Identifying the Subjectivity of Articles in Sports Media: A Classification Problem

Salma Khalfallah

May 13, 2025

1 Introduction

For any sort of entertainment presented for a wider audience that gets produced, commentary is invited and created every day from casual remarks such as, "Did you see the game last night?" to larger conversations and professional articles written regarding the current state of the sport at large. As productive as conversations are, it is important to bear in mind while reading articles on these topics that the nature of such discourse introduces heavy subjectivity and bias. However, it is difficult to distinguish between articles that are objective or subjective in order to gain a clearer picture of a topic. The goal of this project is to test the success of various machine learning models in classifying various sports articles as objective or subjective based on its attributes, in order to help create a clearer and more productive conversation in sports media.

1.1 Introducing Dataset and Variables

The dataset for this project is a dataset from the UC Irvine Machine Learning Repository called "Sports articles for objectivity analysis" in which there are 1000 sports articles as well as a file of all the articles numbers and their attributes. The data set includes information including non-numeric information such as Text-ID and URL, as well as numeric information including the text complexity of the article and a counter of various text attributes in the article such as number of words, pronouns, verbs, nouns, and so on. Finally, all of the 1000 articles are labeled as either subjective (articles written primarily based on personal opinions) or objective (articles written based on neutral information, such as data) articles.

2 Cleaning and Exploring Data

2.1 Data Clean-up

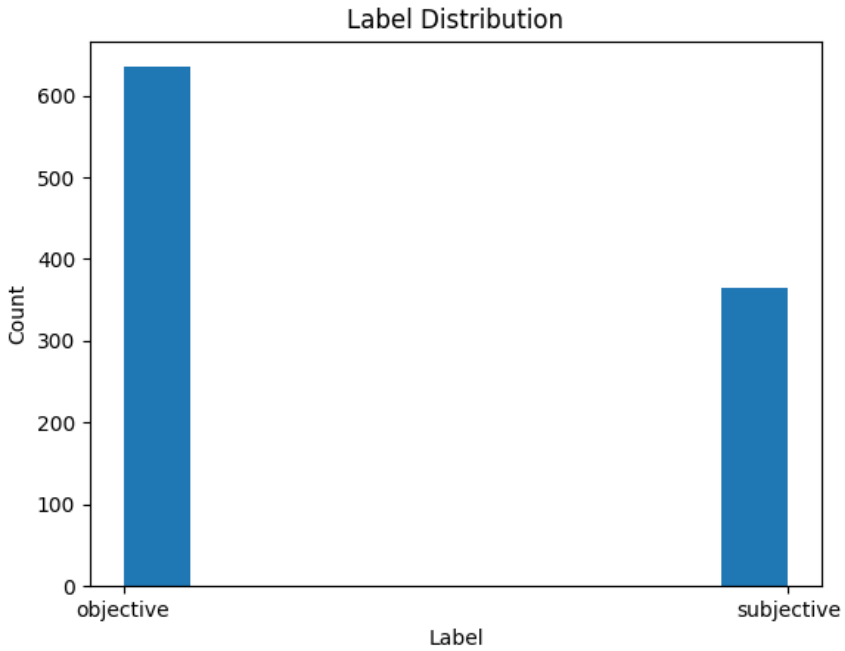
Data clean up usually consists of identifying and correct/removing errors, invalid data, and inconsistencies that may be in the original dataset. The normal first step in any data cleaning assignment is identifying missing values and correcting them if possible. Luckily, the original dataset does not contain any missing values, therefore there is no need to remove any instances in the data. I removed any unnecessary/non-numeric values from the dataset, specifically the URL and Text-ID attributes that do not contribute to any machine learning models. Finally, duplicate rows were removed to prevent inconsistency in the dataset. The final filtered dataset contained 1000 rows and 59 columns, which we will then be working on.

In order to output consistent results across all of my findings, the testing and training splits were set to a random state of 45.

2.2 Data Exploration

It is important to note, before beginning modeling, one of the key parts of the entire test: the data set is imbalanced. Out of 1000 instances, roughly 600 of the articles in the training dataset are labeled as objective articles while only 400 are labeled as subjective. This leads to modeling that potentially

favors the objective class over the subjective class. This is not ideal, since the purpose of the project is to identify subjectivity in sports articles online.



The recall metric in the evaluation stage of classification modeling comes into play here:

$$Recall = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

The recall metric of classification evaluation evaluates what percentage of the positive labels the model was able to predict correctly. This metric is significant in an imbalanced dataset, as opposed to accuracy, since it becomes increasingly more important than general accuracy for a model to identify correct labels. Due to this phenomenon, we will be paying special attention to the recall metric throughout the project.

3 K-Nearest Neighbors

3.1 Introducing Model

K-nearest neighbors is a non-parametric supervised machine learning algorithm that compares an inputted point with the k closest "neighbors" and comparing their labels to find the majority label among all of the k points. The model does all the learning solely by storing all of the training data at once. A new point is classified by finding the k training examples closest to the new data point (found through Euclidean distance) and classifying the new data point by taking a majority vote on all the closest k points. It should be noted that k should always be odd so that a majority label exists (or else a chance for a tie will occur). The choice of k *does* matter: too low of a value k would lead to overfitting of the training data, while too high of a k would potentially lead to underfitting the training data due to a smoother class boundary.

I chose to work with this model due to its simplicity as well as its popular usage as a classification algorithm. (since the output is a class membership for a specific point)

3.2 Pros/Cons of Model

As previously mentioned, learning in K-NN is very easy in that there is no learning algorithm because there is no parameter to learn respective to the dataset. This makes the algorithm computationally

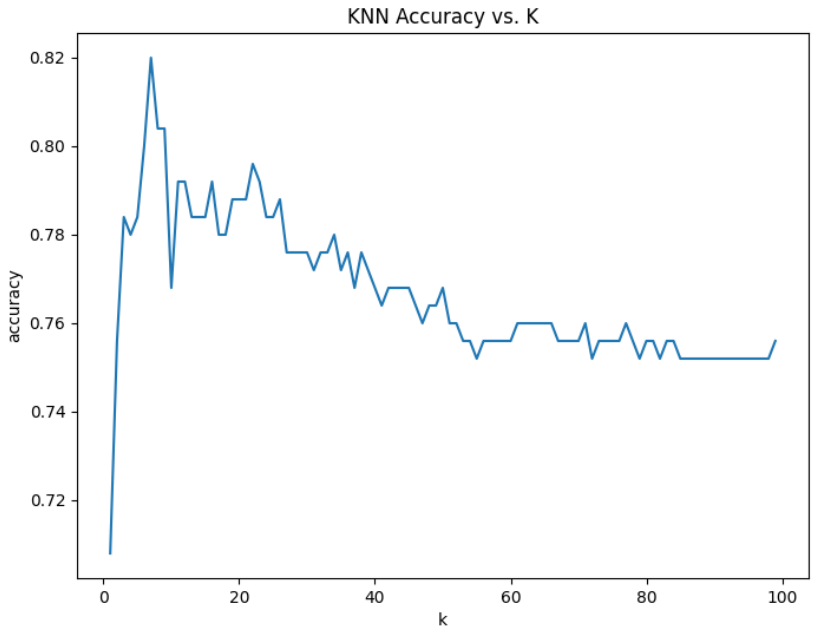
inexpensive compared to other classification algorithms. As well as this, K-NN can learn very complicated decision boundaries, since it is not following a single learning function but rather creating boundaries based on the relationship between points. This makes k-nearest neighbors straightforward to implement and potentially very effective.

On the other hand, classification in k-nearest neighbors is relatively slow. Because there is no learning algorithm, the classification is performed by calculating the distances of each point in the testing data set to every single point in the training dataset and storing the k-nearest points to the new testing point. This produces a speed of $O \cdot (nd + kn)$, where $O \cdot (n \times d)$ represents the speed of computing the distance of every point in the testing set to every point in the training set and $O \cdot (n \times k)$ represents the time it takes to loop through the training observations to find the k closest neighbors.

Another disadvantage to k nearest neighbors is related to calculations using Euclidean distance. One disadvantage is that the algorithm can be easily misled by irrelevant attributes since each attribute is weighed equally in the Euclidean distance formula, distorting true distances between points. Another way in which the Euclidean distance formula can be mislead is through the magnitude of attributes. If features are very different in scale, Euclidean distance could potentially be misleading since the distance would significantly go higher or lower based on the range of possible values in a specific attribute. The latter problem can be resolved using normalization techniques (re-centered around means, scaled by standard deviation).

3.3 Hyper-parameter Tuning: The K in K-NN

As mentioned earlier in the report, the hyper-parameter k is one of the most important parts of the model. Since there aren't that many parameters in a k-nearest neighbors algorithm, it only makes the existing parameters that more important. In order to identify the best value k for my dataset, I iterated through values from 1 to 100, and fit my dataset to a KNN algorithm using the iterator as the number of neighbors. Afterward, I stored the accuracy of the model using the iterator value as k, and if the accuracy was better than a previous fit, I would store the values of k, the score as well as the model as my best models in the algorithm. Finally, I plotted my results on a scatterplot to visualize my results.

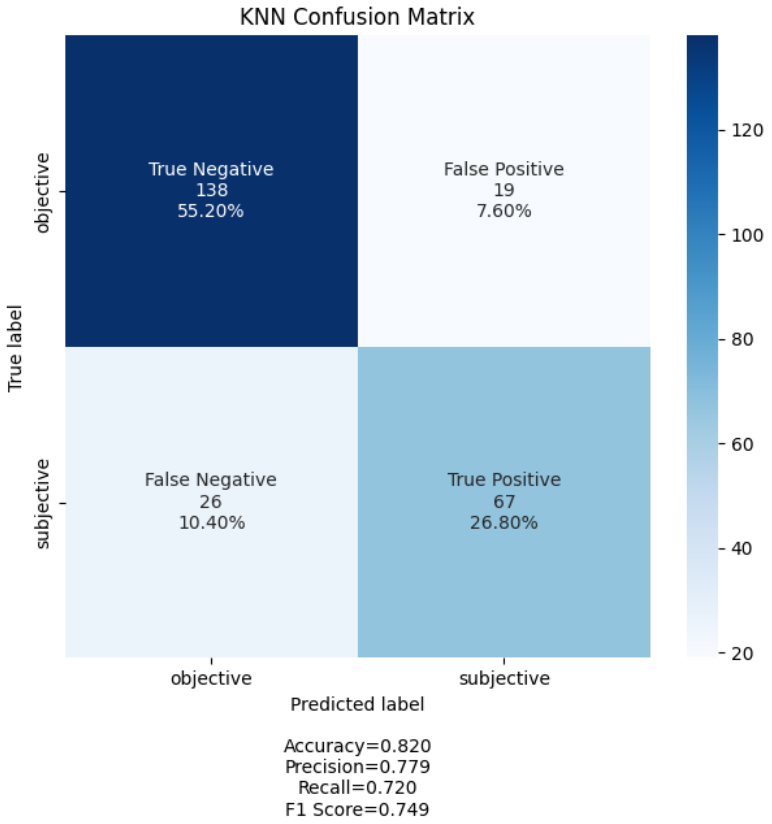


Based on the outputted results, the highest score achieved was 0.82 at $k = 7$. The model appeared to initially overfit to the data with a score of less than 0.72, reaching the peak at $k = 7$ before slowly decreasing due to the underfitting of the model, highlighting the delicate balance mentioned above.

The optimal k value for our dataset is therefore 7.

3.4 Initial Fitting and Evaluation

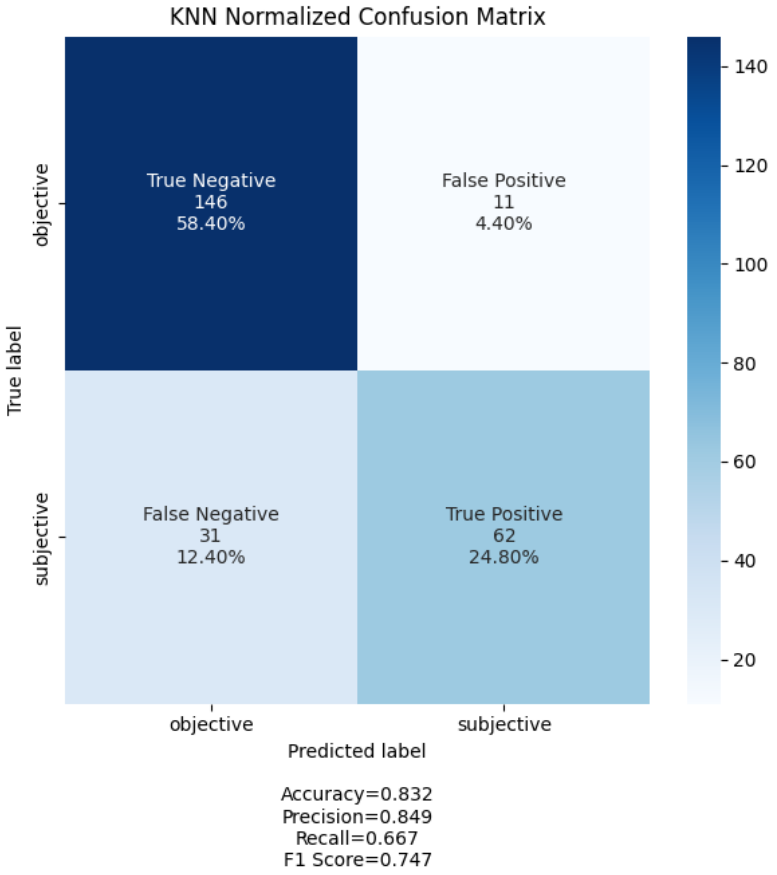
After finding the best k, the K-NN algorithm was subsequently run with the optimized hyperparameter with a training accuracy of 0.86 and a test accuracy of 0.82. To evaluate the model properly, I created a confusion matrix to assess my model. A confusion matrix is a figure which depicts the success of the model to evaluating new testing points. A label of "True Negative" or "True Positive" indicates that the model successfully predicted the correct value of a testing point, while a label of "False Negative" or "False Positive" indicates a failure of the model to correctly classify a specific testing point.



The optimized KNN algorithm reached a score of 0.82, a precision score of 0.78, a recall score of 0.72, and an F1 score of 0.75.

3.5 Normalized Fitting and Evaluation

Normalization is one of the most important parts of implementing k nearest neighbors because of problems related to Euclidean distance calculations and attribute magnitudes. By normalizing attribute values, more accurate representations of distance between points in a space can be computed to reflect true distances between elements. After investigating the accuracy of the initial model without normalization, I introduced normalization to stabilize the attributes with greater magnitudes compared to others, such as word count in an article.



After scaling the attribute values and refitting the data, the training accuracy decreased to 0.85, however the test accuracy improved to 0.83. The normalized algorithm reached a score of 0.82, a precision score of 0.85, a recall score of 0.667, and finally an F1 score of 0.747. Recalling the significance of the recall metric in our specific dataset, the fact that our recall metric decreased despite the accuracy increasing slightly suggests that the model did not benefit from normalization and should be reverted accordingly. This makes sense, since most of the attributes are scaled similarly (being discrete counts) with few attributes needing scaling in the first place.

3.6 Final Thoughts

Working with k-nearest neighbors was surprisingly powerful and efficient relative to the simplicity and efficiency of the model. With an initial model accuracy of 0.82 and an F1 score of 0.75, the model achieved solid baseline classification performance. However, challenges in normalization as well as the imbalanced nature of the dataset limited its performance and dropped the recall metric, a key metric in our analysis. Although KNN serves well as a baseline classification model, more sophisticated models may be needed to better suit the needs of our problem more effectively.

4 Decision Trees

4.1 Introducing Model

Decision trees are simple in concept: deciding on a question based on a series of questions asked regarding a set of attributes associated with a data set. When a new data point is inserted into the decision tree, the data point flows from the root in the tree until the leaf node where a final class membership is associated with a new data point. The foundation of decision trees lies in basic probability and

information gain, where the creation of the decision tree is based on how much information is gained based on partitioning the dataset across specific attributes.

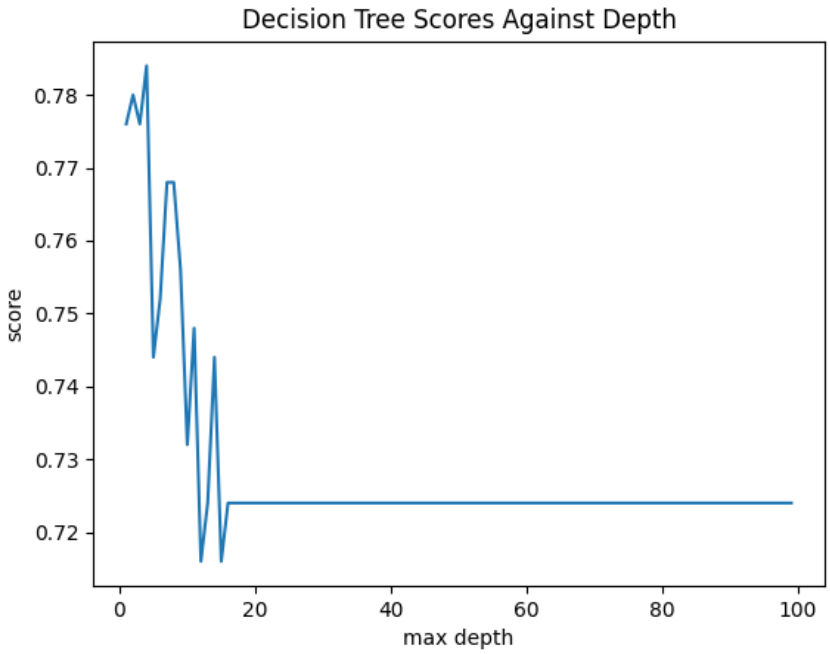
4.2 Pros/Cons of Model

Decision trees are very simple to interpret; when inputting a new point into the tree, the path of the point can be visualized very easily through graphics and basic boolean logic. Decision trees are also notably faster than k-nearest neighbors - the nature of a tree like data structure means that classification of new data points takes only $log(n)$, where n is the number of data points used to train the tree.

The nature of a decision tree also provides some disadvantages: if a maximum depth of the tree is not previously provided as a stopping point of iteration, the decision tree can create overly complex trees where every single training point maps to a unique leaf node. This kind of tree does not generalize the data well and is overfit to the training set.

4.3 Hyper-parameter Tuning: How Deep is Your Tree?

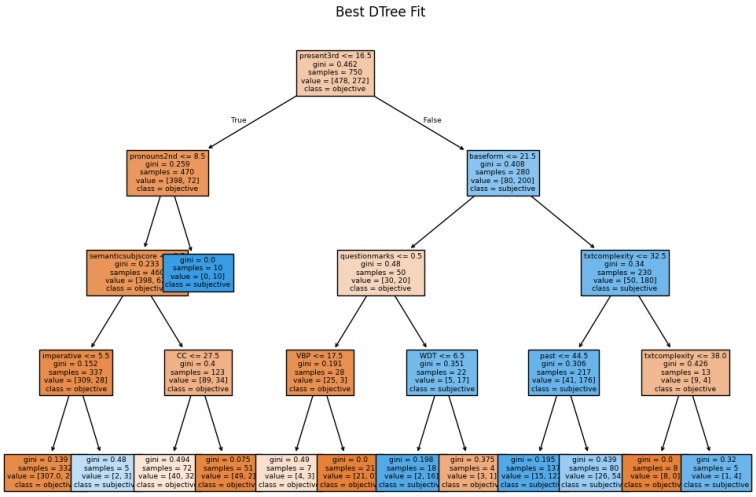
It is important to address the matter of the depth of a decision tree: how deep should the decision tree be? This is a hyper-parameter set before training the model to the data set, and can therefore be optimized to our data set. As done previously with our k in k-nearest neighbors, I iterated through a range of 1 to 100 and fit my dataset to a decision tree with maximum depth d. I saved the score of the decision tree of iteration d, and saved the iteration as my best iteration if the score of the specific iteration d outperformed my previously best score. Finally, I visualized the results of my findings.



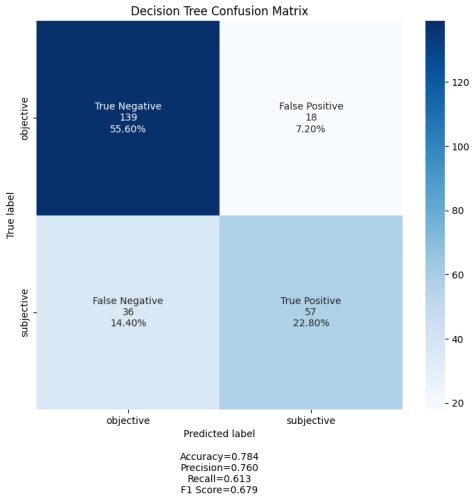
The decision tree seems to succeed in much smaller depths. The tree peaks at 0.78 at $d = 4$, before trickling down slowly as d increases until around 20, where scores flat-lines to 0.72 for the rest of the iterations of d. Therefore, the optimal decision tree length for the dataset occurs at $d = 4$.

4.4 Initial Fitting and Evaluation

After finding the optimal maximum depth for the decision tree, the data set was fitted using $d = 4$. I also visualized the decision tree:



The final training accuracy was high at 0.85, with our test accuracy at 0.78. I also plotted the confusion matrix for the decision tree as well as key evaluation statistics for the model.



The decision tree had a final precision score of 0.76, a recall score of 0.613, as well as an F1 score of 0.679. Since our recall metric is highly important for the specific dataset, a lower recall score indicates much room for improvement, which we can easily do. Finally, when looking into values to split on, one attribute stood out on in the model as being highly influential in classification of a specific article: $'present3rd'$, with a feature importance of almost 0.6. $'present3rd'$ is an attribute counting the frequency of present tense verbs with third person pronouns, for example "She studies hard." The analysis indicates that lesser counts of these in articles are more likely to be from objective articles, while greater counts are more likely to be from subjective articles.

4.5 Final Thoughts

The decision tree algorithm was very beneficial for analysis due to its simplicity, allowing easy visualization of significant features in the dataset as well as quicker computation. On the other hand, a training accuracy of 0.85 and a testing accuracy of 0.78 suggests that the model might have been overfit to the training set and failed to generalize sufficiently to new testing data. Finally, a lower recall metric of 0.613 indicates much needed room for improvements to stabilize the model.

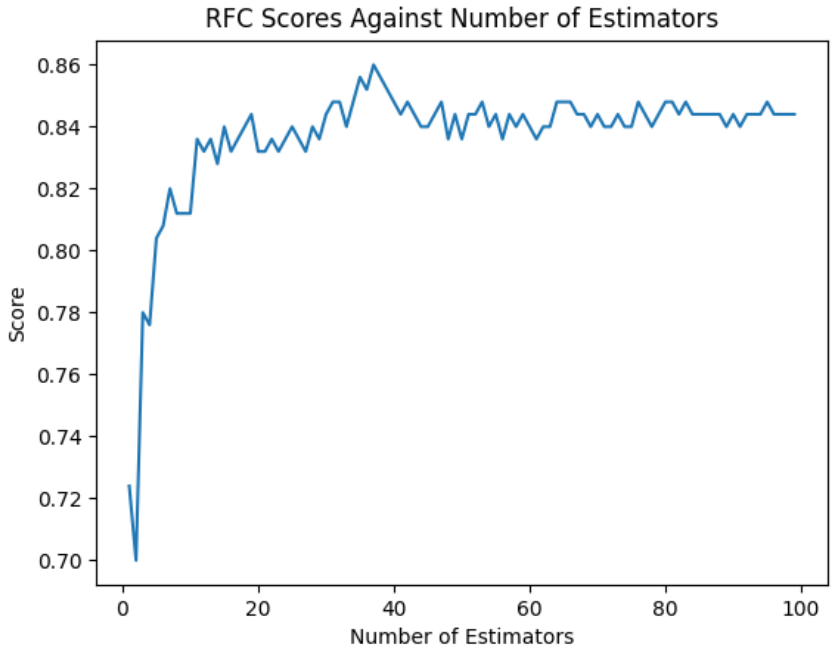
5 Random Forests

5.1 Introducing Model

A random forest algorithm is an ensemble method introduced to improve the instability of decision tree algorithms through bootstrapping a sample of the training data and splitting from a random sample of attributes rather than all attributes in the dataset. The goal of random forests is to stabilize decision tree models by combining the results of many decision trees to make a final prediction in order to reduce overfitting in models and stabilize predictions, which is needed after analyzing the results of the decision tree algorithm.

5.2 Hyper-parameter Tuning: Number of Estimators

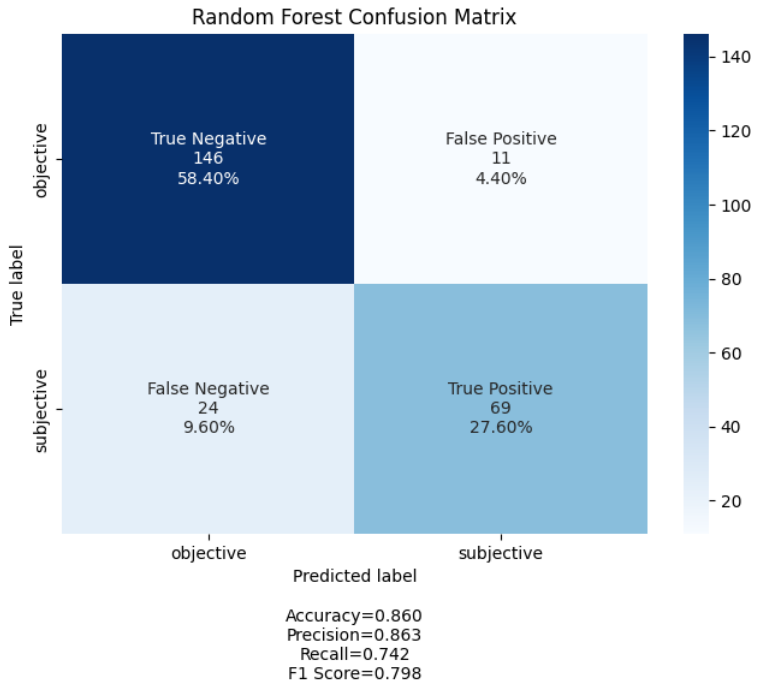
The number of decision trees used as estimators in random forests is a hyper-parameter in the model that can be tuned and optimized. As usual, the number of estimators n was tested by iterating through a range of values from 1 to 100, testing each value as the number of trees in the model to build before estimating the final decision tree in the end. The score of each iteration was saved in a list, and the best score of the iterations as well as the best n and model were stored for later visualization and use. Finally, the results of the iterations were visualized for analysis:



The best score occurred at 0.86 where $n = 37$. After hitting 37, the model stabilizes at around 0.84 for the rest of the iterations.

5.3 Fit and Evaluations

The random forest classifier was fit using the optimized hyper-parameter, number of estimators, $n = 37$. The outputting final model has a training accuracy of 1 and a testing accuracy of 0.86. As well as finding these metrics, our usual confusion matrix was made and visualized to find key evaluation metrics important to our analysis:



Our final metrics improved significantly and worked to stabilize our original decision tree model. These metrics include a precision score of 0.86, a recall score of 0.742, and an F1 score of 0.798.

5.4 Final Thoughts

The random forest algorithm worked to improve the decision tree model with an 8% increase in test accuracy and a 15% increase in training accuracy. Additionally, the recall metric improved significantly, with a 13% increase from the decision tree to the random forest algorithm. All of these metrics suggest an overall success in improving the problems presented in the decision tree model, including instability and overfitting of the decision tree to the training data. On the other hand, other problems such as a 14% difference in the training and testing accuracy suggest other improvements can be made to improve the overall performance of the dataset outside of the random forest model. Regardless, the random forest algorithm became the most successful algorithm out of the testing models thus far.

6 Naive Bayes

6.1 Introducing Model

Lastly, the Naive Bayes algorithm was evaluated. Naive Bayes uses Bayesian probability to estimate the conditional probability of classes in the dataset based on the assumed probability of attributes occurring. These probabilistic classifiers lie on the foundation that assumes that all of the attributes X are conditionally independent of one another given a specific class Y. Classification of a new point x^* is then performed by:

$$y^* = h_{NB}(x) = \text{argmax}_y p(y)p(x_1, ..., x_m|y) = \text{argmax}_y p(y) \prod_i^m p(x_i|y)$$

Naive Bayes is a simple yet powerful tool to use especially in text classification, making this potentially the ideal model to use for subjectivity analysis in text articles.

6.2 Pros/Cons of Model

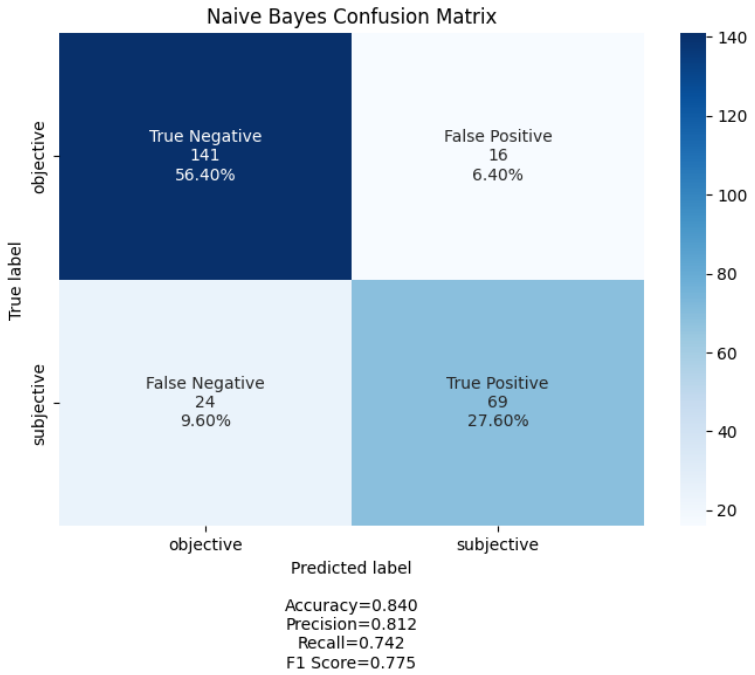
Naive Bayes is a simple algorithm to implement, and does not require that many training samples. It contains a relatively quick computation compared to other more sophisticated machine learning

algorithm thanks to the conditional independence assumption, since every attribute probability can be viewed as a one dimensional distribution problem. It is also a highly scalable classifier that can handle continuous and discrete data.

On the other hand, the conditional independence assumption for predictors rarely actually occurs, limiting applications of the model in real-life settings. Although the model oftentimes works despite the frequent violation of this assumption, it is important to use the model with precaution due to this key assumption. This algorithm also potentially fails when a training dataset has no instances for a specific variable but appears in a testing dataset. When this occurs, a smoothing technique such as using a prior is important.

6.3 Fit and Evaluations

After fitting the data to the Naive Bayes model, the final training and testing accuracy was 0.83 and 0.84 respectively. Interestingly, this was the only model that had a greater testing than training accuracy, suggesting that the model can easily generalize to new testing data.



The model had a precision score of 0.812, a recall metric of 0.742, as well as an F1 score of 0.775.

6.4 Final Thoughts

Despite very minimal tuning of the data set and no hyper-parameter adjusting, the model performed relatively well compared to the other models tested. Most notably, a greater test score than training score suggests easy generalization and a recall metric of 0.74 suggests Naive Bayes is a very powerful and quick tool to use for real time computation and analysis of the data set compared to other models. With further tuning of the model and dataset preprocessing, Naive Bayes can be a very successful tool for subjectivity analysis.

7 Conclusion

Working with K-NN, Decision Trees, Random Forests, and Naive Bayes all produced different results based on the nature and function of each model. K-NN algorithm served as a baseline classification model, providing results for comparison. Decision Trees provided a simple visualization of important features, but introduced instability. Random forests was implemented to solve that instability, but

was slightly overfit to the training data. Finally, Naive Bayes provided the best model with little optimization, but was still slightly less efficient than random forests.

Through this testing, key challenges such as hyper-parameter tuning as well as introducing effective models for language-based feature analysis. However, some issues still remained, such as problems related to an imbalanced dataset (recall the recall metric). As a continuation of this project, it is suggested that work should be done to address issues related to the imbalanced nature of the dataset to better address the needs of the problem. Finally, introducing more sophisticated machine learning models such as Neural Networks could prove to be fruitful.