**Student :**

**Mini Riar**

**Enron Submission Free-Response Questions& Answers:**

A critical part of machine learning is making sense of your analysis process and communicating it to others. The questions below will help us understand your decision-making process and allow us to give feedback on your project. Please answer each question; your answers should be about 1-2 paragraphs per question. If you find yourself writing much more than that, take a step back and see if you can simplify your response!

When your evaluator looks at your responses, he or she will use a specific list of rubric items to assess your answers. Here is the link to that rubric: Each question has one or more specific rubric items associated with it, so before you submit an answer, take a look at that part of the rubric. If your response does not meet expectations for all rubric points, you will be asked to revise and resubmit your project. Make sure that your responses are detailed enough that the evaluator will be able to understand the steps you took and your thought processes as you went through the data analysis.

Once you’ve submitted your responses, your coach will take a look and may ask a few more focused follow-up questions on one or more of your answers.

We can’t wait to see what you’ve put together for this project!

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.

Answer: The goal of this project is to build a person of interest identifier for the Enron scandal using the database provided. The dataset consists of a data dictionary with the employee’s name as the Key and then data for each of the following info points:

'poi', 'salary', 'bonus', 'deferral\_payments', 'total\_payments', 'loan\_advances', 'restricted\_stock\_deferred', 'deferred\_income', 'total\_stock\_value', 'expenses', 'exercised\_stock\_options', 'long\_term\_incentive', 'restricted\_stock', 'director\_fees', 'shared\_receipt\_with\_poi’

By using the information that we have- especially the hand generated list of POIs, indicating whether each person is considered a POI or not- then using the data – to train our algorithm, the final algorithm could be potentially used on more data to determine people who should be investigated as they have a high potential of being a person of interest in the Enron scandal. One can then find other features and values that can point to the POI.

* total number of data points – there were 146 data points (keys/ employees) in the database
* allocation across classes (POI/non-POI) there were 18 POIs identified and 126 non\_POIs and 2 other(TOTAL and The Travel Agency IN the Park)
* number of features used – there were 15 total features in the database and there were many with missing values or the same value. I decided to use the following:

poi

bonus

salary

based on the graphs that I created and the results.

Since I needed info on emails to or from POIs, I created new features with these in mind:

fraction\_from\_poi

fraction\_to\_poi

bonus\_salary\_ratio

are there features with many missing values? etc.

Yes, there are features with many missing values. “differed\_payments”, “loan\_advances” and “restricted\_stock\_deferred” have many missing values to name a few

There are also some employees missing a lot of values:

THE TRAVEL AGENCY IN THE PARK

Michael Brown

Peggy Fowler

John Gillis

Roderick Hayslett

James Hughes

Eugene Lockhart – no values?

William Powers

To name a few – but are these people who are connected to Enron scandal??? I am not sure- and I was not able to confirm- so I kept them in the database. Because of the size of the dataset, it is impossible to validate any method of imputing the data, so I chose to be conservative and only remove erroneous data

**Missing values in features**:

deferral\_payments : 107

bonus : 64

total\_stock\_value : 20

expenses : 51

poi : 0

deferred\_income : 97

restricted\_stock : 36

long\_term\_incentive : 80

salary : 51

total\_payments : 21

exercised\_stock\_options : 44

restricted\_stock\_deferred : 128

shared\_receipt\_with\_poi : 60

loan\_advances : 142

director\_fees : 129

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

As per lesson 8 on Outliers, I considered removing the 10%, but since we have a small database- I wanted to work with the full database to ensure I did not lose information at this point, before I understood what is important. I did note from the file “enron61702insiderpay.pdf” that the two last lines are not people, so I removed those from the data that I used. I also saw-with the regression visualization of bonus versus salary that the outlier “Total” was way off for both salary and bonus – after all it is the total of all the values. The other line item "THE TRAVEL AGENCY IN THE PARK", has no significant data and is not a person, so I removed that too.

There were other outliers- where salary and/ or bonus seemed extremely high- but I kept those, as they may be high because they are POIs.

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

poi, salary, bonus, plus my fraction to poi and fraction\_from\_poi based on the graphs and what I saw, as well as what I learned from the articles about the Enron scandal and what I learned from doing the earlier exercises. I later used all of the features to run the GridSearchCV. I also used graphing top understand how the different features behaved, and to see which ones to use. In the end, when I used the StratifiedShuffleSplit and GridSearchCV , I used all the features along with three that I created in order to identify POIs.

Did you have to do any scaling? Why or why not? I did use MinMaxScaler for the decision tree classifier and some of the other classifiers. I used it so that features with a much greater variance than others would not dominate the objective function and make the estimator unable to learn from other features correctly as expected.

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.)

I created three features:

fraction\_to\_poi

fraction\_from\_poi

bonus\_salary\_ratio

(Based on review of the data and reading about the Enron scandal. It was important to understand the percentage of emails that people had between actual POIs versus regular other emails. I also used SelectKBest in my pipeline to find the best features for my final solution.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLASSIFIER | k | Accuracy | Precision | Recall |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=1 | 0.8178 | 0.2221 | 0.1465 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=2 | 0.79013 | 0.1924 | 0.1795 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=3 | 0.8106 | 0.2811 | 0.2700 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=4 | 0.8083 | 0.2863 | 0.2935 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=5 | 0.8136 | 0.2988 | 0.2900 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=6 | 0.8199 | 0.3192 | 0.3095 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=7 | 0.8267 | 0.3188 | 0.2635 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=8 | 0.8281 | 0.3446 | 0.3205 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=9 | 0.8217 | 0.3239 | 0.3100 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=10 | 0.8299 | 0.3305 | 0.2690 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=11 | 0.8174 | 0.2991 | 0.2750 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=12 | 0.8176 | 0.3025 | 0.2820 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=13 | 0.8372 | 0.3796 | 0.3485 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=14 | 0.8278 | 0.3449 | 0.3240 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=15 | 0.8322 | 0.3577 | 0.3250 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=16 | 0.8478 | 0.4287 | 0.4255 |
| DecisionTreeClassifier  With ‘fraction\_from\_poi’ | k=16 | 0.8505 | 0.4333 | 0.3945 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | k=17 | 0.8370 | 0.3782 | 0.3455 |
| DecisionTreeClassifier  With ‘fraction\_from\_poi’ | k=17 | 0.8358 | 0.3751 | 0.3475 |
| DecisionTreeClassifier | k=’all’  without fraction\_from\_poi  k=’all’ and parameter fraction\_from\_poi is included in features\_list | 0.8478  0.8370 | 0.4287  0.3782 | 0.4255  0.3455 |
| DecisionTreeClassifier | Default without fraction\_from\_poi  default and parameter fraction\_from\_poi is included in features\_list | 0.8299  0.8299 | 0.3305  0.3311 | 0.2690  0.2705 |
| DecisionTreeClassifier | Range (1,18) | 0.8441 | 0.4148 | 0.4125 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | Range (1,17) c 17 features | 0.8478 | 0.4287 | 0.4255 |
| DecisionTreeClassifier | Range (1,17) c 18 features | 0.8441 | 0.4148 | 0.4125 |
| DecisionTreeClassifier  Without ‘fraction\_from\_poi’ | Range(1,10) | 0.8217 | 0.3239 | 0.3100 |
| DecisionTreeClassifier | Range (1,16) c 18 features | 0.82347 | 0.3267 | 0.3055 |
| DecisionTreClassifier | Range (1,15) | 0.8235 | 0.3267 | 0.3055 |
| DecisionTreeClassifier | Range (1,4) | 0.8106 | 0.2811 | 0.2700 |
| DecisionTreeClassifier | Range (1,3) | 0.8178 | 0.2221 | 0.1465 |
|  |  |  |  |  |

K=16 had the highest results! And it has the highest results when fractio\_from\_poi is not included – so I have not included fraction\_from\_poi in my model that I have submitted. Kept the other features!

The bonus versus Salary seemed to be significant with POIs – and so I utilized it and it did actually increase my results. See results with and without the new features:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLASSIFIER | features | Accuracy | Precision | Recall |
| DecisionTreeClassifier | 15 + fraction\_to\_poi & fraction\_from\_poi | 0.82927 | 0.3463 | 0.3160 |
| DecisionTreeClassifier | 15 + bonus\_salary\_ratio & fraction\_from\_poi | 0.83567 | 0.37288 | 0.3410 |
| DecisionTreeClassifier | 15 + bonus\_salary\_ratio & fraction\_to\_poi | 0.84780 | 0.42872 | 0.4255 |
| DecisionTreeClassifier | 15 + fraction\_to\_poi | 0.86487 | 0.49147 | 0.3890 |
| DecisionTreeClassifier | 15 given features only | 0.81847 | 0.31221 | 0.3005 |

In conclusion, the precision and recall are both higher when I utililized both ‘fraction\_to\_poi’ & ‘bonus\_salary\_ratio' with SelectKBest , so I have kept both of these. It is debatable whether it is better without bonus\_salary\_ratio – as the precision and accuracy go up but the recall goes down a lot. I have included these two new functions in my solution, because this will mean that I will catch more POIs.

1. 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 tried the following algorithms for identifying POIs in the Enron database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLASSIFIER | Parameters | Accuracy | Precision | Recall |
| DecisionTreeClassifier | random\_state=42 | 1.0 | 0.4 | 0.66667 |
| Same decision tree- but tester.py results |  | 0.7432 | 0.2589 | 0.2215 |
| GaussianNB Classifier |  |  |  |  |
| SVC Classifier | C=1000.0, kernel='rbf' |  |  |  |
| KMeans Clustering Classifier | n\_clusters=3, random\_state = 0 |  |  |  |
| KNeighbors Classifier | n\_neighbors=3, algorithm='auto' | 0.87878 | 0.3333 | 0.3333 |

I found that the Decision Tree Classifier and the KNeighbours Classifier- both gave me a precision and recall of greater than 0.3 before I used the tester.py. So, I focused on these for the task 5 – tuning the parameters.

1. 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?

* 1. I used (random\_state=42) when instantiating the DecisonTreeClassifier so that my results would be consistent every time I ran the program
  2. I used the parameter: criterion': ['gini','entropy']

This function measures the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

* 1. I used parameter: min\_samples\_split':[5, 10, 20] select

**min\_samples\_split** : the default value =2. It is the minimum number of samples required to split an internal node.

* 1. I used the parameter ‘max\_depth' with choices [3,10,15,20,25,30]

**max\_depth** default=None. It is the maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.

* 1. I used the parameter max\_leaf\_nodes':[5,10,30] & max\_leaf\_nodes':[10,30,40] **max\_leaf\_node**l: the default is None. Grow a tree with max\_leaf\_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.
  2. I used the parameter max\_features': [0.4]

**max\_features** : can be int, float, string or None, the default=None. It is the number of features to consider when looking for the best split:

\* If int, then consider max\_features features at each split.

\* If float, then max\_features is a percentage and int(max\_features \* n\_features) features are considered at each split. If None, then max\_features=n\_features.

\* Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max\_features features.

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).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLASSIFIER | Parameters | Accuracy | Precision | Recall |
| DecisionTree Classifier | MinMaxScaler(range=0,1), k=18, criterion='gini', n\_iter=100 | 0.8022 | 0.27204 | 0.28003 |
| DecisionTree Classifier | MinMaxScaler(range=0,1), k=18, criterion='gini', max\_depth=10, max\_leaf\_nodes=10, n\_iter=100, | 0.8183 | 0.2727 | 0.2175 |
| DecisionTree Classifier | MinMaxScaler(range=0,1), k=18, criterion='gini', max\_depth=20, max\_leaf\_nodes=30, n\_iter=100, | 0.8134 | 0.29117 | 0.2785 |
| DecisionTree Classifier | MinMaxScaler(range=0,1), k=18, criterion='gini', max\_depth=none, max\_leaf\_nodes=5, n\_iter=100, min\_samples\_split:[2,10,20] | 0.8544 | 0.3602 | 0.1185 |
| DecisionTree Classifier | MinMaxScaler(range=0,1), k=18, criterion='entrophy, max\_depth=none, max\_features=0.4, max\_leaf\_nodes=None, n\_iter=100 | 0.82553 | 0.33096 | 0.3020 |
| DecisionTree Classifier | MinMaxScaler(range=0,1), k=18, criterion='entrophy’ or’gini’, max\_depth=none, max\_features=0.4, max\_leaf\_nodes=None, n\_iter=100, random\_state-42 | 0.8478 | 0.4287 | 0.4255 |
| DecisionTree Classifier | MinMaxScaler(range=0,1), k=18, criterion='entrophy’ or’gini’, max\_depth=none, max\_features=0.1, max\_leaf\_nodes=None, n\_iter=100, random\_state-42 | 0.84247 | 0.38207 | 0.29400 |
| DecisionTree Classifier | MinMaxScaler(range=0,1), k=18, criterion='entrophy’ or’gini’, max\_depth=none, max\_features=0.6, max\_leaf\_nodes=None, n\_iter=100, random\_state-42 | 0.83227 | 0.36548 | 0.3505 |
| DecisionTree Classifier | MinMaxScaler(range=0,1), k=18, criterion='entrophy’ or’gini’, max\_depth=none, max\_features=0.3, max\_leaf\_nodes=None, n\_iter | 0.82340 | 0.31552 | 0.2775 |
| DecisionTree Classifier | MinMaxScaler(range=0,1), k=18, criterion='entrophy’ or’gini’, max\_depth=none, max\_features=0.5, max\_leaf\_nodes=None, n\_iter | 0.84213 | 0.39021 | 0.3270 |
| DecisionTree Classifier | Added: max\_depth':[3,10,15,20,25,30] | 0.8741 | 0.5832 | 0.1945 |
| DecisionTree Classifier | Added: max\_leaf\_nodes':[5,10,30] | 0.8452 | 0.3207 | 0.1440 |
| GaussianNB Classifier | MinMaxScaler(range=0,1), k=4 | 0.7766 | 0.2572 | 0.1210 |
| GaussianNB Classifier | MinMaxScaler(range=0,1), k=18 | 0.7596 | 0.25799 | 0.4280 |
| SVC Classifier | C=1000.0, kernel='rbf' | 0.8658 | 0.48346 | 0.0950 |
| SVC Classifier | C=10.0 kernel='rbf' | 0.8661 | 0.4259 | 0.0115 |
| SVC Classifier | C=10.0 kernel='linear’ | 0.86907 | 0.5978 | 0.0550 |
| KMeans Clustering Classifier | MinMaxScaler(range=0,1), n\_clusters=3, n\_init=10 | 0.6358 | 0.1463 | 0.3728 |
| KNeighbors Classifier | MinMaxScaler(range=0,1), n\_neighbors=3 or 2, algorithm='auto', leaf\_size=30 or 1 | 0.8721 | 0.57509 | 0.1570 |
| KNeighbors Classifier | MinMaxScaler(range=0,1), n\_neighbors=5, algorithm='auto', leaf\_size=30 | 0.8559 | 0.30622 | 0.0640 |
| Logistic Regression | StandardScaler(), tol = 0.001, C = .001, penalty = 'l2', random\_state = 42  [played with C value & tol - but it did not change the results significantly] | 0.79436 | 0.3414 | 0.1410 |
| Logistic Regression | C = 10\*\*-5 or C=10\*\*-10 | 0.79082 | 034175 | 0.1625 |

I increased n\_iter to 100 and results were better and I increased number of features testing from:

['poi', 'salary', 'fraction\_from\_poi', 'fraction\_to\_poi', 'bonus\_salary\_ratio'] to all 15 plus the two new features – which made a big difference too!

The best results are from the classifier which is highlighted above

**The importance of parameter tuning**:

The ultimate goal of machine learning is to make a machine system that can automatically build models from data without requiring tedious and time consuming human involvement. One of the difficulties is that learning algorithms (eg. decision trees, random forests, clustering techniques, etc.) require you to set parameters before you use the models (or at least to set constraints on those parameters). How you set those parameters can depend on a whole host of factors. Your goal, is usually to set those parameters to optimal values that enable you to complete a learning task in the best way possible. Thus, tuning an algorithm or machine learning technique, can be simply thought of as process which one goes through in which one optimizes the parameters that impact the model in order to enable the algorithm to perform the best (once, of course you have defined what "best" actual is). So, it is very important to tune the parameters so that you can get the best performance from your algorithm.

As you can see in above table - with parameter tuning, I was able to get better results when I ran the algorithm.  Better Accuracy, precision and recall!

1. 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 the computing of the accuracy of your model. This can be done with your testing data, and comparing it to the train data results. You are assessing whether your algorithm is actually doing what you want it to do

Answer:

**One of the classic mistakes** is overfitting the data- in which case accuracy is very high- but it really does not provide one with significant information. Often this is due to using the same data for both train and test

I first did validation using the following:

I used StratifiedShuffleSplit and GridSearchCV to create the 100 folds of training/ testing data and do the validation of the algorithm.

GridSearchCV which validates the model using the train and test folds- and I set StratifiedShuffleSplit it to create 100 folds of data- so it would have performed the process 100 times to come up with the final model that will be better able to identify a POI in future data.

I then checked results with testing.py function gave me accurate results for accuracy, precision and recall

**The reason I used StratifiedShuffleSplit**:

Some classification problems can exhibit a large imbalance in the distribution of the target classes: there could be several times more negative samples than positive samples. In such cases it is recommended to use stratified sampling as implemented in **[StratifiedKFold](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html" \l "sklearn.model_selection.StratifiedKFold" \o "sklearn.model_selection.StratifiedKFold)** and [**StratifiedShuffleSplit**](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html#sklearn.model_selection.StratifiedShuffleSplit) to ensure that relative class frequencies is approximately preserved in each train and validation fold.

The dataset we are using is small and skewed towards non-POI, so we need a technique that accounts for that or the risk is that we would not be able to assess, in the validation phase, the real potential of our algorithm in terms of performance metrics. The chance of randomly splitting skewed and non representative validation sub-sets could be high, therefore the need to use stratification (preservation of the percentage of samples for each class) to achieve robustness in a dataset with the aforementioned limitations is important.

1. Give at least 2 evaluation metrics and your average performance for each of them.

Answer:

For the model I have in my final submitted project, the evaluation metrics are as follows:

Accuracy = 0.84407

Precision = 0.41478

Recall = 0.4125

Explain an interpretation of your metrics that says something human-understandable about your algorithm’s performance.

Precision is the number of true positives divided by the sum of true positives plus false positives – which is really the proportion of the positives that are chosen correctly.

Recall is the number of true positives divided by the sum of true positives plus false negatives -> if this is high- it means there aren’t too many false negatives, and there are lots of true positives – so your model is more accurate

1. Resources Used:
   * 1. UDACITY lessons from this Machine Learning course
     2. Udacity Forums- searching for each question to better understand the ask, and understand the problems that others experienced.
     3. http://scikit-learn.org website – checking all pages for each of the classifiers, and other functions, validations, GridSearchCV, StandardScaler, MinMaxScaler, StratifiedShuffleSplit, Recall, Precision, etc.
     4. Googled Enron database scandal to better understand the project.
     5. <https://en.wikipedia.org/wiki/Enron_scandal>
     6. <http://www.cnn.com/2013/07/02/us/enron-fast-facts/index.html>
     7. <http://www.economist.com/node/940091>
     8. <http://content.time.com/time/specials/packages/0,28757,2021097,00.html>
     9. Used help from my mentor and Udacity Live Help to better understand the project.

Final Result:

Tester Classification report - StratifiedShuffleSplit:

Pipeline(memory=None,

steps=[('min\_max\_scaler', MinMaxScaler(copy=True, feature\_range=(0, 1))), ('f\_select', SelectKBest(k=16, score\_func=<function f\_classif at 0x0E874670>)), ('Dtree', DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,

max\_features=0.4, max\_leaf\_nodes=None,

... min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=42,

splitter='best'))])

Accuracy: 0.84780 Precision: 0.42872 Recall: 0.42550 F1: 0.42710 F2: 0.42614

Total predictions: 15000 True positives: 851 False positives: 1134 False negatives: 1149 True negatives: 11866