# Objective

The main goal of the present project was to identify how a product is perceived in the market. This fall in the area of Sentimental Analysis, which is an important part of Natural Language Processing. This technology handles with the big challenge of make human language understandable to computers.

As complicated this might look, we chose a small and manageable objective: we wanted to create a model capable of classify comments on a product into positive and negative, in order to have a first insight in the consumers product perception. We thought that the best way to do this was to trough gathering social media comments in order to classify them into positive and negative, and then feed a machine learning classification model.

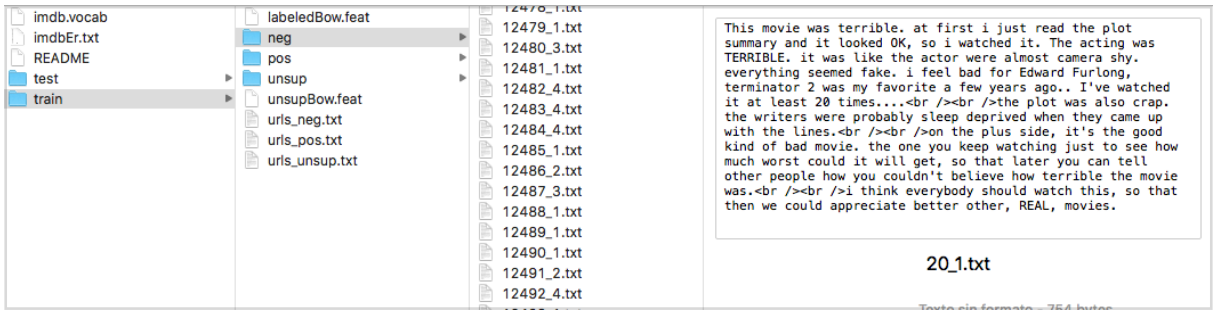
At first we tried to gather this data ourselves but soon it becomes evident that this was a major task out of our time constrains. So we decided to resolve a similar problem with existing data. Fortunately we found a well classified dataset of movie's comments posted on IMDB, who save us half of the work.

We thought that this was a good star, because movie industry is a multimillionaire business in which each project has a lot to lose if a production if fails to meet public interests and wishes. So it needs feedback of his production to meet public expectations. So the product we want to help to analyze were movies.

In this matter, Sentimental analysis is a particular useful machine learning method to help movie industry to meet public interests. On this respect we pretend to use Natural Language Processing model capable of identify the connotation of a movie comment, so in union with some other methods we can improve our understanding on public taste.

# Database Construction

The data we used was compose of 50,000 comments, posted in IMDB, who were gathered for the Annual Meeting of the Association for Computational Linguistics: Human Language Technologies( A. Maas et.al, 2011). Large Movie Review Dataset v1.0 This comments where composed by 30 comments maximum on one movie, and were already labeled as negative or positive.



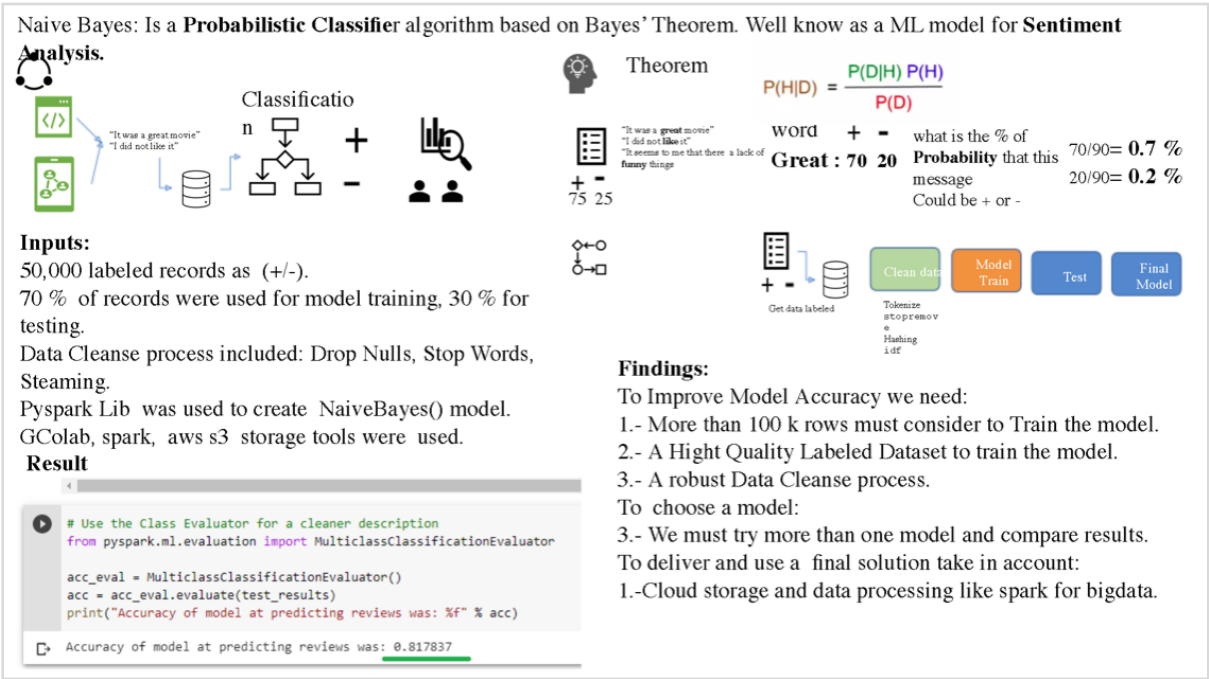

The first problem we faced was that the data was provided in a multiple ".txt" files, so the

first task was to iterate throw it and put them inside a unified data frame. This task was best made with python pandas. Our data were provided in two folders differentiating them in positive and negative, so we have to add the correct label, and then apply a One-Hot Encoding in order to transform labels into numbers. This was our result:

| | 0 | negative | positive |
|---|---|---|---|
| 0 | Story of a man who has unnatural feelings for ... | 1.0 | 0.0 |
| 1 | Airport '77 starts as a brand new luxury 747 p... | 1.0 | 0.0 |
| 2 | This film lacked something I couldn't put my f... | 1.0 | 0.0 |
| 3 | Sorry everyone,,, I know this is supposed to b... | 1.0 | 0.0 |
| 4 | When I was little my parents took me along to ... | 1.0 | 0.0 |

## Choosing a method

Determine if something is positive or negative is better manageable with a classifier model and binary data. At the end we chose a Support Vector Machine, because is the more straight forward classifier we know and was the default metaparameters provided by the sklearn library we finally use. Before reaching this point we have two options the Naive Bayes model from PySpark and using google Colab and Sklearn stochastic gradient descent (SGD) classifier. First we tried the Naive Bayes model which is explained in this image:




With this model we got an score of 81.56 accuracy as maximum. At the end we decided to change model, not because of the results, but because we found some limitation to deploy our model in a web app, so we decided to use sklearn instead, which luckily gave us better results as we will explain. But before use the sklearn model we need to meet

specific requirements of preprocessing.

## Preprocessing

The first preprocessing task was to remove all punctuation to facilitate the recognition on every word. This was a decision on our own, that came with a cost because certain punctuation as "!" or "?" express certain emotions that could be lost. But putting this into a balance it was preferable to lose little accuracy than to got a duplicated word because a point or a comma, that can affect the weight our model can give to some words. This task was easily completed using the `string` library.

| | text | label |
|---|---|---|
| 0 | Easily the worst movie I have ever seen in my ... | 0 |
| 1 | Ambushed is no ordinary action flick Its much ... | 0 |
| 2 | I loved this movie but then again I am a big C... | 1 |
| 3 | In 1933 Dick Powell and Ruby Keeler sang and d... | 1 |
| 4 | To make any film about the supposed end of the... | 0 |
| ... | ... | ... |
| 49995 | While its true that the movie is somewhat inte... | 0 |
| 49996 | From the upper shelf of great Classic Books co... | 1 |
| 49997 | Good ideashame about the actual movie Would of... | 0 |
| 49998 | An unusual film for an audience outside the US... | 1 |
| 49999 | I really enjoyed The 60s Not being of that gen... | 1 |

50000 rows × 2 columns

**Vectorize**

The second preprocessing task was something asked by sklearn model. We need to transform words into numbers via tokenization. To achieve this we had to vectorize our strings using `CountVectorizer` which transform our strings into a matrix of words count. For each word it find it create a column, and for each string it create a row. So we got a matrix who counts how many times a word appear in a string. If we would liked we could create more complex matrix not counting words but counting pair of words, we just had to specify `ngram_rang=(2,2)` but this would had create a bigger matrix and made a heavier process, with interferes with our propose to create a light web app. In this image we use an example sentence to show up how vectorize works.

Out[5]:

| | text |
|---|---|
| 0 | Hello everybody |
| 1 | my name is Jose Alvarez everybody |
| 2 | my age is 32 years and have been living in CDM... |

Out[28]:

| | 32 | age | alvarez | cdmx | everybody | hello | jose | living | years |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 2 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 2 |

Out[1]:

| | text | label |
|---|---|---|
| 0 | This movie has some things that are pretty ama... | 1 |
| 1 | Duchess and her three kittens are enjoying the... | 1 |
| 2 | The Class is a comedy series that portrays a b... | 0 |
| 3 | Latter days is the best gay movie of the homos... | 1 |
| 4 | There is part of one sequence where some water... | 0 |
| ... | | ... |
| 49995 | This movie succeeds at being one of the most u... | 0 |
| 49996 | There is a reason Chairman of the Board got a ... | 0 |
| 49997 | My Favorite part was when the credits started ... | 0 |
| 49998 | Jack Frost no kids its not the warm hearted fa... | 0 |
| 49999 | Has there ever been a movie more charming than... | 1 |

| | Token 1 | Token 2 | Token 3 | ... | ... | Token 108083 |
|---|---|---|---|---|---|---|
| 0 | X | X | X | ... | ... | X |
| 1 | X | X | X | ... | ... | X |
| 2 | X | X | X | ... | ... | X |
| .... | ... | ... | ... | ... | ... | ... |
| 49997 | X | X | X | ... | ... | X |
| 49998 | X | X | X | ... | ... | X |
| 49999 | X | X | X | ... | ... | X |

**Transform**

The next step was to assign weights to our words. This is managed by the library `TfidfTransformer`. This resource transform our count matrix into a tf-idf *Term frequency – Inverse document frequency*, a common weighting scheme in information retrieval. This scale the impact of more frequent tokens in the hole corpus of documents using this equation:

```
tf-idf_{t,d} = (1 +\log tf_{t,d}) \cdot \log \frac{N}{df_t}
```

This method define the weight of our words taking how many times a word appear in the hole corpus of sentences.

$$tf - idf_{t,d} = (1 + \log tf_{t,d}) \cdot \log \frac{N}{df_t}$$

Out[28]:

| | 32 | age | alvarez | cdmx | everybody | hello | jose | living | years |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 2 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 2 |

Out[30]:

| | 32 | age | alvarez | cdmx | everybody | hello | jose | living | years |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.508542 | 0.861037 | 0.000000 | 0.000000 | 0.000000 |
| 1 | 0.000000 | 0.000000 | 0.652491 | 0.000000 | 0.385372 | 0.000000 | 0.652491 | 0.000000 | 0.000000 |
| 2 | 0.593683 | 0.296841 | 0.000000 | 0.296841 | 0.175319 | 0.000000 | 0.000000 | 0.296841 | 0.593683 |

# Model Training

As we said before we choose to use the stochastic gradient descent classifier (SGD) in their default metaparameters for the sake of simplicity, for the same reason and because of lack of time we didn't used a Grid search, but maybe in future we can try this. The SGD Classifier can apply an SVC or a Linar Regression, the default model uses a SVC. At the end of the day this seemed to be the right decision as long as we got high scores with training and test data, 0.93136, and 0.899056 respectively, that were better than the 0.8156 we got with pyspark Naive Bayes model. The split we made in our data in order to train it were of 75 train, 25 test, as the SGD Classifier asked for.

# Results and Conclussion

Once we accomplished the training of our model we were able to pass it some foreign comments. We saved our model and add it to a python script who runs behind an static web app created with Flask (we tried Django but Flask was more easy to use). In there we

put a text box where the user can put their opinion and our model can tell if it is positive or negative.

When the text was clearly a bad or a good opinion we found great accuracy. But we found some complications when a comment were ambiguous, or had bad and good words combined. In this cases our model tend to fail, so this give us feedback, about maybe create a new category called neutral and train our model again.
Another problem we notice was that the stop words they have actually an emotional value, particularly thinking in words like "not", and our model tend to ignore this. We were worried about some false positives, but this seems not to affect to much as, we already tell, clear messages were well predicted.
Considering the little amount of time we had, we felt satisfied with the results we got. We think we have a good and working start point, to perfect our model. Maybe adding more data, adding more stop words for obvious irrelevant words as "movie".

## References

==========

Maas, A., Daly, R., Pham, P., Huang, D., Ng, A. and Potts, C. (2011). Learning Word Vectors for Sentiment Analysis: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. [online] Portland, Oregon, USA: Association for Computational Linguistics, pp.142–150. Available at: http://www.aclweb.org/anthology/P11-1015.

Building a Sentiment Classifier using Scikit-Learn | by Dorian Lazar | Towards Data Science