Using Machine Learning Classification Models to Identify Fradulent Job Postings

Miriam Farrington
Department of Engineering & Computer Science, Syracuse University
Spring 2020
mifarrin@syr.edu

Abstract: Online job boards contain thousands of postings which prospective candidates read and submit their applications daily. Unfortunately, some of these job postings are fake and used in order to illegally mine or collect candidate personal information without their knowledge. In this project I have tried to tackle this problem by training and evaluating the performance of several machine learning models to identify these fake job postings with a high degree of accuracy.

I. INTRODUCTION

Machine learning has many practical applications that can benefit us in solving practical problems. One of these is to save time and efficiency and help to reduce fraud. Today's job search candidates utilize many online job boards to search and apply for jobs and it is both wasteful of time and potentially dangerous if they apply for a job posting that is fake. Worse yet, they can unknowingly give up personal information or other potential security concerns by responding to fake job postings. It would be highly beneficial if we can save the time for these candidates by identifying and labeling those postings which are suspected to be fake. Using predictive modeling we can try to tackle this problem by defining machine learning algorithms to try and accurately identify these fake postings.

II. PROCESS

A. The Dataset and Problem Statement

The problem of identifying a fake job posting from a group of legitimate job postings is a good problem space for utilizing machine learning to perform binary classification. In this case, I am making use of a publicly available data set on Kaggle which contains tens of thousands of job postings which are given binary target class labels (1= fake, 0=not-fake) [1]. Additionally this dataset contains 18 features (columns) including the target class label column.

Much like a real-world scenario, the dataset of job postings is predictably unbalanced. In a set of 17,880 labeled job postings, only ~800 are classified as fraudulent. This scenario is common in other areas of identifying fraud, as we expect the fraudulent data records to be less numerous in the data than the legitimate records.

B. Exploratory Data Analysis and Data Cleaning

Upon inspecting the dataset, I found that it was relatively clean from the outset with no duplicates or special characters. Of the 18 features in the dataset, all but 5 were of categorical object type. This data set consists mostly of descriptive, categorical values. There are 5 numeric values but these are 1/0 labels representing Boolean labels such as 'has_company_logo'.

When examining the data set I found missing values present in the data; however upon further examination it appears that these values were Missing At Random (MAR), and not Missing Not At Random (MNAR). Because of this, I felt confident that I could handle the missing values without impacting the outcome of the models. I chose the approach that any column with > 70% missing values I would drop, and the remaining columns I would bucket the missing values into a category in order to preserve as many records of data as possible. This approach resulted in dropping one feature 'salary_range' with > 80% missing values, and the remaining features were preserved with the missing values bucketed into categories such as 'N/A' and 'Other.' Where possible I attempted to incorporate the missing values into an existing category such as 'N/A' without creating a new category just for the missing data.

Figure 1: Feature Missing Value Identification

```
job_id                 0.00
title                  0.00
location               1.94
department            64.58
salary_range          83.96
company_profile       18.50
description            0.01
requirements          15.07
benefits              40.32
telecommuting          0.00
has_company_logo       0.00
has_questions          0.00
employment_type       19.41
required_experience   39.43
required_education    45.33
industry              27.42
function              36.10
fraudulent             0.00
dtype: float64
```

Because this data was predominantly categorical in nature, I did not perform additional normalization steps such as detecting and removing outliers which might have been necessary in a dataset that has a large number of continuous features. Additionally I chose to drop the identifier column 'job_id' as it is purely for identification and should not be used to inform any of the classifier models.

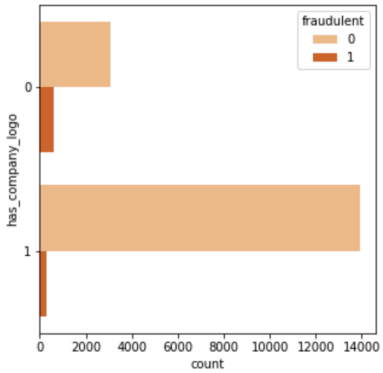
III. Visualizing The Data

After performing Exploratory Data Analysis, identifying and handling missing values, I used data visualization to try and identify any natural correlations between the target class labels and the features.

A. Feature Correlations and Class Lables

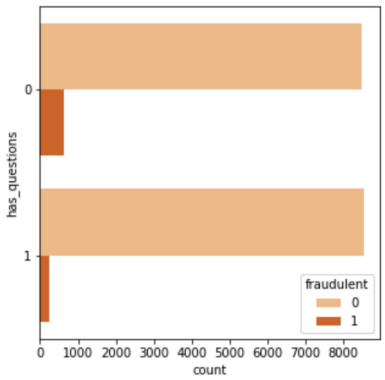
Due to the relatively small number of features in this data set, I was able to perform visualization on almost all of them to determine whether there was any correlation between certain feature values and the occurrence of the target class label of 'fraudulent'. The results were promising, and I found a number of examples where certain feature values were associated with a higher prevalence of class label fraud. Some of these correlations were intuitive; for example whether or not a job posting featured a company logo: in instances were no company logo was provided the number of instances of fraudulent class labels was higher. One might correctly assume that a fraudulent job poster would not bother to go through the additional work to create a company logo.

Figure 2: Chart of Company Logo Classification by number of records labeled fraudulent.



Similarly, whether or not a job posting contained questions for the job candidate (has_questions) was correlated with a higher number of records labeled as fraudulent. Intuitively, we would assume that a fraudulent job poster would not bother with the additional work of posing questions to candidates.

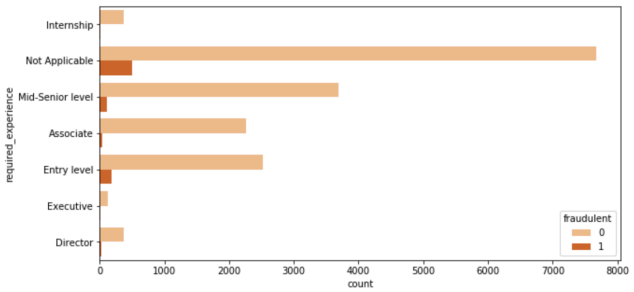
Figure 3: Chart of Has Questions Classification by number of records labeled fraudulent.



It's important to note that while these correlations are certainly interesting, they are not conclusive indicators of a fraudulent posting; there may be valid reasons that a legitimate job poster does not put a company logo on the posting or does not ask questions of candidates.

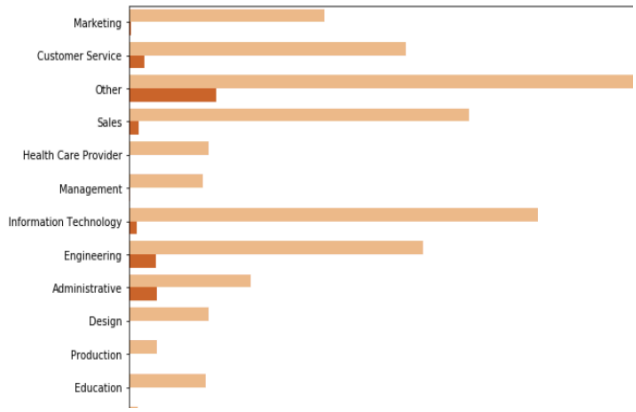
Additionally, other features also seem to carry some degree of predictive power when performing data visualization. When plotting the occurrence of fradulent records against the type of job posting, we can see that there is an increase of fraudulent job posting associated to entry-level job postings when compared to the other job types.

Figure 4: Chart of Job Type by number of records labeled fraudulent.



Likewise certain industries were associated with higher occurrences of fraudulent job postings such as Engineering and Administrative.

Figure 5: Chart of Job Industry Type by number of records labeled fraudulent.



Having performed extensive data visualization, it becomes much easier to see how certain features may weigh into the prediction of the classifier models in the next section.

IV. Building and Evaluating Classifier Models

A. Model Selection

Given that the dataset I have chosen for this task is labeled, I am working with a supervised classification problem; that is to say I already have the target class labels and can evaluate the performance of each model against those ground truth values.

For this task, I chose to evaluate 4 classifiers: Random Forest, Ada Boosting, K Nearest Neighbors and Gaussian Naïve Bayes. Random Forest and Ada Boosting are considered to be ensemble learning methods that combine the classification power of multiple decision tree classifiers to arrive at higher predictive accuracy; whereas K Nearest neighbors is a clustering algorithm that seeks to classify by looking at feature similarities to arrive at a class label prediction. Gaussian Naïve Bayes algorithm attempts to assign conditionally independent probabilities to the attributes in the data, i.e. it Naively assumes that the attributes in the data are independent and never interact with each other. Gaussian Naïve Bayes is an extension of this approach that estimates the distribution of the data by calculating the mean and standard deviation on the training data. It tends to work the best on a data set that is already normally distributed. I chose these models to get a good sense for how each performs on this data set and to evaluate the results after tuning the hyperparameters to achieve as high a accuracy score as possible given the tradeoffs that exist in performance.

For all model evaluations, I am utilizing the holdout method with a 70/30 split between training and testing data. This same holdout set is passed to each model in order to maintain a more accurate comparison between model test results.

B. Evaluating The Random Forest Classifier

The first model I trained was the Random Forest Classifier. a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree. [2] It has a lot of additional hyperparameters to work with, but to start, I chose to set the following.

```
Parameters: {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weig
ht': None, 'criterion': 'gini', 'max_depth': None, 'max_featur
es': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min
_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_sam
ples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_le
af': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': Fa
lse, 'random_state': None, 'verbose': 0, 'warm_start': False}
```

The parameters that most directly impacted this model's performance setting the min_samples_split to 2; and the n_estimators (number of trees) to 100. I sampled larger numbers of trees but found that on this particular dataset increasing the number of trees did not lead to a significant increase in accuracy and I soon ran into performance limitations on execution time. Overall the best performing Random Forest Classifier with this combination of hyperparameters achieved a weighted average accuracy score of 98.0%.

Figure 6: Random Forest Classifier Accuracy

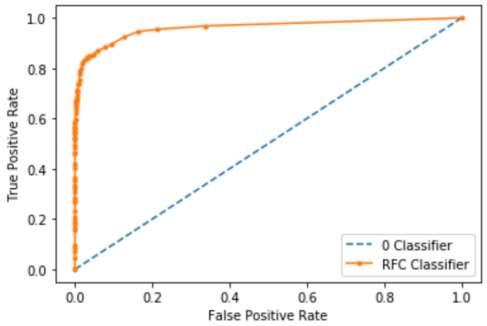
	precision	recall	f1-score	support
0	0.98	1.00	0.99	5143
1	0.99	0.55	0.71	221
accuracy			0.98	5364
macro avg	0.99	0.78	0.85	5364
weighted avg	0.98	0.98	0.98	5364

For each model I evaluated, I plotted the ROC with the True Positive Rate against the False Positive Rate where the True Positive Rate is on y-axis and False Positive Rate is on the x-axis. In this case, the AUC for the RFC is calculated at 0.962, which indicates a high measure of separability as

we want to get the AUC value as close to 1 as possible.

Figure 7: AUC for Random Forest Classifier

```
0 Classifier: ROC AUC=0.500
RFC Classifier: ROC AUC=0.962
```



The Random Forest classifier also contains an attribute feature_importances that allows you to view the relative weights of the features had in the model classification. In this case, the feature 'has_company_logo' and several of the industries had the most importance in this model, whereas some of the individual job titles had relatively low importance.

Figure 8: Feature importance in the RFC Model

	importance
has_company_logo	0.036937
industry_Oil & Energy	0.017447
location_US, TX, Houston	0.015252
title_Data Entry Admin/Clerical Positions - Work From Home	0.014794
function_Administrative	0.014378
...	...
title_Risk Analyst	0.000000
title_Rising Star (3rd Key Holder) - St. Louis Galleria	0.000000
title_Rising Star (3rd Key Holder) - Oakbrook Center	0.000000
title_Rising Star (3rd Key Holder) - Kenwood Towne Centre	0.000000
title_Recruitment Manager (Maternity cover)	0.000000

C. *Evaluating The Ada Boosting Classifier*

Like Random Forest, Ada Boosting Classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases [3]. In this case I chose to fit a Random Forest Classifier using similar hyperparameters to the original Random Forest Model. The Ada Boosting Classifier also has several tunable parameters to select from, in this case I kept the num_estimators set at 100. Using these paramters, the Ada Boosting Classifier achieved a weighted average accuracy score of 96.56%.

```
Parameters:  {'algorithm': 'SAMME.R', 'base_estimator': None,
'learning_rate': 1.0, 'n_estimators': 100, 'random_state': 16}
```

Figure 9: Ada Boosting Classifier Accuracy

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      5143
           1       0.74      0.46      0.57       221

    accuracy                           0.97      5364
   macro avg       0.86      0.73      0.78      5364
weighted avg       0.97      0.97      0.97      5364
```

Because this Ada Classifier was utilizing Random Forest Classifier, I was able to access the relative feature importance in this case as well. Again, similar to Random Forest, 'has_company_logo' had an even higher feature importance in this model of 0.05 while again the specific job titles were assigned no importance. This matches intuitively with some of the correlations already discovered during data visualization.

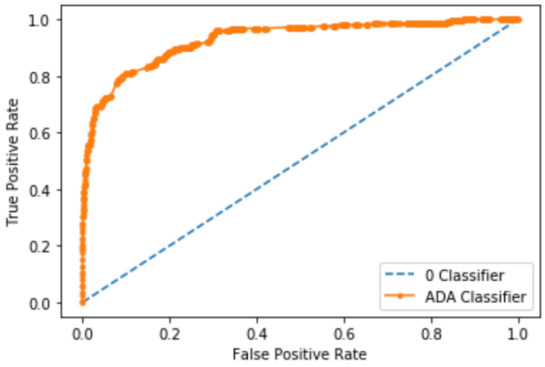
Figure 10: Feature importance in the Ada Boosting Model

	importance
has_company_logo	0.05
location_US, TX, Houston	0.02
employment_type_Full-time	0.02
industry_Hospital & Health Care	0.02
location_US, CA, San Mateo	0.02
...	...
title_Junior Accounts Assistant	0.00
title_Junior Adwords/SEO Specialist	0.00
title_Junior Affiliate Manager	0.00
title_Junior Airframe Structures Design Engineer	0.00
function_Writing/Editing	0.00

When plotting the ROC curve of the Ada Boosting Classifier, we can see that the AUC of this classifier is slightly lower than the Random Forest Classifier at 0.930. However this model still performs well when compared against the no-skill 0 classifier.

Figure 11: AUC for Ada Boosting Classifier

```
0 Classifier: ROC AUC=0.500
ADA Classifier: ROC AUC=0.930
```



D. *Evaluating The K Nearest Neighbors Classifier*

K Nearest Neighbors is distinguishable from the previous two learning methods as it is an instance-based learning method frequently used in classification problems. Unlike ensemble learning methods, KNN utilizes a majority vote based on the nearest neighbors of each point in the data based on a user-provided *k*. The greater the *k* the less sensitive the model may be to noise, but the lines between clusters may become blurred. Likewise a smaller *k* can mean a greater sensitivity to noise in the data but a clearer delineation between neighbors [4].

K Nearest Neighbors has fewer tunable hyperparameters than Ada Boosting and Random Forest, however the importance is finding a *k* value that is a good tradeoff between sensitivity to noise, execution time and accuracy. In this case, after several attempts and testing I found that k=5 achieved a good balance between these factors.

```
Parameters:  {'algorithm': 'auto', 'leaf_size': 30, 'metric':
'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbo
rs': 5, 'p': 2, 'weights': 'uniform'}
```

When run on the test data the KNN Classifier achieves a weighted average accuracy score of 98%.

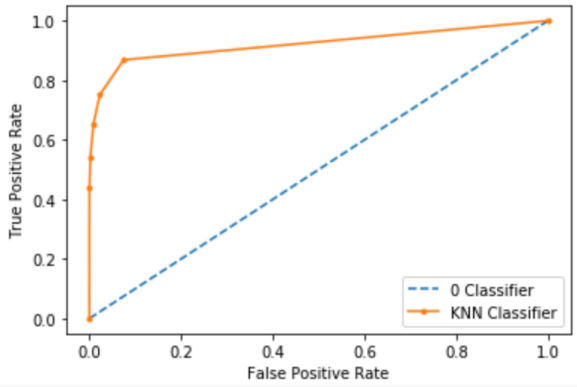
Figure 12: KNN Classifier Accuracy

	precision	recall	f1-score	support
0	0.99	0.99	0.99	5143
1	0.76	0.65	0.70	221
accuracy			0.98	5364
macro avg	0.87	0.82	0.85	5364
weighted avg	0.98	0.98	0.98	5364

When plotting the ROC curve of the KNN Classifier, we can see that the AUC of this classifier is comparable to the Ada Boosting Classifier at 0.921. However, even though weighted average score for this model was slightly higher than Ada Boosting, the AUC value is slightly lower, which could indicate a lower level of separability.

Figure 13: AUC for KNN Classifier

```
0 Classifier: ROC AUC=0.500
KNN Classifier: ROC AUC=0.921
```



E. *Evaluating The Gaussian Naïve Bayes Classifier*

The final model I evaluated was Gaussian Naïve Bayes classifier. Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variables.[5] Unlike the previous models, there is relatively little hyper-parameter input or tuning needed to inform the model or affect the outcome score.

In this case, the Gaussian Naïve Bayes model performed with a weighted average accuracy score of 81%.

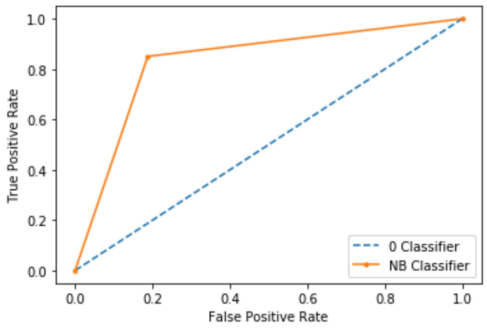
Figure 14: Gaussian NB Classifier Accuracy

	precision	recall	f1-score	support
0	0.99	0.81	0.89	5143
1	0.16	0.85	0.27	221
accuracy			0.81	5364
macro avg	0.58	0.83	0.58	5364
weighted avg	0.96	0.81	0.87	5364

When plotting the ROC curve of the Gaussian NB Classifier, we can see that the AUC of this classifier is lower overall than the previous classifiers measured at 0.831. Also, the ROC is less curve-like in this case. It looks like this may be do to a lesser number of points present in the input data to even out the curve.

Figure 15: AUC for Gaussian NB Classifier

```
0 Classifier: ROC AUC=0.500
NB Classifier: ROC AUC=0.831
```



In this case, my speculation is that Gaussian NB classifier is performing with a lower accuracy than the other classifiers based on how the data is distributed after splitting the categorical features using pandas in order to feed into the model. Because of this, it is highly unlikely that my data set is normally distributed, which is the assumption made by the Gaussian NB model; also the algorithms presupposes that the attributes are independent, and when I split out the categorical features it created new interrelated features. For example, the features:

- function_Science,
- function_Strategy/Planning
- function_Supply Chain

are all associated to the categorical feature of Job Function. However, the Gaussian NB will assume and treat them as independent, which is not the case.

In this case, due to the structure of this data set, Naïve Bayes family of Classifiers may not be as good a choice.

V. Conclusion

After evaluating various Classifier models on the Job posting data set, I was able to see how the various hyperparameter selections and tradeoffs affected performance and accuracy.

My conclusions are that the ensemble learning methods (Random Forest, Ada Boost) and the instance-based method K Nearest Neighbors achieved the highest levels of classification accuracy. In terms of Random Forest, I speculate the performance accuracy is higher because many of the original categorical features in the data appear to have some level of predictive power (as shown in the data visualization portion of this task) and the features are relatively uncorrelated. Ensemble learning methods tends to work well for this type of data set.

Likewise, K Nearest Neighbors classifier worked very well, which could be due to the shared similarities in certain features between the class labeled postings, enough to make accurate predictions as to the correct categorization for a posting with a high degree of accuracy.

Finally, Gaussian Naïve Bayes classifier performed acceptably, but likely did not perform as well as it might have given how the categorical features were converted prior to input into the model. It may perform better on a data set that contains more continuous features or can be placed in a normal distribution prior to modeling. Additionally, the feature conversion meant that there were reduced independence between the features which goes against one of the primary assumptions of Naïve Bayes Classification.

References

[1] Kaggle Job Board Postings Dataset used in training the classifiers: https://www.kaggle.com/shivamb/real-or-fake-fake-jobposting-prediction/data#

[2] Random Forest Classifier Definition from scikit-learn https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[3] Ada Boosting Classifier Definition from scikit-learn https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[4] K Nearest Neighbors Classifier from scikit-learn https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[5] Gaussian Naïve Bayes Classifier from scikit-learn https://scikit-learn.org/stable/modules/naive_bayes.html