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"]**

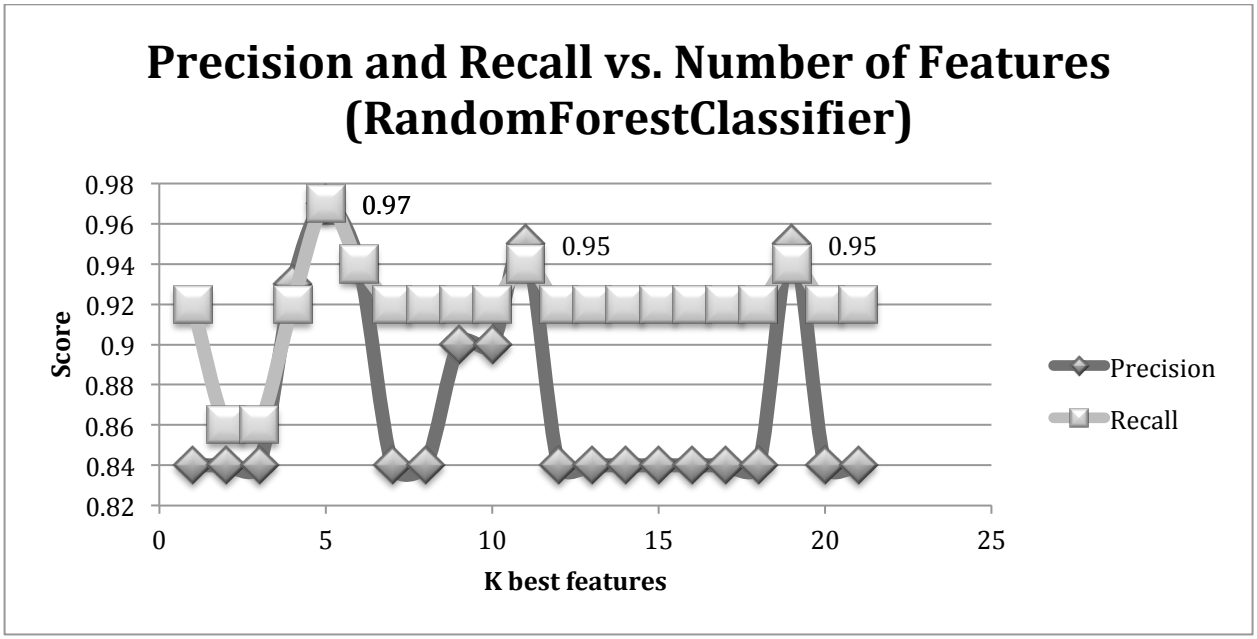
The goal of this project was to determine persons of interest (POIs), or those who have possibly committed financial fraud, from a public dataset containing the financial and email data of around 146 Enron employees. This was done by selecting and scaling relevant features, eliminating outliers, and ultimately fitting and fine-tuning a machine learning algorithm to the data in an effort to most precisely and accurately distinguish POIs. The enron_data dataset contained 146 individuals, 18 POIs (128 non-POIs), each containing 21 total financial and email features. Since I planned on using SelectKBest to select the 10 most relevant features, I included all the features in the preliminary feature list. Several features were populated with missing, or 'NaN' values:
['deferral_payments',
'restricted_stock_deferred',
'loan_advances',
'director fees',
'deferred income',
'long term incentive'].

Features with many values missing provide much less information and are less useful for classification. For example, only 3 individuals have a value under loan advances, which makes this feature useless for POI prediction.

When examining the data, I noticed 3 outlier individuals: LOCKHART EUGENE E, THE TRAVEL AGENCY IN THE PARK, and TOTAL. Every field under Eugene Lockhart was empty, The Travel Agency in the Park entry exists only to represent payments towards business-related travel to the said place, and Total just sums all of the financial data for each feature. Each of these entries were removed from the final dataset with the pop() function.

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"]

I used SelectKBest to select the top 5 relevant features. This was determined by graphing precision and recall against number of features and selected the best performing value of k. In the chart below, a value of k = 5 produced the highest precision and recall (0.97).



Here are the features and their scores in descending order:

Feature	Chi2	P-value
'exercised_stock_options'	'6.85'	'0.009'
'loan_advances'	'6.69'	'0.010'
'total_stock_value'	'5.48'	'0.019'
'bonus'	'5.12'	'0.024'
'fraction_to_poi'	'4.64'	'0.031'

Higher chi2 scores for a given df (degrees of freedom) indicate a significant dependence between features. Lower chi2 scores were not as significant, as shown by the higher p-value. Higher k value might increase the predictive power at the risk of overfitting.

Steven Yeh

I decided to use a MinMaxScaler to scale all of my selected features so that the machine-learning algorithm can proceed without being negatively affected by wildly different values for different fields. For example, 'fraction_to_poi' lies between 0 and 1, which would make it a negligible feature when compared to bonus, salary, or any other feature since their values lie in the tens or hundreds of thousands of dollars. I created two new features, 'fraction_from_poi' and 'fraction_to_poi' that calculated the fraction of total outgoing and incoming messages to POIs for each individual. The rationale behind these features is that POIs would probably tend to have a higher fraction of emails sent to other POIs, though not necessarily from POIs. It is possible that normal employees have a good volume of non-fraud related emails sent from POIs, and this is reflected in the feature selection as SelectKBest only chose 'fraction_to_poi' and not 'fraction_from_poi'. Decision trees, and subsequently RandomForest do not require feature scaling since there is no tradeoff when classifying. KNeighbors requires feature scaling since it measures distances between data points, which can be skewed towards features with a large range.

Here is the best performing algorithm (RandomForest) without the new features:

	Precision	Recall	f1-Score	Support
Not POI	0.94	0.94	0.94	33
POI	0.33	0.33	0.33	3
Avg/Total	0.89	0.89	0.89	36

With the new features, there is increased precision, recall, and F1 score when predicting POIs with the new features.

	Precision	Recall	f1-Score	Support
Not POI	0.97	1.00	0.99	33
POI	1.00	0.67	0.80	3
Avg/Total	0.97	0.97	0.97	36

Here are the feature importances for the DecisionTree, KNeighbors, and RandomForest classifiers.

	DecisionTree	Importance	RandomForest	Importance
1	shared_receipt_with_poi	0.2266	restricted_stock	0.1712
2	fraction_to_poi	0.2182	fraction_to_poi	0.1289
3	total_stock_value	0.1143	total_stock_value	0.1263
4	deferred_income	0.1077	shared_receipt_with_poi	0.1047
5	salary	0.1034	bonus	0.0886
6	total_payments	0.0689	salary	0.0875
7	exercised_stock_options	0.0646	total_payments	0.0823
8	bonus	0.0517	deferred_income	0.0778
9	restricted_stock	0.0446	exercised_stock_options	0.0730
10	long_term_incentive	0.0	long_term_incentive	0.0599

This suggests that for the DecisionTree, the first 5 features are informative, while the remaining are not. For RandomForest, the importances are much more similar to each other, making it ambiguous as to whether any of the features after restricted_stock are significantly more important.

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 tested several different algorithms: GaussianNB, DecisionTreeClassifier, KNeighborsClassifier, and RandomForestClassifier.

GaussianNB

	Precision	Recall	f1-Score	Support
Not POI	0.94	0.94	0.94	33
POI	0.33	0.33	0.33	3
Avg/Total	0.89	0.89	0.89	36

DecisionTreeClassifier

	Precision	Recall	f1-Score	Support
Not POI	0.97	0.91	0.94	33
POI	0.40	0.67	0.50	3
Avg/Total	0.92	0.89	0.90	36

KNeighborsClassifier

	Precision	Recall	f1-Score	Support
Not POI	0.92	1.00	0.96	33
POI	0	0	0	3
Avg/Total	0.84	0.92	0.88	36

RandomForestClassifier

	Precision	Recall	f1-Score	Support
Not POI	0.97	1.00	0.99	33
POI	1.00	0.67	0.80	3
Avg/Total	0.97	0.97	0.97	36

RandomForestClassifier ended up producing the best results with higher average/total precision and recall than the other classifiers. KNeighborsClassifier returned 0 precision and 0 recall. DecisionTreeClassifier returned a similar POI recall as RandomForestClassifier, but the average/total precision and recall for RandomForestClassifier was higher (Precision: 0.97 > 0.92, Recall: 0.97 > 0.89). GaussianNB returned lower precision and recall for both classifications than RandomForestClassifier.

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 algorithm parameters refers to trying different values for specific parameters within a machine-learning algorithm in order to achieve higher valuation metrics, in this case precision and recall. Every algorithm has default values for parameters, but it is not wise to blindly use these settings without considering the characteristics of the dataset one is analyzing. There are several possible consequences of failing to adequately tune parameters. In the case of algorithms utilizing decision trees (DecisionTreeClassifier), the default value of 2 for min_samples_split is not appropriate for every dataset. Small values for this parameter can often lead to overfitting, producing an inflated accuracy value, while a large value will not allow the algorithm to learn the dataset. Similarly, the max_depth parameter in the same classifier can be used to prevent overfitting by controlling the tree size. When using Support Vector Machines (SVMs) such as SVC or LinearSVC, one can increase the C parameter to increase the accuracy at the cost of boundary complexity.

Sometimes, the best way to determine the value for a parameter is to try out several values for a parameter and compare the performance of each one. I executed a pipeline through GridSearchCV to systematically execute, tune parameters, and compare the performances of several algorithms. The main parameter of the algorithm I chose, RandomForestClassifier, was n_estimators. Larger values for this parameter result in higher accuracy and higher computing times. However, these results may not be optimal, and after a critical number, the accuracy increase is negligible. I tried n_estimators values of 5, 25, and 50. The highest n_estimator of 50 produced the best accuracy, and the other values were omitted from the final pipeline to reduce computing time.

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 the method of assessing the algorithm for its effectiveness and performance, and check for issues such as overfitting. In a specific type of validation, cross-validation, the data is separated into training and testing data; in my case, 30% of the data was held for evaluating the classifier while the rest was utilized to train the algorithm to the dataset. Validation is important because it allows one to test the appropriate parameters for the algorithm on a different dataset than the trained data. The classic mistake of validation, overfitting, is a problem caused when training data and testing data are the same, which can be mitigated by cross-validation. If the two datasets were the same, then the algorithm would regurgitate the labels of the samples to which it was previously trained, providing a perfect accuracy, but no insight since no predictions on new data have been made.

I used StratifiedShuffleSplit (SSS), a cross-validation iterator, to split the data 30-70 (test_size = 0.3), reshuffle and split iterations 20 times (n_iter = 20), and set a pseudo-random number generator state (random_state = 42) to control the randomness for reproducibility. SSS creates stratified randomized folds, which is much more useful than K-fold since the latter does not preserve the distribution of each class (percentage of samples in each target class) when creating its folds, possibly creating skewed folds. SSS is more useful than StratifiedKfold since in addition to the preserving distribution, SSS also uses ShuffleSplit to randomly select data points from each class. The Enron dataset that is being examined has relatively few observations that need to be predicted, and is not large enough to use a training/validation/testing split. By using SSS to shuffle through the data and create stratified training and validation sets with every split, we can then measure the algorithm performance with an average over all splits.

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"]

I utilized 2 evaluation metrics to examine the effectiveness of the machine learning algorithm: precision and recall. Precision refers to the fraction of relevant retrieved instances, or the number of true positives over number of trues positives plus false positives. Recall is the fraction of retrieved relevant instances, or the number of true positives over true positives plus false negatives. Here is the evaluation metrics for the RandomForestClassifier below:

	Precision	Recall	f1-Score	Support
Not POI	0.97	1.00	0.99	33
POI	1.00	0.67	0.80	3
Avg/Total	0.97	0.97	0.97	36

My classifier has a weighted average of 0.97 precision and 0.97 recall. In colloquial terms, in the context of the Enron dataset, a 0.97 precision means that POIs and non-POIs are mostly identified correctly. A precision of 1.00 in the POI row means that POIs are correctly identified every time they appear. A 0.97 recall means would generally mean that the majority of results are returned. However, ignoring the weighted average recall at the individual categories, one can see that the recall for specifically POIs was only 0.67. This means that there was only a 67% chance that a randomly selected POI was identified and returned. **Therefore, when determining POIs, the algorithm had perfect precision and low recall, meaning that not many POIs were identified, but when they were identified, they were correct every single time.** All non-POIs were identified (recall = 1.00) with mostly correct identifications (precision = 0.97). This is also reflected in the F1 score of 0.99, showing that non-POIs were almost perfectly identified reliably and accurately.

Steven Yeh

This was my best achieving algorithm through tester.py:

Pipeline(steps=[('minmaxer', MinMaxScaler(copy=True, feature_range=(0, 1))),
('selectkbest', SelectKBest(k=5, score_func=<function chi2 at 0x1094a9c08>)),
('randomizedpca', RandomizedPCA(copy=True, iterated_power=3, n_components=4,
random_state=None, whiten=True)),
('gaussianNB', GaussianNB())])

Accuracy	0.84127
Precision	**0.38475**
Recall	**0.31800**
F1	0.34821
F2	0.32943
Total Predictions	15000
True Positives	636
False Positives	1017
False Negatives	1364
True Negatives	11983

I used randomizedPCA to transform the original input features into principal components to reduce dimensionality and reduce information loss. I used 4 components, or the four components with the most variance (information) as the new input features in order to balance discrimination and overfitting. This also makes the algorithm run better because of the fewer inputs.