**CS599 (Deep Learning)**

**Homework – 5**

1. **Python Code:**

import pandas as pd

import matplotlib

import numpy as np

matplotlib.use("agg")

from sklearn.model\_selection import KFold, GridSearchCV, ParameterGrid

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

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\_01\_df.iloc[:, ~is\_label\_col]

data\_labels = data\_01\_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)

#print(data\_scaled)

data\_dict.pop("zip\_scaled")

data\_dict.pop("spam")

class MyLogReg:

def \_\_init\_\_(self, max\_iterations, step\_size):

self.max\_iterations = max\_iterations

self.step\_size = step\_size

def fit(self, X, y):

self.X\_train = X

self.y\_train = y

data\_nrow, data\_ncol = X.shape

self.intercept\_ = 0.0

self.weight\_vec = np.repeat(0.0, data\_ncol).reshape(data\_ncol,1) # Initialize weight vector with zeros

for i in range(self.max\_iterations):

data\_mat = X

pred\_vec = np.matmul(data\_mat, self.weight\_vec) + self.intercept\_

label\_pos\_neg\_vec = np.where(y == 1, 1, -1).reshape(data\_nrow,1)

grad\_loss\_wrt\_pred = -label\_pos\_neg\_vec / (1 + np.exp(label\_pos\_neg\_vec \* pred\_vec))

loss\_vec = np.log(1 + np.exp(-label\_pos\_neg\_vec \* pred\_vec))

grad\_loss\_wrt\_weight = np.matmul(data\_mat.T, grad\_loss\_wrt\_pred)/data\_nrow

self.weight\_vec -= self.step\_size \* grad\_loss\_wrt\_weight

self.intercept\_ -= self.step\_size \* grad\_loss\_wrt\_pred.mean()

return loss\_vec.mean()

def decision\_function(self, X):

return np.matmul(X, self.weight\_vec).reshape(X.shape[0], 1) + self.intercept\_

def predict(self, X):

scores = self.decision\_function(X)

return np.where(scores > 0, 1, 0)

class MyLogRegCV:

def \_\_init\_\_(self, max\_iterations, step\_size, num\_splits):

self.max\_iterations = max\_iterations

self.step\_size = step\_size

self.num\_splits = num\_splits

def fit(self, X, y):

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

self.scores\_ = pd.DataFrame(columns = ["iteration", "set\_name", "loss\_value"])

best\_loss = float("inf")

best\_lr = None

for i in range(1, self.max\_iterations + 1):

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

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

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

lr = MyLogReg(i, self.step\_size)

subtrain\_loss = lr.fit(subtrain\_data["X"], subtrain\_data["y"])

y\_pred = lr.predict(val\_data["X"])

#subtrain\_loss = lr.loss\_function(subtrain\_data["X"], subtrain\_data["y"])

val\_loss = lr.fit(val\_data["X"], val\_data["y"])

self.scores\_ = pd.concat([self.scores\_, pd.DataFrame({"iteration": [i], "setname": ["subtrain"], "loss\_value": [subtrain\_loss]})], ignore\_index = True, sort = False)

self.scores\_ = pd.concat([self.scores\_, pd.DataFrame({"iteration": [i], "setname": ["validation"], "loss\_value": [val\_loss]})], ignore\_index = True, sort = False)

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

subtrain\_loss\_values = self.scores\_[self.scores\_["setname"] == "subtrain"]["loss\_value"].values

validation\_loss\_values = self.scores\_[self.scores\_["setname"] == "validation"]["loss\_value"].values

#print("Val Loss: ", val\_loss)

#print("Best Loss: ", best\_loss)

if val\_loss < best\_loss:

best\_loss = val\_loss

best\_lr = lr

self.best\_iterations = i

#print(self.best\_iterations)

self.lr = MyLogReg(self.best\_iterations, self.step\_size)

self.lr.fit(X,y)

return subtrain\_loss\_values, validation\_loss\_values

def predict(self, X):

return self.lr.predict(X)

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)

# Logistic Regression

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

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

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

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

#MyLogReg + MyLogRegCV

mylogreg = MyLogRegCV(200, 0.1, 5)

subtrain\_loss, validation\_loss = mylogreg.fit(X\_train, y\_train)

lr\_cv\_pred = mylogreg.predict(X\_test)

most\_frequent\_class = y\_train\_series.value\_counts().idxmax()

print("Most Frequent Class = ", most\_frequent\_class)

# create a featureless baseline

featureless\_pred = np.full\_like(y\_test, most\_frequent\_class)

# store predict data in dict

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

'linear\_model': lr\_pred,

'MyLogRegCV': lr\_cv\_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)

import matplotlib.pyplot as plt

# Number of iterations

iterations = range(1, len(subtrain\_loss) + 1)

# Plot subtrain loss

plt.figure(figsize=(8, 6))

plt.subplot(1, 2, 1)

plt.plot(iterations, subtrain\_loss, label='Subtrain Loss', color='red')

plt.xlabel('Iterations')

plt.ylabel('Loss')

plt.title('Subtrain Loss vs. Iterations')

plt.grid(True)

# Plot validation loss

plt.subplot(1, 2, 2)

plt.plot(iterations, validation\_loss, label='Validation Loss', color='blue')

plt.xlabel('Iterations')

plt.ylabel('Loss')

plt.title('Validation Loss vs. Iterations')

plt.grid(True)

plt.tight\_layout()

plt.savefig("test.png")

1. **Output:**

**>>> 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:**

**... ...**

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

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

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

Best N-Neighbors = 1

Most Frequent Class = 0

gridSearch + nearest neighbors Test Accuracy: 100.0

linear\_model Test Accuracy: 99.51923076923077

MyLogRegCV Test Accuracy: 99.03846153846155

featureless Test Accuracy: 58.65384615384615

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

Best N-Neighbors = 1

Most Frequent Class = 0

gridSearch + nearest neighbors Test Accuracy: 99.51923076923077

linear\_model Test Accuracy: 99.03846153846155

MyLogRegCV Test Accuracy: 98.5576923076923

featureless Test Accuracy: 57.21153846153846

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

Best N-Neighbors = 3

Most Frequent Class = 0

gridSearch + nearest neighbors Test Accuracy: 99.03381642512076

linear\_model Test Accuracy: 99.03381642512076

MyLogRegCV Test Accuracy: 99.03381642512076

featureless Test Accuracy: 57.00483091787439

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

Best N-Neighbors = 5

Most Frequent Class = 0

gridSearch + nearest neighbors Test Accuracy: 90.28683181225554

linear\_model Test Accuracy: 91.39504563233378

MyLogRegCV Test Accuracy: 90.74315514993481

featureless Test Accuracy: 60.88657105606258

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

Best N-Neighbors = 6

Most Frequent Class = 0

gridSearch + nearest neighbors Test Accuracy: 89.4393741851369

linear\_model Test Accuracy: 92.63363754889178

MyLogRegCV Test Accuracy: 90.80834419817471

featureless Test Accuracy: 60.104302477183836

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

Best N-Neighbors = 5

Most Frequent Class = 0

gridSearch + nearest neighbors Test Accuracy: 90.01956947162427

linear\_model Test Accuracy: 92.8897586431833

MyLogRegCV Test Accuracy: 92.10697977821265

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 linear\_model 0.995192

2 zip 0 MyLogRegCV 0.990385

3 zip 0 featureless 0.586538

4 zip 1 gridSearch + nearest neighbors 0.995192

5 zip 1 linear\_model 0.990385

6 zip 1 MyLogRegCV 0.985577

7 zip 1 featureless 0.572115

8 zip 2 gridSearch + nearest neighbors 0.990338

9 zip 2 linear\_model 0.990338

10 zip 2 MyLogRegCV 0.990338

11 zip 2 featureless 0.570048

12 spam\_scaled 0 gridSearch + nearest neighbors 0.902868

13 spam\_scaled 0 linear\_model 0.913950

14 spam\_scaled 0 MyLogRegCV 0.907432

15 spam\_scaled 0 featureless 0.608866

16 spam\_scaled 1 gridSearch + nearest neighbors 0.894394

17 spam\_scaled 1 linear\_model 0.926336

18 spam\_scaled 1 MyLogRegCV 0.908083

19 spam\_scaled 1 featureless 0.601043

20 spam\_scaled 2 gridSearch + nearest neighbors 0.900196

21 spam\_scaled 2 linear\_model 0.928898

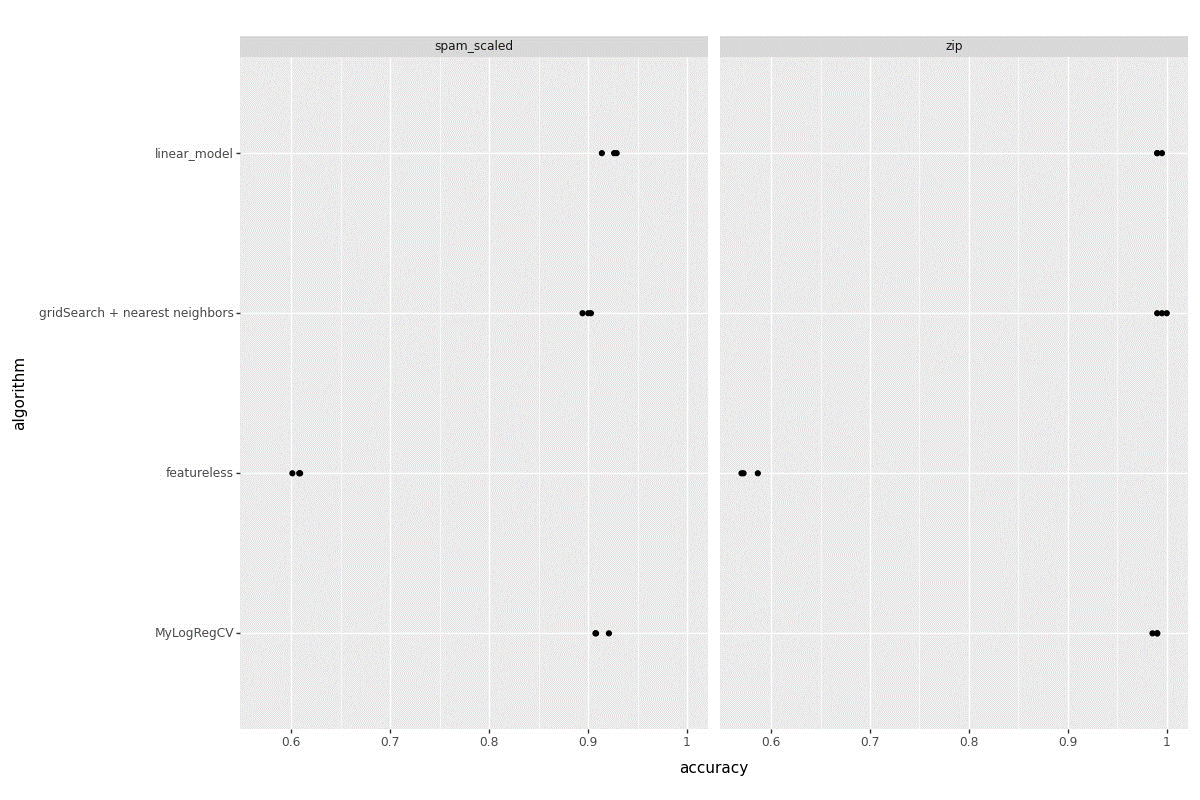
22 spam\_scaled 2 MyLogRegCV 0.921070

23 spam\_scaled 2 featureless 0.607958

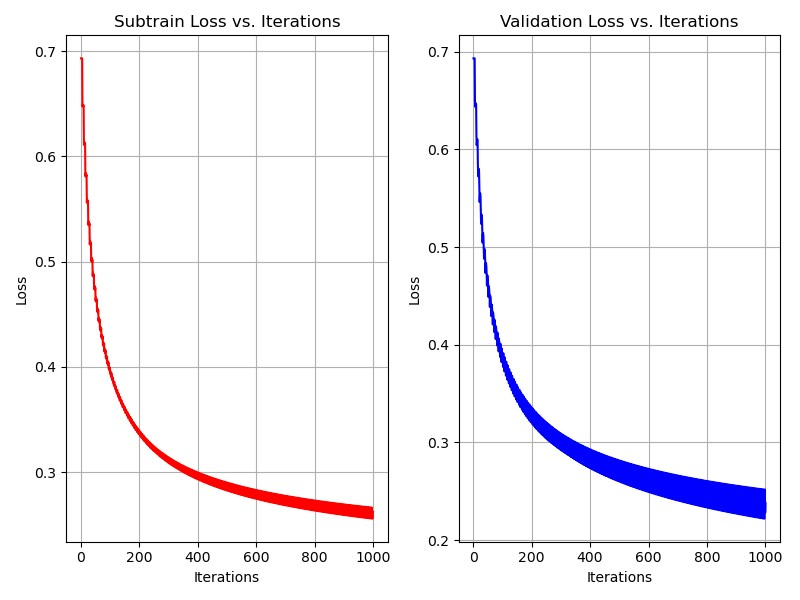
**>>> gg = p9.ggplot(total\_accuracy\_df, p9.aes(x ='accuracy', y = 'algorithm'))+\**

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

**>>> gg.save("output.png", height = 8, width = 12)**



**>>> plt.savefig("test.png")**



1. **Summary:**

* Create the MyLogReg and MyLogRegCV functions with the given requirements.
* Scale the spam dataset since zip dataset is already scaled.
* Need to plot the graph comparing KNeighbors, Linear Model, Featureless and created MyLogRegCV.
* Plot the subtrain loss & validation loss with respect to iterations