Introduction:

Enron Corporation was one of the world's major electricity, natural gas, communications and pulp and paper companies with approximately 20,000 staff before its bankruptcy at the end of 2001. The basis of their downfall was the corporate fraud by their company employees. The investigation proved the fraud and all the data was made public. This is probably one of the largest email corpuses present on the Internet along with thousands of financial and email data.

The goal of this machine-learning project was to build a prediction model to classify the Person of Interest (POI) from the dataset. I've built my model/classifier with the help of Python's scikit-learn library.

Understanding the Dataset and Question:

The goal of this project was to utilize the financial and email data from Enron to build a predictive model that could identify whether an individual could be considered POI or not. The branch of machine-learning this dataset deals with is the Supervised Classification. We have labels /output variable that have previously been classified as POI or not and we use this information to build a predictive classifier. This dataset contained 146 records with 14 financial features, 6 email features, and one class label (POI). Of the 146 records, 18 were labeled as POI. The dataset also contains NaN values for a lot of features, these datapoints can either be removed, ignored or set to some arbitrary value. In the testing, we chose to ignore them.

Outliers:

`TOTAL` was removed, as it was simply a record totaling all of the financial statistics from the financial data.

`Eugene E. Lockhart` was removed during data processing since it had no data.

`THE TRAVEL AGENCY IN THE PARK`: This record did not represent anything in particular.

Feature Selection/Engineering:

So this was the most challenging part of the data munging phase. Firstly, I added 2 new features, which were the total_finances (addition of all the financially valuable assets) and ratiosPOI (this was the ratio of emails to and from the POI person to each individual). After running my basic model, no improvements were shown, they decreased the precision I was getting for my model earlier and therefore I decided to add Logarithmic and Squared features for all the input features. I thought this would meet the project rubric in precision, but to no avail! The precisions reached were below the expected value of 0.3. This made me change my approach to apply the SelectKBest Features in the GridCVSearch using the pipeline.

I then added GridCVSearch pipeline to my model and that helped me narrow down on my features. But before that, I also applied Feature Scaling to all my features and put it in range (0,1) with the help of MinMaxScaler . This helped a lot as a lot of features were very extreme and Feature Scaling also works optimally with Logistic Regression.
PCA and SelectKBest were two dimension reduction techniques that helped increase the overall performance of the classifier. K (k=17) best features and n components (n=2) were fed to the model to generate best results and these selections were made based on GridCVSearch. The features that were used in the classifier were:
salary
to_messages
total_payments
exercised_stock_options
bonus
restricted_stock
shared_receipt_with_poi
total_stock_value
expenses
loan_advances
other
from_this_person_to_poi
director_fees
deferred_income
long_term_incentive
total_compensation
director_fees
These features were chosen using the GridCVSearch, I tried playing around with the values of k as 'selection__k': [7, 9, 12, 17, 'all'] and this resulted in 17. That is why I used 17 features to get the desired outputs.

Pick and Tune an Algorithm:

I began with Support Vector Machine with Linear kernel. I thought this would be a good start because SVM can easily build complex plots and in my head, it was always going to be a very complicated plot. Turned out this wasn't a good choice even though I tried tuning the C, tol and k values. I then put my thoughts on SVM with RBF kernel but this performed even poorly than the Linear SVM.
Finally, I chose to use Logistic Regression as my classification algorithm and I used the same parameter tuning as it gave me better insight on an algorithm's performance now that I had set a baseline for all of them.
Below are the results on the test dataset of the 3 algorithms I chose.

Model	Precision	Recall
Logistic Regression	0.30458	0.90200
Linear SVM	0.29657	0.88750
RBF SVM	0.29269	0.83050

The best parameters for Logistic Regression were:

1. C (Value of the regularization constraint): 1e-3
2. Class_weight(Over-/undersamples samples of each class (inversely proportional to class frequencies)): ‘auto’
3. Tol (Tolerance for stopping criteria): 1e-64

The basis of hyper-parameter optimization is to ensure that the model does not overfit its data by tuning. The traditional way of performing parameter optimization is used in my project. Grid search is simply an exhaustive searching through a manually specified subset of the parameter space of a learning algorithm.

Validate and Evaluate:

The technique of validation comes from the fact that how well our classifier/model performs when subject to unseen test data. The most common mistake that can be made here is the problem of overfitting. This comes from the fact that we train our classifier with so much finesse and tune our parameters to increase the scoring metric that when we apply the same model to unseen data, we get very bad results because we have trained too hard on our training dataset and not thought about the possible factors that could affect our model.
One way to check if our model has overfit or not we perform cross-validation. In this process we randomly partition data in k subsets, we use (k-1) subsets as training samples and 1 subset as testing sample. We run our experiment multiple times to generate an average model.
For our case the model selection process was validated using 1000 randomized stratified cross-validation splits and selecting the parameters, which performed best on average over the 1000 splits.
For our project, since the dataset was so imbalanced, if we predicted that no on was POI we would have gotten an accuracy of roughly 87%. Therefore accuracy wasn’t a great tool for measuring the performance of our model. We used precision and recall to define our performance metric. Precision is the ratio of how often your model is actually correct in identifying a positive label to the total times it guesses a positive label. In our dataset, it means that if we were using the model to judge whether or not to investigate someone as a possible person of interest, it would be how often the people we chose to investigate turned out to really be persons of interest. On the other hand recall is, the ratio of how often your model correctly identifies a label as positive to how many total positive labels there actually are. For our model, to decide whether or not to investigate someone as a possible person of interest, it would be how many persons of interest did we identify out of the total amount of persons of interest that there were.
Precision = True Positive/(True Positive+False Positive)
Recall = True Positive/(True Positive+False Negative)

Conclusion:

Since the dataset was imbalanced and small, adding a lot of features did not really help the cause of getting a good model. This really helped me understand

what features should be tuned when and how to approach an imbalanced dataset which had very few positive cases.
For further improvement I wish to add something like k-means clustering, that way it would add unsupervised portion to the classifier and may give us patterns that we cannot see with supervised algorithms.
This was a great learning experience!

Files:

Poi_id.py
Feature_add.py
Evaluate.py
In order to run the project, have the files in the same folder and run python poi_id.py. This would create the pkl files for testing and then run python tester.py.