**Enron Submission Free-Response Questions**

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: [Link to the rubric](https://www.google.com/url?q=https://docs.google.com/a/knowlabs.com/document/d/17-JwNQH1aRxtqMkJ6zpCL_68kh5F6uSbDXcJS26vZWY/pub&sa=D&usg=AFQjCNGXFmH5PbWY9NdA_HaW3ioo6c3tng) 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. Were there any outliers in the data when you got it, and how did you handle those?  [relevant rubric items: “data exploration”, “outlier investigation”]

In this project I’ll play the role of a detective and use Machine Learning skills learned in the class to select/create best fit features and build reliable and accurate algorithm to identify Enron POIs who may have committed fraud based on the public Enron financial and email dataset.

There are total 145 records (145 persons in question) in given dataset, of which 18 are POIs and the rest 127 are non\_POIs. Obviously this is an unbalanced dataset where the number of non\_POI is much more than the one of POIs.

The given Enron dataset includes total 21 features which fall into three major categories, specifically financial features including salary, bonus, stock, etc, email features comprising all emails to/from this person and also the ones to/from any POI, and the POI label which is Boolean represented as integer (0 means NonPOI and 1 means POI). All features except ‘poi’ have a number of missing values (“NaN’) ranging from 20 to 142; among them, these with highest missing values are (feature, # of missing value):

[('long\_term\_incentive', 80), ('deferred\_income', 97), ('deferral\_payments', 107), ('restricted\_stock\_deferred', 128), ('director\_fees', 129), ('loan\_advances', 142)]

Notice that all ‘NaN’ values will be replaced by 0 before fed into cross validation.

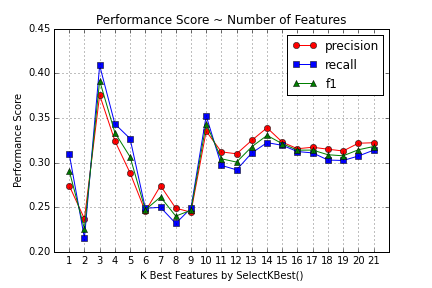
When I plot salary-bonus chart it easily tells me that there is an outliner which is way beyond the normal extend. I take a further look at the excel spreadsheet and figure out that outliner is named ‘TOTAL’; it’s actually an excel spreadsheet squirt so I remove it from the dataset [see “task1: Remove outliers” in poi\_id.py].

1. 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.  [relevant rubric items: “create new features”, “properly scale features”, “intelligently select feature”]

I create two new features, namely ratio\_to\_poi which is the ratio of the emails sent from this person to any poi against the total emails sent from this person and ratio\_from\_poi which is the ratio of the emails from any poi to this person against the total emails received by this person. I believe the person who has a high ratio of email communications to/from a POI could also be a POI. It’s approved that “ratio\_to\_poi” is ranked No.5 of best fitted features selected by SelectKBest function and “ration\_from\_poi” is No.15. The following table shows the performance improvement with the new features added into a common algorithm (PCA(n\_components=2) + DecesionTree(random\_state=10)):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Features\_list | Accuracy | Precision | Recall | F1 |
| 'bonus', 'total\_stock\_value', 'salary', 'exercised\_stock\_options' | 0.82023 | 0.41407 | 0.40600 | 0.41000 |
| + 'ratio\_to\_poi' | 0.83407 | 0.41848 | 0.4145 | 0.41648 |
| + 'ratio\_from\_poi' | 0.83407 | 0.41848 | 0.4145 | 0.41648 |

I use SelectKBest() to select the best ranked features. The number of best features ranges from 1 up to 21, and the following chart shows the fluctuation of performance scores (precision, recall and f1) vs. K (number of selected best features). Note that the algorithm used along this process is DecisionTree Classifiler with random\_state = 10. See detail in data\_discovery.py



Obviously, K=3 results in the best scores and this will be used in the final algorithm. In detail, the features used in the final algorithm is ['bonus', 'total\_stock\_value', 'exercised\_stock\_options']

The feature scores created by SelectKBest() is showed in below table. Also please run poi\_id.py or data\_discovery.py and refer to the printed list.

|  |  |  |
| --- | --- | --- |
| Item | Features | Score by SelectKBest() |
| 1 | 'exercised\_stock\_options' | 25.097541528735491 |
| 2 | 'total\_stock\_value' | 24.467654047526398 |
| 3 | 'bonus' | 21.060001707536571 |
| 4 | 'salary' | 18.575703268041785 |
| 5 | 'ratio\_to\_poi' | 16.641707070468989 |
| 6 | 'deferred\_income' | 11.595547659730601 |
| 7 | 'long\_term\_incentive' | 10.072454529369441 |
| 8 | 'restricted\_stock' | 9.3467007910514877 |
| 9 | 'total\_payments' | 8.8667215371077717 |
| 10 | 'shared\_receipt\_with\_poi' | 8.7464855321290802 |
| 11 | 'loan\_advances' | 7.2427303965360181 |
| 12 | 'expenses' | 6.2342011405067401 |
| 13 | 'from\_poi\_to\_this\_person | 5.3449415231473374 |
| 14 | 'other’ | 4.204970858301416 |
| 15 | 'ratio\_from\_poi | 3.2107619169667441 |
| 16 | 'from\_this\_person\_to\_poi' | 2.4265081272428781 |
| 17 | 'director\_fees | 2.1076559432760908 |
| 18 | 'to\_messages' | 1.6988243485808501 |
| 19 | 'deferral\_payments' | 0.2170589303395084 |
| 20 | 'from\_messages' | 0.16416449823428736 |
| 21 | 'restricted\_stock\_deferred' | 0.06498431172371151 |

Regarding features scaling, it depends on which algorithm is used. For example, my final algorithm is decision tree and feature scaling is not needed in that case because the scale of the features won’t affect the decision cut. On the other hand, feature scaling is mandatory for SVM and KNN algorithm which could be dominated by large-scale features if feature scaling is not preprocessed.

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 have tried 5 algorithms as well as some pipelines of combining PCA with algorithms, and my final algorithm is the pipeline of PCA and DecisionTree with tuned up parameters based on the recall/precision/F1 scores. Notice that the given dataset is very unbalanced the accuracy score is not a fitted criterion for this project. In addition, as a POI detective tool it’s critical to apply strict algorithm to make sure no potential POI is missing; from this perspective recall performance should be the No.1 criterion when selecting the best algorithm.

[Note]: normally feature should be preprocessed by data normalization before fed into PCA, but it’s not for this case. I have included StandardScaler+PCA+Tree in poi\_id.py but the best algorithm is achieved without this feature standardization.

Here is the best and final algorithm with tuned parameters that I use as well as the associated performance scores:



As showed, the Recall/Precision/F1 are all above the required 0.3.

This table shows other algorithms that I tried and their performance scores. Note that the best parameters are also showed inside each algorithm. Please refer to the poi\_id.py for details.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Accuracy | Precision | Recall | F1 |
| clf\_NB = GaussianNB() | 0.84300 | 0.48581 | 0.35100 | 0.40755 |
| scaler = MinMaxScaler()  svc = svm.SVC(kernel = 'rbf',  gamma = 0,  tol = 0.01,  class\_weight = {False: 1, True: 8})  clf\_pipe\_SVM = Pipeline([('scaler', scaler),  ('svc', svc)]) | 0.80169 | 0.35179 | 0.34300 | 0.34734 |
| scaler\_knn = MinMaxScaler()  knn = neighbors.KNeighborsClassifier(3, weights = 'uniform')  clf\_pipe\_knn = Pipeline([('scaler\_knn', scaler\_knn),  ('knn', knn)]) | 0.85954 | 0.58286 | 0.30600 | 0.40131 |
| pca\_RF = PCA(n\_components = 2)  random\_forest = RandomForestClassifier(n\_estimators = 17,  criterion = 'gini',  min\_samples\_split = 2,  random\_state = 10)  clf\_pipe\_RandomForest = Pipeline([('pca', pca\_RF),  ('rf', random\_forest)]) | 0.84531 | 0.49534 | 0.29200 | 0.36741 |
| pca\_Adaboost = PCA(n\_components = 2)  adaboost = AdaBoostClassifier(n\_estimators = 3,  algorithm = 'SAMME.R',  random\_state = 10)  clf\_pipe\_Adaboost = Pipeline([('pca', pca\_Adaboost),  ('adaboost', adaboost)]) | 0.84531 | 0.49106 | 0.15100 | 0.23098 |

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? (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).  [relevant rubric item: “tune the algorithm”]

The algorithm is only as good as the best tuned parameters. If parameters are not tuned or not the best fitted we could miss the best algorithm resulting in sub-optimal estimator. Before tuning I think it’s critical to determine the criterion that parameters tuning is based on. As previous stated, I use “recall” as the No.1 criterion for this particular project and other performances such as precision and F1 are also factored in. For my final algorithm Decision Tree Classifier, the main parameters being tuned up include “criterion” and “min\_samples\_split”. “Criterion” has two options - ‘gini’ which is the default choice and ‘entropy’ which is detailed in the lecture and uses information gain for decision cut. In general, the smaller the value of “min\_samples\_split” is, the more accurate the prediction is but more time-consuming. The number of principal components “n\_components” for PCA is also a main test during tuning process.

I take my final algorithm (decision tree) as an example to show how I tune up parameters and approach my final parameters set. Note that the similar methodology is applied to all algorithms for parameters tuning. Below table shows all the parameters and values that I use during the tuning process and their associated performance scores.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Accuracy | Precision | Recall | F1 |
| clf\_tree = tree.DecisionTreeClassifier(criterion = 'entropy',  min\_samples\_split = 2,  random\_state = 10) | 0.80454 | 0.37517 | 0.40650 | 0.39021 |
| pca\_tree = PCA(n\_components = 1)  tree\_1 = tree.DecisionTreeClassifier(criterion = 'gini',  min\_samples\_split = 2,  random\_state = 10)  clf\_pipe\_tree = Pipeline([('pca', pca\_tree),  ('tree', tree\_1)]) | 0.79054 | 0.30658 | 0.28650 | 0.29620 |
| pca\_tree = PCA(n\_components = 3)  tree\_1 = tree.DecisionTreeClassifier(criterion = 'entropy',  min\_samples\_split = 2,  random\_state = 10)  clf\_pipe\_tree = Pipeline([('pca', pca\_tree),  ('tree', tree\_1)]) | 0.81900 | 0.40045 | 0.35500 | 0.37636 |
| pca\_tree = PCA(n\_components = 3)  tree\_1 = tree.DecisionTreeClassifier(criterion = 'entropy',  min\_samples\_split = 4,  random\_state = 10)  clf\_pipe\_tree = Pipeline([('pca', pca\_tree),  ('tree', tree\_1)]) | 0.82292 | 0.41450 | 0.3660 | 0.38874 |
| pca\_tree = PCA(n\_components = 3)  tree\_1 = tree.DecisionTreeClassifier(criterion = 'gini',  min\_samples\_split = 2,  random\_state = 10)  clf\_pipe\_tree = Pipeline([('pca', pca\_tree),  ('tree', tree\_1)]) | 0.82654 | 0.43021 | 0.39300 | 0.41077 |
| pca\_tree = PCA(n\_components = 2)  tree\_1 = tree.DecisionTreeClassifier(criterion = 'gini',  min\_samples\_split = 2,  random\_state = 10)  clf\_pipe\_tree = Pipeline([('pca', pca\_tree),  ('tree', tree\_1)]) | 0.82469 | 0.42930 | 0.42350 | 0.42638 |
| pca\_gs = PCA()  tree\_3 = tree.DecisionTreeClassifier(random\_state = 10)  pipe\_tree\_gs = Pipeline([('pca\_gs', pca\_gs),  ('tree', tree\_3)])  parameters = {'pca\_gs\_\_n\_components':[1, 2, 3],  'tree\_\_criterion': ['gini', 'entropy'],  'tree\_\_min\_samples\_split': [2, 3, 4]}  clf\_pipe\_tree\_gs = GridSearchCV(pipe\_tree\_gs,  parameters,  scoring = 'recall') | 0.81592 | 0.39384 | 0.36450 | 0.37860 |

The best performance score with the best parameters is highlighted in orange. Note that all parameters are manually tuned except the last one that’s tuned by GridSearchCV with “scoring” equal to ‘recall’. Since the data is split by stratified shuffle split with 1000 folding, using GridSearchCV in each folding is computationally complicated, time consuming and doesn’t warrant best performance score in this case.

1. 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 the methodology to split data into train and test sets so that the prediction function can be learned/trained in the train set and then the performance of the trained predictor can be evaluated in the test set. The classic mistake is called overfitting where the prediction function is trained and tested on the same dataset resulting in perfect training score but poor performance on new dataset.

Due to the fact that the given Enron dataset is small size (only 145 labels) and very unbalanced (18 POIs vs. 127 non\_POIs), I use stratified shuffle split cross validation (StratifiedShuffleSplit()) for this project. Stratified split can make each split set containing approximately the same percentage of samples of each label class as the original complete dataset, this is critical to conquer unbalance. Since the given data is independent and ordering is not arbitrary shuffling it first may be essential to get a meaningful cross validation result.

1. 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”]

Below is the final algorithm and the associated performance metrics.



Accuracy: 0.82469.

The probability that this predictor can correctly identify whether or not a given person is a POI is 82.469%.

Precision: 0.42930

When the predictor predict this person is a POI, it’s 42.93% possibility that she/he is really a POI.

Recall: 0.42350

When a real POI is given to the predictor, the probability that this predictor can successfully predict she/he is a POI is 42.35%