UDACITY

PROJECT

Identify Fraud from Enron Email

A part of the Data Analyst Nanodegree Program

PROJECT REVIEW

CODE REVIEW 3

NOTES

poi_id.py 3

```
#!/usr/bin/python
from __future__ import division
from matplotlib import pyplot as plt
from time import time
import numpy as np
import matplotlib
import pandas as pd
import seaborn as sns
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn import svm
from sklearn import neighbors
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import GaussianNB
import collections

from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler
from sklearn.feature_selection import SelectKBest, f_classif

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

from feature_format import featureFormat, targetFeatureSplit
from tester import *

### Task 1: Select what features you'll use.
def load_data():

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

    return data_dict


### Task 2: Remove outliers
def remove_outlier(data_dict):
    data_dict.pop( "TOTAL", 0 )

```

SUGGESTION

I will leave hints on a few other outliers we could remove.

```
45     return data_dict
46
47 ### Task 3: Create new feature(s)
48 ### Store to my_dataset for easy export below.
49 def create_new_features(data_dict, features_list):
50
```

SUGGESTION

I have two suggestions here:

1. Further comment each new method that has been created using PEP 257, and
2. Move the auxiliary methods to a separate file, thus making the main code shorter and easier to read.

```
51     bProc_shared_receipt_with_poi = False
52     bProc_ratio_stock_option = False
53     bProc_salary_bonus_ratio = False
54     for person, item in data_dict.items():
55
56         ## Checking if there's any correlation if the person sends more email to poi vs all the messages to indicate i
57         key = 'ratio_from_poi_fr_msg'
58         if not(key in item) and not(key in features_list):
59             b_ratio_from_poi_fr_msg = True
60             features_list.append(key)
61
62         if b_ratio_from_poi_fr_msg:
63             item[key] = computeFraction(item['from_poi_to_this_person'],item['from_messages'])
64
65         ## Also checking the other way around
66         key = 'ratio_to_poi_to_msg'
67         if not(key in item) and not(key in features_list):
68             b_ratio_to_poi_to_msg = True
69             features_list.append(key)
70
71         if b_ratio_to_poi_to_msg:
72             item[key] = computeFraction(item['from_this_person_to_poi'],item['to_messages'])
73
74         ## and then check if total combined communication with the poi
75         key = 'total_to_poi_from_poi'
76         if not(key in item) and not(key in features_list):
77             b_total_to_poi_from_poi = True
78             features_list.append(key)
79
80         if b_total_to_poi_from_poi:
81             item[key] = computeAddition(item['ratio_to_poi_to_msg'],item['ratio_from_poi_fr_msg'])
82
83         ## Seeing a high correlation with stock and salary hence using both as a new total feature
84         key = 'total_salary_stock'
85         if not(key in item) and not(key in features_list):
86             b_total_salary_stock = True
87             features_list.append(key)
88
89         if (b_total_salary_stock):
90             item[key] = computeAddition(item['salary'], item['total_stock_value'])
91
92
93         ## Create a new feature which combine both bonus and stock value
94         key = 'total_bonus_stock'
95         if not(key in item) and not(key in features_list):
96             b_total_bonus_stock = True
97             features_list.append(key)
98
99         if b_total_bonus_stock:
100            item[key] = computeAddition(item['total_stock_value'], item['bonus'])
101
102
103    return data_dict, features_list
104
105
106
107 ### Task 4: Try a varity of classifiers
108 ### Please name your classifier clf for easy export below.
109 ### Note that if you want to do PCA or other multi-stage operations,
110 ### you'll need to use Pipelines. For more info:
```

```
### http://scikit-learn.org/stable/modules/pipeline.html
def apply_classifiers():
    clf = svm.LinearSVC()
    clf2 = neighbors.KNeighborsClassifier(n_neighbors=4, weights="distance", leaf_size=30, algorithm='brute')
    clf3 = GaussianNB()
    clf4 = RandomForestClassifier(n_estimators=100)

    clfs = collections.OrderedDict()
    clfs['LinearSVC'] = clf
    clfs['KNearestNeighbour'] = clf2
    clfs['NaiveBayes'] = clf3
    clfs['RandomForest'] = clf4

    return clfs


### 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
def tune_classifier(features_train, labels_train):
    # Example starting point. Try investigating other evaluation techniques!
    #features_train, features_test, labels_train, labels_test = \
    #    train_test_split(features, labels, test_size=0.3, random_state=42)

    tuned_clfs = collections.OrderedDict()
    svrs = collections.OrderedDict()

    Cs = [0.01, 0.1, 1, 10, 100, 1000, 10000]
    param_grid = {'C': Cs}
    svr = svm.LinearSVC()
    svrs['Tuned LinearSVC'] = [svr, param_grid]

    n_neighbours = range(1,10)
    weights = ['distance', 'uniform']
    leaf_sizes = range(1,10)
    algorithms = ['kd_tree', 'brute', 'ball_tree']

    param_grid = {'n_neighbors': n_neighbours, 'weights' : weights, 'algorithm':algorithms, 'leaf_size':leaf_sizes}
    svr = neighbors.KNeighborsClassifier()
    svrs['Tuned KNearestNeighbour'] = [svr, param_grid]

    n_estimators = [1,5,10,100]
    max_features = ['sqrt', 'log2']
    min_samples_splits = range(2,10)
    min_samples_leafs = range(1,10)

    param_grid = {'n_estimators': n_estimators, 'max_features':max_features, 'min_samples_split':min_samples_splits, '
    svr = RandomForestClassifier()
    svrs['Tuned RandomForest'] = [svr, param_grid]

    for key, item in svrs.iteritems():
        clf_gs = GridSearchCV(item[0], item[1])
        clf_gs = clf_gs.fit(features_train, labels_train)
        clf_gs.best_params_
        print "Best estimator found by grid search:"
        print clf_gs.best_estimator_, "\n"

        tuned_clfs[key] = clf_gs.best_estimator_

    return tuned_clfs

### 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.
def dump_classifier_results(clf, dataset, features_list):
    dump_classifier_and_data(clf, dataset, features_list)


### Common functions
def plotScatter(data, xlab, ylab, xidx=0, yidx=1):
    fig = plt.figure()
    ax = fig.add_subplot(111)
    #ax.set_xticks(np.arange(0, 2, 1))


    ### your code below
    for point in data:
```

```
        x = point[xidx]
        y = point[yidx]
        if point[0] == 1.0:
            selcolor = 'r'
        else:
            selcolor = 'b'
        plt.scatter( x, y, color=selcolor, alpha=.4 )

    plt.xlabel(xlab)
    plt.ylabel(ylab)
    plt.show()

def show_data_overview(data_dict, features_list):
    print "\ntotal number of data points: ", len(data_dict)
    print "allocation across classes (POI/non-POI): ", sum(x['poi'] for x in data_dict.values()), "/", sum(x['poi']==0
    print "no of features per person: ", len(data_dict[data_dict.keys()[0]])
    print "number of features used: ", len(features_list)
    print "selected features:", ", ".join(features_list)

def select_features(data_dict, filter_pct):
    features_list = []
    print "Are there features with many missing values? etc."
    sorted_nan_dict = {}
    for item in data_dict[data_dict.keys()[0]]:
        sorted_nan_dict[item] = sum(x[item]=="NaN" for x in data_dict.values())
        #print " ", item,": ", sum(x[item]=="NaN" for x in data_dict.values())

    # remove any features that have missing values above the given threshold
    for k,v in sorted(sorted_nan_dict.iteritems(), key=lambda(k,v) : (v,k), reverse = True):
        pct = (v / len(data_dict))
        print " ", k, ": ", v, "Missing data percentage:", round(pct*100,2), "%"
        # filter out email address which is not a numeric value
        if(pct <= filter_pct and k != "email_address"):
            features_list.append(k)

    return features_list

def computeFraction( poi_messages, all_messages ):
    """ given a number messages to/from POI (numerator)
        and number of all messages to/from a person (denominator),
        return the fraction of messages to/from that person
        that are from/to a POI
   """


    ### you fill in this code, so that it returns either
    ###     the fraction of all messages to this person that come from POIs
    ###     or
    ###     the fraction of all messages from this person that are sent to POIs
    ### the same code can be used to compute either quantity

    ### beware of "NaN" when there is no known email address (and so
    ### no filled email features), and integer division!
    ### in case of poi_messages or all_messages having "NaN" value, return 0.
    fraction = 0.
    if (poi_messages != "NaN" and all_messages != "NaN"):
        fraction = poi_messages / all_messages

    return fraction

def computeAddition( var1, var2):
    total = 0
    if var1 != "NaN" and var2 != "NaN":
        total = var1 + var2

    return total

def plotCorrMatrix(seldata, features_list):
    sns.set(style="white")

    d = pd.DataFrame(data=seldata, columns=features_list)
    #print d

    # Compute the correlation matrix
    corr = d.corr()

    # Generate a mask for the upper triangle
    mask = np.zeros_like(corr, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True

    # Set up the matplotlib figure
```

```
    f, ax = plt.subplots(figsize=(11, 9))

    # Generate a custom diverging colormap
    cmap = sns.diverging_palette(220, 10, as_cmap=True)

    # Draw the heatmap with the mask and correct aspect ratio
    sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1., center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5});

    plt.show()

def split_to_label_features(data_dict, features_list):
    ### Extract features and labels from dataset for local testing
    data = featureFormat(data_dict, features_list, sort_keys = True)
    labels, features = targetFeatureSplit(data)

    return labels, features

def autoselect_features(data_dict, features_list):
    labels, features = split_to_label_features(data_dict, features_list)

    # looking for the top 5 features to use
    selector = SelectKBest(f_classif, k = 5)
    selector.fit(features, labels)
    scores = zip(features_list[1:], selector.scores_)
    print 'SelectKBest scores: ', sorted(scores, key=lambda x: x[1], reverse=True)

    #print len(features)
    final_features = selector.transform(features)
    #print len(final_features)
    return final_features

def apply_robust_feature_scaling(data):
    scaler = RobustScaler()
    scaler.fit(data)
    return scaler.transform(data)

def apply_standard_feature_scaling(data):
    scaler = StandardScaler()
    scaler.fit(data)
    return scaler.transform(data)

def apply_minmax_feature_scaling(data):
    scaler = MinMaxScaler()
    scaler.fit(data)
    return scaler.transform(data)


def clf_score_and_evaluate(clf,features_train, labels_train,features_test, labels_test):
    clf.fit(features_train, labels_train)
    pred = clf.predict(features_test)
    sel_avg = 'macro'
    display_precision = 5
    accuracy = round(clf.score(features_test, labels_test),display_precision)
    #print "accuracy: ", round(accuracy,3)

    precision = round(precision_score(labels_test, pred, average=sel_avg), display_precision)
    #print "precision: ", round(precision, 3)

    recall = round(recall_score(labels_test, pred, average=sel_avg), display_precision)
    #print "recall: ", round(recall, 3)

    return accuracy, precision, recall

def compare_algorithms(clfs, features_train, labels_train, features_test, labels_test, sc_features_train, sc_labels_tr

    datamatrix = []
    for name, clf in clfs.iteritems():

        clf.fit(features_train, labels_train)
        accuracy, precision, recall = clf_score_and_evaluate(clf,features_train, labels_train,features_test, labels_te

        #print "\nRescaled features:"
        clf.fit(sc_features_train, sc_labels_train)
        sc_accuracy, sc_precision, sc_recall = clf_score_and_evaluate(clf,sc_features_train, sc_labels_train,sc_featur

        datarow = [name,accuracy, precision, recall,sc_accuracy, sc_precision, sc_recall]
        datamatrix.append(datarow)

    return datamatrix
```

```
353 def test_algorithms(clfs, dataset, features_list):
354     for name, clf in clfs.iteritems():
355         test_classifier(clf, dataset, features_list)
356
357
358 def apply_cross_validation(features, labels, scaled_features):
359     features_train, features_test, labels_train, labels_test = train_test_split(features, labels, test_size=0.3, rando
```

SUGGESTION

I will also leave a suggestion about the validation scheme on the other tab.

```
360
361     sc_features_train, sc_features_test, sc_labels_train, sc_labels_test = train_test_split(scaled_features, labels, t
362
363     return features_train, features_test, labels_train, labels_test, sc_features_train, sc_features_test, sc_labels_tr
```

▸ readMe.rmd

RETURN TO PATH

Rate this review

Student FAQ