**CS599 (Deep Learning)**

**Homework – 4**

1. **Python Code:**

import pandas as pd

import numpy as np

import matplotlib

#matplotlib.use("agg")

data\_set\_dict = {"zip" : ("zip.test.gz", 0),

"spam" : ("spam.data", 57)}

data\_dict = {}

for data\_name, (file\_name, label\_col\_num) in data\_set\_dict.items():

data\_df = pd.read\_csv(file\_name, sep = " ", header = None)

data\_label\_vec = data\_df.iloc[:, label\_col\_num]

is\_01 = data\_label\_vec.isin([0, 1])

data\_01\_df = data\_df.loc[is\_01, :]

is\_label\_col = data\_df.columns == label\_col\_num

data\_features = data\_df.iloc[:, ~is\_label\_col]

data\_labels = data\_df.iloc[:, is\_label\_col]

data\_dict[data\_name] = (data\_features, data\_labels)

#scaling the data

n\_data\_features = data\_features.shape[1]

data\_mean = data\_features.mean().to\_numpy().reshape(1, n\_data\_features)

data\_std = data\_features.std().to\_numpy().reshape(1, n\_data\_features)

data\_scaled = (data\_features - data\_mean)/data\_std

data\_name\_scaled = data\_name + "\_scaled"

data\_scaled = data\_scaled.dropna(axis = "columns")

data\_dict[data\_name\_scaled] = (data\_scaled, data\_labels)

from sklearn.model\_selection import KFold, GridSearchCV

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import LogisticRegressionCV

from sklearn.pipeline import make\_pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

from collections import Counter

class MyKNN:

def \_\_init\_\_(self, n\_neighbors):

"""store n\_neighbors as attribute"""

self.n\_neighbors = n\_neighbors

def fit(self, X, y):

"""store data"""

self.X\_train = X

self.y\_train = y

def decision\_function(self, X):

"""Compute vector of predicted scores.

Larger values mean more likely to be in positive class."""

scores = []

for x in X:

distances = np.sqrt(np.sum((x - self.X\_train) \*\* 2, axis = 1))

n\_neighbor\_indices = np.argsort(distances)[:self.n\_neighbors]

n\_neighbor\_labels = [self.y\_train[i] for i in n\_neighbor\_indices]

most\_common\_label = Counter(n\_neighbor\_labels).most\_common(1)[0][0]

scores.append(most\_common\_label)

return scores

def predict(self, X):

return self.decision\_function(X)

class MyCV:

def \_\_init\_\_(self, estimator, param\_grid, cv):

self.estimator = estimator

self.param\_grid = param\_grid

self.cv = cv

def fit\_one(self, param\_dict, X, y):

self.estimator.\_\_init\_\_(param\_dict)

self.estimator.fit(X, y)

def fit(self, X, y):

validation\_df\_list = []

kf = KFold(n\_splits=self.cv, shuffle=True, random\_state=3)

for validation\_fold, (train\_index, test\_index) in enumerate(kf.split(X)):

train\_data = {"X": X[train\_index], "y": y[train\_index]}

test\_data = {"X": X[test\_index], "y": y[test\_index]}

for param\_dict in self.param\_grid:

self.fit\_one(param\_dict, \*\*train\_data)

y\_pred = self.estimator.predict(test\_data["X"])

accuracy = np.mean(y\_pred == test\_data["y"])

validation\_row = pd.DataFrame({

"validation\_fold": [validation\_fold],

"accuracy": [accuracy],

"param\_value": [param\_dict]

})

validation\_df\_list.append(validation\_row)

validation\_df = pd.concat(validation\_df\_list)

best\_param\_dict = validation\_df.groupby("param\_value")["accuracy"].mean().idxmax()

self.fit\_one(best\_param\_dict, X, y)

def predict(self, X):

return self.estimator.predict(X)

class Featureless:

def fit(self, X\_train, y\_train):

y\_train\_series = pd.Series(y\_train)

self.most\_freq\_labels = y\_train\_series.value\_counts().idxmax()

def predict(self, x\_test):

test\_nrow, test\_ncol = x\_test.shape

return np.repeat(self.most\_freq\_labels, test\_nrow)

accuracy\_data\_frames = []

for data\_name, (data\_features, data\_labels) in data\_dict.items():

kf = KFold(n\_splits=3, shuffle=True, random\_state=3)

enum\_obj = enumerate(kf.split(data\_features))

for fold\_num, (train\_index, test\_index) in enum\_obj:

X\_train, X\_test = np.array(data\_features.iloc[train\_index]), np.array(data\_features.iloc[test\_index])

y\_train, y\_test = np.ravel(data\_labels.iloc[train\_index]), np.ravel(data\_labels.iloc[test\_index])

# K-nearest neighbors

knn = KNeighborsClassifier()

hp\_parameters = {"n\_neighbors": list(range(1, 21))}

grid = GridSearchCV(knn, hp\_parameters, cv=5)

grid.fit(X\_train, y\_train)

best\_n\_neighbors = grid.best\_params\_['n\_neighbors']

print("Best N-Neighbors = ", best\_n\_neighbors)

knn = KNeighborsClassifier(n\_neighbors=best\_n\_neighbors)

knn.fit(X\_train, y\_train)

knn\_pred = knn.predict(X\_test)

#KNN

knn1 = MyKNN(n\_neighbors = 3)

#KNN + gridCV

gridcv = MyCV(estimator = knn1, param\_grid= [n\_neighbors for n\_neighbors in range(1,21)], cv=5)

gridcv.fit(X\_train, y\_train)

knn1\_pred = gridcv.predict(X\_test)

# Logistic Regression

pipe = make\_pipeline(StandardScaler(), LogisticRegressionCV(cv=5, max\_iter=2000))

pipe.fit(X\_train, y\_train)

lr\_pred = pipe.predict(X\_test)

my\_learner\_instance = Featureless()

my\_learner\_instance.fit(X\_train, y\_train)

featureless\_pred = my\_learner\_instance.predict(X\_test)

# store predict data in dict

pred\_dict = {'gridSearch + nearest neighbors': knn\_pred,

'KNN + CV': knn1\_pred,

'linear\_model': lr\_pred,

'featureless': featureless\_pred}

test\_accuracy = {}

for algorithm, predictions in pred\_dict.items():

accuracy = accuracy\_score(y\_test, predictions)

test\_accuracy[algorithm] = accuracy

for algorithm, accuracy in test\_accuracy.items():

print(f"{algorithm} Test Accuracy: {accuracy \* 100}")

accuracy\_df = pd.DataFrame({

"data\_set": [data\_name],

"fold\_id": [fold\_num],

"algorithm": [algorithm],

"accuracy": [test\_accuracy[algorithm]]})

accuracy\_data\_frames.append(accuracy\_df)

print(f"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of {data\_name}({fold\_num})\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

total\_accuracy\_df = pd.concat(accuracy\_data\_frames, ignore\_index=True)

print(total\_accuracy\_df)

import plotnine as p9

gg = p9.ggplot(total\_accuracy\_df, p9.aes(x='accuracy', y='algorithm')) + \

p9.facet\_grid('.~data\_set') + p9.geom\_point()

gg.save("Output.png", height=8, width=12)

1. **Output:**

**>>> for data\_name, (data\_features, data\_labels) in data\_dict.items():**

**... kf = KFold(n\_splits=3, shuffle=True, random\_state=3)**

**... …**

**...**

**... # K-nearest neighbors**

**... knn = KNeighborsClassifier()**

**... hp\_parameters = {"n\_neighbors": list(range(1, 21))}**

**... ...**

**...**

**... for algorithm, accuracy in test\_accuracy.items():**

**... print(f"{algorithm} Test Accuracy: {accuracy \* 100}")**

**... …**

**... print(f"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of {data\_name}({fold\_num})\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")**

Best N-Neighbors = 1

gridSearch + nearest neighbors Test Accuracy: 100.0

KNN + CV Test Accuracy: 100.0

linear\_model Test Accuracy: 99.51923076923077

featureless Test Accuracy: 58.65384615384615

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of zip(0)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 1

gridSearch + nearest neighbors Test Accuracy: 99.51923076923077

KNN + CV Test Accuracy: 99.51923076923077

linear\_model Test Accuracy: 99.03846153846155

featureless Test Accuracy: 57.21153846153846

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of zip(1)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 3

gridSearch + nearest neighbors Test Accuracy: 99.03381642512076

KNN + CV Test Accuracy: 100.0

linear\_model Test Accuracy: 99.03381642512076

featureless Test Accuracy: 57.00483091787439

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of zip(2)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 2

gridSearch + nearest neighbors Test Accuracy: 100.0

KNN + CV Test Accuracy: 100.0

linear\_model Test Accuracy: 99.51923076923077

featureless Test Accuracy: 58.65384615384615

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of zip\_scaled(0)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 1

gridSearch + nearest neighbors Test Accuracy: 99.03846153846155

KNN + CV Test Accuracy: 99.03846153846155

linear\_model Test Accuracy: 99.03846153846155

featureless Test Accuracy: 57.21153846153846

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of zip\_scaled(1)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 1

gridSearch + nearest neighbors Test Accuracy: 99.03381642512076

KNN + CV Test Accuracy: 99.03381642512076

linear\_model Test Accuracy: 99.03381642512076

featureless Test Accuracy: 57.00483091787439

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of zip\_scaled(2)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 3

gridSearch + nearest neighbors Test Accuracy: 79.85658409387223

KNN + CV Test Accuracy: 82.13820078226858

linear\_model Test Accuracy: 91.39504563233378

featureless Test Accuracy: 60.88657105606258

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of spam(0)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 5

gridSearch + nearest neighbors Test Accuracy: 77.90091264667535

KNN + CV Test Accuracy: 79.92177314211213

linear\_model Test Accuracy: 92.63363754889178

featureless Test Accuracy: 60.104302477183836

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of spam(1)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 1

gridSearch + nearest neighbors Test Accuracy: 81.99608610567515

KNN + CV Test Accuracy: 81.60469667318982

linear\_model Test Accuracy: 92.8897586431833

featureless Test Accuracy: 60.79582517938682

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of spam(2)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 5

gridSearch + nearest neighbors Test Accuracy: 90.28683181225554

KNN + CV Test Accuracy: 91.13428943937419

linear\_model Test Accuracy: 91.39504563233378

featureless Test Accuracy: 60.88657105606258

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of spam\_scaled(0)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 6

gridSearch + nearest neighbors Test Accuracy: 89.4393741851369

KNN + CV Test Accuracy: 90.67796610169492

linear\_model Test Accuracy: 92.63363754889178

featureless Test Accuracy: 60.104302477183836

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of spam\_scaled(1)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 5

gridSearch + nearest neighbors Test Accuracy: 90.01956947162427

KNN + CV Test Accuracy: 90.93281148075668

linear\_model Test Accuracy: 92.8897586431833

featureless Test Accuracy: 60.79582517938682

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of spam\_scaled(2)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**>>> total\_accuracy\_df = pd.concat(accuracy\_data\_frames, ignore\_index=True)**

**>>> print(total\_accuracy\_df)**

data\_set fold\_id algorithm accuracy

0 zip 0 gridSearch + nearest neighbors 1.000000

1 zip 0 KNN + CV 1.000000

2 zip 0 linear\_model 0.995192

3 zip 0 featureless 0.586538

4 zip 1 gridSearch + nearest neighbors 0.995192

5 zip 1 KNN + CV 0.995192

6 zip 1 linear\_model 0.990385

7 zip 1 featureless 0.572115

8 zip 2 gridSearch + nearest neighbors 0.990338

9 zip 2 KNN + CV 1.000000

10 zip 2 linear\_model 0.990338

11 zip 2 featureless 0.570048

12 zip\_scaled 0 gridSearch + nearest neighbors 1.000000

13 zip\_scaled 0 KNN + CV 1.000000

14 zip\_scaled 0 linear\_model 0.995192

15 zip\_scaled 0 featureless 0.586538

16 zip\_scaled 1 gridSearch + nearest neighbors 0.990385

17 zip\_scaled 1 KNN + CV 0.990385

18 zip\_scaled 1 linear\_model 0.990385

19 zip\_scaled 1 featureless 0.572115

20 zip\_scaled 2 gridSearch + nearest neighbors 0.990338

21 zip\_scaled 2 KNN + CV 0.990338

22 zip\_scaled 2 linear\_model 0.990338

23 zip\_scaled 2 featureless 0.570048

24 spam 0 gridSearch + nearest neighbors 0.798566

25 spam 0 KNN + CV 0.821382

26 spam 0 linear\_model 0.913950

27 spam 0 featureless 0.608866

28 spam 1 gridSearch + nearest neighbors 0.779009

29 spam 1 KNN + CV 0.799218

30 spam 1 linear\_model 0.926336

31 spam 1 featureless 0.601043

32 spam 2 gridSearch + nearest neighbors 0.819961

33 spam 2 KNN + CV 0.816047

34 spam 2 linear\_model 0.928898

35 spam 2 featureless 0.607958

36 spam\_scaled 0 gridSearch + nearest neighbors 0.902868

37 spam\_scaled 0 KNN + CV 0.911343

38 spam\_scaled 0 linear\_model 0.913950

39 spam\_scaled 0 featureless 0.608866

40 spam\_scaled 1 gridSearch + nearest neighbors 0.894394

41 spam\_scaled 1 KNN + CV 0.906780

42 spam\_scaled 1 linear\_model 0.926336

43 spam\_scaled 1 featureless 0.601043

44 spam\_scaled 2 gridSearch + nearest neighbors 0.900196

45 spam\_scaled 2 KNN + CV 0.909328

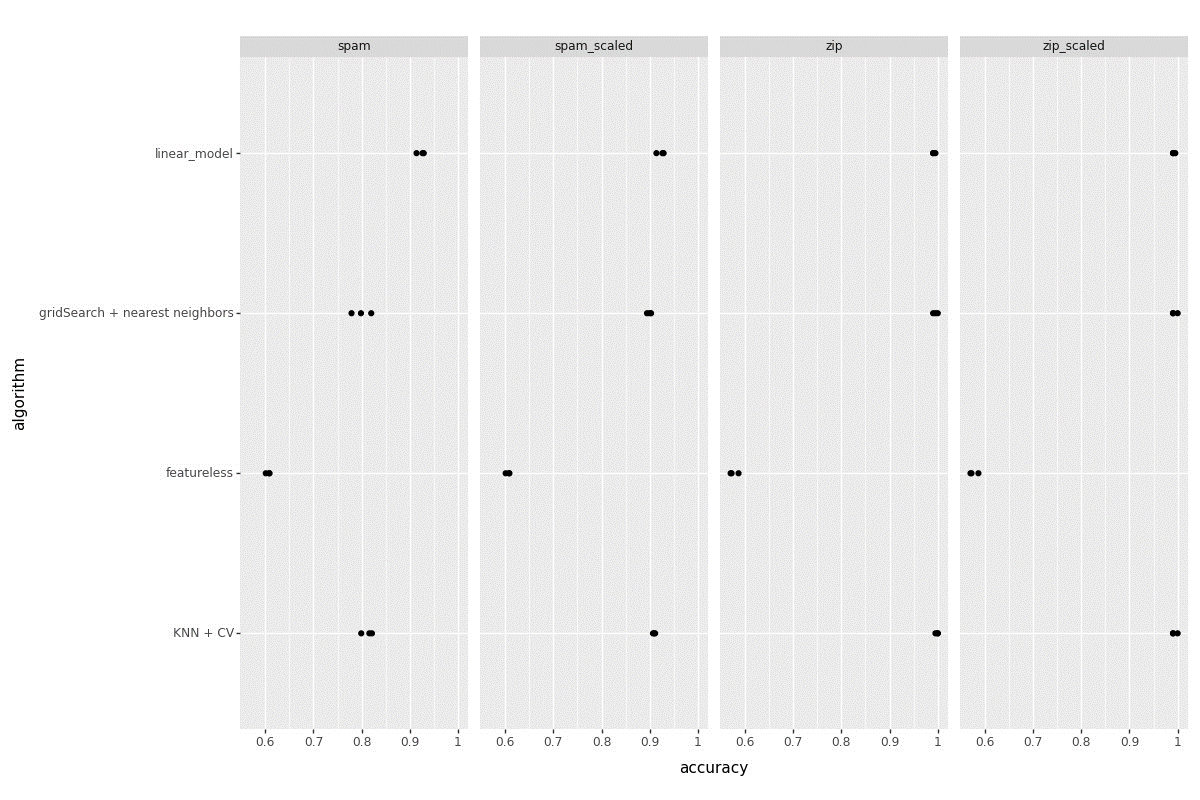
46 spam\_scaled 2 linear\_model 0.928898

47 spam\_scaled 2 featureless 0.607958

**>>> gg = p9.ggplot(total\_accuracy\_df, p9.aes(x='accuracy', y='algorithm')) + \**

**... p9.facet\_grid('.~data\_set') + p9.geom\_point()**

**>>> gg.save("Output.png", height=8, width=10)**



1. **Summary:**

* Similar to the HW3, we need to perform binary classification using KNN.
* Here, we need to scale the both datasets.
* Need to create MyKNN class, MyCV class from scratch.
* Need to plot the graph with all 4 datasets i.e., normal and scaled, with the algorithms MyKNN + MyCV, gridsearch + k-neighbors, linear model and featureless.