**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&ust=1492358895117000&usg=AFQjCNGqSWtGntXAUt7zLmQy0ZFO5zL8xg) 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”]

**Dataset:** Enron was a large company in 2000 but collapsed into bankruptcy due to widespread corporate fraud. Federal investigation published Enron dataset, the biggest Corpus of emails contains hundreds of thousands of emails and detailed financial data for top executives of Enron.  Udacity has combined this data with a hand-generated list of persons of interest(POI) in the fraud case. Thus, we have labelled data for this project and we are applying machine learning techniques/algorithms to classify POI based on financial and email data.

Brief Description of the data:

* Number of data points: 146 (Number of person: 145)
* Number of features for each person: 21
* Number of POIs in the dataset: 18
* Number of POIs as per Federal Investigation: 35
* A feature doesn’t have a well-defined value is denoted as ‘NaN’
* Number of people in the dataset have ‘NaN’ for their total payments: 18
* Number of POIs in the dataset have ‘NaN’ for their total payments: 0

**How machine learning can be used on the Enron dataset?**

We can use machine learning algorithm to find out patterns like who are all the players in the fraud, how much money were they making, who is sending email to whom.

Some example of using ML on the dataset are:

* Using Regression, we can understand the relationship between different features in the dataset like Salaries Vs Bonuses of Enron employees.
* We can use unsupervised learning like clustering to figure out Regular Employees Vs Board of Directors
* We will use outlier detection and removal to find bugs in the data set that we need to clean up before performing analysis.

**Outlier:** In supervised learning we ignore outliers; but in fraud detection (like detecting POIs) we pay attention to outliers. In the outlier nano-project we found that biggest Enron outlier has a dictionary key ‘TOTAL’. It is a Excel quirk so we have remove it from dataset using ‘*dictionary.pop( key, 0 )’* . There are other outliers, two people ‘LAY KENNETH L’ & ‘SKILLING JEFFREY K’ made bonuses of at least 5 million dollars, and a salary of over 1 million dollars. These 2 data points are valid.

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”]

**Feature Addition:** We know human intuition is that POIs might have particularly strong email connections between each other. In other words, they send each other emails at a higher rate than the people in the population send emails to POI. We already have features 'from\_this\_person\_to\_poi’ and 'from\_poi\_to\_this\_person' features in dataset. From visualization in the Feature Selection mini-project we have seen that the number of emails from a POI to a given person ('from\_poi\_to\_this\_person') is not useful. It is same for number of emails a person sends to a POI ('from\_this\_person\_to\_poi’). As per the human intuitions we want to scale these features by total number of messages to or from this person.

We have added 2 new features:

‘fraction\_from\_poi\_to\_this\_person’

‘fraction\_from\_this\_person\_to\_poi '

We have dropped 4 features:

‘from\_poi\_to\_this\_person’, ‘from\_this\_person\_to\_poi', ‘to\_messages’, ‘from\_messages’

To measure the impact of the new feature; I calculated features' scores for the original feature set and then compare them with the scores obtained by the dataset with the new features.

**I used DecisionTreeClassifier() to train dataset & measure feature importance using it’s attribute feature\_importances\_:**

clf = DecisionTreeClassifier()

clf = clf.fit(features, labels)

print "feature\_list:",features\_list

print "feature\_importances\_:",clf.feature\_importances\_

The importance of the old features are following:

'to\_messages' : 0.0

'from\_poi\_to\_this\_person' : 0.0555

'from\_messages' : 0.0481

'from\_this\_person\_to\_poi' : 0.0073

The importance after new features:

'fraction\_from\_poi\_to\_this\_person': 0.0

'fraction\_from\_this\_person\_to\_poi': 0.1360

Clearly the new feature 'fraction\_from\_this\_person\_to\_poi' has more importance than the older features. So I dropped 4 older features & added the new feature 'fraction\_from\_this\_person\_to\_poi' for supervised learning. Since importance of another new feature 'fraction\_from\_poi\_to\_this\_person' is very low; I haven’t included the feature for supervised classification.

**Feature Scaling:** Feature scaling is a type of feature preprocessing that you should perform before some classification and regression tasks. Since we are using 'from\_this\_person\_to\_poi’ and “total\_payments” as features **for classification; scaling is critical**. Scaling is most important when we calculate some distance matrix (K-Means, SVM with RBF Kernel), so that all features have equal weightage. Classifier like Linear Regression or Decision Tree doesn’t need feature scaling but there is no harm if we use them. I have used SKLearn MinMaxScaler that convert each features in the range.

**Feature Selection:** We know if we use very few feature then it is an oversimplified classification with large error. If we include lot of features then classifier will over fit and quality of the classifier will go down. We need to find the number of features for which the quality of the model will be maximized.

In this project I used PCA and don’t want to do feature selection before we go in the PCA. PCA is going to find a way to combine information from potentially many different input features together. So if we throwing out input features before PCA, we are throwing out the information that PCA might be able to use for finding salient features. I am trying optimize the n\_components of PCA()in GridSearchCV().

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:

1. NAiVE BAYES Algorithm finds a linear decision surface. When the class are very overlapping, you have to count independent evidence. That's where then a Naive Bayes classifier would be better.
2. DECISION TREE classifies a dataset by asking multiple linear questions one after another. Machine learning task is to learn the decision tree from training data.

Decision Tree parameters:

criterion: “gini"

max\_depth:[None,3,5,7,9]

min\_samples\_split: [2, 3, 4]

1. SVM finds a separating line, more generally called a hyper plane, between data of two classes. The hyper plane maximise the distance (margin) to nearest point, and it does this relative to both classes. As it maximise margin, it seems to be most robust to classification errors.

SVM Parameter

* Kernel: [“rbf”]
* C: [10000, 5000, 1000, 100, 10]
* gamma: [0.0005, 0.005, 0.05, 0.1, 0.5]

Accuracy, Precision, Recall & F1 score obtained by

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithm** | **GridSearchCV**  **Params** | **Best params on training set** | **Accuracy on testing set** | **Precision on testing set** | **Recall on testing set** | **F1 score on testing set** |
| GaussianNB | {'reduce\_dim\_\_k': [5, 7, 9, 11, 13]} | {'reduce\_dim\_\_k': 5} | 0.82993 | 0.34409 | 0.30400 | 0.32280 |
| GaussianNB | {'reduce\_dim\_\_n\_components': [5, 7, 9, 11, 13]} | {'reduce\_dim\_\_n\_components': 9} | 0.83360 | 0.36681 | 0.34150 | 0.35370 |
| Decision Tree | {'reduce\_dim\_\_n\_components': [5, 7, 9],'clf\_\_max\_depth': [None, 3, 5, 7, 9], 'clf\_\_min\_samples\_split': [2, 3, 4]} | {'reduce\_dim\_\_n\_components': 5, 'clf\_\_max\_depth': None, 'clf\_\_min\_samples\_split': 2} | 0.81053 | 0.26080 | 0.22950 | 0.24415 |
| SVC | {'reduce\_dim\_\_n\_components': [5, 7, 9, 11, 13], 'clf\_\_gamma': [0.0005, 0.005, 0.05, 0.1, 0.5], 'clf\_\_C': [10000, 5000, 1000, 100, 10], 'clf\_\_kernel': ['rbf']} | {'reduce\_dim\_\_n\_components': 9, 'clf\_\_gamma': 0.5, 'clf\_\_C': 10000, 'clf\_\_kernel': 'rbf'} | 0.80853 | 0.24533 | 0.21000 | 0.22629 |
| AdaBoost | {'reduce\_dim\_\_n\_components': [5, 7, 9, 11, 13], 'clf\_\_learning\_rate': [1.0], 'clf\_\_n\_estimators': [50]} | {'reduce\_dim\_\_n\_components': 11, 'clf\_\_learning\_rate': 1.0, 'clf\_\_n\_estimators': 50} | 0.81847 | 0.24380 | 0.17200 | 0.20170 |

In this supervised classification problem, we want to find class labels (POI or non-POI) by finding decision boundary. We will not use regression which is mainly used for predicting continuous value.

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”]

Machine Learning algorithm parameters also help us to tune bias & variance. A high bias machine learning algorithm is one that practically ignores the data. It has almost no capacity to learn anything, and it is called bias. Another extreme is high variance algorithm that is extremely perceptive to data and it can only replicate things it’s seen before. The problem with high variance algorithm is that it’ll react very poorly in situations it hasn’t seen before because it doesn’t have right bias to generalize new stuff. So we want something in the middle, it is called bias-variance trade-off. You want machine learning algorithm that has some authority to generalize, but still very open to listen to the data. We achieve bias-variance trade-off by parameter tuning. Parameter tuning can have some influence on how well the model fits the complexity in the data.

**Following is how I tuned C & gamma parameters of SVC:**

**Parameter Tuning for SVC:** I was using SVC without any parameter so the default parameters of C (1.0) & gamma was used (1/n\_features). In this case accuracy, precision & recall was very poor. I tuned C & gamma to get better result.

**C** : float, optional (default=1.0) Penalty parameter C of the error term.

**C controls the cost of misclassification on the training data. C controls trade-off between smooth decision boundary and classifying training point correctly. Large value of C we will get more training points correct thus model will overfit and decision boundary will not be smooth.**

**gamma** : float, optional (default=’auto’) Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’. If gamma is ‘auto’ then 1/n\_features will be used instead.

Gamma defines how far the influence of a single training point reaches.

* Low value – every point has a far reach
* High value – each training point has a close reach, we can end up with wiggly decision boundary that indicates overfitting

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”]

In machine learning Validation Technique is used to assess whether the algorithm is doing what we wanted to do by testing it and validating it.

In machine learning we split a data set into training and testing data to assess our Machine Learning algorithm. We generally specify what fraction of data is testing and what is training. We want to maximize both sets. We want to have as much data points in the training sets to get the best learning results, and want to maximize number of data items in the test set to get the best validation. But obviously there’s an inherent trade-off here, which is every data point you take out of the training set into the test is lost for the training set. Cross-validation partition the data set into k bins of equal size. In k-fold cross validation, you run k separate learning experiments. In each of those, CrossValidator pick one of those k subsets as testing set. The remaining k minus one bins are put together into the training set, then you train your machine learning algorithm and access the performance on the testing set. We average the test results from these k experiments, thus the assessment of the learning algorithm is more accurate. And in a way, all data is used for both training & testing.

In the final project I have used Stratified ShuffleSplit cross-validator. It provides train/test indices to split data in train/test sets. It uses random permutation to split data (like ShuffleSplit cross validator) and also preserve the percentage of samples for each class (like StratifiedKFold cross validator). The random permutation eliminate the training error occurs if the data is ordered by the classes. The train\_test\_split splits the dataset into a single set not in K sets. I have used it to get a random test set for validating performance of the model.

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”]

I used accuracy, precision & recall for evaluating average performance of the above models.

**Accuracy** is measured by

**Precision** measures out of all items classified as positive by classifier, how many truly belongs to positive class.

**Recall** measures how many positive items are recalled from dataset ie. out of all items that are truly positive, how many were correctly classified as positive.

**In the context of Enron dataset and Classifier, we can define Recall & Precision as follows:**

**Accuracy:** Number of persons classified correctly as POI or non-POI divided by the total number of persons in the dataset.

**True Positive:** Number of data points with actual class as POI and classifier also classified them as POI.

**False Positive:** Number of non-POI classified as POI.

**False Negative:** Number of POI classified as non-POI.

**Recall:** The probability that POI will be classified correctly by our learning algorithm.

**Precision:** Suppose a person is classified as POI by learning algorithm, then what is the probability that the person is really a POI

Precision & Recall is very important evaluation criteria. We can tune our algorithm for exactly what we want it to be optimized for. In the project we have tuned our algorithm for both precision & recall score more than 0.3.

References:

* 1. <http://scikit-learn.org/stable/modules/tree.html>
  2. <http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>
  3. http://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.StratifiedKFold.html
  4. <https://www.quora.com/What-are-C-and-gamma-with-regards-to-a-support-vector-machine>
  5. <https://discussions.udacity.com/t/how-to-use-pipeline-for-feature-scalling/164178/2>