UDACITY

PROJECT

Identify Fraud from Enron Email

A part of the Data Analyst Nanodegree Program

PROJECT REVIEW

CODE REVIEW 4

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Dear Student,

Great work!. Yours is a strong project!. You demonstrate a good understanding of Machine Learning and your report is written in explanatory terms allowing your audience to follow and understand the work done.

Congratulations on passing your exam and stay Udacious!

Quality of Code

Code reflects the description in the answers to questions in the writeup. i.e. code performs the functions documented in the writeup and the writeup clearly specifies the final analysis strategy.

Your written response perfectly describes your strategy, includes the required level of detail when required and your code includes all the processes mentioned. Machine learning is not just about building good models, it is also about communicating results to your audience in a clear and direct way. You achieve both goals. Well done!.

As a suggestion, this project is a great opportunity for you to create a new repository in Github that becomes part of your online portfolio and allow potential employers to review your work. This report defines your credentials, so it is important that you put special attention not just to the technical side of the project but also the communications side since this is a critical characteristic for any data scientist. For your reference, check this Kaggle post for further reference, as you can see this is really a hot topic in the data science world! 😉

poi_id.py can be run to export the dataset, list of features and algorithm, so that the final algorithm can be checked easily using tester.py.

All required `.pkl` files are included and `poi_id.py` worked without problems.

Understanding the Dataset and Question

Student response addresses the most important characteristics of the dataset and uses these characteristics to inform their analysis. Important characteristics include:

- **total number of data points**
- **allocation across classes (POI/non-POI)**
- **number of features used**
- **are there features with many missing values? etc.**

Your report includes a description of the main characteristics of the dataset.

Note these numbers are particularly important since they describe the main dataset characteristics:

1. The small data set is why the tester.py file uses StratifiedShuffleSplit instead of a simpler cross-validation method such as TrainTestSplit. StratifiedShuffleSplit will make randomly chosen training and test sets multiple times and average the results over all the tests.

2. The data is unbalanced with many more non-POIs than POIs. StratifidShuffleSplit also makes sure that the ratio of non-POI:POI is the same in the training and test sets as it was in the larger data set.
3. The unbalanced data is also why we use precision and recall instead of accuracy as our evaluation metric.

For your reference, some techniques to handle unbalanced datasets and this repo with plenty of tools.

Student response identifies outlier(s) in the financial data, and explains how they are removed or otherwise handled.

Excellent work on the outlier detection.

Optimize Feature Selection/Engineering

At least one new feature is implemented. Justification for that feature is provided in the written response, and the effect of that feature on the final algorithm performance is tested. The student is not required to include their new feature in their final feature set.

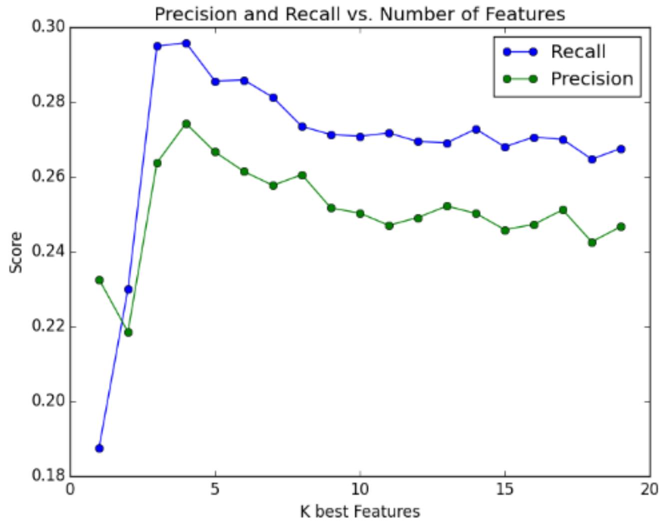
Good intuition!, your engineered features definitely improve your results.

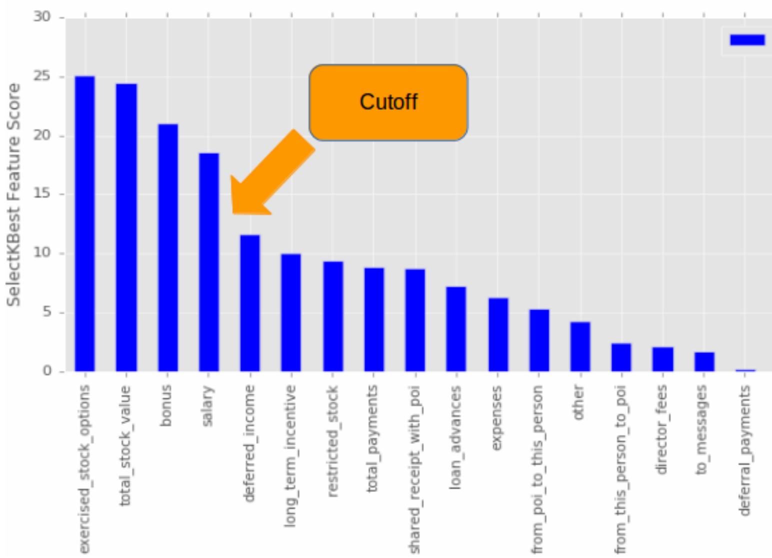
Univariate or recursive feature selection is deployed, or features are selected by hand (different combinations of features are attempted, and the performance is documented for each one). Features that are selected are reported and the number of features selected is justified. For an algorithm that supports getting the feature importances (e.g. decision tree) or feature scores (e.g. SelectKBest), those are documented as well.

Good work in this section. Note feature selection process is key in Machine Learning problems, the idea behind it is that you want to have the minimum number of features than capture trends and patterns in your data. Your machine learning algorithm is just going to be as good as the features you put into it. For that reason, this is definitely a critical step into any ML problem and the methodology deployed must be scientific and exhaustive without room for intuition.

Having said so, your methodology is scientific and exhaustive as required, SelectKBest ranks the level of correlation between features and the target label, and by testing the different feature sets generated for each value of K in its range, your methodology is exploratory. Now, since you repeated this process for different classifiers, you identified which is the best performing classifier for this dataset, that goes beyond the project expectations, but definitely, it is worth to do in the real world. Well done!

As a side comment, when working with larger datasets, since training and testing your models for each feature selection is computationally expensive, a good decision is to use the feature scores (ordered by ranking) to determine where the feature scores drop and so determine the cutoff as explained in the picture below.





If algorithm calls for scaled features, feature scaling is deployed.

It is great you scaled your features since some of the classifiers attempted calls for it. However, it would be great if you could include few words in your written response describing which of these algorithms call for scaled features.

Pick and Tune an Algorithm

At least 2 different algorithms are attempted and their performance is compared, with the more performant one used in the final analysis.

Good work testing several algorithms and comparing their performance in terms of precision&recall (and f1 and f2).

As a side comment, note F1 is a subclass of the F-Scores as here described. For example, F0.5 puts more importance over Precision than Recall, it is up to your problems needs to decide any F-Score between 0 (only considers Precision) and infinite (only consider Recall). Here you can find more information.

Response addresses what it means to perform parameter tuning and why it is important.

Good understanding of tuning.

At least one important parameter tuned with at least 3 settings investigated systematically, or any of the following are true:

- GridSearchCV used for parameter tuning
- Several parameters tuned

- Parameter tuning incorporated into algorithm selection (i.e. parameters tuned for more than one algorithm, and best algorithm-tune combination selected for final analysis).

Excellent use of GridCV to tune your classifier, I especially like how you optimized it by passing the right CV object and oriented your search to maximize the score of your choice.

As a suggestion, when working with larger datasets (= real world, competitions) RandomizedGridCV is a good alternative (much faster than GridSearchCV) since it explores a subset of points in the parameter space defined.

For your reference, note you can also visualize your grid results.

Validate and Evaluate

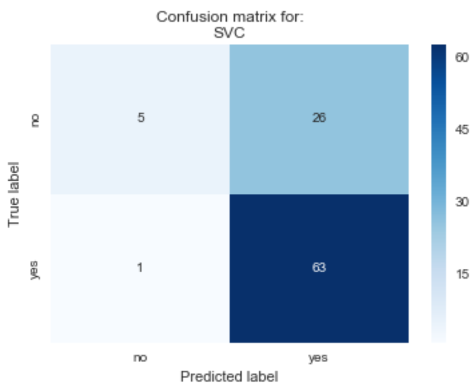
At least two appropriate metrics are used to evaluate algorithm performance (e.g. precision and recall), and the student articulates what those metrics measure in context of the project task.

Good work using precision&recall to evaluate your classifier. Also, good definition of both in terms of POI detection.

As a side comment: Note accuracy is not a good score in this case since the dataset in this project is very small and the ratio of negatives to positives is highly skewed (18 POIs out of 146 total cases), a classifier that predicted only non-POIs as output, would obtain an accuracy score of 87.4%. In this regard, accuracy is not a strong validation metric for our classifier. For your reference, this interesting post.

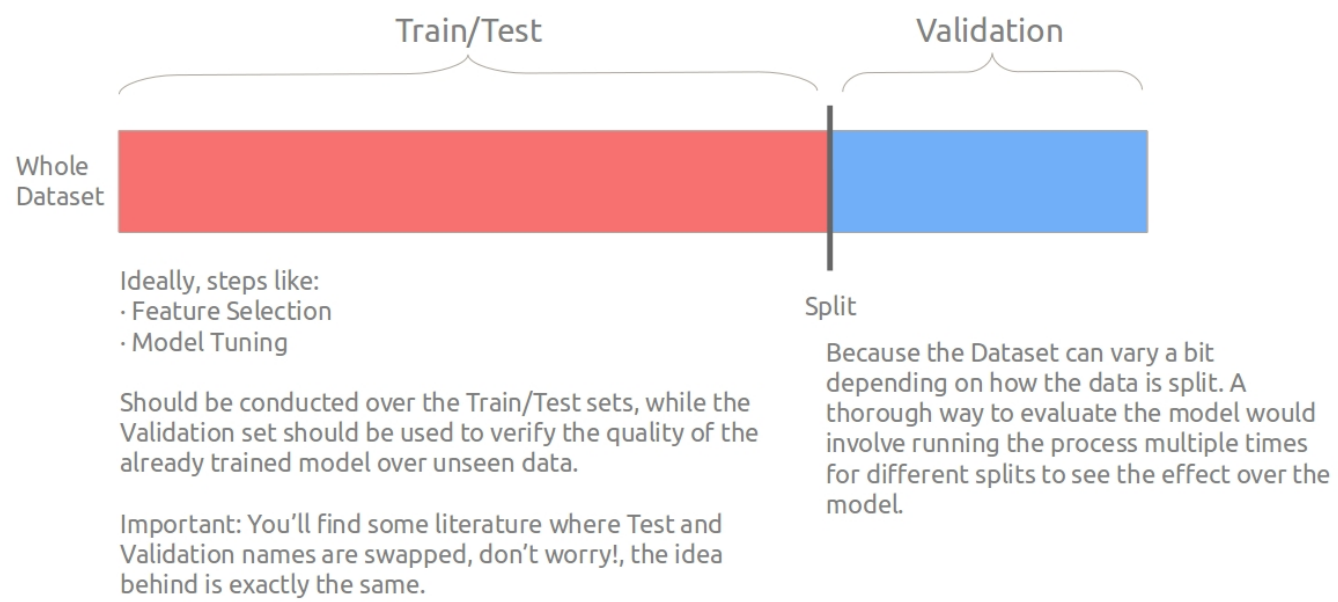
As a suggestion, consider a confusion matrix for more detailed information about classifier performance.

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Compute confusion matrix for a model
model = clf_C
cm = confusion_matrix(y_test.values, model.predict(X_test))

# view with a heatmap
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=['no', 'yes'], yticklabels=['no', 'yes'])
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title('Confusion matrix for:\n{}'.format(model.__class__.__name__));
```



Response addresses what validation is and why it is important.

Good understanding of validation. As a side comment, since this dataset is really really small, validation is performed just using the test set. In the real world, you would split your dataset in train/test/validation in order to validate your trained&tuned model against completely unseen data. For further reference, check this excellent answer.



Performance of the final algorithm selected is assessed by splitting the data into training and testing sets or through the use of cross validation, noting the specific type of validation performed.

For this particular problem, a stratified shuffle split is preferred for a number of reasons. Since the dataset is small, a shuffle split will randomly assign entries to test and training sets in a fold. However, the stratification preserves the percentage in the target class as in the complete set. This is particularly important Validation for our investigation as we have such a small percentage of PoIs - if we did not stratify we could easily have a test or train set which contained no entries that were PoIs. If we had no entries that were PoIs in either a test or train set then the model would perform very badly.

When tester.py is used to evaluate performance, precision and recall are both at least 0.3.

Well done!. Your classifier is over 0.3 for precision&recall scores.

DOWNLOAD PROJECT

4 CODE REVIEW COMMENTS ›

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH

Rate this review

Student FAQ