Summary and Goal

The goal of this project is to utilize the financial and email dataset from Enron Corpus, which was made public by US Federal Energy Regulatory Commission during its investigation of Enron, is to establish a model that predicts an individual as a “Person of Interest” (POI). The corpus contains email and financial data of 146 people, most of which are senior management of Enron. The corpus is widely used for various machine learning problems.

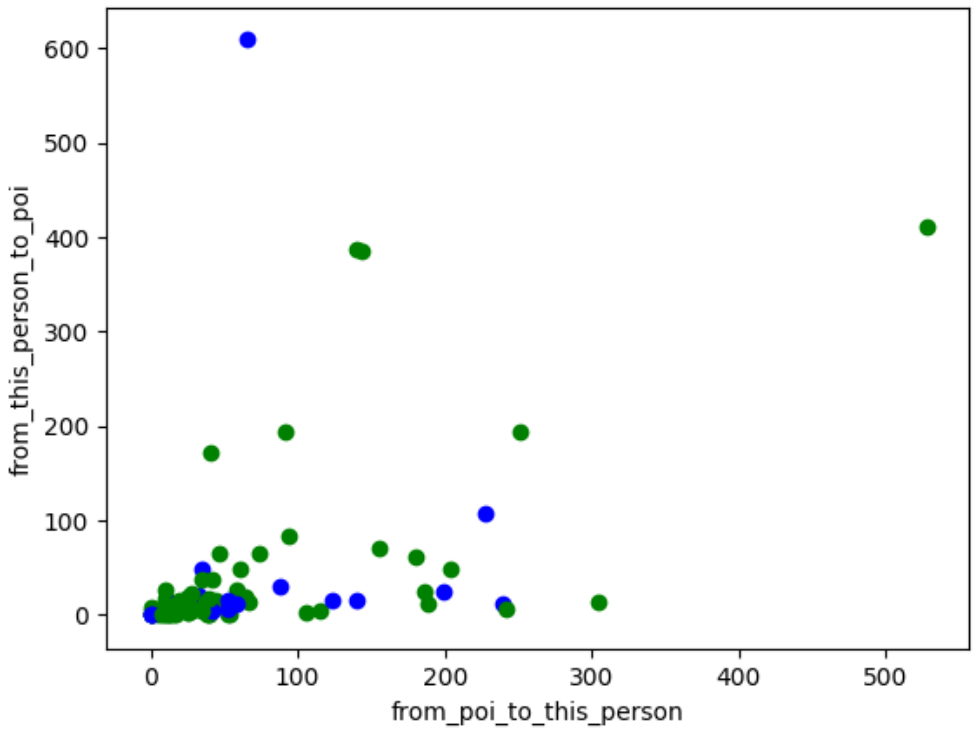
The dataset contains 146 records with 18 POI, 128 Non-POI, 21 total features.

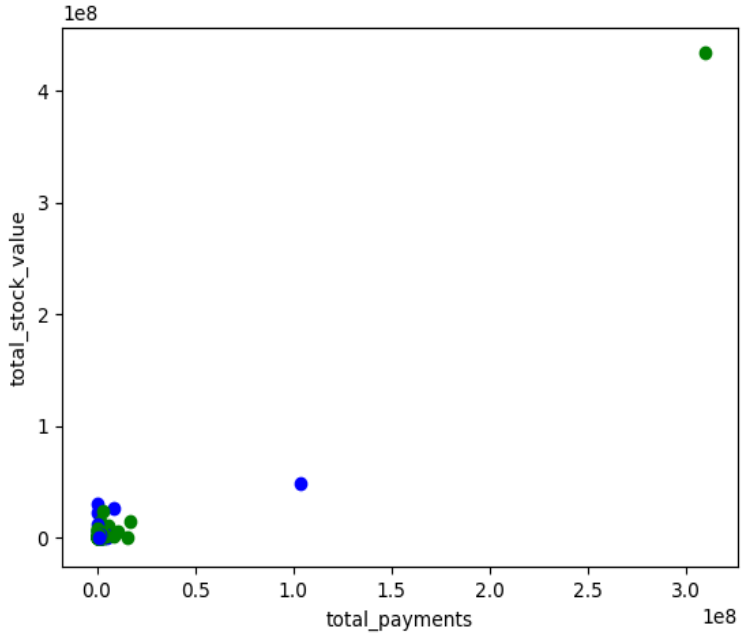
The dataset contains missing values and some outliers.

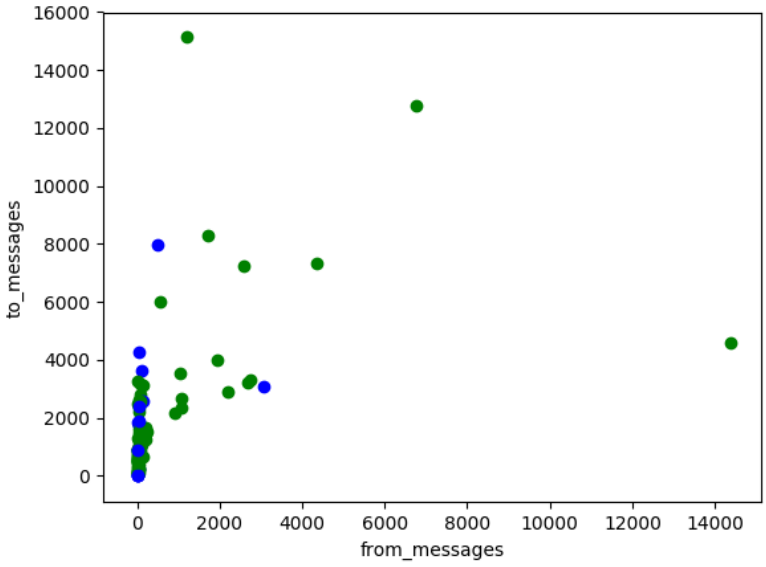
The outliers are:

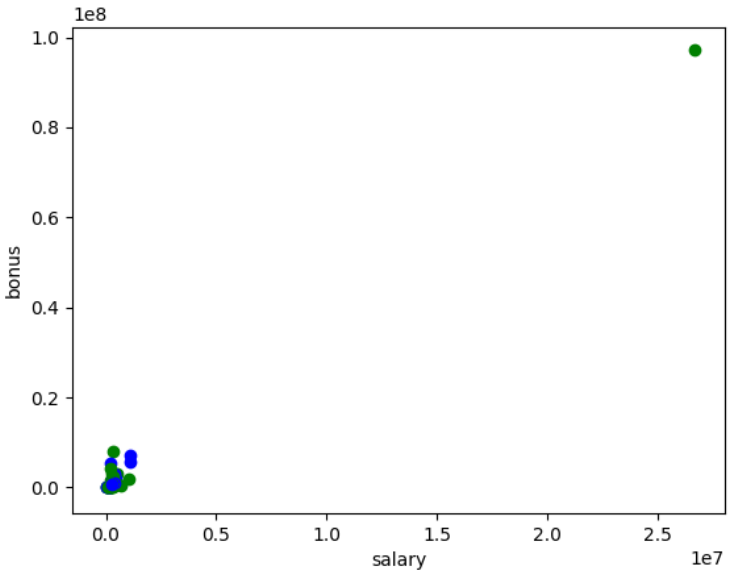
- TOTAL: Added aggregated data to everything
- THE TRAVEL AGENCY IN THE PARK: this is listed as name
- LOCKHARD EUGENE E: contains only NaN

Spot Outlier




1e8
4
3
2
1
0
total_stock_value
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e8
total_payments


16000
14000
12000
10000
8000
6000
4000
2000
0
to_messages
0
2000
4000
6000
8000
10000
12000
14000
from_messages



Features Selection

I used scikit-learn SelectKBest to select the best influential factors. I decided to use 12 as K. The K-best approach is an automated univariate feature selection algorithm. Also it selects the K features that are most powerful (where K is a parameter), in this case. I decided to use 12 as K because after running .best_params, k value is returned as 12. I also added a new feature thinking I might be missing email features in the resulting dataset, so I added 'Shared_receipt_with_poi'. The main purpose of creating this feature, ratio of POI messages, is that we expect POI contact each other more often than non-POIs. And the fact that 'Shared_receipt with Poi' is included after using SelectKBest proved that it is quite crucial. The precision score and recall under Gaussian after the new feature is added went up to [0.5,0.6].

The scores for each feature: `('Selected features and their scores: ', {'salary': 18.289684043404513, 'total_payments': 8.7727777300916792, 'loan_advances': 7.1840556582887247, 'bonus': 20.792252047181535, 'total_stock_value': 24.182898678566879, 'shared_receipt_with_poi': 8.589420731682381, 'fraction_to_poi': 16.409712548035799, 'exercised_stock_options': 24.815079733218194, 'deferred_income': 11.458476579280369, 'expenses': 6.0941733106389453, 'restricted_stock': 9.2128106219771002, 'long_term_incentive': 9.9221860131898225})`

Algorithm Used

I tried using Random Forest Classifier, Support Vector Machine, GaussianNB, and Logistic Regression, KMeans and I ended up choosing Support Vector Machine.

Algorithm	**Precision**	**Recall**
Random Forest Classifier	0.52	0.29

Support Vector Machine	0	0
GaussianNB	0.5	0.6
Logistic Regression	0.6	0.43
KMeans	0	0
Decision Tree Classifier	0.17	0.2

Algorithm Tuning

To tune the parameters of an algorithm means adjusting the algorithm when training it, so the fit on the test set can be improved. The more tuned the parameter, the more biased the algorithm will be to the training data. There might be cases of overfitting, which leads to poor performance.

I tried to tune of algorithm in a way that it is not over fitting, making increment changes to the parameters. As the result shows, I can get good results with Logistic Regression and GaussianNB. However, GaussianNB provides better result without the gap between recall and precision, which I will explain the significance of both metrics later.

I was hoping Support Vector Machine would do the trick, but it ended up giving poor performance.

Validation

Validation comprises set of techniques to make sure the models generalize with remaining part of the dataset. A classic mistake is to over-fit the model when it was actually performing well on training set but poorly on test est. I validated my analysis using cross_validation with 1000 trials. The trials is inspired by both a project I came across and by tutoring a student college level statistics. By testing the dataset repeatedly, we can obtain more correct result. The test size is 0.3, meaning 3:1 training-to-test ratio.

Precision vs Recall

I used precision and recall as 2 main evaluation metrics. The algorithm of my choosing 'GaussianNB' produced a precision of 0.5 and a recall score of 0.6.

Precision refers to ratio of true positive – predicted POI matches actual result.

Recall refers to ratio of true positive of people flagged as POI. In English, my result indicated that if the model predicts 100 POIs, there would be 50 people that are actually POIs and the rest of 50 are not. With recall score of 0.6, the model finds 60% of all real POIs in prediction. This model is good at finding bad guys without missing anyone.

Accuracy is not a good measurement as even if non_poi are all flagged, the accuracy score yield high success rate.

References:

Scikit-learn documentation

Udacity: Machine Learning Intro

Github