Udacity Machine Learning Project - Enron Data - Randy Crane

Part 1

Project Overview & Goal

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives.

In this project, I will play detective, and build a person of interest ("POI") identifier based on financial and email data made public as a result of the Enron scandal. This data has been combined with a hand-generated list of persons of interest in the fraud case, which means individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity. [*Adapted from Udacity project description*]

Machine learning is useful in trying to accomplish this goal because we are looking for patterns in a large amount of data, but we don't necessarily know what all those patterns are. We may have a good idea about some, so we can train our algorithms using that knowledge, but there is a lot we don't know—or what we think we know may not match reality. By s=using what we do know about what we're looking for—the POI list—machine learning can help us find that characteristics data in others that have not been identified, thus aiding in the investigation.

Overview of What's to Come

In **Part 2**, I will begin by answering some basic questions about the data.

In **Part 3**, I will explore the features by visualizing some correlations/outliers. At that time, I will also remove identified outliers that could hamper the continued analysis and identify how many data points for each feature do not contain data (they are "NaN"). I will also create a pair-plot visualization. Honestly, I don't expect it to be terribly useful in the final analysis. I just like them—and you never know what could spark a connection.

In **Part 4**, having established the baselines for the data set, I will add a few new features that I think might be useful later and create a scatter plot for two of these new features. I will then run two more pair-wise plots. Finally for this section, I will use SelectKBest to identify the strongest, and thus potentially most useful, features.

In **Part 5**, it's time to start looking at some classifiers. This part consists mainly of a chart showing the performance of 6 classifiers, all with their default values. I will also include a chart with 4 linear regressions. I don't expect to use them in the final analysis, but it is a good exercise.

In **Part 6**, I will transform/scale the features and select those that will be most useful in my POI identifier, creating an abridged feature list with only the 10 strongest, again identified by SelectKbest. Finally here, I will recreate the train/test split of the data using StratifiedShuffleSplit.

In **Part 7**, with that transformational work complete, I will have a new chart for the same 6 features, this time with parameters tuned using GridSearchCV.

In **Part 8**, I will conclude with some final thoughts regarding the project and the performance of my best-performing classifier.

Part 2

Overview of the Data

Example Dictionary of Initial Features ("Allen, Phillip K" used for reference):

'bonus'	'from_messages'	'restricted_stock'
'deferral_payments'	'from_poi_to_this_person'	'restricted_stock_deferred'
'deferred_income'	'from_this_person_to_poi'	'salary'
'director_fees'	'loan_advances'	'shared_receipt_with_poi'
'email_address'	'long_term_incentive'	'to_messages'
'exercised_stock_options'	'other'	'total_payments'
'expenses'	'poi'	'total_stock_value'

21 features

Task 1: Select what features you'll use.

I am excluding email address from the list of used features, as that is not something measurable or countable, and the information I could get from it is redundant to the key value. For now, I will leave all the rest in.

'poi'	'deferred_income'	'director_fees'
'salary'	'total_stock_value'	'to_messages'
'deferral_payments'	'expenses'	'from_poi_to_this_person'
'total_payments'	'exercised_stock_options'	'from_messages'
'loan_advances'	'other'	'from_this_person_to_poi'
'bonus'	'long_term_incentive'	'shared_receipt_with_poi'
'restricted_stock_deferred'	'restricted_stock'	

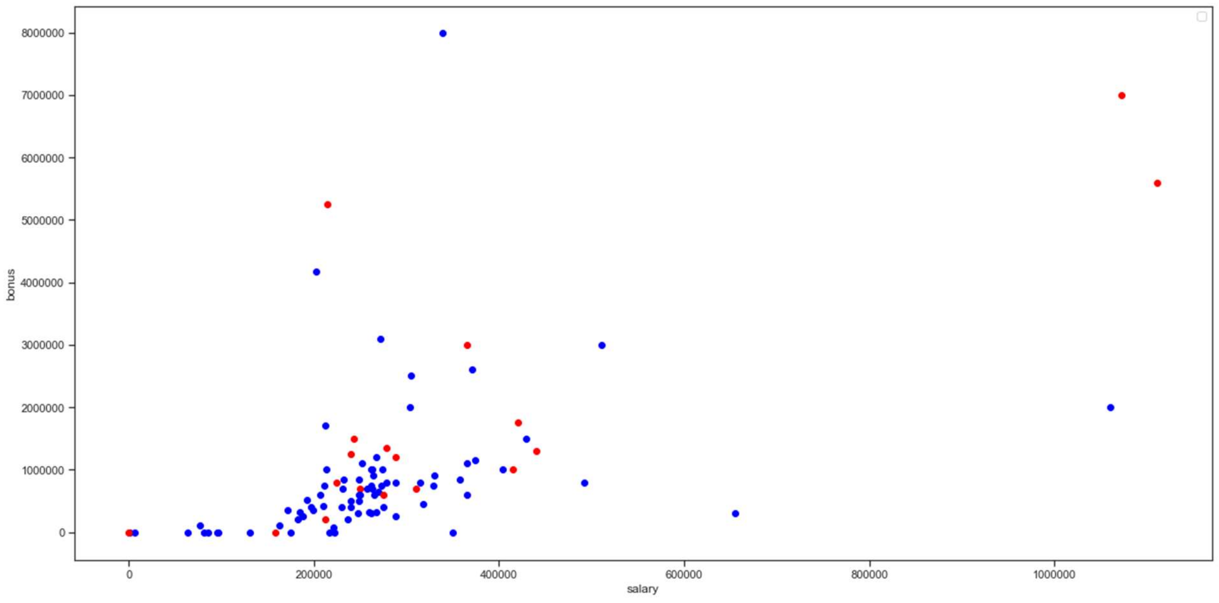
Total Number of data points:	146
Number of POIs:	18
Number of Non-POIs:	128
Number of features:	20

<u># of Missing Values by Feature:</u>

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

Part 3

Task 2: Remove outliers.

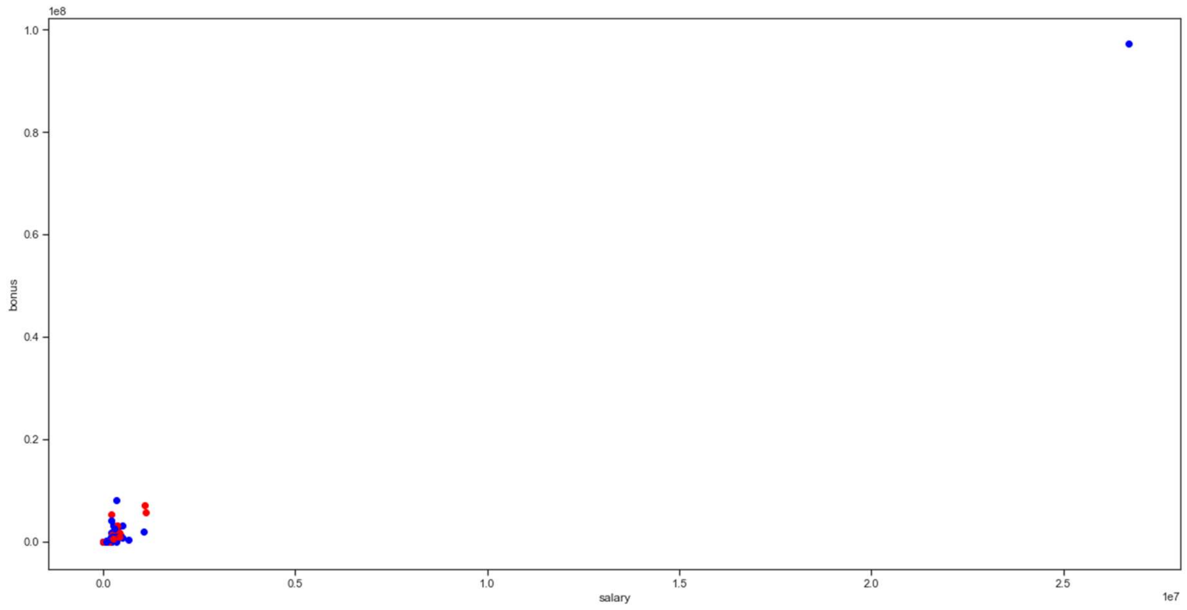


Visualization of Salary & Bonus, to help identify outliers. (Red = POI)

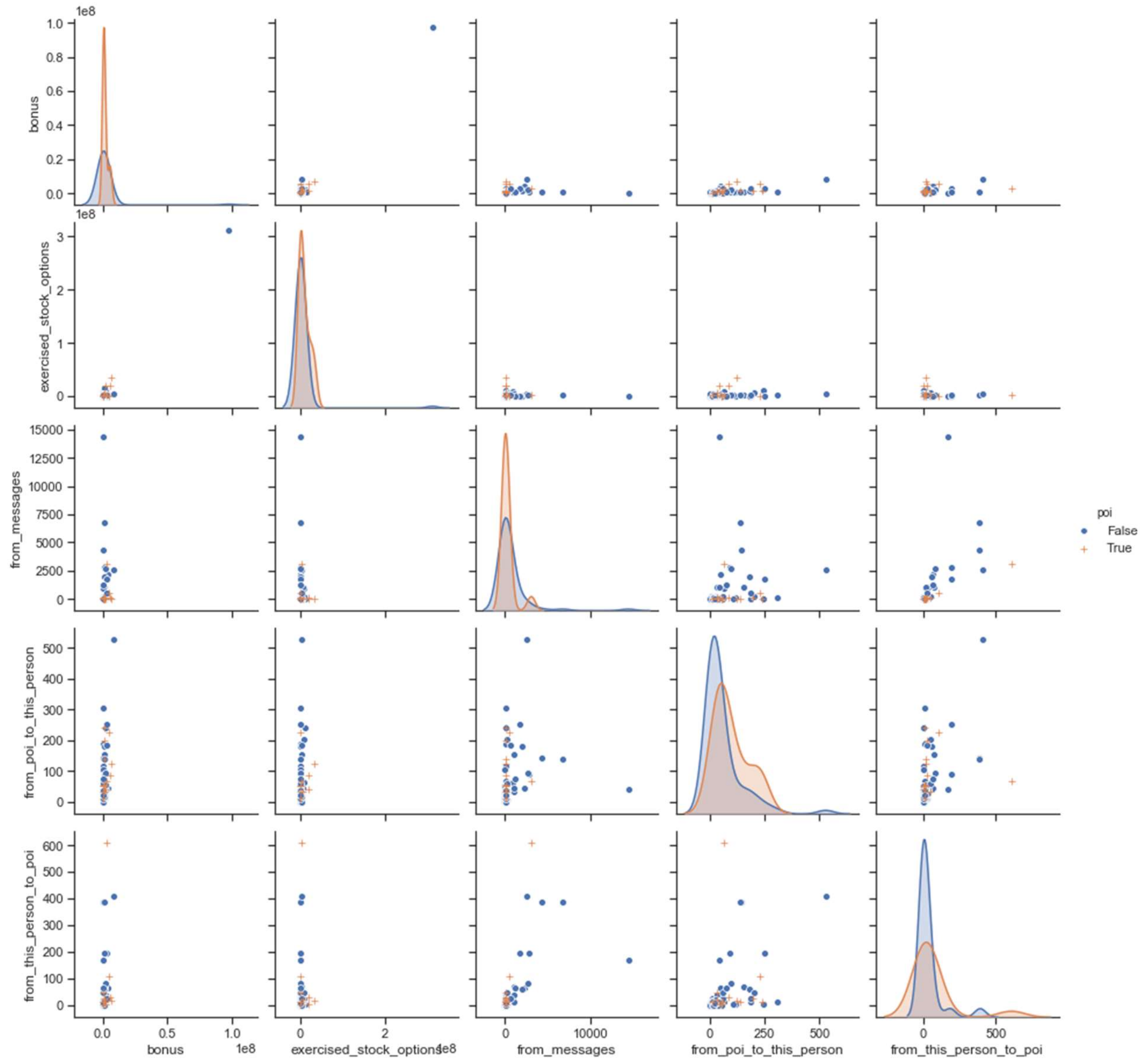
Removed outlier "Total", the point in the extreme upper-right, which is just the summary/total line from the spreadsheet.

Also removed outlier "LOCKHART EUGENE E", for whom all data is either NaN or 0.

Visualizations of Salary & Bonus, with outliers removed. (Red = POI)



Create pairsplot of some key features to get an overview of what interesting relationships there may be between features that could be used to engineer new features.



Some interesting relationships here, but nothing I want to explore in more detail right away.

Part 4

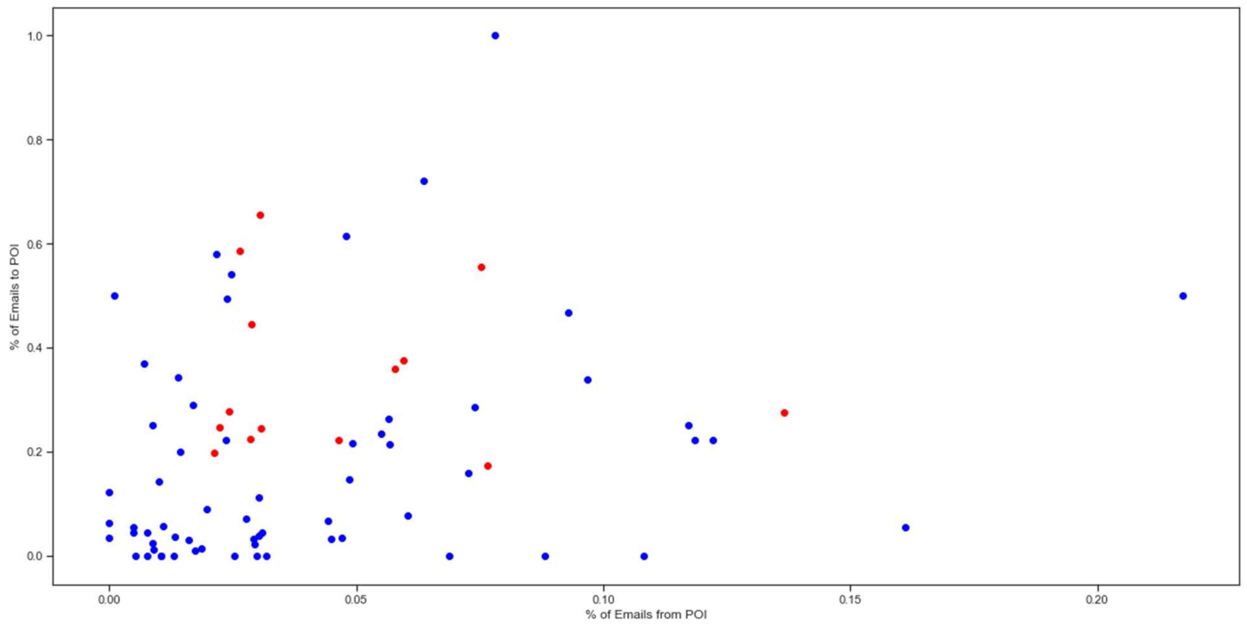
Task 3: Create new feature(s)

New Features:

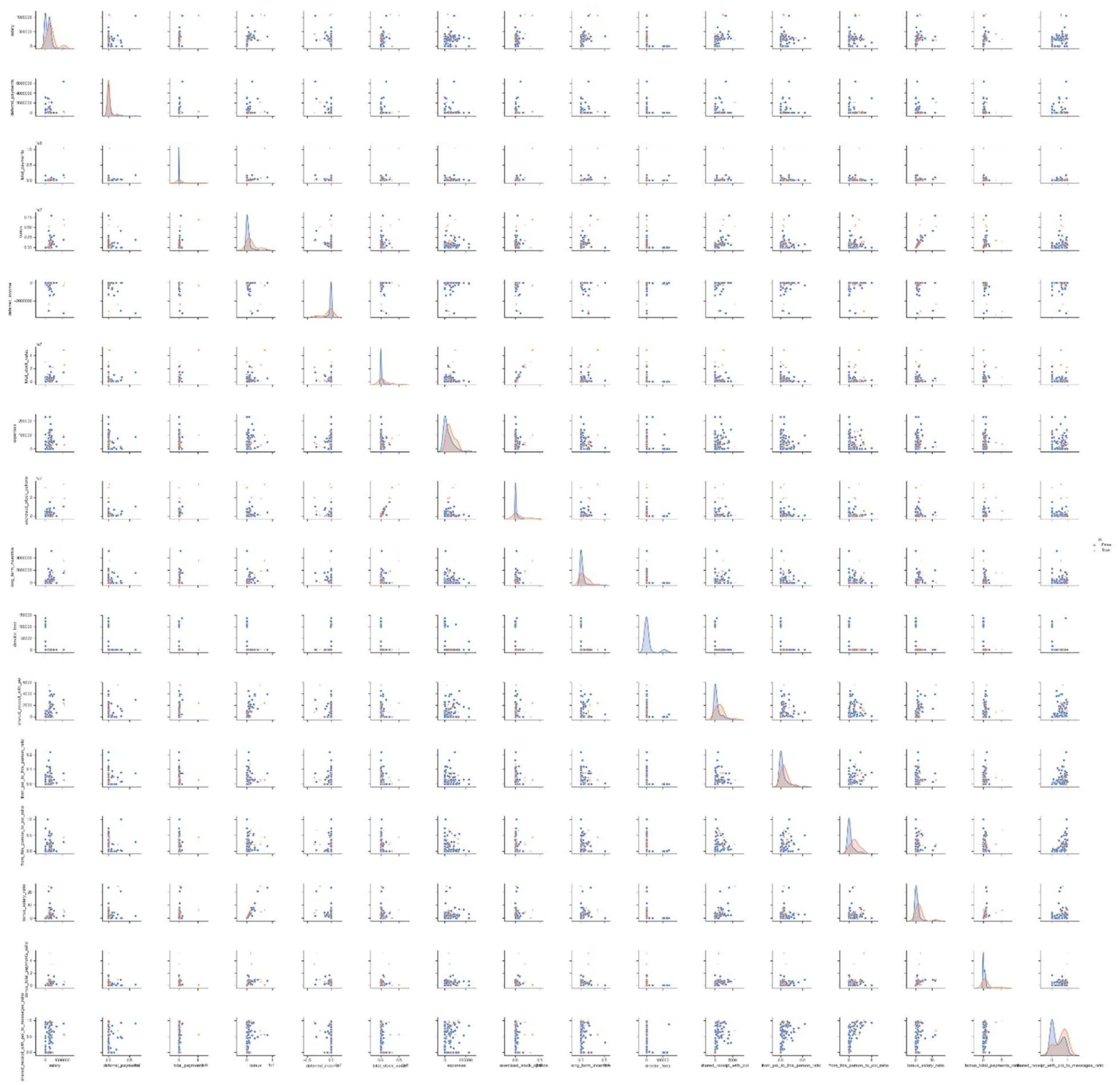
- 'from_poi_to_this_person_ratio'
- 'from_this_person_to_poi_ratio'
- 'bonus_salary_ratio'
- 'bonus_total_payments_ratio'
- 'shared_receipt_with_poi_to_messages_ratio'

Plot a pair of these new features

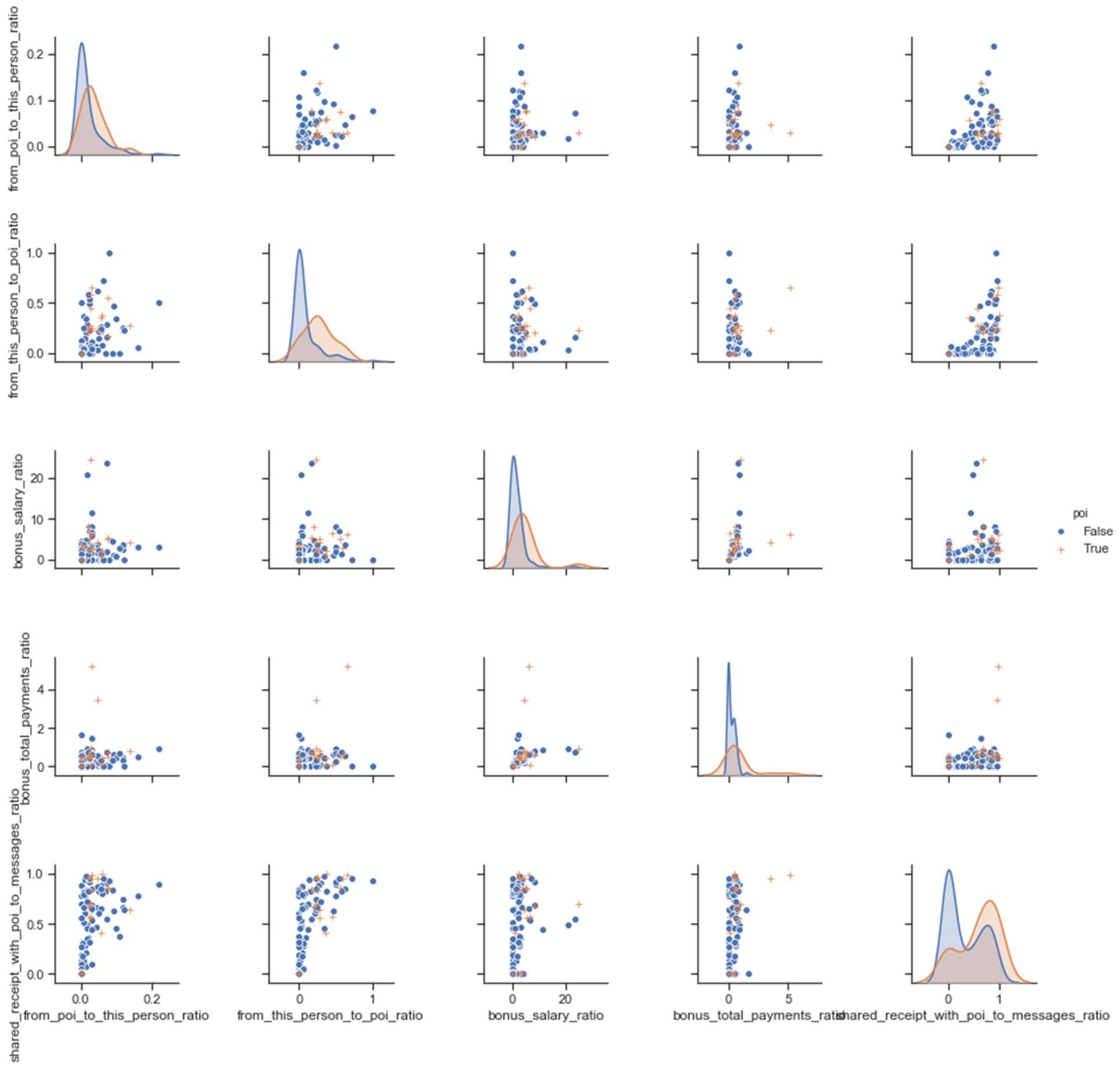
% of Emails from POI vs. % of Emails to POI (Red = POI)



Create pairsplot of all features, honestly just for fun. It’s going to be way too much to make any sense of.



As expected, that is way too much data. Let’s do another one, this time of just the new features.



Interesting. Definitely some potential relationships here, but this may be something to explore so me other time.

SELECTED 15 BEST FEATURES BY KBEST:

FEATURE	STRENGTH
exercised_stock_options	25.0975415287
total_stock_value	24.4676540475
bonus	21.0600017075
bonus_total_payments_ratio	20.9887684881
salary	18.575703268
from_this_person_to_poi_ratio	16.6417070705
deferred_income	11.5955476597
bonus_salary_ratio	10.9556270745
long_term_incentive	10.0724545294
restricted_stock	9.34670079105
shared_receipt_with_poi_to_messages_ratio	9.29623187148
total_payments	8.86672153711
shared_receipt_with_poi	8.74648553213
loan_advances	7.24273039654
expenses	6.23420114051

Part 5

Task 4: Try a variety of classifiers (Green cells meet or exceed the 0.3 threshold required)

Classifiers – Defaults

Classifier	Accuracy	Precision	Recall	F-1 Score
Decision Tree	0.7955	0.2000	0.2500	0.3077
Gaussian Naive Bayes	0.8864	0.4000	0.5000	0.4444
Support Vector	0.8864	0.0000	0.0000	0.0000
K-Neighbors	0.8864	0.0000	0.0000	0.0000
Random Forest	0.9091	0.4000	0.6667	0.5000
AdaBoost	0.8182	0.2000	0.2000	0.2000

Decision Tree Classifier Feature Ranking:

1. salary (0.2885)
2. deferral_payments (0.2788)
3. total_payments (0.1768)
4. loan_advances (0.1492)
5. bonus (0.0624)
6. restricted_stock_deferred (0.0442)

Surprisingly, these were the only features that had non-zero values.

Linear Regressions

Feature Pair	Coefficient	Intercept	Training r-squared Score	Test r-squared Score
Bonus / Salary	6.0333	-398,434.2152	0.2390	0.0380
Exercised Stock Options / Bonus	2.1997	459,262.8073	0.3580	-0.3900
Bonus / Long Term Incentive	1.2371	455,869.2375	0.2620	0.4950
Exercised Stock Options / Long Term Incentive	7.2147	-1,375,943.4476	0.6120	-3.0260

Part 6

Task 5: Tune your classifier to achieve better than 0.3 precision and recall

Until now, I’ve been working just with the raw data. Some of the results have been terrible and/or useless (SVC and K-Neighbor in particular), but some have been very good. Gaussian Naïve Bayes in particular performed admirably.

However, now it’s time to do some scaling and engineering on the data before tuning the algorithms. To accomplish that, I used two tools:

- MinMaxScaler
- SelectKBest

The scaling ended up being irrelevant, because none of the features I used after running SelectKBest benefitted from it. The 10 best features are:

'salary'	'from_this_person_to_poi_ratio'
'bonus'	'deferred_income'
'restricted_stock'	'bonus_salary_ratio'
'total_stock_value'	'long_term_incentive'
'exercised_stock_options'	'bonus_total_payments_ratio'

To ensure the machine learning algorithm generalizes the data well, we use validation. A common mistake when it comes to validation is to testing on a training data set, evaluating the algorithm on the data that was used to classify it. While this is an easy mistake to make, it’s also a relatively easy one to catch, because it results in overfitting—a deceptively high performance for the algorithm on the training data, but a very low performance on the testing data.

If the algorithm is being evaluated based on how well it predicts the values of the training set, of course it is going to do exceptionally well because it already knows all of that data.

With these transformations complete, it was time to create new test data sets. In this project, I used a StratifiedShuffleSplit ("sss") cross-validation to split 70% of the data into training data and the remaining 30% into testing data.

Part 7

Finally, I tuned the classifiers. To accomplish this, I used GridSearchCV on some key parameters for each one.

Classifiers - Tuned by GridSearchCV

Classifier	Accuracy	Precision	Recall	F-1 Score
Decision Tree	0.6905	0.6000	0.2143	0.3158
Gaussian Naive Bayes	0.8333	0.6000	0.3750	0.4615
Support Vector	0.9286	0.4000	1.0000	0.5714
K-Neighbors	0.8095	0.4000	0.2857	0.3333
Random Forest	0.8881	0.4000	0.5000	0.4444
AdaBoost	0.8095	0.4000	0.2857	0.3333

For comparison, here again are the pre-tuned results:

Classifiers – Defaults

Classifier	Accuracy	Precision	Recall	F-1 Score
Decision Tree	0.7955	0.2000	0.2500	0.3077
Gaussian Naive Bayes	0.8864	0.4000	0.5000	0.4444
Support Vector	0.8864	0.0000	0.0000	0.0000
K-Neighbors	0.8864	0.0000	0.0000	0.0000
Random Forest	0.9091	0.4000	0.6667	0.5000
AdaBoost	0.8182	0.2000	0.2000	0.2000

When I first ran GridSearchCV, I ended up with worse scores than the untuned versions. However, I realized that it was tuning for accuracy and so the rest of the metrics were suffering. Since I was less concerned about accuracy than I was the rest of the metrics, I set the GridSearchCV "scoring" parameter to F1 so I could optimize the F1 score, which in turn would optimize the combination of Precision and Recall, and ran the parameter tuning again. (Citation)

With the parameters tuned, all 6 classifiers achieved the required 0.3 threshold. Only 2 achieved it in Recall, and one of those (Support Vector) had a perfect 1.0, which is a sign of overfitting. Considering that unrealistically high value, I am going to discard SVC as a possible final choice.

The two best performers in the non-tuned set were **Gaussian Naïve Bayes** and **Random Forest**. Disregarding SVC, **Random Forest** also came in strong in the tuned set. Interestingly, in the tuned version of Random Forest, accuracy and recall both decreased, as did the F1 score.

Part 8

Explanation of Evaluation Criteria

The evaluation matrices I used throughout this project were precision, recall, and F1. I did, of course, use accuracy, but that is not a particularly useful evaluation metric for a data set like this.

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. Precision ensures that innocent people are excluded from the list of POIs at the expense of also excluding some actual POIs.
Precision = True Positives / (True Positives + False Positives)

Recall is the ratio of correctly predicted positive observations to all observations in the class. Recall ensures more POIs are identified at the expense of including some innocent people in the list of POIs.
Recall = True Positives / (True Positives + False Negatives)

If I have to choose between the two, for this analysis I would want to maximize the recall score. I would rather include some non-POIs in the investigation and then rule them out as we investigate rather than missing some POIs. I think it would be harder to correct for that in the investigation.

However, if possible I would like to maximize both precision and recall simultaneously. To accomplish this, my primary metric to focus on was the F1 score. In some way, the F1 score can be thought of "the best of both worlds."

The F1 score "considers both the Precision and the Recall of the test to compute the score. ... The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0." (Citation)
*F1 = 2 * ((Precision * Recall) / (Precision + Recall))*

F1 ensures that both False Positives and False Negatives rates are as low as possible. To me, this is the ideal.

Final Thoughts

Considering all of the classifiers, the two best performers in their default, non-tuned states were **Gaussian Naïve Bayes** and **Random Forest**.

Classifiers – Defaults

Classifier	Accuracy	Precision	Recall	F-1 Score
Gaussian Naive Bayes	0.8864	0.4000	0.5000	0.4444
Random Forest	0.9091	0.4000	0.6667	0.5000

Classifiers - Tuned by GridSearchCV

Classifier	Accuracy	Precision	Recall	F-1 Score
Random Forest	0.8881	0.4000	0.5000	0.4444

In the tuned set of classifiers, **Random Forest** was the only one to achieve the required threshold without overfitting. It also came in strong in the untuned set. I am confident that this classifier is performing well, and could likely be tuned even finer to achieve better results. **Gaussian Naïve Bayes** performs well enough untuned that I would consider keeping this one as a viable option as well.