

# **Predicting the Fraudulence of Job Posts with Binary Classifier:**

## **Stat 154 Final Project Report**

Group 2: Jiayin Guo, Eric Li, Ziyue Wang, Yujiao Zhao

May 9, 2022

<b>0. Cover Sheet</b>	<b>1</b>
<b>1. Exploratory Data Analysis</b>	<b>2</b>
<b>2. Data Cleaning</b>	<b>6</b>
<b>3. Feature Creation</b>	<b>8</b>
3.1. Adding Length of Text	8
3.2. Vectoring the Words according to the GloVe Dictionary	8
3.3. Identifying the most popular words for fraudulent and non-fraudulent job posts	9
3.4. Power Feature Creation	10
<b>4. Unsupervised Feature Filtering</b>	<b>11</b>
4.1 Addressing multicollinearity	11
4.2 Applying ridge selection	11
<b>5. Preliminary Models</b>	<b>12</b>
5.1. Random Forest	12
5.2. Support Vector Machine	13
5.3. XGBoost	14
<b>6. Unsupervised Clustering</b>	<b>16</b>
<b>7. Final Model</b>	<b>17</b>
<b>8. Discussion</b>	<b>18</b>

## 0. Cover Sheet

Team Number: 2


Team Leader: Ziyue Wang 3034679220

Member 1: Jiayin Guo 3035418572

Member 2: Eric Li 3036058661

Member 3: Yujiao Zhao 3034542252

We acknowledge each team member (including the leader) participated in coding for the project as below. We certify that the classifier we built for the project was trained only on the training set handed to us by our GSI for the project.

	Leader	Member 1	Member 2	Member 3
Coding %	25%	30%	25%	20%
Coding Description	EDA, PCA, part of sec. 3.3	Pre-processing pipeline	XGBoost, clustering, parts of SVM	Tuning random forest/svm, model evaluation
Signature				

Team Leader Signature: \_\_\_\_\_

Date: \_\_\_\_\_

# 1. Exploratory Data Analysis

With the rise of internet scams, fraudulent job posts are a common problem with which jobseekers, employers, and job board services must contend. According to the FBI, thousands of victims lost over \$59 million in 2020 alone to employment scams. While the best defense against these scams are diligent, informed jobseekers forewarned with tips on how to identify them, it is incumbent on job board services to identify, flag, and evaluate potentially fraudulent job posts.

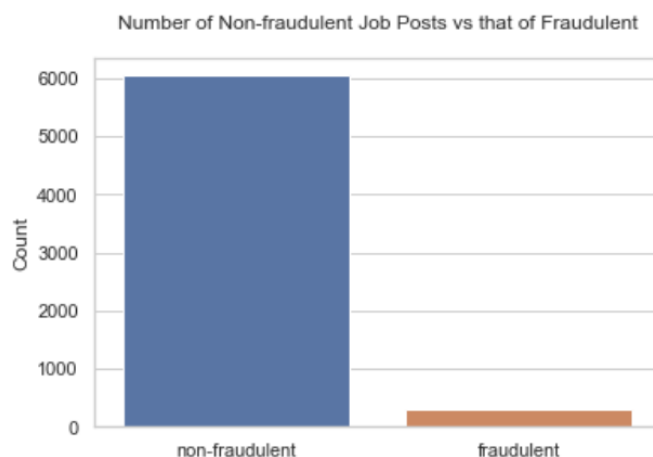
Our objective is to create an effective binary classification model to identify fraudulent job posts. Our training set of job posts consists of 6362 rows, 16 features, one unique identifier column, and one classifier of fraudulent/legitimate. Only 4 of the 16 features are numeric and the rest 12 are text information which needs to be processed to a machine-readable language:

```
RangeIndex: 6362 entries, 0 to 6361
Data columns (total 18 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   job_id                 6362 non-null   int64
1   title                  6362 non-null   object
2   location                6248 non-null   object
3   department              2255 non-null   object
4   salary_range           1024 non-null   object
5   company_profile         5150 non-null   object
6   description              6361 non-null   object
7   requirements             5386 non-null   object
8   benefits                3822 non-null   object
9   telecommuting           6362 non-null   int64
10  has_company_logo        6362 non-null   int64
11  has_questions           6362 non-null   int64
12  employment_type         5152 non-null   object
13  required_experience      3870 non-null   object
14  required_education      3456 non-null   object
15  industry                4637 non-null   object
16  function.               4073 non-null   object
17  fraudulent              6362 non-null   int64
dtypes: int64(5), object(13)
memory usage: 894.8+ KB
```

Empty values in non-binary columns were assumed to be deliberate as an optional input. Our test set is the `job_verification_data` in the afternoon competition. It contains 1000 rows.

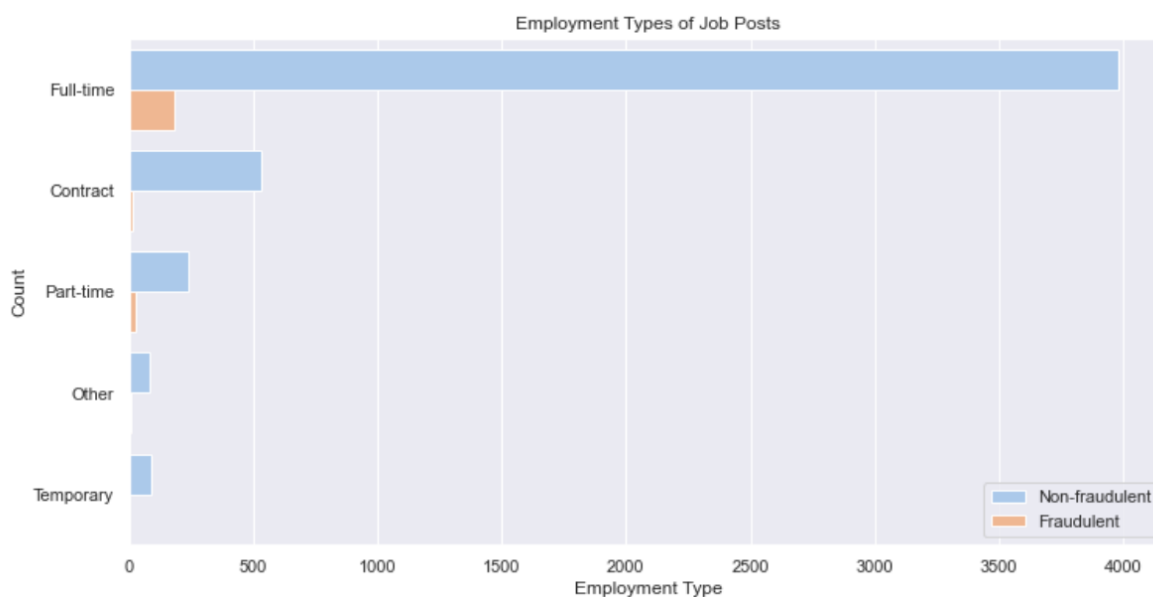
Notice that there are null values in most of the columns, so we need to take care of them in data cleaning. Certain columns that consist primarily of null values like `salary\_range` may need to be dropped.

Also, our target, `fraudulent`, is extremely unbalanced:



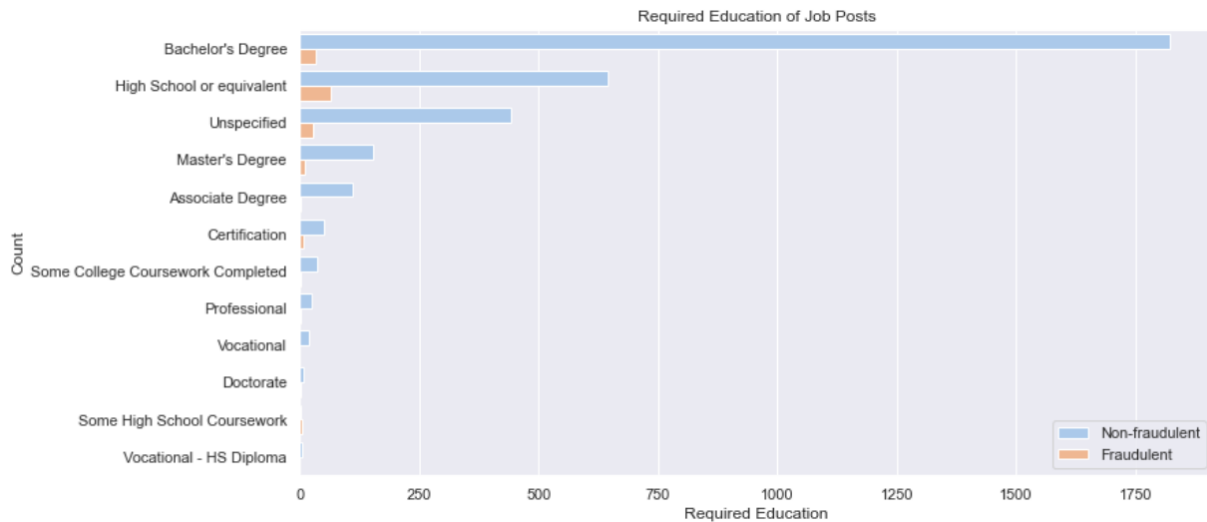
Only about 4.8% of the data are fraudulent. We propose to use the SMOTE technique to balance it in the data cleaning part. SMOTE balances the data by creating synthetic fraudulent rows using random nearest neighbors of a random fraudulent row.

Below are a few visualizations of the data:

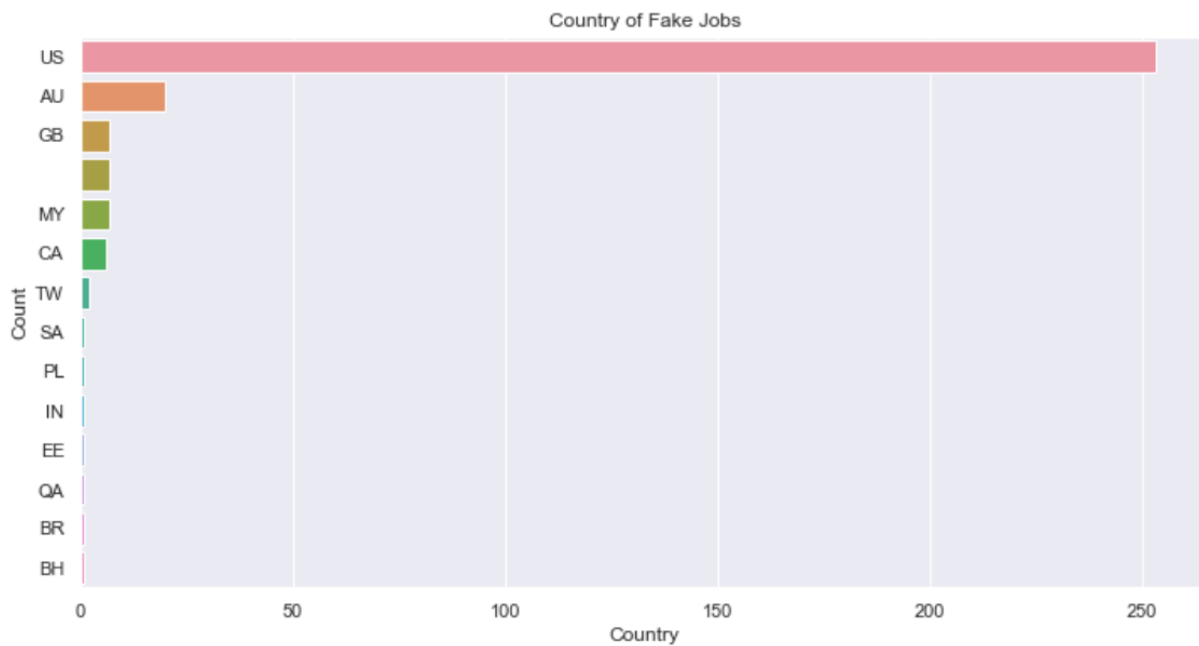


We can see that after dropping the null values, the most popular type of employment is full-time

for both fraudulent and non-fraudulent job posts. Interestingly, the percentage of fraudulent part-time jobs is higher than that of the other employment types.

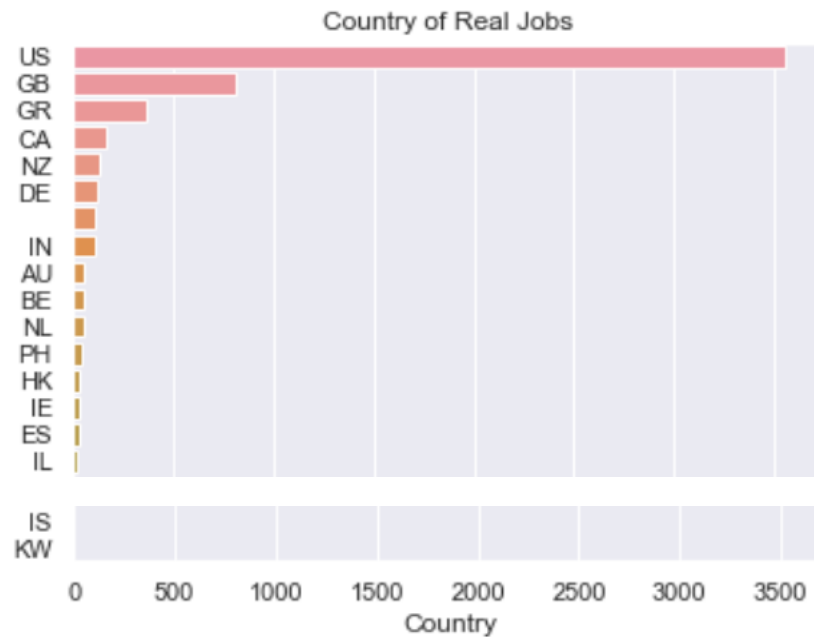


Most job posts require a bachelor's degree though high school or equivalent and unspecified also appear on a relatively frequent basis. Among job posts that desire a high school level education or do not specify their requirements, there seem to be more fraudulent job posts.



Fake jobs primarily come from the U.S.

However, that does not necessarily mean that having the job location in the U.S. means that it is a fake job post because most of the job posts, fraudulent and non-fraudulent, come from the U.S. Nonetheless, one can see that there are disproportionately more fraudulent jobs posts from AU and MY.



Hence, separating the country from 'location' might make it a more useful feature.



A correlation heatmap between the binary categorical variables in the data shows that 'has\_company\_logo' is more correlated with the response than the other. Nonetheless, by these preliminary examinations, there does not seem to be a column that accurately separates fraudulent job posts from non-fraudulent ones. And considering that most of the information are in the text columns, it is probably going to be very hard for human beings to read a job post and then confidently tell whether it is fake.

## 2. Data Cleaning

- **Drop the columns that are irrelevant or not helpful**

First of all, we have dropped `job\_id` because the id of a job has nothing to do with its fraudulence. We have also dropped `salary\_range` because there are too many missing values in that column for it to be helpful for the prediction.

- **Leave the 0/1 columns alone**

Since `telecommuting`, `has\_company\_logo`, and `has\_questions` are binary categorical variables and do not have any missing values, they are kept without further alterations.

- **Label the ordinal variables**

`employment\_type`, `required\_experience`, and `required\_education` are ordinary categorical variables. We have labeled them by increasing integers starting from 0 as is seen in the dictionary below and dropped the original columns.

```
replacement_dict = {
    'employment_type': {'Full-time':0,
                        '': 1,
                        'Part-time':2,
                        'Contract':3,
                        'Other':4,
                        'Temporary':5},
    'required_experience': {'':0,
                           'Not Applicable':1,
                           'Internship':2,
                           'Entry level':3,
                           'Associate':4,
                           'Mid-Senior level':5,
                           'Director':6,
                           'Executive':7},
    'required_education': {'':0,
                           'Unspecified':1,
                           'Some High School Coursework':2,
                           'Vocational':3,
                           'High School or equivalent': 4,
                           'Vocational - HS Diploma':5,
                           'Certification':6,
                           'Some College Coursework Completed':7,
                           'Associate Degree':8,
                           'Bachelor's Degree': 9,
                           'Master's Degree': 10,
                           'Professional':11,
                           'Doctorate':12}
```

- **One-hot encode the other categorical variables**

`industry` and `function` are also categorical variables but without an intrinsic order within them. Labeling them would introduce an arbitrary distance between different levels that is not in the data. Hence, we need to use one-hot encoding. In order to not overfit our model by having too many features in it, we have only encoded the categories with occurrences greater or equal to 15. If a job's industry/function didn't reach this threshold, we created `others\_indus`/`others\_func` and labeled it as 1.

After extracting the country from `location`, we have selected the top 15 countries that appear in fraudulent job posts and performed OHE per our observation in EDA.

The original columns are dropped.

- **Text columns**

It is obvious that most of the information in this dataset is contained in the text columns.

Because the cleaning of these columns is closely connected to our feature creation, we will discuss it in [section 3.1](#).



### 3. Feature Creation

(text columns in this section refer to `'company_profile'`, `'description'`, `'requirements'`, and `'benefits'`, `'location'`, `'title'`, `'department'`.)

#### 3.1. Adding Length of Text

First of all, we have calculated the length of each of the text columns and appended them to the dataframe as `'xx_len'`.

For the sake of easier text mining, we have:

- replaced the missing values in the text columns,
- removed punctuations and meaningless stopwords (e.g., a, the, of),
- lowercased every word,
- Used the porter stemming algorithm to remove the common morphological and inflexional endings from words (i.e., making “learning”, “learnt”, and “learns” the same),

and merged the remaining information into a new column `'text'`. Now, we have one big bag of words for each observation instead of multiple columns.

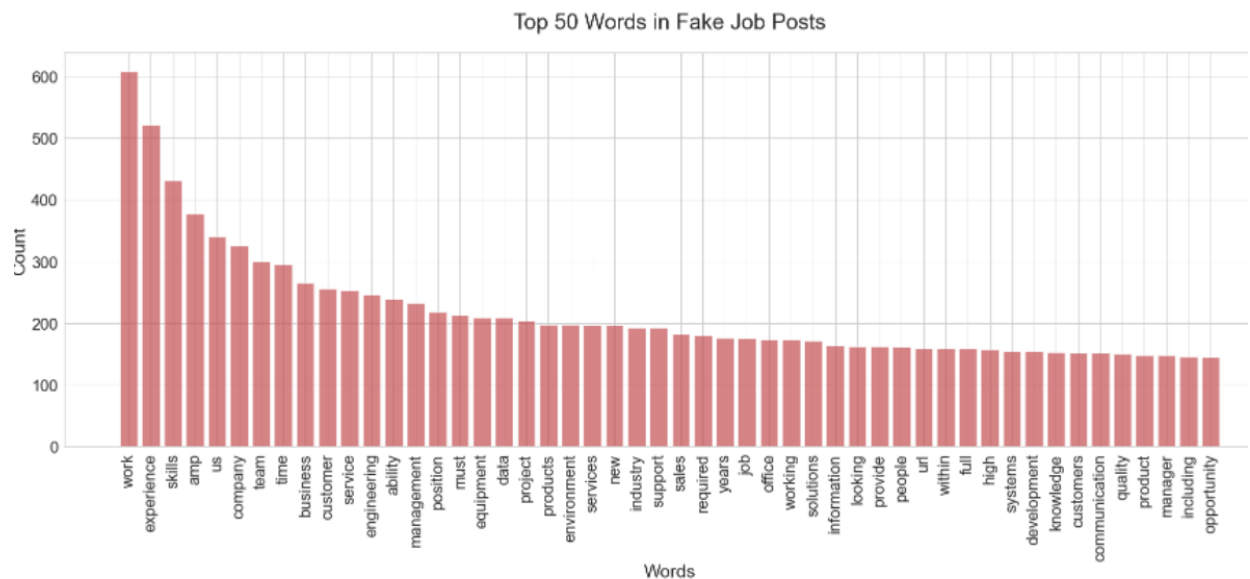
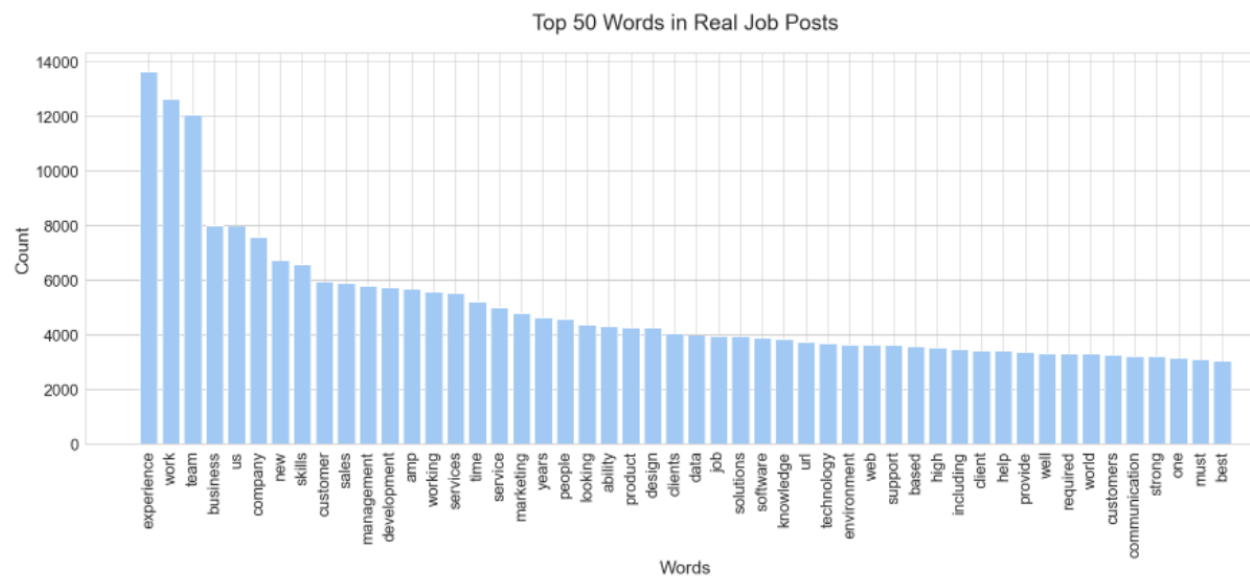
#### 3.2. Vectoring the Words according to the GloVe Dictionary

The Global Vectors for Word Representation (abbreviated as GloVe) algorithm is an extension to the word2vec method which represents a word with a vector and trains a set of word vectors so that it places similar words close to each other.

We use a file with pre-trained vectors GloVe which replaces each word in `'text'` with a 300 by 1 vector and averages across each observation. The result is 300 columns of float numbers that capture the characteristics of the text columns which we append to our feature matrix.

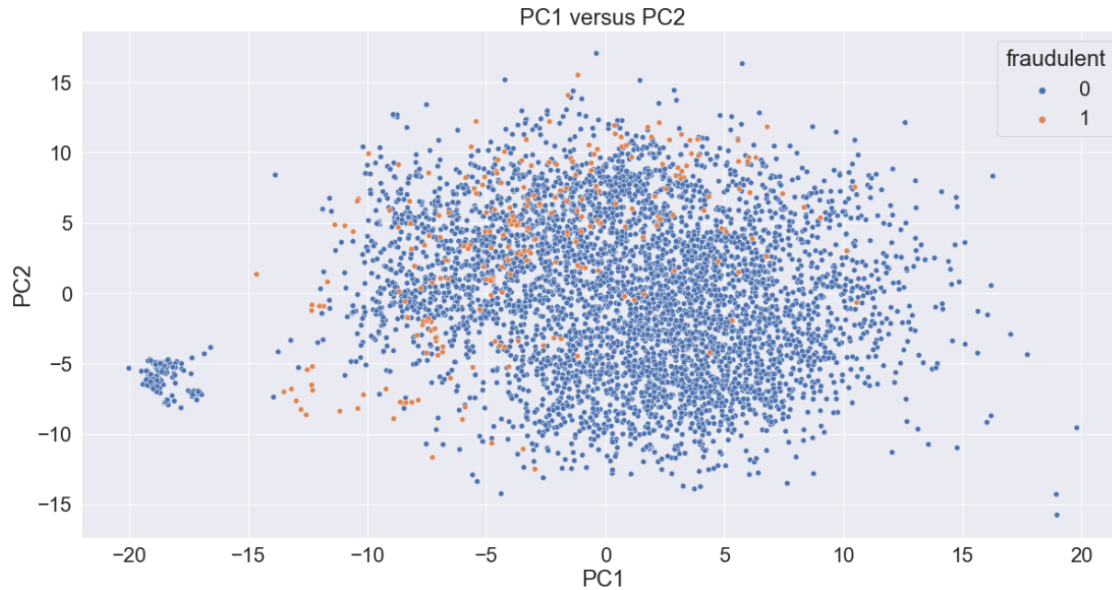
### 3.3. Identifying the most popular words for fraudulent and non-fraudulent job posts

We have also counted the occurrences of words in the fraudulent and non-fraudulent job posts respectively. After making two lists of words in descending order of their appearance frequencies and crossing out the words that appear in both, we have picked the top 15 most popular words for the two kinds of posts and added them to our feature matrix.

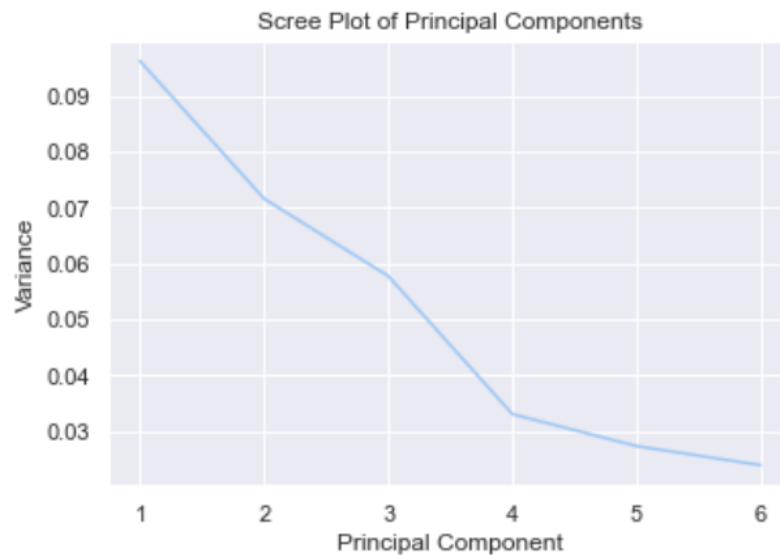


### 3.4. Power Feature Creation

We have also tried Principal Component Analysis, or PCA, to see if they can offer any insight on the data or be better predictors as linear combinations of our current features.



It seems that fraudulent posts' PC2 tend to be higher than their PC1 in comparison to non-fraudulent posts. Non-fraudulent posts have some outliers that have significantly lower PC1 and PC2 than the majority. Unfortunately, we have failed to come up with a particularly good explanation for these observations.



Based on the scree plot, we decide to use the first four PC's as additional features.

## **4. Unsupervised Feature Filtering**

### **4.1 Addressing multicollinearity**

To reduce model overfitting caused by excessive and redundant features, we employed unsupervised feature filtering by computing pairwise correlations between all features, including our transformed text features and created power features. Features were dropped based on a cutoff correlation of 0.5.

This procedure leads to great performance improvements in the SVM model, increasing the TPR from 0.08 to 0.68 since SVM is essentially a linear method-based model; the random forest also has some improvements after it but not significantly.

### **4.2 Applying ridge selection**

Besides dealing with multicollinearity, selecting the features using ridge and the L1 norm has also been tried. Unfortunately, it yields no improvement in both models, so in our final script, we didn't adopt it.

## 5. Preliminary Models

To increase model sensitivity, we first rebalanced the data by oversampling with the SMOTE technique such that approximately 50% of the job posts are fraudulent. For our binary classification problem, we used a random forest, a support vector classifier, and gradient boosting. Per assignment instructions, we used 10-fold cross-validation for hyperparameter tuning.

### 5.1. Random Forest

As it is one of the best tree-based models, we naturally begin with a random forest. With our random forest model, we tuned four hyperparameters: the number of trees, maximum tree depth, and class weight. After running sequential cross-validation to gradually narrow down our intervals and choices for the hyperparameters, we found the best choice for the hyperparameters were 105, 36, and {1:8} respectively. The default number of features considered at each split is approximately  $\sqrt{p}$  where  $p$  is the number of features.

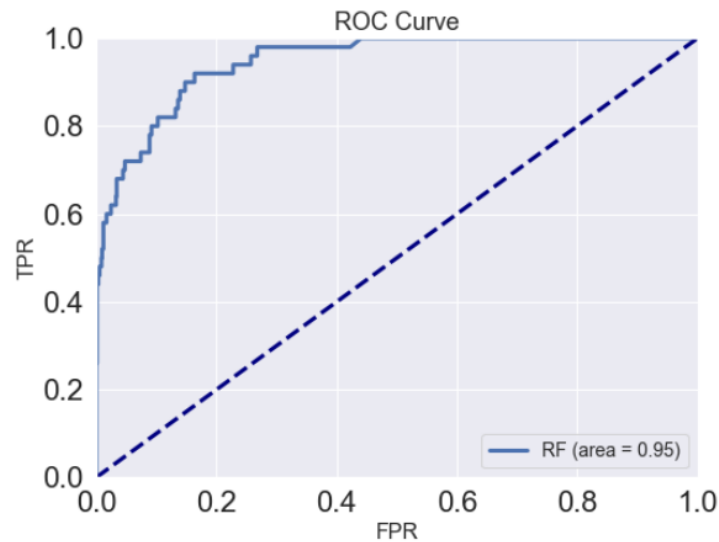
With this model, we have a true positive rate (TPR) of 0.6, a true negative rate (TNR) of 0.9799, and accuracy of 0.9609. Our accuracies (10-fold each) with each feature set are [0.9885 0.9885 0.9886 0.9888 0.9890 0.9890].

Confusion matrix on the test data:

	Actual Fake Job	Actual Real Job
Predicted Fake Job	30	19
Predicted Real Job	20	930

We can see that our model does a really good job recognizing non-fraudulent job posts, but its predictions for fraudulent job posts is not that ideal.

To better assess our model, we have also plotted the ROC curve which portrays the ratio of True Positive Rate (TPR) over False Positive Rate (FPR). The closer the curve is to the left and top boundaries of the graph, the better our model is.



We can see that our ROC curve is very close to the y-axis around 0 and close to  $y = 1$  for  $FPR > 0.4$ . Hence, we have done a reasonable job but there is still room for improvement.

## 5.2. Support Vector Machine

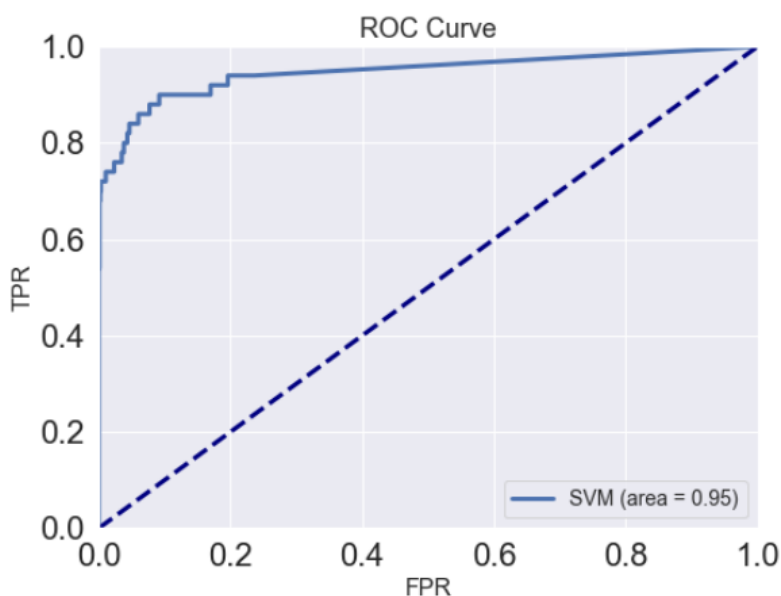
Because of our high-dimensional data, a support vector machine is well-suited to our task. For our support vector classifier, we chose to tune four hyperparameters:  $C$  (which controls the soft margin), kernel choice, the kernel coefficient  $\gamma$ , and class weight. The radial basis function kernel and the 'auto' choice for  $\gamma$  (corresponding to  $1/(\text{number of features})$ ) were consistently the best options, with radial kernel exhibiting a marked increase in accuracy over the linear, polynomial, and sigmoid kernels. This discovery greatly expedited the cross-validation process, yielding values of 2.8 and 'balanced' (class weight = {1:2}) for  $C$  and class weight respectively. Our accuracies with each feature set (10-fold each) are [0.9985 0.9985 0.9985 0.9985 0.9985 0.9985 0.9985 0.9985 0.9985 0.9985].

We achieved with this model a TPR of 0.68, a TNR of 0.9989, and accuracy of 0.9829. Our support vector classifier vastly outperformed our random forest (0.6, 0.9799, 0.9609).

Confusion matrix on the test data:

	Actual Fake Job	Actual Real Job
Predicted Fake Job	34	1
Predicted Real Job	16	948

ROC Curve:



The SVM classifier has a higher sensitivity than our Random Forest classifier while having an approximately equal, if not higher, overall accuracy.

### 5.3. XGBoost

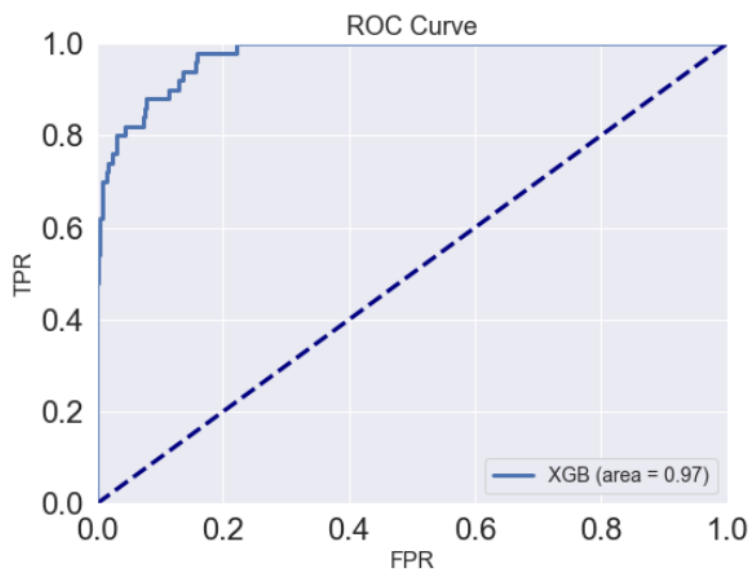
XGBoost, or eXtreme Gradient boosting, is a form of tree-based gradient boosting that uses both L1 and L2 loss. It is famed for its success in Kaggle and other machine learning competitions. After tuning with cross-validation, we have decided to use a learning rate of 0.02 with 300 trees, max tree depth of 20, minimum sum of instance weight = 1, gamma (minimum loss reduction for further partition) of 0.5, subsample ratio of 0.8, and colsample\_bytree (subsample ratio by tree) of 0.6.

This model achieved 0.62 TPR, 0.9926 TNR, and 0.9739 accuracy. As expected from this famous algorithm, it does indeed have a higher AUC (0.97) than our random forest (0.95) and support vector classifier (0.95). However, the sensitivity and accuracy is markedly lower than the support vector classifier. Our accuracies for each feature set (each with  $k = 10$  runs) is [0.9912 0.9960 0.9942 0.9912 0.9946 0.9935 0.9914 0.9916 0.9961 0.9945].

Confusion matrix on the test data:

	Actual Fake Job	Actual Real Job
Predicted Fake Job	31	4
Predicted Real Job	19	945

ROC Curve:





## 6. Unsupervised Clustering

Since the data we are dealing with contains two pre-specified classes, we choose K-means clustering as our cluster method, with  $k = 2$ .

K-means clustering only finds the local minimum, so we did 6 rounds of it with different initial label assignments to make sure the global best result is included.

```
TPR: 0.20711974110032363 ; TNR: 0.5826862712704444 ; Accuracy: 0.5644451430367808
TPR: 0.7928802588996764 ; TNR: 0.4159920700479101 ; Accuracy: 0.4342973907576234
TPR: 0.7928802588996764 ; TNR: 0.4161572773831158 ; Accuracy: 0.4344545740333228
TPR: 0.20711974110032363 ; TNR: 0.5833471006112672 ; Accuracy: 0.5650738761395787
TPR: 0.20711974110032363 ; TNR: 0.5782256732198909 ; Accuracy: 0.5602011945928953
TPR: 0.7928802588996764 ; TNR: 0.4161572773831158 ; Accuracy: 0.4344545740333228
```

There are two kinds of results out of the 6 rounds:  $\text{TPR} = 0.207$ ,  $\text{Accuracy} = 0.564$ ; and  $\text{TPR} = 0.793$  and  $\text{Accuracy} = 0.434$ .

Note that although in the second result the TPR has increased a lot, both yield extremely low accuracy given the dataset has 95% of the data in one label. This indicates that unsupervised clustering can not track the classes in the dataset very well.

## 7. Final Model

Based on our experiments in section 5, we have decided to use our [SVM classifier](#) as our final model. We have trained our model with all data we have received before the competition on May 8 and used cross-validation to tune the model's parameters. The final parameters we have used are:

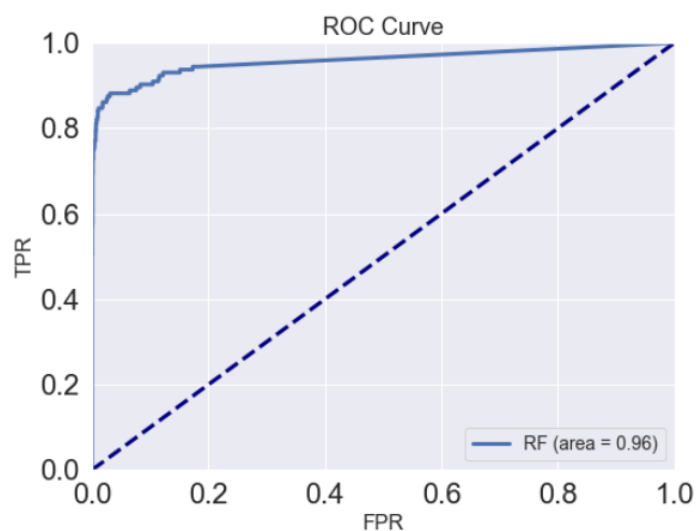
- `random_state = 0`
- `C = 2.8`
- `kernel = 'rbf'`
- `gamma = 'auto'`
- `class_weight = 1:2`
- `probability = True`

The results we get are 0.6875 sensitivity, 0.9989 specificity, and 0.9838 overall accuracy.

Confusion matrix on the test data:

	Actual Fake Job	Actual Real Job
Predicted Fake Job	99	3
Predicted Real Job	45	2825

ROC Curve:



Overall, our model performs pretty well, though the sensitivity definitely could still be improved.

## 8. Discussion

Given our objective to create an effective binary classification model to identify fraudulent job posts, our final SVM model has done a reasonably well job. Even if impressiveness of a 98% overall accuracy is undermined by the fact that as only 5% of the data are fraudulent, so a constant model should have an accuracy of 95%, our true positive rate of 69% and true negative rate of 99.9% still indicate that our model is able to differentiate fraudulent job posts from legitimate one for most of the times.

The challenge for this prediction is to capture the characteristics of fraudulent job posts when they only make up a small fraction of the entire data set. By bootstrapping our original data with SMOTE to generate a balanced sample and adjusting class weight when initiating the model to put more focus on the fraudulent class, we have managed to achieve a true positive rate of almost 70%. Nonetheless, suppose we have more time to work on this project, further improving the sensitivity of our model while maintaining an overall accuracy above 95% should still be our main focus.

Potential strategies to refine our model include doing a better job in text mining and finding features that are more helpful. Right now, we have only been considering the text columns as bag of words, analyzing the frequency of certain indicative words without paying attention to phrases, sentences, and logic of the paragraph. Performing sentiment analysis and opinion mining may yield features that capture the characteristics of job posts better.

If we were to have access to computers that are faster and more powerful, we could also do best subset selection or at least forward or backward selection which should give up a better list of parameters than what is given by unsupervised feature filtering alone. Based on the fact that our test error is approximately the same as our training error, we argue that our model is not overfitting. But we would also like to cross-validate, tune, and build models for different numbers of features to see where our choice of number of features lands on the bias-variance tradeoff curve.