Southern University of Science and Technology-Department of Computer Science and Engineering

Course: Machine Learning(CS 405)-Professor: Qi Hao

Homework #2

Due date: October, 7th, 2019

In [38]:

```
"""
Import libraries that you might require.

"""

import numpy as np
import math
import matplotlib.pyplot as plt
import operator

from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from prettytable import PrettyTable
```

We will implement the KNN algorithm for the breast cancer dataset. Refer to the pdf and the following functions for the instructions. Complete all the functions as indicated below. The four functions would be autograded as mentioned in the pdf.

In [51]:

```
"""
Task 1: Classification

Please implement KNN for K: 3, 5, and 7 with the following norms:
L1
L2
L-inf
"""

# Read data (Breast Cancer Dataset). Remember to comment out the code not contained in a functio
n.
from sklearn.datasets import load_breast_cancer

breast = load_breast_cancer()

X = breast['data']
y = breast['target']

np.random.seed(100)
p = np.random.permutation(len(X))
X, y = X[p], y[p]

X_train, y_train = X[:400], y[:400]
X_val, y_val = X[400:500], y[400:500]
X_test, y_test = X[500:], y[500:]


def distanceFunc(metric_type, vec1, vec2):
    """
    Computes the distance between two d-dimension vectors.

    Please DO NOT use Numpy's norm function when implementing this function.

    Args:
        metric_type (str): Metric: L1, L2, or L-inf
        vec1 ((d,) np.ndarray): d-dim vector
        vec2 ((d,)) np.ndarray): d-dim vector

    Returns:
        distance (float): distance between the two vectors
    """

    diff = vec1 - vec2
    if metric_type == "L1":
        distance = np.sum(np.abs(diff))  #complete

    if metric_type == "L2":
        square_diff = np.square(diff)
        sum_diff = np.sum(square_diff)
        distance = np.sqrt(sum_diff) #complete

    if metric_type == "L-inf":
        distance = np.max(np.abs(diff)) #complete

    return distance


def computeDistancesNeighbors(K, metric_type, X_train, y_train, sample):
    """
```

```
    Compute the distances between every datapoint in the train_data and the
    given sample. Then, find the k-nearest neighbors.

    Return a numpy array of the label of the k-nearest neighbors.

    Args:
        K (int): K-value
        metric_type (str): metric type
        X_train ((n,p) np.ndarray): Training data with n samples and p features
        y_train : Training labels
        sample ((p,) np.ndarray): Single sample whose distance is to computed with every entry i
n the dataset

    Returns:
        neighbors (list): K-nearest neighbors' labels
    """

    # You will also call the function "distanceFunc" here
    # Complete this function
    neighbors = []
    dists = []

    for i in range(len(X_train)):
        dists.append(distanceFunc(metric_type, X_train[i], sample))

    for i in range(7):
        index = np.argmin(dists)
        neighbors.append(y_train[index])
        dists = np.delete(dists, index)
        y_train = np.delete(y_train, index)

    if K == 3:
        return neighbors[0:2]

    if K == 5:
        return neighbors[0:4]

    if K == 7:
        return neighbors[0:6]


def Majority(neighbors):
    """
    Performs majority voting and returns the predicted value for the test sample.

    Since we're performing binary classification the possible values are [0,1].

    Args:
        neighbors (list): K-nearest neighbors' labels

    Returns:
        predicted_value (int): predicted label for the given sample
    """

    # Performs majority voting
    # Complete this function
    predicted_value = 0
    if(np.sum(neighbors) > len(neighbors)/2):
        predicted_value = 1

    return predicted_value
```

```
def KNN(K, metric_type, X_train, y_train, X_val):
    """
    Returns the predicted values for the entire validation or test set.

    Please DO NOT use Scikit's KNN model when implementing this function.

    Args:
        K (int): K-value
        metric_type (str): metric type
        X_train ((n,p) np.ndarray): Training data with n samples and p features
        y_train : Training labels
        X_val ((n, p) np.ndarray): Validation or test data

    Returns:
        predicted_values (list): output for every entry in validation/test dataset
    """

    # Complete this function
    # Loop through the val_data or the test_data (as required)
    # and compute the output for every entry in that dataset
    # You will also call the function "Majority" here
    predictions = []

    for i in range(len(X_val)):
        neighbors = computeDistancesNeighbors(K, metric_type, X_train, y_train, X_val[i])
        predicted_value = Majority(neighbors)
        predictions.append(predicted_value)

    return predictions


def evaluation(predicted_values, actual_values):
    """
    Computes the accuracy of the given datapoints.

    Args:
        predicted_values ((n,) np.ndarray): Predicted values for n samples
        actual_values ((n,) np.ndarray): Actual values for n samples

    Returns:
        accuracy (float): accuracy
    """

    return accuracy_score(predicted_values, actual_values)


def main():
    """
    Calls the above functions in order to implement the KNN algorithm.

    Test over the following range K = 3,5,7 and all three metrics.
    In total you will have nine combinations to try.

    PRINTS out the accuracies for the nine combinations on the validation set,
    and the accuracy on the test set for the selected K value and appropriate norm.

    REMEMBER: You have to report these values by populating the Table 1.
    """
```

```
## Complete this function

K = [3,5,7]
norm = ["L1", "L2", "L-inf"]

print("<<<<VALIDATION DATA PREDICTIONS>>>>")

## Complete
table1 = PrettyTable(["K", "norm", "accuracy"])
for k in K:
    for metric_type in norm:
        predictions = KNN(k, metric_type, X_train, y_train, X_val)
        accuracy = evaluation(predictions, y_val)
        table1.add_row([k, metric_type, accuracy])
print(table1)

print("<<<<TEST DATA PREDICTIONS>>>>")

## Complete
table2 = PrettyTable(["K", "norm", "accuracy"])
for k in K:
    for metric_type in norm:
        predictions = KNN(k, metric_type, X_train, y_train, X_test)
        accuracy = evaluation(predictions, y_test)
        table2.add_row([k, metric_type, accuracy])
print(table2)
```

Uncomment the code below to run the main function (Remember to recomment the code before submitting).

In [52]:

```
# Finally, call the main function
# main()
main()
```

```
<<<<VALIDATION DATA PREDICTIONS>>>>
+---+-------+----------+
| K |  norm | accuracy |
+---+-------+----------+
| 3 |   L1  |   0.94   |
| 3 |   L2  |   0.93   |
| 3 | L-inf |   0.92   |
| 5 |   L1  |   0.94   |
| 5 |   L2  |   0.94   |
| 5 | L-inf |   0.92   |
| 7 |   L1  |   0.94   |
| 7 |   L2  |   0.94   |
| 7 | L-inf |   0.95   |
+---+-------+----------+
<<<<TEST DATA PREDICTIONS>>>>
+---+-------+--------------------+
| K |  norm |      accuracy      |
+---+-------+--------------------+
| 3 |   L1  | 0.927536231884058  |
| 3 |   L2  | 0.927536231884058  |
| 3 | L-inf | 0.9130434782608695 |
| 5 |   L1  | 0.9130434782608695 |
| 5 |   L2  | 0.8840579710144928 |
| 5 | L-inf | 0.8840579710144928 |
| 7 |   L1  | 0.927536231884058  |
| 7 |   L2  | 0.927536231884058  |
| 7 | L-inf | 0.8985507246376812 |
+---+-------+--------------------+
```

Answer the following questions here:

1. How could having a larger dataset influence the performance of KNN?
2. Tabulate your results from `main()` in the table provided.
3. Finally, mention the best K and the norm combination you have settled upon and report the accuracy on the test set using that combination.

In [ ]:

```
# 1. It will be slower, but maybe have higher accuracy, it depends on the dataset
# 3. The best combination maybe [k, norm] = [3, L1] or [7, L1] or [7, L2]
```