Advanced Machine Learning - Term Project

Julia Sanders

March 20, 2024

Introduction

The aim of this project was to classify news articles as ”true” or ”fake”, based on the words in their titles. The news articles looked to be focused on US politics.

Preprocessing

Initially, I used the CSV files provided in Moodle as the training data. However, this did not give good accuracy in any of the tried models, and I was unable to get any models to have an accuracy of above around 78% with this data.

Firstly, I added more words output to the bag of words CSVs, by changing the count variable in the preprocessing script. Increasing the number of words made a big difference in the overall model accuracy, and the final model uses around 2000 words for its training data.

Secondly, I changed to only consider words that were at least two characters or longer (by un-commenting the line in the provided preprocessing script). Again, this gave a significant increase in the overall accuracy of the model.

Once the CSVs were ready, I merged the two csvs and added an indicator column of whether the data point was from the real or fake news. I next split the training data into a training and test set, using a stratified sampling to ensure both sets represented a similar balance between the classes. The test set used had 40% of the training data. This was used for testing and tuning the models. The exception was for the final models that were submitted to Kaggle, which were trained on the whole available dataset.

Models used

The final model was made using a blend of the three best performing models, taking a majority vote to decide on the classification of each data point. Taking a majority vote should lead to a more accurate model overall, as each model has less influence over the final outcome, so things like overfitting in one model will have less impact on the final classification.

The final model takes a majority vote between Naive Bayes, Perceptron and SVM. Initially, I also ran some experiments with other classifiers (including Logistic Regression and K-nearest neighbours) however these did not give good results, so were not included in the final model.

Naive Bayes

The Naive Bayes classifier calculates probabilities of each class using the Bayes formula on the training data and classifies the outcome based on the class with highest probability. The underlying assumption of the Naive Bayes classifier is that each feature is independent, which may not always be the case. In the data set for this project, it is likely that the features do not appear independently, since it is based on newspaper headlines and some words may appear together more frequently.

Despite this limitation, it is still a strong model overall and obtained good results on my test set, with its accuracy peaking at 88% when tested on its own in Kaggle.

Perceptron

The Perceptron classifier is a single-layered neural network, receiving input signals with a weighting and bias from the training data, and outputting a class assignment based on this. The implementation in this project uses a step transfer function to assign the classes, assigning to 1 if the output is positive, and 0 if it is negative.

A perceptron will always converge to some boundary between the two classes, however it may not be that with the largest margin. The algorithm weights and biases are found from the training data, using stochastic gradient descent to optimise them.

On its own, the perceptron scored around 92% on the unseen data set in Kaggle.

SVM

The good performance of the Perceptron indicated that it was likely for the data to be linearly separable. The SVM classifier is similar to the perceptron, in that it also searches for the hyperplane that separates the two classes. However, the hyperplane found by SVM maximises the distance between the two classes. SVM is effective in this case because a lot of features were used in the training, however it can be overfitted, since it mainly relies on finding the boundaries using support vectors (the vectors closest to the decision boundary) so it is very sensitive to any outliers in the training data.

The Linear SVM scored 93% on its own in the Kaggle test.

Feature Selection and Extraction

Feature selection helps to reduce the size of the input data, speeding up the training process, and helps to prevent overfitting the model. The goal is to reduce the dimensionality of the data so that only the features which make the most significant contribution to the outcome are considered.

Due to the size of the data set, I tested the feature selection initially using only the Naive Bayes model. I tried using two separate methods, using PCA and SelectKBest method. The features are all on the same scale, since they are word counts, but in some experimentations I normalised them to be all between 0 and 1.

The graphs below show the impact of the number of features used and the accuracy of the model. Although PCA showed some improvement at around 80 features, this did not seem to make a noticeable impact when applied to the overall final classifier during experimentation, so given the additional time taken to run the code, I decided to not use feature selection in the final model.

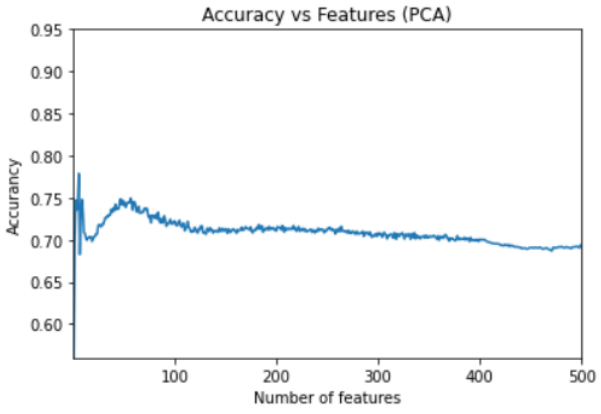

Figure 1: Feature Selection using PCA with NaiveBayes

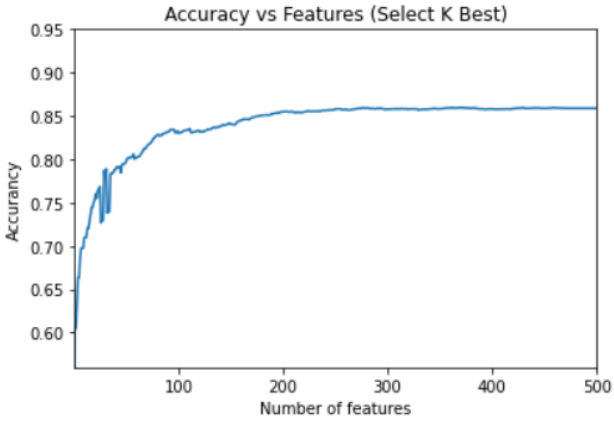

Figure 2: Feature Selection using k-best chi2 score with NaiveBayes

Chi2 using Select K Best

This method of feature selection carries out a chi-squared statistical test, which determines how dependent the value of a feature is on its class. The features which are least dependent (or independent) of the class can then be removed from the model. The selection was implemented using the SelectKBest function in the Python sklearn library, which scores each feature in the data set and keeps only k-best to train the model.

Principal Component Analysis

PCA is a feature extraction technique, since the original features are replaced. This is an unsupervised method, where the empirical covariance matrix of the data is calculated, and the eigenvectors and eigenvalues are found. The full data is then projected onto the found eigenbasis. Since the empirical covariance matrix is used, this means that the new data is influenced most heavily by the features which explain the highest variance.

Boosting Model Performance with Adaboost

In order to improve the performance of my models, I boosted the weak learner classifiers using Adaboost. The Adaboost algorithm takes the output of a weak learner and boosts the results by recursively adjusting the weights used in the algorithm to create an ensemble model.

The hyperparameters (number of components and learning rate) for Adaboost were found using a random grid search and the accuracies of the parameters were measured using 5-fold cross-validation on the train and test set. Due to the size of the data set, I initially tried a wide range of values and updated the values to search nearer to those which had a good performance.

Overall, the accuracy of all three algorithms was improved when using them with Adaboost. For example, SVM on its own scored 89%, and 93% after boosting.

Conclusion

The best performing individual model was the Adaboost with the Perceptron classifier. Due to the high performance of both the Perceptron and SVM models, it seems that the data is linearly separable.

Although the performance of each individual boosted model was good, blending the models added around 0.5% to the overall accuracy. Since SVM and perceptron algorithms are quite similar, I think that the Naive Bayes acted as a 'tie-breaker' in situations where the two did not agree, and helped with classifying those points closest to the decision surfaces.

I think that it would be unlikely that this classification model would be generalisable to practical classification tasks. This is because the algorithm is trained on data on the specific subject matter (in this case US politics), so while it may work well for this kind of data, it is unlikely that the same model could not be used on headlines about different topics. My model's final best accuracy from Kaggle fluctuated around 94.5% showing that the classifier was fooled around 5% of the time.

All the code for the project, including final model and experimentation, can be found here, where the graphs and heatmaps produced by the individual models, as well as various other experimentation are also included. My Kaggle username is julia-sand.