**CS599 (Deep Learning)**

**Homework – 10**

1. **Python Code:**

import torch

import pandas as pd

import matplotlib

matplotlib.use("agg")

import numpy as np

import plotnine as p9

import math

import torchvision

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),

}

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\_nrow, data\_ncol = data\_df.shape

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

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]

print("%s %s" %(data\_name, data\_features.shape))

data\_dict[data\_name] = (

torch.from\_numpy(data\_features.to\_numpy()).float(),

torch.from\_numpy(data\_labels.to\_numpy()).flatten()

)

ds = torchvision.datasets.MNIST(

root="c:/Users/Anudeep Kumar/OneDrive/Desktop/Fall 2023/CS599-Deep Learning/Homework/HW10",

download=True,

transform=torchvision.transforms.ToTensor(),

train=False)

dl = torch.utils.data.DataLoader(ds, batch\_size=len(ds), shuffle=False)

for mnist\_features, mnist\_labels in dl:

pass

mnist\_features.flatten(start\_dim=1)

mnist\_labels.numpy()

data\_dict["MNIST"] = (mnist\_features.flatten(start\_dim=1), mnist\_labels)

class TorchModel(torch.nn.Module):

def \_\_init\_\_(self, units\_per\_layer):

super(TorchModel, self).\_\_init\_\_()

seq\_args = []

second\_to\_last = len(units\_per\_layer)-1

for layer\_i in range(second\_to\_last):

next\_i = layer\_i+1

layer\_units = units\_per\_layer[layer\_i]

next\_units = units\_per\_layer[next\_i]

seq\_args.append(torch.nn.Linear(layer\_units, next\_units))

if layer\_i < second\_to\_last-1:

seq\_args.append(torch.nn.ReLU())

self.stack = torch.nn.Sequential(\*seq\_args)

def forward(self, features):

return self.stack(features)

class CSV(torch.utils.data.Dataset):

def \_\_init\_\_(self, features, labels):

self.features = features

self.labels = labels

def \_\_getitem\_\_(self, item):

return self.features[item,:], self.labels[item]

def \_\_len\_\_(self):

return len(self.labels)

class TorchLearner:

def \_\_init\_\_(

self, units\_per\_layer, step\_size=0.1,

batch\_size=20, max\_epochs=100):

self.max\_epochs = max\_epochs

self.batch\_size=batch\_size

self.model = TorchModel(units\_per\_layer)

self.loss\_fun = torch.nn.CrossEntropyLoss()

self.optimizer = torch.optim.SGD(

self.model.parameters(), lr=step\_size)

def fit(self, split\_data\_dict):

ds = CSV(

split\_data\_dict["subtrain"]["X"],

split\_data\_dict["subtrain"]["y"])

dl = torch.utils.data.DataLoader(

ds, batch\_size=self.batch\_size, shuffle=True)

train\_df\_list = []

for epoch\_number in range(self.max\_epochs):

#print(epoch\_number)

for batch\_features, batch\_labels in dl:

self.optimizer.zero\_grad()

loss\_value = self.loss\_fun(

self.model(batch\_features), batch\_labels)

loss\_value.backward()

self.optimizer.step()

for set\_name, set\_data in split\_data\_dict.items():

pred\_vec = self.model(set\_data["X"])

set\_loss\_value = self.loss\_fun(pred\_vec, set\_data["y"])

train\_df\_list.append(pd.DataFrame({

"set\_name":[set\_name],

"loss":float(set\_loss\_value),

"epoch":[epoch\_number]

}))

self.train\_df = pd.concat(train\_df\_list)

def decision\_function(self, test\_features):

with torch.no\_grad():

pred\_vec = self.model(test\_features)

return pred\_vec

def predict(self, test\_features):

pred\_scores = self.decision\_function(test\_features)

\_, predicted = torch.max(pred\_scores, 1)

return predicted.numpy()

class TorchLearnerCV:

def \_\_init\_\_(self, n\_folds, units\_per\_layer):

self.units\_per\_layer = units\_per\_layer

self.n\_folds = n\_folds

def fit(self, train\_features, train\_labels):

train\_nrow, train\_ncol = train\_features.shape

times\_to\_repeat=int(math.ceil(train\_nrow/self.n\_folds))

fold\_id\_vec = np.tile(torch.arange(self.n\_folds), times\_to\_repeat)[:train\_nrow]

np.random.shuffle(fold\_id\_vec)

cv\_data\_list = []

for validation\_fold in range(self.n\_folds):

is\_split = {

"subtrain":fold\_id\_vec != validation\_fold,

"validation":fold\_id\_vec == validation\_fold

}

split\_data\_dict = {}

for set\_name, is\_set in is\_split.items():

set\_y = train\_labels[is\_set]

split\_data\_dict[set\_name] = {

"X":train\_features[is\_set,:],

"y":set\_y}

learner = TorchLearner(self.units\_per\_layer)

learner.fit(split\_data\_dict)

cv\_data\_list.append(learner.train\_df)

self.cv\_data = pd.concat(cv\_data\_list)

self.train\_df = self.cv\_data.groupby(["set\_name","epoch"]).mean().reset\_index()

#print(self.train\_df)

valid\_df = self.train\_df.query("set\_name=='validation'")

#print(valid\_df)

best\_epochs = valid\_df["loss"].argmin()

self.min\_df = valid\_df.query("epoch==%s"%(best\_epochs))

print("Best Epoch: ", best\_epochs)

self.final\_learner = TorchLearner(self.units\_per\_layer, max\_epochs=(best\_epochs + 1))

self.final\_learner.fit({"subtrain":{"X":train\_features,"y":train\_labels}})

return self.cv\_data

def predict(self, test\_features):

return self.final\_learner.predict(test\_features)

accuracy\_data\_frames = []

loss\_data\_dict = {}

min\_df\_dict = {}

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, index\_tup in enum\_obj:

zip\_obj = zip(["train", "test"], index\_tup)

split\_data = {}

for set\_name, set\_indices in zip\_obj:

split\_data[set\_name] = (data\_features, data\_labels)

#x = {data\_name:X.shape for data\_name, (X,y) in split\_data.items()}

#print(f"{data\_name}: ", x)

train\_features, train\_labels = split\_data["train"]

nrow, ncol = train\_features.shape

print(f"{data\_name}: ", nrow, ncol)

test\_features, test\_labels = split\_data["test"]

#kneighbors

knn = KNeighborsClassifier()

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

grid = GridSearchCV(knn, hp\_parameters, cv=3)

grid.fit(train\_features, train\_labels)

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

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

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

knn.fit(train\_features, train\_labels)

knn\_pred = knn.predict(test\_features)

#print(knn\_pred)

#loss = mean\_squared\_error(test\_labels, knn\_pred)

#print(f"Knn Loss {data\_name} : ", loss)

#linear model

pipe = make\_pipeline(StandardScaler(), LogisticRegressionCV(cv=3, max\_iter=2000))

pipe.fit(train\_features, train\_labels)

lr\_pred = pipe.predict(test\_features)

#print(lr\_pred)

#loss\_linear = mean\_squared\_error(test\_labels, lr\_pred)

#print(f"Linear\_loss {data\_name} : ", loss\_linear)

#Featureless

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

#mean\_train\_label = y\_train\_series.mean()

#print("Mean Train Label = ", mean\_train\_label)

# create a featureless baseline

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

print("Most Frequent Label = ", most\_frequent\_label)

featureless\_pred = np.repeat(most\_frequent\_label, len(test\_features))

#featureless\_loss = mean\_squared\_error(test\_labels, featureless\_pred)

#print(f"Featureless Loss {data\_name} : ", featureless\_loss)

#TorchLearnerCV

linear\_learner = TorchLearnerCV(3, [ncol, 10])

#print("ncol:", ncol)

linear\_loss = linear\_learner.fit(train\_features, train\_labels)

ll\_pred = linear\_learner.predict(test\_features)

#print(ll\_pred)

#loss\_torchlinear = mean\_squared\_error(test\_labels, ll\_pred)

#print(f"Torch Linear\_loss {data\_name} : ", loss\_torchlinear)

#TorchLearnerCV + Deep

deep\_learner = TorchLearnerCV(3, [ncol, 100, 10, 10])

deep\_loss = deep\_learner.fit(train\_features, train\_labels)

dl\_pred = deep\_learner.predict(test\_features)

#print(dl\_pred)

#loss\_deeplearner = mean\_squared\_error(test\_labels, dl\_pred)

#print(f"Torch Deep\_loss {data\_name} : ", loss\_deeplearner)

linear\_loss = linear\_loss.groupby(['set\_name', 'epoch']).mean().reset\_index()

deep\_loss = deep\_loss.groupby(['set\_name', 'epoch']).mean().reset\_index()

valid\_df = linear\_loss.query("set\_name=='validation'")

index\_min = valid\_df["loss"].argmin()

min\_df = valid\_df.query("epoch==%s" % index\_min)

valid\_df\_deep = deep\_loss.query("set\_name=='validation'")

index\_min\_deep = valid\_df\_deep["loss"].argmin()

min\_df\_deep = valid\_df\_deep.query("epoch==%s" % index\_min\_deep)

min\_df\_dict[data\_name] = {'min\_df linear': min\_df,

'min\_df deep': min\_df\_deep}

loss\_data\_dict[data\_name] = {'TorchLearnerCV Linear': linear\_loss,

'TorchLearnerCV Deep': deep\_loss}

# store predict data in dict

pred\_dict = {'KNeighborsClassifier + GridSearchCV': knn\_pred,

'LogisticRegressionCV': lr\_pred,

'TorchLearnerCV Linear': ll\_pred,

'TorchLearnerCV Deep': dl\_pred,

'featureless': featureless\_pred}

test\_accuracy = {}

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

#print(f"{algorithm}:", predictions.shape)

#test\_loss = mean\_squared\_error(test\_labels, predictions)

accuracy = accuracy\_score(test\_labels, 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("c:/Users/Anudeep Kumar/OneDrive/Desktop/Fall 2023/CS599-Deep Learning/Homework/HW10/output.png", height = 8, width = 12)

zip\_loss = loss\_data\_dict["zip"]

mnist\_loss = loss\_data\_dict["MNIST"]

zip\_min = min\_df\_dict["zip"]

mnist\_min = min\_df\_dict["MNIST"]

gg1 = p9.ggplot() + p9.geom\_line(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = zip\_loss["TorchLearnerCV Linear"])\

+ p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = zip\_min["min\_df linear"]) + p9.ggtitle("Subtrain/Validation Loss vs Epochs(zip Data - Linear)")

gg1.save("Torch\_validation\_graph1.png", height = 8, width = 12)

gg2 = p9.ggplot() + p9.geom\_line(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = zip\_loss["TorchLearnerCV Deep"])\

+ p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = zip\_min["min\_df deep"]) + p9.ggtitle("Subtrain/Validation Loss vs Epochs(zip Data - Deep)")

gg2.save("Torch\_validation\_graph2.png", height = 8, width = 12)

gg3 = p9.ggplot() + p9.geom\_line(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = mnist\_loss["TorchLearnerCV Linear"])\

+ p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = mnist\_min["min\_df linear"]) + p9.ggtitle("Subtrain/Validation Loss vs Epochs(MNIST Data - Linear)")

gg3.save("Torch\_validation\_graph3.png", height = 8, width = 12)

gg4 = p9.ggplot() + p9.geom\_line(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = mnist\_loss["TorchLearnerCV Deep"])\

+ p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = mnist\_min["min\_df deep"]) + p9.ggtitle("Subtrain/Validation Loss vs Epochs(MNIST Data - Deep)")

gg4.save("Torch\_validation\_graph4.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)**

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

**... for fold\_num, index\_tup in enum\_obj:**

**... zip\_obj = zip(["train", "test"], index\_tup)**

**... split\_data = {}**

**... for set\_name, set\_indices in zip\_obj:**

**... split\_data[set\_name] = (data\_features, data\_labels)**

**... #x = {data\_name:X.shape for data\_name, (X,y) in split\_data.items()}**

**... #print(f"{data\_name}: ", x)**

**... ...**

**...**

**... 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})\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")**

zip: 2007 256

Best N-Neighbors = 1

Most Frequent Label = 0

Best Epoch: 6

Best Epoch: 7

KNeighborsClassifier + GridSearchCV Test Accuracy: 100.0

LogisticRegressionCV Test Accuracy: 97.50871948181366

TorchLearnerCV Linear Test Accuracy: 93.47284504235176

TorchLearnerCV Deep Test Accuracy: 97.45889387144993

featureless Test Accuracy: 17.887394120577977

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

zip: 2007 256

Best N-Neighbors = 1

Most Frequent Label = 0

Best Epoch: 7

Best Epoch: 6

KNeighborsClassifier + GridSearchCV Test Accuracy: 100.0

LogisticRegressionCV Test Accuracy: 97.50871948181366

TorchLearnerCV Linear Test Accuracy: 95.41604384653712

TorchLearnerCV Deep Test Accuracy: 96.81116093672148

featureless Test Accuracy: 17.887394120577977

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

zip: 2007 256

Best N-Neighbors = 1

Most Frequent Label = 0

Best Epoch: 7

Best Epoch: 9

KNeighborsClassifier + GridSearchCV Test Accuracy: 100.0

LogisticRegressionCV Test Accuracy: 97.50871948181366

TorchLearnerCV Linear Test Accuracy: 94.22022919780767

TorchLearnerCV Deep Test Accuracy: 97.85749875435974

featureless Test Accuracy: 17.887394120577977

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

MNIST: 10000 784

Best N-Neighbors = 3

Most Frequent Label = 1

Best Epoch: 18

Best Epoch: 11

KNeighborsClassifier + GridSearchCV Test Accuracy: 97.72999999999999

LogisticRegressionCV Test Accuracy: 94.57

TorchLearnerCV Linear Test Accuracy: 94.55

TorchLearnerCV Deep Test Accuracy: 99.99

featureless Test Accuracy: 11.35

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of MNIST(0)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MNIST: 10000 784

Best N-Neighbors = 3

Most Frequent Label = 1

Best Epoch: 17

Best Epoch: 8

KNeighborsClassifier + GridSearchCV Test Accuracy: 97.72999999999999

LogisticRegressionCV Test Accuracy: 94.57

TorchLearnerCV Linear Test Accuracy: 94.62

TorchLearnerCV Deep Test Accuracy: 99.7

featureless Test Accuracy: 11.35

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of MNIST(1)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MNIST: 10000 784

Best N-Neighbors = 3

Most Frequent Label = 1

Best Epoch: 14

Best Epoch: 8

KNeighborsClassifier + GridSearchCV Test Accuracy: 97.72999999999999

LogisticRegressionCV Test Accuracy: 94.57

TorchLearnerCV Linear Test Accuracy: 94.35

TorchLearnerCV Deep Test Accuracy: 99.47

featureless Test Accuracy: 11.35

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of MNIST(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 KNeighborsClassifier + GridSearchCV 1.000000

1 zip 0 LogisticRegressionCV 0.975087

2 zip 0 TorchLearnerCV Linear 0.934728

3 zip 0 TorchLearnerCV Deep 0.974589

4 zip 0 featureless 0.178874

5 zip 1 KNeighborsClassifier + GridSearchCV 1.000000

6 zip 1 LogisticRegressionCV 0.975087

7 zip 1 TorchLearnerCV Linear 0.954160

8 zip 1 TorchLearnerCV Deep 0.968112

9 zip 1 featureless 0.178874

10 zip 2 KNeighborsClassifier + GridSearchCV 1.000000

11 zip 2 LogisticRegressionCV 0.975087

12 zip 2 TorchLearnerCV Linear 0.942202

13 zip 2 TorchLearnerCV Deep 0.978575

14 zip 2 featureless 0.178874

15 MNIST 0 KNeighborsClassifier + GridSearchCV 0.977300

16 MNIST 0 LogisticRegressionCV 0.945700

17 MNIST 0 TorchLearnerCV Linear 0.945500

18 MNIST 0 TorchLearnerCV Deep 0.999900

19 MNIST 0 featureless 0.113500

20 MNIST 1 KNeighborsClassifier + GridSearchCV 0.977300

21 MNIST 1 LogisticRegressionCV 0.945700

22 MNIST 1 TorchLearnerCV Linear 0.946200

23 MNIST 1 TorchLearnerCV Deep 0.997000

24 MNIST 1 featureless 0.113500

25 MNIST 2 KNeighborsClassifier + GridSearchCV 0.977300

26 MNIST 2 LogisticRegressionCV 0.945700

27 MNIST 2 TorchLearnerCV Linear 0.943500

28 MNIST 2 TorchLearnerCV Deep 0.994700

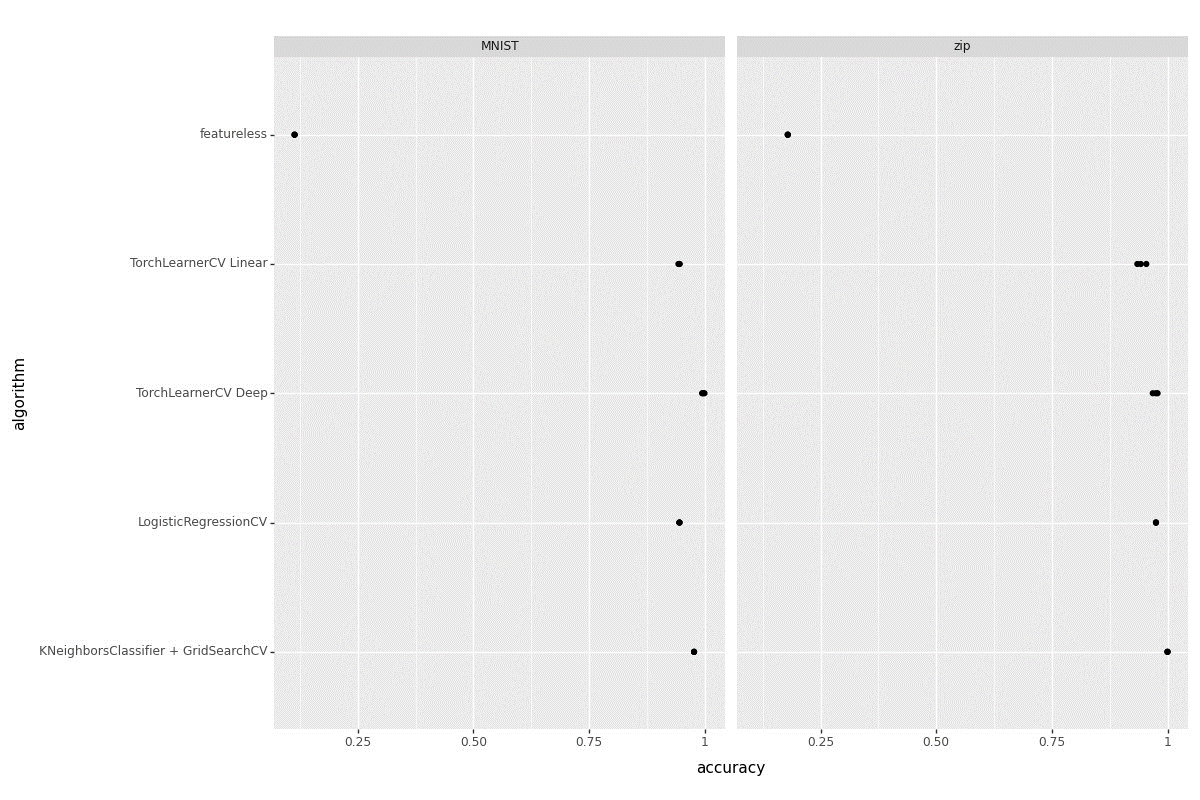
29 MNIST 2 featureless 0.113500

**Test Loss Square Graph:**

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

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

**>>> gg.save("Test\_square\_loss.png", height = 8, width = 12)**

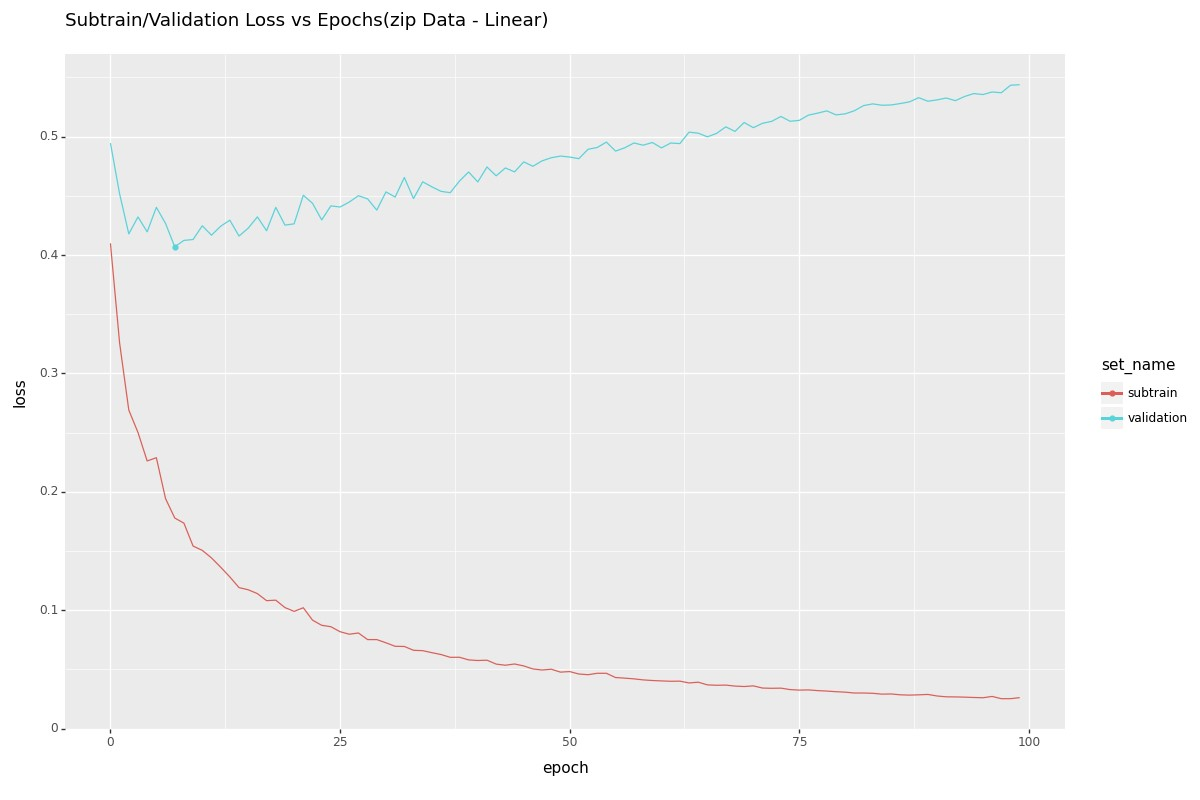


**Linear subtrain/validation loss graph (zip):**

**>>> gg1 = p9.ggplot() + p9.geom\_line(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = zip\_loss["TorchLearnerCV Linear"])\**

**... + p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = zip\_min["min\_df linear"]) + p9.ggtitle("Subtrain/Validation Loss vs Epochs(zip Data - Linear)")**

**>>> gg1.save("Torch\_validation\_graph1.png", height = 8, width = 12)**

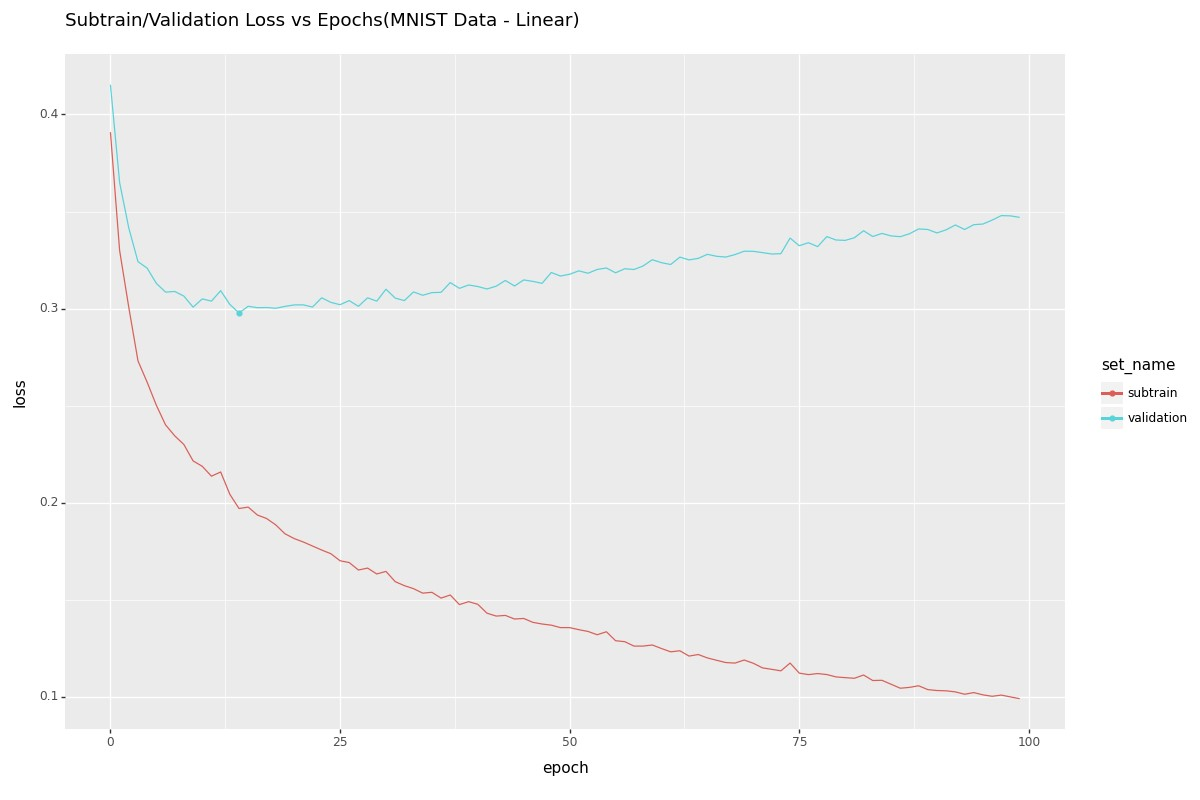


**Linear subtrain/validation loss graph (MNIST):**

**>>> gg3 = p9.ggplot() + p9.geom\_line(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = mnist\_loss["TorchLearnerCV Linear"])\**

**... + p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = mnist\_min["min\_df linear"]) + p9.ggtitle("Subtrain/Validation Loss vs Epochs(MNIST Data - Linear)")**

**>>> gg3.save("Torch\_validation\_graph3.png", height = 8, width = 12)**

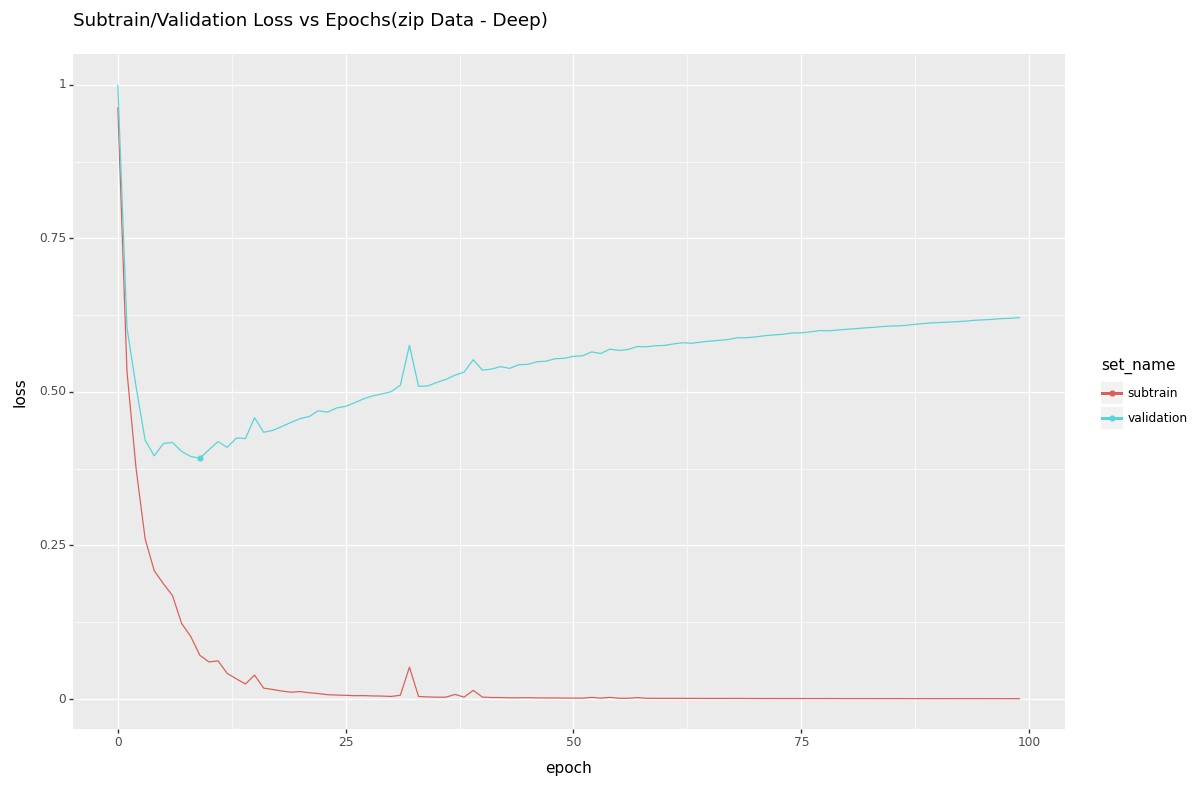


**Deep subtrain/validation loss graph (zip):**

**>>> gg2 = p9.ggplot() + p9.geom\_line(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = zip\_loss["TorchLearnerCV Deep"])\**

**... + p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = zip\_min["min\_df deep"]) + p9.ggtitle("Subtrain/Validation Loss vs Epochs(zip Data - Deep)")**

**>>> gg2.save("Torch\_validation\_graph2.png", height = 8, width = 12)**



**Deep subtrain/validation loss graph (MNIST):**

**>>> gg4 = p9.ggplot() + p9.geom\_line(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = mnist\_loss["TorchLearnerCV Deep"])\**

**... + p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = mnist\_min["min\_df deep"]) + p9.ggtitle("Subtrain/Validation Loss vs Epochs(MNIST Data - Deep)")**

**>>> gg4.save("Torch\_validation\_graph4.png", height = 8, width = 12)**

