Identify Fraud from Enron Email

Machine Learning Project

Udacity Data Analysts Nanodegree Program
Brian J Hartman
2/13/2018

Project Overview

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives. I will use this dataset to create a predictive model that will help determine whether a person in the dataset is a 'POI' or 'Person of Interest'.

Project Goal

The goal of this project is to build a predictive model using a machine learning algorithim to predict if someone is a person of interest based off of financial and email data gather from the aftermath of the Enron scandal. This problem is solved using supervised classification machine learning where we are trying to predict the output as one of two distinct classes: POI or not POI.

Project Dataset

The Enron dataset consist of 146 data points and 21 features from financial and email data of employees of Enron. Of the 146 records for employees, 18 are considered POIs and the remaining 128 are labeled as non-POI.

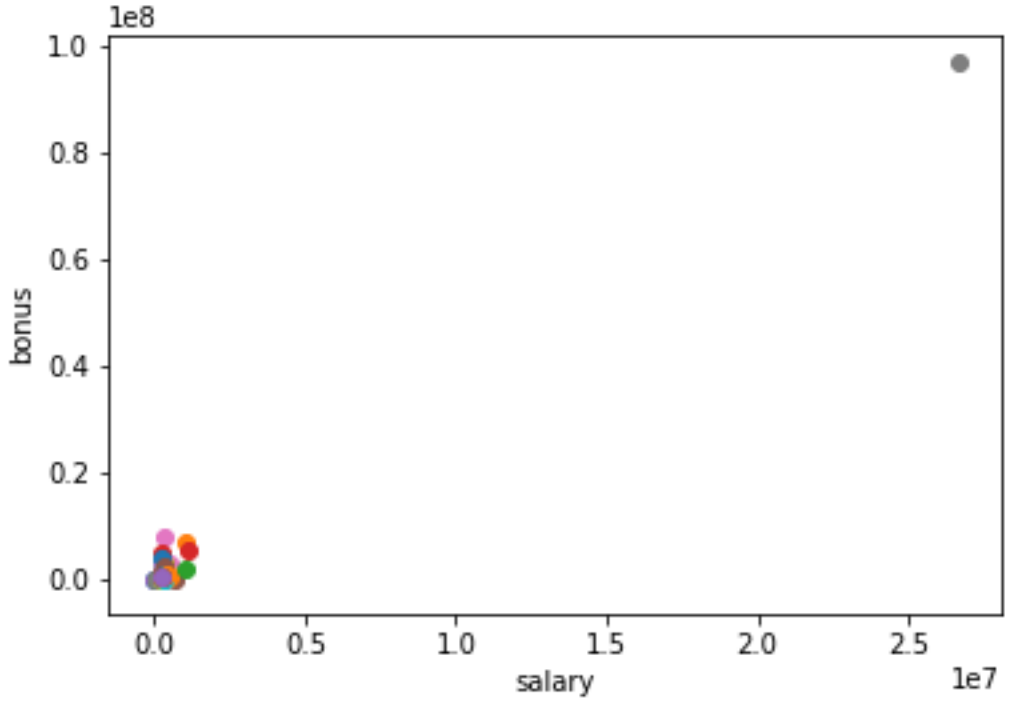
```
<class 'pandas.core.frame.DataFrame'>
Index: 146 entries, ALLEN PHILLIP K to YEAP SOON
Data columns (total 21 columns):
salary                       95 non-null float64
to_messages                  86 non-null float64
deferral_payments            39 non-null float64
total_payments               125 non-null float64
exercised_stock_options      102 non-null float64
bonus                        82 non-null float64
restricted_stock             110 non-null float64
shared_receipt_with_poi      86 non-null float64
restricted_stock_deferred    18 non-null float64
total_stock_value            126 non-null float64
expenses                     95 non-null float64
loan_advances                4 non-null float64
from_messages                86 non-null float64
other                        93 non-null float64
from_this_person_to_poi      86 non-null float64
poi                          146 non-null float64
director_fees                17 non-null float64
deferred_income              49 non-null float64
long_term_incentive          66 non-null float64
email_address                146 non-null object
from_poi_to_this_person      86 non-null float64
dtypes: float64(20), object(1)
memory usage: 25.1+ KB
```

Missing Data

There is also a considerable amount of NaN values(1323) in the dataset. These missing values pertain to features such as Loan Advances, Director Fees, and Deferred Payments. Given that this data is financial data the NaN values represent zeros and not missing data, so these NaN values will be replaced with 0 for the purpose of this analysis.

Outliers

Initial exploration lead to detecting the following outlier:

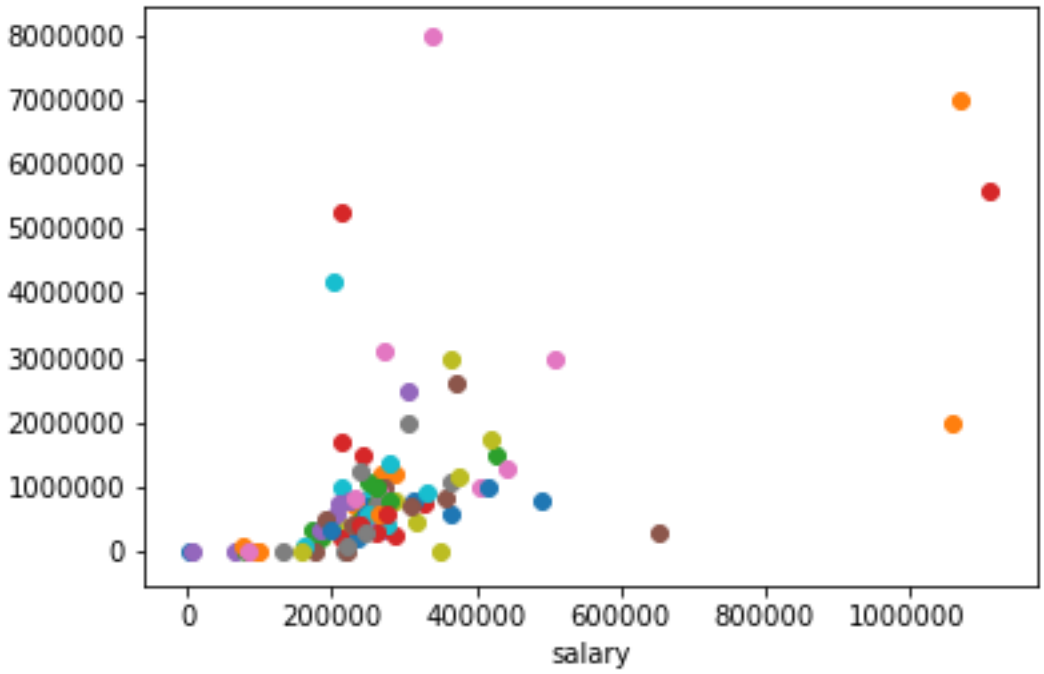


Plotting salary and bonus shows an outlier in the above plot. Further investigation in the table below shows the outlier as the entry "Total". This is an obvious outlier and needs to be removed from the data.

	salary	to_messages	deferral_payments	total_payments	exercised_stock_options
TOTAL	26704229.0	NaN	32083396.0	309886585.0	311764000.0
SKILLING JEFFREY K	1111258.0	3627.0	NaN	8682716.0	19250000.0
LAY KENNETH L	1072321.0	4273.0	202911.0	103559793.0	34348384.0
FREVERT MARK A	1060932.0	3275.0	6426990.0	17252530.0	10433518.0
PICKERING MARK R	655037.0	898.0	NaN	1386690.0	28798.0

5 rows × 21 columns

After removing Total from the data we now have a much better looking plot. Now it appears as though there may be 4 other outliers, but from looking at the max values table these are know POIs and therefore I will not be removing this data.



Feature Selection

To begin with, I will utilize all 21 of the features with the exception of 'email_address' and 'other'. 'email_address' is only an identifier and there is no supporting documentation on what 'other' category contributes to in the dataset.

Create New Features

In addition to the 19 features mention above, I would like to create two new features that may give us additional insight. It could be that POIs have a higher rate of emails between each other and therefore we may be able to add some accuracy to our model knowing this. The two new features are:

fraction_to_poi: this is a calculation of emails sent from a person to a particular known 'poi'
fraction_from_poi: this is a calculation of emails a person recieved from a known 'poi'

Feature Scores

I utilized SelectKBest to return the scores of the features ranked by their importance. SelectKBest, f_classif computes an ANOVA F-statistic between features for classification purposes. Using this method SelectKBest came up with the following list of features and their score:

```
Features and Scores:
exercised_stock_options 25.098
total_stock_value 24.468
bonus 21.06
salary 18.576
fraction_to_poi 16.642
deferred_income 11.596
long_term_incentive 10.072
restricted_stock 9.347
total_payments 8.867
shared_receipt_with_poi 8.746
loan_advances 7.243
expenses 6.234
from_poi_to_this_person 5.345
fraction_from_poi 3.211
from_this_person_to_poi 2.427
director_fees 2.108
to_messages 1.699
deferral_payments 0.217
from_messages 0.164
restricted_stock_deferred 0.065
```

Given this ranking of the features we can see break of importance between the 5th and 6th features; therefore, I will set k=5 and utilize the top 5 features for further analysis.

Feature Scaling

After selecting and creating additional features the next step was to utilize Sklearn's Minmax Scaler to scale the data. Scaling the dataset normalizes

the data and transforms the features to analyze individually to a value between 0 and 1. This scaling is performed so that each feature contributes proportionately to the mean.

Algorithms

The following algorithms were tested and their performance recorded:

Naive_Bayes

Model accuracy: 0.886
Model precision: 0.5
Model recall: 0.6

Support Vector Machine

Model accuracy: 0.886
Model precision: 0.0
Model recall: 0.0

Decision Tree

Model accuracy: 0.795
Model precision: 0.167
Model recall: 0.2

From the performance statistics we can see that, out of the three, the Naive_Bayes algorithm performed the best. It has the highest accuracy as well as the highest precision, in that, precision is a measure of the the ratio of true positives to true positives plus false positives. It also has the highest recall which is a measure of the ratio of true positives to true positives plus false negatives. In all cases the higher the value between 0 and 1 the better the model performs.

In our model we are looking to correctly identify POIs without unnecessarily wasting time on those that are not POIs. With a precision score of .5 our model predicts the for every 40 people it identifies as a POI 20 will in fact be a POI and the remains 20 will be non-POIs. And with a recall score of .6, this tells us that the model can predict 60% of all true

POIs. So, in this application, it would be preferable to have a model with a higher recall to help us effectively identify as many true POIs as possible.

Parameter Tuning and Validation

Tuning the parameters helps get the best performance we can out of the algorithm. A poorly tuned algorithm can lead to excessive run times as in the case of overfitting. An overfit model also picks up a lot of noise that impacts it's ability to perform well with new data. Conversely, an under fit model also performs poorly due to it's inability to model on the training data nor the test data.

Utilizing GridSearchCV I was able to tune the DecisionTree model and achieve the following results:

Training Time: 0.825s
Prediction Time: 0.001s

Performance after tuning:
Accuracy: 0.864
Precision: 0.4
Recall: 0.4
Best Parameters: {'Dtree**min_samples_leaf': 8,**
'Dtreecriterion': 'entropy', 'Dtree__min_samples_split': 3}

Although this improved the results of the DecisionTree algorithm, with a recall of .6 and precision of .5 the Naive_Bayes algorithm still proves to be a wiser choice given that it would lead to more true positives.

The final step of the process was validating the parameters. Validation is a vital step in ensure the best model performance. A poorly tuned model can lead to overfitting which occurs when we do not hold a small sample of data, in this case the model may not know how to react to new data it hasn't seen before. Utilizing cross_validation I withheld 30% of the data while utilizing 70% for training through the train_test_split helper function. This allowed me to randomly split the data into train and test subsets of data therefore giving me the ability to test data unseen by the model in the training phase.

References:

Sources

http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
http://www.dataschool.io/comparing-supervised-learning-algorithms/
http://scikit-learn.org/0.17/modules/cross_validation.html
https://www.cs.cmu.edu/~schneide/tut5/node42.html
http://scikit-learn.org/stable/modules/generated/
sklearn.cross_validation.StratifiedShuffleSplit.html
http://scikit-learn.org/stable/modules/generated/
sklearn.feature_selection.f_classif.html#sklearn.feature_selection.f_classif
http://scikit-learn.org/stable/modules/cross_validation.html
http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.ParameterGrid.html
https://stackoverflow.com/questions/9012487/matplotlib-pyplot-savefig-outputs-blank-image