INTRO TO MACHINE LEARNING

IDENTIFYING FRAUD FROM ENRON EMAIL AND FINANCIAL DATA

Vishal Vikram Singh

Introduction

In 2000, Enron was one of the biggest companies in USA.By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting federal investigation a significant amount of private data entered into the public domain including thousands of emails and financial data of top executives.

In this project i applied my machine learning skills to create person of interest identifier based on financial and email data made public by Federal Agency after Enron Scandal.

1.Datasets and Outliers

The dataset provided consist of 146 data points and have 21 features. Out of this 146 persons 18 are labeled as Person of Interest. After plotting the "salary" feature against "poi" feature ,one outlier can be detected with the index"TOTAL".Hence this was removed from the dataset."poi" feature column is used as labels here.

2.Feature creation and selection

After carefully examining the dataset,intuitively it becomes obvious that combined features can be generated by combining "salary","bonus","exercised_stock_options" ,"long_term_incentive" .i stored the this new feature in my dataframe column named "wealth".Apart from that i generated two other features "fraction_from_poi", and"fraction_to_poi", which contains the fraction of emails received from person of interest and sent to person of interest respectively.

On selection part, intuitively i selected all the features at first except "email-address" and "director_fees" and fed to the SelectKBest algorithm to identify top scoring features.

```
from sklearn.feature_selection import SelectKBest
select_best=SelectKBest(k=5)
select_best.fit(features,labels)
print(select_best.scores_)
```

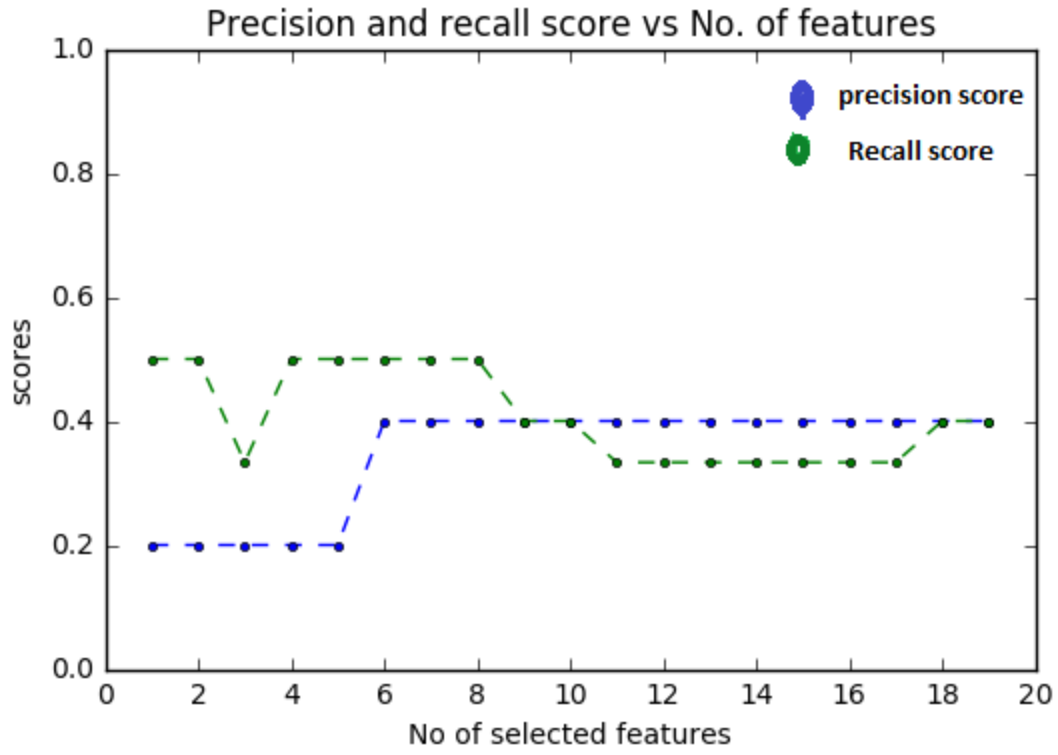
```
[ 21.32789041   0.20970584  11.73269808  25.3801053    6.37461449
   0.15877024   5.44668748   2.47052122   7.30140665  10.22290421
   4.26357664   9.4807432    0.06447703  18.86179532   8.90382156
   1.75169428   8.96781935  24.75252302  19.70514459   4.79734906
   4.24369185]
```

Top high scoring features i selected are as follows:

```
final_features=df[["bonus","exercised_stock_options","salary","total_stock_value","wealth","deferred_income"]]
```

Justification for selection top 6 scoring features can be given by the following diagram which is plotted using obtained scores against different value of **k** parameter in **SelectKBest** .We can see that when selected features are 6 , we obtained better precision and recall score. Although similar performance can be achieved with other number of features but here we chose the minimum to avoid overfitting in some classifiers like Decision Tree.

It is also obvious from the plot that when i included the feature named"wealth" the performance of the GaussianNB classifier improved significantly. When i used top 3 features precision and recall score was approximately 0.2 and 0.3 respectively but after including extra feature "wealth" , the recall score jumped to 0.5 which is significant improvement in the performance of the Classifier.



Hence i selected the top 6 scoring features using **SelectKBest** , with the following scores:

Feature name	**score**
1."exercised_stock_options"	25.3801053
2."Total_stock_value"	24.75252302
3."bonus"	21.32789041
4."wealth"	19.70514459
5."salary"	18.86179532
6."deferred_income"	11.73269808

3.Feature scaling

After feature selection , i performed feature scaling by using scikit-learn module named **MinMaxScaler.**

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(final_features)
scaled_features=scaler.transform(final_features)
print(scaled_features.shape)
```

```
(145L, 6L)
```

4.Algorithm Selection

I tested different classifier along with scikit-learn's **GridSearchCV** module which performed parameter tuning and cross-validation.Before applying any algorithm i have splitted my dataset into training and testing set using scikit-learn's **train_test_split** function.Now we have training set [X_train,Y_train] and testing set[X_test,Y_test].

4.1 GaussianNB Classifier

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import precision_score,recall_score
clf_nb=GaussianNB()
clf_nb.fit(X_train,Y_train)
pred_nb=clf_nb.predict(X_test)
print("precision score :",precision_score(pred_nb,Y_test))
print("Recall score:",recall_score(pred_nb,Y_test))
```

```
('precision score :', 0.5714285714285714)
('Recall score:', 0.6666666666666663)
```

4.2 K Nearest Neighbour Classifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.grid_search import GridSearchCV
params={"n_neighbors":[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,Y_train)
pred_knn=knn.predict(X_test)
print("precision score:",precision_score(pred_knn,Y_test))
print("Recall score:",recall_score(pred_knn,Y_test))
```

```
('precision score:', 0.14285714285714285)
('Recall score:', 0.5)
```

4.3 Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
params_dt={"criterion":["gini","entropy"],"min_samples_split":[2,3,4,5,6,7,8,10,12,13,15],"min_samples_leaf":[1,2,3,4,5,6]}
dt=DecisionTreeClassifier()
clf_dt=GridSearchCV(dt,params_dt)
clf_dt.fit(X_train,Y_train)
print(clf_dt.best_params_)
pred_dt=clf_dt.predict(X_test)
print("precision score:",precision_score(pred_dt,Y_test))
print("Recall score:",recall_score(pred_dt,Y_test))
```

```
{'min_samples_split': 2, 'criterion': 'gini', 'min_samples_leaf': 2}
('precision score:', 0.5714285714285714)
('Recall score:', 0.8000000000000004)
```

In all three classifier tested and tuned to their best parameter ,DecisionTree classifier seems to have better recall score but when this classifier is tested and cross- validated using sklearn's **StratifiedShuffleSplit** function , this classifier performed poorly as shown below.

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=2,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
        Accuracy: 0.81993       Precision: 0.25124      Recall: 0.17700 F1: 0.20769     F2: 0.18812
        Total predictions: 15000        True positives:  354    False positives: 1055   False negatives: 1646   True negatives:
 11945
```

While evaluating all the classifier using StratifiedShuffleSplit , GaussianNB performed much better as compared to others as shown below.

```
GaussianNB()
        Accuracy: 0.86280       Precision: 0.48046      Recall: 0.35650 F1: 0.40930     F2: 0.37590
        Total predictions: 15000        True positives:  713    False positives:  771   False negatives: 1287   True negatives:
 12229
```

5.Algorithm Tuning

Tuning the parameters of the algorithm is extremely important because it can affect the end-result drastically and often we may get entirely useless result.For example,*min_samples_split* parameter in DecisionTreeClassifier needs to be tuned accordingly otherwise we may overfit the data.Sometimes tuning can give better results in some cases but not all. As was the case with DecisionTreeClassifier ,when cross-validated using **StratifiedShuffleSplit** its performance reduces drastically .I tuned all the algorithm using **GridSearchCV** except the case of K Nearest Neighbors where i did tuning manually.

6.Validation

Validating the classifier is one of the most important part of machine learning. Initially i cross validated my classifier using train_test_split function, which splitted the dataset into training set and testing set where training set can be used to train our model and testing set can be used to evaluate our model performance.Another way provided in tester.py is **StratifiedShuffleSplit** which is a variation of *ShuffleSplit*, which returns stratified splits, *i.e* which creates splits by preserving the same percentage for each target class as in the complete set.With **StratifiedShuffleSplit** cross-validation **GaussianNB Classifier** seemed to give better Recall score of 0.35 (i.e. the proportion of individuals identified as POI,who actually are POI) without compromising much on precision score of 0.48(i.e. The proportion of POI who have successfully been identified).This could be a problem if false positives presented real risks, such as in the medical field, but in case of the Enron investigation false negatives are

arguably more detrimental, given that individuals identified as POIs will not be convicted based on the results of the analysis alone, but rather undergo further inquiry.