**Artificial Intelligence  
IMDB Sentiment Analysis**

**David Ciocoiu  
40323308**

**Introduction**

Sentiment analysis is a natural language processing technique used to interpret and classify emotions in subjective data, often performed on textual data to detect sentiment in emails, survey responses, social media data, and beyond. (MonkeyLearn)  
This helps companies gaining a better insight into their customers and acting accordingly (i.e. to improve efficiency and/or increase revenue).  
This paper will propose a critical discussion on a sentiment analysis experiment, using the [IMDB](https://keras.io/api/datasets/imdb/) Keras dataset (created by researchers at Stanford University achieving 88.89% accuracy), containing 50000 movie reviews in English, labelled by sentiment, either positive or negative.

**Feature Extraction**

The model chosen is a decision tree, adopting the [DecisionTreeClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html) provided by scikit-learn and using entropy as a criterion; the algorithm will generate a tree like structure splitting each attribute in the dataset according to its information gain in order to achieve entropy homogeneity.  
Many iteration of the experiment have been performed, each time adjusting text-processing values and other parameters aiming for the best overall accuracy, with the best result being 74% accuracy on a 10 fold cross validation using (unigrams) vectors obtained via the Bag of Words method.

First the dataset has been retrieved from Keras, concatenating the training and testing sets and decoding the sentences back to their textual form in order to strip them from any impurities.  
The data is evenly balanced, meaning it’s equally divided into 25000 positive reviews and 25000 negative reviews, therefore there is no need to split the dataset.  
Unwanted features have been eliminated, such as meaningless characters and strings (i.e. “br”) and stop words which would otherwise introduce noise in the data. Text pre-processing not only can had a positive impact on accuracy but decreased the vocabulary size with positive impact on algorithmic speed.

Subsequently a series of experiments helped understanding whether to apply lemmatization, which aims to to standardize each of the inflectional alternates and derivationally related forms to the base form.   
As a rule if the model’s performance does not increase lemmatization should not be applied (ModelOp, A. Schumacher, 2019), especially as this adds a noticeable overhead to the data preparation stage with drastic effects on machines with limited computational power.   
Lemmatization has been performed with different parameters and successful results have been observed solely when lemmatizing adverbs with an increase in accuracy of 1%.

Then sentences have been vectorized using CountVectorizer from scikit-learn.  
Although n-grams bigger than 1 can provide deeper insight into the data by retaining some of the context, unigrams are the preferred approach in this case as they yield a higher accuracy.  
Here, only words occurring in a maximum of 70% of the sentences have been retained, filtering the result by the 1500 most occurring words, this is because words that occur in almost every document are generally not suitable for classification (StackAbuse, U. Malik, 2019).   
The vocabulary obtained has the 1500 topmost relevant features, used to encode vectors as bag of words.

**Model training and evaluation**

Lastly the model has been ran with the data obtained from the above feature extraction process.  
K folds validation has been preferred over a one-run method, since it averages the results of running the experiment on different dataset bins providing a better insight. For the model at hand optimality has been found by using 9 bins, which means the algorithm will divide the dataset in nine parts, one for the test set and eight for the training set, running different iterations for each part (test set).  
Overfitting has been reduced by pruning the tree at depth 13, yielding a final accuracy result of 74% and an f1 score of 76%.

An iteration has been ran in order to visualize a confusion matrix, setting the test set to 20% of the whole data, obtaining the results in Fig.1.  
The confusion matrix shows that the model is better at predicting positive reviews with an harmonic score of 76% as opposed to the negative one of 71%.   
F1 score is a good measure as it seeks for a balance between precision and recall, providing information about the misclassified instances (as opposed to accuracy). In the proposed example we can see that the return results indicate the model has more false positives (1744) than false negatives (879), meaning there are more negative reviews wrongly predicted as positive than positive reviews falsely predicted as negatives. To confirm it precision and recall have been calculated obtaining a higher recall (0.82) than precision (0.70), this indeed indicates that there are less false negatives than false positives.

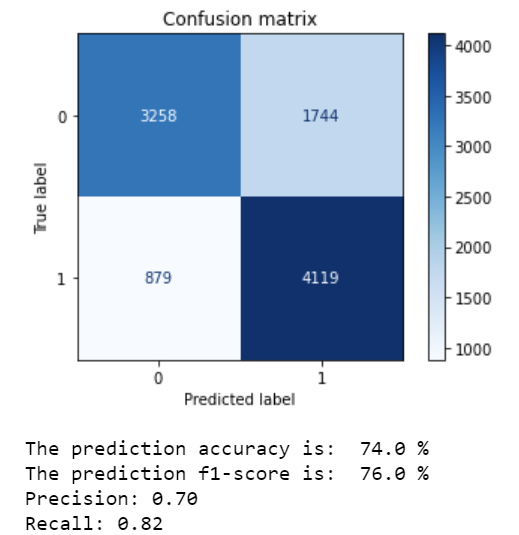
The decision tree visualization has been omitted as it is quite wide for this document, this can be visualized by running the code provided, although a pruned version has been included in Fig.2 to provide a general idea.  
A quick inspections of the words chosen as topmost nodes shows that features such as “bad”, “worst”, “wonderful” and “best” contain high entropy, hence they are not enough to tell whether a reviews is positive and negative and they must be supported by other sequences of nodes. It is interesting to notice that similar sequences can lead to different outcomes, for instance in Fig.2   
It is interesting to notice along the designated paths outlined by red dots that the sequence “bad, worst, wonderful, first” leads to two different paths, one with entropy 0 (a leaf node) labelled as positive, and one with entropy 0.7 (high entropy), labelled as negative, from which in the non-pruned version other nodes arise.

Considering decision tree are quite unstable and sensitive to noise and overfitting, the obtained model is relatively good, especially when considering that better results range around 80% accuracy by using much more powerful techniques as CNNs.

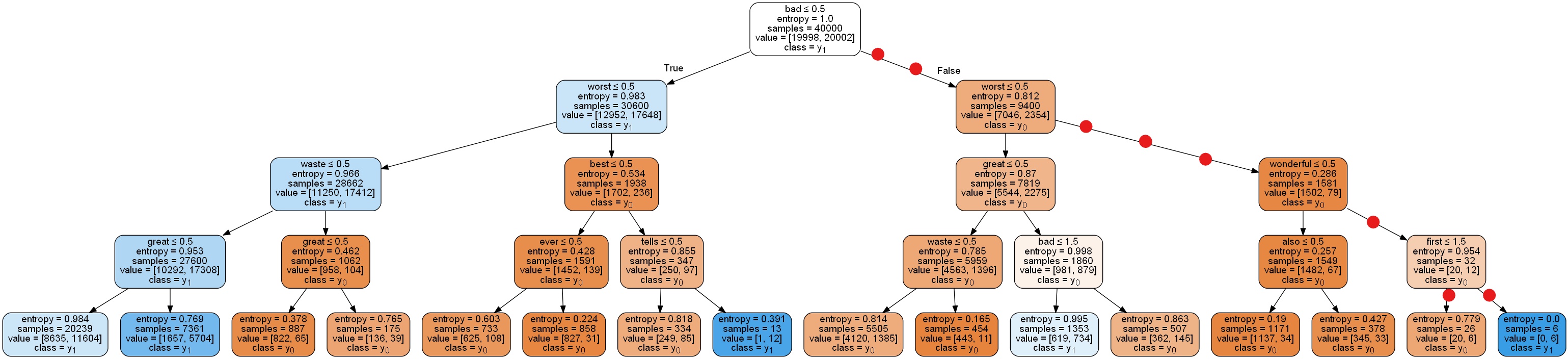
**References**

* Sentiment Analysis: A definitive guide, MonkeyLean. Available at: <https://monkeylearn.com/sentiment-analysis/>
* How to build a NN with Keras using the IDM dataset, Niklas Donges, 2020, BuiltIn. Available at: <https://builtin.com/data-science/how-build-neural-network-keras>
* When (not) to Lemmatize or Remove Stop Words in Text Preocessing, Alex Schumacher, 2019, ModelOp. Available At: <https://opendatagroup.github.io/data%20science/2019/03/21/preprocessing-text.html#:~:text=The%20general%20rule%20for%20whether,improve%20performance%2C%20do%20not%20lemmatize.&text=For%20example%2C%20a%20popular%20sentiment,not%20be%20stemmed%20or%20lemmatized>.
* Precision-Recall, Scikit. Available at: <https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html>

**Appendix**



**Fig. 1** Note that Python displays the confusion matrix as follows: 1st row (TN FP), 2nd row (FN, TP)

****

**Fig.2** Pruned decision tree. Original tree can be obained by running the code provided with the document.