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 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"]

The goal of the project is to correctly identify Persons of Interest (POI) in the Enron fraud case using machine learning algorithms and public available information. An Enron employee is considered a POI if he/she was indicted, reached a settlement or testified in exchange of immunity.

The dataset is composed of two types of information: financial and communicative (email metadata) from several Enron employees. Financial data consists of information such as the salary, stock value or bonuses received by the employee, while email metadata (derived from the pubic corpus of Enron emails) includes information about the number of messages sent or the number messages from a POI that each employee in the dataset received, among others. It is plausible that both the financial and communicative features can hold some information about the different behavior of POIs and non-POIs, and thus create a classifier that can differentiate between the two. The dataset also contains a label - `poi` - that determines if the employee was a POI or not, in order to test our algorithm.

While exploring the data, I noticed an outlier using a salary and bonus scatterplot: `TOTAL`. The `TOTAL` entry represents the sum of all the information from the rest of employees. I discarded it for being not being a valid employee but a spreadsheet error. Another erroneous entry that I noticed while exploring the data was `THE TRAVEL AGENCY IN THE PARK`. Not being an Enron employee, I also discarded it.

Finally, although not technically outliers, I also deleted the data from the employees I considered that did not have enough feature information for the analysis. `LOCKHART EUGENE E` was discarded as had only one valid entry in his dictionary: the `poi` label. The dataset used in the analysis had 143 entries, of which 18 were POIs.

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

The feature and algorithm selection in this project has been an iterative process. As I published in the forums, f1 scores for different feature selection, its preprocessing and algorithms parameter tuning were extremely variable.

To start with, of the 21 features available in the dataset, I discarded some for being too biased or not having enough employee information: `restricted_stock_deferred` and `director_fees` features were not valid in any POI and only three employees had valid values on `loan_advances`. I also did not make use of the `email_address` feature.

Although not necessary for all the different features and algorithms used (e.g. Decision Trees and its ensemble derivates), I applied `MinMaxScaler` to all the features in the dataset to account for the great differences in the feature value scales.

I started the feature selection using a Decision Tree and analyzing the importance of each feature to better understand the dataset. The results were the following:

Feature	*Importance*
exercised_stock_options	0.1998
bonus	0.1409
restricted_stock	0.1205
expenses	0.1179
total_payments	0.1130
long_term_incentive	0.1090
shared_receipt_with_poi	0.0578
from_poi_to_this_person	0.0556
from_messages	0.0481

other	0.0300
from_this_person_to_poi	0.0073
salary	0.0000
to_messages	0.0000
deferral_payments	0.0000
total_stock_value	0.0000
deferred_income	0.0000

Interestingly, in the end I did not select the starting 'most important' features as `exercised_stock_options` or `restricted_stock`, but `bonus` was part of the final feature list. After that, I started to create small modifications of the features, i.e., converting the absolute features to proportional for each employee if the score. I kept the modified feature if the f1 was greater than with the raw feature, as follows:

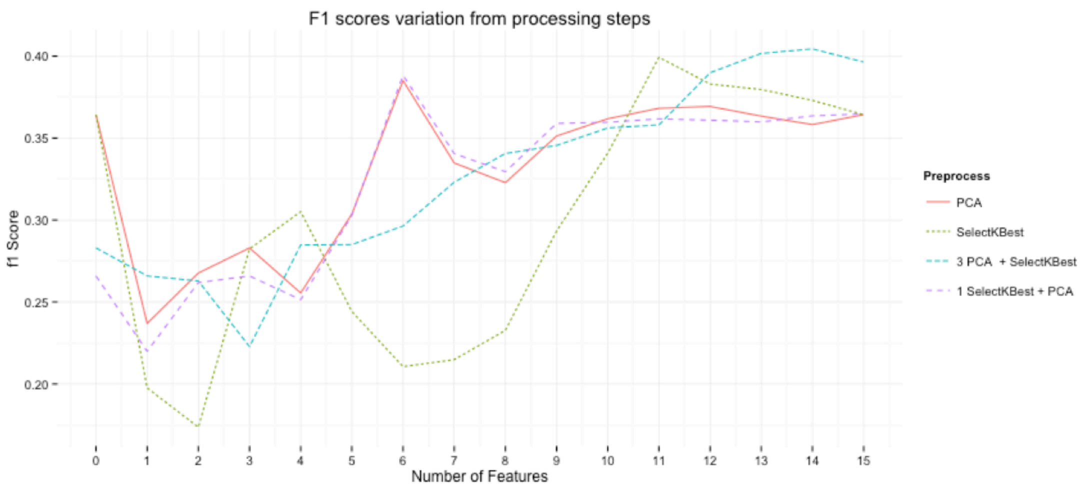
	f1 before	*f1 after*	*Kept*
proportion_stock_exercised	0.22273	0.22251	No
proportion_bonus	0.22273	0.24141	Yes*
proportion_restricted_stock	0.24141	0.27795	Yes
proportion_long_term_incentive	0.27795	0.29139	Yes
sharing_importance	0.29139	0.33377	Yes
proportion_received_from_poi	0.33377	0.35104	Yes
proportion_sent_to_poi	0.35104	0.29502	No

* In the final iteration I converted bonus back to absolute values to increase the f1.

Finally, deleting the `from_this_person_to_poi` feature increased the f1 to 0.39100.

At this stage, I started working on the algorithm performance and decided to use the `KNearestClassifier` as the final algorithm for this project (view other answers of the project for the rationale). Once the algorithm was settled, I went back to the feature selection and analyzed the use of two preprocessing steps: `SelectKBest` and `PCA`. As expected, using different component numbers as the selected features and components affected the f1 score.

I also analyzed how the combined use of `SelectKBest` and `PCA` affected the algorithm performance. The results can be found in the graphic below. The peak f1 score for `SelectKBest` was 11 features, and the peak for `PCA` was decomposing the features into 6 components, with f1 scores of 0.3993 and 0.38512, respectively. Using 14 features chosen by `SelectKBest` and 3 components with `PCA` resulted with a f1 score of 0.40443. Finally, combining the above peaks (11 features selected by `SelectKBest` and 6 components) I obtained the greatest performance score with feature processing steps: 0.40917.



Nevertheless, I was not happy with the results, as I knew because of my previous submission (and probably by pure luck) that I could get higher algorithm performance using the right selected features without feature processing steps. In fact, using all the fatures without preprocessing steps resulted in a higher f1 score: 0.43401

Out of curiosity, I ended up approaching the feature selection process with brute-force and trying combinations of up to 5 features. The f1 scores of the different combinations can be found in the files 1Feature.txt to 5Features.txt. The final selected features were `proportion_long_term_incentive`, `from_messages`, `bonus`, `proportion_received_from_poi` and `sharing_importance`, with a final f1 score of 0.59822.

3. 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 ended up using the `KNearestClassifier` algorithm. As the feature selection process, the algorithm selection was an iterative process with different features and retuning the algorithm for best performance. During the process, I tried the `GaussianNB`, `SVC`, `DecisionTreeClassifier`, `AdaBoostClassifier` and `RandomForestClassifier` algorithms. The final decision was made after algorithm tuning (as exposed in the next question), using all the features without preprocessing step. The evaluation metrics for each algorithm at this stage (without tuning) can be found below:

Algorithm	*Precission*	*Recall*	*F1*
GaussianNB	0.38468	0.231	0.28866
DecissionTree	0.391	0.0391	0.391
SVC	-	-	-
AdaBoostClassifier	0.42246	0.316	0.36156
RandomForestClassifier	0.49841	0.157	0.23878
KNeighborsClassifier	0.04651	0.004	0.00737

SVC gave an error message (*Precision or recall may be undefined due to a lack of true positive predictions*), because the algorithm probably considered all employees as non-POI (that would give an approximate accuracy of 0.85). Strikingly, the algorithm I ended selected after tuning (`KNearestClassifier`) was the worst performant with the automatic parameters.

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? (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"]

To tune the parameters of an algorithm means to change its input variables in order to get the best performance for a given dataset. I tuned my algorithms using the sklearn function `GridSearchCV` aiming for the best `f1` values and using the same cross-validation as in the `tester.py` script. The parameters tuned for each algorithm were the following:

- `GaussianNB: None`
- `SVC: C, gamma, kernel`
- `DecisionTreeClassifier: criterion, min_samples_split, max_features`
- `AdaBoostClassifier: algorithm, n_estimators`
- `RandomForestClassifier: criterion, min_samples_split, n_estimators`
- `KNearestClassifier: n_neighbors, algorithm, weights`

Algorithm	*Before Tuning*			*After Tuning*		
	Precission	*Recall*	*F1*	*Precission*	*Recall*	*F1*
GaussianNB	0.38468	0.231	0.28866	0.38468	0.231	0.28866
DecissionTree	0.391	0.0391	0.391	0.391	0.0391	0.391
SVC	-	-	-	0.17162	0.208	0.18807
AdaBoostClassifier	0.42246	0.316	0.36156	0.45884	0.3205	0.37739
RandomForestClassifier	0.49841	0.157	0.23878	0.49664	0.1845	0.26905
KNeighborsClassifier	0.04651	0.004	0.00737	**0.44072**	**0.4275**	**0.43401**

The decision of using `KNearestClassifier` as the algorithm for the submission was made at this stage with these results.

The final tuned algorithm parameters can be found at the end of the report.

5. 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 process performed after creating a Machine Learning algorithm to evaluate its general performance. A classic mistake in Machine Learning is to overfitting. In an overfit model, the model performs really well on the training data, but really poorly on the rest of the data. This is because the model created is too specific for its inputs and does not generalize well.

In order to validate my analysis (and evaluate algorithm tuning), I implemented the same cross validation techniques than in `tester.py`, that is, the creation of training and testing sets with an approximate 3:1 ratio 1000 folds with `StratifiedShuffleSplit` in the parameter tuning step. `StratifiedShuffleSplit` was selected because this dataset is small, and very skewed towards non-POI (18 POI in 143 entries), which results of a high risk of having training and test sets as biased and non-representative. Using 1000 folds of `StratifiedShuffleSplit` accounts for that risk and that's why it was selected as cross-validation technique.

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"]

In this project, I analyzed three different metrics to evaluate the algorithm performance: precision, recall and f1, although all my decision were finally based on the f1 score. Precision is the proportion of correctly identified data points class (true positives) to the total number of data points predicted for the class (true positives + false positives). In this example: the precision is the proportion of correctly identified POIs from all the predicted POIs.

Recall is the proportion of correctly identified data points in a class (true positives) to the total number of real data points in the class (true positives + false negatives). In this example: the recall is the proportion of POIs that were successfully labeled as POIs.

Finally, the f1 is a weighted average of the precision and recall, and this why I selected it as the metric to evaluate the algorithm perfomance. The obtained values for each algorithm tested with the final selected dataset in this project can be seen below:

Algorithm	*Precission*	*Recall*	*F1*
GaussianNB	0.56304	0.2925	0.385
DecissionTree	0.41507	0.336	0.37137
SVC	0.63298	0.1785	0.27847
AdaBoostClassifier	0.43665	0.274	0.33671
RandomForestClassifier	0.55	0.22	0.31429
KNeighborsClassifier	0.6744	0.5375	0.59822

In consequence, the final selected algorithm was `KNeighborsClassifier`, with the features `proportion_long_term_incentive`, `from_messages`, `bonus`, `proportion_received_from_poi` and `sharing_importance`.

Tuned algorithm and its tuned parameters:

```
GaussianNB()

SVC(C=10000, cache_size=200, class_weight=None, coef0=0.0, degree=3,
gamma=0.1, kernel='rbf', max_iter=-1, probability=False,random_state=None,
shrinking=True, tol=0.001, verbose=False)
parameters = {
  'C': [1, 1e2, 1e3, 1e4, 1e5, 1e7],
  'gamma': [0.0, 0.001, 0.01, 0.1],
}

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0, random_state=42,
splitter='best')
parameters = {
  'criterion': ['gini', 'entropy'],
  'min_samples_split': [2, 5, 7, 10, 15, 30],
}

AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
learning_rate=1.0, n_estimators=10, random_state=42)
parameters = {
  'algorithm' : ['SAMME', 'SAMME.R'],
  'n_estimators' : [10,100,1000],
}

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
max_depth=None, max_features='auto', max_leaf_nodes=None, min_samples_leaf=1,
min_samples_split=5, min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
oob_score=False, random_state=42, verbose=0, warm_start=False)
parameters = {
  'criterion': ['gini', 'entropy'],
  'min_samples_split': [2,5,7,10,15,30],
  'n_estimators' : [10,100,1000],
}

KNeighborsClassifier(algorithm='ball_tree', leaf_size=30, metric='minkowski',
metric_params=None, n_neighbors=1, p=1, weights='uniform')
parameters = {
  'n_neighbors':[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15],
  'algorithm':['ball_tree','kd_tree','brute','auto'],
  'weights': ['uniform', 'distance'],
  'p': [1, 2],
}
```