NLP FINLE TITAL

Colin Pillsbury
Swarthmore College
Department of Computer Science
cpillsb1@swarthmore.edu

Tai Warner
Swarthmore College
Department of Computer Science
twarner2@swarthmore.edu

Abstract

We compare a few classifiers' performance on the dataset, including XGBoost, which has gained a reputation for itself on Kaggle. We perform an experiment to measure the performance of the classifiers, and another to more qualitatively understand the nuances of how XGBoost makes its decisions. We find that a certain tuning of XGBoost is the best classifier in our suite, and that it performs best because it pays attention to publisher indicators in the formatting of the article.

1 Introduction

Fake news is a growing phenomenon on the internet. Ever since the 2016 election, when platforms like FaceBook were used to disseminate unverified information in order to influence voters, the fear of fabricated falsehoods clever enough to look true has occupied the minds of most users of the internet. While the first fake news articles were completely fabricated stories conjecturing about objectionable personal associations the Clintons had, the phrase has nowadays been more or less diluted to mean publications released with the intention of informing the public in a biased way. Organizations like MSNBC and Breitbart publish more of this type of biased news because they align with a certain political party and want to both push the ideals of that party and cater to its members. By being aware of the source of news, and looking in the writing style for clues that it is attempting to trigger politically divisive human psychological faculties, readers can be better informed about the quality of the information they trust.

Our goal is to achieve the highest performance at the hyperpartisan news classification task that we can by using the XGBoost classifier and different perspectives on the data. We assess this against our Lab 8 baseline performance with the MultinomialNB classifier. We also use a K-Nearest-Neighbors classifier to put the XGBoost performance in more context.

2 Related Work

2.1 Text Categorization with Many Redundant Features

Previous work in a similar area as ours includes (Gabrilovich and Markovitch, 2004), titled "Text Categorization with Many Redundant Features", which focuses on the feasibility of feature selection in text categorization contexts. Specifically, the paper looks at how in problems with redundant features, decision tree algorithms like C4.5 can outperform SVMs, but with significant feature selection, SVMs can beat C4.5 by a slight margin. As context, the paper cites several earlier studies that suggest that the majority of features in a bag of words model are relevant for classification, and that SVMs perform best with no additional feature selection. Using data from Open Directory Project, they construct classification problems where there are significant amounts of redundant features. They give the example of the task of predicting whether an article is talking about Boulder, Colorado or Dallas, Texas. Here, there are obviously proper nouns such as place names or local businesses that would be very helpful for distinguishing the two, but many of the other features would not be helpful.

The study looks at a variety of feature selection techniques (Information Gain, Document Frequency, Chi-square, etc.) and finds that while C4.5 performs better when neither algorithms are performing feature selection, SVMs (with a linear kernel) performed better than C4.5 after doing feature selection.

This paper is similar to our topic in that it focuses on feature reduction for bag of words models, but their work seems to be more focused on a specific subset of text classification problems (tasks with lots of redundant features), which our task does not seem to align completely with.

2.2 XGBoost

(Chen and Guestrin, 2016), titled "XGBoost: A Scalable Tree Boosting System" details a supervised learning algorithm that has had a profound impact on the field of machine learning. It describes a novel algorithm, XGBoost, that at a high level is a highly optimized form a gradient tree boosting. Similar to other boosting algorithms such as AdaBoost, XGBoost trains an ensemble of decision trees. However, unlike regular gradient tree boosting, XGBoost has certain optimizations to prevent overfitting, and to also allow it to be more parallelizable.

XGBoost has had a lot of success in Kaggle contests over the past few years, frequently appearing in the winning ensembles of classifiers. Because of its high performance and adaptability, XGBoost has also been used in many NLP classification tasks. It is relatively easy to plug in XGBoost in place of whatever previous classifier was being used, perform some hyperparameter tuning, and get significant results. XGBoost is able to handle high dimensionality data like bag-of-words features by performing feature selection, allowing it to ignore less impactful features.

3 Methodology

We compared the performance of a few different types of classifiers against each other. We used the data as is, and then used a single classifier to compare performance on different datasets. Finally, we used the best classifier from the first experiment on the different datasets from the second experiment to see how it performed.

The data that we used was made up of one million news articles, labeled hyperpartisan or not. Importantly, the fact that it had been labeled algorithmically rather than by hand meant that it was a silver-standard dataset, and that the real meaning of our prediction was correlated with what news source an article was published by, rather than its true bias. The text was cleaned and tokenized with `spacy`, for the main dataset. The clean text from that dataset was lemmatized to make a second dataset with only stems. A third dataset was created from only the parts of speech of each word from the main dataset.

3.1 Comparing Classifiers

For the first experiment, we compared Multinomial Naive Bayes, `KNeighborsClassifier`, and XGBoost against one another. Actually, we ran a sub-experiment in order to find the best parameters for XGBoost. Then, we ran programs to train on one hundred thousand articles' full (spacy-fied) text and test on twenty thousand, and to train and test on the entire dataset. We ran one smaller program and one larger program for each classifier, for a total of six runs in this experiment. We compared the accuracy, precision, recall, and f-measure of each run.

3.1.1 Tuning XGBoost

The XGBoost classification algorithm has a *lot* of different parameters. We didn't have time to try tweaking every one, and in the literature that we could find, only a handful of the parameters were focused on in tuning. We decided to tune four parameters: `max_depth` between 3, 6, and 12, which limits the maximum number of decisions that one of the trees in the classifier can take; `eta` between 0.01, 0.1, and 1, which is the learning rate, a multiplier to scale the weight the model puts on a misclassification of a given instance in a given iteration; `gamma` between 0.1, 1, and 10, which helps the model decide whether the split from introducing another decision is likely to truly help minimize entropy, or just lead to overfitting; and `colsample_bytree` between 0.1, 0.5, and 1, which is the percentage of features to be randomly picked for each tree in the ensemble to be able to choose features to decide on. We used `GridSearchCV` to run all combinations of these parameters. We found that the best XGBoost classifier was the one with `max_depth` = 12, `eta` = 0.01, `gamma` = 0.1, and `colsample_bytree` = 1.

3.1.2 Important Features

We did another subexperiment, with the fact in mind that the XGBoost module we used allows us to see the features that the classifier deemed the most important in differentiating categories. We wanted to understand the classifier better by looking at what it pays attention to. XGBoost can conceive of important features in terms of a few measures, among them weight and gain. Weight is how many trees that feature is used in, and gain

Model Evaluation	Accuracy	Precision	Recall	F1 Score
XGBoost	0.611	0.573	0.872	0.691
Multinomial Naive Bayes	0.610	0.572	0.871	0.691
K Nearest Neighbors	0.577	0.557	0.753	0.640

Table 1: *Accuracy, precision, recall, and F1 score for the three models when predicting on the validation set*

Feature Set	Accuracy	Precision	Recall	F1 Score
Full Text	0.610	0.572	0.871	0.691
Tags	0.469	0.472	0.525	0.497
Lemmas	0.608	0.570	0.879	0.692

Table 2: *Accuracy, precision, recall, and F1 score for Multinomial Naive Bayes when trained and tested on the three dataset representations*

is the average decrease in entropy that splitting on that feature gives. We measured and plotted the top ten words for both measures.

3.2 Comparing Datasets

For the second experiment, we used the MultinomialNB classifier and tested its performance on the full text, the lemmatized text, and the POS tags of the text. We expected that the full text would do best, but thought the lemmatization and pure POS tags would add some information that the full text missed. For instance, the lemmatized text groups conjugations together, which probably do make sense to represent as one meaning. The POS tags offer a syntactic angle for this data, and our hypothesis was that some tag strings like, for example, `JJ JJ NN`, would be more common in hyperpartisan news since the author is trying to push their point of view and attach associations to the subject matter.

4 Results

4.1 Model Performance

For one of our experiments, we compare different types of classifiers against one another with our `MultinomialNB` classifier from Lab 8 as a baseline. We use K-Nearest Neighbors, XGBoost, and Multinomial Naive Bayes. Table 1 shows the resulting accuracy, precision, recall and F1-scores when training on the full text training set and testing on the validation set. XGBoost had slightly higher accuracy, precision, and recall than MultinomialNB, but not significantly high enough to make strong conclusions. Both of these models had higher scores than K-Nearest Neighbors across the board.

The three confusion matrices show the number of TP, FP, FN, and TN's that each model had on the full validation set. The X-axis is the model's predicted label (Hyperpartisan on left, Non-hyperpartisian on right) and the Y-axis is the true label (Hyperpartisan on top, Non-hyperpartisan on bottom). As we see, each of the models predicted positive (Hyperpartisan) much more frequently than they predicted negative (Non-hyperpartisan). We will analyze these results more in the discussion section.

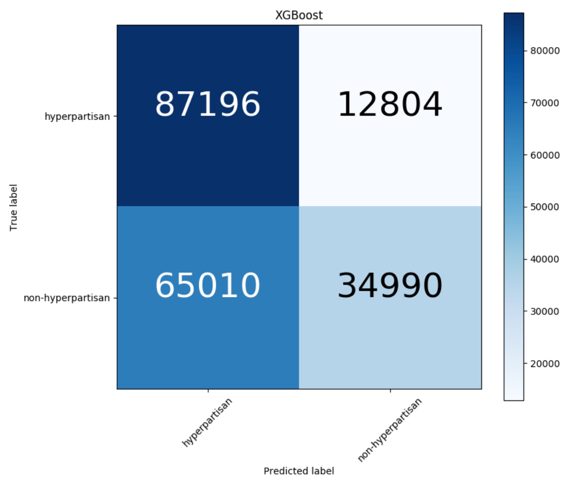

Figure 1: Confusion Matrix for XGBoost on validation set

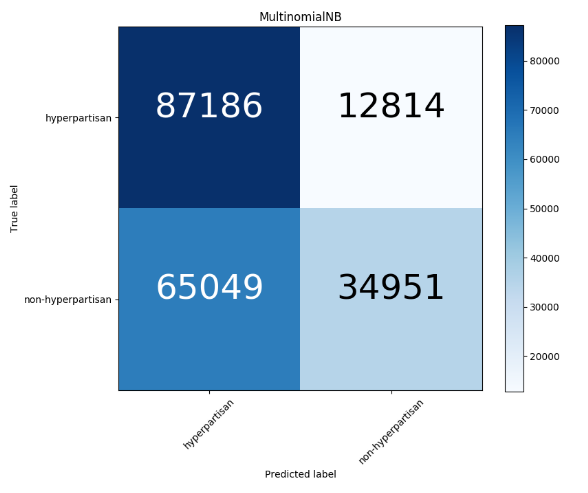

Figure 2: Confusion Matrix for Multinomial Naive Bayes on validation set

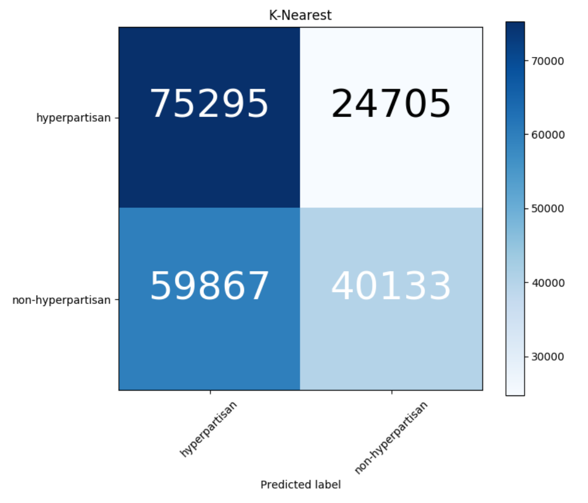

Figure 3: Confusion Matrix for K-Nearest Neighbors on validation set

4.1.1 Important Features

The most important features for the XGBoost classifier are shown below ranked by weight, with the most important features ranked by gain following.

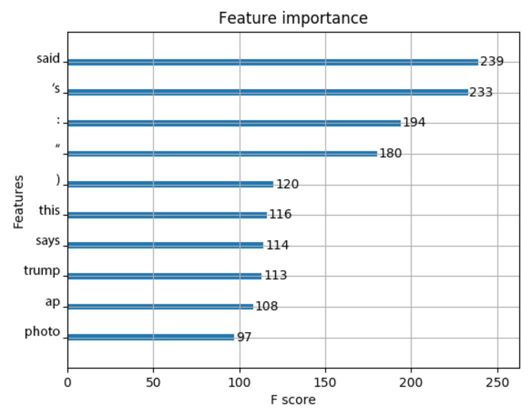

Figure 4: Most important features in XGBoost by weight. The number to the right of each bar is the number of trees the feature was used in.

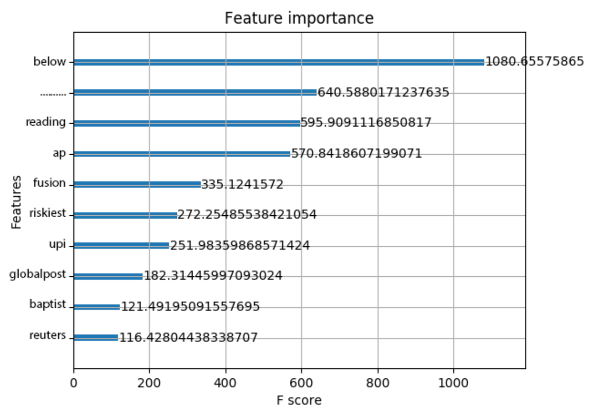

Figure 5: Most important features in XGBoost by gain. The number to the right of the bar is the factor that feature contributes to the model in cutting down the entropy of a sample.

4.2 Comparing Datasets

Table 2 shows the resulting accuracy, precision, recall, and F1-scores when running Multinomial Naive Bayes on the full text training set and testing on the validation set. Unsurprisingly, our model did not perform well using just bag-of-tags as features. It is interesting that our model somehow performed worse than random guessing or Most Frequent Class. We will discuss this odd scenario more in the Discussion section.

The table also shows that using the full text yielded approximately the same results as using just lemmas for features. Full Text has a slightly higher accuracy (61.0% v.s. 60.8%), but has a slightly lower recall (87.1% v.s. 87.9%). The lemma features yielded the highest F1 score, just slightly above the F1 score for the full text features (69.2% v.s. 69.1%). It seems that using bag-of-lemmas is very similar to just using bag-of-words, and does not provide any significant benefit or drawback.

5 Discussion

5.1 Model Performance

We found that K-Nearest Neighbors had the lowest accuracy of the models that we tested, while also having the longest runtime (roughly one day) by a significant amount. This is likely due to the fact that K-Nearest must compute distances in very high dimensions (30,000 features) and also must recompute distances between every point each time it predicts for a given instance. It seems that the combination of the bag-of-word representation and having a very large training set caused K-Nearest to be particularly ill-suited for our task.

We were somewhat disappointed to find that XGBoost did not significantly outperform Multinomial Naive Bayes, producing nearly identical scores across the board.

When looking at the confusion matrices for the three models, it is apparent that each of the models were much more inclined to predict Hyperpartisan. This tendency could be caused by a variety of factors, but because all of the models have this trend, one possible explanation could be that the validation set is not representative of the training set. Another explanation is simply that our models overfit to the training set, leading to skewed performance on the validation set. Either way, this behavior led to our models having relatively high recall scores, with relatively low precision scores. In other words, our models were able to correctly predict a lot of the TP's, but our models also had a lot of FP's.

5.2 XGBoost

5.2.1 Tuning

Out of the parameters we tuned, we found that the best performance was achieved with `max_depth` = 12, `eta` = 0.01, `gamma` = 0.1 and `colsample_bytree` = 1. Deep trees could be a concern because they could lead to overfitting, but using twelve features out of 30 thousand is probably small enough that this wouldn't happen. Eta is related to the step size that the gradient ascent part of XGBoost takes, and the smaller the step, the more precise the algorithm can be and the less likely it is to overshoot, so it makes sense that our smallest value would be the best one. Gamma has to do with the expected improvement necessary to make a new decision node from a leaf in a tree, but since Figure 5 shows gain values around 1000, we are not sure how 0.1 relates to that. As for the percentage of features available for each tree to use to split, it seems that with so many features, a lower percentage would usually introduce more stochasticity which would probably increase performance. However, it may be that there is a sudden dropoff in the informativity of the features, in which case random subsampling could lead to the important features being unavailable to trees.

5.2.2 Important Features

Upon inspection, most of the most discriminative words used by the model are ones used in the typography from certain news sources. For example, presumably because of the formatting on a typical computer screen, articles from Fox business contain the phrase "Continue reading below". Some important words, like "reuters" are obvious enough — many news articles begin with something like "Washington, D.C. (Reuters) —" prefacing the article. It is exciting that the model picked up on these specific phrases tied to publishers, because that is ultimately what we are predicting. Since the dataset is not a gold standard, it was labeled according to what news source the article is from, which seems to be exactly what XGBoost is trying to figure out.

5.3 Comparing Datasets

The dataset comparison experiments did not end up yielding very interesting results. As mentioned in the results, the bag-of-tags representation yielded a surprisingly low accuracy that was below 50%. This means that our model actually learned negative correlation relative to our validation set. If we simply predicted the opposite of what our model suggested, we would perform better. We interpreted this result as a sign that there is no general correlation between the unordered counts of tags and whether or not an article is hyperpartisan.

Although we did not expect the lemmatized features to result in a significant change from the full text features, we did not expect to have nearly identical results for the two. We had thought that lemmatizing might have some impact, because things like verb tense or pluralization do not seem like they would be indicative of hyperpartisanship.

6 References

References

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Evgeniy Gabrilovich and Shaul Markovitch. 2004. Text categorization with many redundant features: using aggressive feature selection to make svms competitive with c4.5. In *ICML*.