**Enron Submission Free-Response Questions**

**Lekhraj Sharma**

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?  [relevant rubric items: “data exploration”, “outlier investigation”]*

The goal of this project is to identify Enron Employees who may have committed fraud (<https://en.wikipedia.org/wiki/Enron_scandal>) based on the public Enron financial and email dataset. A machine learning based approach is quite useful as we have good amount of data to look at. The size of dataset makes it impractical for a human to laboriously look for patterns which identify our person of interest. But machine learning based algorithms are good at shifting through large volume of data to identify possible patterns which when coupled with human intuition help build a model that can identify our person of interest with good amount of accuracy.

The original dataset used in this project came from Enron Email Dataset (<https://www.cs.cmu.edu/~./enron/enron_mail_20150507.tgz>) collected and distributed by the CALO project (<https://www.cs.cmu.edu/~./enron/>). This dataset was further scrubbed and combined with other inputs (e.g. hand generated list of persons of interest in fraud case). As preprocessing to this project, the Enron email and financial data has been combined into a dictionary (**final\_project\_dataset.pkl**), where each key-value pair in the dictionary corresponds to one person. The dictionary key is the person's name, and the value is another dictionary, which contains the names of all the features and their values for that person.

final\_project\_dataset.pkl : The dataset for the project, more details below

Total Number of Data Points in the dataset: 146

Total Number of Features: 22

These features fall into three major types, namely financial features, email features and POI labels.

**financial features**: ['salary', 'deferral\_payments', 'total\_payments', 'loan\_advances', 'bonus', 'restricted\_stock\_deferred', 'deferred\_income', 'total\_stock\_value', 'expenses', 'exercised\_stock\_options', 'other', 'long\_term\_incentive', 'restricted\_stock', 'director\_fees'] (all units are in US dollars)

**email features**: ['to\_messages', 'email\_address', 'from\_poi\_to\_this\_person', 'from\_messages', 'from\_this\_person\_to\_poi', 'shared\_receipt\_with\_poi'] (units are generally number of emails messages; notable exception is ‘email\_address’, which is a text string)

**POI label**: [‘poi’] (boolean, represented as integer, value 1 signifies person of interest)

We are interested in building an identifier which can determine POI (and non-POI) label based on selected combination of financial and email features.

In several data points, several of financial and email features have missing value (NaN). For example, the person named ‘BAY FRANKLIN R’ does not have value for director\_fees, exercised\_stock\_options, from\_messages, from\_poi\_to\_this\_person, from\_this\_person\_to\_poi, loan\_advances, long\_term\_incentive, shared\_receipt\_with\_poi and to\_messages. All these missing values (NaN) were replaced with zero when converting to numpy array data as part of feature formatting.

Yes, there was an outlier entry in this dataset with key as “TOTAL”. This possibly was artifact of spreadsheet, its total row. This was removed from data dictionary (see poi\_id.py for the code to remove this outlier). From visual inspection of bonus versus salary plot, there were four more possible outliers that stood out. But on closer examination they turned out to be point of special interest as they belonged to person of special interest such as “LAY KENNETH” or “SKILLING JEFFEREY” (first one the founder and second one the COO of Enron). Therefore, they were not removed.

1. *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.  [relevant rubric items: “create new features”, “properly scale features”, “intelligently select feature”]*

The final list of features used in POI identifier are these five features:

**'salary', 'total\_stock\_value', 'shared\_receipt\_with\_poi', 'bonus', 'expenses'**

Started with initial guess of salary as a feature and used a simple and fast Gaussian Naïve Bayes based identifier to evaluate the accuracy. The Gaussian Naïve Bayes is quite fast and allowed to quickly iterate over several features to see which features gave better accuracy. After several such iteration settled on above set of features. Here is quick sample of these iterations:

#features\_list = ['poi','salary']

# Accuracy: 0.25560 Precision: 0.18481 Recall: 0.79800 F1: 0.30011

#features\_list = ['poi','salary', 'total\_payments', "bonus", "exercised\_stock\_options", "total\_stock\_value", "long\_term\_incentive", "expenses", "from\_this\_person\_to\_poi", "from\_poi\_to\_this\_person", "shared\_receipt\_with\_poi" ]

# Accuracy: 0.20487 Precision: 0.11431 Recall: 0.73550 F1: 0.19786

#features\_list = ['poi','salary', 'total\_payments', "total\_stock\_value", "from\_this\_person\_to\_poi", "from\_poi\_to\_this\_person", "shared\_receipt\_with\_poi" ]

# Accuracy: 0.62400 Precision: 0.15543 Recall: 0.41050 F1: 0.22549

#features\_list = ['poi','salary', 'total\_payments', "from\_this\_person\_to\_poi", "from\_poi\_to\_this\_person" ]

# Accuracy: 0.63721 Precision: 0.11387 Recall: 0.22700 F1: 0.15166

#features\_list = ['poi','salary', 'total\_stock\_value', "from\_this\_person\_to\_poi", "from\_poi\_to\_this\_person" ]

# Accuracy: 0.49807 Precision: 0.16321 Recall: 0.60900 F1: 0.25742

#features\_list = ['poi','salary', 'total\_stock\_value', "from\_this\_person\_to\_poi", "from\_poi\_to\_this\_person", "shared\_receipt\_with\_poi" ]

# Accuracy: 0.60064 Precision: 0.15884 Recall: 0.41800 F1: 0.23021

#features\_list = ['poi','salary', "from\_this\_person\_to\_poi", "from\_poi\_to\_this\_person", "shared\_receipt\_with\_poi" ]

# Accuracy: 0.65392 Precision: 0.14342 Recall: 0.21650 F1: 0.17254

#features\_list = ['poi','salary', "bonus", "from\_this\_person\_to\_poi", "from\_poi\_to\_this\_person", "shared\_receipt\_with\_poi" ]

# Accuracy: 0.42050 Precision: 0.16050 Recall: 0.58550 F1: 0.25194

features\_list = ['poi','salary', 'total\_stock\_value', 'shared\_receipt\_with\_poi', 'bonus', 'exercised\_stock\_options' ]

#Accuracy: 0.79414 Precision: 0.29393 Recall: 0.31450 F1: 0.30386

#features\_list = ['poi','salary', 'total\_stock\_value', 'shared\_receipt\_with\_poi', 'bonus', 'expenses', 'loan\_advances' ]

#Accuracy: 0.81493 Precision: 0.36501 Recall: 0.39950 F1: 0.38148

**#features\_list = ['poi','salary', 'total\_stock\_value', 'shared\_receipt\_with\_poi', 'bonus', 'expenses' ]**

**#Accuracy: 0.81457 Precision: 0.36601 Recall: 0.40700 F1: 0.38542**

Did try initially scaling of features using *MinMaxScaler* as we had substituted value of zero for several missing values. But later on removed it when settling down on Decision Tree classifier. Decision tree algorithm is not impacted due to scaling since each feature is only compared against itself and not traded off against other features.

Did engineer a new feature (called ‘**high\_worth’**) as proxy to identifying high net worth individuals. Hypothesis was that a person committing fraud is more likely to have financial gains, therefore he/she likely to be a high net worth individual (e.g. having more than 1 million in salary plus bonus). It simply had boolean value, 1 denoting that this individual had high net worth. But did NOT end up using it since it did not make significant impact on performance of POI identifier.

<Put here the impact of new feature>

1. *What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?  [relevant rubric item: “pick an algorithm”]*

Finally ended up using “Decision Tree” algorithm as it gave the best result. Did try multiple other algorithms starting with simple and fast Gaussian Naïve Bayes, KNeighborsClassifier, to more complex ensemble algorithms such as RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier. Here is quick summary of the results with these algorithms:

#clf = GaussianNB()

#Accuracy: 0.60064 Precision: 0.15884 Recall: 0.41800 F1: 0.23021

#clf = KNeighborsClassifier(n\_neighbors=15, weights='distance')

#Accuracy: 0.84086 Precision: 0.05118 Recall: 0.00650 F1: 0.01154

#clf = AdaBoostClassifier(n\_estimators=20, learning\_rate=2.0)

#Accuracy: 0.75157 Precision: 0.26863 Recall: 0.42900 F1: 0.33038

#clf = RandomForestClassifier(n\_estimators=10, max\_depth=None, min\_samples\_split=1, random\_state=0)

#Accuracy: 0.83693 Precision: 0.32290 Recall: 0.12900 F1: 0.18435

#clf = GradientBoostingClassifier(n\_estimators=100, learning\_rate=1.0, max\_depth=1, random\_state=0)

#Accuracy: 0.82607 Precision: 0.35662 Recall: 0.27050 F1: 0.30765

#clf = tree.DecisionTreeClassifier(random\_state=0)

#Accuracy: 0.81779 Precision: 0.37684 Recall: 0.42150 F1: 0.39792

As can be seen from above numbers, Decision Tree classifier outperformed in balanced combination of accuracy, precision and recall scores. This is also reflected in its high F1 score.

1. *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).  [relevant rubric item: “tune the algorithm”]*

Initially did try to tune parameters of various algorithms (e.g. KNeighbors, AdaBoost, Decsion Tree, etc.) by hand by reading their documentation. For example tried to very number of estimators and/or learning rate for AdaBoost and saw if it helped in boosting the performance. Tried similar tuning with algorithms as well. Since Decision Tree started to give better results even with default settings, spend more time tuning its parameters. Based on documentation, tried to vary by hand its parameter such as max\_depth, min\_samples\_split and random\_state.

<Include final best parameter values for Decision Tree>

Later on did use GridSearchCV to see if it could help in finding the values which better the above performance gained by hand selection.

#parameters = {'criterion':('gini', 'entropy'), 'splitter':('best', 'random'), 'random\_state':[0,10], 'min\_samples\_split':[1,8]}

#clf\_grid = tree.DecisionTreeClassifier()

#clf = grid\_search.GridSearchCV( clf\_grid, parameters)

# Accuracy: 0.80629 Precision: 0.27179 Recall: 0.21200 F1: 0.23820

# clf.best\_params\_ : {'min\_samples\_split': 1, 'splitter': 'random', 'random\_state': 0, 'criterion': 'gini'}

As can be seen from above, while it came close to values selected by hand but did not better the performance of hand tuned parameters, so ended up using hand tuned parameters only.

1. *What is validation, and what’s a classic mistake you can make if you do it wrong? How did you validate your analysis?  [relevant rubric item: “validation strategy”]*

Validation is art of splitting the dataset in two parts, training and test data, such that smaller training dataset we use to develop and tune our identifier gives best learning results on larger test data set. Smaller training dataset allows us to quickly iterate our features, algorithm and its parameters and check for any potential issues such overfitting, scaling of parameter issues, outliers, etc.

One classic mistake is of **overfitting** to training data. This can happen when we either end up training with test data (use test data in fit for classifier algorithm instead of training data) or end up choosing much larger training dataset (compared to test dataset) or select too many features. When we end up overfitting, our identifier will not generalize well to real test data and end up delivering bad performance on test data.

Ended up using cross validation to split 1/3rd of dataset randomly as training data and rest as test data.

#from sklearn.cross\_validation import train\_test\_split

#features\_train, features\_test, labels\_train, labels\_test = \

# train\_test\_split(features, labels, test\_size=0.33, random\_state=42)

End up using the training dataset to fit to classifier and user test dataset to predict the performance of identifier.

#clf = clf.fit(features\_train, labels\_train)

#pred = clf.predict( features\_test )

#print "Accuracy:", accuracy\_score(labels\_test, pred), "Precision:", precision\_score(labels\_test, #pred), “Recall:", #recall\_score(labels\_test, pred)

1. *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. [relevant rubric item: “usage of evaluation metrics”]*

The evaluation metrics corresponding to our final identifier (built on Decision Tree algorithm) are accuracy, precision, recall and F1-score.

**#Accuracy: 0.81779 Precision: 0.37684 Recall: 0.42150 F1: 0.39792**

While **accuracy** of around **81%**, therefore 81% of total data appoints were classified correctly, is good, it may not give true picture of performance on our kind of skewed dataset. In our dataset, there are only few persons of interest (POI) compared to large number of non-POI. In these kind of skewed distribution, it can be easy to predict non-POI and still get good accuracy but still not solve the problem of identifying the real POIs (e.g. an identifier which just returns non-POI always can still have high accuracy!).

The metrics **precision** and **recall** are better measure of performance for imbalanced dataset like ours. A **precision score** of around **37%** means we have reasonable confidence that identified POI is likely to be a real POI and not a false alarm. A **recall score** of around **42%** means we have reasonable confidence that our identifier will correctly identify a POI if it is given a real POI, and it is less likely to flag a non-POI wrongly as POI.

Having reasonable **F1 score**, around **39%**, indicates our identifier is balanced identifier not overly skewed to either reporting non-POIs as POI or missing a real POI. Therefore, if our identifier finds a POI than that person is almost certainly be a POI, and if the identifier does not identify someone as POI, then they are almost certainly not a POI.

In our case of identifying persons of interest who may have committed the fraud in Enron fraud case, we may want to prefer a higher precision score as we wrongly do not want to identify someone as POI. It is Ok to let go of few real POIs rather than raise a false alarm by wrongly identifying a doubtful case as POI. As they say “a person is innocent unless proven guilty”!