**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 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 coach evaluates 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://docs.google.com/a/knowlabs.com/document/d/17-JwNQH1aRxtqMkJ6zpCL_68kh5F6uSbDXcJS26vZWY/pub)

Each question has one or more specific rubric items associated with it, so before you submit an answer, take a look at that the rubric.  If your response does not meet expectations, you will be asked to resubmit.

Once you’ve submitted your responses, your coach will take a look and 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”]

The goal of the project is to identify person’s of interest (POI) among Enron employees who may have committed fraud or broken the law. To do this, I will build a POI identifier that uses email and financial data released as part of the court case. Machine learning is useful for building a model that can determine whether someone is a POI. I will use Machine Learning techniques to build a classifier, to select the most important features to use for classification, and to test the accuracy of my classifier. Initial data discovery uncovered the following:

* total no. of data points = 146 people in the data set
* allocation across classes (POI/non-POI) = 18 POI, 128 non-POI
* number of features = 21 features are available
* are there features with many missing values? Yes.

There were many features with ‘NaN’ values. Features with large and moderate number of missing values: loan\_advances, restricted\_stock\_deferred, director\_fees, deferral\_payments, long\_term\_incentive, deferred\_income. I decided to use SelectKBest to select the highest scoring features rather than decide which features to remove using trial and error.

For outlier investigation: I created scatter plots of certain features (salary, bonus, total payments). A discovered an outlier - FREVERT MARK A – who was often grouped with clusters of POI but is not a POI. I did not remove him because I want to avoid over cleansing the data and remove useful information that might cause future new incoming data to be wrongly classified.



Through the exercise of creating the scatterplots, I discovered a major outlier – TOTAL – which I removed as it is a meaningful outlier (it’s a “feature” that is the sum of all training examples).

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 doesn’t 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.)  If you used an algorithm like a decision tree, please also give the feature importance of the features that you use.  [relevant rubric items: “create new features”, “properly scale features”, “intelligently select feature”]

**Final feature set:** 'from\_this\_person\_to\_poi', 'expenses', 'deferred\_income'

**Feature importances:**
array([ 0.15168195, 0.48905665, 0.3592614 ])

**My selection process:** I used the SelectKBest algorithm to determine which features were most important using an objective and systematic process. I plotted the k-values against precision and recall for training and test sets to avoid overfitting. I ran SelectKBest against Gaussian Naïve Bayes, Decision Tree, C-Support Vector Classification, and KNearestNeighbor classifiers one-by-one. I ended up going with Decision Tree because it was the only algorithm that showed precision and recall above 0.3. SelectKBest showed that in a Decision Tree classifier, a k-value = 3 was the point that had the highest precision and recall without overfitting. Hence, I chose the 3 highest scoring features, according to SelectKBest, as the feature list to use in my classifier model (see feature list above).



Next, I engineered a new feature: the new feature is a measure of the ratio of “suspicious” emails over total emails sent and received by a person. Here are the steps I took: I processed all emails in the emails\_by\_address directory and created a text vectorizer from this email corpus. Using the vectorizer, I created a term frequency–inverse document frequency (tf-idf) matrix. This matrix weighs the importance of words based on their frequency in the email corpus. As a result, I didn’t remove common keywords such as email signatures because the tf-idf weighting scheme should weigh against email signatures and other common keywords. Next, I ran K-means with 60 clusters on the tf-idf matrix to cluster email content. For each cluster, I calculated the ratio of emails from POI over the total number of emails in that cluster. I looked for the top 5 email clusters with the greatest ratio of emails from POIs. I considered these clusters the most “suspicious” because they had a greater ratio of emails from POIs. My logic is that people who are POIs are probably also emailing other POIs more frequently and the content of their emails are more similar, therefore they appear in the same clusters. So the new feature is a calculation of the ratio of emails that are part of “suspicious” clusters over total number of emails. My new feature is called email\_suspicious\_ratio. The higher someone’s ratio of suspicious emails, the more they are communicating with POIs, and the more suspicious they are.

I didn’t end up using the feature because it decreased my precision and recall score. Using SelectKBest with my new feature, the most optimal k-value was 5 features ('shared\_receipt\_with\_poi', 'from\_poi\_to\_this\_person', 'suspicious\_email\_ratio', 'from\_messages', 'director\_fees'). This gave me the following metrics, which was lower than performance without the new feature. Even if I stuck with my original 3 features and added the new feature, I still got lower performance in terms of precision and recall:



|  |  |  |
| --- | --- | --- |
|  | Decision Tree without new feature | Decision Tree with new feature |
| Accuracy | 0.81583 | 0.80262 |
| Precision | 0.44385 | 0.35137 |
| Recall | 0.41500 | 0.33450 |
| F1 | 0.42894 | 0.34273 |
| F2 | 0.42047 | 0.33774 |

1. What algorithm did you end up using?  What other one(s) did you try? [relevant rubric item: “pick an algorithm”]

I used a Decision Tree algorithm. I tried a Naïve Bayes, C-Support Vector Classification, and KNearestNeighbor algorithm. I found Naïve Bayes and Decision Tree algorithm performed the best in terms of precision and recall score; the SVC and KNN both produced high accuracy but close to 0 precision and 0 recall. In the end, I chose to use the Decision Tree algorithm because I am limited in how much I can tune the parameters of a Naïve Bayes algorithm to optimize how it classified and the Decision Tree performed much better for precision. I did not choose to scale my features because neither the Naïve Bayes nor Decision Tree requires feature scaling for better performance.

Performance metrics without parameter tuning

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naïve Bayes | Decision Tree | K Nearest Neighbors | SVM (SVC) |
| Accuracy | 0.35036 | 0.78871 | 0.84664 | 0.85714 |
| Precision | 0.15873 | 0.27015 | 0.12308 | 0.00000 |
| Recall | 0.82500 | 0.28150 | 0.01200 | 0.00000 |
| F1 | 0.26624 | 0.27571 | 0.02187 | 0.00000 |
| F2 | 0.44849 | 0.27916 | 0.00738 | 0.00000 |

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 don’t 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 if you used, say, a decision tree classifier). [relevant rubric item: “tune the algorithm”]

Tuning parameters of an algorithm allows me to balance bias and variance. I used GridSearchCV to tune the parameters of Decision Tree algorithm and precision and recall scores were my validation metrics; the higher the precision and recall scores, the better tuned the algorithm.

I tried the following parameters using GridSearchCV and Decision Tree:

|  |  |  |
| --- | --- | --- |
| Parameter |  | Values Tried |
| min\_samples\_split |  | 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 15, 20 |
| min\_samples\_leaf |  | 1, 2 ,3 ,4, 10, 20 |
| max\_depth |  | None, 1, 2, 3 |
| splitter |  | 'best', 'random' |
| criterion |  | 'gini', 'entropy' |

Based on GridSearchCV, the best estimator is the following parameters:

DecisionTreeClassifier(compute\_importances=None, criterion=gini, max\_depth=None, max\_features=None, min\_density=None, min\_samples\_leaf=1, min\_samples\_split=2, random\_state=None, splitter=best)

Tuning parameters well helps ensure I made accurate predictions. Badly tuned algorithms can have 2 major issues: 1. I can create a bad classifier that doesn’t work at all with training or test data sets, and/or 2. I can fine-tune based on my training set, which ends up causing bad performance on my test set and overall generalizing for future incoming new data.

Based on GridSearchCV, I discovered that keeping the default values for all the parameters in the Decision Tree performed the best when I optimized for recall. I optimized for recall and not accuracy or precision because accuracy and precision were already above 0.3 but recall was consistently below 0.3. In the end, these were the following parameters I used in my Decision Tree algorithm:

clf = tree.DecisionTreeClassifier(criterion='gini', max\_depth=None, max\_features=None, min\_density=None, min\_samples\_leaf=1, min\_samples\_split=2, random\_state=None, splitter='best')

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 checking I didn’t overfit to my training data set so my algorithm is generalizable to independent future incoming data. I validated my analysis by splitting my dataset into two groups: a training set, which is used to select features, train my classifier, and parameter tune my algorithm, and a test set which is used to check that I didn’t overfit my algorithm. In addition to splitting data into training and test sets, validation also involves looking at metrics to objectively gauge how well my algorithm is performing.

I validated my analysis by using the tester.py script, which used StratifiedShuffleSplit with 1000 folds to separate the data set into training and test data. StratifiedShuffleSplit is more reliable than K-fold and train\_test\_split for this specific dataset because the dataset has very few training examples and is heavily skewed towards non-POI, which means a higher risk of a split that as no POI in the test set. Therefore, shuffling the split, taking several splits and averaging the results increases the chances that we didn’t split out an unusual test set that has either too many or no POIs which could happen because our sample size / number of training examples is small.

Note: I used train\_test\_split to split the dataset into training and test set for SelectKBest because it is less computationally intensive to run even though its results are less accurate than the StratifiedShuffleSplit. I was OK with this tradeoff because I was using the split training and test sets just for feature selection and not for testing my classifier model. Once I was ready to test my classifier, I used StratifiedShuffleSplit for validation.

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”]

**2 metrics:** precision = 0.44385, recall = 0.41500

Precision is the measure: of all the people I say are POIs, how many of them are actually POIs. Recall is the measure: of all the people who are truly POIs, how many of them was I able to identify as POI. The numbers provided (precision = 0.44385, recall = 0.41500) is the average metrics across multiple validation sets.

My metrics relied on precision and recall being at least 0.3 and not accuracy because I discovered that accuracy alone was unreliable because it was often high (at least 0.8) but precision and recall would both equal 0, which meant the algorithm was not performing well. This happens because the test set was so skewed toward non-POI, partially because of the small data set and partially because we had few POI training examples. Precision and recall are more granular metrics concerned with true positives, false positives, false negatives that help me recognize if my algorithm is actually doing a good job classifying.