Julianna Cybrynski

Tiara Mathur

Jeffrey Mei

Final Paper

PROJECT OVERVIEW

In current times, the most popular and efficient way to search for jobs is through online postings. As many of us have searched for internships, part-time jobs, and eventually full-time jobs, it is imperative that the job postings we apply to are legitimate. Not only would applying to a fake job waste time and energy, but it would also grant scammers access to our personal information.

Our dataset consists of eighteen columns of both categorical and numerical variables describing job posts, with over eighteen-thousand rows of observations. The dataset provides valuable information surrounding a job posting's location, requirements, field of study, salary, and key job descriptions that we will use to classify whether or not a job is real.

Our goal is to create classification algorithms to determine a job posting's legitimacy. We will use multiple methods learned in class, such as the Naïve Bayes classifier, K-Nearest Neighbors, PCA, Logistic Regression, LDA, QDA, SVMs, boosting, bagging, and decision trees, as well as sentiment analysis which we studied outside of class. We will analyze these different approaches and ultimately determine a model, or models, that have a successful classification rate, all while reducing type one and two errors.

DATA CLEANING AND PREPARATION

Data Cleaning

Before creating models, graphs, and further analysis, the data required extensive cleaning.

During the initial cleaning, redundant columns, such as department, industry, and function, were reduced to only one of these repetitive columns. In addition, Salary Range was parsed into three new columns consisting of low range salary, mean salary, and high range salary, so that the string value initially provided could be transformed into a numeric value. The focus of this project was also on job postings within the United States, so international job postings were filtered out.

There was quite a bit of textual data that needed to be more generalized. From our dataset, the four main columns of textual data were Company Profile, Description, Requirements and Benefits. The data cleaning for textual data involved removing stop words such as "the", "is", "and" and any other words deemed unimportant. Contractions were separated into two words and non-ASCII characters were converted into their corresponding ASCII character. Finally, word stemming and word lemmatization were incorporated to reduce each word to its simplest form. Examples of this include converting a word from its plural form to its singular form or converting a word from past tense to present tense.

Preparation

Additional numeric variables were derived from the textual data. These new features were the sentiment analysis score and the length of text for each observation's textual feature. In total, eight new features were added, with two for each textual feature (Company Profile, Description, Requirements and Benefits).

The sentiment analysis score was derived by using the textBlob library in Python. It parses through the words and averages out the sentiment score based on the "severity" score of words. For example, the word "bad" might have a severity score of -1 whereas the word "horrible" might have a severity score of -3. An average sentiment score below 0 would indicate that the text feature has a negative connotation overall, a score between 0 to 0.05 indicates the text feature has a neutral connotation, and anything above 0.05 indicates the text feature has a positive connotation.

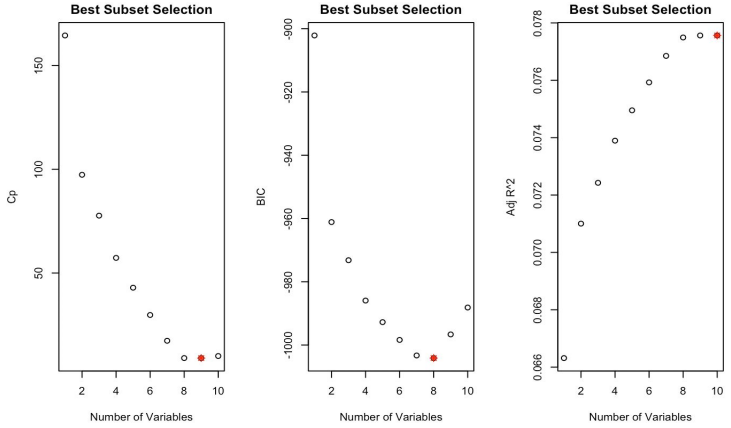
The length of each text feature was derived by just counting the number of words in the text feature. This was done after the stop words were already removed. We hypothesized that fake job postings would have very sparse textual data so this was our rationale for including the length of text as a new feature.

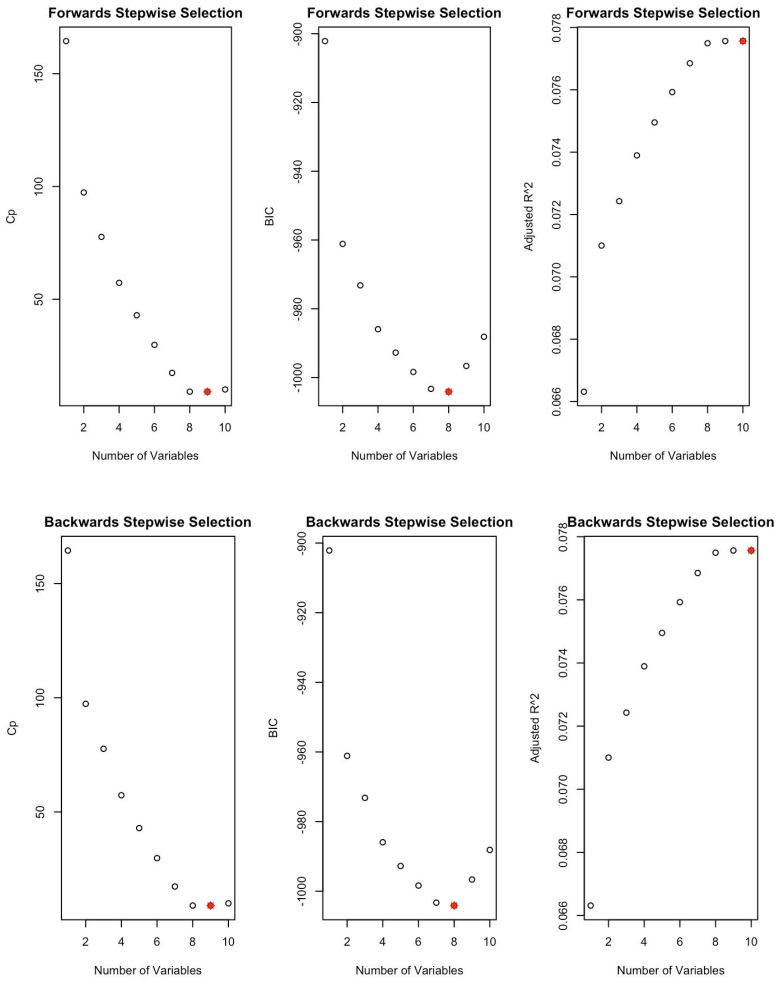
PROJECT BREAKDOWN

Because the project was a classification model, we used all sorts of applicable methods. We broke these down into two main groups- one group of models primarily focused on numerical and categorical variables and the other group of models primarily focused on the textual data. There is a tradeoff between these two different groups so we did not define any model as the "most useful"; the relative importance and performance is very subjective. The models used for numerical features include logistic regression, LDA, QDA, PCA, and SVMs. The models used for textual features included KNN, Naïve Bayes Classifier, Decision Trees, Random Forest and Bagging.

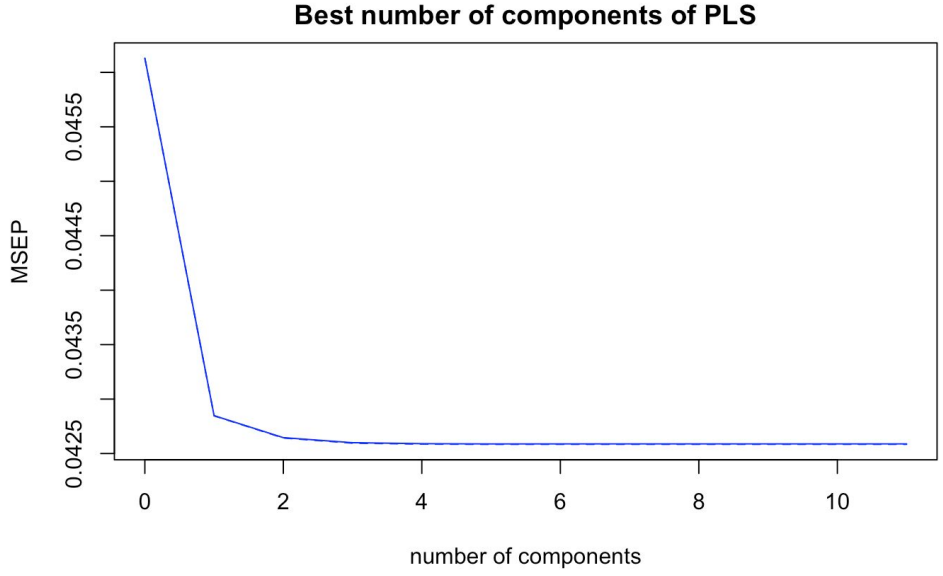
NUMERICAL AND CATEGORICAL ANALYSIS

Feature Selection





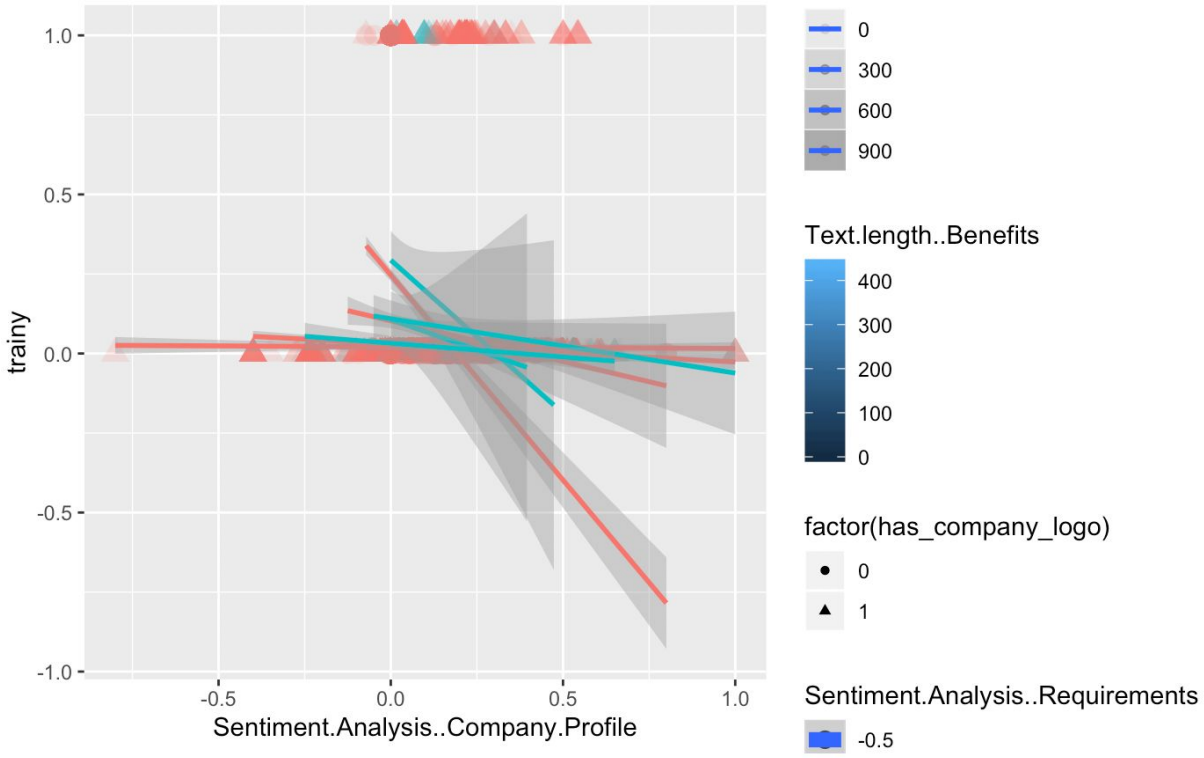
Creating a least squares linear regression model gives us a test error of 0.04981329 and using the features most commonly selected by best subset selection, forward stepwise selection, and backwards stepwise selection gives us a test error of 0.04982897.



Partial least squares regression with the best number of components, 3, gives us a test error of 0.04981387. These are higher error estimates than other better models and there are also a number of issues with using linear regression to solve a classification problem.

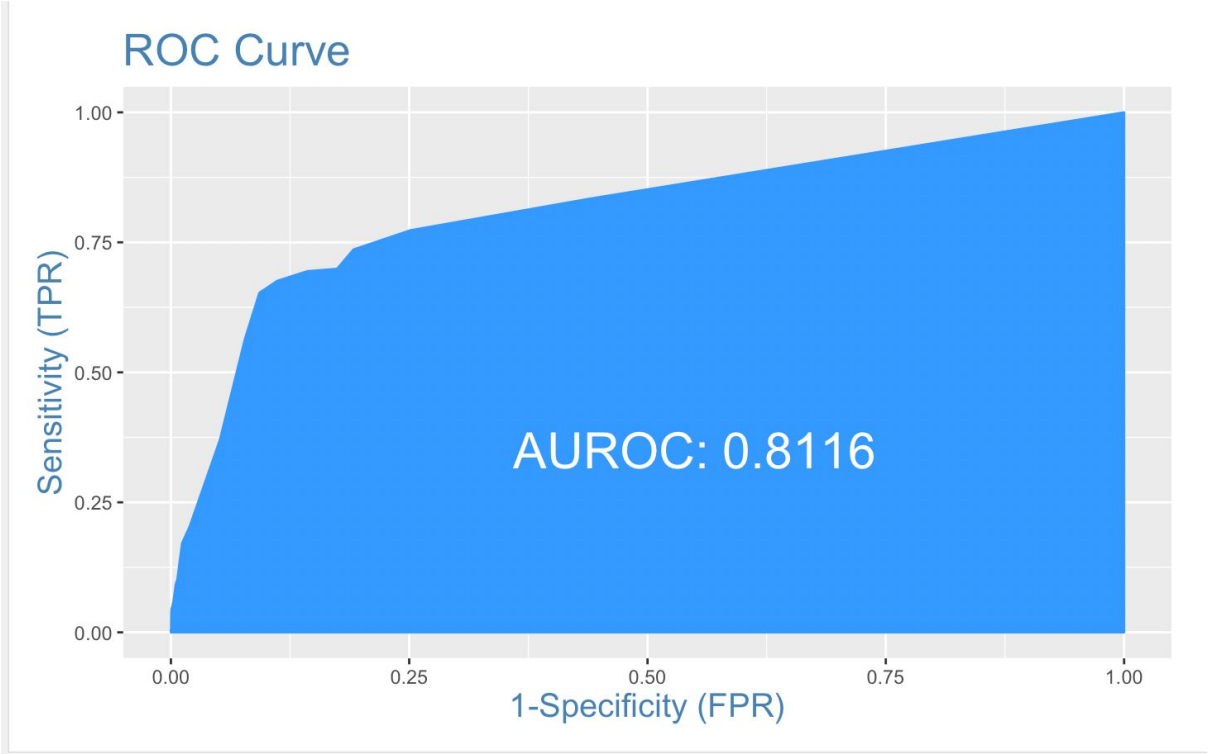
For one, it predicts continuous values instead of a number 0 or 1, and we do not even have an interpretation for other values. 0 represents a job that is legitimate and 1 represents a job that is fraudulent, so if our model predicts values such as 2.45 or -1 for example, we would have no way to interpret what this prediction value is telling us. Linear regression is also quite sensitive to imbalanced data, so especially depending on the sample, it is inclined to give a worse prediction than a classification model such as logistic regression which has a curve that is more optimal for binary classification. Finally, as we can see in our models, choosing the best subset of features caused 9 variables to be selected - a quite large number, since only 2 were excluded, and partial least squares selected three components.

Yet still the test error from using a model with only these variables was higher than the test error from using a model with all numerical variables. This means there was a significant discrepancy between the low training errors and high test errors which reflects high variance and a tendency for overfitting, so these models would likely perform worse on additional data, which is the opposite of our goal in machine learning.



For example in this plot created from the best subset of variables selected, we can see that linear regression creates straight lines that do not visually fit the shape of the data, which is limited to values on the y-axis of 0 and 1, and the shadow of the lines shows standard error, which is quite large.

Logistic Regression



Logistic regression gives us an error rate of 0.0465, and a false positive rate of 0.0004701457. However, it gives an extremely high false negative rate of 0.9537037.

Logistic regression is a better model, which is more well suited for classification, but this model's false negative rate is concerning. Since 95.37% of the job postings it cleared as legitime were actually fraudulent, this could cause someone seeking employment, who used this model, to put themselves at serious risk by applying to an alarming number of fake jobs that they believed to be real.

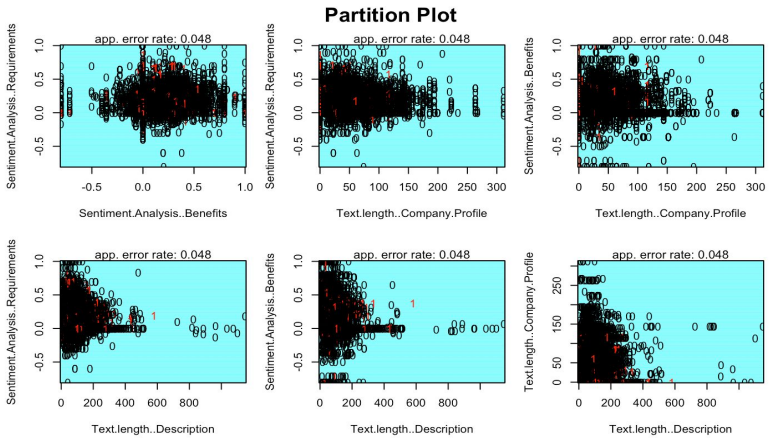
```{r}
logitMod <- glm(trainy~., data=ftrain, family=binomial(link="logit"))
predicted <- plogis(predict(logitMod, ftest))
optCutOff <- optimalCutoff(testy, predicted)[1]
misClassError(testy, predicted, threshold = optCutOff)
```

```{r}
plotROC(testy, predicted)
```

```{r}
c <- confusionMatrix(testy, predicted, threshold = optCutOff)
c
```

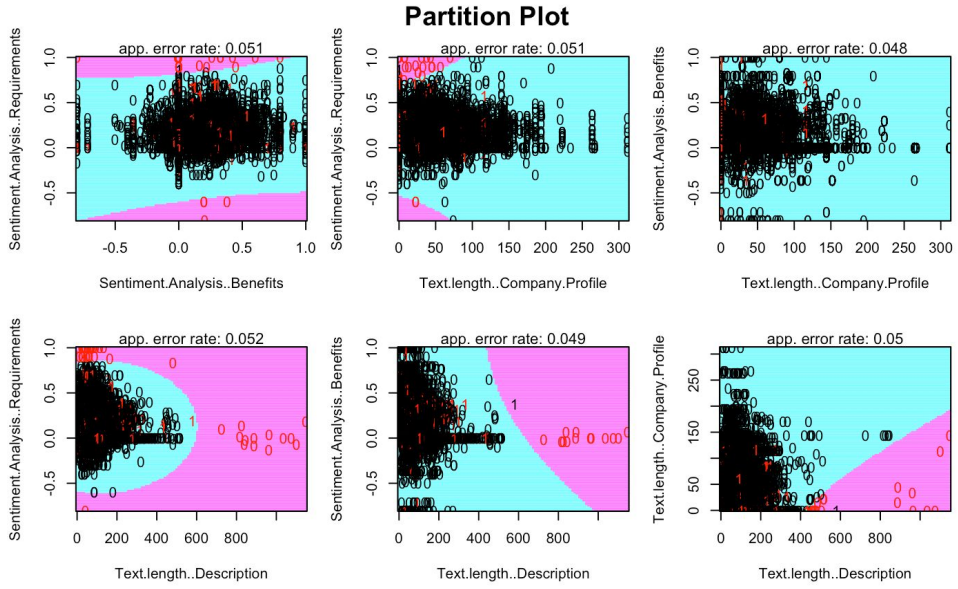
	Legitimate - 0	Fraudulent - 1
Classified as Real - 0	4252	206
Classified as Fake - 1	2	10

Linear Discriminant Analysis



For linear discriminant analysis, the test error is 0.04720358, which is higher than other models. False positive rate is 0.0007052186, which is quite low, but false negative rate is 0.962963 which is very high.

Quadratic Discriminant Analysis



Quadratic discriminant analysis has a quite higher test error of 0.0876957. The false positive rate is also quite higher at 0.06464504, but the false negative rate is quite lower, 0.5416667.

However, false negatives (where a job was classified as 0 for not fraudulent, when it should have a classification value of 1 for fraudulent) should be considered more important than false positives, because while skipping an application to a company with a background that does not seem real but actually turns out to be legitimate would not cause actual harm, on the other hand giving personal information to even one illegitimate company can cause harm to the applicant and there is a high chance of fraud and identity theft.

These plots show the partition from LDA and QDA on a subset of the features in the dataset. It is hard to clearly view the partitions in LDA, but for QDA partitions are more evident and clearly separate a group of non-fraudulent job points. Therefore, we can see how there are less false negatives in QDA.

```r
ldamodel = lda(as.factor(trainy)~., data=ftrain)
ldapred = predict(ldamodel, newdata=ftest)
mean(ldapred$class!=testy)
```

```r
ldadf = cbind.data.frame(testy, ldapred$class, stringsAsFactors = FALSE)
```

```r
fpfn <- function(df) {
  lowhigh = as.integer(0) # low classified as high
  highlow = as.integer(0) # high classified as low
  low = as.integer(0) # actually low
  high = as.integer(0) # actually high

  for (row in 1:nrow(df)){
    tru <- df[row, 1]
    predc <- df[row, 2]
    if (tru == 0) {
      low <- low + 1
      if (predc == 1) {
        lowhigh <- lowhigh + 1
      }
    } else {
      high <- high + 1
      if (predc == 0) {
        highlow <- highlow + 1
      }
    }
  }
  falsepositive = lowhigh/low
  falsenegative = highlow/high
  print("False positive rate:")
  print(falsepositive)
  print("False negative rate:")
  print(falsenegative)
}
```

```r
fpfn(ldadf)
```

```r
qdamodel = qda(as.factor(trainy)~., data=ftrain)
qdapred = predict(qdamodel, newdata=ftest)
mean(qdapred$class!=testy)
qdadf = cbind.data.frame(testy, qdapred$class, stringsAsFactors = FALSE)
```

```r
fpfn(qdadf)
```

Support Vector Machines with Linear and Radial Basis Kernels

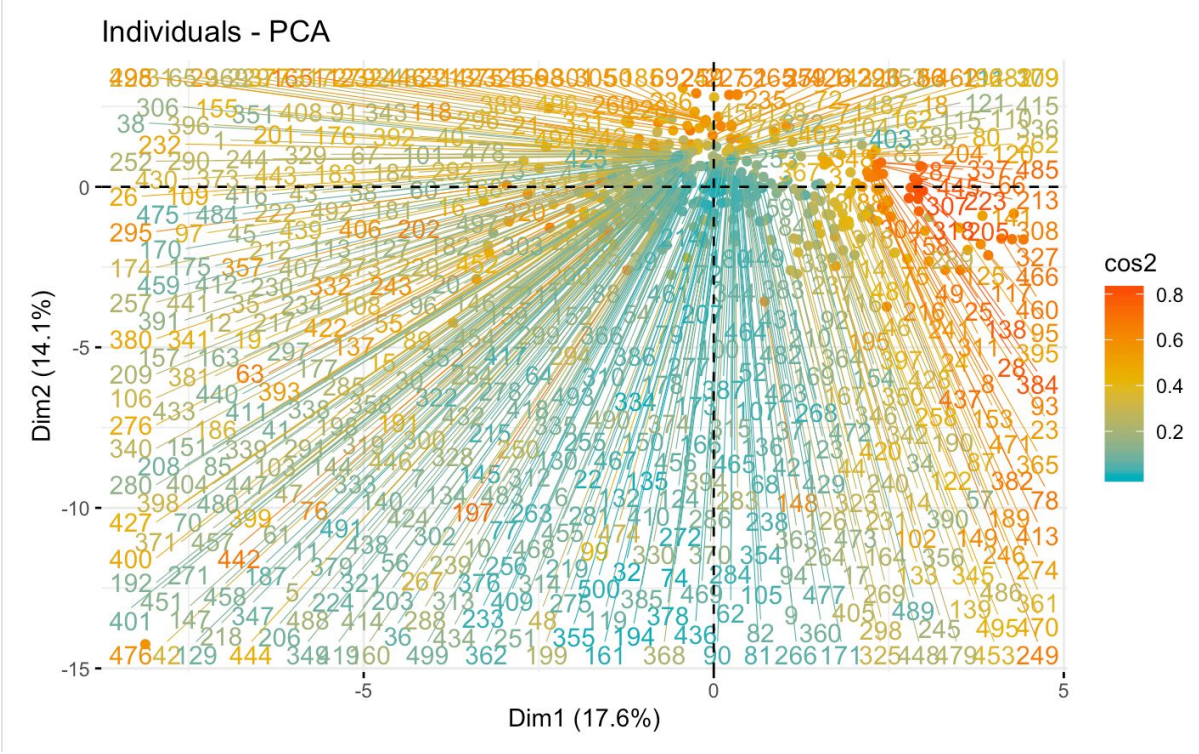
Support vector machines using a linear and radial basis kernel with optimal values of degree, gamma, and cost selected from {0.001, 0.1, 1, 10, 100} had an error of 0.05 and 0.055 respectively.

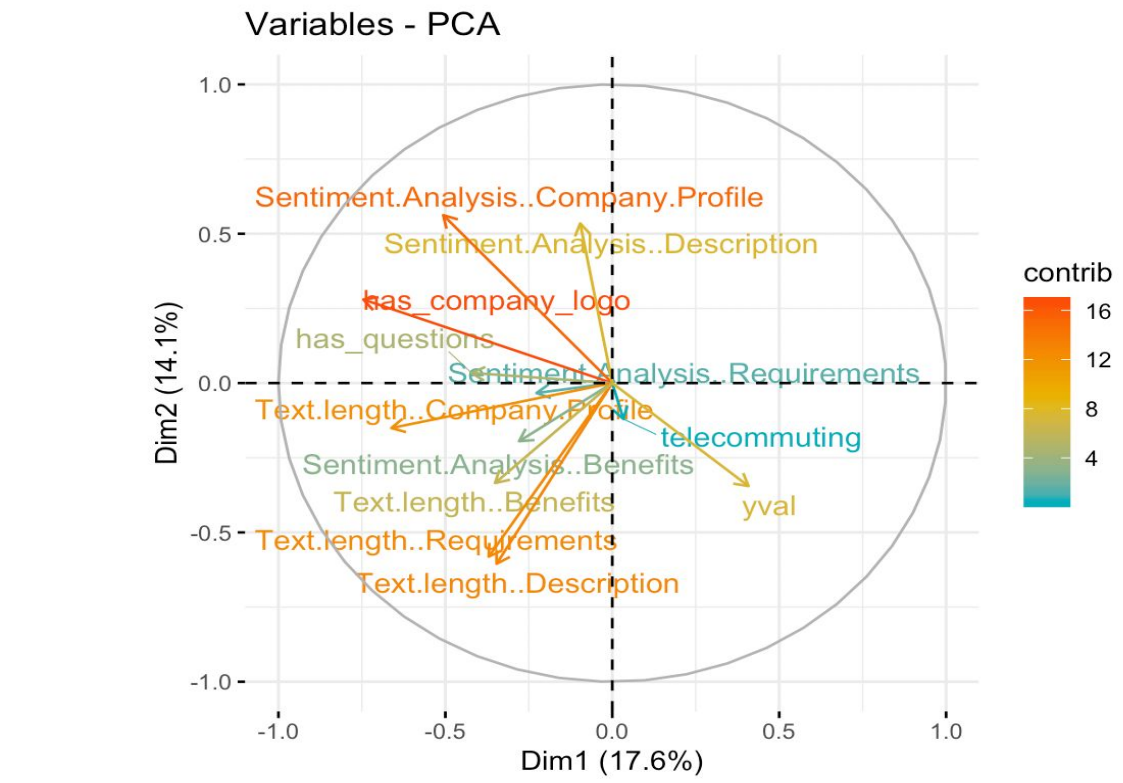
However, these models were very computationally intense and required a smaller training set to be used in order to avoid running out of memory, which limits the amount of information the model has to learn from and limits how accurate these models have the capability to be.

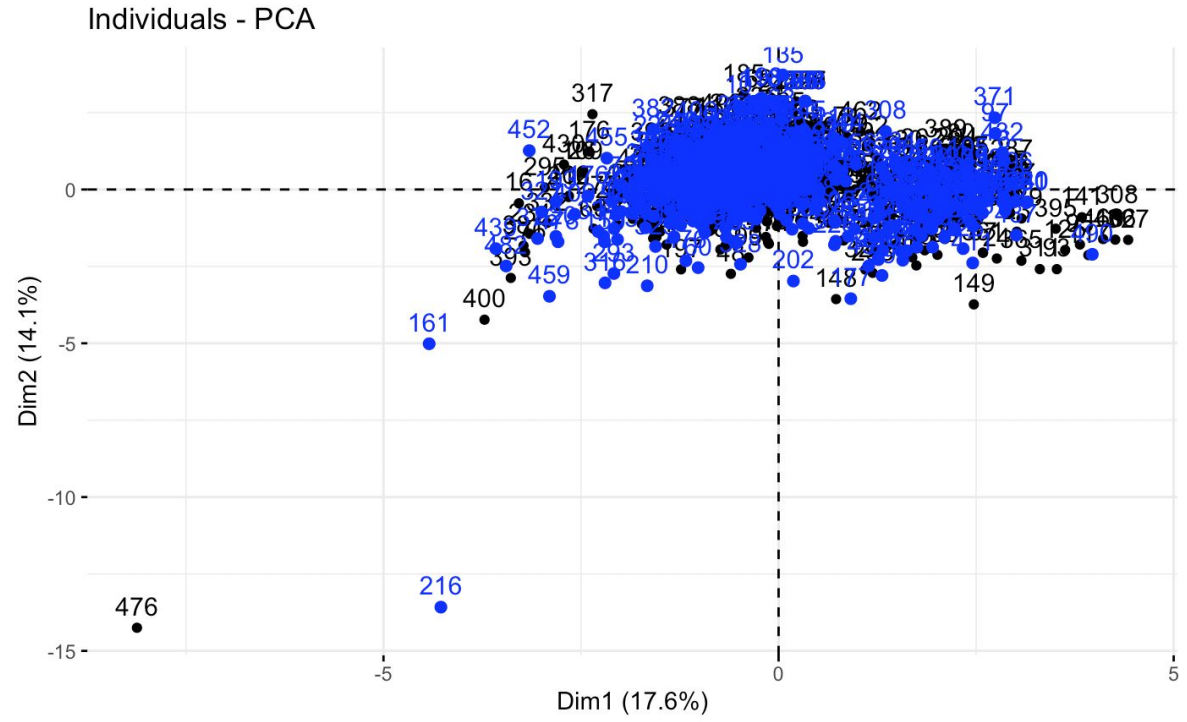
```{r}
linsvmmod <- tune.svm(x = strainx, y = as.factor(strainy), kernel = "linear", cost =
c(0.001,0.1,1,10,100), tunecontrol=tune.control(sampling = "cross"), cross=10)
summary(linsvmmod)
```

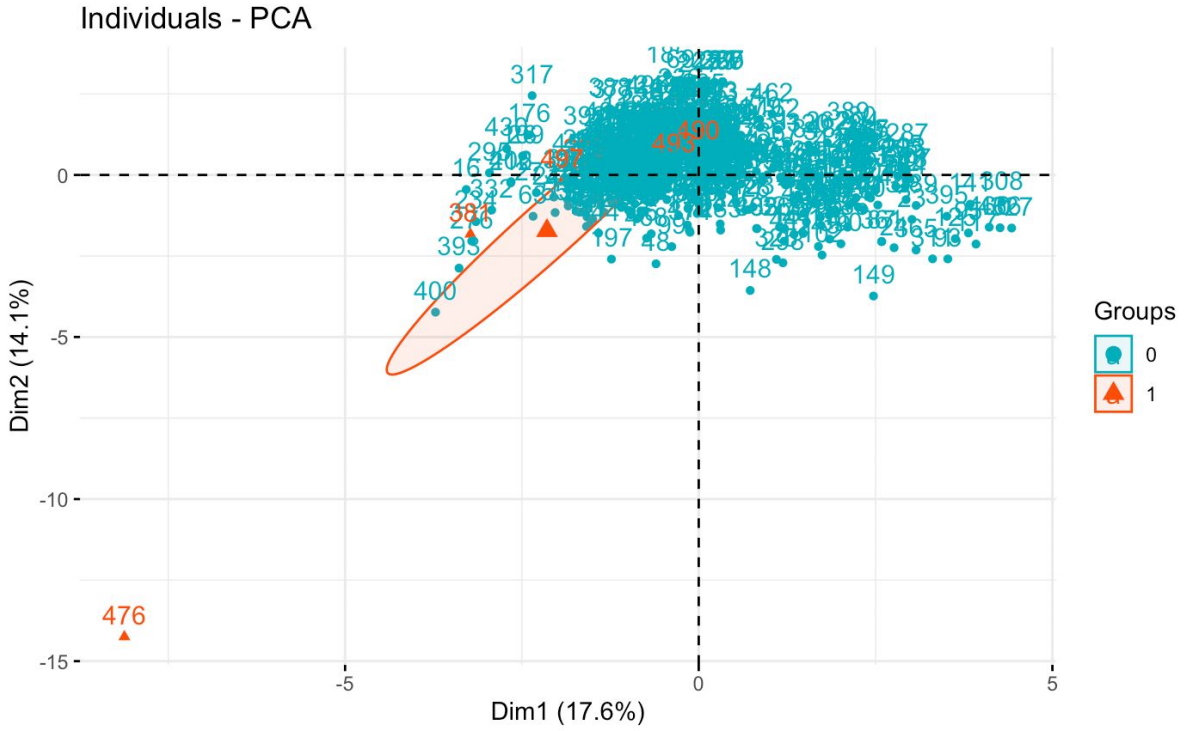
```{r}
radsvmmod <- tune.svm(x = strainx, y = as.factor(strainy), kernel = "radial", degree =
c(0.001,0.1,1,10,100), gamma = c(0.001,0.1,1,10,100), cost = c(0.001,0.1,1,10,100),
tunecontrol=tune.control(sampling = "cross"), cross=10)
summary(radsvmmod)
```

Principal Component Analysis




Variables - PCA
Sentiment.Analysis..Company.Profile
Sentiment.Analysis..Description
has_company_logo
has_questions
Sentiment.Analysis..Requirements
Text.length..Company.Profile
telecommuting
Sentiment.Analysis..Benefits
Text.length..Benefits
yval
Text.length..Requirements
Text.length..Description
contrib
16
12
8
4
Dim2 (14.1%)
Dim1 (17.6%)
1.0
0.5
0.0
-0.5
-1.0


Individuals - PCA
317
452
371
459
400
161
216
476
149
Dim2 (14.1%)
Dim1 (17.6%)
0
-5
-10
-15
5



We implemented PCA, as it allows for us to better visualize the variation present in the dataset, since we have many variables. PCA works best on wide datasets, such as ours, although ours has proportionally many more observations, still making it a tall and thin matrix.

This issue could account for the reason why the plots we created were so difficult to read. So much variation is being explained by the first and second dimensions, that the plots appear like giant blobs centralized to those areas.

Principal Component Analysis does a decent job in certain areas separating the fake jobs from real jobs, but overall there is too much overlap in the predicted groups for this to be an effective model.

TEXTUAL ANALYSIS

Preliminary preparation

For almost every model, the textual features of each observation were converted into different arrays using the Tfidf-vectorizer. This Python tool calculates word frequencies for each observation and weighs them relative to the word frequency over the entire dataset. The output is a numeric array of the frequency of each unique word in an observation's textual feature.

The key thing we looked for in these textual classification models was the confusion matrix, especially the number of false positives - fake jobs that are misclassified as real. There is much more damage in the real world by misclassifying fake jobs as real than by misclassifying real jobs as fake. While type two errors result in wasted time and a scammer's access to personal information, type one errors result in a missed opportunity to apply for a job, ultimately having a negative effect on one's career.

Naïve Bayes

Similar to the spam detection examples gone over in class, we believed that Naïve Bayes could also be a good way of detecting if a job posting is real or fake.

The formula simplified down to:

$$P(Fake/Word) = [P(Word/Fake) * P(Fake)] / P(Word)$$

$$P(Word) = P(Word/Fake) * P(Fake) + P(Word/Real) * P(Real)$$

We implemented Naïve Bayes by first classifying jobs based on each textual feature individually. Multinomial Naïve Bayes was used since the focus was on counting the occurrence of each feature. In the algorithm implementation, there was a smoothing parameter and it used prior fit, so prior class probabilities were used. The resulting confusion matrix of using this model on the textual features is shown below.

- **Company Profile**

	Predicted Real Job	Predicted Fake Job
Real Job	3384	3
Fake Job	27	162

The overall accuracy was 0.9916. There were only 30 misclassifications in total and 86% of fake job postings were correctly classified.

- **Description**

	Predicted Real Job	Predicted Fake Job
Real Job	3407	1
Fake Job	58	110

The overall accuracy was 98.35%. There were 59 misclassifications in total and 65% of fake job postings were correctly classified.

- **Requirements**

	Predicted Real Job	Predicted Fake Job
Real Job	3383	3
Fake Job	168	22

The overall accuracy was 95.22%. There were 168 misclassifications in total and 12% of fake job postings were correctly classified.

- **Benefits**

	Predicted Real Job	Predicted Fake Job
Real Job	3401	6
Fake Job	156	13

The overall accuracy was 95.47%. There were 156 misclassifications in total and 8% of fake job postings were correctly classified.

From using Naïve Bayes on these different textual features, it seems that Company Profile and Description are very important in classifying jobs. This was probably because the company profiles and description are unique to each job posting. Requirements and Benefits were not very important in classifying jobs, as these are general assumptions and do not vary much between jobs of the same category. Any job posting could list generic requirements or benefits since they are pretty standard. In addition, many job postings do not include Benefits at all regardless of if they are real or fake.

KNN and geospatial data

One of the textual features that we had trouble incorporating were the job locations. It did not really fit well with the other textual features so we came up with the idea of using KNN on geographic coordinates to literally calculate the proximity of fake job postings with each other. Based on the job's location to other job's locations and using majority voting, KNN can determine what the job posting would be classified as.

The Geocode package of the Google Maps API was then used to translate the location to exact geographic coordinates in latitude and longitude. The slow speed and API connection errors meant that only 3300 observations were processed to include a latitude and longitude. The KNeighorsClassifier of the sklearn Python library was implemented. For the distance calculations, we decided to use Manhattan distance.

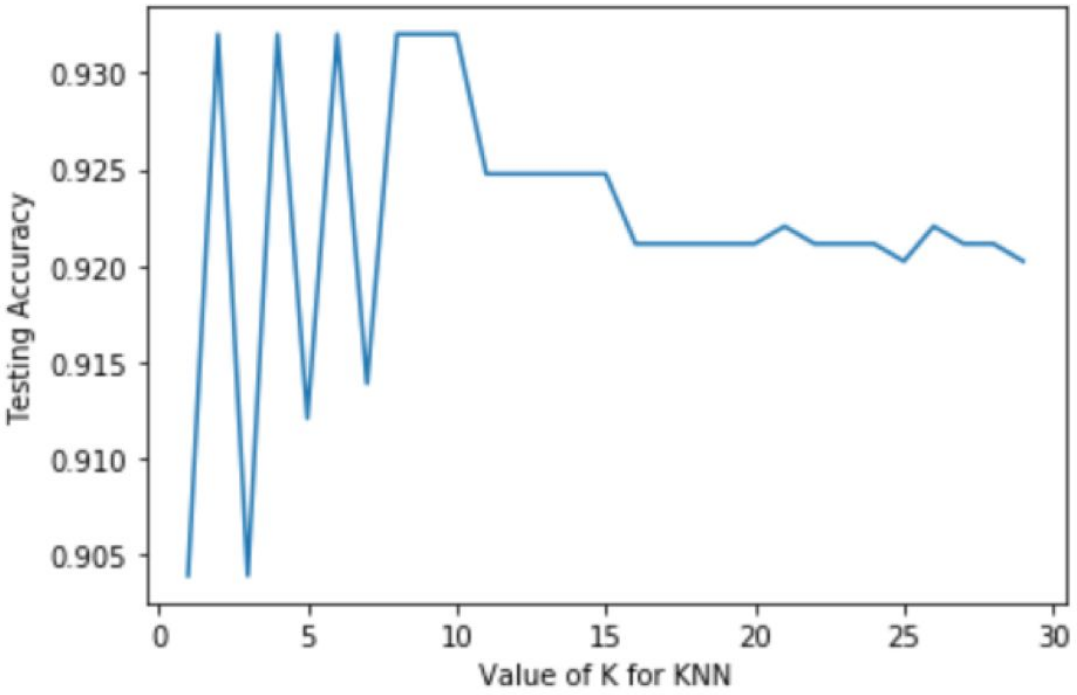
Below is the code to see what the optimal value for k-neighbors would be.

```
k_range = range(1,30)

from sklearn import metrics

scores = {}
scores_list = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors = k, p = 2, metric='manhattan')
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)
    scores[k] = metrics.accuracy_score(y_test, y_pred)
    scores_list.append(metrics.accuracy_score(y_test, y_pred))
```

The result was a graph where the x-axis was the number of neighbors and y-axis was the accuracy. It is shown below.



A k-value of 6 seemed appropriate, so a confusion matrix was generated with that k-value. The confusion matrix result was…

	Predicted Real Job	Predicted Fake Job
Real Job	1023	1
Fake Job	74	5

The accuracy of this model was 0.932.

The conclusion of this model was that while the accuracy was a high number of 0.932, KNN of only geographic coordinates is not a good model for detecting fake jobs. The only reason why the accuracy of the model was that high was because there was a much higher proportion of real jobs to fake jobs. In addition, it is very worrisome that out of 79 fake job postings, only 5 were correctly classified. The rest were misclassified as real jobs.

Decision Trees

For the decision tree, bagging, boosting and random forest, the four textual features of each observation were combined to form an aggregate textual feature. This aggregate text was converted into a numerical array using the Tfidf-vectorizer and used for the models.

Random forest

	Predicted Real Job	Predicted Fake Job
Real Job	3805	1

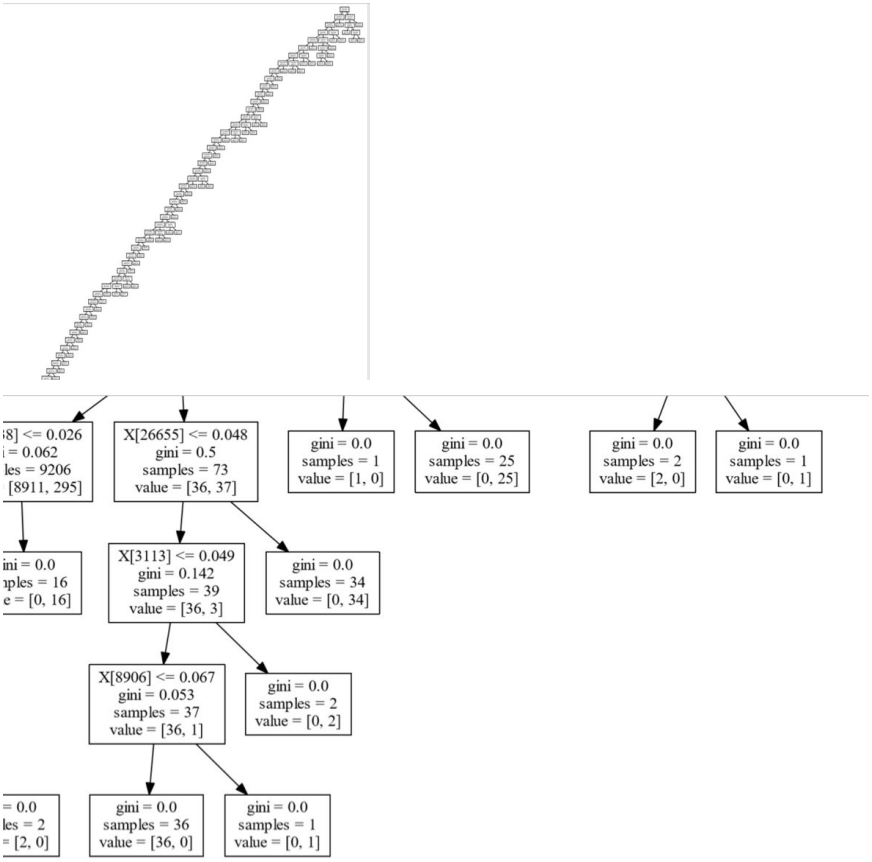
Fake Job	105	112

Accuracy 0.9736

Decision tree

	Predicted Real Job	Predicted Fake Job
Real Job	3776	30
Fake Job	66	151

Accuracy 0.976

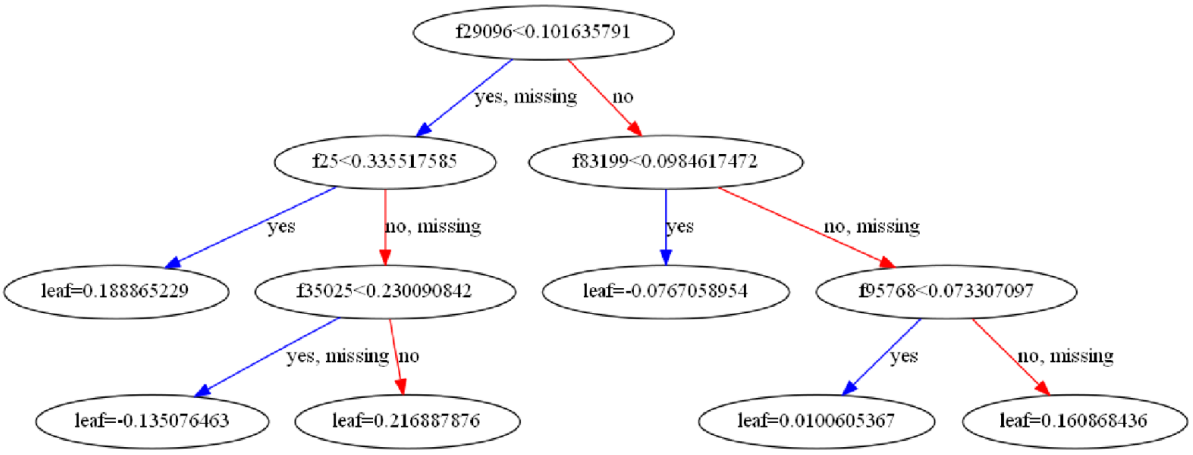


Each X[array] value represents a single word. Each branch, then, extends from that word with similar characteristics, such as sentiment score, length, etc.The decision tree produced a computationally easy and accurate model, far more than the random forest had.

Boosting (XgBoost)

	Predicted Real Job	Predicted Fake Job
Real Job	3802	4
Fake Job	91	126

Accuracy 0.976



XgBoost is a version of boosting. In boosting, the model is built sequentially by minimizing errors from previous models. XgBoost adds onto this by using gradient descent, tree pruning, and regularization to prevent overfitting and bias. The accuracy was the same as the decision tree, although the decision tree is easier to implement and understand.

Bagging

	Predicted Real Job	Predicted Fake Job
Real Job	3805	1
Fake Job	120	97

Accuracy 0.969

We implemented bagging, as it is used to improve the performance of simple models, while reducing overfitting for complex models. Here, several models were fitted on different samples of the data from our training set, with replacement. The models were aggregated using the averages of the previous models, where we trained regression trees on the model.

The base estimator for bagging is a SVC (support vector classification). SVMs used with bagging help to reduce variance, hence avoid overfitting. It allowed us to get a big picture idea of the data.

Conclusion

Our categorical and numerical analysis proved to be not as useful as our textual analysis. Out of all of the numerical and categorical models, the SVM with radial basis vectors proved to have the highest accuracy and was a good, comprehensive model, although its implementation was very tedious and difficult.

Decision trees, despite being the simplest of the models used, were actually the most accurate.

Bagging actually produced the worst result. Random forest produced the second worst result among the models while boosting improved it slightly, although not as much as decision trees. Decision Trees are usually most useful where the explainability between the variables is prioritised over accuracy. They are also easy to compute and explain why a particular variable is more significant than others. The tree can be visualized, and hence,

for non-technical users, it is easier to explain the results from model implementation. Since our data was more non-parametric in nature, we found that decision trees produced the most accurate and easy to understand model.