Enron POI Report

Question 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? </style>

Overview

This project provided partially conclusive findings of suspected Enron employees in the 2002 Enron fraudlent scenario, **Person of Interest(POI)**, by using machine learning techniques.

> We define POI as someone who was indictedor settled in the case without admitting guilt. We utilized several techniques to come closer to answering the previously mentioned goal.

The Data

The dataset we operated on was JSON formatted information on Enron employee financial statuses and email interactions. These features were integral in answering our question from the following reasoning:

- Financial information is a central theme around industry corruption.
- Communicating with a corrupt individual quite often is an indicator of aiding in abetting.

To solve for POI concerns, we turned to machine learning algorithms for identifying corruption with our financial and email information. A classical approach in machine learning is e-mail classification from logistic regression within supervised learning. However, we implemented two other supervised learning algorithms to determine Enron POI.

However, to implement these algorithms, we inspected the data for:

1. Misinformation (typos)
2. Lack of information (Null values)
3. Obscure information (outliers)

Handling Outliers and more

Using the inter-quartile range, we removed obscure data from our analysis and imputated some information.

> We removed outlier entries such as "TOTAL" and two other entries due to "NaN" or absurd information recorded. This removal of infectious data enabled us to proceed with the analysis in better fashion than before.

In the case of removing outliers, I just set some condition for extracting/neglecting outlier information as we update the dataset.

In the case of imputing data, I created a function to replace existing "NaN" values with the integer "0." Thereafter, we verified the changes in our introductory data analysis of Salary versus Bonus.

Question 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.

We select several numerical features. Moreoever, we factor in the four engineered features, seen in the following subsection

> We will implement four new features to potentially indentify Eron POI.
>
> The **first two** additions are the ratio between an employee and the CEO, *Jeffrey K. Skilling*.
>
> The implementation for these four new features can lead to the following:
>
> - If the ratio in bonus between an employee and the CEO is significantly closer to the value 1, then we can suspect some type of corruption occuring.
> - If the ratio in salary between an employee and the CEO is significantly closer to the value 1, then we can suspect some type of corruption occuring.
>
> These features are linear transformations of the salary and bonus information presented. We factor how close ones's financial information is with respect to the CEO to determine POI relationship.
>
> The **remaining two** additional features are the inbound and outbound email ratios between a POI.
>
> The implementation for these four new features can lead to the following:
>
> - If the ratio $\frac{\text{Received emails from POI}}{\text{All received Emails}}$ is closer to the value 1, we can suspect an individual is a POI.
> - If the ratio $\frac{\text{Sent emails from POI}}{\text{All Sent Emails}}$ is closer to the value 1, we can suspect an individual is a POI.

We then implemented a few algorithms to decide on what top three to four features we should implement for predicting people of interest in the Enron email scandal.

I.e. we zoom out to consider the following features, then zoom in to precise feature selections to predict Enron fraudsters.

1. POI
2. CEO to Employee Bonus Ratio
3. Total Payments
4. Exercised Stock Options

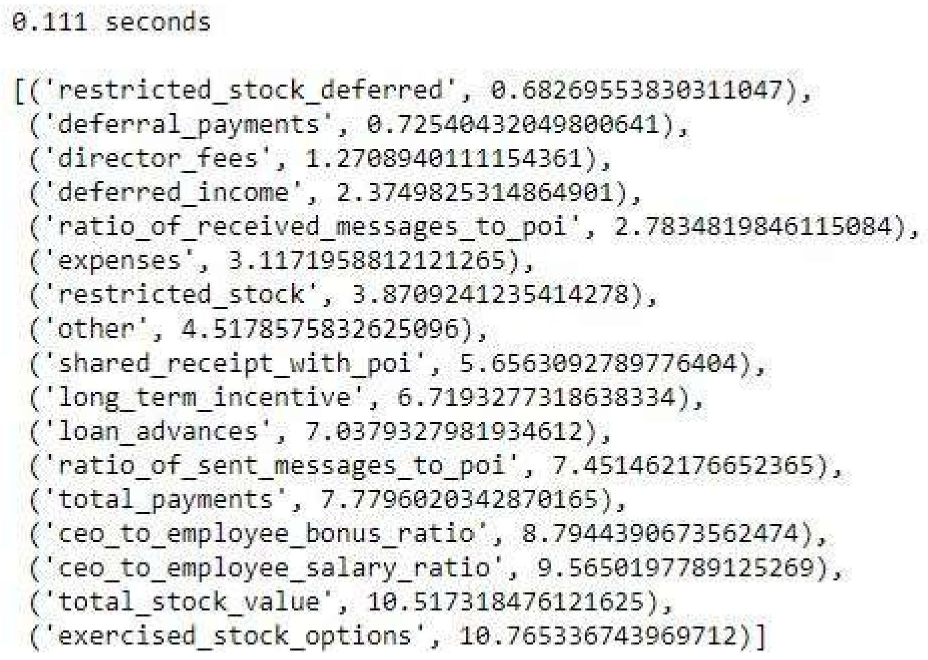
5. CEO to Employee Salary Ratio
6. Restricted Stock
7. Shared Receipt with POI
8. FROM POI to This Person
9. From Messages
10. From this Person to POI
11. Ratio of Sent Messages to POI
12. Ratio of Received Messages to POI
13. Deferral Payments
14. Loan Advances
15. Restricted Stock Deferred
16. Deferred Income
17. Expenses
18. Other
19. Long Term Incentive
20. Director Fees

Feature Selection with SKBest Procedure

We implement the SelectKBest (http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) method for selecting features according to the **k** highest scores. These k factors will be one of our few considerations for feature selection.

> Added: Per correction/comment, 'k' looks at the features after score computation and keeps the top features for next fitting process.

The following are the ranked features with the parameter **k**='all':



```
0.111 seconds

[('restricted_stock_deferred', 0.68269553830311047),
 ('deferral_payments', 0.72540432049800641),
 ('director_fees', 1.2708940111154361),
 ('deferred_income', 2.3749825314864901),
 ('ratio_of_received_messages_to_poi', 2.7834819846115084),
 ('expenses', 3.1171958812121265),
 ('restricted_stock', 3.8709241235414278),
 ('other', 4.5178575832625096),
 ('shared_receipt_with_poi', 5.6563092789776404),
 ('long_term_incentive', 6.7193277318638334),
 ('loan_advances', 7.0379327981934612),
 ('ratio_of_sent_messages_to_poi', 7.451462176652365),
 ('total_payments', 7.7796020342870165),
 ('ceo_to_employee_bonus_ratio', 8.7944390673562474),
 ('ceo_to_employee_salary_ratio', 9.5650197789125269),
 ('total_stock_value', 10.517318476121625),
 ('exercised_stock_options', 10.765336743969712)]
```

we observe that CEO to Employee Bonus Ratio and CEO to Employee Salary Ratio ranked within top 5 features for our model selection, as seen below:

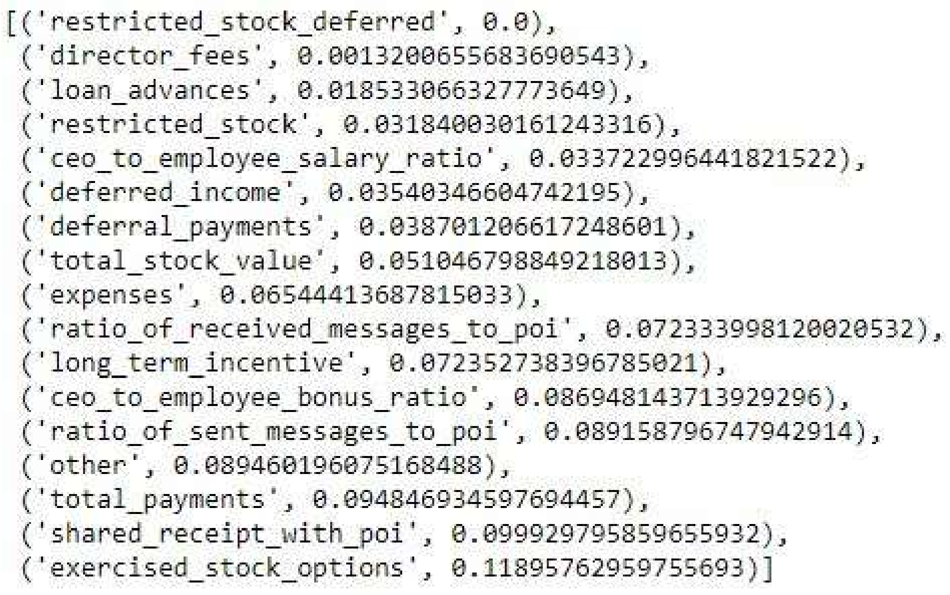
1. Exercised Stock Options

2. Total Stock Value
3. CEO To Employee Salary Ratio
4. CEO To Employee Bonus Ratio
5. Total Payments

Feature Selection with Extra Trees Implementation

The Extra Trees classifier is a variant of the popular Random Forest algorithm. However, each step of the Extra Trees implementation has random decision boundaries selected, rather than the best one. Moreover, Extra Trees classifier is great for our numerical features.

The following are the ranked features with no altered parameters, default parameters.



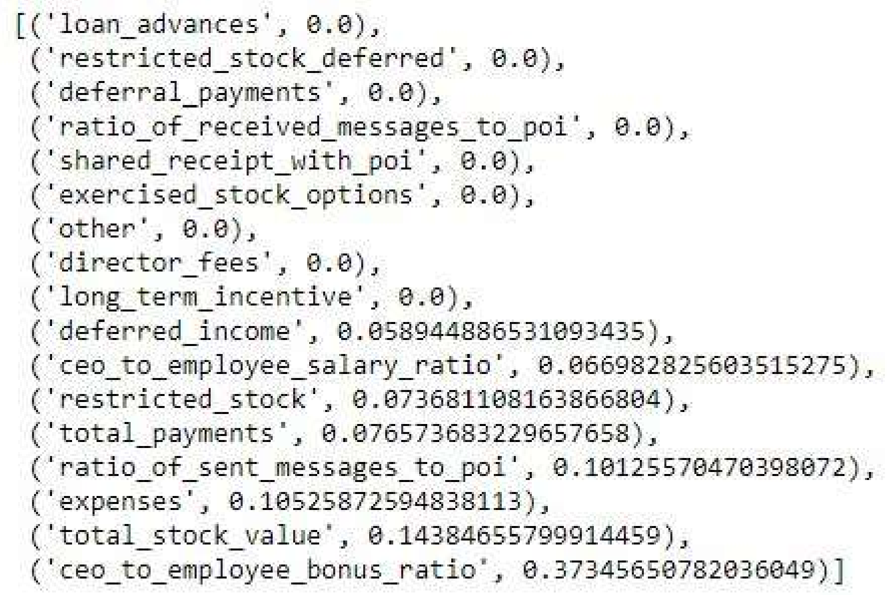
```
[('restricted_stock_deferred', 0.0),
 ('director_fees', 0.0013200655683690543),
 ('loan_advances', 0.018533066327773649),
 ('restricted_stock', 0.031840030161243316),
 ('ceo_to_employee_salary_ratio', 0.033722996441821522),
 ('deferred_income', 0.03540346604742195),
 ('deferral_payments', 0.038701206617248601),
 ('total_stock_value', 0.051046798849218013),
 ('expenses', 0.06544413687815033),
 ('ratio_of_received_messages_to_poi', 0.072333998120020532),
 ('long_term_incentive', 0.072352738396785021),
 ('ceo_to_employee_bonus_ratio', 0.086948143713929296),
 ('ratio_of_sent_messages_to_poi', 0.089158796747942914),
 ('other', 0.089460196075168488),
 ('total_payments', 0.094846934597694457),
 ('shared_receipt_with_poi', 0.099929795859655932),
 ('exercised_stock_options', 0.11895762959755693)]
```

Our findings within the Extra Trees implementation had Exercised Stock Options as our top feature recommendation.

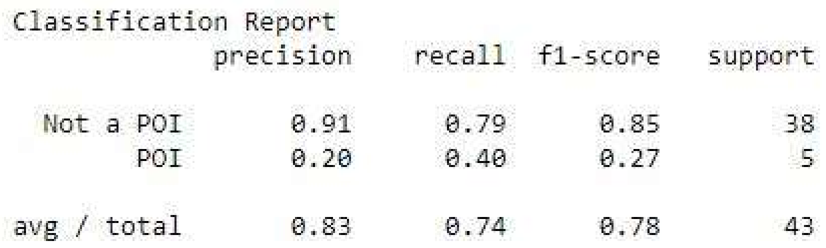
Feature Selection with Decision Tree Algorithm

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

The following are the ranked features with no altered parameters, default parameters.



```
[('loan_advances', 0.0),
 ('restricted_stock_deferred', 0.0),
 ('deferral_payments', 0.0),
 ('ratio_of_received_messages_to_poi', 0.0),
 ('shared_receipt_with_poi', 0.0),
 ('exercised_stock_options', 0.0),
 ('other', 0.0),
 ('director_fees', 0.0),
 ('long_term_incentive', 0.0),
 ('deferred_income', 0.058944886531093435),
 ('ceo_to_employee_salary_ratio', 0.066982825603515275),
 ('restricted_stock', 0.073681108163866804),
 ('total_payments', 0.076573683229657658),
 ('ratio_of_sent_messages_to_poi', 0.10125570470398072),
 ('expenses', 0.10525872594838113),
 ('total_stock_value', 0.14384655799914459),
 ('ceo_to_employee_bonus_ratio', 0.37345650782036049)]
```



```
Classification Report
             precision    recall  f1-score   support

  Not a POI       0.91      0.79      0.85        38
        POI       0.20      0.40      0.27         5

avg / total       0.83      0.74      0.78        43
```

For ranking the importance of all recommended features, The top important features are:

1. Total Stock Value
2. CEO to Employee Bonus Ratio
3. Expenses

Our top features only have one commmon feature in common from the results of our other features. This feature is CEO to Employee Bonus Ratio.

Moreover, this scenario has a precision of 0.20 and recall of 0.40 for determining POI. This is partially good to see, however we should observe that this, and the past outcomes, is not optimal for model implementation. I.e.,Notice that we implemented the Decision Tree algorithm with default parameters, and previous algorithms as well. This consideration occurs because we do not know if our feature selection process was optimal in selection.

Feature Selection: Final Decision

The top re-occuring features to select from where, in frequency:

1. CEO To Employee Bonus Ratio
2. CEO To Employee Salary Ratio
3. Exercised Stock Options

Additionally, we include two additionaly engineered features:

1. ratio_of_received_messages_to_poi
2. ratio_of_sent_messages_to_poi

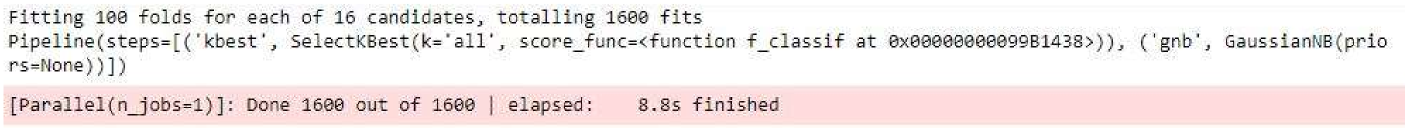
Question 3

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

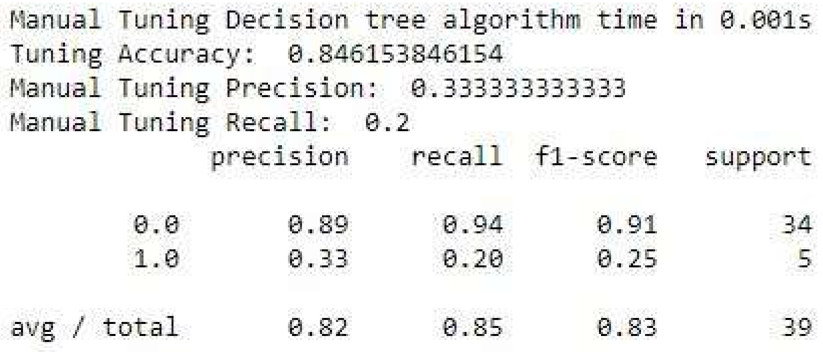
We implemented SelectKbest, Extra Trees, and Decision Tree algorithm for obtaining an optimal feature count for model fitting and ranking features to best classify POI of Enron employees.

Thereafter, we compared and contrasted between Gaussian Naive Baye's algorithm and Decision Tree implementions to best predict POI.

The following are the results from the Gaussian Naive Baye's implementation:



```
Fitting 100 folds for each of 16 candidates, totalling 1600 fits
Pipeline(steps=[('kbest', SelectKBest(k='all', score_func=<function f_classif at 0x00000000099B1438>)), ('gnb', GaussianNB(prio
rs=None))])
[Parallel(n_jobs=1)]: Done 1600 out of 1600 | elapsed:    8.8s finished
```

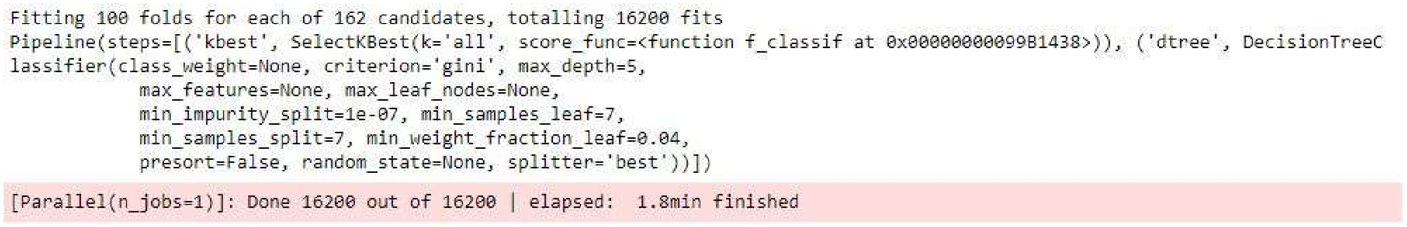


```
Manual Tuning Decision tree algorithm time in 0.001s
Tuning Accuracy:  0.846153846154
Manual Tuning Precision:  0.333333333333
Manual Tuning Recall:  0.2
             precision    recall  f1-score   support

        0.0       0.89      0.94      0.91        34
        1.0       0.33      0.20      0.25         5

avg / total       0.82      0.85      0.83        39
```

This process took 8.8seconds from a total of 1600 fits. We had a recommendation of priors being "None" for the Naive Baye's Implementation. The resulting model performance from this recommendation of hypertuning was a precision of 33%, recall of 20%, and an accuracy of ~84%.

This is somewhat acceptable. we now turn to the Decision Tree implemenation to hopefully obtain more desirable results.

The following are the results from the Decision Tree's implementation:



```
Fitting 100 folds for each of 162 candidates, totalling 16200 fits
Pipeline(steps=[('kbest', SelectKBest(k='all', score_func=<function f_classif at 0x00000000099B1438>)), ('dtree', DecisionTreeC
lassifier(class_weight=None, criterion='gini', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=7,
            min_samples_split=7, min_weight_fraction_leaf=0.04,
            presort=False, random_state=None, splitter='best'))])
[Parallel(n_jobs=1)]: Done 16200 out of 16200 | elapsed:  1.8min finished
```

Gaussian Naive Baye's Results

This process took 1.8minutes from a total of 16200 fits. We had parameter recommendations of:

1. class_weight=None
2. criterion='gini'
3. max_depth=5,

4. max_features=None
5. max_leaf_nodes=None
6. min_impurity_split=1e-07
7. min_samples_leaf=7
8. min_samples_split=8
9. min_weight_fraction_leaf=0.04
10. splitter='best'

for the Naive Baye's Implementation.

The resulting model performance from this recommendation of hypertuning was a precision of 67%, recall of 40%, and an accuracy of ~89.74%.

Though we sacrificed time in the Decision Tree implementation, we obtained better results than that of the Gaussian Naive Baye's algorithm. Therefore, we proceeded into our analysis with the Decision Tree implementation.

Question 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?

Tuning parameters for an algorithm is a rigorous attempt at finding some optimal factors/identifiers to best create a statistical forecast model. If you we failed to reject or even disvalue the importance of tuning, we fail to find the a both precise and accurate model for future cases outside some past observations.

We obtained several model performance results from Gaussian Baive Baye's Implementation in the following:

Metrics	**Gaussian Naive Bayes Algorithm Time before Tuning(seconds)**	**Accuracy before Tuning**	**Precision Before Tuning**	**Recall before Tuning**	**F1-Score before Tuning**	**Gaussian Naive Bayes Algorithm Time after Tuning (seconds)**	**Accuracy after Tuning**	**Precision after Tuning**	**Recall after Tunin**
Default Settings	0.005	0.846	0.33	0.20	0.25	**0.005**	**0.846**	**0.33**	**0.20**
Non-POI 90%	0.005	0.846	0.33	0.20	0.25	**0.002**	**0.128**	**0.13**	**1.00**
Non-POI 60%	0.005	0.846	0.33	0.20	0.25	**0.001**	**0.795**	**0.20**	**0.20**
Non-POI 10%	0.005	0.846	0.33	0.20	0.25	**0.001**	**0.846**	**.33**	**0.20**

We observe the Gaussian Naive Bayes implementation is best with default settings, or prior probability of Non-POI at/around 10%.

The following is outputs for Decision Tree Implementation with manual tuning:

Metrics	Decision Tree Algorithm Time before Tuning(seconds)	Accuracy before Tuning	Precision Before Tuning	Recall before Tuning	F1-Score before Tuning	Decision Tree Algorithm Time after Tuning (seconds)	Accu after Tunir
Default Settings	0.003	0.821	0.33	0.40	0.36	**0.003**	**0.821**
min_weight_fraction_leaf=0.0001	0.003	0.821	0.33	0.40	0.36	**0.002**	**0.821**
min_samples_split=3	0.003	0.821	0.33	0.40	0.36	**0.001**	**0.821**
min_samples_leaf=5	0.003	0.821	0.33	0.40	0.36	**0.002**	**0.846**
max_depth = 7	0.003	0.821	0.33	0.40	0.36	**0.001**	**0.821**

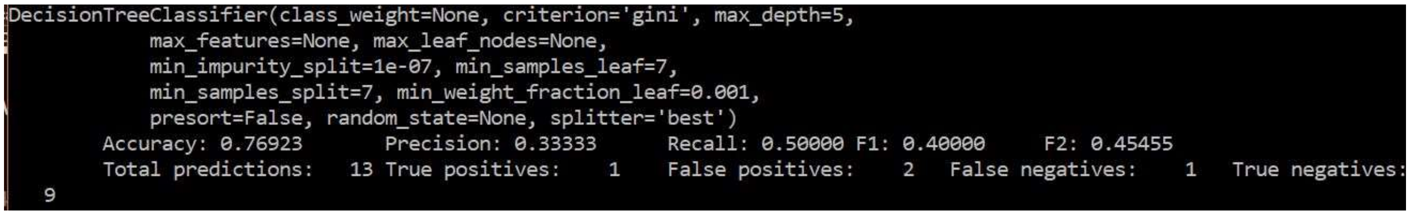
Everything highlighted in bold, the latter five columns are different parameters being tuned. The firt five columns are the default Decision Tree Algorithm model output.

Utilizing the Pipeline and GridSearchCV libraries, the following code provided

fitting 1000 folds for each of 432 candidates, totalling 432000 fits, as seen below

```
Pipeline(steps=[('kbest', SelectKBest(k='all', score_func=\<function f_classif at
0x00000000099DB518>)),

('dtree', DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,

max_features=None, max_leaf_nodes=None,

min_impurity_split=1e-07, min_samples_leaf=5,

min_samples_split=7, min_weight_fraction_leaf=0.001,

presort=False, random_state=None, splitter='best'))])
```

Again we see, utilizing the recommended parameters and feature_list features that we manually selected, we receive the following output from tester.py:



```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=7,
            min_samples_split=7, min_weight_fraction_leaf=0.001,
            presort=False, random_state=None, splitter='best')
        Accuracy: 0.76923       Precision: 0.33333      Recall: 0.50000 F1: 0.40000     F2: 0.45455
        Total predictions:   13 True positives:    1    False positives:    2   False negatives:    1   True negatives:
    9
```

We have recall and precision scores of .30+. Moreover, our accuracy is ~0.86%!

Question 5

What is validation, and what's a classic mistake you can make if you do it wrong?

Validation is referred to as the process where a trained model is evaluated with a testing data set. With this partitioning of data, validation serves a purpose of using the testing data to test a trained model for generalizations. We test our trained model's precision, accuracy, and recall rates.

One classic mistake if you can do it wrong is incorrectly guessing future cases in which affect production/company performance. This incorrect predictions are commonly due to overfitting or underfitting the model shaped by the training data.

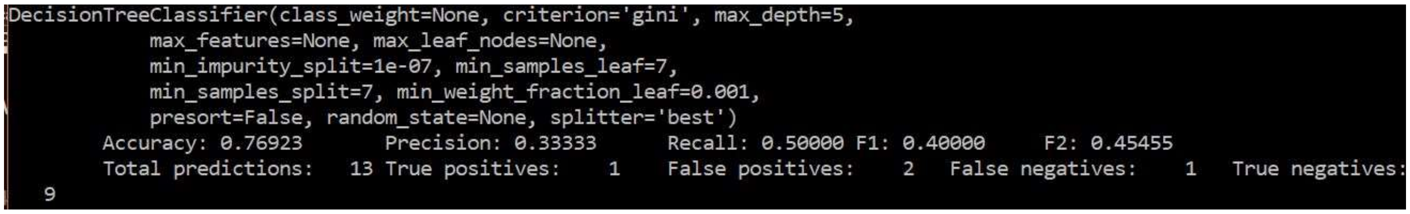
Question 6

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"]

Fitting 100 folds for each of 162 candidates, totalling 16200 fits, as seen below

> Pipeline(steps=[('kbest', SelectKBest(k='all', score_func=\<function f_classif at 0x00000000099DB518>)),
>
> ('dtree', DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
>
> max_features=None, max_leaf_nodes=None,
>
> min_impurity_split=1e-07, min_samples_leaf=5,
>
> min_samples_split=7, min_weight_fraction_leaf=0.001,
>
> presort=False, random_state=None, splitter='best'))])

Utilizing the recommended parameters and feature_list features that we manually selected, we receive the following output from tester.py:


```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=7,
            min_samples_split=7, min_weight_fraction_leaf=0.001,
            presort=False, random_state=None, splitter='best')
        Accuracy: 0.76923       Precision: 0.33333      Recall: 0.50000 F1: 0.40000     F2: 0.45455
        Total predictions:   13 True positives:    1    False positives:    2   False negatives:    1   True negatives:
   9
```

We have recall and precision scores of .30+. Moreover, our accuracy is ~0.86!

Note: When TP < FP, then accuracy will always increase when we change a classification rule to always output “negative” category. Conversely, when TN < FN, the same will happen when we change our rule to always output “positive This following section also provides the Precision, Recall, and F1-Score related to our implemented models.

In our case,

Precision (TP)/(TP+FP) cares about whether the positive examples predicted by our model were correct. In our case, what's the % Enron employees classified as POI correctly out of all classified Enron Employees classified as POI.

Recall (TP)/(TP+FN) cares more on whether we have predicted all positive examples in the data. In our case, what is the percent of predictions were correctly identified POI, for all actual POI.

where TP:=True Postive, FN:=False Negative, FP:= False Postives, TN:= True Negatives, as seen below

True State/Diagnosis	NOT POI	POI
NOT POI	TN	FP
POI	FN	TP