MLFA_ASSGN_1

January 30, 2022

[ ]:
```
# MLFA Assignment 1

# Team Members -
# Deepansh Agrawal - 19MI10018
# Rohit Ranjan - 20CS30066
# Neha Gupta - 20CH10094
# Gautam Jaju - 20AG30015
# Siddharth Madhupati - 20ME30083
# Madiha Hanifa - 20MF10018
# Himadri Pandya - 20ME10047
```

[1]:
```
# import commands
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statistics
from statistics import mode
from sklearn.neighbors import NearestNeighbors
```

[2]:
```
# installing wget on Colab then downloading dataset
!pip install wget
import wget
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00537/sobar-72.
↪csv'
filename = wget.download(url)
```

```
Requirement already satisfied: wget in /usr/local/lib/python3.7/dist-packages
(3.2)
```

[3]:
```
df = pd.read_csv('sobar-72.csv')
df.shape
```

[3]:
```
(72, 20)
```

[4]:
```
df.head()
```

[4]:
```
   behavior_sexualRisk  behavior_eating  …  empowerment_desires  ca_cervix
0                   10               13  …                    8          1
1                   10               11  …                    4          1
2                   10               15  …                   15          1
3                   10               11  …                    4          1
4                    8               11  …                    7          1

[5 rows x 20 columns]
```

[5]:
```python
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

[6]:
```python
# dividing train and test sets using specified instructions

X_train = X[0:-15]
y_train = y[0:-15]

X_test = X[-15:]
y_test = y[-15:]
```

[7]:
```python
X_train.shape
```

[7]: (57, 19)

[8]:
```python
X_test.shape
```

[8]: (15, 19)

[9]:
```python
y_train.shape
```

[9]: (57,)

[10]:
```python
y_test.shape
```

[10]: (15,)

[11]:
```python
# utility function that returns euclidean distance between any two feature␣
↪vectors
def distance(a,b):
    return np.sqrt(np.sum(np.square(a-b)))
```

[12]:
```python
# utility function that performs k-nn using  the set S
# set S is generally the condensed set of data points with corresponding labels
# x is the feature vector whose label is predicted
# k is the number of nearest neihbours looked at
def compute_label(S,x,k=1):
    min_array = np.ones((len(S),))
```

```
    for i in range(len(S)):
        s = S[i,:-1]
        min_array[i] = distance(s,x)
    list1 = min_array.tolist()
    list2 = S.tolist()
    sorted_list = sorted(zip(list1, list2))
    top_k_labels = [ sorted_list[i][1][-1] for i in range(k)]
    label = mode(top_k_labels)
    return label
```

[13]:
```
# utility function for CNN algorithm
# X is the set of feature vectors for all data points
# y is the set of corresponding ground truth labels
# returns set S which is the minimum consistent subset when starting with first␣
 ↪member as the first data point
def CNN(X,y):
    S = list()
    S.append(np.append(np.array(X.iloc[0,:]),y.iloc[0]).tolist())
    while(True):
        flag=0
        for i in range(len(X)):
            x = np.array(X.iloc[i,:])
            ground_truth = y.iloc[i]
            if(compute_label(np.array(S),x) != ground_truth):
                S.append(np.append(x,y.iloc[i]).tolist())
                flag=1
                break
        if(flag==1):
            continue
        break

    return S
```

[14]:
```
# utility function that computes accuracy using our CNN and our own K-NN␣
 ↪algorithm
# S is the condensed set output from CNN
# X is the dataset to be tested
# y is the corresponding ground truth labels
def accuracy_ours(S,X,y,k=1):
    correct=0
    for i in range(len(X)):
        x = np.array(X.iloc[i,:])
        ground_truth = y.iloc[i]
        if(compute_label(np.array(S),x,k) == ground_truth):
            correct = correct+1
    print(correct/len(X))
```

[15]:
```python
# computing condensed set S using train dataset
S = CNN(X_train,y_train)
```

[16]:
```python
# the number of points in the condensed set is much less than the total train␣
↪set
print("Length of the condensed set: ",len(S))
```

```
Length of the condensed set:  20
```

[17]:
```python
# proof of the fact that S is indeed the minimum consistent subset
accuracy_ours(S,X_train,y_train)
```

```
1.0
```

[18]:
```python
# accuracy of our CNN + KNN (with k = 5) on the test set
accuracy_ours(S,X_test,y_test,5)
```

```
0.6
```

[19]:
```python
# creating an object of the NearestNeighbours class from sklearn package
# fitting it on the entire training dataset
nbrs = NearestNeighbors(n_neighbors=5, algorithm='brute').fit(X_train)

# generating nearest neighbours on the test dataset
distances, indices = nbrs.kneighbors(X_test)
```

[20]:
```python
# utility function that computes accuracy using sklearn's K-NN algorithm (k=5)␣
↪using entire train set
# X is the dataset to be tested
# y is the corresponding ground truth labels
# y_train is the set of ground truth labels for the train set
# nbrs is the NearestNeighbours trained object
def accuracy_sklearn(X,y,y_train,nbrs):
    correct = 0
    distances, indices = nbrs.kneighbors(X)
    for i in range(len(indices)):
        each = indices[i]
        top_k_labels = [y_train[idx] for idx in each]
        pred_label = mode(top_k_labels)
        if pred_label == y.iloc[i]:
            correct = correct+1
    print(correct/len(indices))
```

[21]:
```python
# accuracy using sklearn's K-NN algorithm (k=5) using entire train set for␣
↪predictions
accuracy_sklearn(X_test,y_test,y_train,nbrs)
```

```
0.6666666666666666
```

[22]:
```
# fitting sklearn's KNN on our condensed dataset only
nbrs_2 = NearestNeighbors(n_neighbors=5, algorithm='brute').fit(np.array([S[i][:
 ↪-1] for i in range(len(S))]))

# accuracy using sklearn's K-NN algorithm (k=5) using entire train set for␣
 ↪predictions
accuracy_sklearn(X_test,y_test,np.array([S[i][-1] for i in␣
 ↪range(len(S))]),nbrs_2)
```

```
0.6

/usr/local/lib/python3.7/dist-packages/sklearn/base.py:444: UserWarning: X has
feature names, but NearestNeighbors was fitted without feature names
  f"X has feature names, but {self.__class__.__name__} was fitted without"
```

[23]:
```
# We suspect that the dataset isn't properly shuffled and needs stratifying
# We now shuffle the dataset and then divide into train and test sets

df_shuffled = df.sample(frac=1)
X_shuffled = df_shuffled.iloc[:,:-1]
y_shuffled = df_shuffled.iloc[:,-1]
X_train_shuffled = X_shuffled[0:-15]
y_train_shuffled = y_shuffled[0:-15]
X_test_shuffled = X_shuffled[-15:]
y_test_shuffled = y_shuffled[-15:]
```

[24]:
```
# computing condensed set S using train dataset
S_shuffled = CNN(X_train_shuffled,y_train_shuffled)
# the number of points in the condensed set is much less than the total train␣
 ↪set
print("Length of the condensed set: ",len(S_shuffled))
```

```
Length of the condensed set:  15
```

[25]:
```
# proof of the fact that S is indeed the minimum consistent subset
accuracy_ours(S_shuffled,X_train_shuffled,y_train_shuffled)
```

```
1.0
```

[26]:
```
# accuracy of our CNN + KNN (with k = 5) on the test set
accuracy_ours(S_shuffled,X_test_shuffled,y_test_shuffled,5)
```

```
0.8
```