

Lab Assignment 2

(Refer to NumPy tutorial)

PART A: Prerequisite for kNN implementation. (Q1. – Q.12 may help you to implement kNN on your own)

1. Create two vectors using NumPy and check how many values are equal in the two vectors.

Example

$V1 = [1\ 6\ 7\ 9]$

$V2 = [1\ 0\ 6\ 9]$

Here, the output should be 2.

(Use np. sum(V1==V2))

2. Matrix creation using NumPy

- a. Create a matrix M with 10 rows and 3 columns and populate with random values.

Example:

```
[[60 97 34]
 [66 37
 65]
 ...
 [64 64
 44]]
```

- b. Print size of M. ($M.shape$)
- c. Print only the number of rows of M($M.shape[0]$)
- d. Print only the number of columns of M
- e. Write a simple loop to modify the third column as follows: If the sum of the first two columns is divisible by 4, Y should be 1 else, 0.

Example: The above matrix will change as

```
[[60 97
 0]
 [66 37
 0]
 ...
 [64 64
 1]]
```

3. Create pandas dataframe 'df' from the created matrix M and name the columns as X1, X2, and Y. (Refer Lab1)
4. Plot X1 and X2 using scatter plot. Color (X1, X2) red if the corresponding Y is 1 else, blue.

```
col = df.Y.map({0:'b', 1:'r'})      #df is the dataframe you created for Q.3
df.plot.scatter(x='X1', y='X2', c=col)
plt.show()
```

5.
 - a. For two columns X1, X2, find squared error: $(x1 - x2)^2$
(Use *np.square*)

Example: Matrix M will have
 [1369 841 0]

- b. Find the sum of the squared error.
(Use

np.sum)

6. Find Euclidean distance between the first two rows of matrix M.
Compare the result with the inbuilt function [numpy.linalg.norm\(a-b\)](#), where a is the first row and b is the second row.
7. Create a vector V with two random values. Find the Euclidean distance between each row of M with V. Store the distance in a vector and print.
8. Create a matrix A with 10 rows and 2 columns. Add a new column to a matrix. (Use *np.column_stack*). Add a new row to a matrix (Use *np.vstack*)

```
A=np.array([[1,2,3],[2,3,4]])
print(A)
C=np.array([6,7])
A=np.column_stack((A,C))
print(A)
```

```
R=np.array([1,1,1,1])
A=np.vstack((A,R))
print(A)
```

9. Create a matrix M' with two columns X1' and X2' and populate with random values.
Find the Euclidean distance between each row of M' with each row of M. Store the distance in a matrix Dist with 3 columns. The first column is the row id of M, the second column is the row id of M', and the third column is the distance value. Compare the result with the following code

```
from sklearn.metrics.pairwise import euclidean_distances
euclidean_distances(M, M')
```

10. Sort the Dist matrix based on the last column.

Use(`print(a[a[:,n].argsort()])`) where `a` is the matrix and `n` is the column based on which you need to sort.

11. Get the initial `k` rows from the sorted matrix

12. Find the number of 1s and 0s in the `k` rows above. Print 1 if the number of 1s is more else, print 0.

PART B : KNN implementation

- a. Load diabetes dataset as done in Lab 1.
- b. Peek at a few rows as done in Lab 1
- c. Split the dataset into 80% training and 20% testing using numpy slicing.
- d. Use the inbuilt function to do splitting and interpret results

```
from sklearn.model_selection import train_test_split
arr=data.values
X=arr[:,0:8]
Y=arr[:,8]
X_train, X_test, y_train, y_test = train_test_split(X,
Y, test_size=0.20) print( X_test)
```

- e. Do normalisation of training as well as testing dataset using `StandardScaler` as done in Lab 1. Is it required to execute the following code for `X_test`, too?

```
scaler=StandardScaler().fit(X_train)
```

- f. Invoke inbuilt kNN function.

```
from sklearn.neighbors import
KNeighborsClassifier classifier =
KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train) y_pred =
classifier.predict(X_test)
```

Interpret the output obtained.

- g. Evaluate kNN

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Explain the output obtained.

- h. Find the total number of correct predictions.
- i. Repeat f, g, h for different values of `k` in kNN. And plot the graph.

Bonus points: Implement kNN on your own using the above exercises and apply them to the diabetes dataset. Compare the results with python kNN.