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.

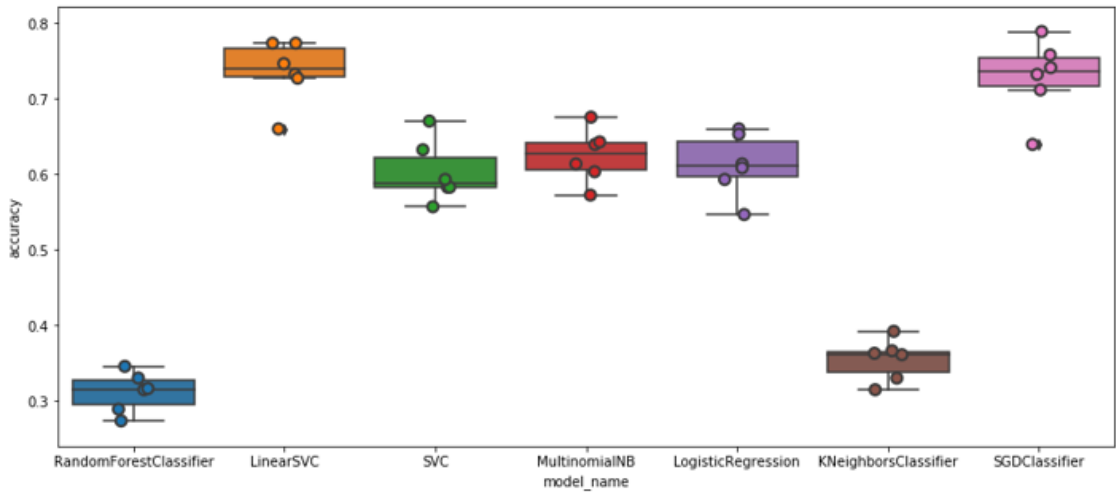
The data to analyze are comments from different hospital's clients in Spanish language from 2017 to 2019 obtained through a tablet terminal where the customers are asked how their experience at the Hospital was.
The Dataset contains the Name of the hospital, the comment and the topic of the comment.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

Vector models for this kind of tasks proven very efficient in these kind of task such as **LinearSVC** and **SGDClassifier**.
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.

Linear SVC proven the algorithm with most accuracy in comparison to other models:



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

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.

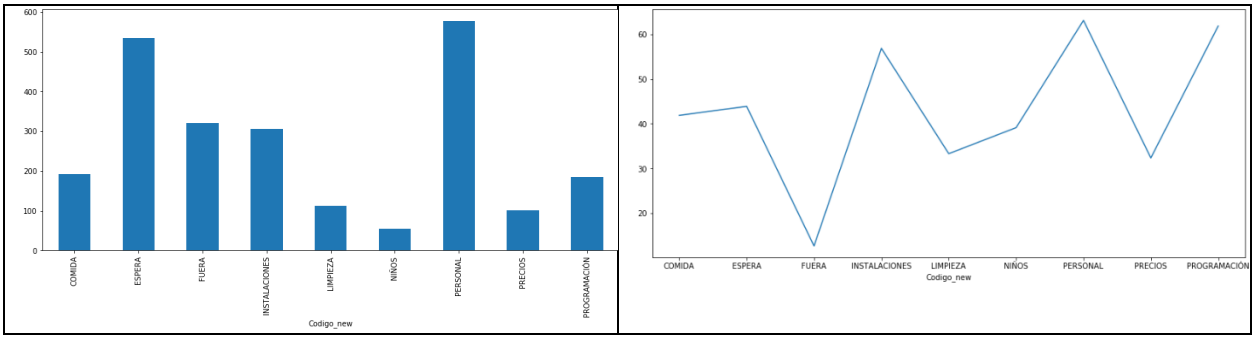
Finally, I created a length column based con word count and character count in case I consider to include this data to the model.

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.

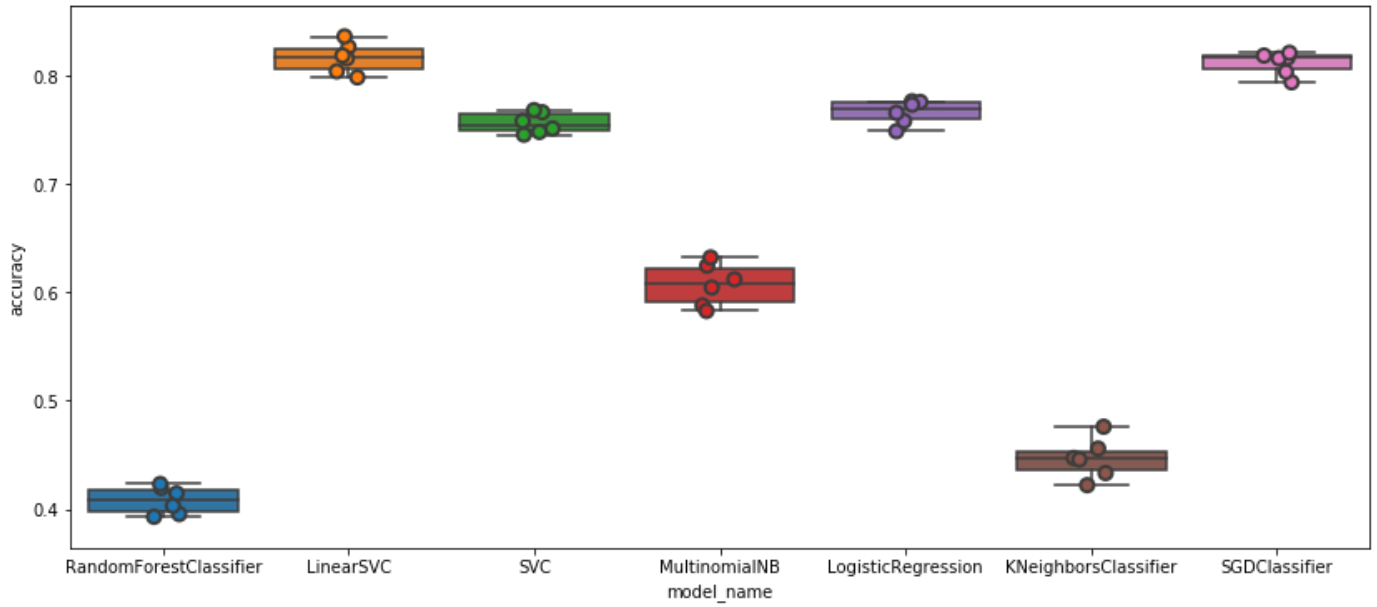
2. Once preprocessing is done I saved the processed data into a custom directory and proceed to do some data visualization to get some insight about the data.

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



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.

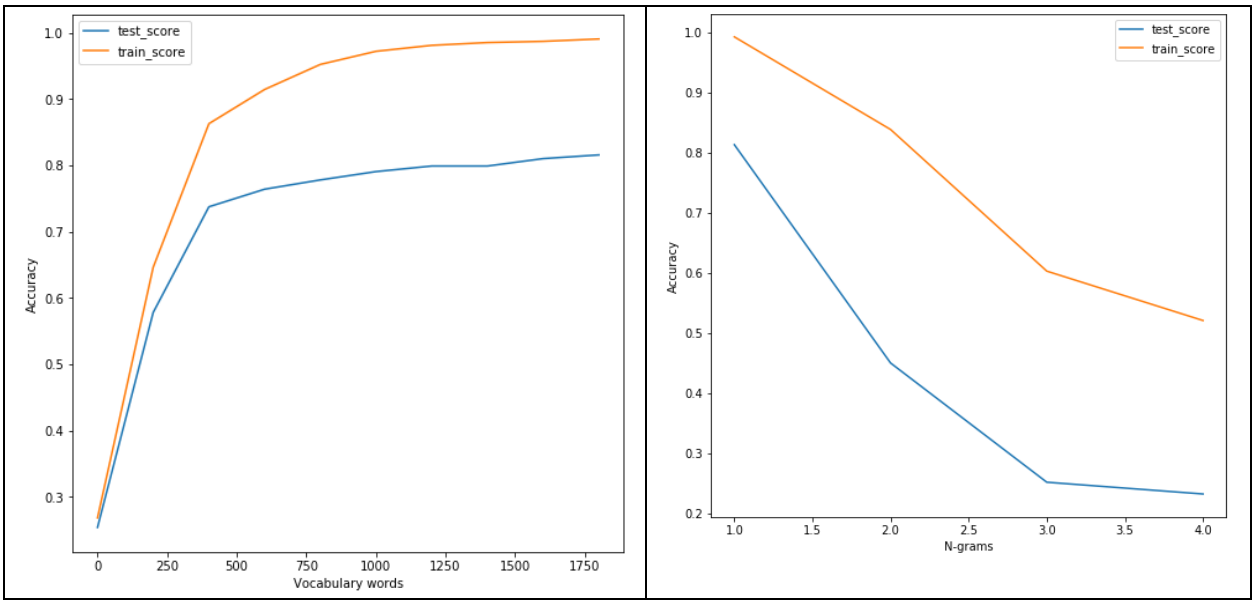
I decided to approach the second option as when exploring the data set I noticed that there are some infrequent words that carry a lot of importance to identify a comment as a specific category.

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:



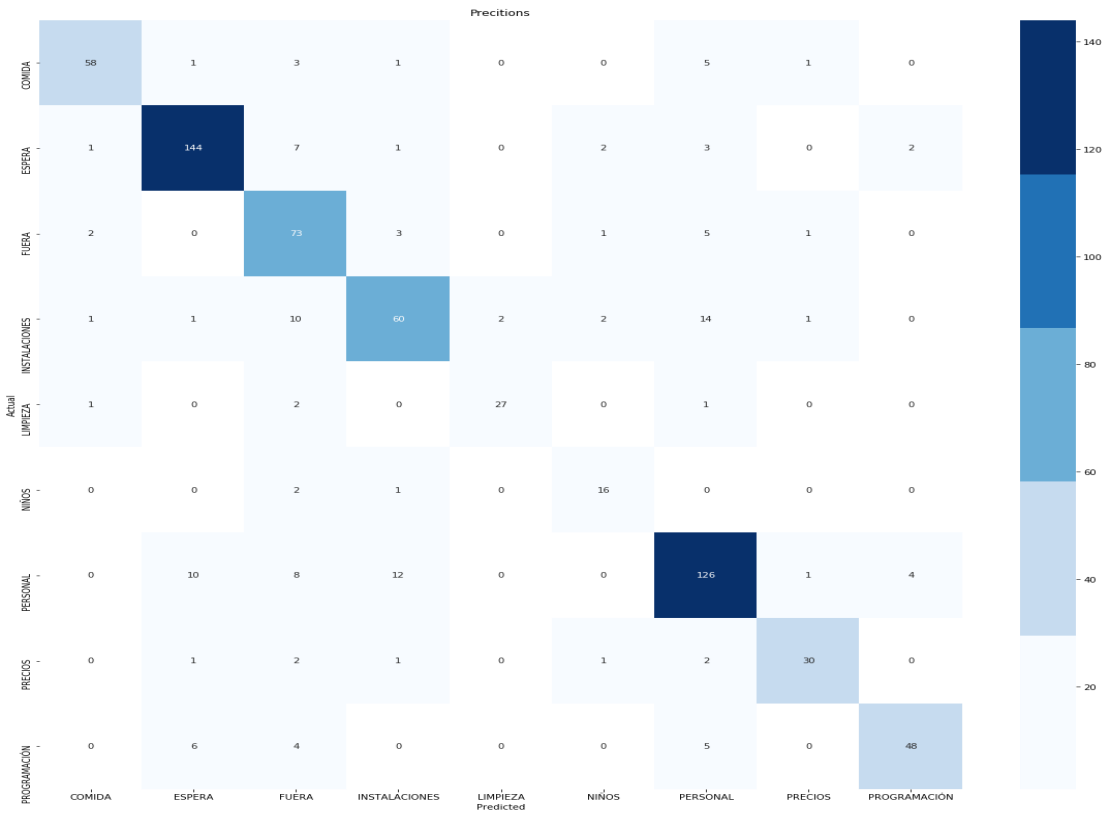
Metrics:

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

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:



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.

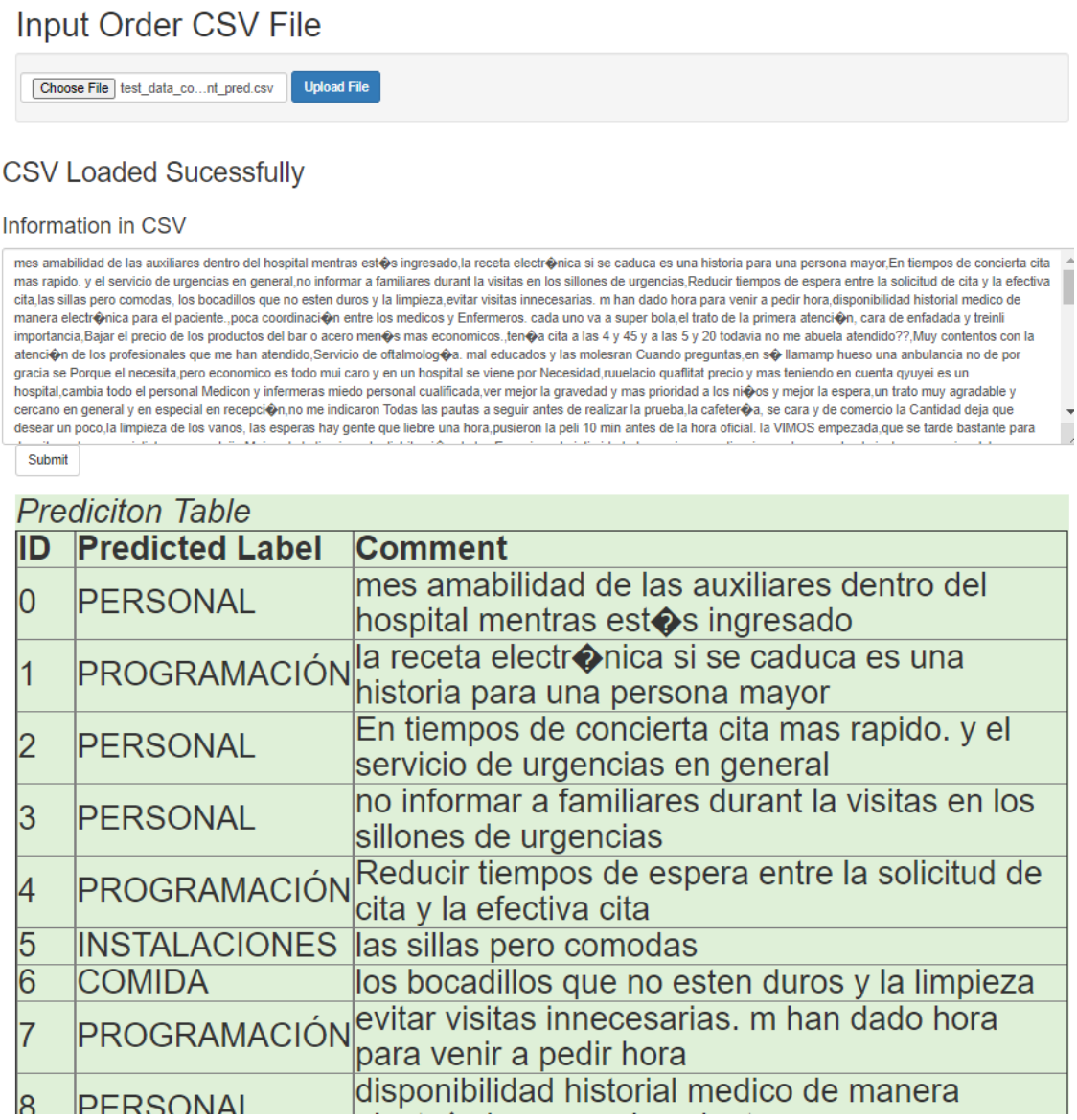
3.This string is sent to our lambda function that will use the function “comment_processing_csv” to preprocess the data in a format the model can read and give an output prediction.

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.



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.