Sentiment Analysis of Movie Reviews

Abdurrahmaan Desai

# Introduction

This report details the process of creating models to classify the sentiment of a movie review. The aim of the report is to compare how effective different models are in multiple areas including accuracy as well as time taken to produce models. The models that were considered for this analysis were Multinomial Naive Bayes, Decision Tree, Support Vector Machine, Logistic Regression and Multi-layer Perceptron. The Support Vector Machine model performed the best with an accuracy of 89.4%, edging out the Logistic Regression and Multi-layer Perceptron models but also took the longest: over 3 hours. The Multinomial Naive Bayes model was the fastest but with only an 85% accuracy and the Decision Tree model performed the worst with an accuracy of 72%.

# Background Reading

Perhaps the most appropriate paper to refer to for this task is the one written by the original curators of the ‘Large Movie Dataset’. The main focus of the results of this paper was to understand how different features affect the outcome of the model. The model that is used is a custom model created by the authors that uses unsupervised learning to derive semantic understanding and uses supervised learning to understand the sentiment of the review (Maas, et al., 2011). Whilst this report only deals with supervised learning, it is still useful to understand the results that the authors found using their own dataset. The authors used a linear SVM model and received a maximum accuracy of 88.89%. Using a smaller dataset of 2000 movies, Baid et al reported that the Naives Bayes algorithm performed with an accuracy of 81.4%, the K-Nearest Neighbour with a 55.30% accuracy and a Random Forest model with a 78.65% accuracy (Baid, Gupta, & Chaplot, 2017).

This report will attempt to expand on the report by Baid et al. and will test more models with a much larger dataset.

# Data

The dataset in question is the ‘IMDB Dataset of 50k Movie Reviews’ taken from Kaggle, also known as the ‘Large Movie Review Dataset’. This dataset comprises of 50,000 movie reviews taken from IMDB with an associated sentiment of positive or negative for each. The labels for each movie were generated based off the review score with a score of 7 or more out of 10 labelled as positive and a score of 4 or less out of 10 labelled as negative (Maas, et al., 2011).

Figure 1 in the supplementary material shows a subsection of 10 reviews in the dataset as well as the sentiment that is associated with it. Immediately, a couple problems can be noted. The second review contains html tags (<br />). Additionally, each review contains additional punctuation and formatting that is not necessary when creating a model (apostrophes, speech marks, commas and full stops). Thus, it is important to process the data to ensure that we can make the model as accurate as possible by only providing the data that is necessary to predict the sentiment and by removing the noise. More information about how this was accomplished will be detailed in the Model Development section.

# Model Development

The first thing to do when building a model is to clean the data. As discussed in the previous section, a couple of problems were identified which need to be solved.

The first problem that was identified was the inclusion of html tags which can be solved by using the code shown in figure 2. This line of code uses a regular expression to match html tags and replaces it with a white space (further explanation is given in the supplementary material).

The second problem can also be solved using a similar line of code, shown in Figure 3. This line of code uses a regular expression to match everything that is **not** a letter of the alphabet and replaces it with a white space (further explanation is given in the supplementary material).

Figures 4 and 5 in the supplementary material show the difference in a particular review after the two lines of code have been run. It is clear that all punctuation, html tags and extra content that are not useful for ascertaining sentiment has been removed.

With these two problems solved, the next step is to remove unnecessary parts of the review. The first technique is to apply stemming to each word in the review. Stemming is a technique used to standardise words by removing suffixes (Yucebas & Tintin, 2021). This results in the model using a singular repeated word to judge a single sentiment for each instead of using multiple words and judging a different sentiment on a word-by-word basis thus allowing for an accuracy increase. An example of this is ‘likes’, ‘liked’ and ‘likely’ which can all be stemmed to ‘like’, thus providing a singular word to base the sentiment off (Lang, 2022).

The second technique is to remove the stop words from each of the reviews. A stop word is a word that is commonly used in the English language that does not necessarily have any impact on the sentence in terms of sentiment (in this sentence, ‘a’, ‘is’, ‘that’ and ‘on’ may be considered stop words). Interestingly, negation words (such as not) are considered to be stop words in the NLTK dictionary and as such will be removed. This may cause unintended effects as a sentence such as ‘I did not like this movie’ will have the negation removed and therefore cause the sentiment to be positive when in reality, it should be negative. Model accuracy involving the removal of such stop words and without removal will be discussed in the next section.

Figures 6 and 7 in the supplementary material show the list of words before and after the above two techniques were used. It is clear that removing the stop words vastly shortens the list of words that will be inputted into the model. Additionally, words such as ‘unassuming’ and ‘performed’ have been shorted to ‘unassum’ and ‘perform’.

The next step is to vectorise the data. This is done using the TfidfVectorizer function by sklearn. TF-IDF stands for Term Frequency Inverse Document Frequency. As machine learning algorithms use a set of numbers to train models, this function converts the text into numbers. However, it also has another important feature which is to lower the impact of frequently occurring words. This is important as words that occur multiple times in a sentence can sometimes cloud over more important yet less frequently used words (Sckikit-learn, n.d.).

The final step of model development is to create the models. As the investigation was to discover how different models perform, each model was created in its own function, an example of which is shown in figure 8 in the supplementary material.

Figure 9 show a custom function called one\_hot\_encoder which uses sklearn’s label encoder to ensure that the sentiment of each review is in a binary representation to enable the model to understand the categorical representation.

# Model Evaluation

As this paper discusses 5 models, each model will be given a short description and then the results shown in a tabular format. Additional details of the ablation study will follow after the table.

Multinomial Naive Bayes – The multinomial Naive Bayes variant from the family of Naive Bayes models. Naive Bayes assumes that a specific feature (or word in this case) is independent and unrelated to any other feature/word (Ray, 2023).

Decision Tree Classifier – The decision tree classifier is a rule-based classifier that, in simple terms, asks questions based on the value of a feature and then splits into new nodes based on the answer. The splitting of the nodes gives a tree like structure and at each node, the model considers the available data and then decides which node to go to next. This process is repeated until the final outcome is reached.

Support Vector Machine (SVM) – An SVM uses the idea of a hyperplane to divide the dataset into classes, in this case positive and negative. The hyperplane does not have to be a straight line, it can curve to allow for a greater degree of accuracy in order to maximise the difference between two classes. Predictions are then made by measuring the distance between new data points and the hyperplane (Stecanella, 2017).

Logistic Regression – Logistic regression creates a linear model with the goal of predicting a binary outcome. The relationship between independent and dependent variables are then tuned using coefficients to allow the best prediction of the label class.

Multi-layer Perceptron – A Multi-layer Perceptron is an artificial neural network that consists of multiple layers of nodes that feed data throughout the network. The layers between the input layer and the output layer are called hidden layers and are one of the key differences between this and logistic regression (Scikit-learn, n.d.).

## Results

The following results were obtained from running cross-validation. Cross-validation is where the test set is split into n number of smaller test sets which are then used to find the performance of the model (Scikit-learn, n.d.). The average value is then computed in a loop. The n value used was 10.

|  |  |  |
| --- | --- | --- |
| Model | Accuracy | Time to Fit |
| Multinomial Naive Bayes | 85.4% | 0.2 seconds |
| Decision Tree | 72.2% | 39 seconds |
| Support Vector Machine (SVM) | 89.4% | 11067 seconds (>3 hours) |
| Logistic Regression (LR) | 88.9% | 15 seconds |
| Multi-layer Perceptron (MLP) | 89.3% | 17 seconds |

The results show that the Support Vector Machine, Logistic Regression and Multi-layer Perceptron models all have a very similar accuracy with the Support Vector machine just edging out on top by 0.1%. The Decision Tree performed the worst. The Support Vector Machine, though the most accurate, also took the longest (over 3 hours) with the Multinomial Naive Bayes model taking the shortest time of 0.2 seconds.

Whilst the Multinomial Naive Bayes model takes the shortest time by far, it is also 4% less accurate than SVM, LR and MLP. The SVM model takes over 3 hours to fit which is not good, especially when LR and MLP models perform similarly but in a matter of seconds. As the time difference between the MLP and LR models is only 2 seconds, the trade-off for a 0.4% increase in accuracy is worth it to use the Multi-layer Perceptron.

It should be noted that the reason for the SVM taking so long is that “The fit time scales at least quadratically with the number of samples” (Scikit-learn, n.d.). Sklearn suggests to use a LinearSVC model but even with 5000 iterations taking over 8 minutes, the model failed to converge and was therefore not included in the results. It also only predicted with an accuracy of 85%, thus would still not have beaten the MLP model.

Compared to the published methods in the background reading section, the MLP model outperforms the results from both papers. The Naive Bayes model also performed better than the model created by (Baid, Gupta, & Chaplot, 2017) by around 5% which can be put down to the large increase in the amount of data (2,000 rows to 50,000 rows).

## Investigating the impact of changes to the models

To ensure that the results were comparable throughout multiple iterations, set seeds were used as much as possible. This is shown in the train\_test\_split() function call in figure 10. The seed is set as random\_state=87, thus ensuring that the data is split the same each time the program is ran.

The Multi-layer perceptron model gives the largest impact when changing parameters at model creation. For example, setting the alpha value to 0.1 give an accuracy of 89% whereas setting the value to 0.5 gives an accuracy of 50%. The alpha is a regularisation hyperparameter whereby the smaller the value, the higher the accuracy but the higher likelihood of overfitting. Additionally, the solver can be changed for the MLP. This report uses the ‘adam’ solver as it works the best on large datasets in terms of accuracy and training time. However, solvers such as ‘lbfgs’ perform much worse with an accuracy of 50%.

## Investigating the impact of changes to cleaning the data

The table below shows the impact of changing the amount of cleaning that is done to the data, ranging from no cleaning to only removing html tags to removing html tags, punctuation stop words and stemming words. The Support Vector Machine model was not included in this investigation due to the exceedingly large fitting times.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naive Bayes | Decision Tree | Logistic Regression | Multi-layer Perceptron |
| No cleaning | 85.4% | 71.4% | 89.1% | 89.5% |
| Removal of html tags only | 85.4% | 71.7% | 89.1% | 89.6% |
| Full clean | 85.4% | 72.2% | 88.9% | 89.3% |

Interestingly, there isn’t much noticeable change for each model. As is suggested by Mass et al., perhaps this is because some words that are removed when cleaning the data are actually useful in judging sentiment as mentioned earlier in this report. Additionally, the removal of all punctuation will result in emoticons or exclamation marks also being removed which can also be used to judge sentiment (Maas, et al., 2011).

## Investigating Confusion Matrices to Understand Model Predictions

Figures 11, 12, 13 and 14 in the supplementary show model confusion matrices.

The confusion matrix for the naive bayes model shows that there is a very slight bias to predicting more negative reviews correctly whereas the decision tree model is more biased to predicting positive reviews correctly. The logistic regression matrix shows the highest bias favouring predicting negative reviews correctly whereas the Multi-layer Perceptron predicts positive review correctly more often.

## Visualising the Tree for the Decision Tree Model

Figure 15, 16 and 17 in the supplementary material shows visualisations of the tree structure of the Decision Tree model. Figure 17 shows a more zoomed in version of the graph with information about each node in the tree. It can be seen that the root node is predicting positive with 35,000 samples referring to it (the amount of training data supplied). As the data flows down the tree, the number of samples lower and the classes change. Interestingly, only the left-most node on each level is of the negative class.

# Conclusion

The results from this investigation show that the Multi-layer Perceptron model is the most suited to this analysis as it performs the best within an acceptable amount of time. The Support Vector Machine, whilst the most accurate, also takes the longest by far. Additionally, cleaning the data does not have as much effect on the accuracy as it may initially seem. Further extensions into this analysis could involve additional involvement in the cleaning of data, perhaps creating a new bag of words and exploring how cleaning of data actually affects the accuracy in more depth.

# References

Baid, P., Gupta, A., & Chaplot, N. (2017). Sentiment Analysis of Movie Reviews using Machine. *International Journal of Computer Applications* , (pp. 45-49).

Lang, N. (2022, February 19). *Stemming vs. Lemmatization in NLP*. Retrieved from Towards Data Science: https://towardsdatascience.com/stemming-vs-lemmatization-in-nlp-dea008600a0

Maas, A., Daly, R., Pham, P., Huang, D., Ng, A., & Potts, C. (2011). Learning Word Vectors for Sentiment Analysis. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* (pp. 142-150). Portland, Oregon, USA: Association for Computational Linguistics. Retrieved from Learning Word Vectors for Sentiment Analysis.

Ray, S. (2023, February 3). *Naive Bayes Classifier Explained: Applications and Practice Problems of Naive Bayes Classifier*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

Scikit-learn. (n.d.). *Cross-validation: evaluating estimator performance*. Retrieved from Scikit-learn: https://scikit-learn.org/stable/modules/cross\_validation.html

Scikit-learn. (n.d.). *Neural network models (supervised)*. Retrieved from Scikit-learn: https://scikit-learn.org/stable/modules/neural\_networks\_supervised.html

Scikit-learn. (n.d.). *sklearn.svm.SVC*. Retrieved from Scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

Sckikit-learn. (n.d.). *sklearn.feature\_extraction.text.TfidfTransformer*. Retrieved from sckikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.TfidfTransformer.html#sklearn.feature\_extraction.text.TfidfTransformer

Stecanella, B. (2017, June 22). *Support Vector Machines (SVM) Algorithm Explained*. Retrieved from MonkeyLearn: https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

Yucebas, S., & Tintin, R. (2021). GovdeTurk: A Novel Turkish Natural Language Processing Tool for Stemming, Morphological Labelling and Verb Negation. *The International Arab Journal of Information Technology*, 148-157.

# Supplementary Material

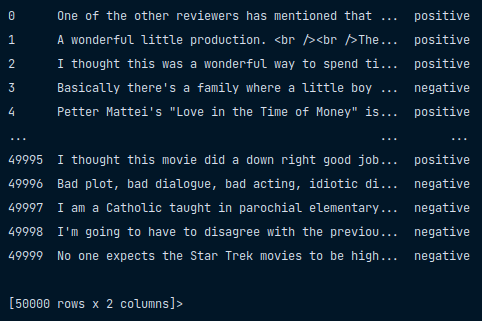


Figure df.head() - shows the dataframe in a nutshell.

## Regular Expressions Explanations

### Html Tag matching



Figure Code to remove html tags.

* re.sub() is used to replace a matching pattern.
* <.\*?> is the pattern to look for
  + The ‘<’ is the first character that the search will look for
  + The ‘.’ matches any character
  + The ‘\*’ will repeat the previous token (in this case, the ‘.’)
  + The ‘?’ makes the ‘\*’ a lazy quantifier
    - The ‘?’ is necessary **to only replace the html tag and not any text within two tags**
      * i.e. given ‘<b> This text is bold </b>’, the ‘?’ will only remove the html tags whereas without it the whole text will be removed.
  + The ‘>’ is the last character that the search will look for
* “ ” is what the html tag will be replace by (a white space)
* review\_val is the variable that will be searched

### Non-alphabet characters matching



Figure Code to remove punctuation.

* re.sub() is used to replace a matching pattern.
* [^a-zA-Z] is the pattern set
  + [] (the square brackets) are used to create a set of characters
  + ‘^’ matches any characters that are not in the set
  + ‘a-zA-Z’ are all the letters of the alphabet
  + Thus the pattern matches any characters that are not within the alphabet
* “ ” is what the matches will be replaced by (a white space)
* review\_val is the variable that will be searched

## Differences in Reviews after Running the Regex Code

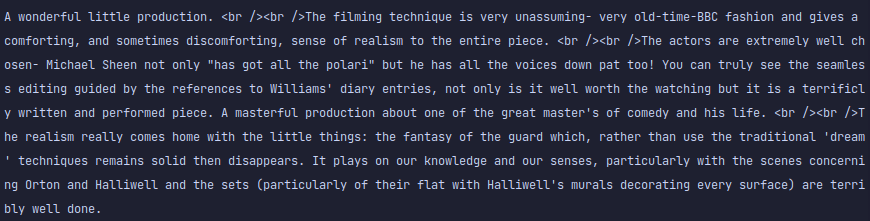


Figure A review before both regex lines of code is run.

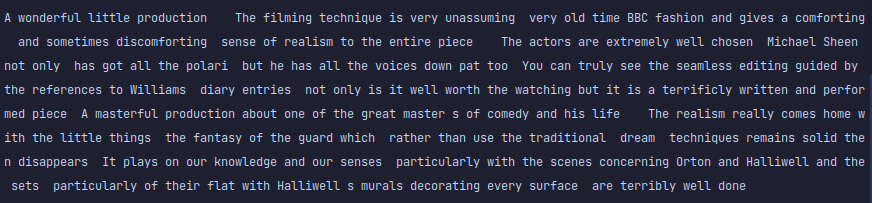


Figure A review after both regex lines of code is run.

## Differences in Reviews after Stemming and Removing Stop Words.

Figure A review before stemming and removal of stop words.

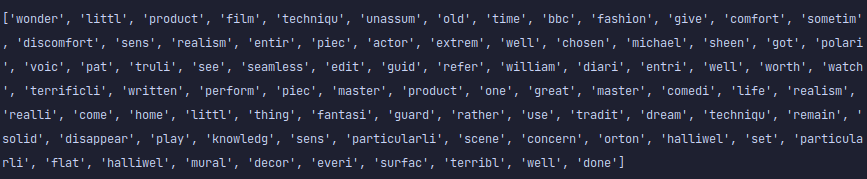


Figure A review after stemming and removal of stop words.

## Model creation

Text

Description automatically generated

Figure An example function for creating and testing a model

Text

Description automatically generated

Figure One hot encoder function



Figure The random\_state variable ensures the data is split the same each iteration

## Confusion Matrices for Models

Chart, treemap chart

Description automatically generated

Figure Naive Bayes confusion matrixl

Chart, treemap chart

Description automatically generated

Figure Decision Tree confusion matrix

Chart, treemap chart

Description automatically generated

Figure Logistic Regression confusion matrix

Chart, treemap chart

Description automatically generated

Figure Multi-layer Perceptron confusion matrix

## Decision Tree Visualisation

A picture containing text

Description automatically generated

Figure Full tree visualisation

A picture containing outdoor object

Description automatically generated

Figure 3 level visualisation

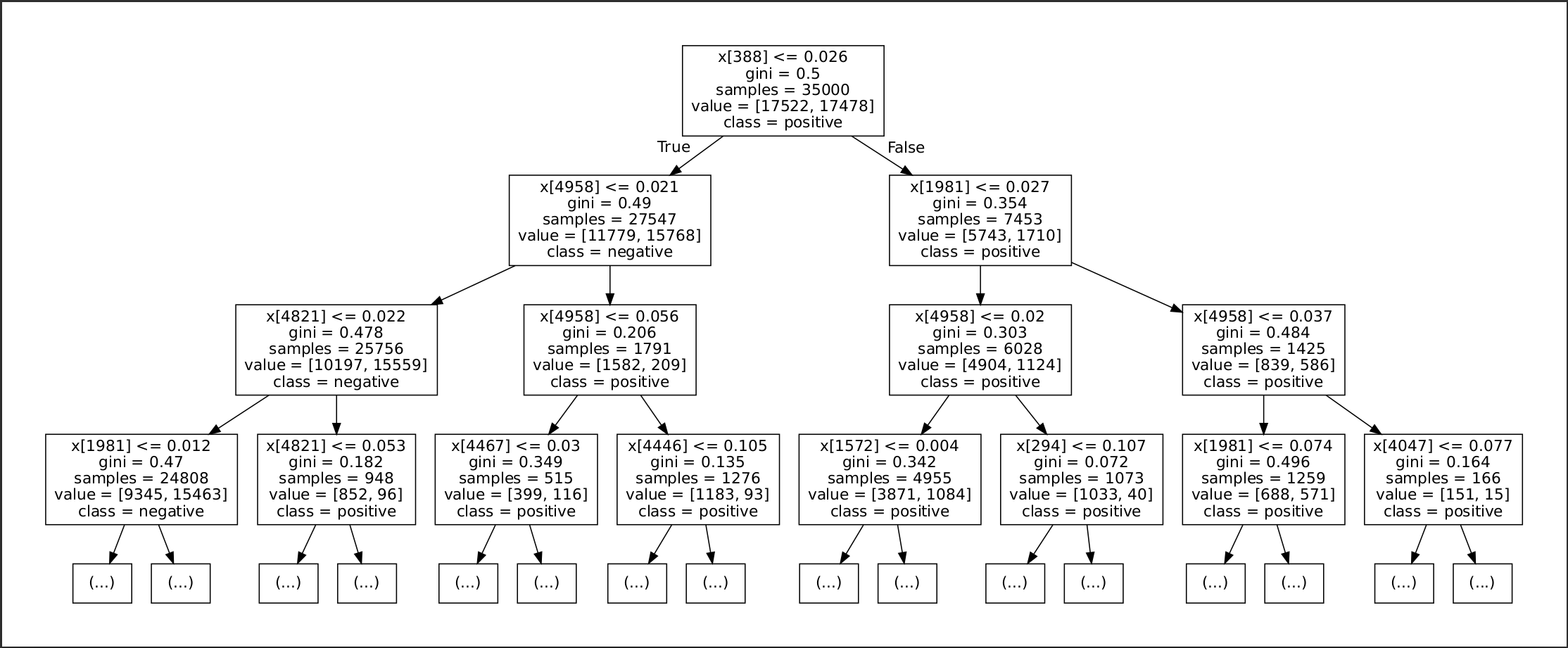


Figure 3 level visualisation with additional information

## Notebook location

The notebook and additional information will be stored at the following location:

<https://github.com/abdurrahmaan-desai/ai-assignment-submission>

## Code

# Import packages  
import re  
import sys  
import pandas as pd  
import numpy as np  
import nltk  
from nltk.corpus import stopwords  
from nltk.stem import PorterStemmer  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.model\_selection import train\_test\_split  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix  
from sklearn import tree, svm  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neural\_network import MLPClassifier  
from sklearn.preprocessing import LabelEncoder  
import time  
import graphviz  
from sklearn.model\_selection import KFold  
from sklearn.model\_selection import cross\_val\_score  
from numpy import mean, std  
import seaborn as sns  
import matplotlib.pyplot as plt  
#%%  
def pretty\_print(output: str):  
 *"""  
 pretty\_print formats the print statement a bit nicer* ***:param*** *output: Output to format* ***:return****:  
 """* print(f'\*\*\*\*\*\*\*\*\*\* {output} \*\*\*\*\*\*\*\*\*\*\n')  
#%%  
def investigate(df: pd.DataFrame):  
 *"""  
 investigate prints out important detail about the dataframe  
 Prints out the shape of the dataframe  
 Prints out the amount of null cells in the dataframe  
 Prints out a description of the dataframe (Count, unique values)  
 Prints out information about the dataframe (Number of entries, column names and non-null count)* ***:param*** *df: Dataframe to investigate* ***:return****:  
 """* pretty\_print('Shape')  
 print(df.shape)  
 print()  
  
 pretty\_print('Number of null values')  
 print(df.isnull().sum())  
 print()  
  
 pretty\_print('Description of dataframe')  
 print(df.describe())  
 print()  
  
 pretty\_print('Information about dataframe')  
 print(df.info())  
 print()  
#%%  
def clean\_data(reviews: list) -> list:  
 *"""  
 clean\_data iterates over the reviews, removes html tags, removes punctuation, lowers all capital letters, removes  
 stop words and stems the remaining words* ***:param*** *reviews: reviews to clean* ***:return****: returns the cleaned list  
 """* nltk.download('stopwords')  
 ps = PorterStemmer()  
 corpus = []  
  
 for review in reviews:  
 review\_val = re.sub('<.\*?>', " ", review)  
 review\_val = re.sub("[^a-zA-Z]", " ", review\_val)  
 review\_val = review\_val.lower()  
 review\_val = review\_val.split()  
 review\_val = [ps.stem(word) for word in review\_val if word not in set(stopwords.words('english'))]  
 review\_val = ' '.join(review\_val)  
 corpus.append(review\_val)  
  
 return corpus  
#%%  
def remove\_html\_tags(reviews: list) -> list:  
 *"""  
 remove\_html\_tags removes the html tags from the review and lowers all capital letters* ***:param*** *reviews: reviews to clean* ***:return****: returns the list of reviews  
 """* corpus = []  
  
 for review in reviews:  
 review\_val = re.sub('<.\*?>', " ", review)  
 review\_val = review\_val.lower()  
 corpus.append(review\_val)  
  
 return corpus  
#%%  
def vectorise\_data(corpus, save\_data=False):  
 *"""  
 vectorise\_data turns the words into a numerical representation* ***:param*** *corpus: List to vectorise* ***:param*** *save\_data: Option to save the vectorised representation to a text file* ***:return****:  
 """* cv = TfidfVectorizer(max\_features=5000)  
 data = cv.fit\_transform(corpus).toarray()  
  
 if save\_data:  
 np.savetxt('vectorised\_data', data)  
  
 return data  
  
#%%  
def print\_train\_test\_shapes(x\_train\_param, x\_test\_param, y\_train\_param, y\_test\_param):  
 *"""  
 print\_train\_test\_shapes prints the shape of each parameter* ***:param*** *x\_train\_param: Review data to train* ***:param*** *x\_test\_param: Review data to test* ***:param*** *y\_train\_param: Sentiment data to train* ***:param*** *y\_test\_param: Sentiment data to test* ***:return****:  
 """* pretty\_print('Train and test shapes for x and y')  
 print(x\_train\_param.shape)  
 print(x\_test\_param.shape)  
 print(y\_train\_param.shape)  
 print(y\_test\_param.shape)  
  
#%%  
def output\_timings(start, fit, predict):  
 *"""  
 output\_timings prints out the time taken to fit the model and predict the output* ***:param*** *start: Time when the function started* ***:param*** *fit: Time after the fitting of model had completed* ***:param*** *predict: Time after the prediction had been completed* ***:return****:  
 """* fit\_timing = fit - start  
 prediction\_timing = predict - fit  
  
 fit\_timing\_output = f'Time taken to fit the model: {fit\_timing} seconds'  
 prediction\_timing\_output = f'Time taken to predict the output: {prediction\_timing} seconds'  
  
 print()  
 pretty\_print(fit\_timing\_output)  
 pretty\_print(prediction\_timing\_output)  
#%%  
def output\_predictions(model: str, predictions, y\_test\_param):  
 *"""  
 output\_predictions prints out the accuracy score, the confustion matrix and the classification report of the model  
 that has been created* ***:param*** *model: Model that has been created* ***:param*** *predictions: The predictions that were made with the model* ***:param*** *y\_test\_param: The sentiment test values* ***:return****:  
 """* print('\n\n')  
 pretty\_print(f'{model} Predictions')  
  
 print(accuracy\_score(y\_test\_param, predictions))  
  
 cm = confusion\_matrix(y\_test\_param, predictions)  
 print(cm)  
 sns.set()  
 ax = sns.heatmap(cm, annot=True, fmt='.0f', xticklabels=['positive', 'negative'],  
 yticklabels=['positive', 'negative'])  
 plt.show(block=True)  
  
 print(classification\_report(y\_test\_param, predictions))  
#%%  
def one\_hot\_encoder(arr):  
 *"""  
 one\_hot\_encoder converts the text representation of the sentiment to binary integer values* ***:param*** *arr: Array to be encoded* ***:return****: The encoded array  
 """* label\_encoder = LabelEncoder()  
 integer\_encoded = label\_encoder.fit\_transform(arr)  
  
 return integer\_encoded  
  
#%%  
def cross\_validation(model, x, y):  
 *"""  
 cross\_validation runs the accuracy calculation 10 times and computes the average* ***:param*** *model: Model to compute* ***:param*** *x: Review test values* ***:param*** *y: Sentiment test values* ***:return****:  
 """* cv = KFold(n\_splits=10, random\_state=1, shuffle=True)  
 scores = cross\_val\_score(model, x, y, scoring='accuracy', cv=cv, n\_jobs=1)  
 pretty\_print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))  
#%%  
def run\_naive\_bayes(x\_train\_param, x\_test\_param, y\_train\_param, y\_test\_param):  
 *"""  
 run\_naive\_bayes creates, fits and predicts the Multinomial Naive Bayes model  
 Timing of the fitting and predicting process also occurs  
 Data is sent to the output\_predictions and output\_timings functions* ***:param*** *x\_train\_param: Review data to train* ***:param*** *x\_test\_param: Review data to test* ***:param*** *y\_train\_param: Sentiment data to train* ***:param*** *y\_test\_param: Sentiment data to test* ***:return****:  
 """* pretty\_print("Creating Naive Bayes model")  
 mnb = MultinomialNB()  
  
 pretty\_print("Fitting model")  
 start = time.time()  
 mnb.fit(x\_train\_param, y\_train\_param)  
 fitting\_time = time.time()  
  
 pretty\_print("Predicting test data")  
 pred = mnb.predict(x\_test\_param)  
  
 prediction\_time = time.time()  
  
 output\_predictions('Naive Bayes', pred, y\_test\_param)  
 output\_timings(start, fitting\_time, prediction\_time)  
  
 pretty\_print("Cross validating")  
 cross\_validation(mnb, x\_test\_param, y\_test\_param)  
#%%  
def run\_decision\_tree(x\_train\_param, x\_test\_param, y\_train\_param, y\_test\_param):  
 *"""  
 run\_decision\_tree creates, fits and predicts the Decision Tree model  
 Timing of the fitting and predicting process also occurs  
 Data is sent to the output\_predictions and output\_timings functions* ***:param*** *x\_train\_param: Review data to train* ***:param*** *x\_test\_param: Review data to test* ***:param*** *y\_train\_param: Sentiment data to train* ***:param*** *y\_test\_param: Sentiment data to test* ***:return****:  
 """* pretty\_print("Creating Decision Tree model")  
 clf = tree.DecisionTreeClassifier()  
  
 pretty\_print("Fitting model")  
 start = time.time()  
 clf = clf.fit(x\_train\_param, y\_train\_param)  
 fitting\_time = time.time()  
  
 pretty\_print("Predicting test data")  
 pred = clf.predict(x\_test\_param)  
 prediction\_time = time.time()  
  
 pretty\_print("Plotting tree")  
 tree.plot\_tree(clf)  
 dot\_data = tree.export\_graphviz(clf, out\_file=None, max\_depth=3, class\_names=['positive', 'negative'])  
 graph = graphviz.Source(dot\_data)  
 graph.render("test")  
  
 output\_predictions('Decision Tree', pred, y\_test\_param)  
 output\_timings(start, fitting\_time, prediction\_time)  
  
 pretty\_print("Cross validating")  
 cross\_validation(clf, x\_test\_param, y\_test\_param)  
#%%  
def run\_support\_vector\_machine(x\_train\_param, x\_test\_param, y\_train\_param, y\_test\_param):  
 *"""  
 run\_support\_vector\_machine creates, fits and predicts the Support Vector Machine model  
 Timing of the fitting and predicting process also occurs  
 Data is sent to the output\_predictions and output\_timings functions* ***:param*** *x\_train\_param: Review data to train* ***:param*** *x\_test\_param: Review data to test* ***:param*** *y\_train\_param: Sentiment data to train* ***:param*** *y\_test\_param: Sentiment data to test* ***:return****:  
 """* pretty\_print("Creating Support Vector Machine model")  
 clf = svm.SVC()  
  
 pretty\_print("Fitting model")  
 start = time.time()  
 clf.fit(x\_train\_param, y\_train\_param)  
 fitting\_time = time.time()  
  
 pretty\_print("Predicting test data")  
 pred = clf.predict(x\_test\_param)  
 prediction\_time = time.time()  
  
 output\_predictions('Support Vector Machine', pred, y\_test\_param)  
 output\_timings(start, fitting\_time, prediction\_time)  
  
 pretty\_print("Cross validating")  
 cross\_validation(clf, x\_test\_param, y\_test\_param)  
#%%  
def run\_logistic\_regression(x\_train\_param, x\_test\_param, y\_train\_param, y\_test\_param):  
 *"""  
 run\_logistic\_regression creates, fits and predicts the Logistic Regression model  
 Timing of the fitting and predicting process also occurs  
 Data is sent to the output\_predictions and output\_timings functions* ***:param*** *x\_train\_param: Review data to train* ***:param*** *x\_test\_param: Review data to test* ***:param*** *y\_train\_param: Sentiment data to train* ***:param*** *y\_test\_param: Sentiment data to test* ***:return****:  
 """* pretty\_print("Creating Logistic Regression model")  
 clf = LogisticRegression(random\_state=1, max\_iter=1500, solver='saga')  
  
 pretty\_print("Fitting model")  
 start = time.time()  
 clf.fit(x\_train\_param, y\_train\_param)  
 fitting\_time = time.time()  
  
 pretty\_print("Predicting test data")  
 pred = clf.predict(x\_test\_param)  
 prediction\_time = time.time()  
  
 output\_predictions('Logistic Regression', pred, y\_test\_param)  
 output\_timings(start, fitting\_time, prediction\_time)  
  
 pretty\_print("Cross validating")  
 cross\_validation(clf, x\_test\_param, y\_test\_param)  
#%%  
def run\_mlp(x\_train\_param, x\_test\_param, y\_train\_param, y\_test\_param):  
 *"""  
 run\_mlp creates, fits and predicts the Multi-layer Perceptron model  
 Timing of the fitting and predicting process also occurs  
 Data is sent to the output\_predictions and output\_timings functions* ***:param*** *x\_train\_param: Review data to train* ***:param*** *x\_test\_param: Review data to test* ***:param*** *y\_train\_param: Sentiment data to train* ***:param*** *y\_test\_param: Sentiment data to test* ***:return****:  
 """* pretty\_print("Creating Multi-layer Perceptron model")  
 clf = MLPClassifier(solver='adam', activation='logistic', alpha=1e-5, hidden\_layer\_sizes=(16, 16),  
 random\_state=1, max\_iter=200, verbose=False, early\_stopping=True)  
  
 pretty\_print("Fitting model")  
 start = time.time()  
 clf.fit(x\_train\_param, y\_train\_param)  
 fitting\_time = time.time()  
  
 pretty\_print("Predicting test data")  
 pred = clf.predict(x\_test\_param)  
 prediction\_time = time.time()  
  
 output\_predictions('Machine learning classification', pred, y\_test\_param)  
 output\_timings(start, fitting\_time, prediction\_time)  
  
 pretty\_print("Cross validating")  
 cross\_validation(clf, x\_test\_param, y\_test\_param)  
#%%  
def generate\_data(clean\_level: int = 0) -> (list, list, list, list):  
 *"""  
 generate\_data reads in the data from the csv and passes it to the investigate function  
 If the clean\_level parameter is 0, no cleaning will be done to the data  
 If the clean\_level parameter is 1, only html tags will be removed  
 If the clean\_level parameter is 2, the data will be cleaned using the clean\_data function  
 The reviews are vectorised  
 The sentiments are one hot encoded  
 The test and train data are split into a 70/30 split with a random state of 87 to allow for reproducible tests  
 The shapes of the split data are printed out* ***:param*** *clean\_level: The level of cleaning to be done to the data* ***:return****: The test and train data  
 """* pretty\_print("Reading csv")  
 df = pd.read\_csv('IMDB Dataset.csv')  
 investigate(df)  
  
 reviews, sentiments = df['review'], df['sentiment']  
  
 if clean\_level == 0:  
 pretty\_print("Data will not be cleaned")  
 elif clean\_level == 1:  
 pretty\_print("Removing html tags")  
 corpus = remove\_html\_tags(reviews)  
 elif clean\_level == 2:  
 pretty\_print("Cleaning data")  
 corpus = clean\_data(reviews)  
 else:  
 sys.exit("Invalid clean level option")  
  
 if clean\_level == 0:  
 pretty\_print("Vectorising review data")  
 vectorised\_data = vectorise\_data(reviews)  
 else:  
 pretty\_print("Vectorising review data")  
 vectorised\_data = vectorise\_data(corpus)  
  
 pretty\_print("Encoding sentiment data")  
 sentiments = one\_hot\_encoder(sentiments)  
  
 pretty\_print("Splitting data")  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(vectorised\_data, sentiments, train\_size=0.7, random\_state=87)  
  
 print\_train\_test\_shapes(x\_train, x\_test, y\_train, y\_test)  
  
 return x\_train, x\_test, y\_train, y\_test  
#%%  
x\_train, x\_test, y\_train, y\_test = generate\_data(0)  
#%%  
run\_naive\_bayes(x\_train, x\_test, y\_train, y\_test)  
#%%  
run\_decision\_tree(x\_train, x\_test, y\_train, y\_test)  
#%%  
# Note: This takes over 3 hours to run!  
run\_support\_vector\_machine(x\_train, x\_test, y\_train, y\_test)  
#%%  
run\_logistic\_regression(x\_train, x\_test, y\_train, y\_test)  
#%%  
run\_mlp(x\_train, x\_test, y\_train, y\_test)