Fraud Detection in the Enron email dataset

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, there was a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for to executives. Using Sklearn's machine learning algorithms, I detected the persons of interest that were involved in the fraud from a set of Enron emails

Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it.

The goal of the project was to find out the person of interest from a set of Enron email data by checking out the financial and email features of the dataset. Using the features and various classifiers, a predictive model is built which can be utilized to detect the Persons of Interest information from a dataset.

Were there any outliers in the data when you got it, and how did you handle those?

There were altogether 146 records. On using the outlier removal process and visualizing it, I found **"TOTAL"** as an extreme outlier.
On exploring the data more, I found two more outliers:

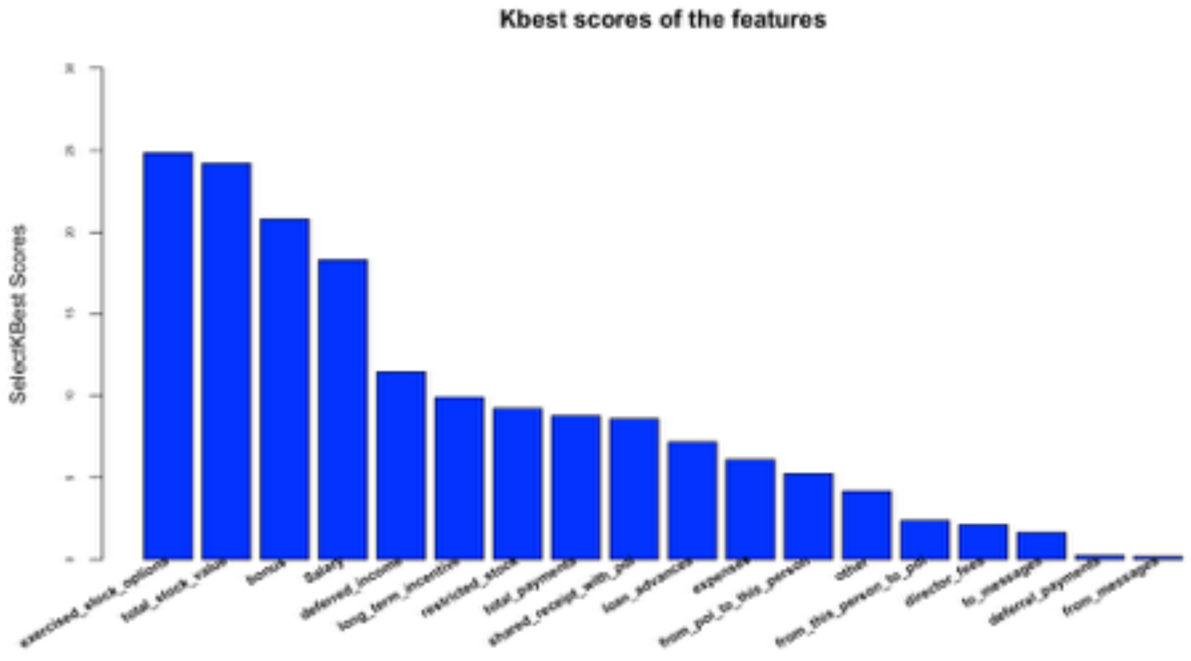
THE TRAVEL AGENCY IN THE PARK: It mostly contained NaN values, and the other data didn't make any sense.

LOCKHART EUGENE E: The fields of the this data point contained all NaN values and didn't have any relevant information.

After using the dict_object.pop(key, 0) function, to remove the outliers, a total of 143 records remained.

What features did you end up using in your POI identifier, and what selection process did you use to pick them?

Initially, I checked the best scores of all the features but by using all of them, I did not get good k values nor precision/recall scores.



Then, I used the SKlearn's SelectKBest function and selected the 10 best relevant features in the data set. The total number of relevant values for each field and their scores are as follows:

Features	Scores	Non NaN values
salary	18.29	94
total_payments	8.772	123
loan_advances	7.184	3
bonus	20.792	81
total_stock_value	24.183	125
shared_receipt_with_poi	8.589	86
exercised_stock_options	24.815	65
deferred_income	11.458	101
restricted_stock	9.212	48
long_term_incentive	9.922	109

Did you have to do any scaling? Why or why not?

I did feature scaling using MinMaxScaler(). This ensured a standardization of the units of financial and email features.

While using sklearn's SelectKBest, using different k values, I found that the precision and accuracy of most of the algorithms reported low values.
For example, in case of GaussianNB with k=10,

```
precision: 0.342252912927
recall:    0.33847958153
```

Therefore, I added two features:

poi_email_interaction: the total number of emails to and from a POI divided by the total number of emails received, score = 5.3993702880944232

financial_aggregate: The sum of exercised_stock_options, salary and total_stock_value that reveals the total wealth a POI had, score = 15.971747347749965

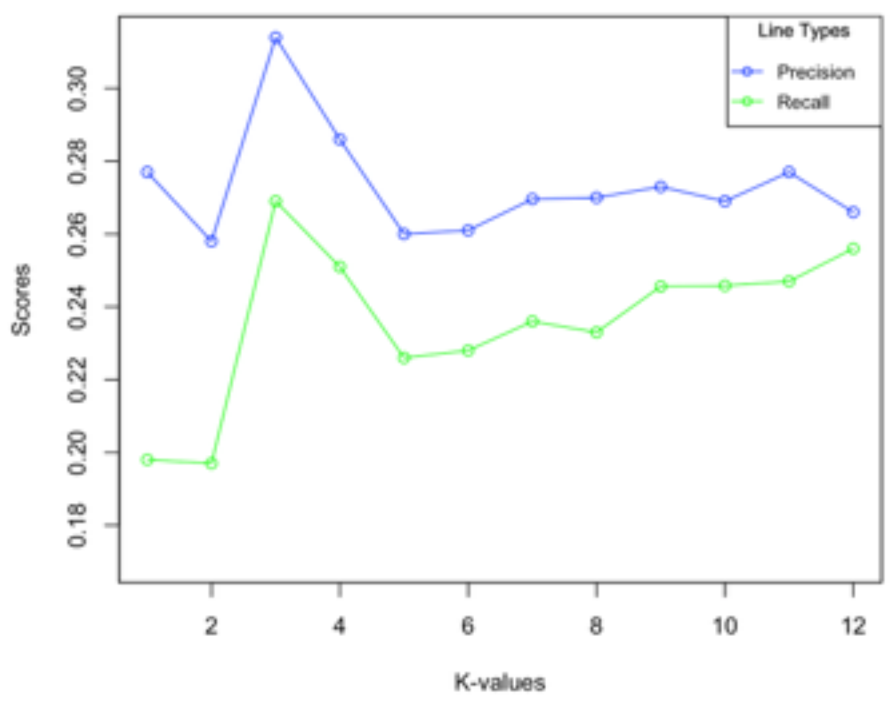
On addition of both the features, the precision and accuracy increased for the algorithms.

```
precision: 0.347019322038
recall:    0.343308946609
```

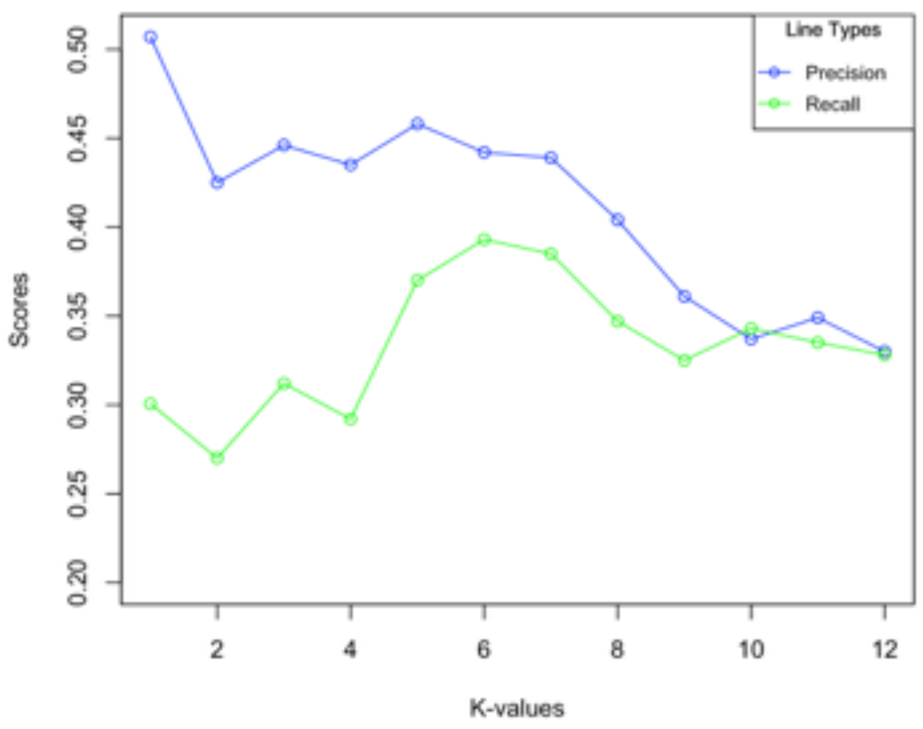
What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? [relevant rubric item: "pick an algorithm"]

The algorithm, I ended up using is the Naive Bayes. I tried several other algorithms like K-means, SVM, DecisionTree, Random Forest Classifier and Adaboost, but they didn't give me good results during validation. Although the K-means performs well in a two class problem, where the data needed to be separated in a POI and non-POI class and it gave sufficiently good results, but it is used in case of unsupervised problem and this problem is a supervised one.
I used the DecisionTree classifier with max_depth =5 and compared its results with Gaussian Naive Bayes to find out which algorithm performed better.
GaussianNB provided me with better precision, recall and accuracy results.

Precision and recall scores of a DecisionTreeClassifier



Precision and recall scores of a GaussianNB



Although, GaussianNB is the most simplest one, but as compared to DecisionTree, it gave better recall and precision scores(above 0.3). Also, from the above graph, we can see that GaussianNB gave consistent precision and score results for the k value 10. And also, a higher value of recall will be necessary in order to probe into the flagged POIs. Therefore, this led me to choose GaussianNB as my classifier with 10kBest.

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?

I tuned the parameters of the algorithms using GridSearchCV as follows:

```
Classifier:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
        metric_params=None, n_jobs=1, n_neighbors=3, p=2,
        weights='uniform')

precision: 0.216745238095
recall:    0.0931337301587
Accuracy: 0.84 (+/- 0.00)
==========================================================================


Classifier:
SVC(C=0.025, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

precision: 0.0
recall:    0.0
Accuracy: 0.88 (+/- 0.00)
==========================================================================

Classifier:
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma=2, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

precision: 0.00308333333333
recall:    0.00198333333333
Accuracy: 0.88 (+/- 0.00)
==========================================================================

Classifier:
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
          max_features=None, max_leaf_nodes=None,
        min_impurity_split=1e-07, min_samples_leaf=1,
        min_samples_split=2, min_weight_fraction_leaf=0.0,
        presort=False, random_state=None, splitter='best')

precision: 0.256217628205
recall:    0.236975829726
Accuracy: 0.74 (+/- 0.00)
==========================================================================

Classifier:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
        max_depth=5, max_features=1, max_leaf_nodes=None,
        min_impurity_split=1e-07, min_samples_leaf=1,
        min_samples_split=2, min_weight_fraction_leaf=0.0,
        n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
        verbose=0, warm_start=False)

precision: 0.290735714286
recall:    0.102880952381
Accuracy: 0.84 (+/- 0.00)
==========================================================================

Classifier:
GaussianNB(priors=None)


precision: 0.347019322038
recall:    0.343308946609
Accuracy: 0.86 (+/- 0.00)
```

Parameter tuning is important because using different values for the parameters of an algorithm will give different results. Tweaking the values of the parameters, will give better results. For example, K-neighbors performed well with n_neighbor=3 than n_neighbors = 2.

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 performed to ensure that the machine learning algorithm generalizes the data well. It is used to check whether the algorithm is performing well.
The classic mistake we can make if we validate it wrong is overfitting. The training data is overfitted and we see a decline in the performance of testing and cross validation datasets.

I validated my analysis using three methods:

evaluation.py: Here I calculated the precision and recall with parameters num_iters = 1000 and test_size = 0.3
Stratified k fold method: I used 3 folds of data in order to ensure each class is equally represented across each fold.
StratifiedShuffleSplit: The StratifiedShuffleSplit with folds = 1000 and random_state = 42.

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.

I used precision and recall as the evaluation metrics.
Precision is the retrieved fraction of the sample that can be considered relevant. It is the ability of the classifier not to label as positive a sample that is negative.

Recall is the relevant instances of the retrieved fraction from the sample. It is the ability of the classifier to find all the positive samples.

In this particular case, precision ensures that there are true positives and false alarms are not raised in the dataset and recall measures how likely the identifier will flag a POI(if present) in the dataset. Accuracy is not a good measurement since, even if there are false positives(non POIs flagged as POIs), it will yield a higher accuracy.

Validation results:
For n = 1000:

Classifier	Precision Score	Recall Score
GaussianNB	0.338433001441	0.340695815296
Decision Tree	0.257123731824	0.24471031746

Stratified k-folds(k=3)

Classifier	Precision Score	Recall Score
GaussianNB	0.350168350168	0.5
Decision Tree	0.207070707071	0.333333333333

StratifiedShuffleSplit(n_iters=1000)

Classifier	Precision Score	Recall Score
GaussianNB	0.34048	0.30950
Decision Tree	0.28837	0.248

In all the three cases, the GaussianNB yielded consistent recall and precision values throughout, which led me to use GaussianNB as the final algorithm.

List of references:

1. http://stackoverflow.com/questions/29438265/stratified-train-test-split-in-scikit-learn
2. http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShuffleSplit.html
3. http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV
4. http://stackoverflow.com/questions/28863944/scikit-learn-typeerror-if-no-scoring-is-specified-the-estimator-passed-should
5. http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/#example-3-stacked-classification-and-gridsearch
6. http://scikit-learn.org/stable/auto_examples/model_selection/grid_search_digits.html
7.