Assignment 2- Part 2- Application

November 14, 2021

Import the package kNeighborsClassifier.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib as plt
import matplotlib.pyplot as plt
import sklearn as sklearn
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import matplotlib.pyplot as plt
import statsmodels.tools.tools as stattools
from sklearn.naive_bayes import MultinomialNB
from sklearn.cluster import KMeans
from scipy import stats
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import fetch_openml, load_digits
```

[19]:
```
#importing all the right tools/ packages
from __future__ import print_function
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn import datasets
from skimage import exposure
import numpy as np
pip install cv2
import sklearn
```

```
  File "<ipython-input-19-ede5c285a1f9>", line 8
    pip install cv2
        ^
SyntaxError: invalid syntax
```

[21]:
```
pip install --user imutils
imutils.resize
```

```
Requirement already satisfied: imutils in c:\users\jdrex\anaconda3\lib\site-
packages (0.5.4)Note: you may need to restart the kernel to use updated
packages.
```

[4]:
```
#handle older versions of SKLEARN
if int((sklearn.__version__).split(".")[1]) < 18:
    from sklearn.cross_validation import train_test_split
    from sklearn.model_selection import train_test_split
```

[5]:
```
mnist = datasets.load_digits()
```

Be mindful of the train-test split and set the parameters accordingly (justify your choice).

[6]:
```
#code from k-nearest neighbor reference
# take the MNIST data and construct the training and testing split, using 75%␣
 ↪of the
# data for training and 25% for testing
(trainData, testData, trainLabels, testLabels) = train_test_split(np.
 ↪array(mnist.data),
    mnist.target, test_size=0.20, random_state=42)
```

[7]:
```
#code from k-nearest neighbor reference
# now, let's take 10% of the training data and use that for validation
(trainData, valData, trainLabels, valLabels) = train_test_split(trainData,␣
 ↪trainLabels,
        test_size=0.1, random_state=84)

# show the sizes of each data split
print("training data points: {}".format(len(trainLabels)))
print("validation data points: {}".format(len(valLabels)))
print("testing data points: {}".format(len(testLabels)))
```

```
training data points: 1293
validation data points: 144
testing data points: 360
```

[58]:
```
#to validate the 75/ 25 split here, my best assumption is that we need a good␣
 ↪percentage (75) to be able to train with, so that we assure we get all sorts␣
 ↪of examples of different kind of handwriting.
```

Identify the variables in the dataset and define the Euclidean distance between an element in the test set and the training set.

[59]:
```
#code from k-nearest neighbor reference
# initialize the values of k for our k-Nearest Neighbor classifier along with␣
 ↪the
# list of accuracies for each value of k
kVals = range(1, 30, 2)
```

```
accuracies = []
```

[9]:
```
#code from k-nearest neighbor reference
# loop over various values of `k` for the k-Nearest Neighbor classifier
for k in range(1, 30, 2):
        # train the k-Nearest Neighbor classifier with the current value of `k`
        model = KNeighborsClassifier(n_neighbors=k)
        model.fit(trainData, trainLabels)

        # evaluate the model and update the accuracies list
        score = model.score(valData, valLabels)
        print("k=%d, accuracy=%.2f%%" % (k, score * 100))
        accuracies.append(score)

# find the value of k that has the largest accuracy
i = int(np.argmax(accuracies))
print("k=%d achieved highest accuracy of %.2f%% on validation data" % (kVals[i],
        accuracies[i] * 100))
```

```
k=1, accuracy=100.00%
k=3, accuracy=100.00%
k=5, accuracy=100.00%
k=7, accuracy=99.31%
k=9, accuracy=99.31%
k=11, accuracy=99.31%
k=13, accuracy=99.31%
k=15, accuracy=98.61%
k=17, accuracy=97.92%
k=19, accuracy=97.22%
k=21, accuracy=97.22%
k=23, accuracy=96.53%
k=25, accuracy=95.14%
k=27, accuracy=95.83%
k=29, accuracy=95.83%
k=1 achieved highest accuracy of 100.00% on validation data
```

[10]:
```
#code from k-nearest neighbor reference
# re-train our classifier using the best k value and predict the labels of the
# test data
model = KNeighborsClassifier(n_neighbors=kVals[i])
model.fit(trainData, trainLabels)
predictions = model.predict(testData)

# show a final classification report demonstrating the accuracy of the␣
 ↪classifier
# for each of the digits
print("EVALUATION ON TESTING DATA")
```

```
print(classification_report(testLabels, predictions))
```

```
EVALUATION ON TESTING DATA
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       0.93      1.00      0.97        28
           2       1.00      1.00      1.00        33
           3       0.97      1.00      0.99        34
           4       0.98      0.98      0.98        46
           5       0.98      0.98      0.98        47
           6       0.97      1.00      0.99        35
           7       1.00      0.97      0.99        34
           8       1.00      0.93      0.97        30
           9       0.95      0.93      0.94        40

    accuracy                           0.98       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.98      0.98       360
```

[42]:
```
# loop over a few random digits
for i in list(map(int, np.random.randint(0, high=len(testLabels), size=(5,)))):
        # grab the image and classify it
        image = testData[i]
        prediction = model.predict(image.reshape(1, -1))[0]

        # convert the image for a 64-dim array to an 8 x 8 image compatible␣
 ↪with OpenCV,
        # then resize it to 32 x 32 pixels so we can see it better
        image = image.reshape((8, 8)).astype("uint8")
        image = exposure.rescale_intensity(image, out_range=(0, 255))
#         image = imutils.resize(image, width=32, inter=cv2.INTER_CUBIC)

        # show the prediction
        print("I think that digit is: {}".format(prediction))
#         cv2.imshow("Image", image)
#         cv2.waitKey(0)
```

```
I think that digit is: 6
I think that digit is: 0
I think that digit is: 5
I think that digit is: 5
I think that digit is: 9
```

[35]:
```
#above, I was able to import the imutils package, but after quite a bit of␣
 ↪efforts, I was unable to resolve this error, open to any advice on how to␣
 ↪solve this in the future
```

Identify the variables in the dataset and define the Euclidean distance between an element in the test set and the training set.

[56]:
```
#I am not completely sure how to identify the varibles in the data set here,␣
 ↪besides the row labels of each number (1-9)
#and here is my attempt to show the euclidean distance, although seems either I␣
 ↪have an incorrect formula usage here, or inputs are incorrect
#using brownlee reference for forumla
def euclidean_distance(testData,trainData)
        distance = 0.0
        for i in range(len((testData-trainData):
                distance += (testData[i] -trainData[i])**2
        return sqrt(distance)
```

```
  File "<ipython-input-56-b16e59bd6e07>", line 4
    def euclidean_distance(testData,trainData)
                                              ^

SyntaxError: invalid syntax
```

Calculate the distance between the test element and each of if its k nearest neighbors.

Count the occurrence of each digit within the k nearest neighbors and identify the most popular digit.Identify the test element as the digit voted as most popular in the set of the k nearest neighbors.Classify the test element accordingly (i.e. based on the popular vote).

[24]:
```
#this is the digit 5, which had 47 total occurences
```

Calculate the error.

[57]:
```
#I don't have an exact error calculation here, but I do have the precision,␣
 ↪recall ,f1, and support values above.
```

References: k-Nearest Neighbor Classification. Pyimagesearch Customers. Retrived from: https://customers.pyimagesearch.com/lesson-sample-k-nearest-neighbor-classification/

Brownlee, J.February 24, 2020. Develop k-Nearest Neighbors in Python From Scratch. Machine Learning Mastery. Retrieved from: https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/