Artificial Intelligence Report (SET09122)

Valeri Vladimirov

School of Computing, Edinburgh Napier University, Edinburgh

40399682@napier.live.ac.uk

1. Introduction

This coursework is about the development of a sentiment classification model based on decision trees for the IMDB dataset provided by the keras library. For the programming component a python application has been created, which returns the unigrams and bigrams for the dataset and shows the 20 most used unigrams and bigrams, implements bag of words, creates and displays a decision tree and applies the appropriate evaluation metrics to the decision tree. This report will focus on the non-programming aspect, which is justifying why bag of words has been chosen as a text representation technique and explaining which evaluation metrics have been used and why.

2. Text representation technique

In the lectures three text representation techniques were introduced:

- One-hot encoding
- Count vectorisation
- Word embeddings

From these methods it was decided that count vectorisation, also known as bag of words, would be the most suitable for this task.

Bag of words [1] is a model for extracting features from documents that can then be used in machine learning algorithms. It collects words from the text and represents their frequency. Count vectorisation is called "bag-of-words" because there is no order for the words, it only cares about whether the word appears in the document and how often, not where exactly. The model involves a vocabulary of the words and the number of their occurrence.

It was preferred over the other two methods of text processing after considering the following:

- One-hot encoding also involves creating a vocabulary as in bag of words, however it doesn't count the number of occurrences of the word. Instead it uses 0's and 1's to express a Boolean value whether the word exists in the sentence. Count vectorisation was preferred because the occurrence count will be a relevant feature when creating the entropy frequency tables for the decision tree [4] and it will also produce better results.
- Word embeddings is a method which represents a word as a numeric vector. It determines relationships with other words, meaning that words similar to each

other will be used together more often. Bag of words was preferred because word embeddings represents each word as its own vector, whereas bag of words represents all the words and their count in one huge vector which will be more useful for the current task and for the creation of a decision tree.

Overall bag of words was found to be the most appropriate technique for the current task. Although it has some drawbacks such as semantic meaning and huge vectors, which could result a lot of computation and time, and sparse vectors, these factors will not affect the performance of the decision tree (see Fig. 7.). The technique was chosen because:

- Simple to understand and easy to implement.
- The vectors for all the reviews will all be equal because they are all equal to the vocabulary size. This will be of use when creating a DataFrame object and using it to create the decision tree in the program.
- The count functionality was helpful for the Entropy criterion when creating a decision tree and led to better results in the evaluation metrics.
- It doesn’t have unnecessary functionalities such as relationship between words, which are irrelevant for the current task.

3. Evaluation Metrics

For the evaluation metrics [3] the bag of words was split into training (80%) and test (20%) sets.

3.1 Confusion Matrix

A confusion matrix [2] (see Fig. 1.) was used to compare the predicted values with the actual values. This metric was chosen because it is a great way of describing the performance of the model. The confusion matrix is also used to visualise other analytics such as accuracy, precision, recall. Figure 1 shows:

- **True Positives (180)** – how many times the model correctly predicted the positive class.
- **True Negatives (157) –** how many times the model correctly predicted the negative class.
- **False Positives (84)** – how many times the model mistakenly predicted the positive class, also known as Type I error.
- **False Negatives (79)** – how many times the model mistakenly predicted the negative class, also known as Type II error.

3.2 Accuracy

Accuracy (see Fig. 2.) is one of the simplest metrics and has been used to measure the percentage of correctly predicted instances for the model. Formula:

Accuracy = (True Positives + True Negatives) / Total Number

3.3 Recall

Recall (see Fig. 3.) has been used to show the percentage of positive instances from entire set of positive instances. It is also known as the Sensitivity of the model. Formula:

Recall = True Positives / (True Positives + False Negatives)

3.4 Precision

Precision (see Fig. 4.) has been used to show the “exactness” of the model. This metric shows the percentage of positive instances actually being predicted as positive. Formula:

Precision = True Positives / (True Positives + False Positives)

3.5 F1 Score

F1 Score (see Fig. 5.) has been used to show the balance between precision and recall. This metric is known as the harmonic mean of the two other metrics. Formula:

*F1 Score = (2 * Precision * Recall) / (Precision + Recall)*

3.6 Specificity

Specificity (see Fig. 6.), although it is not a very common evaluation metric it has been used to show the percentage of the actual negative instances that are correctly identified. It is also known as the “True Negative Rate”. Formula:

Specificity = True Negative / (True Negative + False Positives)

4. Conclusion

To sum up, the results achieved from the evaluation were satisfactory, however they were not as good as anticipated. The confusion matrix showed a relatively high number of Type I and Type II errors (nearly one third of the 500 predicted), which lowered the results on the other metrics. Accuracy showed that 67.40% of the instances were classified correctly, Precision (68%) and Recall (69%) also showed above average outcome when examining the positive instances, which are overall decent results. Probably higher results could have been achieved if a bigger portion of the dataset was used.

5. Appendix

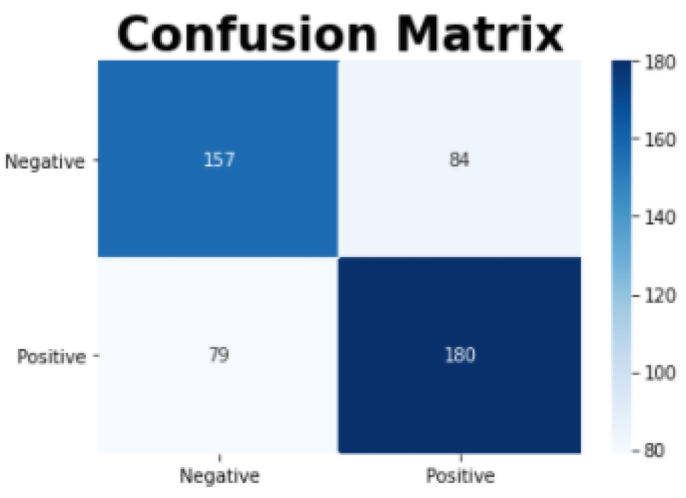
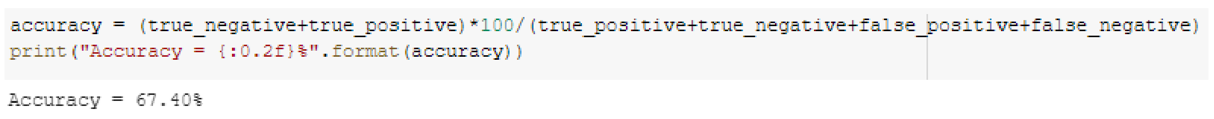


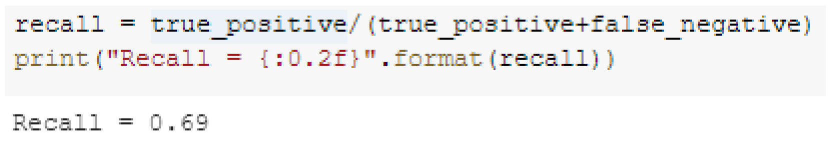
Figure 1. Confusion matrix for the model.



```
accuracy = (true_negative+true_positive)*100/(true_positive+true_negative+false_positive+false_negative)
print("Accuracy = {:0.2f}%".format(accuracy))

Accuracy = 67.40%
```

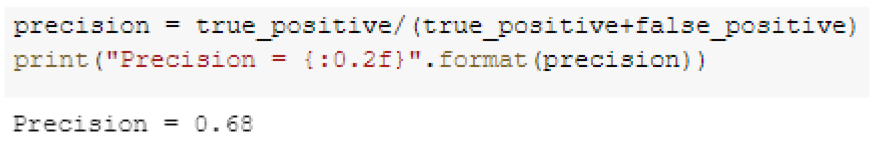
Figure 2. Accuracy for the model.



```
recall = true_positive/(true_positive+false_negative)
print("Recall = {:0.2f}".format(recall))

Recall = 0.69
```

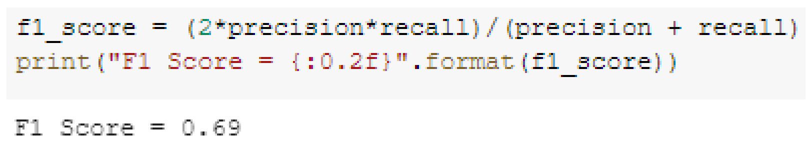
Figure 3. Recall for the model.



```
precision = true_positive/(true_positive+false_positive)
print("Precision = {:0.2f}".format(precision))

Precision = 0.68
```

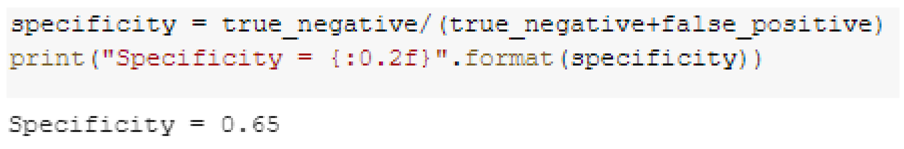
Figure 4. Precision for the model.



```
f1_score = (2*precision*recall)/(precision + recall)
print("F1 Score = {:0.2f}".format(f1_score))

F1 Score = 0.69
```

Figure 5. F1 Score for the model.



```
specificity = true_negative/(true_negative+false_positive)
print("Specificity = {:0.2f}".format(specificity))

Specificity = 0.65
```

Figure 6. Specificity for the model.

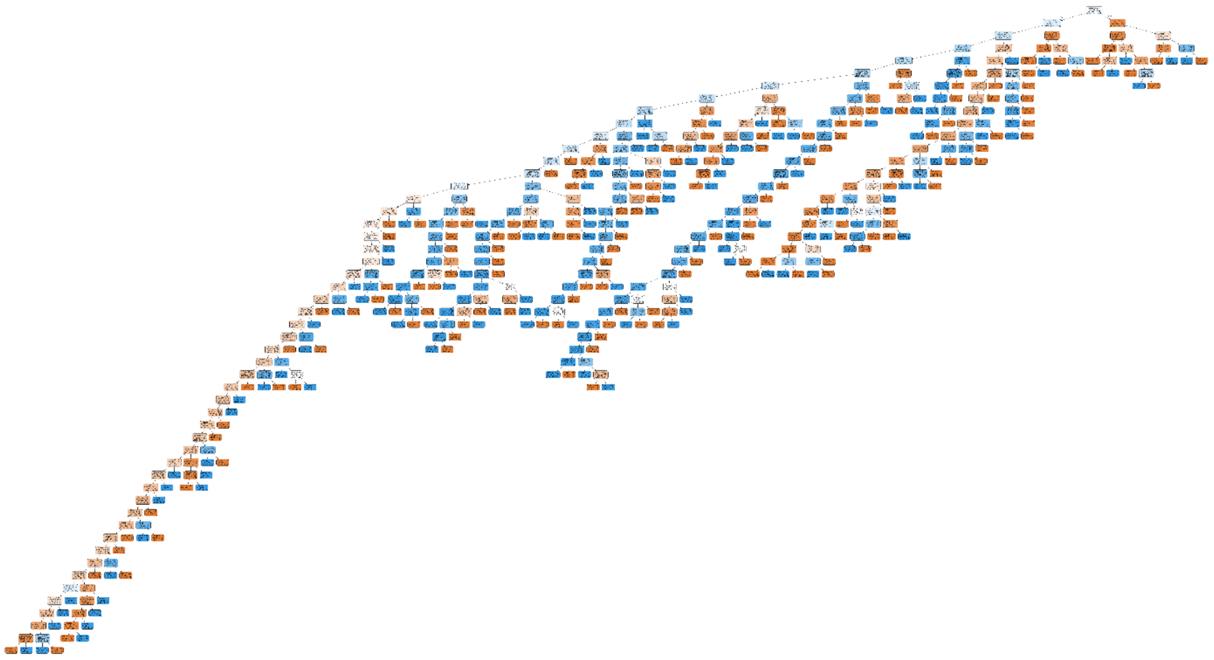


Figure 7. Decision tree

6. References

[1] A Gentle Introduction to the Bag-of-Words Model: https://machinelearningmastery.com/gentle-introduction-bag-words-model/

[2] Confused About The Confusion Matrix? Learn All About It: https://kambria.io/blog/confused-about-the-confusion-matrix-learn-all-about-it/

[3] Evaluation Metrics: https://www.python-course.eu/metrics.php

[4] Decision Trees in Python with Scikit-Learn: https://stackabuse.com/decision-trees-in-python-with-scikit-learn/

[5] Simple guide to confusion matrix terminology: https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/