Notes to reviewer:

- **regarding exception:** please refer to discussion. udacity.com/t/scoring-gridsearchcv/26354.
- **tester.**py: modified to import sklearn.preprocessing, this was the only way I could get a pipeline with a scaler to work. If you know of a way to do this without modifying tester.py, please enlighten me.

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 is to leverage machine learning methods along with financial and email data from Enron to construct a predictive model for identifying potential parties of financial fraud. These parties are termed "persons of interest," but kids these days just say POIs.

The Enron data set is comprised of email and financial data (E + F data set) collected and released as part of the US federal investigation into financial fraud. POI labels were hand curated (this is an artisan feature!) via information regarding arrests, immunity deals, prosecution, and conviction. It is important to note that only those names contained in both the financial and email data set were passed to the final data set (inner join). So while the email data set contains 35 POIs, the final data set contains 18 labeled POIs. Without both financial and email features, it would be difficult to build an accurate and robust model.

The data set is comprised of:

- 146 points, each theoretically representing a person(2 are not people, soylent green is people)
- 18 of these points is labeled as a POI and 128 as non-POI
- Each point/person is associated with 21 features (14 financial, 6 email, 1 labeled)
- financial features: ['salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees'] (Units = USD)
- email features: ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'poi', 'shared_receipt_with_poi'] (units = number of emails messages; except 'email_address', which is a text string)
- POI label: ['poi'] (boolean, represented as integers)

Exploratory data analysis (scatter plot of scaled bonus vs. salary) revealed five outliers, one of which was eliminated from the data set.

Outliers:

- TOTAL: Extreme outlier for numerical features, consisting of spreadsheet summary artifact
- Four high rollers, including Lay and Skilling: While their values are extreme, these are legitimate values of actual POIs, so they were not eliminated.
- One other data point was eliminated (THE TRAVEL AGENCY IN THE PARK) because it does not represent a person, and therefore can't be a POI, leaving 144 data points.

While there is missing data, caution was observed in making any changes that might bias the models, therefore no effort was made to fill or impute values.

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 doesn't 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.) If you used an algorithm like a decision tree, please also give the feature importances of the features that you use. [relevant rubric items: "create new features", "properly scale features", "intelligently select feature"]

My approach to features was first to engineer features and eliminate problematic features. I engineered 3 features which focused on rate of interaction with known POIs: (1) poi_ratio: ratio of the total poi interaction (to, from, and shared emails) to the total emails sent (2) to_poi_ratio: ratio of total emails to a POI to total emails (3) from_poi_ratio: ratio of total emails from a POI to total emails. The rationale for addition of these features is that persons whom interacted with POI's at higher rate (total interaction or to/from) may be colluding with those POIs to commit fraud and therefore be POIs themselves. Following this I removed the email address feature because it was not useful and created exceptions. Features were scaled with a min-max scaler prior to passing into select_k_best or PCA, to effect even weighting of features. This is essential to perform before either of these functions, considering the overall dynamic range of values (several orders of magnitude) and inherent differences in their units (i.e. dollars vs. fraction or number of emails). Either univariate feature selection (select K best), PCA or both were applied to the scaled features. I did not explicitly test my engineered features, but relied on select_k_best to pick the k most influential features. poi_ratio was scored as the 6th most influential feature. PCA revealed that 13 components accounted for 95% of the variance.

Table1. 10 Best Features with Score and Percent Missing (NaN) Data

Feature	Score	Percent_nan
exercised_stock_options	24.8	29.9

total_stock_value	24.18	13.2
bonus	20.79	43.8
salary	18.29	34.7
deferred_income	11.46	66.7
poi_ratio	10.02	40.3
long_term_incentive	9.92	54.9
restricted_stock	9.21	24.3
total_payments	8.77	14.6
shared_receipt_with_poi	8.59	40.3

For exploration and tuning I tried a variety of feature numbers (k) and PCA components as well as a combination of univariate feature selection and PCA components. I tried the 5,6,8,9,10,11,12,and 15 best features; the PCA n_components = .98, .95, or .90, and the 5,6,8,9,10,11,12, and 15 best features piped through PCA with n_components = .98, .95, or .90. The value of k was changed manually but n_components was made part of GridSearchCV (change k, run search, evaluate, repeat). The 10 best features combined with PCA(n_components = .95) was consistently the top performing combination in terms of precision and recall. This combination accounted for 95% of the variance in 6 components. While I had originally decided to try PCA just to try it out and learn now to code up PCA, it ultimately became a critical part of my classifier.

3. What algorithm did you end up using? What other one(s) did you try? [relevant rubric item: “pick an algorithm”]

Ultimately I used adaboost with the default base estimator (DecisionTreeClassifier). I also tried a DecisionTreeClassifier and GaussianNB. I tried a smattering of several others (SVM, logistic regression, k_means) but decided to focus on GaussianNB for its simplicity and the other two because they are able to complement each other.

My first approach was to try GaussianNB since it was provided in the code and is low complexity in terms of parameters (none). I found it a pleasant challenge to try increasing the performance of my classifier through feature selection and PCA. I was able to get recall and precision to > 0.3 but ultimately hit a wall, and decided to move on to other classifiers. I may revisit increasing the performance of this classifier by engineering new features.

Both decision tree and adaboost classifiers were used for mini-projects during the course, so I had some familiarity with the mechanics and implementation of these classifiers. The DecisonTreeClassifier is the default base estimator of adaboost, so I reasoned that this would be a good opportunity to tune a DecisionTreeClassifier which I would ultimately boost with adaboost. Through tuning, I was able to get recall and precision > .3 for the DecisonTreeClassifier, but adding it to my tuned adaboost classifier made little impact and this ensemble was actually a worse performer that adaboost with the default DecisonTreeClassifier settings.

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 don't 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 if you used, say, a decision tree classifier). [relevant rubric item: "tune the algorithm"]

Tuning is the process of altering algorithm parameters to effect performance. Poor tuning can result in poor algorithm performance. It can also result in an algorithm that gives reasonable performance but is not efficient. For my decision tree classifier and adaboost, I used GridSearchCV along with a range a distribution of commonly influential parameters. I also tuned an adaboost classifier with the base estimator as the best performing decision tree classifier. Built in scoring was designed to yield a classifier tuned to the highest F1. I tried a custom scoring function which first selected for a threshold of recall and precision of >0.3, then selected the parameter set with the highest F1, but this was less successful than using 'f1' built in scoring.
GaussianNB does not have parameters to tune. The only "tuning" that can be accomplished with this classifier is feature selection. Both univariate feature selection and PCA dimensionality reduction boosted the recall and precision of the GaussianNB classifier.

Realizing that adaboost was my front runner I continued to try PCA and k settings as mentioned above as well as a RandomForestClassifier as the base estimator. Eventually it was clear that a combination of the 10 best features and PCA(n_components = .95) were my best transformer settings, and that the DecisonTreeBaseEstimator was outperforming the RandomForestClassifier as a base estimator. At this point, I focused on tuning adaboost with GridSearchCV, using the following parameters:

```
{'adaboost__algorithm': ('SAMME', 'SAMME.R'),
 'adaboost__base_estimator': [DecisionTreeClassifier(class_weight=None, cr
iterion='gini', max_depth=None,
              max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
              min_samples_split=2, min_weight_fraction_leaf=0.0,
              random_state=None, splitter='best')],
 'adaboost__learning_rate': [0.1, 0.5, 1, 1.5, 2, 2.5],
 'adaboost__n_estimators': [5, 10, 30, 40, 50, 100, 150, 200],
 'reduce_dim__n_components': [0.95]}
```

My final classifier consisted of this pipeline for local testing, where a tester.py modified for min/max feauture scaling was used for testing (tester_scale.py):

```
Pipeline(steps=[('pca', PCA(copy=True, n_components=0.95, whiten=False)),
('adaboost', Pipeline(steps=[('reduce_dim', PCA(copy=True, n_components=0.
95, whiten=False)), ('adaboost', AdaBoostClassifier(algorithm='SAMME.R', b
ase_estimator=None, learning_rate=2,
          n_estimators=5, random_state=None))]))])
```

and a pipeline for tester.py (modified only to import sklearn.preprocessing):

```
Pipeline(steps=[('scale_features', MinMaxScaler(copy=True, feature_range=(
0, 1))), ('pca', PCA(copy=True, n_components=0.95, whiten=False)), ('adabo
ost', Pipeline(steps=[('reduce_dim', PCA(copy=True, n_components=0.95, whi
ten=False)), ('adaboost', AdaBoostClassifier(algorithm='SAMME.R', base_est
imator=None, learning_rate=2,
          n_estimators=5, random_state=None))]))])
```

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"]

Validation is process of determining how well your model performs or fits, using a specific set of criteria. Cross-validation is used to ensure that the model generalizes well, avoiding over or under-fitting. Essentially you break the data set into a test set (often 20-30% of the data) and a training set (the remainder of the data). You fit on the training set and predict on the test set. Metrics from the test set determine your performance. When using cross-validation on small data set, it is helpful to perform this process multiple times, randomly splitting each time.

I performed validation and cross-validation of my classifiers by withholding 30% of the data for a test set. My evaluator did not accurately predict the performance of my classifiers/model with tester.py. Tester.py used 1000 folds of a cross-validator, which is very different, more rigorous than my evaluator/validator, and more appropriate for the size of the data set (small).

In tester.py I used StratifiedShuffleSplit(labels, folds=1000, random_state = 42) as a cross validator then assessed precision, recall, and F1. 10% of the data was reserved for testing. The process of fitting to training data and predicting/scoring with test data was repeated 1000 times, and the average metrics reported.

Recall, precision, and F1 score were all used as evaluation metrics. Accuracy was intentionally disqualified as a metric due to the skewed nature of the data set. Since POIs are sparse, even failing to identify any POIs would give a reasonably high accuracy score. Precision is the ratio of true positives to total positive identifications (true + false) and therefore penalizes false positive identifications. In broader terms, precision is the fraction of retrieved instances that are relevant. That is, high precision equates to a model/classifier that returns substantially more relevant than irrelevant results (more true positives per all positives). Recall is the ratio of a true positives to true positives + false negatives and therefore penalizes false negative identification. Recall is the fraction of relevant instances that are retrieved. High recall equates to a model/classifier returning most of the relevant results, albeit they may be mixed in with irrelevant results. F1 balances both precision and recall, which is why I used it for to evaluate tuning.

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"]

The best performing adaboost classifier yielded the following metrics when passed through tester_scale.py (min/max scaling before PCA as part of tester):

Precision: 0.73066 Recall: 0.39200 F1: 0.51025

When passed through tester.py (min/max scaling as part of pipeline, before PCA) the metrics were slightly lower:

Precision: 0.67914 Recall: 0.36300 F1: 0.47312

I have yet to find the cause of the discrepancy.

This model has higher precision and lower recall, with a mid-range F1 score (stands to reason considering the precision and recall). That means I tend to penalize false positives at a higher rate than false negatives. So while I am confident that the persons flagged as POIs by this model are likely to be an actual POI, some real POIs are going to slip through. I'm casting a narrow net. The argument could be made that casing a wider net (higher recall which can result in more false positives) is better because it may give investigators a larger suspect pool which to investigate. I would argue the opposite is true. Without even considering the human element (being investigated for a crime in which you are innocent is not likely pleasant), law enforcement agencies have limited resources. From a resource perspective it would be best to confidently flag a solid pool of POIs. Those POIs could then be vigorously investigated, likely leading to others who are colluding with them. That would be a more efficient and ethical use of this tool.

References (random order):

- bryantravissmith.com/udacity/udacity-project-4/
- scikit-learn.org/stable/auto_examples/plot_digits_pipe.html+&cd=1&hl=en&ct=clnk&gl=us
- scikit-learn.org/stable/modules/grid_search.html
- scikit-learn.org/stable/modules/pipeline.html#pipeline
- http://sebastianraschka.com/Articles/2014_ensemble_classifier.html#Appendix-III---GridSearch-Support
- http://napitupulu-jon.appspot.com/posts/evaluation-ud120.html
- https://www.udacity.com/course/viewer#!/c-ud120
- https://github.com/sebasibarguen/udacity-nanodegree-machinelearning#investigating-enrons-scandal-using-machine-learning

- https://github.com/allanbreyes/udacity-data-science/tree/master/p4
- http://scikit-learn.org/stable/model_selection.html#model-selection
- http://scikit-learn.org/stable/index.html
- http://www.amazon.com/Scikit-Learn-Cookbook-Trent-Hauck/dp/1783989483
- http://orenov.github.io/mlproject.html
- en.wikipedia.org/wiki/Precision_and_recall

Files:

Note to reviewers. Piping the output of poi_id.py to a text file will give a human readable output. Also, I modified tester.py to scale features with a min/max scaler.

- UD120_ShortQuestions.pdf: This document
- my_poi_id.py: most relevant work for this project, produces best_aclf_pipe.pkl
- poi_id.py: final classifier code for simplified review
- poi_id_results.txt: pipe of poi_id.py output to text file
- my_poi_id_results.txt: pipe of my_poi_id.py output to text file
- various .ipynb files: ipython notebooks of my much of my work and interactive testing
- tester.py: your evaluator/validator code modified to import sklearn.preprocessiing
- tester.scale.py: your evaluator/validator code modified for feature scaling
- enron_tools.py: helper functions
- enron_evaluate.py: evaluation function and scoring function
- best_aclf_pipe.pkl: intermediate tuned adaboost classifier for poi_id.py
- my_classfier.pkl
- my_dataset.pkl
- my_feature_list.pkl