Michael Christensen
Machine Learning Project
May 16, 2016

Enron Submission Free-Response 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?

Enron Corporation was an American energy, commodities, and services company based in Houston, Texas. The company filed for bankruptcy in December 2001 after massive accounting fraud was discovered. The Enron scandal as it has been come to be known is an example of deliberate corporate malfeasance. The goal of this project was to utilize and tune machine learning algorithms to identify Enron Employees who may have committed fraud based on the public Enron financial and email (F+E) dataset. Here, these employees are known as a person of interest (POI). POIs are defined as individuals who were indicted, reached a settlement, or testified in exchange for prosecution immunity. In the F+E dataset used in this project, information was provided across many different categories (features) for 145 former Enron employees. Some of the financial features available for individuals include exercised stock options, total stock value, bonus, salary, and deferred income. Aggregate email statistics like the number of emails to/from a POI are also included in the dataset. Features from the combined financial and email datasets were used as features for prediction.

Outliers were detected and investigated by doing a univariate analysis of all features. Data was plotted and investigated with boxplots. The item "TOTAL" was removed from the F+E dataset as it was simply a summation of the financial features. "LOCKHART EUGENE E" was also removed as this record contained no actual data. Other outliers apparent in the financial data were not removed as some were POIs and removing these would negatively impact the final machine learning algorithm. Financial data for Robert Belfer and Sanjay Bhatnagar were also updated to match the enron61702insiderpay.pdf. Totals computed for each financial feature across all records matched those in the enron61702.pdf. After removing/updating records, the remaining dataset had 144 entries. Of these, 18 individuals were labeled as a POI while the remaining 126 were not. The entire dataset (after cleaned/updated) had 2880 data points of which 54.89% (1581/2880) were non-empty. I created a function to explore data coverage for each original feature (19 features: excluding poi and email_address) in the dataset for three groups: (1) combined POI and non-POI (2) non-POI (3) POI. This information is visualized in the table below.

Financial Data Coverage (fraction with data, count)			
	Combined	NON-POI	POI
bonus	(0.56, 81)	(0.52, 65)	(0.89, 16)
deferral_payments	(0.26, 37)	(0.25, 32)	(0.28, 5)
deferred_income	(0.34, 49)	(0.3, 38)	(0.61, 11)
director_fees	(0.1, 15)	(0.12, 15)	(0.0, 0)
exercised_stock_options	(0.69, 100)	(0.7, 88)	(0.67, 12)
expenses	(0.67, 96)	(0.62, 78)	(1.0, 18)
loan_advances	(0.02, 3)	(0.02, 2)	(0.06, 1)
long_term_incentive	(0.45, 65)	(0.42, 53)	(0.67, 12)
other	(0.63, 91)	(0.58, 73)	(1.0, 18)
restricted_stock	(0.76, 110)	(0.74, 93)	(0.94, 17)
restricted_stock_deferred	(0.12, 17)	(0.13, 17)	(0.0, 0)
salary	(0.65, 94)	(0.61, 77)	(0.94, 17)
total_payments	(0.86, 124)	(0.84, 106)	(1.0, 18)
total_stock_value	(0.87, 125)	(0.85, 107)	(1.0, 18)

Email Data Coverage (fraction with data, count)			
	Combined	NON-POI	POI
from_messages	(0.6, 86)	(0.57, 72)	(0.78, 14)
from_poi_to_this_person	(0.6, 86)	(0.57, 72)	(0.78, 14)
from_this_person_to_poi	(0.6, 86)	(0.57, 72)	(0.78, 14)
shared_receipt_with_poi	(0.6, 86)	(0.57, 72)	(0.78, 14)
to_messages	(0.6, 86)	(0.57, 72)	(0.78, 14)

(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.

Ultimately, the list of features I selected for my POI identifier included: (1) exercised_stock_options, (2) expenses, and (3) from_this_person_to_poi. These features had the highest importance values from the DecisionTreeClassifier (provided-below) and when used alone produced the best evaluation metric scores (f1-score, precision, recall). When testing other classifiers (Gaussian Naïve Bayes, Adaboost, Support Vector Machine), I did not pre-select features, but rather used the entire feature list and SelectKBest in the GridSearchCV pipeline. By including the entire feature list I allowed my algorithm to select the K-best

features (using f-score). This selection process was carried out in each of the 1000 cross-validation loops I ran when finding optimal parameter tunes.

Prior to training and testing my machine learning algorithm I used MinMaxScaler to transform the data into a 0 to 1 range. This scaling helped to account for the wide ranges in the data and the different feature units. Units used in the dataset varied from millions of dollars to numbers of emails sent/received. Feature scaling isn't necessary for all machine learning algorithms (i.e., Decision Tree), but applying it up-front allowed for greater flexibility and improved performance in testing other algorithms where feature scaling does have an impact (i.e., SVM).

I created two additional variables: fraction_from_poi and fraction_to_poi. The first additional feature (fraction_from_poi) was found by dividing the number of emails an individual received from a POI by the total number of emails the individual received. The second additional feature (fraction_to_poi) was found by dividing the number of emails an individual sent to a POI by the total number of emails the individual sent. My hypothesis was that if an individual sent/received a larger fraction of emails to/from a POI then it likely they too are a POI. Ultimately, however, including these additional features only worsened the performance of the DecisionTree algorithm and thus I did not include them in my final list.

fraction_from_poi = (from_poi_to_this_person / to_messages),
fraction_to_poi = (from_this_person_to_poi / from_messages)

SelectKBest Scores	
total_stock_value	22.78211
exercised_stock_options	22.61053
bonus	21.06
salary	18.5757
deferred_income	11.56189
long_term_incentive	10.07245
total_payments	9.380237
restricted_stock	8.95854
shared_receipt_with_poi	8.746486
loan_advances	7.24273
expenses	5.550684
from_poi_to_this_person	5.344942
fraction_to_poi	5.20965
other	4.219888
fraction_from_poi	3.210762
from_this_person_to_poi	2.426508
director_fees	2.112762
to_messages	1.698824
restricted_stock_deferred	0.761863
deferral_payments	0.221214
from_messages	0.164164

DecisionTreeClassifier Scores	
exercised_stock_options	0.2
expenses	0.131701
from_this_person_to_poi	0.119164
other	0.117413
total_payments	0.112875
bonus	0.109452
shared_receipt_with_poi	0.100048
from_poi_to_this_person	0.056437
salary	0.05291
deferral_payments	0
deferred_income	0
director_fees	0
loan_advances	0
long_term_incentive	0
restricted_stock	0
restricted_stock_deferred	0
total_stock_value	0
from_messages	0
to_messages	0
fraction_from_poi	0
fraction_to_poi	0

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

I considered 4 different classification algorithms for the project: Gaussian Naive Bayes, Decision Tree, AdaBoost, and Support Vector Machine. The Decision Tree Classifier with parameters (class_weight=None, criterion='entropy', max_depth=None, max_features=None, max_leaf_nodes=None, min_samples_leaf=1, min_samples_split=20, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best') performed the best of all of the algorithms tested based on evaluation metrics used in this project (Precision: 0.67126, Recall: 0.41450, F1: 0.51252). This algorithm was run using exercised_stock_options, expenses, and from_this_person_to_poi features which represented the 3 most important features using the DecisionTreeClassifier(). The next most performant algorithms in order from best to worst were as follows: Adaboost (Precision: 0.44557, Recall: 0.30900, F1: 0.36492), GaussianNB (Precision: 0.20939, Recall: 0.37450, F1: 0.26860), and SVM (Precision: 0.51448, Recall: 0.15100, F1: 0.23348).

(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)

Tuning the parameters of an algorithm refers to the process of adjusting its inputs across some range of values to find the combination that optimizes overall performance. If tuning is not well done than the algorithm will not perform as well as it could.

For this project, I used a pipeline of operations inside of GridSearchCV (with stratified shuffle split cross-validation) to systematically tune parameters. Using a pipeline allows multiple operations to be performed to data all at once. Stratified shuffle split cross-validation was used to randomly split the data into 90% training/10% testing portions from which performance metrics were gathered from the testing data. This process was performed 1000 times and the scores averaged.

For GaussianNB, SVM, and Adaboost, SelectKBest was utilized to determine the optimal number of features to use for analysis. For Decision Tree the 3 most important features (as determined from the feature_importances_ class) were used. I determined this by running a simple loop on the full list of sorted (highest to lowest) feature_importances_ and chose the number which maximized the f1 score. The parameters tuned for each algorithm / pipeline operator are shown below:

GaussianNB	Paramater Values	SVM	Paramater Values
SelectKBest	[1,2,3,…21]	SelectKBest	[5,8,10,12,14,16,18,19,20,21]
		C	[1,10,100]
		kernel	['linear','rbf']

Adaboost	Paramater Values	DecisionTree	Paramater Values
SelectKBest	[5,8,10,12,14,16,18,19,20,21]	Top # feature_importances	3
n_estimators	[50,100,200]	min_samples_split	[2, 5, 10, 20, 30, 50]
		criterion	['gini', 'entropy']

(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 of training and testing a machine learning algorithm to see how it performs on unseen data (testing data). A classic mistake is training and testing on the same data and getting what's perceived as very high scoring metrics. In this "over-fitting" case, it is likely that the model will perform poorly when given unseen data. This issue can be avoided through cross-validation. In this project I used the StratifiedShuffleSplit function to randomly split data into training and testing sets (90% / 10%) a pre-defined number of times. This function was used during both parameter optimization in GridSearchCV (model selection) and model validation (in tester.py). Randomly and repeatedly (1000 times) splitting the data in this way helps to avoid another classic mistake which is to not sufficiently shuffling and splitting the training/testing data. This proves to be an issue when a dataset is organized in a non-random fashion.

(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.

Given that there are relatively few POIs in the F+E dataset, I use precision, recall, and the F1-score rather than accuracy to determine the performance of each chosen machine learning algorithm. Accuracy is not well-suited for this unevenly distributed data in that very high accuracy can be attained simply by always guessing an individual is a non-POI because there are so many non-POIs compared with POIs. Precision is calculated by dividing the number of times the algorithm positively identifies a POI (true positive) divided by the total number of positive identifications (true positive + false positive). High precision value means POIs identified by the algorithm tended to be actual POIs, while a low value means there were more false positive identifications (non-POIs flagged as POIs). In this way, precision can be seen as a measure of quality. Recall, on the other hand is a measure of quantity. It is calculated by dividing the number of true positives by the sum of the true positives and false negatives (POIs who are classified as non-POIs). High recall means that if there were POIs in the test set, the algorithm would do a good job of identifying them. Low recall in the context of this project would mean POIs in the test dataset failed to be identified. The F1-score considers both precision and recall and can be thought of as a weighted average of the two. The score is computed by multiplying the product of precision and recall by two and dividing the subsequent value by the sum of precision and recall.

In the table below I provide average performance metrics over 1000 randomized cross-validations splits for the 4 classification algorithms tested. GridSearchCV was utilized with StratifiedShuffleSplit to tune and optimize model parameters to achieve the highest F1-score. Although I ultimately chose the most performant algorithm (Decision Tree), Adaboost also met the precision and recall 0.3 threshold. The DecisionTree algorithm using exercised_stock_options, expenses, and from_this_person_to_poi features was the clear winner for me as it had the highest precision and recall of all the machine learning algorithms I tested.

	Precision	Recall	F1-Score
DecisionTree	0.67	0.41	0.51
Adaboost	0.44	0.3	0.36
GaussianNB	0.2	0.37	0.26
SVM	0.51	0.15	0.23