Machine Learning Capstone Project

Comment Adivine

Definition:

Domain background. The project's domain traces back to a Customer Experience company that wants to feed from customer's feedback and wants to provide their clients with customer insight in real-time. They want to somehow predict the topic of a customer's comment. Based on the comment's characteristics, a supervised approach would be more convenient as the comments aren't long enough to take an unsupervised approach. Now, comments are being manually labeled by their employees and feed to the to the client's dashboard. They spend too much time labeling comments, therefore, clients don't have their customer's feedback until much later when it might be too late to improve the customer experience.

The data to analyze are comments from different hospital's clients in Spanish language from 2017 to 2019 obtained through a tablet terminal where the customers are asked how their experience at the Hospital was.
The Dataset contains the Name of the hospital, the comment and the topic of the comment. I will use this data to train a model after preprocessing some details.

Problem statement

The company cannot analyze in real-time the issues that the comments attend to. Comments must be manually reviewed every day and labeled by a person, the amount of comments per day is too big and can't be handled anymore by people. They need to provide real-time labeled comments so their clients can understand which are the main negative comments on specific topics day to day, so they can apply measures to improve their businesses.

Solution statement

Using a supervised machine learning algorithm and text analytics I would train a model with labeled comments to predict the label of each comment based on the the principles of TFIDF (short for term frequency–inverse document frequency). TFIDF measures not only the term count of a comment, but also the frequency compensating the most and less used terms along the data frame. This will create a matrix of terms and their respective label.

Benchmark Result.

How I chose and defined an efficient model to do this task was researching on the internet which algorithm would be the best for labeling in text analytics. I found all kinds of examples but none similar to my problem. The most interesting was Sentiment analysis, however it just defines whether a review infers a Positive or Negative sentiment. I needed an algorithm that predicts many labels based on different comments.

Nevertheless, Sentiment Analysis was the key to acknowledge what processes and techniques I would need in order to build a model that predicts various comments label. The basis of sentiment analysis is that it goes through each comment and it counts each word, so that comments that contains certain words have a more positive or negative inclination then it sums the numbers and determinates whether the review has more Negative or Positive words, you can do that using the library "Vader". (https://t-redactyl.io/blog/2017/04/using-vader-to-handle-sentiment-analysis-with-social-media-text.html)

I studied this case in a Udemy course called XXXXXXXX.
There you have movie reviews previously labeled as Negative or Positive and you convert the reviews in a matrix of words and counts.

The logic behind my Comment Adivine is similar. Comments with same or similar words would belong to the same Label. I would need a library that counts each word and their frequency, that's where TFIDF from Sklearn comes into play. Then I have a matrix of word's count and frequency, therefore I would need a model that could read a matrix or vectors.

I found two solutions:

Unsupervised model.
I thought that an Unsupervised model would be easier to train, as I wouldn't need labels in the first place. I did some research and found a library that could help to predict topics based on papers or reviews, LatentDirichletAllocation from Sklearn. I immediately noticed that this kind of models doesn't work with short comments, as they predict topics or groups of what the comments are about. It is quite hard to get an idea of what such a short comment is intended to belong in terms of topics, also it's hard to compensate for unbalanced classes.

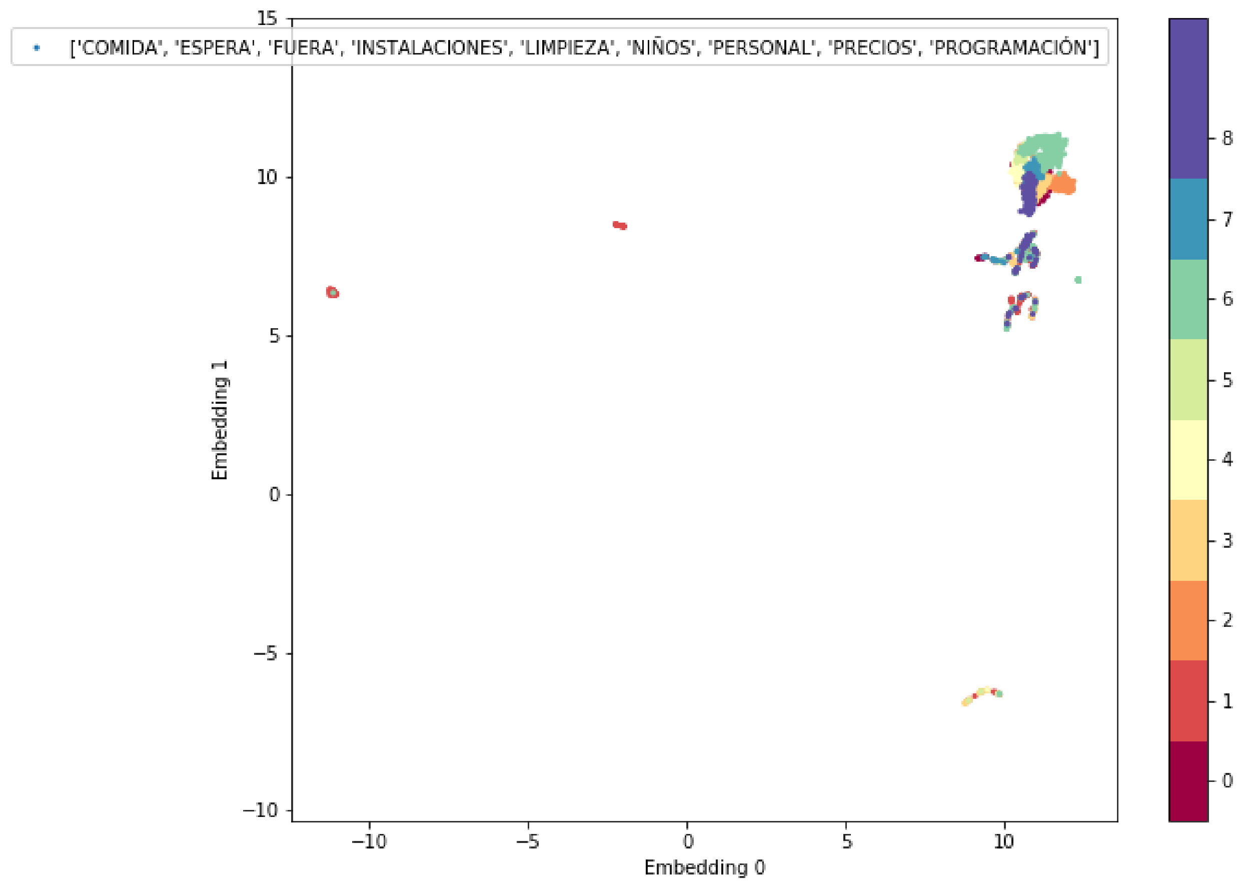


Figure 1: Topic Distribution

Comments from different Labels show mixed up with others, there is no clear separation between comments belonging to different Labels.

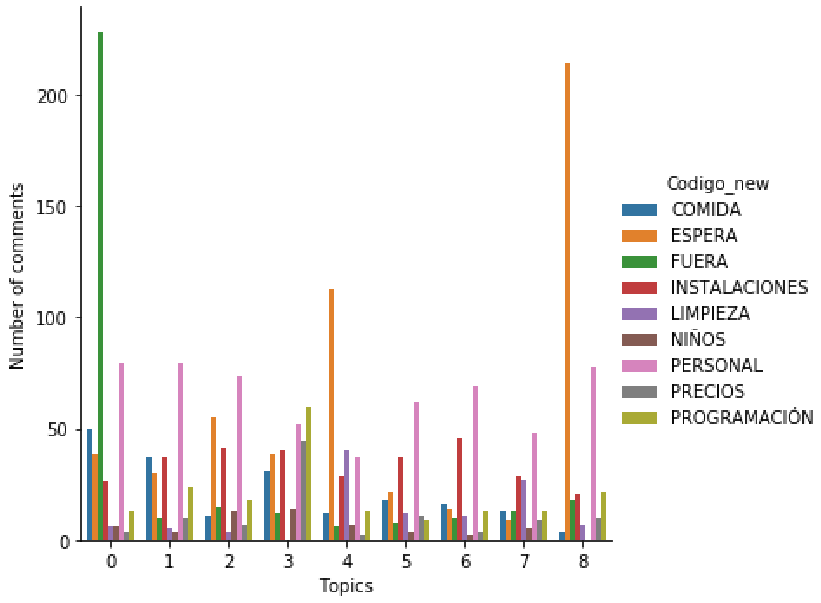


Figure 2 Topic Distribution 2

As you can see, Topics are mixed up between multiple different labels. This might be because of the length of the comments, which doesn't allow the model to identify a document (comment) as a single topic because is not long enough to comprehend the context and make a full matrix of words to compare to.

A Supervised model

After getting an idea about how my data is distributed, I considered that a Supervised model would be the best choice. I will work with vectors as my project is like that of sentiment analysis paper that I studied on Udemy course: "NLP – Natural Language Processing with Python". So, I decided to go for vector model as the course recommended. I simply compared different vectors models to see which one would give a better accuracy in a raw test. Here are the results:

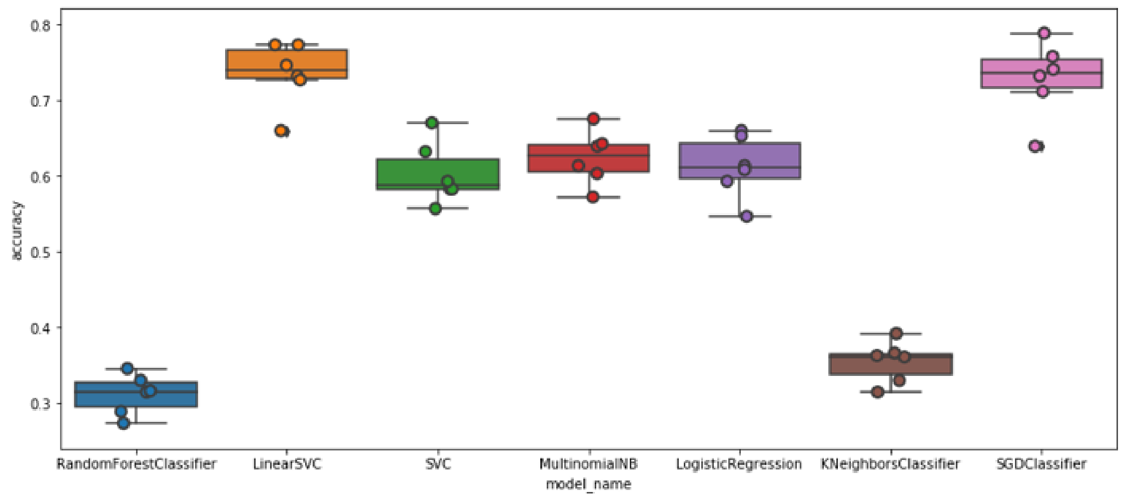


Figure 3 Model Efficiencies

Both Linear SVC and SGD Classifier proven the algorithm with most accuracy in comparison to other models. I believe a vector based algorithm is a good approach for this issue as it will transport a matrix of term frequency vectors to a 2 or 3 dimensional plane where the comments with similar term frequency of words would be close to each other and share the same labels. These kinds of models allow to predict multiple classes, models such as Logistic Regression don't work well with too many classes.

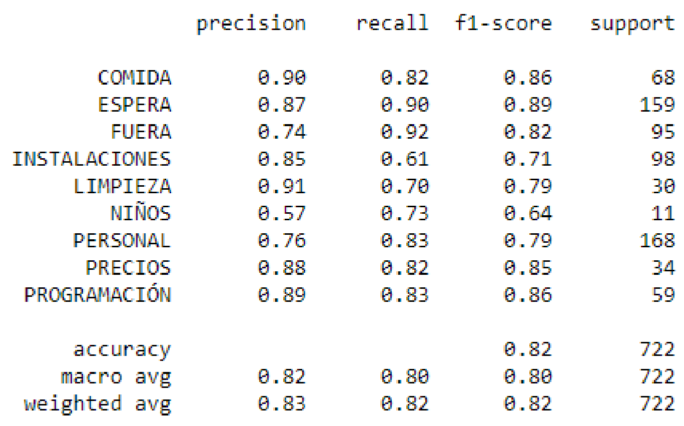
Another model that could work is Naive Bayes, however that approach was more complicated and sensitive and the results of a geometric model as SVM can give are better and easier to compute. Also, Naïve Bayes requires a big data set to train and it's frequently used to predict two classes. (Source: https://stackoverflow.com/questions/35360081/naive-bayes-vs-svm-for-classifying-text-data/35360814#:~:text=The%20biggest%20difference%20between%20the,rbf%2C%20poly%20etc.) .)

This model along with a Web App where the customers can upload their comments and get immediately predictions on what's the comment talking about, will provide the Customer Experience company a huge potential as they will be able to identify in real time their businesses issues.

Evaluation metrics

The solution for the problem is to predict a real-time comment's topic/label related to hospital affairs such as (Food, Waiting Time, Stuff, Cleaning, Facility, Unappropiated (such as badwords), Kids, Appointments, etc.).

This is important for reporting as they can update their client with categorized comments about what's wrong in the hospital and which comments are the most frequent. Therefore, the metric needed to evaluate the performance of the model must be the "Recall" and "Precision", since the classes are unbalanced, "Accuracy" wouldn't work for our model. We need to make sure that the model predicts most of the comments correctly and with high precision or false positives in each Label.



```
              precision    recall  f1-score   support

      COMIDA       0.90      0.82      0.86        68
      ESPERA       0.87      0.90      0.89       159
       FUERA       0.74      0.92      0.82        95
INSTALACIONES       0.85      0.61      0.71        98
    LIMPIEZA       0.91      0.70      0.79        30
       NIÑOS       0.57      0.73      0.64        11
    PERSONAL       0.76      0.83      0.79       168
     PRECIOS       0.88      0.82      0.85        34
 PROGRAMACIÓN       0.89      0.83      0.86        59

    accuracy                           0.82       722
   macro avg       0.82      0.80      0.80       722
weighted avg       0.83      0.82      0.82       722
```

Figure 4 Model Metrics

Analysis:

1.The dataset is composed of xxx comments from visitants from hospitals in 2017-2019. They asked questions about their experience in the hospital and what would they change or didn't like.

I filtered those comments exclusively from hospitals sourcers and remove some of the predefined labels that was provided by the company as we want to focus on those comments relevant for the business.

Comments with label POS (Positive), Negative or Other (Otros) are not useful for our intention of predict specific comments about cocnrete hospital complaints.

Then we filtered those comments whose label have at least 30 comments, otherwise the model wouldn’t be efficient enough to predicts labels from comments that are so infrequent.

Finally, I created a length column based con word count and character count in case I consider to include this data to the model.

Then is time to preprocess the data. Thi include: lowercase reviews, remove stopwords and punctuation, stem and/or lemma words.

I used NLTK and SnowballStemmer libraries to create a list of Spanish stopwords and stemm Spanish words.

2. Once preprocessing is done I saved the processed data into a custom directory and proceed to do some data visualization to get some insight about the data.

I calculated the labels distribution and the comments length distribution to see how balanced the sample is:

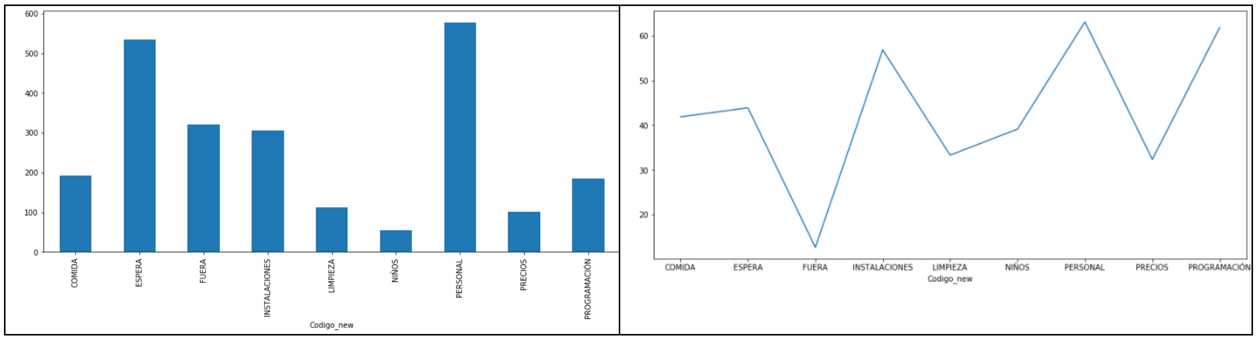
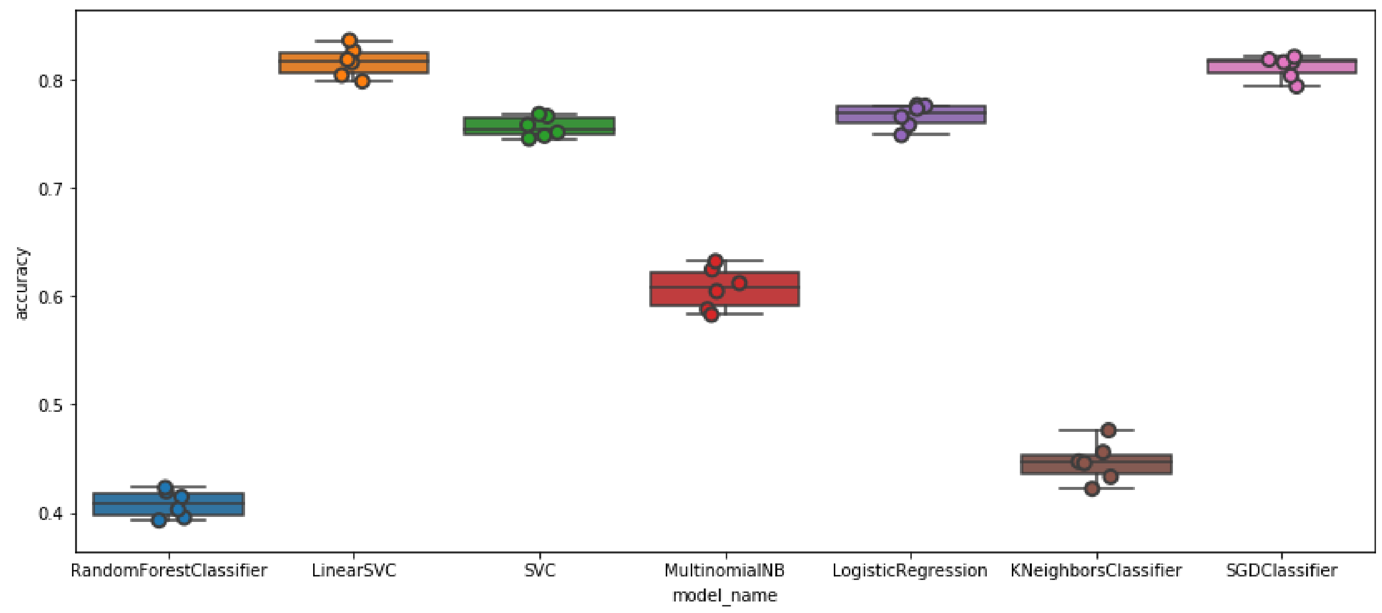


Figure 5 Data set distribution: Count and Length of comments

Also I dropped any empty row which might be a result of removing stopwords.

3. After getting an idea about how my data is distributed I considered different model that could fit my data and my target I read about text analysis and since it works with matrix of numbers, I chose these algorithms that takes matrix and vectors of numbers the result of their accuracy accordingly to my train data are the following:



In order to fit the model I had to vectorize the data frame of comments. There were two approaches.

a. Usin CountVectorizer from Sklearn library which would make a matrix of terms occurrences.
b. Using TFIDF (short for term frequency–inverse document frequency) from Sklearn library. TFIDF measures not only the term count of a comment, but also the frequency compensating the most and less used terms along the data frame.

I decided to approach the second option as when exploring the data set I noticed that there are some infrequent words that carry a lot of importance to identify a comment as a specific category.

That info can be found in "Label_predictor/wordcount-CleanComment.xlsx" or "Label_predictor/wordcount.xlsx"

4. After I selected the most efficient model for my data (Linear SVC) I trained a provisional model in the notebook instance using Sklearn package to make some arrangements.

First I trained this model and took this provisional model vocabulary and sorted it by their coefficients. I also considered an optional function called update_vocabulary() that would update model's vocabulary adding the words I considered important for the model. But didn't apply it.

Then I plot a graph that calculates accuracy according to the length of the sorted vocabulary. Then I did the same with the number of N-grams range my TfidfVectorizer object would take:

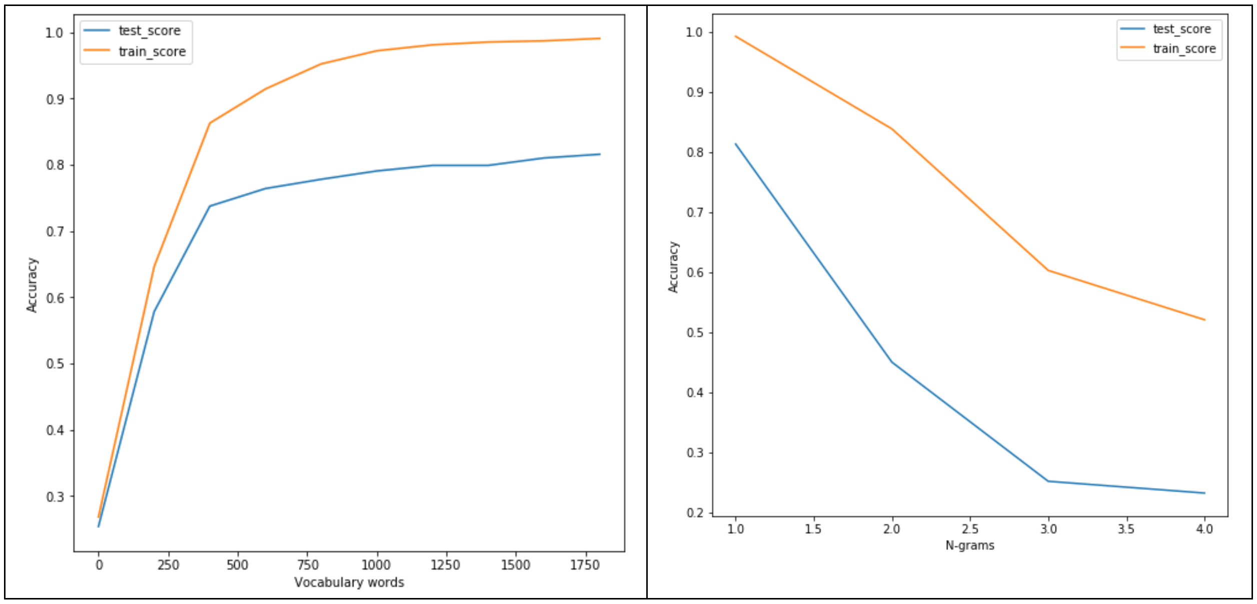
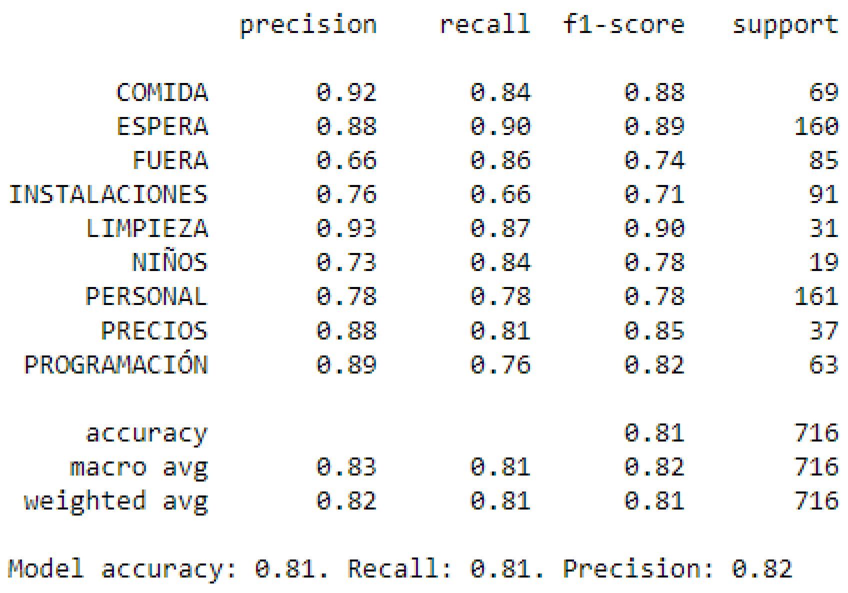


Figure 6 Model Efficiency by Vocabulary length and N-Grams number

Metrics:

Finally, I trained the provisional model using that insight and calculated model metrics:



```
               precision    recall  f1-score   support

       COMIDA       0.92      0.84      0.88        69
       ESPERA       0.88      0.90      0.89       160
        FUERA       0.66      0.86      0.74        85
INSTALACIONES       0.76      0.66      0.71        91
     LIMPIEZA       0.93      0.87      0.90        31
        NIÑOS       0.73      0.84      0.78        19
     PERSONAL       0.78      0.78      0.78       161
      PRECIOS       0.88      0.81      0.85        37
  PROGRAMACIÓN      0.89      0.76      0.82        63

     accuracy                           0.81       716
    macro avg       0.83      0.81      0.82       716
 weighted avg       0.82      0.81      0.81       716

Model accuracy: 0.81. Recall: 0.81. Precision: 0.82
```

Figure 7 Evaluation Metrics

Model efficiency overall looks good. There are some false positives and negatives, but we need to consider that some reviews are ambiguous and therefore they might belong to more than one category. Due to that, I decided to calculate how predictions and real labels are distributed:

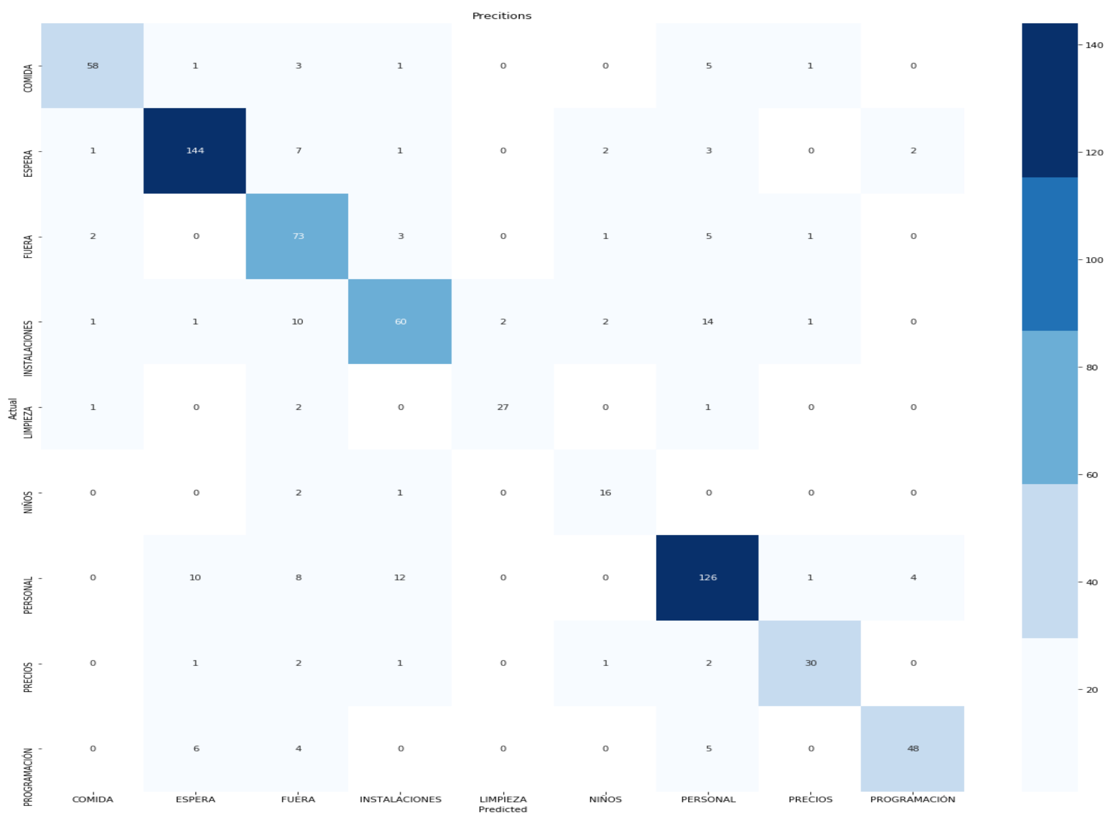


Figure 8 Predictions distribution: False Negative and Positives

I needed to check if the training examples were correctly labelled, so I created a dataframe with the predicted comments, the real label and the prediction label and checked one by one whether the prediction label was more accurate that the real one. After all, the data was manually labeled by humans and they might be errors as well on labeling them.

Doing this I noticed that some comments belong much more to the predicted label than to the real label they had, meaning that some comments were mislabeled at the beginning. So, I went through the original training data set (data.xlsx) and manually check whether the comments were accordingly labeled and corrected the labels in case it wasn't properly done. Then run the whole code again.

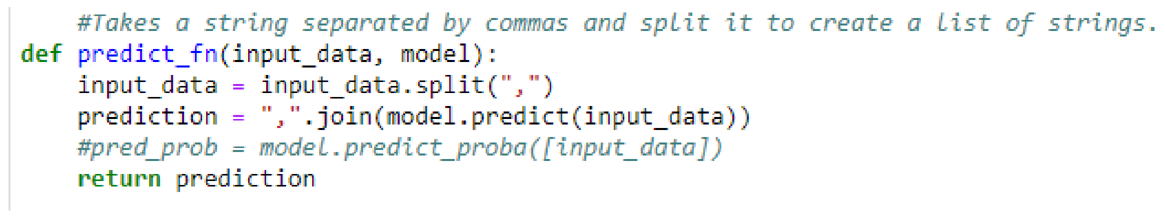
Doing this improved my model precision.

Modeling:

It is time to set up the model using AWS.

1. I created a function to save my train and test data into my Notebook directory.
2. Upload this data to an S3 bucket.
3. I had prepared my script.py script ready to train the build-in Sklearn model in AWS.
4. Create a Sklearn estimator where entry_point = "script.py" file and source_dir = "source_train". I had to some arrangements in the predict method that I will explain later (point 8).

5. Fit the model with the train data uploaded in S3.
6. Deploy the model.
7. Create a Lambda Function that would take a comma separated string and preprocess that string in a format the model could read using the function “comment_processing_csv”.
8. Modify the predict_fn() in the script.py in order it takes as input_data a string of comments separated by commas and splits it to create a list of strings so the model can read it and return the predictions as a single string of predictions separated by commas.



```
    #Takes a string separated by commas and split it to create a list of strings.
def predict_fn(input_data, model):
    input_data = input_data.split(",")
    prediction = ",".join(model.predict(input_data))
    #pred_prob = model.predict_proba([input_data])
    return prediction
```

9. Set up the API Gateway.
10. Deploy a web app that takes a CSV file and reads each row as a single comment separated by commas.

Roadmap:

1.We go to our Web app on the index.html file or using my own domain: https://www.brunopizzani.com/Comment_Adiviner_CSV.html, and upload a CSV where each row is a comment or review.

2.The web app transform the CSV’s first column into a single string with the rows from the csv separated by commas.

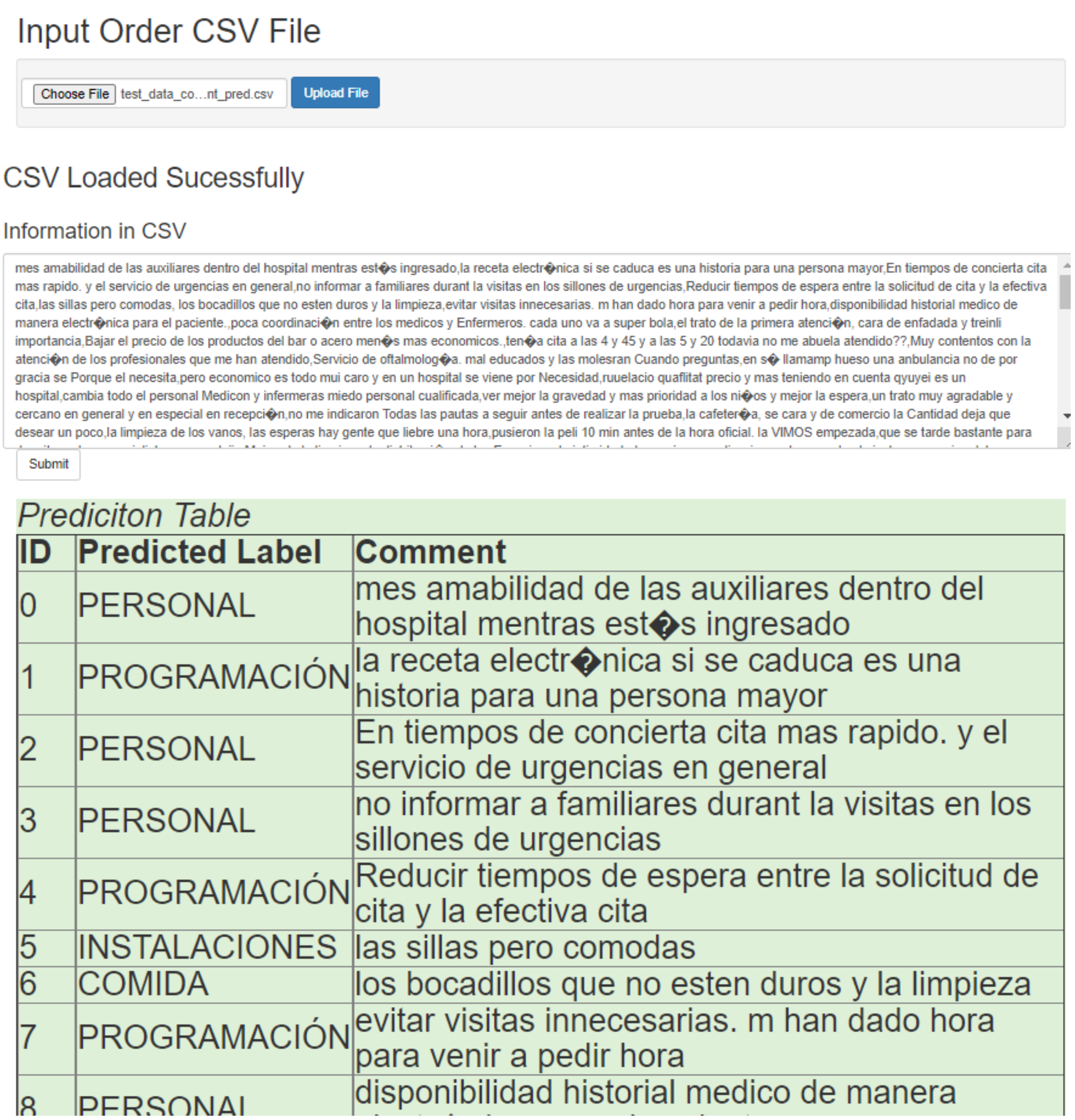
3.This string is sent to our lambda function that will use the function “comment_processing_csv” to preprocess the data in a format the model can read and give an output prediction.

4. Send the string to the model to make predictions.

5.The model returns a list of predictions per each comment, but I modified the predict_fn() so that it returns a single string of predictions separated by commas, otherwise the lambda function can’t read the output:

['PRECIOS', 'PROGRAMACIÓN', 'PERSONAL', 'PERSONAL'] >> 'PRECIOS,PROGRAMACIÓN,PERSONAL,PERSONAL'

6. Send the prediction string to the Web App where I create a dictitionary containing the comma separated predictions and the comments that were read at the beginning. Then it returns a HTML table with the comments and their respective predicted label.



CSV Loaded Sucessfully

Information in CSV

mes amabilidad de las auxiliares dentro del hospital mentras est�s ingresado,la receta electr�nica si se caduca es una historia para una persona mayor,En tiempos de concierta cita mas rapido. y el servicio de urgencias en general,no informar a familiares durant la visitas en los sillones de urgencias,Reducir tiempos de espera entre la solicitud de cita y la efectiva cita,las sillas pero comodas, los bocadillos que no esten duros y la limpieza,evitar visitas innecesarias. m han dado hora para venir a pedir hora,disponibilidad historial medico de manera electr�nica para el paciente.,poca coordinaci�n entre los medicos y Enfermeros. cada uno va a super bola,el trato de la primera atenci�n, cara de enfadada y treinli importancia,Bajar el precio de los productos del bar o acero men�s mas economicos.,ten�a cita a las 4 y 45 y a las 5 y 20 todavia no me abuela atendido??,Muy contentos con la atenci�n de los profesionales que me han atendido,Servicio de oftalmolog�a. mal educados y las molesran Cuando preguntas,en s� llamamp hueso una anbulancia no de por gracia se Porque el necesita,pero economico es todo mui caro y en un hospital se viene por Necesidad,ruuelacio quaflitat precio y mas teniendo en cuenta qyuyei es un hospital,cambia todo el personal Medicon y infermeras miedo personal cualificada,ver mejor la gravedad y mas prioridad a los ni�os y mejor la espera,un trato muy agradable y cercano en general y en especial en recepci�n,no me indicaron Todas las pautas a seguir antes de realizar la prueba,la cafeter�a, se cara y de comercio la Cantidad deja que desear un poco,la limpieza de los vanos, las esperas hay gente que liebre una hora,pusieron la peli 10 min antes de la hora oficial. la VIMOS empezada,que se tarde bastante para

Prediciton Table

ID	Predicted Label	Comment
0	PERSONAL	mes amabilidad de las auxiliares dentro del hospital mentras est�s ingresado
1	PROGRAMACIÓN	la receta electr�nica si se caduca es una historia para una persona mayor
2	PERSONAL	En tiempos de concierta cita mas rapido. y el servicio de urgencias en general
3	PERSONAL	no informar a familiares durant la visitas en los sillones de urgencias
4	PROGRAMACIÓN	Reducir tiempos de espera entre la solicitud de cita y la efectiva cita
5	INSTALACIONES	las sillas pero comodas
6	COMIDA	los bocadillos que no esten duros y la limpieza
7	PROGRAMACIÓN	evitar visitas innecesarias. m han dado hora para venir a pedir hora
8	PERSONAL	disponibilidad historial medico de manera

Figure 9 Web App

String Processing steps:

- **We receive a string from Web App -**

String_v1= “this is a review, this is another review, this is a third review”

- **We preprocess the string -**

String_v2 = “review, another review, third review”

String_v2 is passed to predict_fn() –

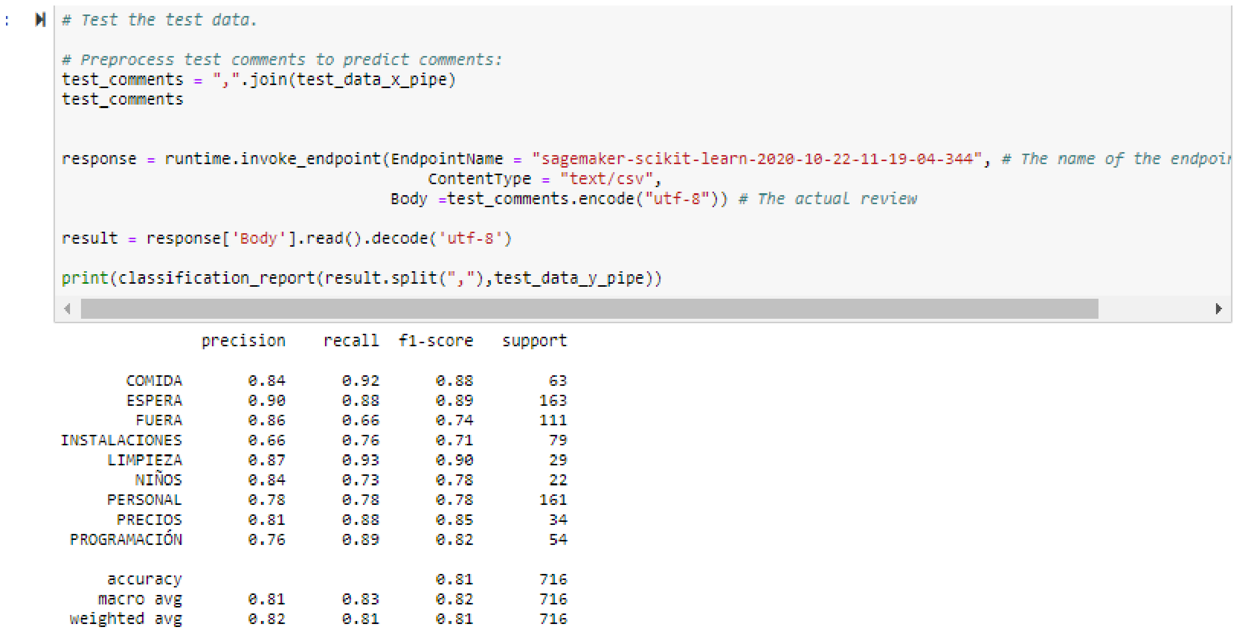
- Predict_fn() transform the string. String_v3 = [“review”, ”another review”,” third review”]
- And returns String_v4 = “,”.join(predic(String_v3)) to Lambda.

String_v4 = “LABEL1,LABEL2,LABEL3”

Finally, Lambda sends this to our Web App and the HTML code transforms the info into a table.

Results:

The model is deployed. Now let's test the test data in the deployed model:



```
# Test the test data.

# Preprocess test comments to predict comments:
test_comments = ",".join(test_data_x_pipe)
test_comments


response = runtime.invoke_endpoint(EndpointName = "sagemaker-scikit-learn-2020-10-22-11-19-04-344", # The name of the endpoi
                                       ContentType = "text/csv",
                                   Body =test_comments.encode("utf-8")) # The actual review

result = response['Body'].read().decode('utf-8')

print(classification_report(result.split(","),test_data_y_pipe))
```

```
               precision    recall  f1-score   support

       COMIDA       0.84      0.92      0.88        63
       ESPERA       0.90      0.88      0.89       163
        FUERA       0.86      0.66      0.74       111
INSTALACIONES       0.66      0.76      0.71        79
     LIMPIEZA       0.87      0.93      0.90        29
        NIÑOS       0.84      0.73      0.78        22
     PERSONAL       0.78      0.78      0.78       161
      PRECIOS       0.81      0.88      0.85        34
 PROGRAMACIÓN       0.76      0.89      0.82        54

     accuracy                           0.81       716
    macro avg       0.81      0.83      0.82       716
 weighted avg       0.82      0.81      0.81       716
```

Figure 10 Testing the Final Model

Recall and Precision is high as mentioned in the previous Analytics section. This means that the model is able to predict correctly the 81% of the comments with a high precision of 81% in average. Meaning that when the model predicts a Label it has an 80% probability of being right. That's a huge step considering that until now there were people manually reading and labeling comments. There might be false positive or negatives, but still is a huge step forward. Also need to take in mind class unbalance and ambiguous comments that may belong to more than one class.

Further steps to consider a better performance might be review the training data looking for human misclassification when labeling the training data. Remove additional stopwords, consider apply a 2-gram vectorization (which I tried, and it wasn't useful as the length of the comments is short and there are many single words that give important information about what's the comment about...). Also removing irrelevant Labels or group some labels together. Seems that the label *Instalaciones* (Facility) is not being predicted very efficiently, probably it shares some vocabulary with other labels meaning they might be similar and we could group them together. This would depend on the company's demands and needs.

So far the results are good but we should check the results on brand new comments. However, we would need to manually check whether the comments match the predicted label as these new comments are unlabeled and make arrangements consequently.