DATAMINING

Detecting Clickbait in Online News Media

Second Project

8 January 2022

Introduction

Many people nowadays get the news from online sources. More than eight-in-ten Americans get news online, from digital devices [1]. As companies fight for users' attention and time and try to get more clicks and generate more traffic, headlines tend to become sensationalized, misleading and deceptive, designed to trick users into clicking on specific links that in most cases do not contain any meaningful information. These titles are called clickbait.

In this project we try to design models using supervised data mining algorithms to classify a news headline as being clickbait or not.

Description of original data

To carry out the project, we have used two sets of news headlines, one representing clickbait and another representing non-clickbait headlines. The data can be obtained from an online repository (see [2]). It was originally used in a paper about detection and prevention of clickbait in online news [3].

The first set consists of 15999 clickbait news titles which were labelled as “True” when we worked with the models. The second one consists of 16001 non-clickbait news titles, which were labelled as “False”. There are 32000 headlines in total. The length of the titles goes from one word to 27 words.

As a curiosity, in the following word cloud diagrams we can see a representation of the most representative words in the two datasets.

Figure 1: Word cloud of **clickbait** dataset

Figure 2: Word cloud of **non-clickbait** dataset

We observed that in the clickbait dataset the most frequent words are those that express more general concepts and are common in everyday life like “Thing”, “Know”,“Time”, “People”, “Life”, “Love”. These are more likely to reach a bigger audience. Whereas in the non-clickbait dataset the most frequent words express ideas mainly oriented towards politics, foreign affairs and crime.

Description of pre-processing of data

The first and most basic thing that comes to mind when dealing with a set of texts is whether the letter case matters. A character is capitalized when it starts a sentence, is the first letter of a proper noun word, when is part of an emphasized word, of an acronym or of a contraction. Taking in account the idea of emphasized words and differentiating the proper nouns from common ones is a too fine grain analysis and might mislead and overfit the model, so we decided that making all the words lowercase would make things easier and maybe give better results.

When working with text data, depending on the context, certain types of words do not provide valuable information for the classification. Such tokens are stop-words, punctuation marks and numbers. By eliminating this kind of tokens we might lose information that the original dataset offers. To minimize the loss we must analyze our situation. In our case we observed that the clickbait headlines often contain and start with numbers like in “16 Perfect Responses ...” or “15 Resolutions ...”, thus we decided that eliminating numbers completely will be a considerable loss of knowledge, instead we decided to replace each number by a specific token NUMBER_SPECIAL_TOKEN that will represent a number. This way the model will be able to make a difference between the headlines that have numbers, have many numbers or don't have numbers at all and not if it contains a specific number or not.

Another matter to take in consideration is whether it is worth having words that originate from the same root as different tokens. For example should the words “run”, “running” and “ran” be treated as different words. Well, in the case of clickbait detection, by visual examination, we don’t see any

evidence that clickbait headlines tend to contain more words in a specific form than the other headlines. Thus we decided to convert all words to the stem they originate from. This process usually is called stemming.

Before proceeding to make and test different models we needed to format the headlines so that they could be treated as input for the classification algorithms. To achieve this we considered an approach named bag of words, where every headline will be represented as a vector for all the words in the dataset and a cell will denote whether the headline contains or not a specific word. This approach is illustrated in table 1.

	the	quick	brown	fox	lazy	dog	and
the quick brown fox	1	1	1	1	0	0	0
the lazy dog	1	0	0	0	1	1	0
the quick fox and the dog	1	1	0	1	0	1	1
the lazy, lazy dog	1	0	0	0	1	1	0

Table 1: Representation of a set of headlines by indicating the presence of a word

Another extension of the idea based on frequencies is to use TF—IDF score. It stands for term frequency, inverse document frequency score. It takes into account not only the frequency of a word in a specific headline but also weights it based on the frequency of the word in the whole dataset. This way the most frequent words of the whole dataset have more importance in the representation of a specific headline.

To summarize, first we merged the datasets and made all characters for all titles lowercase. This was done using the standard python function available for all string instances. Then we replaced the numbers with a specific token, this was done using the replacement function provided by the regular expression module of python. Then, we tokenized the headlines, removed the stop-words, and stemmed the remaining tokens. In the end we generated the TfIdf or the frequency matrix for each title. While implementing the last preprocessing steps, we luckily found two scikit-learn classes TfidfVectorizer and CountVectorizer that provide methods that, with small changes, do just that and saved us a lot of time.

After the preprocessing step we got approximately 16000 features. While looking at the document frequency of the obtained tokens (figure 3), we saw that there are lots of tokens that appear only in a couple of documents and won't be very useful for classification. Also, training and testing with such a big matrix would have been slow and not practical. Thus, we decided to limit the number of features to 1000, that is, choose the tokens that appear in at least 40 headlines.

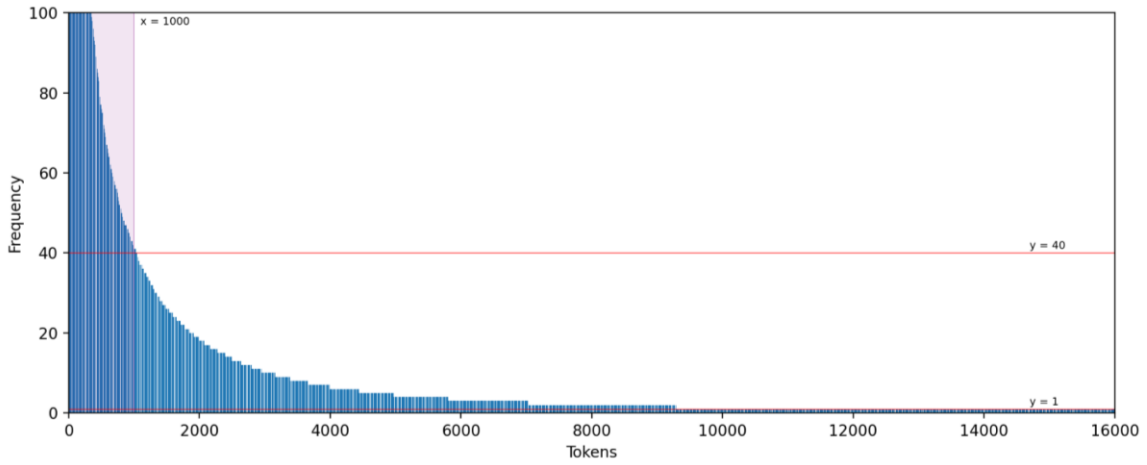
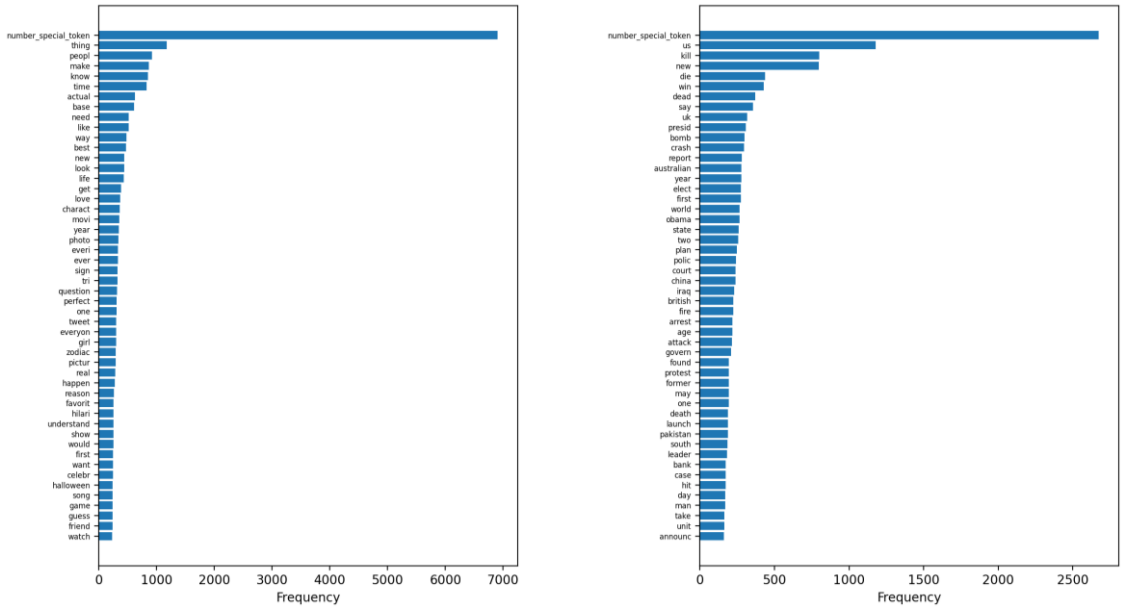


Figure 3: The frequency, in documents, of the tokens obtained after preprocessing. The purple region represents the selected set of tokens.

In order to make an idea about the content of the headlines we generate the graphs for the 50 most common tokens in the datasets.



(a) clickbait dataset

(b) non-clickbait dataset

Figure 4: 50 most frequent tokens, counted in documents, after preprocessing

In the end we split the data into a feature matrix and a label vector, this way we have the data prepared to build the models.

In figure 5 we illustrated a summary of the pre-processing phase.

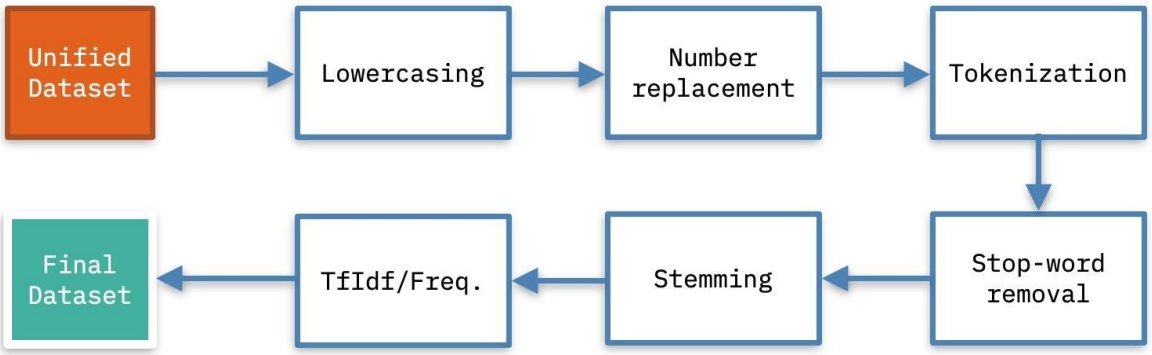


Figure 5: Pre-processing steps

Evaluation criteria of data mining models

The data set we used to train and test the models is quite big, 64000. Using methods like leave-one out would have been quite expensive, this is why to evaluate the models we used the K-Fold cross-validation method with k values consistent with reasonable execution time.

Because the data set is quite balanced 15999 vs. 16001, the acurray measure gives a fairly good comparison measure. In addition, we used the confusion matrix and the measures based on it like the recall, precision and F-measure and the ROC curve.

Execution of different machine learning methods

Naive Bayes

As we discussed previously, our dataset is a set of two textfiles, clickbait and non-clickbait comments, in order to reduce the number of attributes we removed the stop words. We also have changed the verbal forms and the number (singular/plural) to the base form (infinitive) of the verb and finally tried the TfIdf representation. These changes ensure a clean dataset, something very important when it comes to applying certain methods.

Firstly, we applied a separation into data and label, and used a Gaussian function to train the data, we used a Gaussian function because of the continuous data, we used cross validation data with k equal to ten and we obtained a mean score of 0.89, we also adjust the probability threshold to improve the algorithm and finally build a text report to show the main classification metrics and the result was:

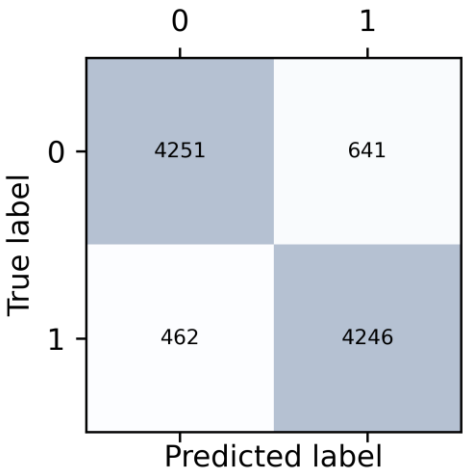


Figure 6: The confusion matrix

	precision	recall	f1-score	support
clickbait	0.90	0.87	0.89	4892
non-clickbait	0.87	0.90	0.89	4708
accuracy			0.89	9600

macro avg	0.89	0.89	0.89	9600
weighted avg	0.89	0.89	0.89	9600

Table 2: The average classification report for 10-fold cross-validation

The results obtained are successful, but in order to improve the metrics even more we used a function called GridSearchCV, that combines all the possibilities given some parameters to achieve the better score of some method. This is somewhat similar to what we have in the previous version code, but now we call a function instead. The values that we try are more fine-grained than the previous version. The final result was:

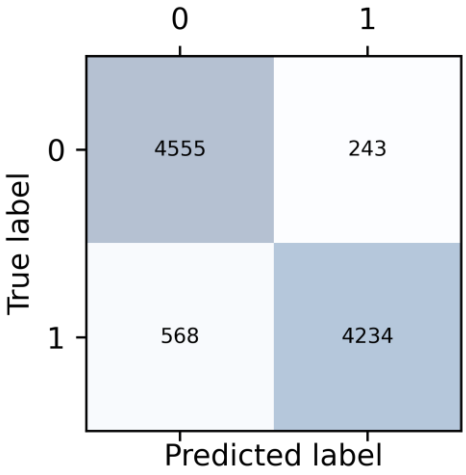


Figure 7: The confusion matrix v2

	precision	recall	f1-score	support
clickbait	0.89	0.95	0.92	4798
non-clickbait	0.95	0.88	0.91	4802
accuracy			0.92	9600
macro avg	0.92	0.92	0.92	9600
weighted avg	0.92	0.92	0.92	9600

Table 3: The average classification report for 10-fold cross-validation v2

As we can see, it is not a big improvement, but is a bit better than the other algorithm. Something that we should remark on is that we executed the code a few times to be sure about the

improvements and results obtained, due to the results can change when the train and test datasets also change.

Something that we notice is that the multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification), so for our case it might be better to use MultinomialNB instead of GaussianNB. As there is no better way to check this than by testing it, we modified the previous code to use this classifier. And the results that we obtained were not significantly better than the previous version.

	precision	recall	f1-score	support
clickbait	0.92	0.92	0.92	4800
non-clickbait	0.92	0.92	0.92	4800
accuracy			0.92	9600
macro avg	0.92	0.92	0.92	9600
weighted avg	0.92	0.92	0.92	9600

Table 4: The average classification report for 10-fold cross-validation v3 with multinomial classifier

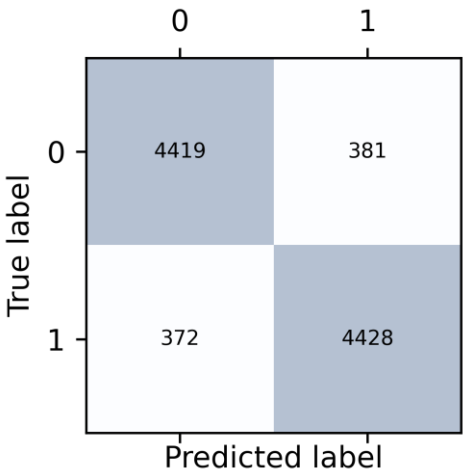


Figure 8: The confusion matrix v3 with multinomial classifier

Naive Bayes is called “naive” because of the assumption of independence of all features, in our case the features are the words, and these are not independent of each other. To construct a phrase we must give a meaning to the sentence, to do this we use words that depend on each other to give the sentence the meaning we want it to have, so this assumption is not entirely correct. But the accuracy scores that we obtained are surprisingly high.

Another thing to discuss is if we have enough elements to obtain reliable probabilities, the answer is yes, that can be seen in the pre-processing section.

K-NN

This method requires a clean dataset and a distance measure. As a dataset we tried the TfIdf representation of the headlines and the frequencies representation, as for the distance measure we tried Euclidean and Manhattan distances. In order to find the parameters that give best results we used the GridSearchCV class that provides methods that using an exhaustive search looks for the best values of specified parameters. Due to the high number of instances we used 5-fold cross-validation. For this method we gave the ranges of parameters presented in table 5.

Type of parameter	Values
Number of neighbors to use for kneighbors queries	1 - 15
Weight function used in prediction	uniform, distance
Power parameter for the Minkowski metric	1, 2

Table 5: Parameters for GridSearchCV

We tried higher powers (p = 3) for the Minkowski distance but it was very costly in time due to the high numbers of exponentiations needed to be computed.

TfIdf

The parameters that gave the best results for this representation are: nr_ neighbors = 4, weights = “distance” and the euclidean distance (p = 2). The accuracy with these parameters was 0.86.

Once we found the best parameters, to analyze the average performance of the model we did a 10-fold cross-validation and obtained an accuracy of 0.857 ± 0.05. The confusion matrix and the average classification report can be seen in figure 9 and table 10 respectively.

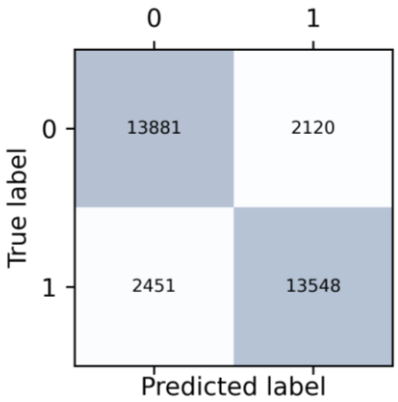


Figure 9: The confusion matrix for the k = 4, weight=”distance”, p = 2

	precision	recall	f1-score	support
clickbait	0.85	0.89	0.87	1600
non-clickbait	0.89	0.84	0.86	1600
accuracy			0.87	3200
macro avg	0.87	0.87	0.87	3200
weighted avg	0.87	0.87	0.87	3200

Table 6: The average classification report for 10-fold cross-validation and nr_neighbors = 4, weight="distance", p = 2

From this results can be said that the model correctly classifies a big portion of the headlines. It has a better precision for the non-clickbait headlines but the proportion of correctly identified non-clickbait headlines is smaller in comparison with the clickbait headlines.

This model is significantly better than a random model. In figure 10 is illustrated the ROC curve graphic of the model.

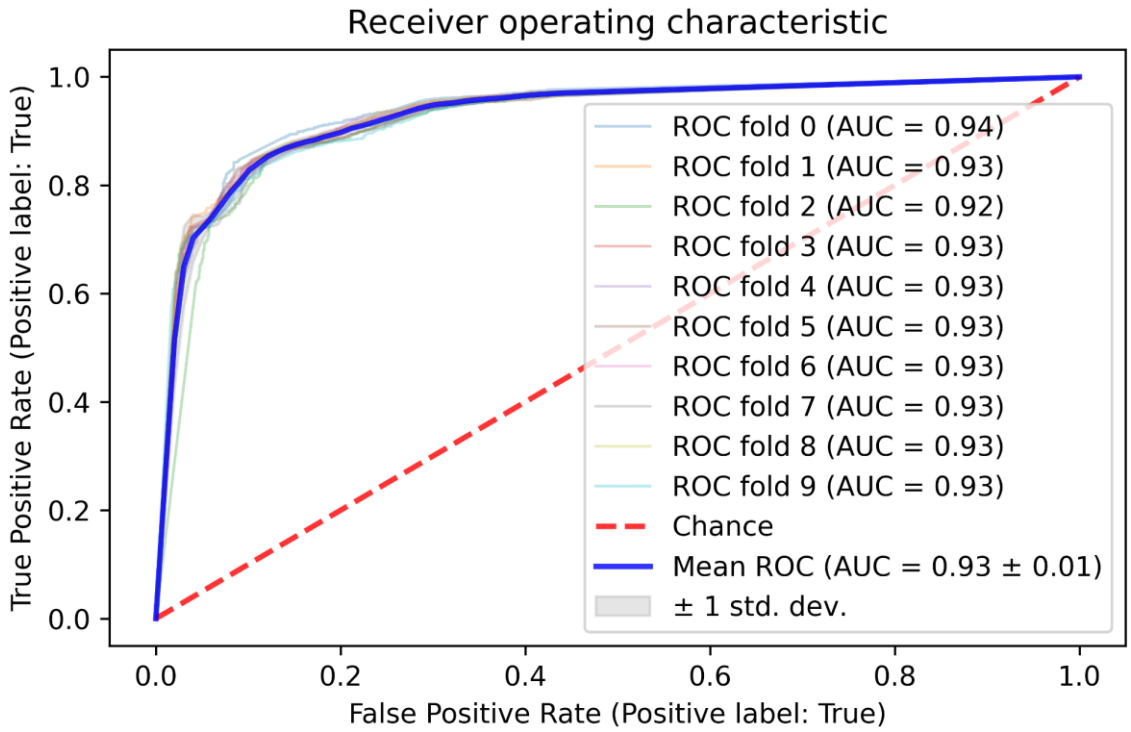


Figure 10: ROC curve and area under the curve for 10-fold cross-validation for the KNN model with k = 4, weight="distance", p = 2 and TfIdf representation

Frequencies

When repeating the procedure for the case when the headlines are represented by a vector of the frequencies of its tokens, that is, the number of times a token appears inside the headline, the parameters that give the best results are: nr_ neighbors = 6, weights = distance and euclidean distance (p = 2). The accuracy with these parameters was also 0.86.

For the 10-fold cross validation we obtained an accuracy of 0.847 ± 0.06. As in the previous case the confusion and the average classification report are presented in figure 11 and table 7 respectively.

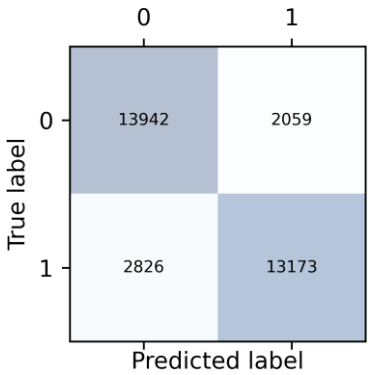


Figure 11: The confusion matrix for the k = 6, weight="distance", p = 2

	precision	recall	f1-score	support
clickbait	0.82	0.91	0.87	1600
non-clickbait	0.90	0.80	0.85	1600
accuracy			0.87	3200
macro avg	0.86	0.86	0.86	3200
weighted avg	0.86	0.86	0.86	3200

Table 7: The average classification report for 10-fold cross-validation

and nr_neighbors = 6, weight="distance", p = 2

Similar to the TfIdf case, the model correctly classifies a bigger portion of click bait then non-clickbait but has a better precision for the non-clickbait headlines. In comparison with the TfIdf the recall for the clickbait and the precision for the non-clickbait is even bigger, whereas the recall for the non-clickbait and the precision for the clickbait headlines is smaller.

The ROC graph is presented in the figure 12.

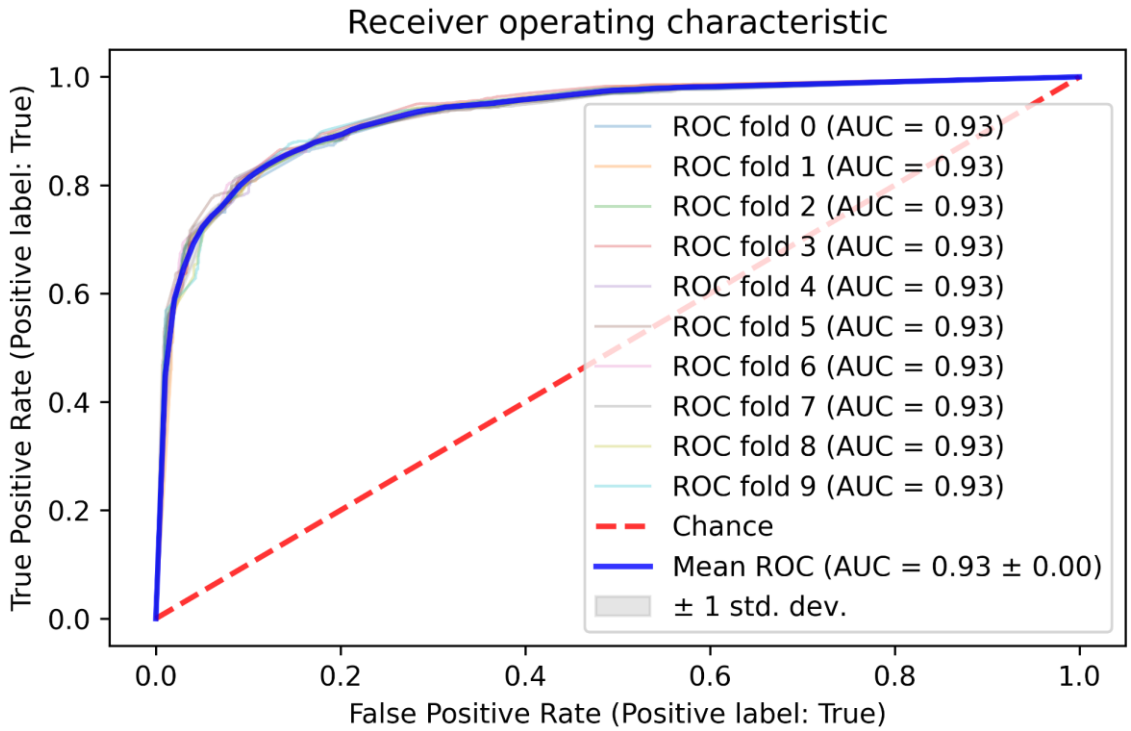


Figure 12: ROC curve and area under the curve for 10-fold cross-validation for the KNN model with k = 6, weight="distance", p = 2 and Frequencies representation

In conclusion, when using TfIdf representation, the obtained model seems more equilibrated with respect to the recall and precision of the Frequency method. Because we are more interested in correctly identifying clickbait titles we think that the TfIdf method would be a better choice because it has a better precision when classifying clickbaits. Although the results of the methods in this case are very close.

Decision trees

A decision tree classifier can be implemented in a number of different ways. For this project we decided to use Sklearn's decision tree classifier which is based on the CART (Classification and Regression Trees Classifier) . The Python script developed to classify with decision trees was based on the notebooks used during the lab session of the course.

Initial run of the decision tree classifier

As with the previous classification techniques used for this project, the preprocessed data with the TfIdf representation was used. The Sklearn decision tree classification algorithm was run with 70% train data and 30% test data. Initially, we ran the algorithm without specifying any of the parameters. In Table 8 the classification report for this tree can be seen. Remember that "True" corresponds to clickbait and that "False" corresponds to non-clickbait.

	precision	recall	f1-score	support
False	0.89	0.85	0.87	4850
True	0.85	0.89	0.87	4750
accuracy			0.87	9600
macro avg	0.87	0.87	0.87	9600
weighted avg	0.87	0.87	0.87	9600

Table 8: The classification report for the first decision tree without optimised parameters

More specifically, the accuracy for this tree is 0.87 and the 95% CI [0.863, 0.877] . To get a more detailed view of the model's performance we can check the confusion matrix for this decision tree, which can be seen in Figure 13. The confusion matrix shows that the number of false positives is higher than the number of false negatives.

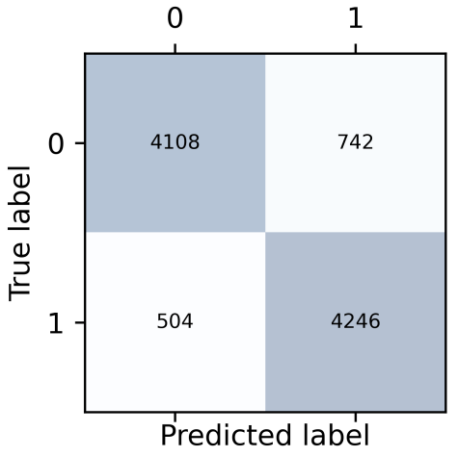


Figure 13: confusion matrix for the first decision tree, without optimised parameters.

The performance of this model can be evaluated with k-fold cross validation. The mean accuracy for running the k-fold cross validation, with k set to 10, is 0.87. That is very similar to that of our model, only a few decimals digits differ. Another thing to note is that we observed a large variance in the resulting decision trees when we generated the trees from the training data. That is something to keep in mind for further analysis.

Second run with GridSearchCV

To obtain a higher accuracy we used Sklearn's GridSearchCV method to find optimised parameters for the decision tree classifier function. This method was executed with the criterion parameter set to entropy. This way entropy is used to measure the quality of the split, which is connected to the information gain. The method was also run with cv set to 10, meaning that 10 cross validations would occur. During the executions of this script, the optimal parameters returned from this function varied. Most often and for the generated tree in this report, the "min_impurity_decrease" obtained was 0.0 while the "min_samples_split" obtained was 13. Both these parameters are connected to whether an internal node of the tree would split. "min_impurity_decrease" 0.0 means that a node would be split if this split would induce an impurity greater than or equal to this value. The default value for this parameter is 0, i.e. the same as the optimal parameter from GridSearchCV. "min_samples_split" set to 13 means that 13 is the minimal number to split an internal node where 2 is the default value.

With only slightly modified parameters, the classification report for this newly generated tree can be seen in Table 9. The tree has marginally higher values than the decision tree without optimised parameters. The accuracy of the second tree is 0.87 and the 95% CI [0.863, 0.877] .

	precision	recall	f1-score	support
False	0.89	0.85	0.87	4850
True	0.85	0.89	0.87	4750
accuracy			0.87	9600
macro avg	0.87	0.87	0.87	9600
weighted avg	0.87	0.87	0.87	9600

Table 9: The classification report for the first decision tree with optimised parameters from GridSearchCV

Furthermore, the second decision tree classifier is slightly better at labelling the test data correctly. This can be seen when comparing the confusion matrix in Figure 14 to the one of the previous decision tree seen in Figure 13. Once again the model has a higher number of false positives than false negatives.

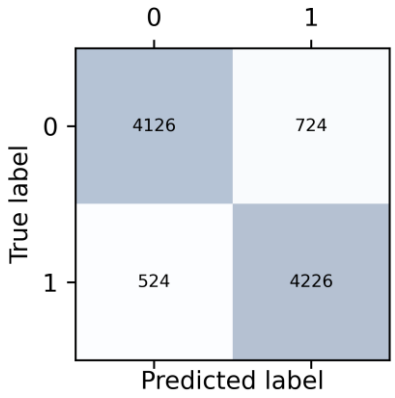


Figure 14: confusion matrix for the second decision tree, with optimised parameters through GridSearchCV.

Yet again k-fold cross validation with k set to 10 was executed to estimate the skill of our model on unseen data. The mean accuracy of these runs was 0.88, which is similar to the mean accuracy obtained for the first decision tree.

What has been established so far is that the GridSearchCV, which takes a long time to execute for our data, does not consistently return the same parameters and results in a model that has almost

the same metrics. When it comes to the structure of the trees they share a lot of similarities too. Since the two decision tree models are similar and that the second decision tree classifier had slightly better metrics, at least for the k-fold cross validation, this tree was used for further study.

One of the advantages with decision trees for classification is that they are easier to overview and interpret than some of the other machine learning tasks that were used for this project. . The tree seen in Figure 15 illustrates this. The number of nodes in the tree is 4125. Since there are a lot of features to consider for the decision tree classifier and not enough space to output the entire tree in this report, the max depth parameter for generating Figure 15 was set to three. The root node of the tree is the “number_special_token”, which corresponds to a headline containing numbers. This is also the variable that best splits the data. The two nodes that are directly reachable from the root are “know” and “kill”, which are reached according to the threshold value of 0.124. If it is less than or equal, we end up in the “know” node where the majority of the samples belong to the non-clickbait class. On the other hand, the majority of samples in the “kill” node belong to the clickbait class. In a similar pattern the following nodes are split until we reach leaves where a classification is made.

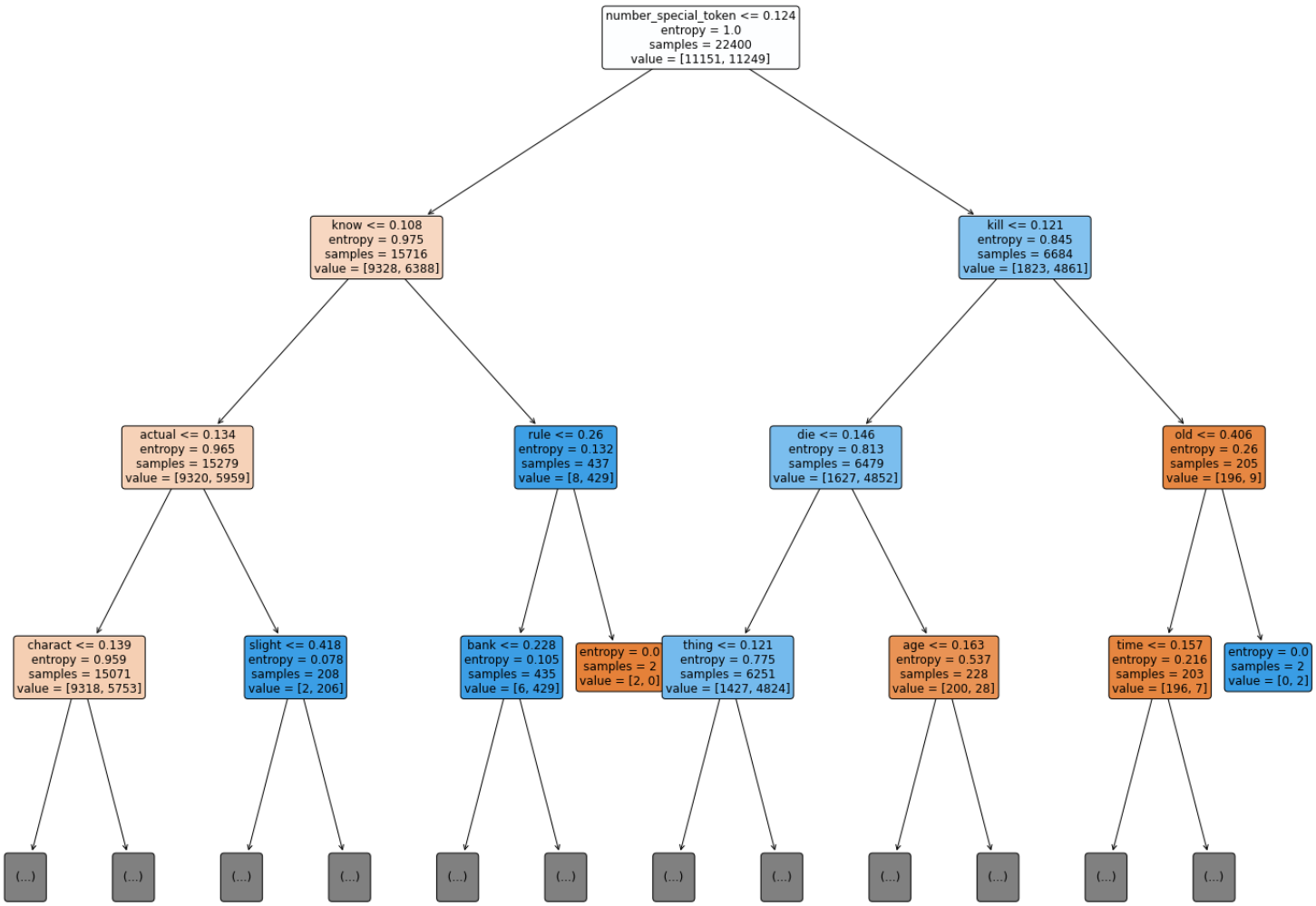


Figure 15: tree plot for the first decision tree, with optimised parameters through GridSearchCV.

As an example to see how our model would behave with an input we could simply choose for validation an instance of data containing different words with Tfldf scores. With an input where "number_special_token" is 0.2, "kill" is 0.2 and 'old' is 0.5 this would be classified as a clickbait as we would navigate down through the tree always taking the right (false) branch. We would eventually end up in the leaf node we see all the way to the right in Figure 15 and would get the classification to be clickbait. This path is represented by the rule

R1: If ¬(number_special_token≤0.124) ∧ ¬(kill≤0.121) ∧ ¬(old≤0.406) Then clickbait = True

Another one of the rules that are shorter and fully visible from the tree is that of:

R2: If (number_special_token≤0.124) ∧ ¬(know≤0.108) ∧ ¬(rule≤0.26) Then clickbait = False

With this tree in particular and the large amount of features mean that some of the rules are very long and are not feasible to fit in this report. Generally it is desirable to have as simple of a decision tree as possible to avoid overfitting.

The leaves we have seen so far are pure, i.e. all data points in the leaf belong to the same class. Sometimes when nodes cannot be expanded but still there are positive and negative values, the leaf is labelled by the majority class. That is an approach that is implemented in this tree too.

To conclude this section, as has been observed when working with this machine learning technique, there is a large variance in the generated trees. This variance stems from the approaches used when generating trees. When running the scripts, a large number of different trees have been generated depending on the split into training data, with different optimised parameters obtained from GridSearchCV as a consequence. Both of the final two trees generated do have a high accuracy but GridSearchCV fails to significantly increase the accuracy of the classifier and is costly to run. Furthermore the vastness of the tree, and the variance from splitting the training data, carries the risk of generating a model that overfits. With this particular dataset, the sheer amount of different features makes it hard to display the trees fully in this report and makes writing out all the rules infeasible. That the optimised tree contains some pure leaf nodes, as seen in Figure 15 is desirable for the classification and enhances the reliability of the tree.

Support Vector Machines

Firstly for this algorithm we have used the tf_idf scores and our target as values for splitting into train and test to use in our algorithm. We used gridsearchCV to find the best possible parameters and a 5-Fold-Cross-Validaton.

Type of parameter	Values
C or Regularization parameter	1 - 10 - 100 - 1000
Kernel type to be used in this algorithm	'linear', 'rbf'

Table 10: Parameters for GridSearchCV.

Despite achieving good precision for our model we decided to show our results in a confusion matrix to try to understand the classification as we can observe in the figure 16.

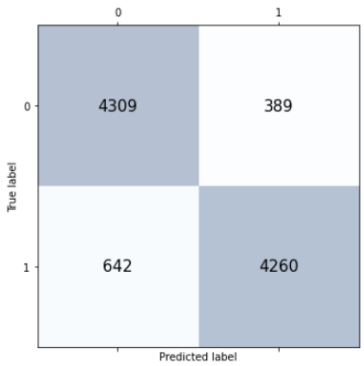


Figure 16: Confusion Matrix for SVM.

This confusion Matrix showed us a misclassification of the 642 false negatives and 389 false positives out of 9600 values.

The parameters that finally gave us a better performance was a Linear Kernel and a C parameter with value equal to 1, with this we have accomplished an accuracy of 0.8926 and a very good precision for both of the datasets, with an 0.87 and a 0.929.

	precision	recall	f1-score	support
clickbait	0.87	0.92	0.89	4698
non-clickbait	0.929	0.87	0.89	4902
accuracy			0.89	9600

macro avg	0.89	0.89	0.89	9600
weighted avg	0.89	0.89	0.89	9600

Table 11: The average classification report for 10-fold cross-validation and C= 1, Kernel="Linear", gamma = "scale"

From this results can be said that the model correctly classifies a big portion of the headlines. It has a better precision for the non-clickbait headlines but the proportion of correctly identified non-clickbait headlines is smaller in comparison with the clickbait headlines.

The next step has been to repeat the same process but instead of using the tf_idf frequencies using the boolean value of tf_idf as the input.

We used exactly the same method, firstly using the GridSearchCV for finding the best possible parameters and then fitting it into the model.

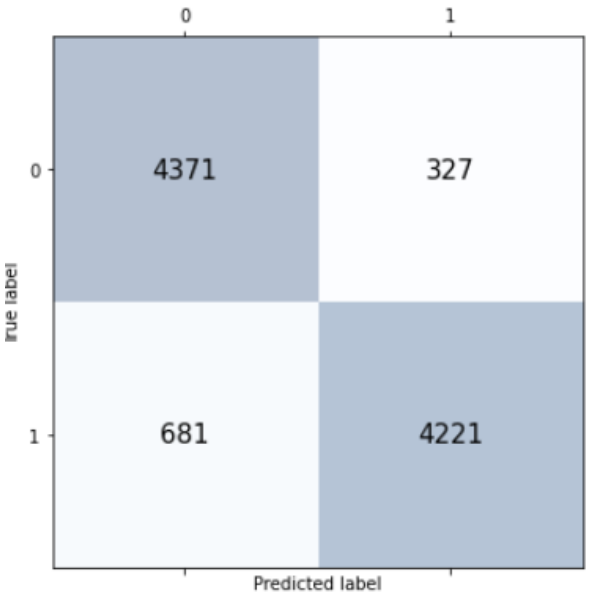


Figure 17: Confusion Matrix for SVM with boolean data.

Finally the precision was 0.895 while its best parameters were an rbfb kernel and a value of C equal to 1. The results for this are quite better than the ones before, so we can assume that the frequencies work worse than the boolean vectors.

	precision	recall	f1-score	support
clickbait	0.87	0.93	0.90	4698
non-clickbait	0.93	0.86	0.89	4902

accuracy			0.90	9600
macro avg	0.90	0.90	0.90	9600
weighted avg	0.90	0.90	0.90	9600

Table 12: The average classification report for 10-fold cross-validation and C= 1, Kernel="Linear", gamma = "scale"

Meta-learning methods

In order to see if we can obtain better results , we will combine the classifiers and try to use different meta learning methods.

The preprocessing data as before, with the TFID of the words appear in the titles of the clickbait and non clickbait. All the runs were with cross-validation of 10.

Voting Scheme

We run the first part of the script with the Voting Scheme. First, we use a combination of three classifiers such as the Naive Bayes, KNN and Decision Trees. For KNN we use the neighbors and weights that we determined in the previous chapter.

The results obtained was the follow:

Accuracy: 0.905 [Naive Bayes]

Accuracy: 0.849 [Knn (3)]

Accuracy: 0.873 [Dec. Tree]

Then we apply the Majority Vote and Weighted voting with the following accuracy:

Type of voting	Accuracy
Majority Voting	0.913
Weighted Voting	0.914

Table 13: Voting scheme with the three classifiers and weighted 2,1,2

As we can see, the results are pretty similar and very good.

Bagging

Then we try to use other techniques with bagging

Bagging	Bagging with max features=0.35
Accuracy: 0.866 [1]	Accuracy: 0.753 [1]
Accuracy: 0.873 [2]	Accuracy: 0.804 [2]
Accuracy: 0.882 [5]	Accuracy: 0.864 [5]
Accuracy: 0.887 [10]	Accuracy: 0.893 [10]
Accuracy: 0.888 [20]	Accuracy: 0.901 [20]
Accuracy: 0.889 [50]	Accuracy: 0.913 [50]
Accuracy: 0.891 [100]	Accuracy: 0.915 [100]
Accuracy: 0.891 [200]	Accuracy: 0.918 [200]

Table 14: Bagging without feature selection and with feature sele

This process took a long time around 1780 minutes to perform all the calculations.

We can observe that With feature selection we have better accuracy.

Random Forest

We try Random Forest with the following results:

Accuracy: 0.860 [1]

Accuracy: 0.865 [2]

Accuracy: 0.893 [5]

Accuracy: 0.898 [10]

Accuracy: 0.902 [20]

Accuracy: 0.902 [50]

Accuracy: 0.902 [100]

Accuracy: 0.902 [200]

Obtaining worts results than the bagging with feature selection (took 107 mins):

Extra Trees

With **ExtraTrees** classifier (took 166 mins):

Accuracy: 0.873 [1]

Accuracy: 0.876 [2]

Accuracy: 0.901 [5]

Accuracy: 0.907 [10]

Accuracy: 0.908 [20]

Accuracy: 0.910 [50]

Accuracy: 0.911 [100]

Accuracy: 0.911 [200]

Boosting

The next meta classifier is using boosting techniques with AdaBoost, with trees and GradientBoosting.

AdaBoost (49 min)	AdaBoost with DT (223 min)	Gradient Boosting (127 min)
Accuracy: 0.632 [1]	Accuracy: 0.700 [1]	Accuracy: 0.674 [1]
Accuracy: 0.641 [2]	Accuracy: 0.731 [2]	Accuracy: 0.684 [2]
Accuracy: 0.668 [5]	Accuracy: 0.776 [5]	Accuracy: 0.702 [5]
Accuracy: 0.707 [10]	Accuracy: 0.812 [10]	Accuracy: 0.719 [10]
Accuracy: 0.750 [20]	Accuracy: 0.846 [20]	Accuracy: 0.733 [20]
Accuracy: 0.808 [50]	Accuracy: 0.887 [50]	Accuracy: 0.768 [50]
Accuracy: 0.846 [100]	Accuracy: 0.905 [100]	Accuracy: 0.808 [100]
Accuracy: 0.880 [200]	Accuracy: 0.904 [200]	Accuracy: 0.839 [200]

Table 15: Accuracy of each boosting algorithms

The next figure 18 we can see the comparison between the different techniques of meta learning methods.

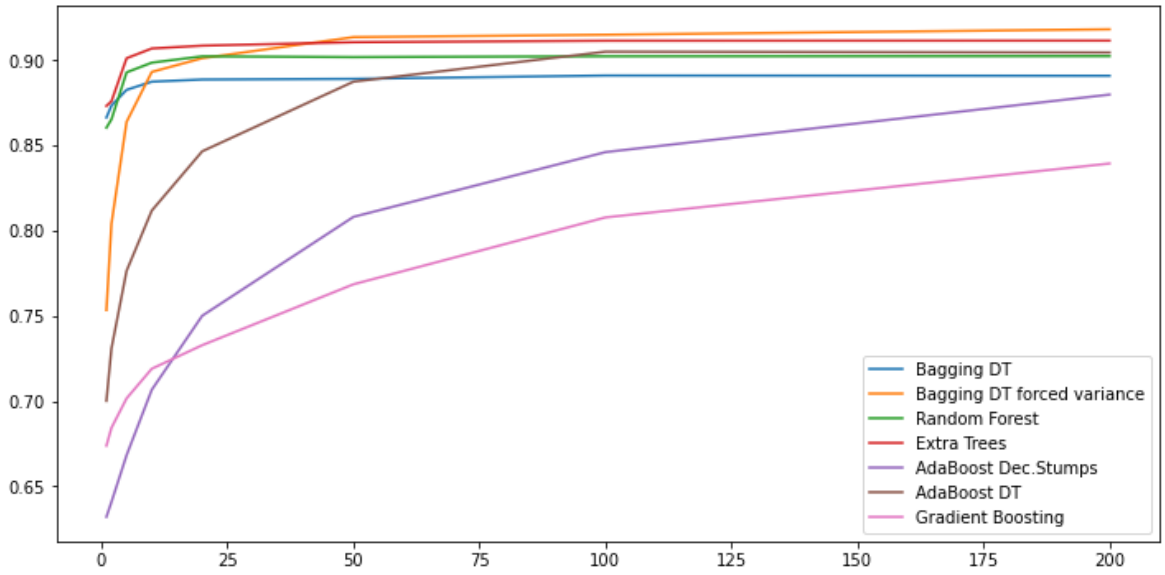


Figure 18: Comparison graph from all the meta learning methods

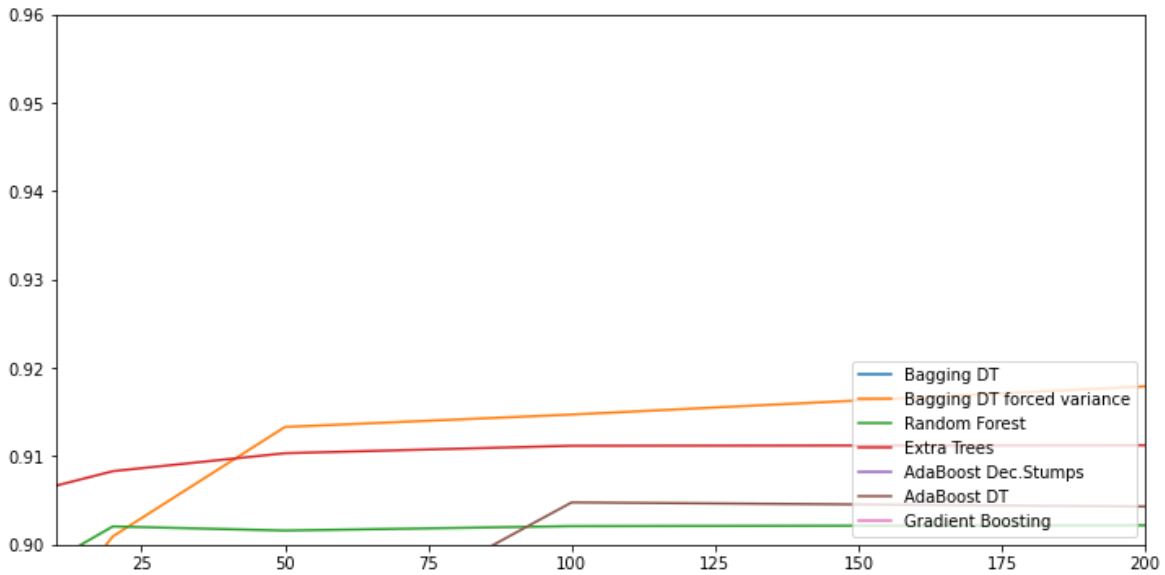


Figure 19: Zoomed comparison graph from all the meta learning methods

As we can observe, the best technique is the Bagging with DT and for the best performance in time, we should get the 50 estimators.

Feature Selection with Forests of Trees

The last technique applied was the Feature selection with random trees, first we need to determine the best threshold as we can observe in the figure 20.

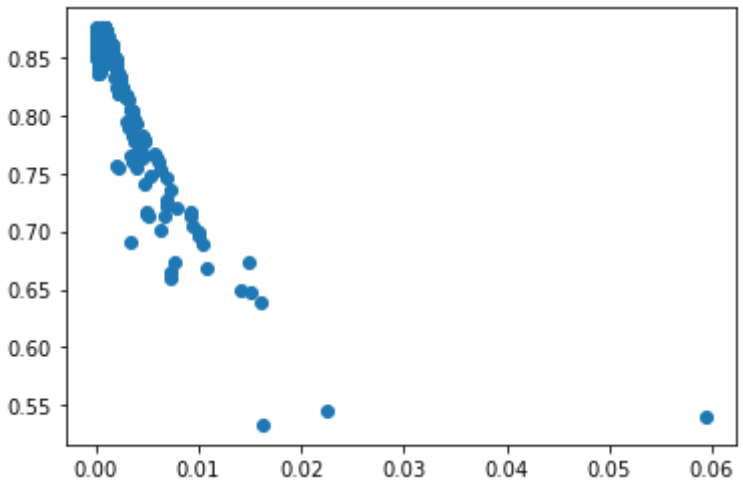


Figure 20: Determination of the best threshold for feature selection

Running the script for the KNN with 1 and the 9 neighbours we obtain the following accuracy (took 268 mins):

Original: 0.8584375

With FS: 0.875375

As we can see, we obtain better accuracy with feature selection. But not the best one, the best accuracy that we can obtain is using the Bagging with DT with 50 estimators.

The preprocessing data as before, with the TFID of the words appear in the titles of the clickbait and non clickbait. All the runs were with cross-validation of 10.

We also test the Meta Methods with the plain frequencies for words instead of their TF-IDF. We again used a cross-validation of 10.

Voting Scheme

We repeat the voting scheme process with this alternative preprocessing.

The results obtained was the follow:

Accuracy: 0.899 [Naive Bayes]

Accuracy: 0.850 [Knn (3)]

Accuracy: 0.882 [Dec. Tree]

If we look at the Majority and Weighted voting we find the following accuracies:

Type of voting	Accuracy
Majority Voting	0.906
Weighted Voting	0.909

Table 16: Voting scheme with the three classifiers and weighted 2,1,2

We can already see that the accuracy is slightly lower than with the previous method but we might as well keep testing this approach.

Bagging

We use the slow and tedious bagging technique again and find the following results:

Bagging	Bagging with max features=0.35
Accuracy: 0.849 [1]	Accuracy: 0.782 [1]
Accuracy: 0.860 [2]	Accuracy: 0.849 [2]
Accuracy: 0.872 [5]	Accuracy: 0.895 [5]

Accuracy: 0.877 [10]	Accuracy: 0.912 [10]
Accuracy: 0.878 [20]	Accuracy: 0.919 [20]
Accuracy: 0.880 [50]	Accuracy: 0.924 [50]
Accuracy: 0.881 [100]	Accuracy: 0.927 [100]
Accuracy: 0.881 [200]	Accuracy: 0.928 [200]

Table 17: Bagging without feature selection and with feature selection

Again, using feature selection we obtain a higher accuracy than without.

Random Forest

We try Random Forest with the following results:

Accuracy: 0.855 [1]

Accuracy: 0.868 [2]

Accuracy: 0.889 [5]

Accuracy: 0.894 [10]

Accuracy: 0.897 [20]

Accuracy: 0.899 [50]

Accuracy: 0.899 [100]

Accuracy: 0.900 [200]

The results are worse than the previously tested bagging with feature selection (took 87 mins):

Extra Trees

With **ExtraTrees** classifier (took 114 mins):

Accuracy: 0.874 [1]

Accuracy: 0.878 [2]

Accuracy: 0.893 [5]

Accuracy: 0.898 [10]

Accuracy: 0.900 [20]

Accuracy: 0.900 [50]

Accuracy: 0.902 [100]

Accuracy: 0.902 [200]

Boosting

The next meta classifier is using boosting techniques with AdaBoost, with decision trees and GradientBoosting.

AdaBoost (36 min)	AdaBoost with DT (129 min)	Gradient Boosting (69 min)
Accuracy: 0.632 [1]	Accuracy: 0.701 [1]	Accuracy: 0.674 [1]
Accuracy: 0.644 [2]	Accuracy: 0.732 [2]	Accuracy: 0.687 [2]
Accuracy: 0.668 [5]	Accuracy: 0.774 [5]	Accuracy: 0.703 [5]
Accuracy: 0.707 [10]	Accuracy: 0.813 [10]	Accuracy: 0.720 [10]
Accuracy: 0.752 [20]	Accuracy: 0.850 [20]	Accuracy: 0.732 [20]
Accuracy: 0.808 [50]	Accuracy: 0.890 [50]	Accuracy: 0.767 [50]
Accuracy: 0.847 [100]	Accuracy: 0.910 [100]	Accuracy: 0.808 [100]
Accuracy: 0.880 [200]	Accuracy: 0.912 [200]	Accuracy: 0.839 [200]

Table 18: Accuracy of each boosting algorithms

The next figure 21 shows a comparison between the results of all the Meta Methods we've already tried so far.

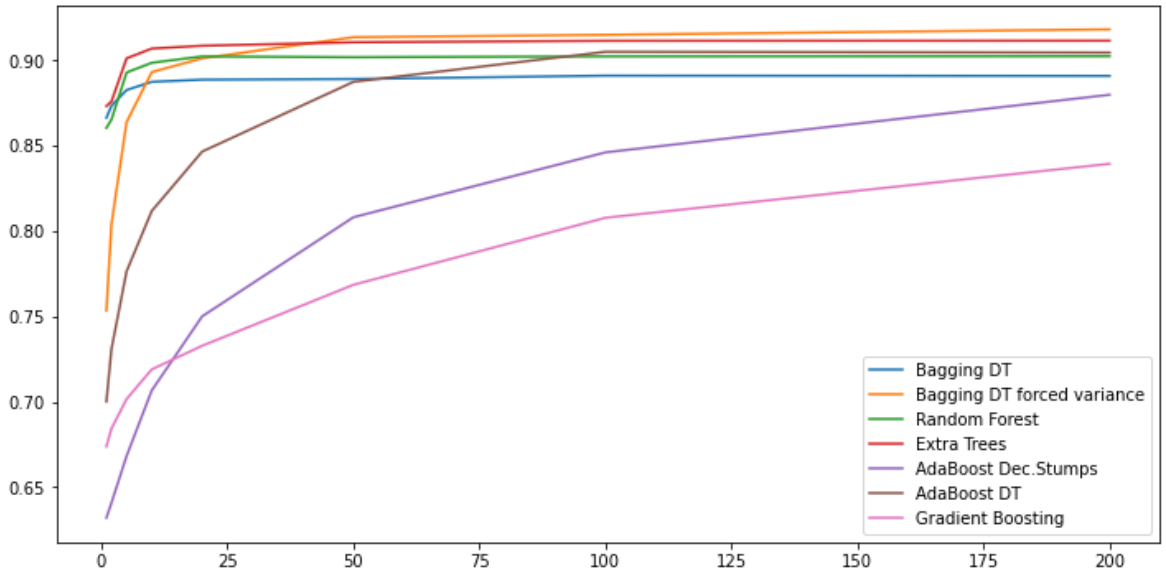


Figure 21: Comparison graph from all the meta learning methods

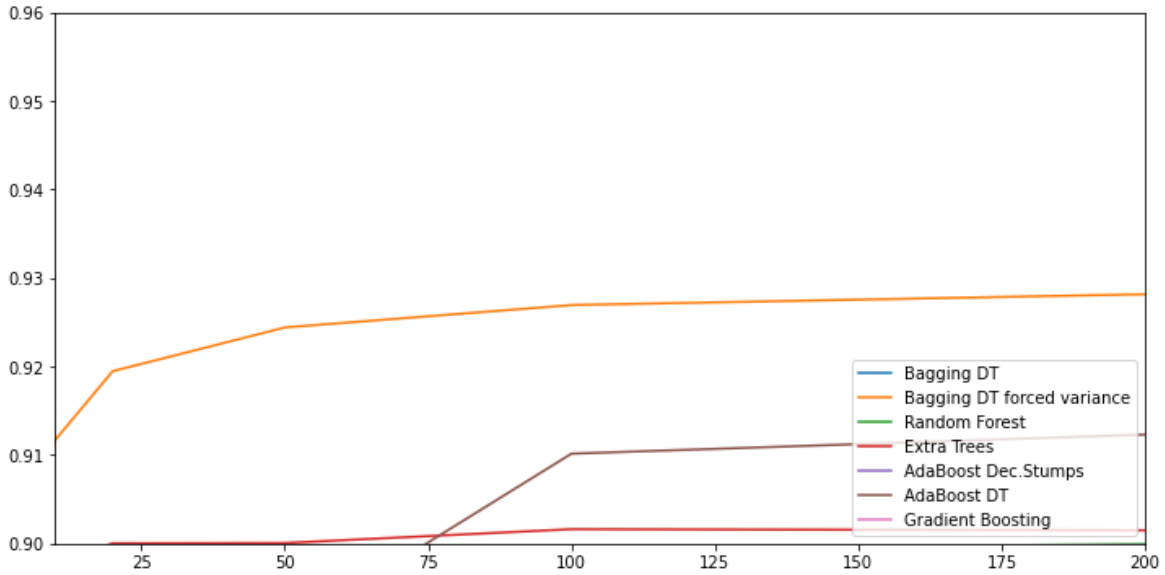


Figure 22: Zoomed comparison graph from all the meta learning methods

As we saw when using Meta Methods with TF-IDF the best method for this data is the Bagging decision trees with forced variance.

We decided not to test the Forests of Trees method because it was taking longer than a day to be processed and we expected it to be worse than Bagging DT like it was with TF-IDF.

Comparison and conclusion

Once all the techniques are executed and tuned, we can summarize the results in order to compare and extract the conclusions.

Classifier	Best selected Accuracy
Naive Bayes	0.89
K-NN	0.86
Decision Trees	0.87
Support Vector Machines	0.895
Using TF-IDF	
Voting majority / weighted	0.913 / **0.914**
Bagging	0.889 [50] / 0.891 [100]
Bagging with DT	**0.913** [50] / **0.918** [200]
Random Forest	0.902 [20]
Extra Trees	0.910 [50] / 0.911 [100]
AdaBoost	0.880 [200]
AdaBoost with DT	0.905 [100]
Gradient boosting	0.839 [200]
Feature Selection with Forests of trees	0.875
Using plain word frequencies	
Voting majority / weighted	0.906 / **0.909**
Bagging	0.880 [50] / 0.881 [100]
Bagging with DT	**0.924** [50] / **0.928** [200]
Random Forest	0.897 [20] / 0.899 [50]
Extra Trees	0.900 [20] / 0.902 [100]

AdaBoost	0.880 [200]
AdaBoost with DT	0.910 [100]
Gradient boosting	0.839 [200]

Table 19: Accuracy of each machine learning method used

We can conclude that the best classifier in this report is the **Bagging with DT** with 200 estimators both when adding TF-IDF or simple word frequency during the preprocessing. This is because it has the highest accuracy. The major issue with this method is that it consumes a lot of time to train the model, so the next in time consuming and accuracy is the **Voting weighted** that has the best accuracy and the time is faster than other meta learning methods.

One case to mention, is the **Support Vector Machine** that has a good accuracy and maybe we need to use better kernels to get better performance such as 'poly', 'sigmoid' or 'precomputed'.

Another thing to remark is that for the Naive Bayes method, although the assumption of independence is not correct, the results obtained were close to the other machine learning methods.

By generating the TfIdf measurements directly in the notebooks and subsequently splitting the data into train and test data for each machine learning method there is a risk that the comparison of the methods suffer. The "train_test_split" method splits the data into random train and test subsets. As a result, each notebook with a machine learning method used different training data. Different training data means that the models have different bases for training and comparing the accuracy scores could get misleading. This was very evident when working with the decision trees, where the trees generated were very different in structure depending on the split into training and test data. To address this, k-fold cross validation was used to be able to determine if the accuracy of a single run was comparable to that of several runs. This issue seemed to be limited to the decision trees. For instance, when running the Naive Bayes several times with different training data, the results were very similar for each run. Furthermore, since the datasets were large and randomised internally, we considered the instances that ended up in each training set for each method were representative of the dataset as a whole.

To conclude this report, there are several machine learning methods that are effective for classifying sentences into clickbait and non-clickbait. This was shown in this report by computing TfIdf scores for different headlines and subsequently training different classifiers on this data. In the following testing stage, essentially all methods that were used showed potential. The spread of accuracy ranged between 0.84 and 0.94. The two classifiers with the highest accuracies, that we considered the best, were the "Bagging with DT" and the "Voting weighted" classifiers. In a world

where the internet is full of fake-news and clickbait articles, it is reassuring to know that there are several classification techniques that work well with distinguishing these from regular news.

Referencies

[1] E. Shearer, (2021). 86% of Americans get news online from smartphone, computer or tablet [Online]. Pew Research Center. Available at: www.pewresearch.org/fact-tank/2021/01/12/more-than-eight-in-ten-americans-get-news-from-digital-devices (Accessed: 24 December 2021).

[2] GitHub - bhargaviparanjape/clickbait [Online]. Bhargaviparanjape/Clickbait. Available at: https://github.com/bhargaviparanjape/clickbait (Accessed: 18 November 2021).

[3] Abhijnan Chakraborty, Bhargavi Paranjape, Sourya Kakarla and Niloy Ganguly. "Stop Clickbait: Detecting and Preventing Clickbaits in Online News Media". In Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), San Francisco, US, August 2016.