Matthew Bellissimo

Enron Submission Free-Response Questions

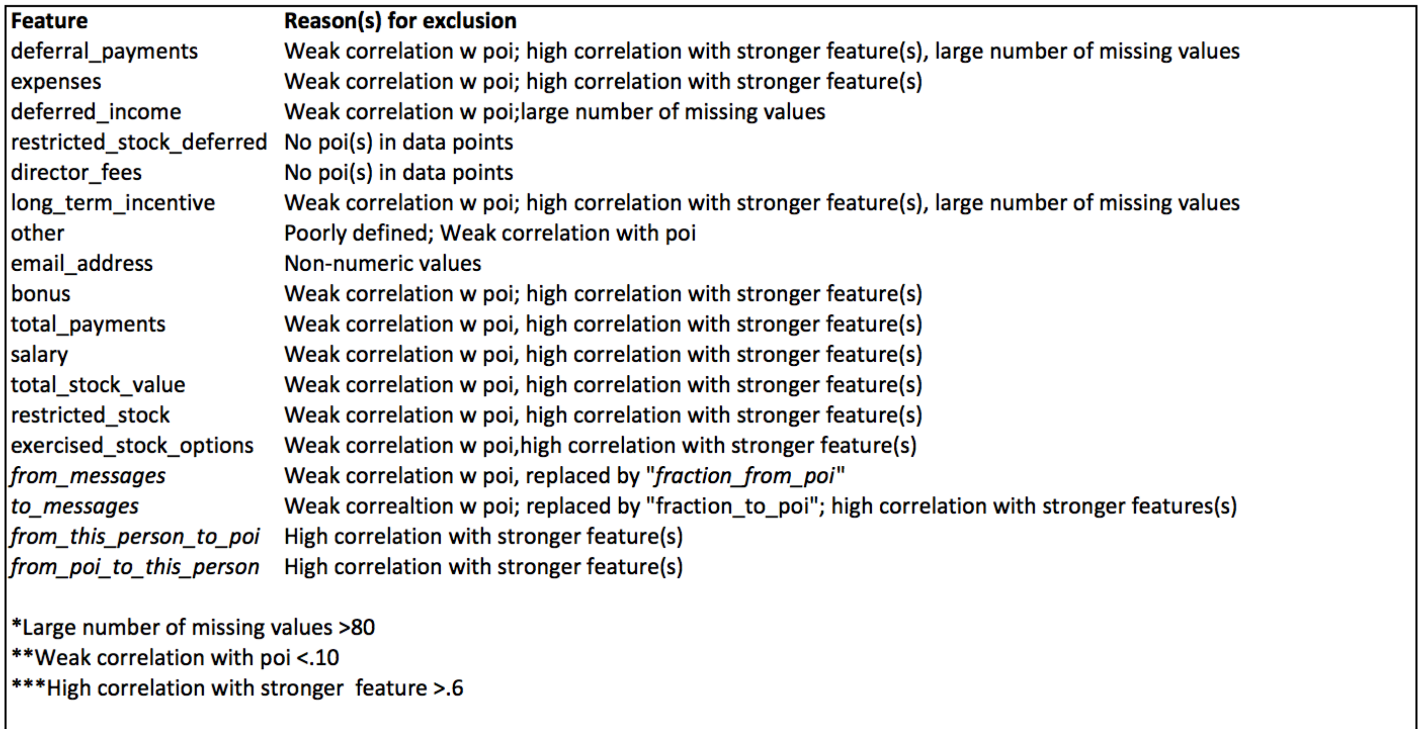
1. *Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those?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

   The goal of this project was to apply machine learning techniques to the Enron Dataset, in order to create a classifier that could predict whether, based off a set of features, an employee was a person of interest (POI) or not. The dataset in question contained 146 data points and 21 original features. 18 out of the 146 data points were POI's, and 1 point was an extreme outlier that I ended up removing, due to the fact that it was a spreadsheet quirk and it's inclusion was an error. While there were some additional outliers in the dataset set (for instance, employees with bonuses over 5 million dollars, and a salary of over 1 million dollars), they represented valid data points, and actually included some POI's, so they were left in. Many features also contained a number of missing values, a representation of how many values were missing in each feature is broken down below

Feature Name	Missing Values	Missing Values-POI
loan_advances	142	17
director_fees	129	18
restricted_stock_deferred	128	18
deferral_payments	107	13
deferred_income	97	7
long_term_incentive	80	6
bonus	64	2
to_messages	60	4
from_this_person_to_poi	60	4
from_messages	60	4
shared_receipt_with_poi	60	4
from_poi_to_this_person	60	4
other	53	0
expenses	51	0
salary	51	1
exercised_stock_options	44	6
restricted_stock	36	1
email_address	35	0
total_payments	21	0
total_stock_value	20	0
fraction_from_poi	0	0
poi	0	0
fraction_to_poi	0	0

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

The final set of features I ended up using for my POI identifier were "loan_advances", "fraction_to_poi", "shared_receipt_with_poi", and "fraction_from_poi". These features were selected after passing an elaborate filter process that was equal parts discretion and statistical logic. The first step in the process was to exclude "email_address", which was a non-numeric feature and offered no information gain. Next I set up a correlation matrix to see which features the most strongly correlated with "poi", which was the class I was trying to predict, and to see which features were strongly correlated with each other. Typically a good feature set contains features that are highly correlated with class, yet uncorrelated with each other, so during my manual feature selection process, I look at which features were highly correlated with each other, if there was a feature that was strongly correlated with another feature(s) that was stronger than it, in terms of it's correlation with the "poi" class, I excluded it. Another consideration that I took into account during feature selection was whether any features had been made redundant through the creation of new features; this was the case with "from_poi_to_this_person","to_messages","from_this_person_to_poi", and "from_messages". I had created "fraction_from_poi", and "fraction_to_poi" out of these original features because I thought that looking at the proportion of messages to and from POIs would do a better job of providing context towards an association with guilty/suspected parties, than just looking at the absolute number of emails alone, which I believed could be misleading. While the manufactured features were not highly correlated with their original components, they essentially represented a more accurate version of the same information. Additionally they were MUCH more correlated than the original components to the "poi" class, because of this, I chose to interpret the new features as an upgrade, and just excluded the old ones. Below are a list of the excluded features, and a short explanation about why each feature was excluded.



Feature	**Reason(s) for exclusion**
deferral_payments	Weak correlation w poi; high correlation with stronger feature(s), large number of missing values
expenses	Weak correlation w poi; high correlation with stronger feature(s)
deferred_income	Weak correlation w poi;large number of missing values
restricted_stock_deferred	No poi(s) in data points
director_fees	No poi(s) in data points
long_term_incentive	Weak correlation w poi; high correlation with stronger feature(s), large number of missing values
other	Poorly defined; Weak correlation with poi
email_address	Non-numeric values
bonus	Weak correlation w poi; high correlation with stronger feature(s)
total_payments	Weak correlation w poi, high correlation with stronger feature(s)
salary	Weak correlation w poi, high correlation with stronger feature(s)
total_stock_value	Weak correlation w poi, high correlation with stronger feature(s)
restricted_stock	Weak correlation w poi, high correlation with stronger feature(s)
exercised_stock_options	Weak correlation w poi,high correlation with stronger feature(s)
from_messages	Weak correlation w poi, replaced by "*fraction_from_poi*"
to_messages	Weak correaltion w poi; replaced by "fraction_to_poi"; high correlation with stronger features(s)
from_this_person_to_poi	High correlation with stronger feature(s)
from_poi_to_this_person	High correlation with stronger feature(s)

*Large number of missing values >80
**Weak correlation with poi <.10
***High correlation with stronger feature >.6

Once I had excluded all the features I knew that I would not use, the next step was to use feature selection to figure out the optimal combination/number of my remaining features. Since I was using a decision tree as my classifier, and decision trees did not require scaling to be effective, I decided to not scale my data for the sake of processing time (Decision Tree: "*is feature x_i >= some_val?").* I ended up using the SelectKbest function when picking my features, because if I was playing around with the number of features to use, I want to make sure the number that I looked at in each test iteration of my program would be picking the best features possible. After iterating through all the possible k values, it turned out that the optimal k value for my tuned classifier was equal to 4. This was great, but it given that I had only 4 features remaining after my manual exclusions, I was curious as to whether k would increase if my feature set had been larger. I also wondered as to what the result would be for my metrics if I had not excluded the *weakly correlated*, and *weaker multicollinear* features from my feature set. To this end, I decided to also look to see what effect different k values would have on my tuned classifier's scores if I had NOT excluded any features.

Features Scores-Full Set

Feature	Select K Best Score	Correlation with poi class
fraction_to_poi	16.87	0.33
shared_receipt_with_poi	8.90	.23
from_poi_to_this_person	5.44	.17
fraction_from_poi	3.29	.15
loan_advances	2.51	.56
from_this_person_to_poi	2.47	.11
to_messages	1.75	.06
director_fees	0.54	NA
total_payments	0.35	.04
deferral_payments	0.24	-.08
exercised_stock_options	0.23	.05
deferred_income	0.22	.01
total_stock_value	0.17	.03
from_messages	0.16	-.07
bonus	0.08	-.01
other	0.07	-.01
restricted_stock	0.03	0.00
long_term_incentive	0.02	-.02
expenses	0.01	-.04
restricted_stock_deferred	~0	NA
salary	~0	-.03

Full Feature Set

SelectKBest Value	Precision Score	Recall Score	F1 Score
K=1	0.53534	0.46200	0.49597
K=2	**0.47419**	**0.58800**	**0.52500**
K=3	0.50717	0.47750	0.49189
K=4	0.54179	0.47650	0.50705
K=5	0.53534	0.46200	0.49597
K=6	0.54530	0.48450	0.51311
K=7	0.54123	0.46600	0.50081
K=8	0.52910	0.49550	0.51175

K=9	0.46289	0.50200	0.48165
K=10	0.38354	0.27500	0.32033

Reduced Feature Set

SelectKBest Value	Precision Score	Recall Score	F1 Score
K=1	0.28149	0.54300	0.37078
K=2	0.34866	0.40200	0.37343
K=3	0.36961	0.39400	0.38141
K=4	**0.51585**	**0.78100**	**0.62132**

Final selection K=4

Feature Name(s)	KBest Feature Score(s)	DT Feature Importance(s)
fraction_to_poi:	14.400092	0.51178343
shared_receipt_with_poi:	7.167587	0.3850618
fraction_from_poi:	2.337903	0.0985505
loan_advances:	2.180175	0.00460426

In all cases, the metrics for the reduced feature set were significantly better than the best scores for the full set of features. This made me exceedingly happy since the results for the reduced feature set were much cleaner.

Features	poi	loan_advances	fraction_to_poi	shared_receipt_with	fraction_from_poi
poi	1.00	0.56	0.33	0.23	0.15
loan_advances	0.56	1.00	0.20	0.29	-0.46
fraction_to_poi	0.33	0.20	1.00	0.04	0.49
shared_receipt_with_poi	0.23	0.29	0.04	1.00	-0.04
fraction_from_poi	0.15	-0.46	0.49	-0.04	1.00

I was also happy to see my intuition had been correct in regards to the creation of my new features, as both of them of were used.

3. *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 ended up using a Decision tree for my classifier. This was after trying a number of other algorithms and getting unsatisfactory results. For instance, the first algorithm I tried KNearestNeighbors, and despite playing around with tuning a bit, I had trouble getting the kind of scores I wanted. The next algorithm I tried was an SVM, but that also turned out to be a bad idea. Support vectors turned out to be a very bad fit for this dataset, because it is so sparse, distributed poorly, and highly imbalanced. When using an "rbf" kernel, for instance, which is the kernel that GridSearch preferred in this instance, the support vector classifier tries to find areas around which pois are common, and areas around which pois are rare. Because the data is fairly spread out,and because there are so few pois, on this data set the SVM ended up essentially building small areas around each poi, and then building enough areas to cover the non-pois elsewhere. To put it simply, in all instances, the SVM was prone to over-fitting, and thus it was not well suited to generalizing rules, which could be applied to new previously unseen data. The Decision Tree ended up working really well, since it was flexible enough to be tuned appropriately, and did a much better job than SVM at reacting to new data (less over fitting). In addition to achieving the best scores, I also liked how I could look at “Feature Importance’s” in order to see what role each feature played in making that classifier’s predictions.

Optimized Classifier	Accuracy	Precision	Recall	f1_score	Params (Feature Selection K=4,filtered set)
SVM	No True Positives	No True Positives	No True Positives	No True Positives	{'Classifier__gamma': 1, 'Classifier__class_weight': {0: 0.2, 1: 0.8}, 'Classifier__kernel': 'rbf', 'Classifier__C': 1}
KNearest Neighbor	0.88333	0.44919	0.22100	0.29625	{'Classifier__algorithm': 'auto', 'Classifier__n_neighbors': 4, 'Classifier__weights': 'uniform'}
DecisionTree	0.89278	0.51585	0.78100	0.62132	{'Classifier__class_weight': {0: 0.15, 1: 0.85}, 'Classifier__min_samples_split': 19, 'Classifier__criterion': 'gini'}

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

Tuning the parameters of an algorithm basically means specifying how, and under what constraints you want your classifier fitted, with the goal of achieving the best results with your dataset. For instance, one of the parameters, on the decision tree that was actually really helpful was class_weight. This parameter basically allowed me to specify how much extra weight I want to assign towards getting certain class values correct. So for instance, in the Enron dataset, we have only 18 out of 145 data points that were POIs. Since we really only really cared about finding out which data points were indicative of guilt, adding extra weight to this class made a lot of sense, and was able to boost my success metrics tremendously. Other parameters that I included I in my tuning were min-sample-split, which specified the minimum number of samples required to split an internal node, and criterion, which specified whether splits should focus on reducing impurity or gaining information. Because there were so many different combinations of features I could use, I ended up tuning my model with GridSearchCV, which is a function that essentially auto tunes my classifier with all of the possible parameters in mind, in order to get the best result. By default, GridSearch uses kfold cross validation, but, again because of the skewed dataset, I specified that it should use a form a Stratified Shuffle split instead. I also told it to focus on optimizing f1 using the micro calculation (Metrics counted globally by counting the total true positives, false negatives and false positives).

Classifier	Accuracy	Precision	Recall	f1_score	Params (Feature Selection K=4,filtered set)
Decision Tree-No Tuning	0.85411	0.34702	0.35500	0.35096	None
Decision Tree-With Tuning	0.89278	0.51585	0.78100	0.62132	{'Classifier__class_weight': {0: 0.15, 1: 0.85}, 'Classifier__min_samples_split': 19, 'Classifier__criterion': 'gini'}

DT Parameter Options:

- *Classifier__min_samples_split=SplitOpt,*
- *Classifier__criterion=CritOpt*
- *Classifier__class_weight=[{0:.2,1:.8},{0:.15,1:.85},{0:.1,1:.9},'balanced'])*

5. *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"]*

In machine learning, training data is the data you train your classifier with, while testing data is data that you've set aside so that you can test the accuracy of your trained classifier. While ideally you would want to maximize the size of both sets of data, in practice, this can be really difficult sometimes. For instance, in the case of the Enron data set, because we have so few data points, separating ANY data is very hard as you are risking either improperly training the classifier, or testing it on an insufficiently sized sample. With cross validation, you get around this problem, by partitioning your data in k sets and running your classifier multiple times, each time rotating which of your partitions is the testing set, and average the result. While this can take up more computational power for training and test, the benefit generally is more accurate results. There are also ways where you can validate your data the wrong way. For example, the Enron data set is very imbalanced. This can be a problem if the partitions you are randomly creating don't contain enough (or any) of the less frequent class, as it makes it more difficult for the classifier to make/learn correct associations. Another problem is if the data has been previously ordered in some way, so that the partitions are abnormally dissimilar to each other (Imagine 10 partitions, 5 made up of all POI data points, and 5 made up of all not poi data points). With these issue in mind, I chose to use a method known as Stratified Shuffle split when doing my cross validation. This means that when I create my partitions the data is shuffled beforehand, with the caveat being that each partition has to maintain the same class distribution as the initial dataset. A final note was that when I was using GridSearch to tuning my classfier, I made sure the seed I used for stratified shuffle split was different from my final tester.py stratified shuffle split so that randomness would be mainted.

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. [relevant rubric item: "usage of evaluation metrics"]*

Two key metrics that I used as a measure of success were precision and recall, which are defined below.

Precision: TP/(TP+FP)
Recall: TP/(TP+FN)

In layman’s terms Precision can be defined as the average probability of a relevant call being correct, in our case how many predicted poi labels were correct, whereas “recall” represents how the fraction of actual poi’s out the total poi’s we’ve identified correctly.

I was able to achieve average scores of 0.54512, and 0.60100 for precision and recall respectively. Given that the metric for success in this project was .3 and the level of imbalance in the dataset, I judged these scores as representing a pretty decent success with the classifier.