad
0.729740	0.013292	0.5	5.0
0.750947	0.015911	0.8	5.0
0.753794	0.009652	1.0	5.0
0.720388	0.013075	0.5	10.0
0.748588	0.010848	0.8	10.0
0.760006	0.006049	1.0	10.0
0.699496	0.010693	0.5	20.0
0.725729	0.010601	0.8	20.0
0.748330	0.005043	1.0	20.0

In this case the best AUC is achieved when the deep is 10 and there is not subsamples (subsample is 1). Using this model in SageMaker and proving with the test data, gets the following ROC Curve:

Figure 16



AUC in this case is 0.52. this is a very bad result and can be explained for the extension of dimension in columns that the input data has.

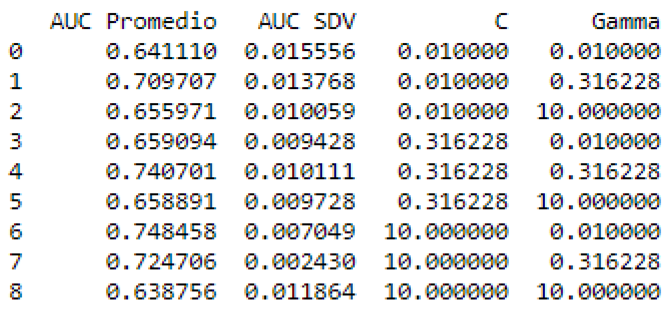
To improve this model, it is suggested to reduce the input dimension and prove with other hyperparameter combinations. The model is performing much worser than the benchmark.

However, as Pytorch model have already get a result which makes me satisfied, the recalibration of this model is suggested as a future work.

- **SVM**

The SVM results after making the grid search and the cross validation was:

Figure 17

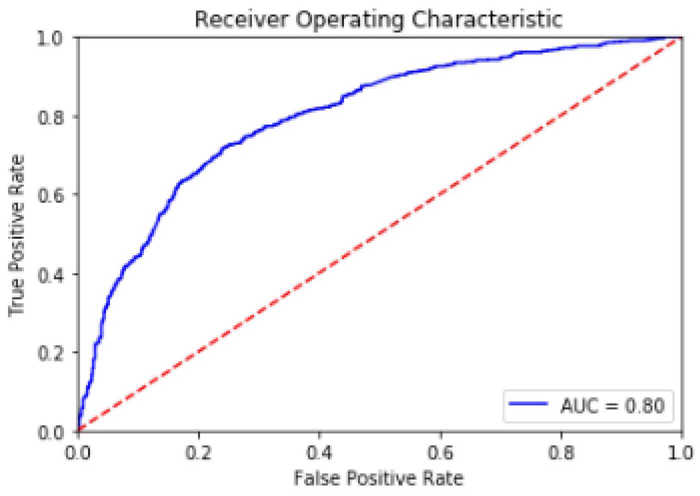


```
   AUC Promedio   AUC SDV          C      Gamma
0      0.641110  0.015556   0.010000   0.010000
1      0.709707  0.013768   0.010000   0.316228
2      0.655971  0.010059   0.010000  10.000000
3      0.659094  0.009428   0.316228   0.010000
4      0.740701  0.010111   0.316228   0.316228
5      0.658891  0.009728   0.316228  10.000000
6      0.748458  0.007049  10.000000   0.010000
7      0.724706  0.002430  10.000000   0.316228
8      0.638756  0.011864  10.000000  10.000000
```

In this case the hyperparameters that optimize the AUC for SVM is C equals to 10 and gamma equals to 0.01 with an AUC of 0.75.

Using this hyperparameters to train a model and test the model gives the next ROC curve for test data:

Figure 18



The AUC in this case was 0.8. This model got the best AUC of all the models proven, however it was almost the same that the one obtained with neural networks, the small difference can be explained for the randomness in the training of models, so we can conclude that both models have a similar performance. The model performed better than the benchmark, so it can be concluded that is a very good model.

4.2. Selection of model

In order of simplify a little the chosen model and considering that the Pytorch model and SVM get similar results in the AUC metric, the selected model to deploy in the application will be the Pytorch model.

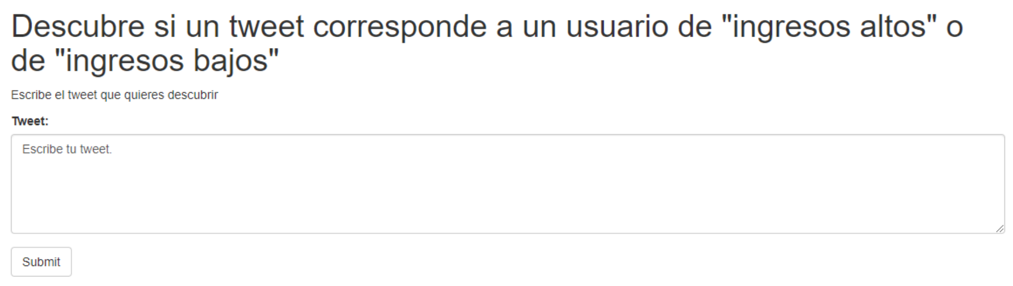
The Pytorch model was selected over the SVM model basically because it can use GPU so is fastest to train, and second because it does not require the variables of sentiment, emojis or spelling, that could be difficult to obtain in the deployed model and get the model more complex.

For the Bank perspective is equally important to have false positives because these clients can be potential defaulters in the future, or false negatives because with these clients the bank is losing a clear opportunity, so the final metric for the model will be F1. The threshold that maximizes the F1 for the Pytorch model was very close to 0.5, so 0.5 value will be used as threshold in the deployed model.

4.3. Deployment

The model was deployed using Sagemaker, and the endpoint created was connected directly to a Web Application, using AWS lambda and AWS Gateway Proxy. The web application was uploaded to a free website to prove it online.

The interface of the application can be seen in the next image:


Descubre si un tweet corresponde a un usuario de "ingresos altos" o de "ingresos bajos"

Escribe el tweet que quieres descubrir

Tweet:

Escribe tu tweet.

In this application the user writes a tweet and the web page tells if it correspond to high income tweet or low income tweet. The web page for the app is https://projectincomecolombia.000webhostapp.com/.

For avoiding extra charges in *Amazon* the endpoint is turned off, but it can be activated whenever the app is going to be proven.

5. Conclusions

5.1. Solving the problem

The final solution solves the problem of the bank for predicting the level of income of people, because based only in a set of words and with the use of the model, the bank can classify someone by their income. This tool can be very powerful to strength the credit risk management of bank because it can be used to define if a credit loan is conceded or nor.

The developed model predicts the level of Colombians income with a great confidence (AUC close to 0.8), so if the model is used in a set of tweets of the same user, the bank can predict the level of income of the user with even higher levels of confidence. This model can be used particularly for clients in the bank, that are in the grey zone, and the bank need to decide how much money it is going to lend.

The model works well with the vocabulary extracted as October 2020, however considering people language tends to change it is convenient to recalibrate it at least each 6 months, to capture the new words used. In addition it will be advisable to include even more cities in the country to represent all the country, or make a different model for each region if it is considered that language is different in each part of the country.

5.2. Problems and recommendations

Training and creating models for this problem in SageMaker had various inconveniences, specially with the creation of python files for training and predicting Pytorch models. It is recommended that each python file that was created for a custom model was previously tested, before using it in the estimator and predictor.

Train Sagemaker models can take long time, while uploading data and creating image for container, so if the model is not too large and it is not going to be deployed, is a better practice to train and test the model in the local machine.

Creating AWS lambda function was a problem specially for the absence of NLKT language in lambda environment, even when a container zip was uploaded to lambda, there was a problem with REGEX package, so it is extremely recommended to avoid using extra packages when using lambda and if it is possible make extra step in the predict python file.

5.3. Future works

To improve the present, work the following extra jobs can be made:

- Collect more data, including extra cities in the country and a longer window of time.
- Develop a better checker of spelling method for Spanish sentences, which detects punctuation or grammar.
- Create your own sentiment analysis in Spanish for sentences in Spanish.
- Use cross validation instead of train-test split
- Use lemmatization when transforming the words.
- Use TF-IF method to structure test data.
- Consider other Machine Learning algorithms to test
- Recalibrate XGBoost parameters, to achieve a better result.
- Consider testing other hyperparameters values.