UDACITY

PROJECT

Identify Fraud from Enron Email

A part of the Data Analyst Nanodegree Program

PROJECT REVIEW

CODE REVIEW 4

NOTES

poi_id.py 4

```
1 #!/usr/bin/python
```

AWESOME

Code is well structured with functions well commented! 👌

```
2
3 '''
4 Harold Finch: [Opening narration from Season One]
5 You are being watched. The government has a secret system, a machine that spies on you every hour of every day.
6 I know because I built it. I designed the machine to detect acts of terror but it sees everything.
7 Violent crimes involving ordinary people, people like you. Crimes the government considered "irrelevant."
8 They wouldn't act, so I decided I would. But I needed a partner, someone with the skills to intervene.
9 Hunted by the authorities, we work in secret. You'll never find us, but victim or perpetrator, if your number's up... we'll find *you*.
```

AWESOME

😎😎

```
10 '''
11
12 import matplotlib.pyplot as plt
13 import numpy as np
14 import pandas
15 import seaborn as sns
16 import math
17
18 from sklearn.preprocessing import MinMaxScaler
19 from sklearn.cross_validation import train_test_split
20 from sklearn.cross_validation import StratifiedShuffleSplit
21 from sklearn.cross_validation import StratifiedKFold
22 from sklearn.decomposition import PCA
23
24 from sklearn.svm import SVC
25 from sklearn.naive_bayes import GaussianNB
26 from sklearn.tree import DecisionTreeClassifier
27 from sklearn.cluster import KMeans
28 from sklearn.ensemble import AdaBoostClassifier
29
30 from sklearn.pipeline import Pipeline
31 from sklearn.model_selection import GridSearchCV
32 from sklearn.model_selection import RandomizedSearchCV
33 from sklearn.metrics import classification_report
34
35 plt.interactive(False)
36
37
38 sequence_nb = 0
39
```

```
### Global variables to adjust the software behaviour
showunivariate = True
showheatmap = True
showcorrelation = True
performalltunings = True

'''
        ------------
        Global tools
        ------------
'''

def GetSequence():
    # This function return a sequence number
    # It will be used to uniquely identify a seabon chart
    global sequence_nb
    sequence_nb = sequence_nb + 1
    return sequence_nb

def GetValues(data, feature1, feature2=""):
    # This function return one or two features from the project data dictionnary
    # Only lines with values are extracted
    # When two features are mentionned, only the lines with the two valued
    #  (not NaN) features are provided

    values1 = []
    values2 = []
    for item in data:
        if feature2 == "":
            if data[item][feature1] != 'NaN':
                values1.append(data[item][feature1])
        else:
            if data[item][feature1] != 'NaN' and data[item][feature2] != 'NaN':
                values1.append(data[item][feature1])
                values2.append(data[item][feature2])

    if feature2 == "":
        return values1
    else:
        return values1,values2

def ShowHist(data, features):
    # This function display an histogram for all provided features

    for feature in features:
        values = GetValues(data,feature)
        sns.plt.figure(GetSequence())
        sns.distplot(values, axlabel = feature)
        sns.plt.show()

def ShowScatter(data, feature1, feature2):
    # This function display an scatter plot for two given features

    values1,values2 = GetValues(data,feature1,feature2)

    df = pandas.concat([pandas.Series(values1,name=feature1), pandas.Series(values2,name=feature2)], axis=1)
    sns.plt.figure(GetSequence())
    sns.regplot(x=feature1,y=feature2, data=df, label = feature1 + " vs " + feature2)
    sns.plt.show()



def removeOutliers(data,feature,removeLower = True, removeUpper = True):
    # Remove outliers for one specific feature
    # It uses the interquantile method to identify outliers

    values = []
    nbRemoved = 0

    # Extract all values for this features
    values = GetValues(data,feature)
    df = pandas.DataFrame(values)

    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1

    # Now, remove outliers
    for item in data:
        if feature in data[item]:
            if data[item][feature] != 'NaN':
                if removeLower and data[item][feature] < Q1 - 1.5*IQR:
                    del data[item][feature]
                    data[item][feature] = 'NaN'
                    nbRemoved = nbRemoved + 1
            if data[item][feature] != 'NaN':
                if removeUpper and data[item][feature] > Q3 + 1.5*IQR:
                    del data[item][feature]
                    data[item][feature] = 'NaN'
                    nbRemoved = nbRemoved + 1
    print "Outliers, feature ", feature, " - ", nbRemoved, " values removed"


def removeAllOutliers(data,featureList):
    # This function removes outliers of all provided features

    for feature in featureList:
        removeOutliers(data,feature)
```

```
138
139 def RemoveFeature(data,feature):
140     # Remove one specific feature from the data
141
142     for item in data:
143         if feature in data[item]:
144             del data[item][feature]
145
146 def RemovePositiveNegative(data,feature,removeNegative=False,removePositive=False):
147     # Remove one specific feature value if positive and/or negative
148     # by default, it does nothing
149
150     nbRemoved = 0
151     for item in data:
152         if feature in data[item]:
153             if removeNegative and data[item][feature] != 'NaN':
154                 if data[item][feature] < 0:
155                     del data[item][feature]
156                     data[item][feature] = 'NaN'
157                     nbRemoved = nbRemoved + 1
158             if removePositive and data[item][feature] != 'NaN':
159                 if data[item][feature] > 0:
160                     del data[item][feature]
161                     data[item][feature] = 'NaN'
162                     nbRemoved = nbRemoved + 1
163
164     print "RemovePositiveNegative for feature", feature, " : ",nbRemoved," removed values"
165
166 def TransformLog(data,feature,positive=True):
167
168     # This function transform a given feature into its log value
169     for item in data:
170         if feature in data[item]:
171             if data[item][feature] != 'NaN':
172                 value = data[item][feature]
173                 if positive and value >= 0:
174                     del data[item][feature]
175                     data[item][feature] = math.log10(1+value)
176                 else:
177                     del data[item][feature]
178                     data[item][feature] = math.log10(1+abs(value))
179
180 def  MakeLog(data_dict):
181     TransformLog(data_dict,'bonus',positive=True)
182     TransformLog(data_dict,'long_term_incentive',positive=True)
183     TransformLog(data_dict,'deferred_income',positive=False)
184     TransformLog(data_dict,'deferral_payments',positive=True)
185     TransformLog(data_dict,'other',positive=True)
186     TransformLog(data_dict,'expenses',positive=True)
187
188     TransformLog(data_dict,'total_payments',positive=True)
189     TransformLog(data_dict,'exercised_stock_options',positive=True)
190     TransformLog(data_dict,'restricted_stock',positive=True)
191     TransformLog(data_dict,'total_stock_value',positive=True)
192
193     TransformLog(data_dict,'to_messages',positive=True)
194     TransformLog(data_dict,'from_poi_to_this_person',positive=True)
195     TransformLog(data_dict,'from_messages',positive=True)
196     TransformLog(data_dict,'from_this_person_to_poi',positive=True)
197     TransformLog(data_dict,'shared_receipt_with_poi',positive=True)
198
199 def CreateRatio(data):
200     # Create the two new ratios
201
202     for item in data:
203         to_messages = data[item]['to_messages']
204         from_poi_to_this_person  = data[item]['from_poi_to_this_person']
205         from_messages = data[item]['from_messages']
206         from_this_person_to_poi  = data[item]['from_this_person_to_poi']
207
208         if to_messages == 'NaN' or from_poi_to_this_person == 'NaN':
209             data[item]['poi_to_ratio'] = 'NaN'
210         else:
211             data[item]['poi_to_ratio'] = float(from_poi_to_this_person) / float(to_messages)
212
213         if from_messages == 'NaN' or from_this_person_to_poi == 'NaN':
214             data[item]['poi_from_ratio'] = 'NaN'
215         else:
216             data[item]['poi_from_ratio'] = float(from_this_person_to_poi) / float(from_messages)
217
218     print "New Ratio crated"
219
220
221 '''
222         --------------------------------
223         Machine learning algorithm tuning
224         --------------------------------
225 '''
226
227 def test_classifier_optim(clf, dataset, feature_list, folds = 1000):
228     labels, features = targetFeatureSplit(dataset)
229     cv = StratifiedShuffleSplit(labels, folds, random_state = 42)
230     true_negatives = 0
231     false_negatives = 0
232     true_positives = 0
233     false_positives = 0
234     for train_idx, test_idx in cv:
            features_train = []
```

```
        features_test  = []
        labels_train   = []
        labels_test    = []
        for ii in train_idx:
            features_train.append( features[ii] )
            labels_train.append( labels[ii] )
        for jj in test_idx:
            features_test.append( features[jj] )
            labels_test.append( labels[jj] )

        ### fit the classifier using training set, and test on test set
        clf.fit(features_train, labels_train)
        predictions = clf.predict(features_test)
        for prediction, truth in zip(predictions, labels_test):
            if prediction == 0 and truth == 0:
                true_negatives += 1
            elif prediction == 0 and truth == 1:
                false_negatives += 1
            elif prediction == 1 and truth == 0:
                false_positives += 1
            elif prediction == 1 and truth == 1:
                true_positives += 1
            else:
                print "Warning: Found a predicted label not == 0 or 1."
                print "All predictions should take value 0 or 1."
                print "Evaluating performance for processed predictions:"
                break
    try:
        total_predictions = true_negatives + false_negatives + false_positives + true_positives
        accuracy = 1.0*(true_positives + true_negatives)/total_predictions
        precision = 1.0*true_positives/(true_positives+false_positives)
        recall = 1.0*true_positives/(true_positives+false_negatives)
        f1 = 2.0 * true_positives/(2*true_positives + false_positives+false_negatives)
        f2 = (1+2.0*2.0) * precision*recall/(4*precision + recall)
        return f1
    except:
        print "Got a divide by zero when trying out:", clf
        print "Precision or recall may be undefined due to a lack of true positive predicitons."
        return 0


def TuneGNB(data,features_list,max_features,verbose=0,val_strategy='basic'):

    print "Tuning Gaussian Naive Bayes"

    labels, features = targetFeatureSplit(data)
    features_train,features_test,labels_train,labels_test = train_test_split(features,labels,test_size=0.3,random_state=42)
    print "Training test size = ", len(features_train)

    scaler = MinMaxScaler(feature_range=(0, 1))
    pca = PCA(iterated_power='auto', n_components=None, random_state=None,svd_solver='auto', tol=0.0, whiten=False)
    classifier = GaussianNB()

    pipe = Pipeline(steps=[('scaler',scaler),('pca', pca), ('gnb', classifier)])
```

AWESOME

Nice work building your pipeline!

```

    params = dict(
        pca__n_components=range(3,max_features))
    if val_strategy == 'basic':
        estimator = GridSearchCV(pipe,params,scoring='f1',verbose=verbose)
```

AWESOME

Excellent optimization of GridCV!

```
    else:
        #  kfold
        cv = StratifiedKFold(labels_train, 10)
        estimator = GridSearchCV(pipe,params,scoring='f1',verbose=verbose,cv=cv)

    estimator.fit(features_train,labels_train)

    # Return the best number of components and associated score
    clf = estimator.best_estimator_
    f1 = test_classifier_optim(clf,data,features_list)

    print "Best f1 vs best estimator score = ", f1, estimator.best_score_

    print "Estimator parameters"
    print "PCA number of components = " , clf.named_steps['pca'].n_components

    return clf,f1


def TuneSVM(data,features_list,max_features,verbose=0,val_strategy='basic'):

    # This function will try to find the best principal component decomposition
    # I use a Support Vector Machine to assess decomposition performance

```

```
    print "Tuning Support Vector Machine"

    labels, features = targetFeatureSplit(data)
    features_train,features_test,labels_train,labels_test = train_test_split(features,labels,test_size=0.3,random_state=42)
    print "Training test size = ", len(features_train)

    scaler = MinMaxScaler(feature_range=(0, 1))
    pca = PCA(iterated_power='auto', n_components=None, random_state=None,svd_solver='auto', tol=0.0, whiten=False)
    classifier = SVC(kernel='rbf',C=10000)

    pipe = Pipeline(steps=[('scaler',scaler),('pca', pca), ('svm', classifier)])

    params = dict(
        pca__n_components=range(3,max_features),
        svm__C=[1,10,100,1000],
        svm__gamma=[0.01,0.001, 0.0001],
        svm__kernel=['rbf','linear','poly'])
    if val_strategy == 'basic':
        estimator = RandomizedSearchCV(pipe,params,scoring='f1',verbose=verbose,random_state=42)
    else:
        #  kfold
        cv = StratifiedKFold(labels_train, 10)
        estimator = RandomizedSearchCV(pipe,params,scoring='f1',verbose=verbose,cv=cv,random_state=42)

    estimator.fit(features_train,labels_train)

    # Return the best number of components and associated score
    clf = estimator.best_estimator_
    f1 = test_classifier_optim(clf,data,features_list)
    print "Best f1 vs best estimator score = ", f1, estimator.best_score_

    print "Estimator parameters"
    print "PCA number of components = ",clf.named_steps['pca'].n_components
    print "SVM C = ",clf.named_steps['svm'].C
    print "SVM gamma = ",clf.named_steps['svm'].gamma
    print "SVM kernel = ",clf.named_steps['svm'].kernel

    return clf, f1

def TuneDT(data,features_list,max_features,verbose=0,val_strategy='basic'):

    print "Tuning Decision Tree"

    labels, features = targetFeatureSplit(data)
    features_train,features_test,labels_train,labels_test = train_test_split(features,labels,test_size=0.3,random_state=42)
    print "Training test size = ", len(features_train)

    scaler = MinMaxScaler(feature_range=(0, 1))
    pca = PCA(iterated_power='auto', n_components=None, random_state=None,svd_solver='auto', tol=0.0, whiten=False)
    classifier = DecisionTreeClassifier()

    pipe = Pipeline(steps=[('scaler',scaler),('pca', pca), ('dt', classifier)])

    params = dict(
        pca__n_components=range(3,max_features),
        dt__criterion=['gini','entropy'],
        dt__max_features=['sqrt','log2',None])

    if val_strategy == 'basic':
        estimator = RandomizedSearchCV(pipe,params,scoring='f1',verbose=verbose,random_state=42)
    else:
        #  kfold
        cv = StratifiedKFold(labels_train, 10)
        estimator = RandomizedSearchCV(pipe,params,scoring='f1',verbose=verbose,cv=cv,random_state=42)

    estimator.fit(features_train,labels_train)

    # Return the best number of components
    clf = estimator.best_estimator_
    f1 = test_classifier_optim(clf,data,features_list)
    print "Best f1 vs best estimator score = ", f1, estimator.best_score_

    print "Estimator parameters"
    print "PCA number of components = ",clf.named_steps['pca'].n_components
    print "DT criterion = ",clf.named_steps['dt'].criterion
    print "DT max features = ",clf.named_steps['dt'].max_features

    return clf, f1




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

from feature_format import featureFormat, targetFeatureSplit
from tester import dump_classifier_and_data


### Load the dictionary containing the dataset
with open("final_project_dataset.pkl", "r") as data_file:
    data_dict = pickle.load(data_file)

# Task 1: Get a global overview on the data

print "Descriptive statistics of salary"
```

```
df = pandas.DataFrame(featureFormat(data_dict, ['salary'], sort_keys = False,remove_NaN=False),columns=['salary'])
print df.describe()

# Delete TOTAL value and perform summary statistics on all features

print "TOTAL line deletion"
del data_dict['TOTAL']

print "Descriptive statistics on all features"
allFeatures = ['salary', 'bonus', 'long_term_incentive', 'deferred_income', 'deferral_payments', 'loan_advances','other','expenses','dir
df = pandas.DataFrame(featureFormat(data_dict, allFeatures, sort_keys = False,remove_NaN=False),columns=allFeatures)
print df.describe()

# Just let see how many poi and non poi we have.
pois = df['poi']
print "POIs = ",pois.sum()," / Non POIs = ",len(pois) - pois.sum()


# Let's remove non usefull features
RemoveFeature(data_dict,'loan_advances')
RemoveFeature(data_dict,'director_fees')
RemoveFeature(data_dict,'restricted_stock_deferred')

RemovePositiveNegative(data_dict,'deferral_payments',removeNegative=True)
RemovePositiveNegative(data_dict,'restricted_stock',removeNegative=True)
RemovePositiveNegative(data_dict,'total_stock_value',removeNegative=True)

allFeatures = ['salary', 'bonus', 'long_term_incentive', 'deferred_income', 'deferral_payments','other','expenses','total_payments', 'ex
df = pandas.DataFrame(featureFormat(data_dict, allFeatures, sort_keys = False,remove_NaN=False),columns=allFeatures)

if showunivariate:
    ShowHist(data_dict,allFeatures)

# Ok, now we do have out list of features and data dictionnary (it includes POIs, so we need to remove one)
print "Number of features = ", len(allFeatures) -1

print "Correlation between features"
if showheatmap:
    plt.figure(GetSequence())
    sns.heatmap(df.corr(),xticklabels=allFeatures,yticklabels=allFeatures)
    sns.plt.show()
    print df.corr()

if showcorrelation:
    ShowScatter(data_dict,"deferral_payments", "deferred_income")
    ShowScatter(data_dict,"restricted_stock", "total_stock_value")

    ShowScatter(data_dict,"to_messages", "from_this_person_to_poi")
    ShowScatter(data_dict,"to_messages", "shared_receipt_with_poi")

# Let's add a new feature
CreateRatio(data_dict)

# Let's have a look on new ratio/poi possible correlation
df = pandas.DataFrame(featureFormat(data_dict, ['poi_to_ratio','poi_from_ratio','poi'], sort_keys = False,remove_NaN=False),columns=['po

if showheatmap:
    plt.figure(GetSequence())
    sns.heatmap(df.corr(),xticklabels=['poi_to_ratio','poi_from_ratio','poi'],yticklabels=['poi_to_ratio','poi_from_ratio','poi'])
    sns.plt.show()
    print df.corr()




### Let's do the tuning
optim_dataset = ''
optim_log = ''
optim_val_strategy = ''
optim_algo = ''
optim_f1 = 0

if performalltunings:
    for dataset in ['full','full_ratio','limited']:
        for log in ['yes','no']:
            for val_strategy in['basic','kfold']:
                for algo in ['NB','SVM','DT']:
                    if dataset == 'full':
                        allFeatures = ['poi','salary', 'bonus', 'long_term_incentive', 'deferred_income', 'deferral_payments','other','e
                    elif dataset == 'full_ratio':
                        allFeatures = ['poi','salary', 'bonus', 'long_term_incentive', 'deferred_income', 'deferral_payments','other','e
                    elif dataset == 'limited':
                        allFeatures = ['poi','total_payments', 'total_stock_value','to_messages', 'from_poi_to_this_person', 'from_messa

                    data = data_dict

                    if log == 'yes':
                        MakeLog(data)

                    finaldata = featureFormat(data, allFeatures, sort_keys = False,remove_NaN=True)

                    print "Test = dataset / log data / val strategy / algo = " , dataset," / ", log, " / " , val_strategy, " / ",algo
                    if algo == "NB":
                        clf,f1 = TuneGNB(data=finaldata,features_list=allFeatures,max_features=len(allFeatures),val_strategy=val_strateg
                    if algo == "DT":
                        clf,f1 = TuneDT(data=finaldata,features_list=allFeatures,max_features=len(allFeatures),val_strategy=val_strategy
                    if algo == "SVM":
                        clf,f1 = TuneSVM(data=finaldata,features_list=allFeatures,max_features=len(allFeatures),val_strategy=val_strateg
```

```
                    if f1 > optim_f1:
                        optim_f1=f1
                        optim_algo = algo
                        optim_dataset = dataset
                        optim_log = log
                        optim_val_strategy = val_strategy
                        optim_clf = clf

                    print "F1 vs Optim F1",f1,optim_f1


print "Optim f1           = ",optim_f1
print "Optim algo         = ",optim_algo
print "Optim dataset      = ",optim_dataset
print "Optim log          = ",optim_log
print "Optim val strategy = ",optim_val_strategy

data = data_dict
if optim_dataset == 'full':
    allFeatures = ['poi','salary', 'bonus', 'long_term_incentive', 'deferred_income', 'deferral_payments','other','expenses','total_
elif optim_dataset == 'full_ratio':
    allFeatures = ['poi','salary', 'bonus', 'long_term_incentive', 'deferred_income', 'deferral_payments','other','expenses','total_
elif optim_dataset == 'limited':
    allFeatures = ['poi','total_payments', 'total_stock_value','to_messages', 'from_poi_to_this_person', 'from_messages', 'from_this

if optim_log == 'yes':
    MakeLog(data)


dump_classifier_and_data(optim_clf, data, allFeatures)



for rnd in [20,30,40,50,60]:
    labels, features = targetFeatureSplit(featureFormat(data, allFeatures, sort_keys = False,remove_NaN=True))
    features_train,features_test,labels_train,labels_test = train_test_split(features,labels,test_size=0.3,random_state=rnd)

    prediction = optim_clf.predict(features_test)
    print classification_report(labels_test,prediction)
```