Scott Burns
Udacity – Data Analyst Nanodegree
Project 4: Identify Fraud from Enron Email

Enron Submission Free-Response Questions

Section 1: Summary

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"]

Goals

The goal of this project is to determine whether features of Enron employees' email correspondence and compensation could be used to accurately predict their participation in fraud, or at least their status as a fraud suspect, as determined by traditional investigative methods.

Machine learning could be helpful in identifying the employee characteristics that are indicative of fraud, enhancing an investigation by the parsing of enormous amounts of data. Drawing on available computational power, we can test hypotheses about email and compensation evidence at a larger scale than possible using other approaches to investigation. Human intuition is an important part of the process, but machines' iterative parsing of available data can help us formulate, evaluate and discard hypotheses more rapidly than we could through more manual testing.

Outliers

The first update I made to the 'poi_id.py' file, was to include all available features in the feature list. Initially I felt that all features could plausibly be associated with an employee's status as a POI, and seemed worth exploring. Furthermore, I recognized that I could drop less useful features using functions like K-Best or consolidate features with PCA as needed.

Applying my own intuition to the dataset, I looked at potential outliers in the email records, which might hinder us from accurately predicting relationships between email features and fraudulent behavior.

First I decided to plot all financial variables in a grid against salary, just to get a better sense of their distribution. Each subplot showed at least one massive outlier, which I suspected was a 'TOTAL' column, as we discovered in lesson 7 of the Intro to Machine Learning course.

I inserted a function to remove the total item, as we had for lesson 7, and plotted the financial features again.

Then, I drew on Sebastian's heuristic for handling outlier, as explained in lesson 7, to 'cleanse' the dataset. That is, I fitted various compensation features on employees' salaries with a linear

regression, then removed the top 10% of data points by their absolute residual of actual feature value minus predicted feature value. I tried the process iteratively over two variables I thought may be the most useful. The removal of several non-POI employees with large residuals yielded updated scatterplots with a relatively low overall number of striking outlier points.

After several trials, removing outliers based on predictions of correlation with salary, I ultimately decided to remove outliers for algorithm training based on 'bonus' and 'exercised_stock_options' feature values, and how each value differed from a projection based on the user's salary.

After removing my outliers for algorithm, the total number of employees (datapoints) in the training dataset fell to 125 from 146 initially (POI outliers were not removed – per best practice and reviewer feedback).

For each employee datapoint in the trimmed set, some attributes were typically missing, with the 'total_stock_value' being the most complete, with 105 datapoints including showing values for this feature. As another example, only 34 datapoints showed values for 'deferral_payments', as one of the more incomplete features.

Section 2: Features

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.

Initial broad feature list

I initially started with all the given features in my model, deciding that they each had plausible correlation with an Enron employee's POI status. After observing a few generated scatterplots of variables versus salary values for employees, re-assessing features' theoretical impact on classification, I decided to drop the features 'directors_fees', 'restricted_stock_deferred' and 'loan_advances'. Following instructions from my first review, I also examined the 'total_payments' feature and noticed that it actually seemed to incorporate 'total_stock_value', regardless of whether the stock had been exercised, which seemed fundamentally incorrect. As a result I decided to remove it from the feature set as well.

Beyond this, I decided that another email-related feature implicit in the dataset, but not directly calculated, may be helpful in improving classification. In the dataset, we have 'to_messages' and 'from_this_person_to_poi', as well as 'from_messages' and 'to_this_person_from_poi'. I hypothesized that the most predictive indicator of being a POI was likely to be the emails received from or sent to POIs as a percentage of total emails received or sent.

Creating 'percent_of_emails' features

Accordingly, I included the features 'percent_of_emails_from_poi' and 'percent_of_emails_to_poi' in the features list. Initially, I played with removing other email-related features due to their correlation with the new features, but found that their inclusion increased recall and precision in subsequent classification test runs, and that the features 'to_messages' and 'from_this_person_to_poi' were also routinely included in feature sets as I decided to KBest features selection tool.

Rescaling features

Because I was incorporating features with wide differing absolute variances, I also decided that it would be helpful to rescale my features, using MinMaxScaler in the preprocessing module of Scikit learn.

I created a deep copy of the 'data_dict' dictionary, then for all features in my features list (other than 'POI', rescaled the feature according to its position relative to the range of values for the feature in the dataset – adding it to a corresponding value entry in my new copied data dictionary, which I entitled 'rescaled_data_dict'.

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"]

K-Best features

After some trial and error, I decided that filtering features automatically with Scikit-learn tools would be helpful in increasing my model's predictive power, and ultimately decided on capping the total number of features selected at 10 (which I found to yield the best recall and precision in my classifiers). As a note, I used chi-2 tests of feature significance when filtering features with the SelectKBest function from the Scikit-learn feature_selection module.

The 10 features automatically selected using the K-Best process were:

['salary', 'deferred_income', 'bonus', 'total_stock_value', 'expenses', 'exercised_stock_options', 'to_messages', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi', 'percent_of_emails_to_poi']

Other ideas (for future exploration)

Aside from the given features I felt the following features could potentially be associated with fraud:

1. **exclusive_poi_exchange**

This feature would be a count of emails sent exclusively to a POI, or emails received, with no cced addressees, from a POI. The feature is similar to both the from_poi_to_this_person and the from_this_person_to_poi, but it excludes emails where the author had addressed a group of people or received a message with many others included. I hypothesize that POIs may have been more likely to leave other recipients out of email traffic if discussing matters related to fraudulent activity.

2. **exclusive_exchange**

Thinking about my hypothesis above, I decided that in general, an employee participating in questionable business practices may be more likely to focus his correspondence, taking pains to limit the number of people on his email chains. I decided it would be useful to create a general record of total emails sent and received by each employee in which there were only two parties in email chain.

3. **spv_mention**

From what I understand of the Enron fraud case, most of the company's schemes to illegitimately present its financial position related to the use of 'special purpose vehicles' or SPVs. I decided to test whether the presence of the strings 'special purpose vehicle' or 'spv' was associated with an author being a POI. This variable would track the number of emails sent by each employee with either of these strings.

I ultimately decided to not implement these features, as I didn't have a full list of the email addresses associated with each individual name in the keys of the data_dict dataset. I contemplated trying to build this set, but finally decided that this was going to require more time than I have available to spend on the project. This is an area I'd like to revisit and explore further in the future.

Section 3: Algorithm

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"]

I ultimately used an AdaBoost classifier to identify POIs from the dataset.

I selected this algorithm as it was demonstrating the highest F1 scores in the given tester from a list of classifiers I tried, including SVM, Naïve Bayesian, Decision Trees, RandomForest and K-nearest Neighbors.

When testing the algorithm with the provided 'poi_tester.py' script, my F1 scores for predictions using this estimator were around 0.59, as seen in tester output below.

Evaluation of AdaBoost Classifier in 'poi_tester.py'

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=1, random_state=None)

Accuracy: 0.89585    Precision: 0.75313   Recall: 0.48050
F1: 0.58669  F2: 0.51800

Total predictions: 13000   True positives:  961  False positives:  315
False negatives: 1039 True negatives: 10685
```

Section 4: Tuning

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"]

'Tuning parameters' refers to adjusting the arguments passed into classifier algorithms in order to ensure that models fitted from training data achieve the desired predictive power, striking the right balance between variance and bias in the model.

To clarify, bias is the tendency of a model to 'under-fit', not generalizing from training data and not providing useful predictions of test data. The degree of bias present in a model is reflected in the 'Recall' score for its predictions, with Recall being the number of elements in a dataset correctly classified to a category divided by the total number of elements that should have been classified to that category.

Variance is the tendency of a model to 'overfit', generalizing too extensively from training data and unable to accurately predict based on test data that differ from training. The degree of variance present in a model is reflected in the 'Precision' score for its predictions, with Precision being the number of elements in a dataset correctly classified to a category divided by the total number of elements that were classified to that category (correctly or incorrectly).

In my exercises I tuned a number of parameters in my tested classifiers, including the criteria for my DecisionTreeClassifier and RandomForestClassifier (switching from 'gini' to 'entropy', the kernel and C value for my SVC Classifier ('rbf' to 'linear' and 1 to 100), as well as other parameters, such as n_neighbors in the K-Neighbors Classifier and min_sample_split in my DecisionTree Classifier. I found the variations to have occasionally substantial impact, but still

found that none of this tuning allowed other classifiers to surpass the predictive power of my AdaBoost classifier.

Also, following optional suggestions from my reviewer, I used GridSearchCV to optimize my parameters for the selected AdaBoost model. In GridSearch, I tried a range of parameter values of 'n_estimators' from 1 to 10 and for 'learning_rate' from 1.0 to 5.0. Based on this, I found that parameter values on the low ends of each tested range maximized my 'F1' score, and decided to use these for both.

Section 5: Validation

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 the use of separate data to train and evaluate the performance of a classifier. A classifier is only as useful as its ability to correctly classify new data outside those it used in training, so in building algorithms, we must be careful to partition our dataset for training and testing.

If a classifier is not correctly cross-validated, it is possible for it to yield a seemingly predictive model, that may not actually be useful at all for classifying outside the current sample. Overfitting is a likely pitfall of insufficient cross-validation.

To validate my classifiers, I split the dataset using the Scikit-learn module 'cross_validation.train_test_split', keeping 30% of elements from the total dataset for testing. Using this approach allowed me to assess how accurate my classifier's predictions were on data outside the training sample.

I also assessed how my classifiers' predictive power shifted with changes to my split of training and testing data. After validating using test_size values from 0.1 to 0.5, I found that 0.3 consistently helped me identify the most predictive models.

Section 6: Evaluation

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"]

Key evaluation metrics can be seen in the output from the 'poi_tester.py' script below – as run to assess the AdaBoost algorithm I ultimately selected.

In the output, model Precision is 0.75 with Recall 0.48. To confirm descriptions from earlier in Section 4, Recall is the likelihood that an actual Enron POI was correctly classified as a POI by

my algorithm, and Precision is the likelihood that an employee classified as a POI by my algorithm was actually a POI.

F1 is a measure of predictive power that incorporates both Precision and Recall – for which my classifier generated a score of 0.59.

Evaluation of AdaBoost Classifier in 'poi_tester.py'

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=1, random_state=None)

Accuracy: 0.89585    Precision: 0.75313   Recall: 0.48050
F1: 0.58669  F2: 0.51800

Total predictions: 13000   True positives:  961  False positives:  315
False negatives: 1039 True negatives: 10685
```

Section 7: References

Section 1:

http://stackoverflow.com/questions/7941207/is-there-a-function-to-make-scatterplot-matrices-in-matplotlib
http://matplotlib.org/examples/pylab_examples/subplot_demo.html
https://plot.ly/matplotlib/subplots/
http://matplotlib.org/api/pyplot_api.html?highlight=subplot#matplotlib.pyplot.subplot

Section 2:

http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html#sklearn.feature_selection.SelectKBest
http://scikit-learn.org/stable/modules/feature_selection.html
https://discussions.udacity.com/t/creating-new-feature-in-dataset/8082/3
https://docs.python.org/2/library/copy.html

Section 3:

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
http://scikit-learn.org/stable/modules/neighbors.html#classification
http://scikit-learn.org/stable/modules/tree.html#classification

Section 4:

https://www.udacity.com/course/viewer#!/c-ud120-nd/l-2286088539/m-2466048540
https://www.udacity.com/course/viewer#!/c-ud120-nd/l-2252188570
https://www.udacity.com/course/viewer#!/c-ud120-nd/l-2258728540/m-2428648555
http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation
http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html

Section 5:

https://www.udacity.com/course/viewer#!/c-ud120-nd/l-2960698751/m-2944138932

Section 6:

https://www.udacity.com/course/viewer#!/c-ud120-nd/l-2945338635