Revised P5: Enron Submission Free-Response Questions

1. **Summarize the goal of this project and how machine learning is useful in trying to accomplish it. As part of the 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, and how were they handled?**

   In this project, I will use machine learning techniques to build an algorithm to identify Enron Employees who may have committed fraud (POIs) based on the public Enron financial and email dataset. The dataset combines the Enron email and financial data into a dictionary, where each key-value pair in the dictionary corresponds to one person. The dictionary key is the person's name, and the value is another dictionary, which contains the names of all the features and their values for that person. The goal of this project is to engineer the features, pick and tune an algorithm, and to test and evaluate the identifier.

   There are originally 146 datapoints (i.e. persons) in the dataset, out of which 18 are allocated to POI. There are 21 features in the dataset which fall into three major types, namely 14 financial features, 6 email features, and the POI labels. By making 2D plots of financial feature values, I found three candidates of outliers, 'TOTAL', 'LAY KENNETH L', and 'BHATNAGAR SANJAY', which all have extremely large or small values for some features. I removed 'TOTAL' since it is not a datapoint. I also removed 'BHATNAGAR SANJAY' because I found that its feature values may have been mistyped by comparing them to the data in the provided file 'enron61702insiderpay.pdf'. On the other hand, I left 'LAY KENNETH L' as it was.

   Additional comments for resubmission:

   I have included three eps plots in the zip file showing outlier candidates (I could not include them in this document due to some misplacement errors). 'outlier_TOTAL.eps' shows the outlier 'TOTAL' in the original dataset. 'outlier_SANJAY.eps' shows the outlier 'BHATNAGAR SANJAY' in the dataset after removing 'TOTAL'. 'outlier_KENNETH.eps' shows the outlier candidate 'LAY KENNETH L' in the dataset after removing both 'TOTAL' and 'BHATNAGAR SANJAY', which I did not remove.

   As for the outlier 'BHATNAGAR SANJAY', I printed out its feature-value pairs and got:

```
BHATNAGAR SANJAY: 'salary': 'NaN', 'deferral_payments': 'NaN', 'total_payments':
15456290, 'exercised_stock_options': 2604490, 'bonus': 'NaN', 'restricted_stock':
-2604490, 'restricted_stock_deferred': 15456290, 'total_stock_value': 'NaN',
'expenses': 'NaN', 'loan_advances': 'NaN', 'other': 137864, 'director_fees':
137864, 'deferred_income': 'NaN', 'long_term_incentive': 'NaN',
```

   while the correct values in the 'enron61702insiderpay.pdf' are different for some features:

```
BHATNAGAR SANJAY: 'salary': 'NaN', 'deferral_payments': 'NaN', 'total_payments':
137864, 'exercised_stock_options': 15456290, 'bonus': 'NaN', 'restricted_stock':
 2604490, 'restricted_stock_deferred': -2604490, 'total_stock_value': 15456290,
'expenses': 137864, 'loan_advances': 'NaN', 'other': 'NaN', 'director_fees':
'NaN', 'deferred_income': 'NaN', 'long_term_incentive': 'NaN',
```

   which justifies the removal of this key as an outlier.

   I also decided to remove 'THE TRAVEL AGENCY IN THE PARK' suggested by the reviewer, which is apparently not a person name. Using 'total_payments' and 'total_stock_value' as the tentative input features and Gaussian Naive Bayes as the tentative algorithm, I got the performance (Precision, Recall, and F1. Detailed afterward in Q. 6) shown in Table 1 before and after each outlier removed. The performance significantly increased by subtracting the outliers from the original dataset.

Table 1: Performance after removing outliers

Removed Outliers	Precision	Recall	F1
a) None	0.08395	0.06200	0.07133
b) ‘TOTAL’	0.38371	0.17900	0.24412
c) b + ‘BHATNAGAR SANJAY’	0.48148	0.22100	0.30295
d) c + ‘THE TRAVEL AGENCY IN THE PARK’	0.49672	0.22750	0.31207

There are also numerous features with many missing values in the dataset. For instance, ‘loan_advances’ is empty for 142 keys, which is more than 97% of the total datapoints. Likewise ‘restricted_stock_deferred’ is empty for 128 and ‘deferral_payments’ is empty for 107. However, I did not exclude those features in the following analysis since it was not apparent to me whether if they were bad features in terms of predicting POIs (i.e. I thought the features with many missing values could be a powerful indicator for POIs).

2. **What features were used in the POI identifier, and what selection process was used 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.**

This part was totally revised for resubmission:

I used Decision Tree without any parameters specified for feature selection as it has an useful attribute ‘feature_importances_’ that allows us to figure out the importance of features. First I put all the 14 financial features into the algorithm and got the top-5 important features shown below in Table 2. The importances for the other features are significantly lower than them.

Table 2: Top 5 important features with Decision Tree classifier

Features	Abbreviation	Importance
exercised_stock_options	eso	0.22
total_payments	tp	0.18
bonus	b	0.16
restricted_stock	rs	0.134
long_term_incentive	lti	0.133

Next I calculated the performance of the algorithm using one to five of the top 5 features. The results are shown in Table 3, from which I concluded that the combination of ‘exercised_stock_options’, ‘total_payments’, and ‘bonus’ provides a good features list.

Table 3: Performance using some of top 5 features

Features used	Precision	Recall	F1
eso	0.31063	0.34200	0.32556
eso + tp	0.31317	0.30550	0.30929
eso + tp + b	0.33824	0.35600	0.34689
eso + tp + b + rs	0.30030	0.30000	0.30015
eso + tp + b + rs + lti	0.26332	0.25950	0.26140
All 14 financial features	0.25648	0.25250	0.25447

In addition to the three financial features, I examined three fractional features 'fraction_from_poi', 'fraction_to_poi', and 'fraction_shared_with_poi' all created from email features. The reason why I engineered new features is that, although the numbers of messages to, from, or shared with POIs are important clues to identify POIs, we should treat them as a fraction to the total number of messages rather than their absolute numbers. I made 2D plots of the new features to investigate outliers but there were nothing clear found.

The performance change when adding each of the three new features to the three financial features found above is shown in Table 4. We can see that to add 'fraction_shared_with_poi' to the features improves the performance, while 'fraction_from_poi' and 'fraction_to_poi' don't help much.

Table 4: Performance when adding new features

Features used	Precision	Recall	F1
eso + tp + b + fraction_from_poi	0.28855	0.27600	0.28214
eso + tp + b + fraction_to_poi	0.32250	0.30750	0.31482
eso + tp + b + fraction_shared_with_poi	0.38907	0.35250	0.36988

Starting from these four features (eso + tp + b + fraction_shared_with_poi) , I carried out feature scaling (using MinMaxScaler), dimension reduction through PCA (using RandomizedPCA), and grid search of parameters (using GridSearchCV) for three different algorithms, namely Gaussian Naive Bayes, Decision Tree, and Support Vector Machine. More precisely, I also included the parameter 'n_components' of PCA in the grid search. I applied feature scaling to all the three algorithms while I thought only SVM strictly requires it.

3. **What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?**

This part was totally revised for resubmission:

I tried Gaussian Naive Bayes, Decision Tree, and SVM. As a first step I ran each algorithm with the following parameters, respectively.

Gaussian NB: n_components = None for PCA

Decision Tree: n_components = None for PCA, min_samples_split = 2

SVM: n_components = None for PCA, C = 10, gamma = 10

And the results are as follows.

Table 5: Difference in model performance between algorithms

Algorithms	Precision	Recall	F1
Gaussian NB	0.46995	0.25800	0.33312
Decision Tree	0.31765	0.27350	0.29393
SVM	0.09213	0.02050	0.03354

*Features are: eso + tp + b + fraction_shared_with_poi

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?**

Most machine learning algorithms have parameters to specify under which conditions they train the data. The algorithms can overfit or underfit the data when the parameters are not well tuned. To tune the three algorithms, I carried out a grid search with a scoring by F1 value (explained below in Q. 6) for each algorithm with the following parameter spaces respectively.

Gaussian NB: n_components $= 2, 3, 4$ for PCA

Decision Tree: n_components $= 2, 3, 4$ for PCA, min_samples_split $= 2, 3, 5$

SVM: n_components $= 2, 3, 4$ for PCA, C $= 10, 100, 1000$, gamma $= 10, 100, 1000$

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

   Validation is a method to estimate how well the machine learning algorithm is doing. A typical mistake when doing validation is to use the entire data for training the algorithm, which does not give a good indication of its performance when asked to make new predictions for data it has not already seen.

   To overcome this problem the data was split into training set and test set in this project, which is called cross-validation. Furthermore, the grid search I used for tuning the parameters employs a k-fold (k = 3) cross-validation, where the data is divided into k subsets and the cross-validation is repeated k times. Then the average performance across the k trials was computed.

   Additional comments for resubmission:

   Moreover, 'tester.py' provided in this project employs stratification (preservation of the percentage of samples both for POI and non-POI) when splitting train and test dataset. This is to mitigate the risk that we could not assess the real performance of the algorithms due to the small and skewed-towards-non-POI dataset. The chance of getting skewed and no representative sets could be high if the dataset is randomly splitted.

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.**

   To evaluate the algorithms I used three metrics: Precision, the fraction of predicted POIs that are actually POIs ($\frac{\text{TruePositives}}{\text{TruePositives}+\text{FalsePositives}}$), and Recall, the fraction of POIs that are predicted correctly ($\frac{\text{TruePositives}}{\text{TruePositives}+\text{FalseNegatives}}$), as well as F1 score ($2.0 \times \frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$). F1 score can be interpreted as a weighted average of the Precision and Recall.

   Since there is a tradeoff between Precision and Recall I used the F1 scores as the primary metrics to evaluate algorithms. The best F1 scores (averaged through k-fold cross-validation) I found and the corresponding parameters for each algorithm are in Table 6.

   The following results were revised for resubmission:

Table 6: Best performance and parameters for each algorithm

Algorithms	Precision	Recall	F1	Parameters
Gaussian NB	0.53757	0.27900	0.36735	n_components = 2 for PCA
Decision Tree	0.32663	0.26000	0.28953	n_components = 3 for PCA, min_samples_split = 3
SVM	0.34663	0.26500	0.30037	n_components = 4 for PCA, C = 1000, gamma = 10

*Features are: eso + tp + b + fraction_shared_with_poi.

These results were not satisfactory to me not only because none of them had Precision and Recall both better than 0.3 but because the performance was even worse than that shown in the third row of Table 4, which was before introducing PCA and any tuning. So finally I decided not to use PCA and reran the tuning. Based on the previous results, I adjusted the parameter spaces for Decision Tree (min_samples_split $= 1, 2, 3, 4$) and SVM (C $= 1000, 10000, 100000$, gamma $= 1, 5, 10$). The new results are shown in Table 7. Decision Tree gives us the best performance, where Precision and Recall are both > 0.3.

Table 7: Best performance and parameters for each algorithm without PCA.

Algorithms	Precision	Recall	F1	Parameters
Gaussian NB	0.55851	0.28400	0.37653	
Decision Tree	0.39800	0.33750	0.36526	min_samples_split = 2
SVM	0.35419	0.29150	0.31980	C = 10000, gamma = 5

*Features are: eso + tp + b + fraction_shared_with_poi.