Intro to Machine Learning: Project 5

Introduction

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, there was a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for to executives.

Utilizing scikit-learn and machine learning methodologies, I built a "person of interest" (POI) identifier to detect and predict culpable persons, using features from financial data, email data, and labeled data--POIs who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

Code:

```
#!/usr/bin/python

import sys

import pprint

import pickle

sys.path.append("../tools/")

from feature_format import featureFormat, targetFeatureSplit

from tester import test_classifier, dump_classifier_and_data

### Task 1: Select what features you'll use.

### features_list is a list of strings, each of which is a feature name.

### The first feature must be "poi".

features_list = ['poi','salary','exercised_stock_options', 'bonus']

### Load the dictionary containing the dataset

with open("final_project_dataset.pkl", "r") as data_file:

    data_dict = pickle.load(data_file)

### Task 2: Remove outliers

for employee in data_dict:

 print employee

for employee in data_dict:

    pprint.pprint(data_dict[employee])
```

```
### Print one example

pprint.pprint(data_dict['LAY KENNETH L'])

#What is the Travel Agency & Total?

pprint.pprint(data_dict['THE TRAVEL AGENCY IN THE PARK']) #mostly NaNs not a person

pprint.pprint(data_dict['TOTAL']) #just the Total

#Let's see if there are any duplicates

employee_names = []

for employee in data_dict:

    employee_names.append(employee)

print len(employee_names)

employee_set = set(employee_names)

print len(employee_set)

#No duplicates...

#Maybe there are names spelled slightly different twice?

#Sort alphabetically and visually inspect

employee_names.sort()

pprint.pprint(employee_names)

#All names except for Total and Travel Agency seem valid

###Remove the 2 outliers from the original dataset

data_dict.pop('THE TRAVEL AGENCY IN THE PARK', 0)

data_dict.pop('TOTAL', 0)

print len(data_dict) #144 records as expected


### Task 3: Create new feature(s)

#POIs

for employee in data_dict:

    if data_dict[employee]['poi']:

        print float(data_dict[employee]['restricted_stock'])/float(data_dict[employee]['total_stock_value'])

#Non-POIs

for employee in data_dict:

    if not data_dict[employee]['poi']:
```

```
        print float(data_dict[employee]['restricted_stock'])/float(data_dict[employee]['total_stock_value'])

        print float(data_dict[employee]['expenses'])/float(data_dict[employee]['salary'])

#About a third of POIs have all their stock as restricted_stock (1.0)

#Let's add it to the data_dict

for employee in data_dict:

    data_dict[employee]['restricted_stock_ratio'] = round(float(data_dict[employee]['restricted_stock']) / \

                                            float(data_dict[employee]['total_stock_value']),2)

### Store to my_dataset for easy export below.

my_dataset = data_dict


### Extract features and labels from dataset for local testing

data = featureFormat(my_dataset, features_list, sort_keys = True)

labels, features = targetFeatureSplit(data)


### scale features via min-max

from sklearn import preprocessing

scaler = preprocessing.MinMaxScaler()

features = scaler.fit_transform(features)


### Task 4: Try a varity of classifiers

### Please name your classifier clf for easy export below.

### Note that if you want to do PCA or other multi-stage operations,

### you'll need to use Pipelines. For more info:

### http://scikit-learn.org/stable/modules/pipeline.html


# Provided to give you a starting point. Try a variety of classifiers.

###FINAL CHOSEN ALGORITHM AND PARAMETERS - Better Recall, more True Positives

from sklearn.neighbors import KNeighborsClassifier


clf = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',

           metric_params=None, n_neighbors=5, p=2, weights='distance')
```

```
### Task 5: Tune your classifier to achieve better than .3 precision and recall
### using our testing script. Check the tester.py script in the final project
### folder for details on the evaluation method, especially the test_classifier
### function. Because of the small size of the dataset, the script uses
### stratified shuffle split cross validation. For more info:
### http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShuffleSplit.html
test_classifier(clf, my_dataset, features_list)


# Example starting point. Try investigating other evaluation techniques!
from sklearn.cross_validation import train_test_split
features_train, features_test, labels_train, labels_test = \
    train_test_split(features, labels, test_size=0.3, random_state=42)


### Task 6: Dump your classifier, dataset, and features_list so anyone can
### check your results. You do not need to change anything below, but make sure
### that the version of poi_id.py that you submit can be run on its own and
### generates the necessary .pkl files for validating your results.


dump_classifier_and_data(clf, my_dataset, features_list)
```

Short Questions

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?

The goal of this project is to build a predictive model to identify a “person of interest” (“poi”),who may be involved in fraud in Enron. Financial data and communication record of people in Enron are essential features that may help to uncover “pois”. Therefore, dataset extracted from Enron email corpus including financial and email information were analyzed and applied for training of machine learning algorithms. To do that, the dataset was explored first. The exploration shows that the dataset has 3066 data points, of which 1708 data points are valid. There are18 persons labeled as “poi” among total 146 persons, and 19 numeric features plus a “poi” feature and an “email_address” feature. The outlier identifier captured: invalid name for a person: “TOTAL” (a summary of all persons) and “THE TRAVEL

AGENCY IN THE PARK" (not a valid name); To keep the quality of analysis, these outliers were removed before further analysis.

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 doesn't come ready-made in the dataset--explain what feature you tried to make, and the rationale behind it. If you used an algorithm like a decision tree, please also give the feature importances of the features that you use.

I used 'domain knowledge' to select the features for the model. After watching the film "Enron: The Smartest Guys in the Room", I became somewhat knowledgeable about the features found in a few of the people of interest such as Lou Pai, Jeff Skilling, Ken Lay, and Andy Fastow. I noticed that Lou had a really large expense account; he was known to spend a lot of the company's money on corporate jet trips and strip clubs for example. Another thing that was frequently mentioned in the movie was that a lot of the POIs exercised a lot of options as they knew that the company's fortunes were bound to eventually turn around due to the fraudulent activities that were commonplace. So for the initial feature selection process, I chose actually 6 features which I hypothesized would give me a good starting model. The new feature I generated was restricted_stock_ratio (restricted_stock/total_stock_value). I originally noticed that a lot of the POIs had a 1/1 ratio (more than 50%) and when doing more research on restricted stock I learned this was a popular form of compensation to executives due to its favorable tax consequences. In the end, I didn't need to add this model because my simpler model with only 3 features (salary','exercised_stock_options', 'bonus') was able to predict POIs accurately with just over 86 % accuracy. In regards to feature scaling, since my K nearest neighbor's model used Euclidean distance to calculate the nearest neighbors, I thought feature scaling would indeed improve my model. However, after writing a function to calculate NaNs, I noticed that a large percentage of my data was NaN for the features I had chosen (salary, exercised_stock_options,bonus); the percentages of NaN respectively were (35%, 30%, 44%). Nonetheless, I tried an imputation strategy of using the median values to replace NaNs and then reran the model. I was shocked to see that accuracy, precision, and recall all went down when I used rescaled features. My hypothesis was that the imputation strategy could have affected the model too much since there were so many NaNs that were imputed with the median. Ultimately, I decided against using any scaled features.

3)What algorithm did you end up using? What other one(s) did you try?

The final algorithm that I ended up using was a Nearest Neighbors (kNN) algorithm in a supervised classification context. I considered the more complicated DecisionTree and the RandomForest algorithms. On my first run, I actually got better results with k Nearest Neighbors and, from my perspective; it is also the simpler model to interpret as well. As I learned in an earlier Statistics course, it is always best to choose the parsimonious model when there are similar alternatives with similar results (in fact this case this simple model also performed better for accuracy, precision, and recall).

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 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 the parameters of an algorithm means to pull some of the 'levers' to influence the results of the model. If you don't tune your parameters appropriately, you will end up using the defaults, which will likely not result in an optimized model. This means the accuracy, precision, recall or other performance measure is not as good as it could be because the model was not customized to the particular dataset's features (in this case features is not referring to the predictor variables). In this case, I tuned the parameters manually by trying some different combinations as I figured it was simple enough without having to use more complicated methods such as GridSearchCV. The main parameter I played with was actually k itself. k refers to the number of surrounding nearest neighbors to look at when voting on the majority class. I found that the optimal number to get the accuracy, precision, and recall I wanted was k = 5. This means if my predicted data point was close to 3 POI and 2 non-POI (the 5 closest neighbors) it would be classified itself as POI because the majority rules. I also found that instead of giving a uniform weight to all neighbors, I decided to weigh the value of each neighbor based on how far away from the data point they were (weights= 'distance'). This means that neighbors that were further away are weighted less than closer points.

5)What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?

Validation is a strategy to separate your data into training and testing dataset initially. This allows you to train your model on a training set and then test that model on an independent test dataset. This reduces the problem of over fitting your model to the dataset that you initially trained it on. If you only tested the model on your training dataset you would be shooting yourself in the foot because you would not know how well your model would perform on a new, unknown dataset. While validating on a test set, if you got much poorer results you would know that there is a chance you over fitted your model to the training dataset. I validated my dataset with the help of the test_classifier () function that used the sklearn StratifiedShuffleSplit () function to do the splitting of the data into a training set and test set. The parameter for n_iter (represented by the folds variable) was 1000. That means the model is run 1000 different times and for each iteration a random test data set is used (this test dataset still comes from the original dataset and is usually different from previous test data sets).

6) Give at least 2 evaluation metrics, and your average performance for each of them. Explain an interpretation of your metrics that says something humanunderstandable about your algorithm's performance. [relevant rubric item: "usage of evaluation metrics"]

Accuracy: Accuracy refers to the ratio of correct predictions out of the total predictions made. In this context, it means how many POIs the model was able to correctly predict. The average performance for the kNN model I tuned was 0.86969. This means nearly 87% of predictions the model made were correct.

Precision: Precision refers to the ratio of correct positive predictions made out of the total positive predictions made (Positive predictions means predicting that the employee is a POI). 1156 total positive predictions were made by the model on the test data and the amount of these positive predictions that was were correct was 731. This is why the precision of the model was just over 63%.

Recall: Recall is a more difficult thing to conceptualize for many (I have attached an image from Wikipedia below that should help). Recall refers to the ratio of correct positive predictions made out of the actual total that were indeed positive (correct positive predictions + incorrect false negative predictions). We are basically looking only at the actual POIs in the test data set and seeing what proportion of them were correctly identified. The model was able to achieve a recall of about 37%. You can see that there is a tradeoff between precision and recall which needs to be balanced in all models depending on what is more important for the model. However, in cases such as this which contains a lot more of one class over another class (way more non-POI than POI), recall and precision are both better measures than accuracy. Just predicting that everyone is not a POI would get a pretty high accuracy due to the fact there are just a lot more non-POIs in the dataset. This is a very important takeaway I learned from this entire process; just because a model has a very high accuracy doesn't necessarily mean it is a great model.

Resources and References

Introduction to Machine Learning (Udacity)

Machine Learning (Stanford/Coursera)

scikit-learn Documentation

data/: dataset files and pickle objects

data/emails_by_address/: email message data (not used)

tools/: helper tools and functions

scripts/enron.py: containes helper functions and exploratory data analysis

scripts/evaluate.py: custom validation metrics

scripts/poi_email_addresses.py: prints dataset email addresses (not used)

scripts/poi_id.py: POI identifier

scripts/tester.py: Udacity-provided file, produces relevant submission files