labs3_alexis_carbillet

November 18, 2019

In [ ]:
```
# Lab 3 ALEXIS CARBILLET
```

In [ ]:
```
# import librairies
import pandas as pd
import re
from nltk.corpus import wordnet, stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, BaggingClass
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.decomposition import TruncatedSVD
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [ ]:
```
# import data
London = pd.read_csv('LondonTrain.csv')
NY = pd.read_csv('NYTrain.csv')
Singapore = pd.read_csv('SingaporeTrain.csv')
```

In [ ]:
```
print('London shape: ', London.shape)        # (3278, 10)
print('London columns: ',London.columns)
```

In [ ]:
```
print('NY shape: ', NY.shape)                # (2261, 10)
print('NY columns: ',NY.columns)
```

In [ ]:
```
print('Singapore shape: ', Singapore.shape)  # (4702, 10)
print('Singapore columns: ',Singapore.columns)
```

The three datasets have the same columns so they can be merged without columns problem (lack of features...)

Columns: ['row ID', 'educationInfoForAgeGroupEstimation','workInfoForAgeGroupEstimation', 'gender', 'realAge', 'ageGroup', 'relationship', 'educationLevel', 'occupation', 'income']

```
In [ ]: ## preprocess data
        def convert_id(data): # convert hex in int
            for i in range(len(data)):
                data.iloc[i]= int(data.iloc[i], 16)
            return data
```

```
In [ ]: def convert(data):  # problem, everything become -1
            x=data.unique()
            for i in range(len(data)):
                for j in range(len(x)):
                    if data[i]==x[j]:
                        data[i]=j
                if data[i]!=data[i]:
                    data[i]=-1
```

```
In [ ]: def preprocess(dataset):
            dataset['row ID']=convert_id(dataset['row ID'])
            convert(dataset['gender'])
            convert(dataset['relationship'])
            convert(dataset['ageGroup'])
            convert(dataset['income'])
            convert(dataset['educationLevel'])
            convert(dataset['occupation'])
            convert(dataset['realAge'])
            convert(dataset['educationInfoForAgeGroupEstimation'])
            convert(dataset['workInfoForAgeGroupEstimation'])
```

```
In [ ]: ## London preprocess
        preprocess(London)
        London.to_csv('London_modified.csv') # avoid to preprocess again the whole data for ea
```

```
In [ ]: ## NY preprocess
        preprocess(NY)
        NY.to_csv('NY_modified.csv') # avoid to preprocess again the whole data for each test
```

```
In [ ]: ## Singapore preprocess
        preprocess(Singapore)
        Singapore.to_csv('Singapore_modified.csv') # avoid to preprocess again the whole data
```

```
In [ ]: def fit(nb,train,test,y,yt,height,height_f1,type,subject, models):
            nb.fit(train, y)
            models.append(nb)
            w=nb.score(test, yt)
            z=f1_score(yt, nb.predict(test),average='weighted')
            # k = cross_val_score(nb, train, y, cv=10)
            print(subject,': the mean accuracy obtained with ',type,' is:',w)
            print(subject,': the f1 score obtained with ',type,' is:',z)
            # print(subject,': the f1 score obtained with svd and ',type,' is:',k)
            height.append(w)
            height_f1.append(z)
```

```
In [ ]: def ml(train,test,y,yt,subject):
            height=[]
            height_f1=[]
            models=[]
            bars=['perceptron','MLP','tree','logistic regression','kNN 3 neighbors','kNN 7 neig
            # perceptron
            nb = Perceptron(tol=1e-3, random_state=0)
            fit(nb,train,test,y,yt,height,height_f1,'perceptron',subject, models)
            # multi-layer perceptron
            nb = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_st
            fit(nb,train,test,y,yt,height,height_f1,'multi-layer perceptron',subject, models)
            # tree classifier
            nb = DecisionTreeClassifier(random_state=0)
            fit(nb,train,test,y,yt,height,height_f1,'tree',subject, models)
            # logistic regression
            nb = LogisticRegression(random_state=0, solver='lbfgs',multi_class='multinomial')
            fit(nb,train,test,y,yt,height,height_f1,'logistic regression',subject, models)
            # kNN 3
            nb = KNeighborsClassifier(n_neighbors=3)
            fit(nb,train,test,y,yt,height,height_f1,'kNN 3 neighbors',subject, models)
            # kNN 7
            nb = KNeighborsClassifier(n_neighbors=7)
            fit(nb,train,test,y,yt,height,height_f1,'kNN 7 neighbors',subject, models)
            # kNN 15
            nb = KNeighborsClassifier(n_neighbors=15)
            fit(nb,train,test,y,yt,height,height_f1,'kNN 15 neighbors',subject, models)
            # SVC
            nb = SVC(gamma='auto')
            fit(nb,train,test,y,yt,height,height_f1,'SVC',subject, models)
            # random forest
            nb = RandomForestClassifier(n_estimators=100, max_depth=2, random_state=0)
            fit(nb,train,test,y,yt,height,height_f1,'random forest',subject, models)
            # extra trees
            nb = ExtraTreesClassifier(n_estimators=100, max_depth=2, random_state=0)
            fit(nb,train,test,y,yt,height,height_f1,'extra trees',subject, models)
            # bagging
            nb = BaggingClassifier(n_estimators=100, random_state=0)
            fit(nb,train,test,y,yt,height,height_f1,'bagging',subject, models)
            # GaussianNB
            nb = GaussianNB()
            fit(nb,train,test,y,yt,height,height_f1,'gaussian',subject, models)
            # GradientBoosting
            nb = GradientBoostingClassifier(n_estimators=100, random_state=0)
            fit(nb,train,test,y,yt,height,height_f1,' Gradient Boosting',subject, models)
            # LinearDiscriminantAnalysis
            nb = LinearDiscriminantAnalysis()
            fit(nb,train,test,y,yt,height,height_f1,'LinearDiscriminantAnalysis',subject, model
            y_pos = np.arange(len(bars))
```

```
        plt.figure()

        title2='F1 score for '+subject+' prediction'
        plt.title(title2)
        plt.bar(y_pos, height_f1)  # Create bars
        plt.xticks(y_pos, bars, rotation=90) # Create names on the x-axis
        plt.subplots_adjust(bottom=0.3, top=0.95) # Custom the subplot layout
        plt.show()    # Show graphic
        print('the best one for ', subject,' is ',bars[height_f1.index(max(height_f1))],' w
        return models[height_f1.index(max(height_f1))]
```

I decided to predict the genre, but any column can be chosen as labels.

```
In [ ]: ## study for London dataset
        London = pd.read_csv('London_modified.csv')
        labels_london = London['gender']
        London = London.drop(['gender'],axis=1)
        X_train, X_test, y_train, y_test = train_test_split(London, labels_london, test_size=0
        ml(X_train, X_test, y_train, y_test, 'London')
```

```
In [ ]: ## study for NY dataset
        NY = pd.read_csv('NY_modified.csv')
        labels_NY = NY['gender']
        NY = NY.drop(['gender'],axis=1)
        X_train, X_test, y_train, y_test = train_test_split(NY, labels_NY, test_size=0.30, rand
        ml(X_train, X_test, y_train, y_test, 'NY')
```

```
In [ ]: ## study for Singapore dataset
        Singapore = pd.read_csv('Singapore_modified.csv')
        labels_singapore = Singapore['gender']
        Singapore = Singapore.drop(['gender'],axis=1)
        X_train, X_test, y_train, y_test = train_test_split(Singapore, labels_singapore, test_s
        ml(X_train, X_test, y_train, y_test, 'Singapore')
```

```
In [ ]: ## whole study
        def whole(x1,y1,x2,y2,x3,y3):
            X_train1, X_test1, y_train1, y_test1 = train_test_split(x1, y1, test_size=0.30, ran
            X_train2, X_test2, y_train2, y_test2 = train_test_split(x2, y2, test_size=0.30, ran
            X_train3, X_test3, y_train3, y_test3 = train_test_split(x3, y3, test_size=0.30, ran

            nb = ml(X_train1, X_test1, y_train1, y_test1, 'London')
            print(nb)
            p=nb.predict(X_test1)
            nb2 = ml(X_train2, X_test2, y_train2, y_test2, 'NY')
            print(nb2)
            p2=nb2.predict(X_test2)
            nb3 = ml(X_train3, X_test3, y_train3, y_test3, 'Singapore')
            print(nb3)
            p3=nb3.predict(X_test3)
```

```
            X=np.concatenate((p,p2),axis=0)
            X=np.concatenate((X,p3),axis=0)
            X=X.reshape(-1, 1)
            y=np.zeros(p.shape)
            y2=np.ones(p2.shape)
            y3=np.ones(p3.shape)*2
            Y=np.concatenate((y,y2),axis=0)
            Y=np.concatenate((Y,y3),axis=0)
            X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30, random_st
            ml(X_train, X_test, y_train, y_test, 'Fusion')
```

In [ ]:
```
whole(London, labels_london, NY, labels_NY, Singapore, labels_singapore)
```