Enron Fraud detectors using Enron financial data and emails.

Submitted by Prasad Pagade, 2/16/2017

Questions:

1) *Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those?*

   The goal of this project was to create a fraud detection algorithm that predicts POIs (person of interest who have been convicted) from a list of Enron employees. Machine learning is useful as it can quickly process the financial and email features to come with a decent prediction of POI. This algorithm can further be applied to similar companies to identify fraud or spam detection.

   Our dataset is a dictionary of 146 records. The keys are employees names and the values are different financial and email features for that employee. There are 18 POIs labeled in the dataset and 128 non-POIs. There are 21 features associated with each employee.

   I found out couple of features missing for some employees. These were investigated and later treated by replacing the "NaN" to 0. Below is the list compiled:

Features	Missing Values
loan_advances	142
director_fees	129
restricted_stock_deferred	128
deferral_payments	107
deferred_income	97
long_term_incentive	80
bonus	64
shared_receipt_with_poi	60
to_messages	60
from_messages	60
from_poi_to_this_person	60
from_this_person_to_poi	60
other	53
expenses	51
salary	51
exercised_stock_options	44
restricted_stock	36
email_address	35
total_payments	21
total_stock_value	20

   I found out 2 outliers in our data:

1) **Total**: I found this by visualizing the scatterplot between salary and bonus. Turns out the Total was a summary record containing aggregated financial and email data for all the employees. We removed it using the "dictionary.pop()" function.
2) `THE TRAVEL AGENCY IN THE PARK:` This doesn't seem like name of the employee and was removed after listing out all the keys in the data dictionary.

2) *What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values.*

For selecting the features for POI prediction I used scikit learn's "**Select_k_best**". I ended up selecting 6 features that gave the best precision/recall for the final selected machine learning algorithm(GaussianNB). Below are the features and the scores:

Features	K_best scores
exercised_stock_options	24.81
total_stock_value	24.18
bonus	20.79
salary	18.28
fraction_to_poi	16.40
deferred_income	11.45

I used "**min_max_scalar**" to scale the features as email and financial data varies by a large amount and would influence its weight on the algorithm. Hence, it was essential to scale the features so that each features contributes an equal weight towards the POI prediction.
I engineered 2 features **"fraction_from_poi'" "fraction_to_poi"**. The rationale behind creating this feature was to find out what percent of the total messages were sent/received from POIs in the dataset. If the percentage of messages exchanged between employee and POI is high, then employee himself/herself can be a POI.

I ended up selecting "**fraction_to_poi**" in my final feature list. The feature "**fraction_from_poi**" has a low K_score of **3.12**. On calculating the precision/recall using "**fraction_from_poi**" I found that the precision score went down but the recall score jumped to **0.5.** Below table summarizes the accuracy with the 2 features.

Features	K_best scores	Precision	Recall
fraction_to_poi	**16.40**	**0.37**	**0.31**
fraction_from_poi	3.12	0.19	0.51

Since, our requirement was to get a score of atleast 0.3 for precision and recall, I decided to drop "fraction_from_poi" from my final feature list.

3) *What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?*

I ended up using "GaussainNB" algorithm. I tried several other algorithms like Kmeans, SVM, DecisionTree, KNearestNeighbors etc. I got the best precision/recall for GaussianNB algorithm. I ended up using a **Pipeline** and **GridSearchCV** to Scale, select features and run it on the algorithm. Below are my findings:

Algorithm	Precision	Recall	F1 Score
GausianNB	**0.37**	**0.31**	**0.34**
SVM	0.39	0.07	0.12
Decision Tree	0.33	0.26	0.29
KNN	0.25	0.15	0.19

4) *What does it mean to tune the parameters of an algorithm, and what can happen if you don't do this well? How did you tune the parameters of your particular algorithm? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier)*

Parameter tuning is done to adjust the algorithm when it is in training stage. This is the most important phase of machine learning as every optimal combination of parameter can play a crucial role in the final outcome of the algorithm. A fine-tuned algorithm a good mix of bias and variance so that it can predict a new data points with higher accuracy.

My final algorithm(GaussianNB) did not have parameter tuning. Although I did performing tuning on the KBestFeatures to pick the best set of features that gave the best accuracy for GaussianNB.

Another algorithm I tuned but not used as my final selection was – **SVC**. I used GridSerachCV to search for the best combination of parameter values for this algorithm. The following were the parameters tuned for SVC:

- **C**: Penalty parameter of the error term. **Range of values used - [0.1, 1, 2, 4, 6, 8, 10]**
- **Gamma**: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. If gamma is 'auto' then 1/n_features will be used instead. **Range of Gamma used – [0.01, 0.1, 1, 10.0, 50.0, 100.0]**

- **Kernel**: Specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable. If none is given, ‘rbf’ will be used. **Range of Kernels used – [‘rbf’, ‘poly’,’sigmoid’]**

The best combination on parameters for SVC was:

*Pipeline(steps=[('min_max_scaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('select_KBest', SelectKBest(k=5, score_func=<function f_classif at 0x0000000008EA9828>)), ('svc', SVC(**C=10**, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, **gamma=0.05, kernel='rbf'**, max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])*

5) *What is validation, and what’s a classic mistake you can make if you do it wrong? How did you validate your analysis?*

Validation is the process measuring the effectiveness of the algorithm. The process is to split the data so that you can train the algorithm on one half and test the algorithm on the other half. The most common mistake you can do is to train and test on the same data, this will cause overfitting resulting into a less bias algorithm.

I validated my algorithm using the **StratifiedShuffleSplit** method and observing the precision, recall and F1 score of the algorithm. Since the POIs and NON-POIs are very imbalanced, the **StratifiedShuffleSplit** balances the ratio of data points and randomizes the splits so that the algorithm is trained in a more optimal way.

6) *Give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm’s performance*.

The evaluation metrics I used were:

Precision: **0.37**. This means that if our model predicts 100 POIs then 37 out of them are relevant POIs. But the other 63 people are NON-POIs.

Recall: **0.31**. This means that 31% of the times my POI identifier correctly identifies a POI (true positive). But remaining 69% of the times it identifies it identifies a POI as a NON-POI (false negative)

Ideally for this algorithm, we would like to have highest possible recall as we cannot afford to identify a true POI as a NON-POI.