Abstract

In these times of substantial job loss due people will need to find new and better jobs. With many people using the internet to search and apply for jobs, people will need to know whether or not these job postings can be trusted. We look at a sample of nearly 18,000 job postings to determine whether we can predict, and to what degree of confidence, whether or not a job posting is fraudulent or legitimate. Our methods will involve categorical and text analysis of the information contained within these job postings.

Research Question

1. Using previous analysis from the Kaggle website, can we find any significant improvements to the ability to detect fraudulent job postings?
2. Is there a combination of text and non-text data that we can use to increase our chances of finding a fraudulent job posting?

Intro

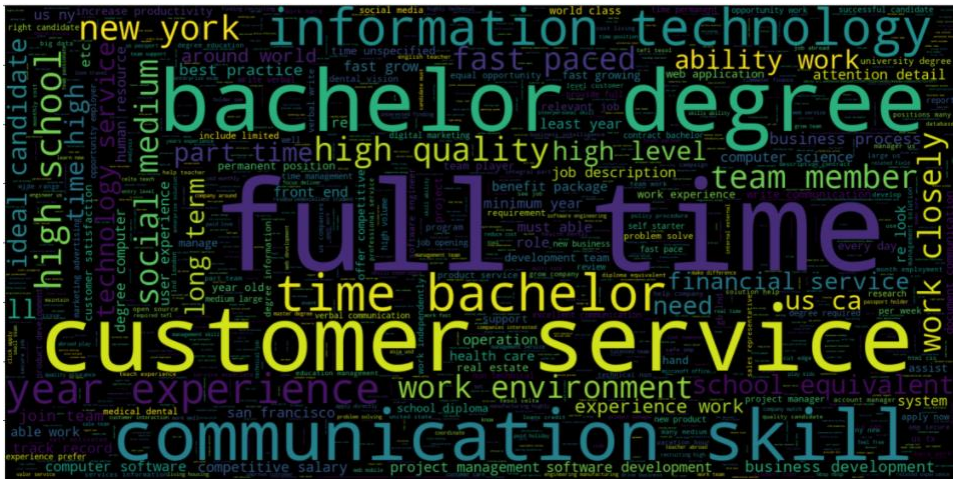
In this project, we will be looking at a Kaggle dataset from https://www.kaggle.com/shivamb/real-or-fake-fake-jobposting-prediction. We will be attempting to improve the success rate of picking a fraudulent job posting. Although analysis from myself and others on Kaggle have yielded high results, because of the unequal distribution of fraudulent to legitimate job postings, the accuracy rating is unreliable. We seek to find a way to combine textual analysis and classification of other aspects of the data so that we may increase the chances that we detect a fraudulent job posting. We will discuss what has previously been performed on the Kaggle website, move into the details of my own preparation of data, and finally move into the results of my predictions. Once we have completed our discussion of what has already been accomplished by others, we will direct our focus to the preparation of the data set that we will analyze. This will consist of the data preprocessing, cleaning and visualizations. Finally, we will discuss what features will progress through to the machine learning algorithms.

Literature Review

On the Kaggle website, there have been over 50 different people that have submitted analysis of the same dataset. The top analysis contains 68 upvotes and 42 comments, the next highest contains 37 upvotes and 14 comments, and the third highest contains 22 upvotes and 13 comments. We will primarily be discussing the two highest rated individuals, but we also have special highlights for others who have yielded interesting results or pieces of code that we will also utilize in our analysis.

The highest rated Kaggle analysis has been done by user Madz2000 and is titled "Text Classification using Keras/NB (97% Accuracy)" (https://www.kaggle.com/madz2000/text-classification-using-keras-nb-97-accuracy). The user states that the dataset is highly imbalanced, which is correct, but does not appear to do anything to address this imbalance. This shows in the final results of his analysis. While there is a 97% accuracy achieved, when using Naïve Bayes the user is not able to attain greater than a 50% correct rating and in one instance produces zero correct predictions. This is a trend that we noticed in our analysis and seek to remedy through improvement in our algorithms. Using the neural network features, provided by Keras, Madz2000 is able again to achieve the 50% correct classification of fraudulent job postings. It is primarily due to Madz2000 work that I would like to see if combining information will result in more accurate predictions for fraudulent job postings, given that we have a severely imbalanced dataset.

One of the great things about Madz2000's analysis, was that he was able to provide great code to make a great word cloud. The word cloud below highlights the words most commonly used in fraudulent job postings.



These word clouds showed some interesting trends in the words used for a legitimate job posting as compared with a fraudulent job posting. Overall Madz2000's analysis provided interesting insight into the problem, although it seems as though the question was not fully answered. There is not yet a definitive method to detect a fraudulent job posting with a high degree of accuracy.

The next highest rated posting on Kaggle was done by Vikas Singh and is titled "Fake Job Post Prediction: Countvec, GloVe, Bert" (https://www.kaggle.com/vikassingh1996/fake-job-post-prediction-countvec-glove-bert). Singh points out the imbalanced dataset again, but only points to looking at precision, recall and F1 score in order to determine accuracy. As with our analysis to come, Singh provides very helpful information as to details within the text of each job posting. Sing establishes that fake posts have less characters in the company profile as opposed to real job postings. The number of characters were compared to each text field, but only the company profile showed any significant difference in its results. Singh uses a stratified k Fold in order to do his intial predictions and compares these to a Glove and Bert predictions. Overall the stratified K Fold produces the highest results with an Area Under the ROC Curve (AUC) score of 0.85. Singh then uses a 2 layer sequential neural network on GloVe Features, which were configured for 200D, and was able to produce an AUC score of 0.81. Singh's final algorithm took a sample of the dataset and combined logistic regression and Bert. The result of this was an AUC score of 0.77, but this is mostly attributed to the datasize being reduced to 2000 samples, instead of the full dataset.

For other notebooks that were present on Kaggle, a few others did not reach the top voted datasets, but I feel should have sections of their code mentioned. Sagar Jha in his post titled "Job Genuine or Note: Random Forest [0.93 F1 CV] utilized the SMOTETomek algorithm in order upsample the data since it is highly unbalanced. While currently I am uncertain as to whether this was an accurate choice to make, I am interested to see how this algorithm will also affect my data analysis.

Another user that provided an interesting chart was Shivam Burnwal in his post titled "NLP(98%acc.) EDA with model using Spacy & Pipeline". In order to provide a way to visualize missing data, they utilized missingno. This created a display grid showing gray bars where there is data and white sections that contains null data. It is a very good quick look to see which features have large sections of null values as compared to other features.

Data Preprocessing and Cleaning

Now let's move into the data preprocessing and cleaning of the data that I performed on my own. When looking at the original dataset, we had 17,880 entries with 18 features. The features were as follows:

1. job_id
2. telecommuting
3. title
4. has_company_logo
5. location
6. has_questions
7. department
8. employment_type
9. salary_range
10. required_experience
11. company_profile
12. required_education
13. description
14. industry
15. requirements
16. function
17. benefits
18. fradulent

One of the features was the flag of whether or not the job was fraudulent. We needed to maintain this row as this is the feature that we will base our analysis results off of. Although it will need to be split from the dataset before we can begin analysis of our machine learning algorithms. Another feature was the job_id field. We were able to remove the job_id in the beginning, as a unique identifier for each job posting would not be helpful to us during our analysis.

The next step in our cleaning was to search for and drop all of the duplicate records in our dataset. Luckily for us, pandas has a built in drop_duplicates function, so we utilized this feature. Since we have already removed the job_id feature, which guarantees that each row is unique, we were able to run the function easily. We were able to remove 281 duplicate records from the dataset.

The next part of our cleaning dealt with null values. However, since we are dealing with job postings, null values can actually represent a job poster not inputting information. So rather than deleting all the null values, we could use this as information the points us towards the effort a company put into their job posting. I am making the assumption here, that an actual job posting

is more likely to contain all of the information fields since having more information would provide a job seeker with better choices for their job applications. So instead of dropping any null values or filling them with some data, we will instead notate whether or not the field has information. This allows us to use the text data fields, without yet having to process the text data. After completing this function, we added eight additional columns that showed whether or not the original column had data. Then we were able to delete the original columns since we did not want information duplicated in our dataset. Although we don't use the text data here, we will utilize it later on in our analysis.

The next step in our cleaning is to deal with the categorical data that we have. Since there isn't a true rating that we can provide to categories such as employment_type, required_experience, required_education and function, we decided to use a pandas feature called get_dummies. This function converts categorical variable into dummy/indicator values (https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html). The downside to using this function is that we went from 18 features to 74 features. However, now many of these features are filled with only zeros and ones, which should make data analysis fairly quick.

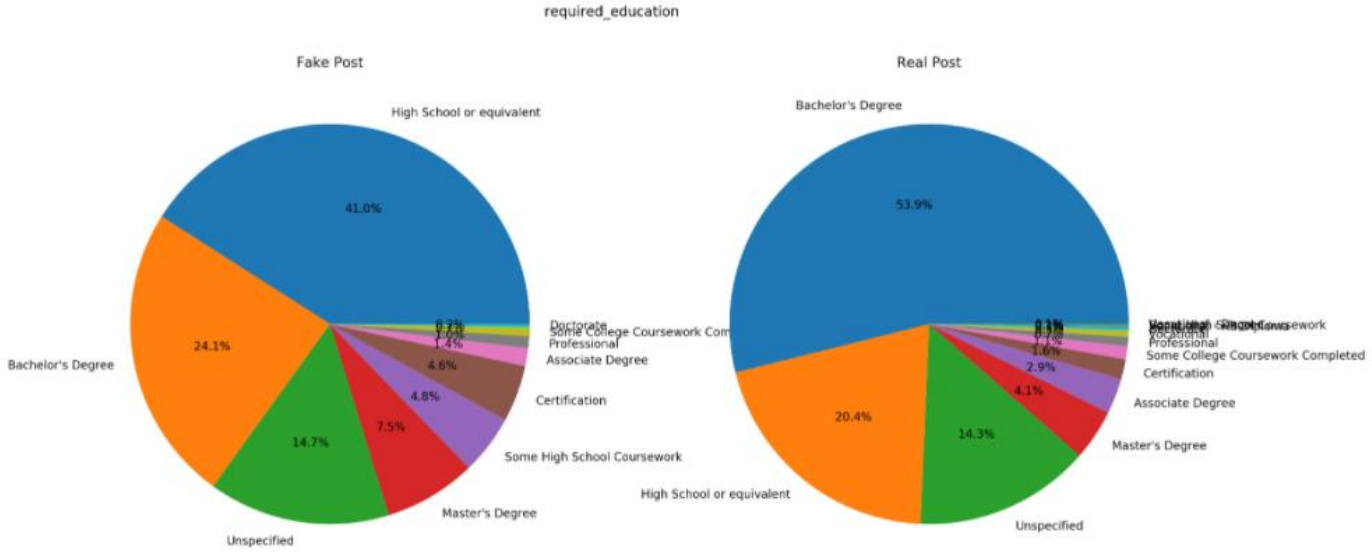
We have now cleaned our non-text data and we can begin to use this for analysis. Our dataset still contains information that could help to improve our accuracy if we utilize the text data though. We will now go through with cleaning our text data so that we can get that cleaned to a point of use in our analysis.

The first thing we did to clean our text data was to remove all the columns that were not text data. These included job_id, salary_range, telecommuting, has_company_logo and has_questions. This left us with only the columns with text values. Next, we needed to handle the null values. Again, rather than deleting these values, we decided this time to fill those values with another value. The value that we decided on was a blank space. This is so that we did not lose any data by throwing it out, but a space would not be counted in our textual analysis. After this, we combined each text column into a single column titled full_text. Once we verified that the text was successfully combined and stored, we deleted every other column so that we did not have duplicated information. The next step was to lemmatize the text so that we could group different inflected forms of a word as a single term. This process has so far taken up the longest time in the data processing, but once it was completed we were able to have a string of text

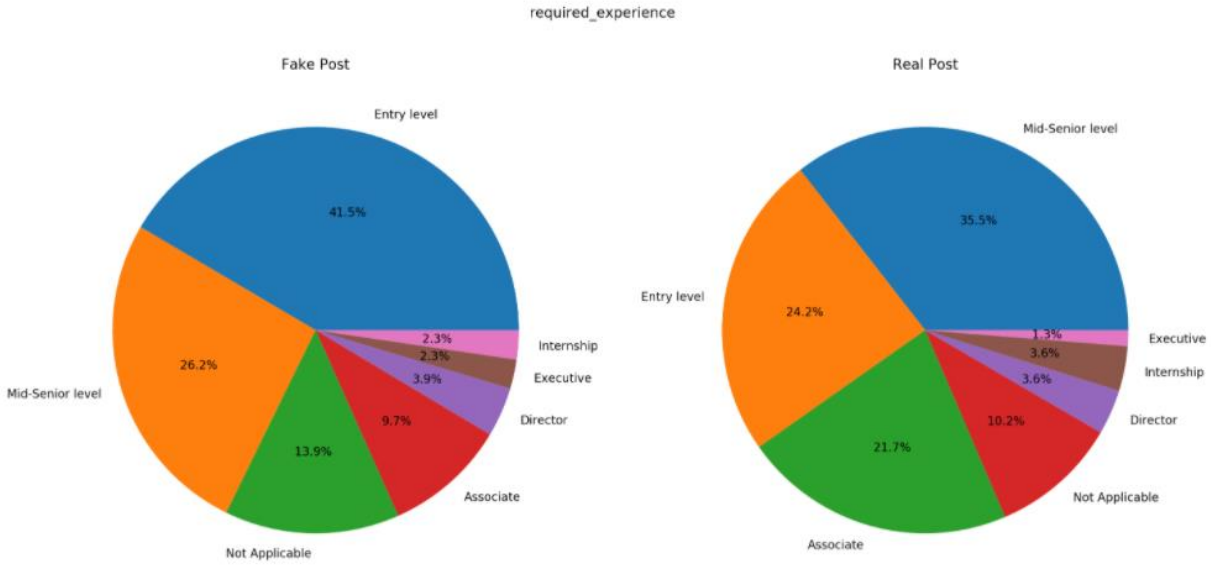
characters for each job entry that we could perform further analysis on. In order to not have to perform this again, due to time constrains, we saved this dataset off to the side so that we can utilize it again when needed later in our processes.

Visualization

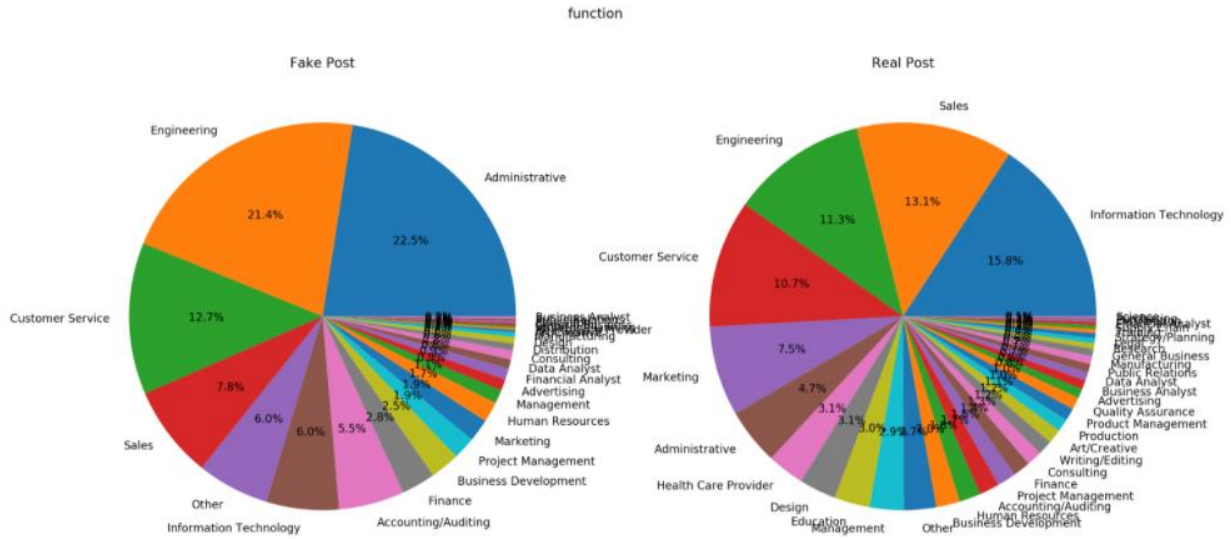
Now that our data here is cleaned up, let's look at some interesting visuals that were detailed through our analysis. We found it interesting the differences in features between the fraudulent job postings and legitimate job postings. The below pie charts are based on totaling the amount of values in either the fraudulent or the real postings. This is so that the larger number of real posts don't overpower the smaller number of fraudulent job postings. One of the first things that I noticed was the difference in required education between a fake and a real job posting. While both types of job postings have the same top three values, high school or equivalent is more prevalent for a fake post, as opposed to a bachelor's degree for real job postings.



The next features that produced some interesting visuals were for required_experience. A fake post is more likely to be listed as entry level as opposed to a real job posting requesting mid-senior level.



The feature called function provided the most surprising results. The fake post's second most likely function is engineering. It's surprising to me that engineering would be requested so highly in the fake job posts. Perhaps they are targeting entry level engineers who are trying to get their foot in the door, but that information is beyond the scope of what we are analyzing.



Feature Selection

We had an interesting time attempting feature selection with this dataset. When performing the SelectKBest function on anything less than the entire dataset, we noticed a decrease in the accuracy of the machine learning algorithms, particularly with our test algorithm of the Random Forest Classifier. We believe this is due to each feature in the processed dataset being a modification from the original dataset field. Each field now becomes important vs the original data. Although, we do feel that more research is needed to establish if this trend holds true for every machine learning algorithm or if this was an anomaly with our short analysis of feature selection. Because of the decrease in accuracy with selecting only certain features, we feel that better results will be provided, if we keep each feature. Thus, our feature selection is to select all of the features. This also applies to our text analysis; we will be using all of the lemmatized text that we can in order to do our analysis. There are other processing that may need to be done for our text analysis, but that will be dependent upon which machine learning algorithms that we decide to run in our final steps.

K-Means Algorithm

The K-Means Algorithm provided the lowest results of any of the algorithms that we attempted. We believe this is because this algorithm is better suited towards datasets where data clusters into spherical shapes and this dataset did not meet that requirement. Because of the high dimensionality of our dataset, we won't be able to produce clear graphs of what the data would look like in our model. Below is the total of values that we received from our analysis for the separated datasets. The highest values in each column has been highlighted. As a note, all times measured are a combined model training and predicting. Speeds were fast enough that splitting them did not make sense in many of the algorithms.

Algorithm and ROC	Value	Algorithm and Acc	Value		Time (s)
kmeans_default_roc	0.4989	kmeans_default_acc	18.18	kmeans_default_acc	1.77
kmeans_default_roc_text1000	0.4989	kmeans_default_acc_text1000	14.49	kmeans_default_acc_text1000	6.63
kmeans_default_roc_text200	0.4536	kmeans_default_acc_text200	14.49	kmeans_default_acc_text200	1.93
kmeans_default_roc_text2000	0.4989	kmeans_default_acc_text2000	14.49	kmeans_default_acc_text2000	19.7
kmeans_default_roc_combined	0.4544	kmeans_default_acc_combined	14.49	kmeans_default_acc_combined	10.3

In the case of K-Means, we never achieve over 19% accuracy or even .50 ROC. Even with the combined data we do not increase ROC, Acc and we even take longer than the previously higher values when the data was separated.

Decision Tree Classifier

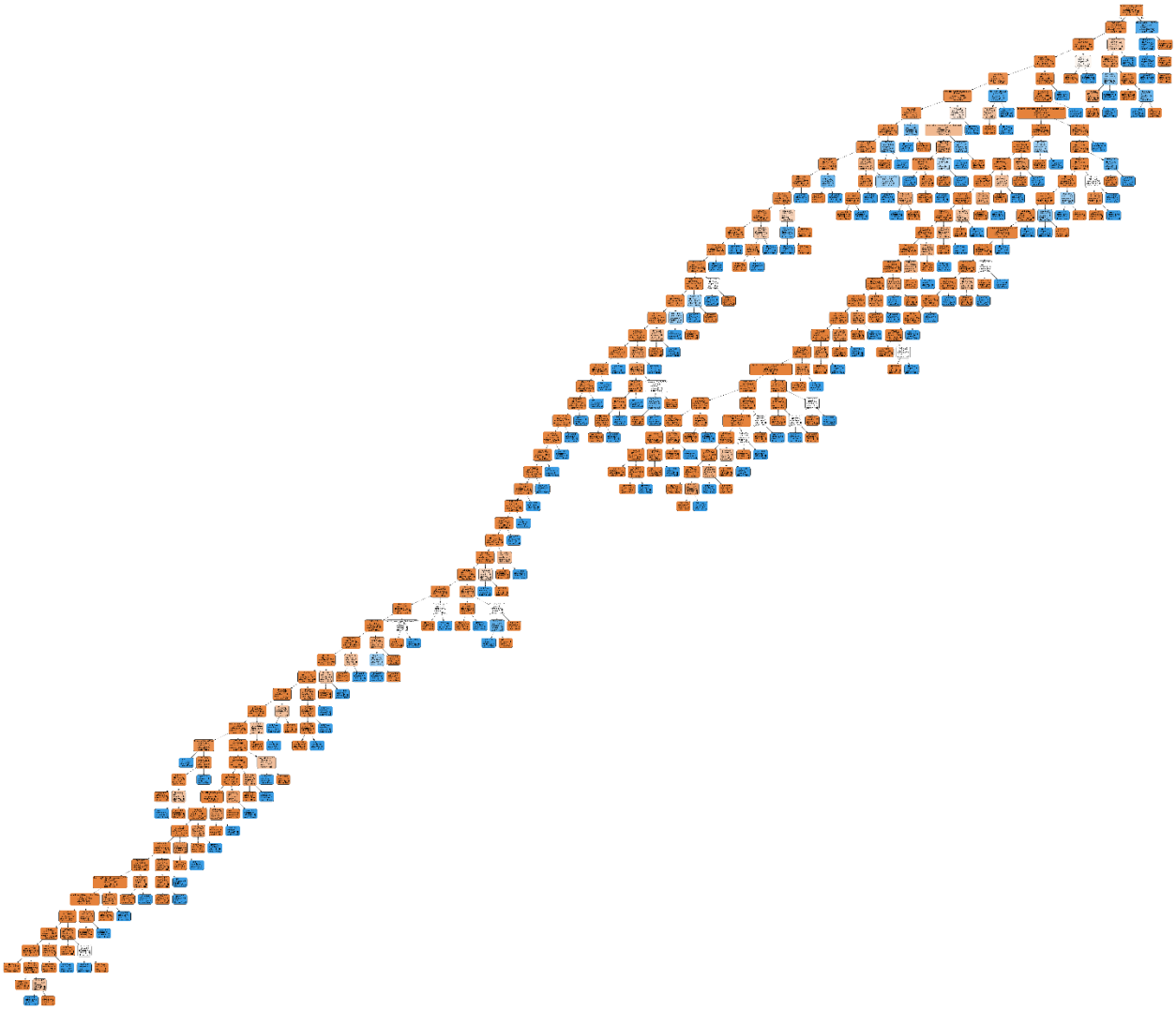
The Decision Tree Classifier provided some of the highest results, in the fastest amount of time.

Algorithm and ROC	Value	Algorithm and Acc	Value		Time (s)
dtc_default_roc	0.7254	dtc_default_acc	96.18	dtc_default_acc	0.094
dtc_default_roc_text1000	0.8788	dtc_default_acc_text1000	97.48	dtc_default_acc_text1000	4.54
dtc_default_roc_text200	0.8471	dtc_default_acc_text200	96.66	dtc_default_acc_text200	0.733
dtc_default_roc_text2000	0.8745	dtc_default_acc_text2000	97.57	dtc_default_acc_text2000	7.41
dtc_default_roc_combined	0.8758	dtc_default_acc_combined	97.83	dtc_default_acc_combined	3.57

The 1,000 word vectored produced the highest ROC with .8788, the combined 1,000 word vectored and non-text data produced the highest accuracy of 97.83%, while only slightly behind the highest ROC with .8758. The fastest time was performed by the non-text data in .094 seconds, but produced both the lowest accuracy and ROC.

The combined data would arguably be the best process to run since it produces the highest accuracy. It also only produces an ROC that is .002 behind the highest value. On top of that, the time it takes for the combined data to run is less than the 1,000 word vectored data by almost a full second, or 21% faster.

Here is what the tree looks like for the combined dataset when outputted. The blue squares indicate fraudulent postings and the redder squares indicate legitimate postings. Our tree is not well balanced and hyper parameter tuning could help us to address this problem, while also making the algorithm take longer to run.



Nearest Neighbors

The Nearest Neighbors classifier did not produce any remarkable results when compared to the other algorithms that were utilized.

Algorithm and ROC	Value	Algorithm and Acc	Value		Time (s)
neigh_default_roc	0.7265	neigh_default_acc	96.09	neigh_default_acc	1.93
neigh_default_roc_text1000	0.7606	neigh_default_acc_text1000	97.06	neigh_default_acc_text1000	3.92
neigh_default_roc_text200	0.7575	neigh_default_acc_text200	97.08	neigh_default_acc_text200	1.07
neigh_default_roc_text2000	0.7588	neigh_default_acc_text2000	97.01	neigh_default_acc_text2000	7.61

For nearest neighbors, the combined dataset showed a significant increase in the amount of time to run without any improvements in the ROC or the accuracy. Because of the large dimensionality of our data we won't be able to produce any meaningful image of the algorithm.

Naïve Bayes

Since we used the Gaussian, Multinomial and Bernoulli Naïve Bayes algorithms. Since they are so similar, we will discuss them all together here and compare them to each other.

Algorithm and ROC	Value	Algorithm and Acc	Value		Time (s)
bnb_roc	0.7072	bnb_acc	93.94	bnb_acc	0.031
bnb_roc_text1000	0.8379	bnb_acc_text1000	83.84	bnb_acc_text1000	0.396
bnb_roc_text200	0.7347	bnb_acc_text200	77.39	bnb_acc_text200	0.078
bnb_roc_text2000	0.8715	bnb_acc_text2000	85.65	bnb_acc_text2000	0.893
gnb_roc	0.5345	gnb_acc	14.35	gnb_acc	0.048
gnb_roc_text1000	0.8103	gnb_acc_text1000	69.69	gnb_acc_text1000	0.286
gnb_roc_text200	0.7756	gnb_acc_text200	65.52	gnb_acc_text200	0.079
gnb_roc_text2000	0.779	gnb_acc_text2000	63.41	gnb_acc_text2000	0.605
mnb_roc	0.5407	mnb_acc	95.37	mnb_acc	0.03
mnb_roc_text1000	0.7428	mnb_acc_text1000	75.55	mnb_acc_text1000	0.094
mnb_roc_text200	0.6788	mnb_acc_text200	69.19	mnb_acc_text200	0.03
mnb_roc_text2000	0.7791	mnb_acc_text2000	78.78	mnb_acc_text2000	0.207
bnb_roc_combined	0.8474	bnb_acc_combined	84.42	bnb_acc_combined	0.462
mnb_roc_combined	0.7449	mnb_acc_combined	76.25	mnb_acc_combined	0.151
gnb_roc_combined	0.8128	gnb_acc_combined	70.15	gnb_acc_combined	0.445

Overall, compared to the other algorithms, Naïve Bayes ran the fastest, which is to be expected. Not a single test that we performed ran for longer than a second, with the closest being .893 seconds. Although we demonstrated great speeds, we did not fare so well on accuracy and ROC. The highest ROC was .8715 and was achieved with the Bernoulli Naïve Bayes. The Multinomial Naïve Bayes produced the highest accuracy of the group with 95.37%. Multinomial Naïve Bayes also produced the fastest times at .03 seconds.

Random Forest

Random Forest does very similar to the Decision Tree Classifier, which is to be expected.

Algorithm and ROC	Value	Algorithm and Acc	Value		Time (s)
rfor_default_roc	0.7184	rfor_default_acc	96.69	rfor_default_acc	0.527
rfor_default_roc_text1000	0.8153	rfor_default_acc_text1000	98.26	rfor_default_acc_text1000	1.26
rfor_default_roc_text200	0.8049	rfor_default_acc_text200	98.13	rfor_default_acc_text200	0.651
rfor_default_roc_text2000	0.8051	rfor_default_acc_text2000	98.16	rfor_default_acc_text2000	1.84
rfor_default_roc_combined	0.8153	rfor_default_acc_combined	98.26	rfor_default_acc_combined	1.28

The combined data did equally well on the 1,000 word vector, but this could be attributed to the random order that the random forest had to go through. In non-recorded tests the numbers would swap places depending on the beginning seed. There did not seem to be as good of an increase with using the combined data in this case.

Gradient Boosting Classifier

The Gradient Boosting Classifier took the longest amount of time when compared to the other algorithms. Since there was not any significant increase in ROC or accuracy, this would not be a helpful algorithm to continue to use through testing.

Algorithm and ROC	Value	Algorithm and Acc	Value		Time (s)
gbc_default_roc	0.6173	gbc_default_acc	96.15	gbc_default_acc	1.34
gbc_default_roc_text1000	0.8232	gbc_default_acc_text1000	97.73	gbc_default_acc_text1000	25
gbc_default_roc_text200	0.7674	gbc_default_acc_text200	97.73	gbc_default_acc_text200	8.5
gbc_default_roc_text2000	0.8229	gbc_default_acc_text2000	98.18	gbc_default_acc_text2000	40.7
gbc_default_roc_combined	0.8347	gbc_default_acc_combined	98.27	gbc_default_acc_combined	26.9

We do see an improvement in ROC and accuracy by using the combined data. Our initial hypothesis that we could improve results by combining text and non-text data is looking more and more like it is true.

Separated Data

We split the data into two initial sets of information into a set that dealt with the text and then the non-text data. Once we completed the analysis of these two datasets on the different algorithms, we combined them to see if we could detect an increase in accuracy. The top five values sorted by accuracy are below:

Algorithm and ROC	Value	Algorithm and Acc	Value		Time (s)
rfor_default_roc_text1000	0.8153	rfor_default_acc_text1000	98.26	rfor_default_acc_text1000	1.26
gbc_default_roc_text2000	0.8229	gbc_default_acc_text2000	98.18	gbc_default_acc_text2000	40.7
rfor_default_roc_text2000	0.8051	rfor_default_acc_text2000	98.16	rfor_default_acc_text2000	1.84
rfor_default_roc_text200	0.8049	rfor_default_acc_text200	98.13	rfor_default_acc_text200	0.651
gbc_default_roc_text1000	0.8232	gbc_default_acc_text1000	97.73	gbc_default_acc_text1000	25

As we can tell from the table, the random forest using the text analysis with 1,000 words vectored produces the highest accuracy of 98.26%, while only taking 1.26 seconds to complete. However, this accuracy does not produce the highest ROC value, but only 0.8153. When the data is sorted to have the highest ROC we get the following table:

Algorithm and ROC	Value	Algorithm and Acc	Value		Time (s)
dtc_default_roc_text1000	0.8788	dtc_default_acc_text1000	97.48	dtc_default_acc_text1000	4.54
dtc_default_roc_text2000	0.8745	dtc_default_acc_text2000	97.57	dtc_default_acc_text2000	7.41
bnb_roc_text2000	0.8715	bnb_acc_text2000	85.65	bnb_acc_text2000	0.893
dtc_default_roc_text200	0.8471	dtc_default_acc_text200	96.66	dtc_default_acc_text200	0.733
bnb_roc_text1000	0.8379	bnb_acc_text1000	83.84	bnb_acc_text1000	0.396

We can see that now the Decision Tree Classifier for the 1,000 vectored words produces the highest ROC of .8788 with an accuracy of 97.48% and taking 4.54 seconds. Between the two we see a 7.78% increase in ROC, a .79% decrease in accuracy and a 260% increase in time to run.

With all of the datasets used and algorithms used, we achieved an average ROC of .7264, an average accuracy of 77.82% and average time of 4.37 seconds. The averages of the ROC are heavily affected by values under .50 while the accuracies have several values that are below 20%. Time is normally below 5 seconds, but we can go upwards of 20 seconds and even 40 seconds in one case.

Combined Data

With the text and non-text data combined, in general, we were able to improve our results, but only marginally. Because the 1,000 word vectored option appeared to produce the best results, on average, we only performed the combined data calculations on the 10,000 word vectored dataset. The highest ROC was achieved by the Decision Tree Classifier with results of .8758 ROC, 97.83% accuracy and taking 3.57 seconds. The highest accuracy was achieved by Gradient Boosting Classifier which produced an accuracy of 98.27%, an ROC of .8347 and took 26.9 seconds. The fastest time was achieved by the Multinomial Naïve Bayes which ran in .151 seconds, had an ROC of .7449 and an accuracy of 76.25%.

In comparison to what others have done on this topic, we did not achieve significantly better results as was expected. Since there was a decrease on the average amount of time for the algorithms to run and an increase in both average accuracy and ROC I would suggest that combining the data is the better alternative. More work would be required in order to fine tune each algorithm to potentially produce even better results. Now, let's now look into how each individual algorithm performed.

Conclusion

In regards to our first research question, we were unable to find improvements as compared to other users on Kaggle who have performed analysis on this same dataset. However, our methods did not look to find the best algorithm or to tune any of our parameters. It would take further study to see if perhaps we could tune our hyper parameters in a way that could provide us with better results.

As for our second research question, we were able to increase our ability to detect a fraudulent job posting by combing the text and non-text data. Our average ROC increased 5.75% with the combined data. Our average accuracy increased 2.66% with our combined data. Our average time to run increased 35.68% (from 4.3 seconds to 5.9 seconds), which is to be expected as we significantly increased the amount of data that had to be processed.