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"]

ANSWER: In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives. The project designer has combined the data with a hand-generated list of persons of interest(POI) in the fraud case, which means individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

The dataset of the project includes people's financial data, emails activity data and the identifier of POI. The goal of this project is to use the dataset, select appropriate features and algorithm, tune and train and predict if a person is a POI, and evaluate the algorithm performance.

In the data exploration phrase, I have these findings:

The total number of data points is 146

The number of poi is 18

The classes are unbalanced!

All features are 21

I found an outlier which key of data dictionary is "TOTAL". I removed this data point. And I checked all features' null value percentage:

the feature salary has null value percentage is 0.349

the feature deferral_payments has null value percentage is 0.733

the feature total_payments has null value percentage is 0.144

the feature loan_advances has null value percentage is 0.973

the feature bonus has null value percentage is 0.438

the feature restricted_stock_deferred has null value percentage is 0.877

the feature deferred_income has null value percentage is 0.664

the feature total_stock_value has null value percentage is 0.137

the feature expenses has null value percentage is 0.349

the feature exercised_stock_options has null value percentage is 0.301

the feature other has null value percentage is 0.363

the feature long_term_incentive has null value percentage is 0.548

the feature restricted_stock has null value percentage is 0.247

the feature director_fees has null value percentage is 0.884

the feature to_messages has null value percentage is 0.411

the feature from_poi_to_this_person has null value percentage is 0.411

the feature from_messages has null value percentage is 0.411

the feature from_this_person_to_poi has null value percentage is 0.411

the feature shared_receipt_with_poi has null value percentage is 0.411

We keep all features before features selection. I noticed that in the featureFormat().all missing values are set to 0.

The unbalanced POI classes mean we need to use cross-validation method like Stratified Shuffle Split other than train_test_split. StratifiedShuffleSplit makes sure the ratio of POI

and non-POI is the same during training and testing. The unbalanced data also indicate that accuracy is not a good evaluation metric, we need to use precision and recall instead.

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”]

ANSWER:I created two features:"rate of poi/from" and "rate of poi/to",they are calculated by:
Df_1["rate of poi/from"]=df_1["from_this_person_to_poi"]/df_1["from_messages"]
df_1["rate of poi/to"]=df_1["from_poi_to_this_person"]/df_1["to_messages"]
I think if there are large part of emails from pois in the person's "to_messages" ,or large part of emails to pois in the person's "from_messages", He will be suspicious to be a poi.
I test the metric performance of the two self-implemented features by test_classifier() of tester.py. Using decisiontreeclassifier, tuned with GridSearchCV, Cross validation with StratifiedShuffleSplit and I got the result:
Accuracy: 0.83200 Precision: 0.34420 Recall: 0.38000 F1: 0.36122 F2: 0.37226
That’s pretty good. But there’s a reason for such good metric is there is possible data leakage. Because "from_this_person_to_poi","from_poi_to_this_person" and "shared_receipt_with_poi", the 3 features rely on information of all data points, including trainsets and testsets. When the classifier is indeed used to predict a person's POI-ness, we obviously do not know beforehand. To avoid possible data leakage, I remove the 2 self-implemented features and
"shared_receipt_with_poi" ,"from_this_person_to_poi","from_poi_to_this_person" from the later training trails.
During features selection phrase, I began with all features except the removed features using decisiontree feature importances attribute, I didn't do scaling, because decisiontreeclassifier uses vertical and horizontal lines won't be affected by the range and scale of data. I used StratifiedShuffleSplit to get trainset and testset. Notice that feature importances ranking are different by using StratifiedShuffleSplit and train_test_split. Since POI classes are unbalanced , we use StratifiedShuffleSplit.
The feature importances of the features are:
Feature Ranking:
feature no. 1: exercised_stock_options (0.226)
feature no. 2: bonus (0.204814821291)
feature no. 3: expenses (0.196718108447)

feature no. 4: restricted_stock (0.166396367595)
feature no. 5: from_messages (0.124659028057)
feature no. 6: total_payments (0.0457368958475)
feature no. 7: to_messages (0.0356747787611)
feature no. 8: other (0.0)
feature no. 9: total_stock_value (0.0)
feature no. 10: salary (0.0)
I began with all features, and got the metrics by tester.py:
Accuracy: 0.80887 Precision: 0.26504 Recall: 0.24450 F1: 0.25
436 F2: 0.24835
features list selected: ['exercised_stock_options', 'bonus', 'expenses',
'restricted_stock', 'from_messages']
Accuracy: 0.81236 Precision: 0.32632 Recall: 0.29450 F1: 0.30
959 F2: 0.30036
features list selected: ['exercised_stock_options', 'bonus', 'expenses']
Accuracy: 0.83064 Precision: 0.40982 Recall: 0.42150 F1: 0.41
558 F2: 0.41911
The third trial was much better than the 2 trial before. That means minimum numbers of features with independency are better than lots features with dependency or non-linear relationships.

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”]

ANSWER: Because this is a supervised classifier problem, the predicted poi is a binary categorical value. I think svm and decision tree is appropriate. I have selected features by using decisiontreeclassifier's attribute:*featureimportances*. And it no need to scaling using decisiontree.So decision tree is a good choice. I got the best metric after features selection using decisiontree is:
features list selected: ['exercised_stock_options', 'bonus', 'expenses']
Accuracy: 0.83064 Precision: 0.40982 Recall: 0.42150 F1: 0.41
558 F2: 0.41911

I also tried SVM with the all features. SVM needs feature scaling.I used MinMaxScaler before SVC. and I've tried kernel=linear/poly,it took more than 15mins and I had to abort before it finished.I also tried kernel=rbf and got “Precision or recall may be undefined due to a lack of true positive predicitons”.So it was not a good choice to use SVC.

As a ensemble algorithm, I also tried AdaBoostClassifier with all features and I got the result by tester.py:.
Accuracy: 0.84273 Precision: 0.38152 Recall: 0.28900 F1: 0.32
888 F2: 0.30373
features list selected: ['exercised_stock_options', 'bonus', 'expenses',
'restricted_stock', 'from_messages']

Accuracy: 0.83643 Precision: 0.39731 Recall: 0.28050 F1: 0.32884 F2: 0.29802
features list selected: ['exercised_stock_options', 'bonus', 'expenses'].
Accuracy: 0.86214 Precision: 0.51898 Recall: 0.47850 F1: 0.49792 F2: 0.48608
The third trail had higher precision and recall than decisiontree . So I select adaboost as final algorithm.

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"]

ANSWER: I found adaboost can be used as a classifier and I began to tuned parameters on the algorithm. I used GridSearchCV to turn parameters of AdaBoostClassifier and find the parameters with better prediction performance. I used StratifiedShuffleSplit() for cross validation to get more accurate evaluation of performance on the unbalanced POI classes dataset. And scoring with recall. I prefer to use recall and precision to evaluate the performance other than accuracy.
a_grid_search=GridSearchCV(estimator=clf_ada,param_grid=parameters,cv=cv,scoring='recall')
In parameters, I set random_state because I wanted the classifier can performed replicate by the same random_state.
parameters={"random_state":[24,42,60],"learning_rate":[1.0,2.0,3.0]}
If I don't use GridSearchCV,I have to turn it manually. That would be tedious and low efficiency.
Here is final best parameter's result:
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
learning_rate=3.0, n_estimators=50, random_state=24)
Accuracy: 0.45386 Precision: 0.19891 Recall: 0.93250 F1: 0.32788 F2: 0.53666
We got 0.9 recall!!!

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"]

ANSWER: We typically divided the dataset as trainset, testset, and validationset. By partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model. A solution to this problem is a procedure called cross-validation. A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called k-fold CV,

the training set is split into k smaller sets (other approaches are described below, but generally follow the same principles). The following procedure is followed for each of the k “folds”:

- A model is trained using k-1 of the folds as training data;
- The resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data (as it is the case when fixing an arbitrary test set), which is a major advantage in problem such as inverse inference where the number of samples is very small. I using cross validation and get the average test result which is more accurate for K experiments.

I think a typical mistake when using CV is when POI is unbalanced classes, we need to use StratifiedShuffleSplit() to ensure the percentage of POI and nonPOI are selected in each trainset and testset pairs. To avoid random selection only select nonPOIs in trainset or testset pairs. It’ll result in bad model. Be noticed that Each time I used CV in the ML case, I used StratifiedShuffleSplit() other than train_test_split().

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”]

ANSWER: In the final model, I used AdaBoostClassifier as final algorithm, features list selected: ['exercised_stock_options', 'bonus', 'expenses']
Here is final best parameter’s result:
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
learning_rate=3.0, n_estimators=50, random_state=24)
Accuracy: 0.45386 Precision: 0.19891 Recall: 0.93250 F1: 0.32788 F2: 0.53666
We got 0.93 recall!!!

POI is skewed class, much more nonPOIs than POIs, accuracy has less meaning, high accuracy does not mean better performance. We should use recall and precision instead of accuracy. Recall and precision are used for binary classification. High recall means less false negative. High precision means less false positive. In the investigation of Enron case, we prefer to accept more false positive than accept more false negative. That means when a person is suspicious, we prefer to mark him as POI and do investigation. That means we prefer high recall than high precision. That why I tuned adaboostclassifier by

GridSearchCV with scoring=recall. In final algorithm, we got 0.93 recall. Suppose our model observed that there’s a POI, the chance that it’s actually a POI is 0.19(precision).The probability of the model to correctly identify is 0.93(recall) provided that the person actually is a POI.