***Movie Reviews Sentiment Analysis***

[**GitHub repository**](https://github.com/javisin22/CSC461projectSinPelegrin)**!!**

# Introduction

After some brainstorming, when we decided to follow the idea of Sentiment Analysis in the context of Movie Reviews, we searched on kaggle for existing models to inspire ourselves. This way, we ended up deciding to use one of them and tweak it so that we improve the work already done using techniques viewed in class and some other techniques that we saw in our home city University from Spain.

Here’s a [link](#_noq7wkygmrn6) to the kaggle model.

# Problem definition

For this project we are going to build a model whose goal is to be able to distinguish between positive and negative reviews given a movie review in text format.

To do so, we are going to use **Sentiment Analysis**, a method of processing and analyzing text to be able to extract its emotional tone and be able to determine whether the text, or in this case movie review, is positive or negative.

Together with this strategy we are going to try different model architectures to see which one gives the best results, mainly [Decision Tree](#_noq7wkygmrn6), [Naive Bayes](#_noq7wkygmrn6) and [Logistic Regression](#_noq7wkygmrn6).

After trying those Supervised Learning Classification models, we have decided to include the use of [Neural Networks](#_hsbmslevbg18) for this last part of the Final Project.

Text preprocessing will also be an important part of this project, we will use **‘Bag of Words’** (BOW) since we need to turn the movie reviews in text format into numerical vectors that our models can utilize and because with it we can account for the frequency of words and not their order which is not as relevant for our purposes.

# Data definition

The kaggle model comes with its respective dataset. It’s based on a tensor consisting of 50000 rows by 2 columns. The features (columns) are the **review** itself, and the **sentiment** (positive || negative).

The dataset is based on a set of IMDB reviews. The *figure 1* shows a snippet of the data.

:

***Figure 1: Dataset snippet***

# Methods definition

In terms of the methods to be used, we’re basing ourselves on **Sentiment Analysis** primarily, as mentioned before. As mentioned before, we’ve used three different models: **Decision Trees**, **Naive Bayes** and **Logistic Regression**. In the former kaggle model, it was also used SVM but we won’t as we haven’t seen it in class.

In addition, we have created a **Neural Network** and tuned it manually to try and improve the results obtained by the previous *Supervised Learning Classification* models.

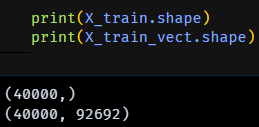
Unlike the original kaggle model, where only the best default configuration model was tuned, we’ve created and tuned every one of the models from scratch so that we try to get the better performing model and we see the differences in the tuning for every one of them.

We’ll also have to use methods for data preprocessing like [**TF-IDF**](#_noq7wkygmrn6) (*Term Frequency-Inverse Document Frequency*), in order to **convert the text into numerical data**, this is the way in which we are going to represent the ‘*bag of words*’. TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It helps in extracting meaningful features from the text data.

In addition, TF-IDF **normalizes the word counts** **by the frequency of the word** in the entire corpus, which helps in reducing the impact of common words that are less informative.

By converting text data into TF-IDF features, the model can better understand the importance of different words and phrases, leading to improved performance in tasks like classification.

Here’s an example of the size for both regular training data, and vectorized training data:



Mention that we’re going to store each individual result for the performance of every model in a [Panda’s Dataframe](#_noq7wkygmrn6) so that we can compare all the models in the end of the notebook.

# Results

Now we’ll provide the achieved results for all the models. Even though we are gathering all the plots and values from the tests, we recommend reading the notebook as there are more insights about the interpretation of the results

After testing and tuning all the different models, we’ve achieved the following results for both **accuracy** and **f1\_score**:

## Decision Tree:

*Default model:*



*Tuned model:*

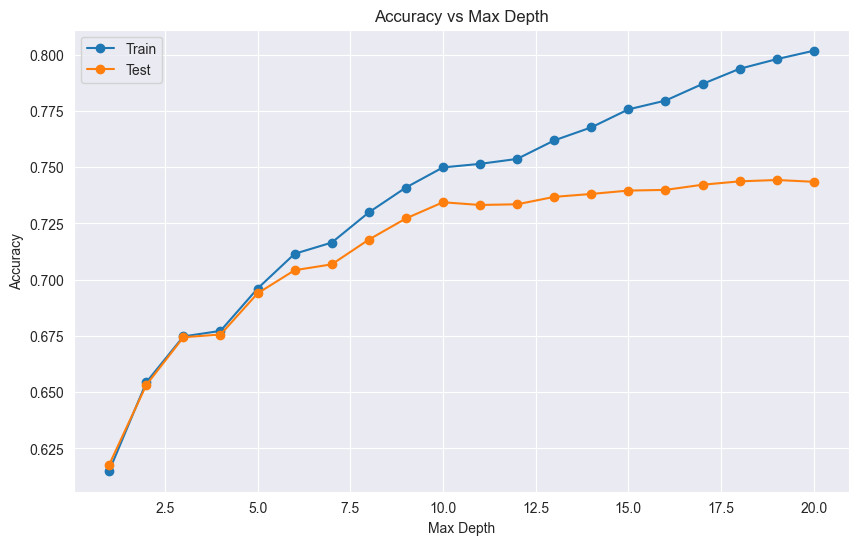


The best parameters for the model are:

* 'max\_depth': 20
* 'min\_samples\_leaf': 10
* 'min\_samples\_split': 2

As we can see, the default model suffers from **overfitting**, reaching the maximum accuracy. This means that the model has learned the patterns from the training data and struggles to generalize so the model has to be tuned to prevent it.

The *figure 2* represents the evolution of the model.



***Figure 2: Decision Tree Accuracy vs Max Depth evolution***

## 

## Naive Bayes:

*Default model:*



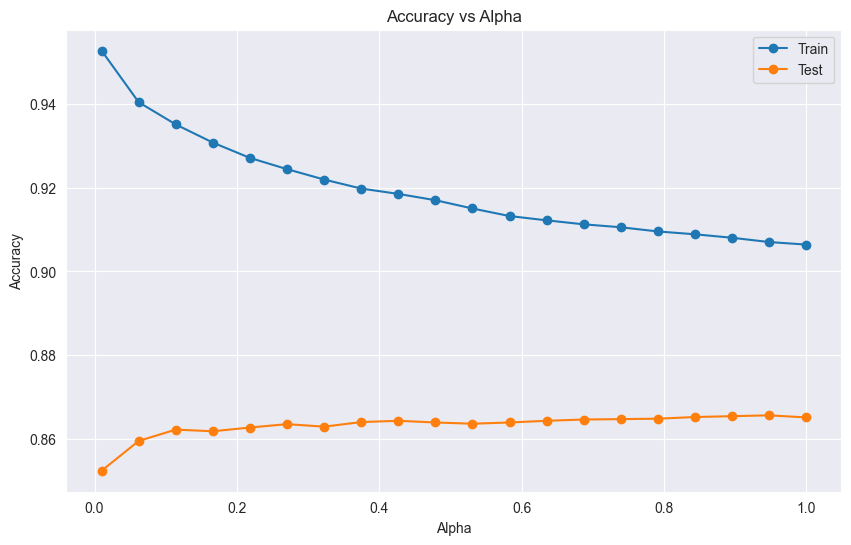
*Tuned model:*



The best parameter for the model is: 'alpha': 0.4789

If we compare the model without and with tuning, it’s visible that even after tuning it, the base one is better in terms of new unseen data as the test metrics are better (by little).

The *figure 3* represents the evolution of the model.



***Figure 3: Naive Bayes Accuracy vs Alpha evolution***

## Logistic Regression:

*Default model:*



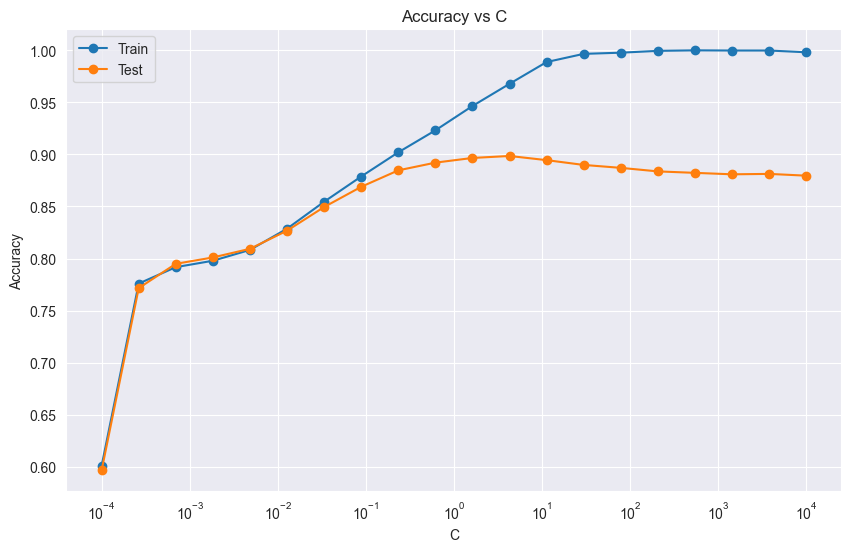
*Tuned model:*



The best parameter for the model is: 'C': 4.2813

In this case, we can see that the tuned model improves both in terms of train and test data.

The *figure 4* represents the evolution of the model.



***Figure 4: Logistic Regression Accuracy vs C evolution***

## Neural Networks:

*Tuned model:*



The structure of the Neural Network is the following one:

fc1 = nn.Linear(input\_dim, 128)

fc2 = nn.Linear(128, 64)

fc3 = nn.Linear(64, 1)

relu = nn.ReLU()

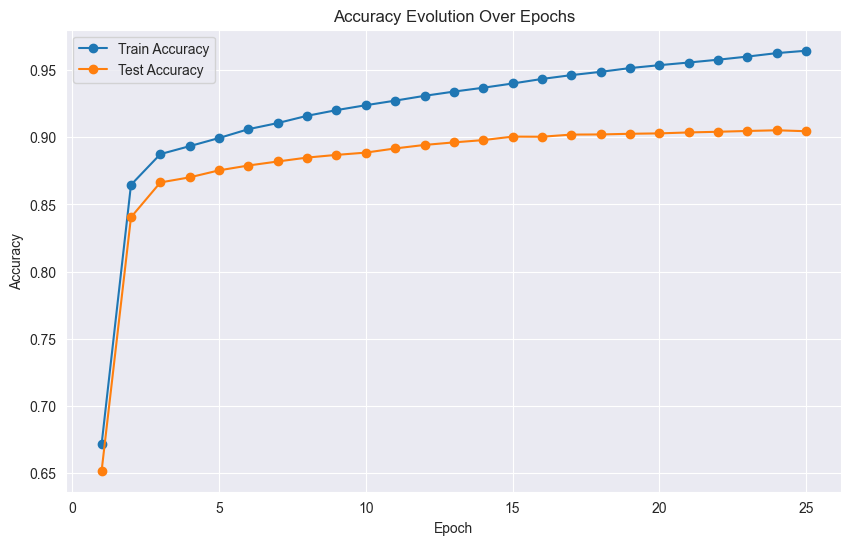
sigmoid = nn.Sigmoid()

At first we expected better results, but after trying multiple different neural network structures, learning rates and number of epochs, this was the one that gave us the best results.

The performance of this model might not be that much better than that of our logistic regression model because of the simplicity of our task, which just consists in dividing reviews in ‘positive’ or ‘negative’.

With that in mind, we still were able to find a configuration that outperformed the rest of our models.

The *figure 5* represents the evolution of the model.



***Figure 5: Neural Network Accuracy Over Epochs evolution***

# References

* [Original Kaggle model](https://www.kaggle.com/code/shubhamptrivedi/sentiment-analysis-on-imdb-movie-reviews/notebook)
* [Pandas Dataframe](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)
* [Sci-kit Learn Metrics](https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics)
  + [Accuracy](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score)
  + [F1 score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score)
* [TF-IDF Vectorizer](https://scikit-learn.org/1.5/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)
* [Decision Tree](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier)
* [Naive Bayes](https://scikit-learn.org/1.5/modules/naive_bayes.html)
* [Logistic Regression](https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.LogisticRegression.html)