MACHINE LEARNING PROJECT: ENRON FRAUD

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?**

Intro

The Enron fraud case is the biggest such phenomenon in the American history. The objective of this project is to use algorithms to detect the persons of interest (POI). A POI is someone who was settled without admitting guilt or indicted.

Data statistics

- Total number of data points equals 146
- Total number of features equals 21
- Each feature and count of missing values:

{'bonus': 64,
'deferral_payments': 107,
'deferred_income': 97,
'director_fees': 129,
'email_address': 35,
'exercised_stock_options': 44,
'expenses': 51,
'from_messages': 60,
'from_poi_to_this_person': 60,
'from_this_person_to_poi': 60,
'loan_advances': 142,
'long_term_incentive': 80,
'other': 53,
'poi': 0,
'restricted_stock': 36,
'restricted_stock_deferred': 128,
'salary': 51,
'shared_receipt_with_poi': 60,
'to_messages': 60,
'total_payments': 21,
'total_stock_value': 20}

- POI: 18
- Non-POI: 128

Outliers

We define outliers the features that are less than the 5% quantile or more than the 95% quantile. For instance, these are observations of 'bonus' less than the 5% quantile:

[('SULLIVAN-SHAKLOVITZ COLLEEN',), ('REYNOLDS LAWRENCE',), ('DODSON KEITH',)].

In the outlier_investigation.py file we have all the instances of outliers.

'Total_payments' and 'total_stock_value' are the two features that have the least missing values. Together with 'POI', these are 3 features that should have data for each data point.

These are the data points that have less than 4 valid values:

- WHALEY DAVID A: 3
- WROBEL BRUCE: 3
- LOCKHART EUGENE: 1
- THE TRAVEL AGENCY IN THE PARK: 3
- GRAMM WENDY L: 3

  We keep the observations WHALEY DAVID A, WROBEL BRUCE and GRAMM WENDY because they each had one financial value, total amount and POI.
  We remove however, LOCKHART EUGENE and THE TRAVEL AGENCY IN THE PARK.
  The reason is that LOCKHART EUGENE only has a POI value and THE TRAVEL AGENCY IN THE PARK is not a person.
  We also remove TOTAL because it is just the sum of the financial data, and not a person.

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.**

Feature Selection and Engineering

We initially selected the features using SelectKBest and select the ones with the highest k scores. We engineered 3 new features: cash_received, cash_received_pctg and poi_messages_pctg. The reason was to combine the financial features and express them as percentages of total_payments and total_stock_value. Besides that, poi_messages_pctg would show the percentage of POI message from the total number of messages. We expect a POI to exchange more messages with POI than with non-POI.

These are the k-scores for the features:

{'bonus': 20.792252047181531,
'cash_received': 19.587427711132936,
'cash_received_pctg': 3.0812591882779259,
'deferral_payments': 0.22461127473600523,
'deferred_income': 11.458476579279765,
'director_fees': 2.126327802007705,
'exercised_stock_options': 24.815079733218212,
'expenses': 6.0941733106389337,
'from_messages': 0.16970094762175433,
'from_poi_to_this_person': 5.2434497133749636,
'from_this_person_to_poi': 2.3826121082276743,

'loan_advances': 7.1840556582887247,
'long_term_incentive': 9.9221860131898243,
'other': 4.1874775069953794,
'poi_messages_pctg': 5.3993702880944179,
'restricted_stock': 9.2128106219770771,
'restricted_stock_deferred': 0.06549965290989139,
'salary': 18.289684043404527,
'shared_receipt_with_poi': 8.5894207316823774,
'to_messages': 1.6463411294419978,
'total_payments': 8.7727777300916738,
'total_stock_value': 24.182898678566886}

We removed 4 features with the lowest k scores: deferral_payments (0.23), restricted_stock_deferred (0.07), director_fees (2.13), and other (4.19).
We can see that the features used to compute the new feature cash_received had the highest k scores. Cash_received_pctg had a score of only 3.08 – much lower than cash_received (19.59) and total_value (16.99). Hence we removed cash_received_pctg, but we included cash_received in the initial model.
The scaled feature poi_messages_pctg has the highest k score among the email features. In addition to shared_receipt_with_poi, poi_messages_pctg were the only email features included in the initial model.
We rescaled all features using sklearn min/max scaler since the finance and email data points were measured by different units and on different scales.

3) **What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?**
We tested 4 algorithms after we removed the features we indicated in the response above. Here are the results:

Algorithm	**Accuracy**	**Precision**	**Recall**
GaussianNB	0.77	0.24	0.32
SVC	NA	NA	NA
DecisionTreeClf	0.81	0.29	0.30
KNeighborsClf	0.88	0.64	0.17

We added the 3 new features we created. Here are the results:

Algorithm	**Accuracy**	**Precision**	**Recall**
GaussianNB	0.77	0.23	0.31
SVC	NA	NA	NA
DecisionTreeClf	0.82	0.31	0.30
KNeighborsClf	0.89	0.77	0.29

Only KNeighborsClassifier had real changes in results. We removed the features bonus, exercised_stock_options, loan_advances and salary because cash_received already represents the summation of these. This way we reduce the number of features. Here are the updated results:

Algorithm	Accuracy	Precision	Recall
GaussianNB	0.85	0.40	0.31
SVC	NA	NA	NA
DecisionTreeClf	0.82	0.34	0.33
KNeighborsClf	0.89	0.84	0.22

GaussianNB had the best improvement among all the algorithms.
DecisionTreeClassifier will be tuned because precision and recall exceed 0.3.
KNeighborsClassifier is the best choice, but it still needs a higher recall.

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? What parameters did you tune? (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).**

We tune the parameters of an algorithm to improve its performance.
If we do not tune the model properly, then we will not have the optimal performance of the algorithm with the features we selected. Namely, if we do not tune the model enough, it can suffer from underfitting. I.e. the model will not capture the trend in the dataset.
And the opposite can happen if we tune the model too much. I.e. if the model captures perfectly the trend in our current dataset, it will not replicate its performance in other datasets because it cannot generalize.
We had zero true positives and zero false positives for SVC. Hence the results were NA, due to division by zero. With recursive feature elimination, we found that there was 1 optimal feature (cash_received). We removed all the other features and we used a linear kernel and 1000 max_iter. Here is the performance:

Algorithm	Accuracy	Precision	Recall
SVC	0.46	0.14	0.51

Regarding DecisionTreeClf and KNeighborsClf, we used StratifiedKFold cross-validation for GridSearchCV. This is the output of features_importance for the DecisionTreeClf:

- Cash_received: 0.42
- Deferred_income: 0.15
- Expenses: 0.03
- Long_term_incentive: 0
- Poi_messages_pctg: 0.22
- Restricted_stock: 0.10
- Shared_receipt_with_poi: 0
- Total_payments: 0.04
- Total_stock_value: 0.04

We removed long_term_incentive and shared_receipt_with_poi due to their lack of importance. These are the updated results:

Algorithm	Accuracy	Precision	Recall

DecisionTreeClf	0.82	0.32	0.29

This is the updated output for the features_importance for DecisionTreeClf:

- Cash_received: 0.42
- Deferred_income: 0.11
- Expenses: 0.04
- Poi_messages_pctg: 0.16
- Restricted_stock: 0.16
- Total_payments: 0.04
- Total_stock_value: 0.09
  We iterate again by removing the least important features: deferred_income, expenses, total_payments and total_stock_value. This is the updated performance:

Algorithm	**Accuracy**	**Precision**	**Recall**
DecisionTreeClf	0.83	0.40	0.39

The performance of the DecisionTreeClf clearly improved after removing the features with least importance.

With the same logic, we will tune the KNeighborsClassifier.

We run GridSearchCV to find the best estimator parameters for n_neighbors, weights, algorithm, metric and leaf_size. This is the output:

KNeighborsClassifier(algorithm = 'auto', leaf_size = 5, metric = 'mikowski', metric_params = None, n_neighbors = 10, p = 2, weights = 'distance')

This is the performance of the model:

Algorithm	**Accuracy**	**Precision**	**Recall**
KNeighborsClf	0.87	0.91	0.02

Accuracy and precision improved a bit, but recall is still poor. We removed the features with the lowest k scores (deferred_income, expenses, long_term_incentive, poi_messages_pctg, and restricted_stock). GridSearchCV now recommend 2 n-neighbors and uniform weights.

KNeighborsClassifier(algorithm = 'auto', leaf_size = 5, metric = 'mikowski', metric_params = None, n_neighbors = 2, p = 2, weights = 'uniform')

This is the updated performance of the model:

Algorithm	**Accuracy**	**Precision**	**Recall**
KNeighborsClf	0.89	0.68	0.27

Accuracy and recall improved but precision suffered. Now we try the default values of 30 leaf_size and 5 n_neighbors.

This is the updated performance of the model:

Algorithm	Accuracy	Precision	Recall
KNeighborsClf	0.90	0.80	0.30

All the scores improved. Now we try to add back the email feature poi_messages_pctg.

This is the updated performance of the model:

Algorithm	Accuracy	Precision	Recall
KNeighborsClf	0.90	0.80	0.30

There is no change in results. Now we remove shared_receipt_with_poi.

This is the updated performance of the model:

Algorithm	Accuracy	Precision	Recall
KNeighborsClf	0.90	0.81	0.31

There is a slight improvement in precision and recall.

These are the final performances of the DecisionTreeClf and KNeighborsClf:

Algorithm	Accuracy	Precision	Recall
DecisionTreeClf	0.83	0.40	0.39

Algorithm	Accuracy	Precision	Recall
KNeighborsClf	0.90	0.81	0.31

We will work with KNeighborsClf as the final algorithm.

5) **What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?**

During validation, we split the dataset intro training and testing data to test the algorithm performance. The reason for that is to avoid overfitting. We do not want a model that perfectly predicts on a certain dataset, but cannot replicate the performance on a different dataset. We want an algorithm that can capture the trend in the data.

We used StratifiedShuffleSplit that returns stratified randomized folds in order to keep the representation of each class in the sample. The tester.py uses 1000 iterations of the StratifiedShuffleSplit. The accuracy, precision and recall scores are an average of all iterations.

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.**

Accuracy is the number of accurate predictions divided by all the data points. KNeighborsClassifer had 621 true positives and 12851 true negatives. Relative to 15000 datapoints this leads to an accuracy of 89.8%.

Precision is true positives divided by true positives plus false negatives.

Recall is true positives divided by true positives plus false negatives. In the context of KNeighborsClassifier, recall is not its strongest asset. SVC was better at it, but we looked at the aggregate performance of the model to choose KNeighborsClassifier as the best option. KNeighborsClassifier had 621 true positives and 149 false positives. This leads to a precision of 80.6%.

KNeighborsClassifier has great precision. I.e if there is POI in the dataset, then it is very likely to be a true POI, and not a mistake. The low recall of KNeighborsClassifier implies that the model is not guessing very well when it's on the edge of POI classification.