**CS599 (Deep Learning)**

**Homework – 07**

1. **Python Code:**

import torch

import pandas as pd

import matplotlib

matplotlib.use("agg")

import numpy as np

import math

import plotnine as p9

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)

spam\_features, spam\_labels = data\_dict.pop("spam")

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

spam\_mean = spam\_features.mean().to\_numpy().reshape(1, spam\_ncol)

spam\_std = spam\_features.std().to\_numpy().reshape(1, spam\_ncol)

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

data\_dict["spam\_scaled"] = (spam\_scaled, spam\_labels)

{data\_name:X.shape for data\_name, (X,y) in data\_dict.items()}

class Node:

def \_\_repr\_\_(self):

return "%s%s"%(self.\_\_class\_\_.\_\_name\_\_, self.value.shape)

class InitialNode(Node):

def \_\_init\_\_(self, value):

self.value = value

def backward(self):

pass

class Operation(Node):

def backward(self):

gradients = self.gradient()

for parent\_node, grad in zip(self.parents, gradients):

if grad is not None and parent\_node.value.shape != grad.shape:

raise ValueError(

"value%s not same shape as grad%s"%(

str(parent\_node.value.shape),

str(grad.shape)))

parent\_node.grad = grad

parent\_node.backward()

class mm(Operation):

def \_\_init\_\_(self, feature\_node, weight\_node):

self.parents = [feature\_node, weight\_node]

self.value = np.matmul(feature\_node.value, weight\_node.value)

def gradient(self):

feature\_node, weight\_node = self.parents

return[

np.matmul(self.grad, weight\_node.value.T),

np.matmul(feature\_node.value.T, self.grad)]

class logistic\_loss(Operation):

def \_\_init\_\_(self, pred\_node, output\_node):

self.parents = [pred\_node, output\_node]

output\_vec = output\_node.value

if not ((output\_vec == 1) | (output\_vec == -1)).all():

raise ValueError("Labels should be only -1 or 1")

self.value = np.log(1 + np.exp(-output\_vec \* pred\_node.value))

def gradient(self):

pred\_node, output\_node = self.parents

# features X is b x p

# weights W is p x u = 1

# pred A is b x u = 1

# where b is batch size

# p is number of input features

# u is number of outputs

# grad\_A(b x u) W(u x p)

pred\_grad = -output\_node.value/(

1 + np.exp(

output\_node.value\*

pred\_node.value

)

)

return [pred\_grad, None]

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 AutoMLP:

def \_\_init\_\_(self, max\_epochs, batch\_size, step\_size, units\_per\_layer):

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

self.max\_epochs = max\_epochs

self.batch\_size = batch\_size

self.step\_size = step\_size

self.weight\_node = InitialNode(

np.repeat(0.0, self.units\_per\_layer[0]).reshape(self.units\_per\_layer[0], 1))

def get\_pred\_node(self, batch\_features):

feature\_node = InitialNode(np.array(batch\_features))

pred\_node = mm(feature\_node, self.weight\_node)

return pred\_node

def take\_step(self, batch\_features, batch\_labels):

label\_node = InitialNode(np.array(batch\_labels))

pred\_vec = self.get\_pred\_node(batch\_features)

loss\_node = logistic\_loss(pred\_vec, label\_node)

loss\_node.backward()

gradient = self.weight\_node.grad

self.weight\_node.value -= gradient \* self.step\_size

return loss\_node.value.mean()

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

ds = CSV(train\_features, test\_features)

dl = torch.utils.data.DataLoader(

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

train\_df\_list = []

for batch\_features, batch\_labels in dl:

loss\_value = self.take\_step(batch\_features, batch\_labels)

def decision\_function(self, X):

pred\_vec = self.get\_pred\_node(X)

return pred\_vec.value.reshape(len(pred\_vec.value),)

def predict(self, X):

pred\_scores = self.decision\_function(X)

return np.where(pred\_scores > 0, 1, 0)

class AutoGradLearnerCV:

def \_\_init\_\_(self, max\_epochs, batch\_size, step\_size, units\_per\_layer, n\_splits):

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

self.max\_epochs = max\_epochs

self.step\_size = step\_size

self.batch\_size = batch\_size

self.n\_splits = n\_splits

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

best\_model = None

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

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

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

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

cv\_data\_list = []

for epoch in range(1, self.max\_epochs + 1):

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

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 = np.where(train\_labels == 1, 1, -1).reshape(train\_nrow, 1)

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

"n": len(set\_y),

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

"y": set\_y[is\_set]}

learner = AutoMLP(self.max\_epochs, self.batch\_size, self.step\_size, self.units\_per\_layer)

learner.fit(split\_data\_dict["subtrain"]["X"], split\_data\_dict["subtrain"]["y"])

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

set\_loss\_value = learner.take\_step(set\_data["X"], set\_data["y"])

cv\_data\_list.append(pd.DataFrame({

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

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

"epoch": [epoch]

}))

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

best\_epoch = self.cv\_data.groupby('epoch')["loss"].mean().idxmin()

best\_learner = AutoMLP(best\_epoch, self.batch\_size, self.step\_size, self.units\_per\_layer)

best\_learner.fit(train\_features, np.where(train\_labels == 1, 1, -1).reshape(train\_nrow, 1))

self.best\_model = best\_learner

return self.cv\_data

def predict(self, test\_features):

return self.best\_model.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.iloc[set\_indices, :].to\_numpy(),

np.ravel(data\_labels.iloc[set\_indices]))

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

nrow, ncol = train\_features.shape

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

#KNN Classifier

knn = KNeighborsClassifier()

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

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

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)

# Logistic Regression

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

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

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

#Featureless

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

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

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

# create a featureless baseline

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

#AutoGradLearnerCV

model\_units = {

"linear": (ncol, 1),

"deep": (ncol, 100, 10, 1)

}

#AutoGradLearnerCV\_linear

linear\_learner = AutoGradLearnerCV(50, 10, 0.015, [ncol, 1], 3)

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

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

#AutoGradLearnerCV\_deep

deep\_learner = AutoGradLearnerCV(50, 10, 0.01, [ncol, 100, 10, 1], 3)

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

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

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 + 1))

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 + 1))

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

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

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

'AutoGradLearnerCV Deep': deep\_loss}

# store predict data in dict

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

'linear\_model': lr\_pred,

'AutoGradLearnerCV Linear': ll\_pred,

'AutoGradLearnerCV Deep': dl\_pred,

'featureless': featureless\_pred}

test\_accuracy = {}

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

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

accuracy = np.mean(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)

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

spam\_loss = loss\_data\_dict["spam\_scaled"]

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

spam\_min = min\_df\_dict["spam\_scaled"]

gg = p9.ggplot() +\

p9.geom\_line(

p9.aes(

x = "epoch",

y= "loss",

color = "set\_name"

),

data = zip\_loss["AutoGradLearnerCV 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 = p9.ggplot() +\

p9.geom\_line(

p9.aes(

x = "epoch",

y= "loss",

color = "set\_name"

),

data = zip\_loss["AutoGradLearnerCV 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 = p9.ggplot() +\

p9.geom\_line(

p9.aes(

x = "epoch",

y= "loss",

color = "set\_name"

),

data = spam\_loss["AutoGradLearnerCV Linear"]) +\

p9.geom\_point(

p9.aes(

x = "epoch",

y = "loss",

color = "set\_name"

),

data = spam\_min["min\_df linear"]) +\

p9.ggtitle("Subtrain/Validation Loss vs Epochs(Spam\_scaled Data - Linear)")

gg3 = p9.ggplot() +\

p9.geom\_line(

p9.aes(

x = "epoch",

y= "loss",

color = "set\_name"

),

data = spam\_loss["AutoGradLearnerCV Deep"]) +\

p9.geom\_point(

p9.aes(

x = "epoch",

y = "loss",

color = "set\_name"

),

data = spam\_min["min\_df deep"]) +\

p9.ggtitle("Subtrain/Validation Loss vs Epochs(Spam\_scaled Data - Deep)")

gg4 = p9.ggplot(total\_accuracy\_df, p9.aes(x ='accuracy', y = 'algorithm'))+\

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

gg.save("Zip\_linear\_SV\_graph.png", height = 8, width = 12)

gg1.save("Zip\_deep\_SV\_graph.png", height = 8, width = 12)

gg2.save("Spam\_linear\_SV\_graph.png", height = 8, width = 12)

gg3.save("Spam\_deep\_SV\_graph.png", height = 8, width = 12)

gg4.save("Accuracy\_graph.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.iloc[set\_indices, :].to\_numpy(),**

**... np.ravel(data\_labels.iloc[set\_indices]))**

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

**... ...**

**...**

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

Best N-Neighbors = 1

Most Frequent Class = 0

gridSearch + nearest neighbors Test Accuracy: 100.0

linear\_model Test Accuracy: 99.51923076923077

AutoGradLearnerCV Linear Test Accuracy: 98.07692307692307

AutoGradLearnerCV Deep Test Accuracy: 99.51923076923077

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

AutoGradLearnerCV Linear Test Accuracy: 98.5576923076923

AutoGradLearnerCV Deep 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

AutoGradLearnerCV Linear Test Accuracy: 98.55072463768117

AutoGradLearnerCV Deep Test Accuracy: 98.55072463768117

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: 91.52542372881356

AutoGradLearnerCV Linear Test Accuracy: 91.13428943937419

AutoGradLearnerCV Deep Test Accuracy: 91.59061277705347

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: 91.78617992177314

AutoGradLearnerCV Linear Test Accuracy: 91.26466753585397

AutoGradLearnerCV Deep Test Accuracy: 91.39504563233378

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: 92.49836921069797

AutoGradLearnerCV Linear Test Accuracy: 92.4331376386171

AutoGradLearnerCV Deep Test Accuracy: 91.71559034572732

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 AutoGradLearnerCV Linear 0.980769

3 zip 0 AutoGradLearnerCV Deep 0.995192

4 zip 0 featureless 0.586538

5 zip 1 gridSearch + nearest neighbors 0.995192

6 zip 1 linear\_model 0.990385

7 zip 1 AutoGradLearnerCV Linear 0.985577

8 zip 1 AutoGradLearnerCV Deep 0.985577

9 zip 1 featureless 0.572115

10 zip 2 gridSearch + nearest neighbors 0.990338

11 zip 2 linear\_model 0.990338

12 zip 2 AutoGradLearnerCV Linear 0.985507

13 zip 2 AutoGradLearnerCV Deep 0.985507

14 zip 2 featureless 0.570048

15 spam\_scaled 0 gridSearch + nearest neighbors 0.902868

16 spam\_scaled 0 linear\_model 0.915254

17 spam\_scaled 0 AutoGradLearnerCV Linear 0.911343

18 spam\_scaled 0 AutoGradLearnerCV Deep 0.915906

19 spam\_scaled 0 featureless 0.608866

20 spam\_scaled 1 gridSearch + nearest neighbors 0.894394

21 spam\_scaled 1 linear\_model 0.917862

22 spam\_scaled 1 AutoGradLearnerCV Linear 0.912647

23 spam\_scaled 1 AutoGradLearnerCV Deep 0.913950

24 spam\_scaled 1 featureless 0.601043

25 spam\_scaled 2 gridSearch + nearest neighbors 0.900196

26 spam\_scaled 2 linear\_model 0.924984

27 spam\_scaled 2 AutoGradLearnerCV Linear 0.924331

28 spam\_scaled 2 AutoGradLearnerCV Deep 0.917156

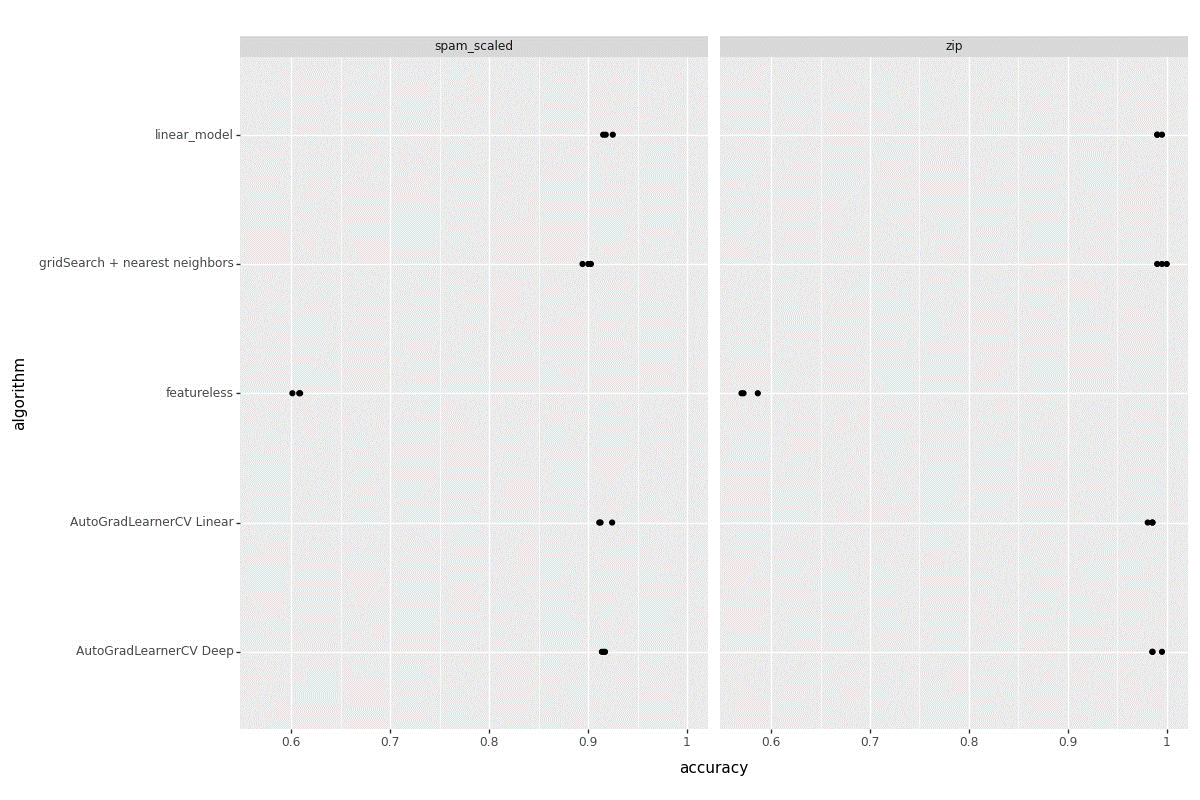
29 spam\_scaled 2 featureless 0.607958

**Accuracy Graph:**

**>>> gg4 = p9.ggplot(total\_accuracy\_df, p9.aes(x ='accuracy', y = 'algorithm'))+\**

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

**>>> gg4.save("Accuracy\_graph.png", height = 8, width = 12)**



**Linear subtrain/validation Loss graph (Zip):**

**>>> gg = p9.ggplot() +\**

**... p9.geom\_line(**

**... p9.aes(**

**... x = "epoch",**

**... y= "loss",**

**... color = "set\_name"**

**... ),**

**... data = zip\_loss["AutoGradLearnerCV Linear"]) +\**

**... p9.geom\_point(**

**... p9.aes(**

**... x = "epoch",**

**... y = "loss",**

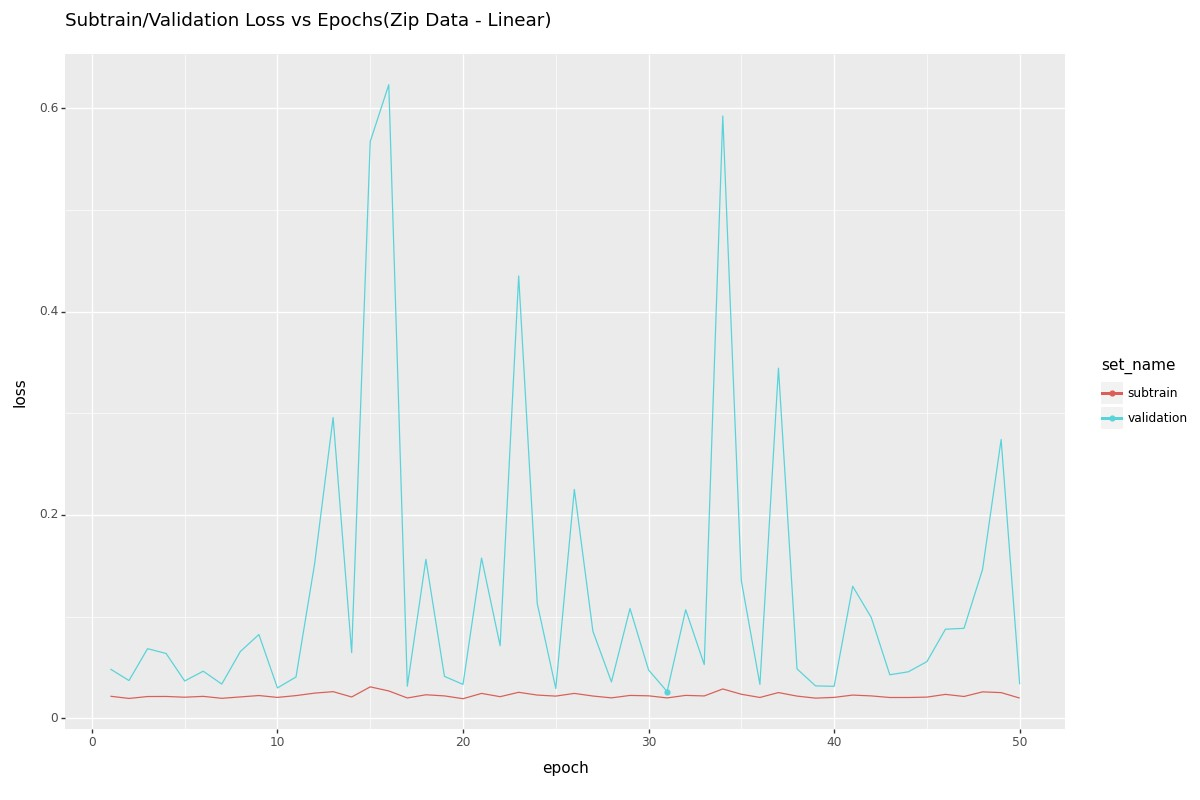
**... color = "set\_name"**

**... ),**

**... data = zip\_min["min\_df linear"]) +\**

**... p9.ggtitle("Subtrain/Validation Loss vs Epochs(Zip Data - Linear)")**

**>>> gg.save("Zip\_linear\_SV\_graph.png", height = 8, width = 12)**



**Subtrain/Validation Loss Graph (Zip Data – Deep Model):**

**>>> gg1 = p9.ggplot() +\**

**... p9.geom\_line(**

**... p9.aes(**

**... x = "epoch",**

**... y= "loss",**

**... color = "set\_name"**

**... ),**

**... data = zip\_loss["AutoGradLearnerCV Deep"]) +\**

**... p9.geom\_point(**

**... p9.aes(**

**... x = "epoch",**

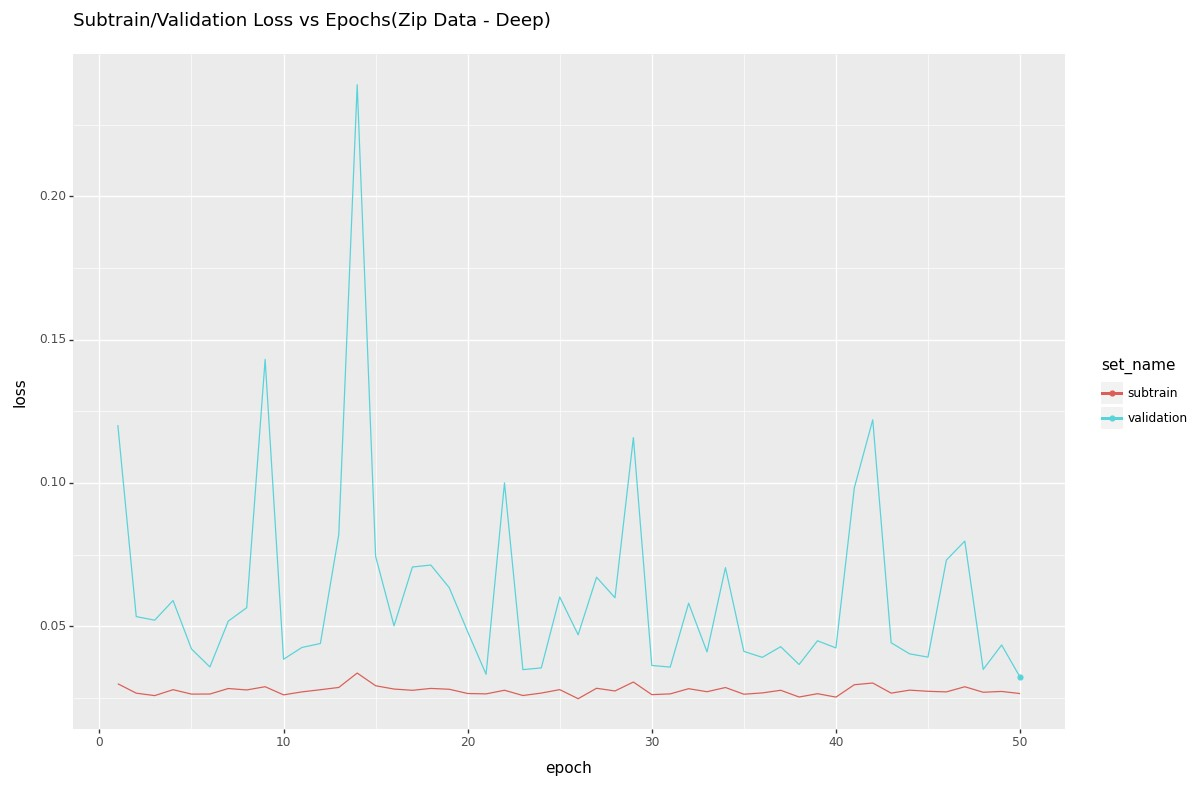
**... y = "loss",**

**... color = "set\_name"**

**... ),**

**... data = zip\_min["min\_df deep"]) +\**

**... p9.ggtitle("Subtrain/Validation Loss vs Epochs(Zip Data - Deep)")**



**Subtrain/Validation Loss Graph (Spam\_scaled Data – Linear Model):**

**>>> gg2 = p9.ggplot() +\**

**... p9.geom\_line(**

**... p9.aes(**

**... x = "epoch",**

**... y= "loss",**

**... color = "set\_name"**

**... ),**

**... data = spam\_loss["AutoGradLearnerCV Linear"]) +\**

**... p9.geom\_point(**

**... p9.aes(**

**... x = "epoch",**

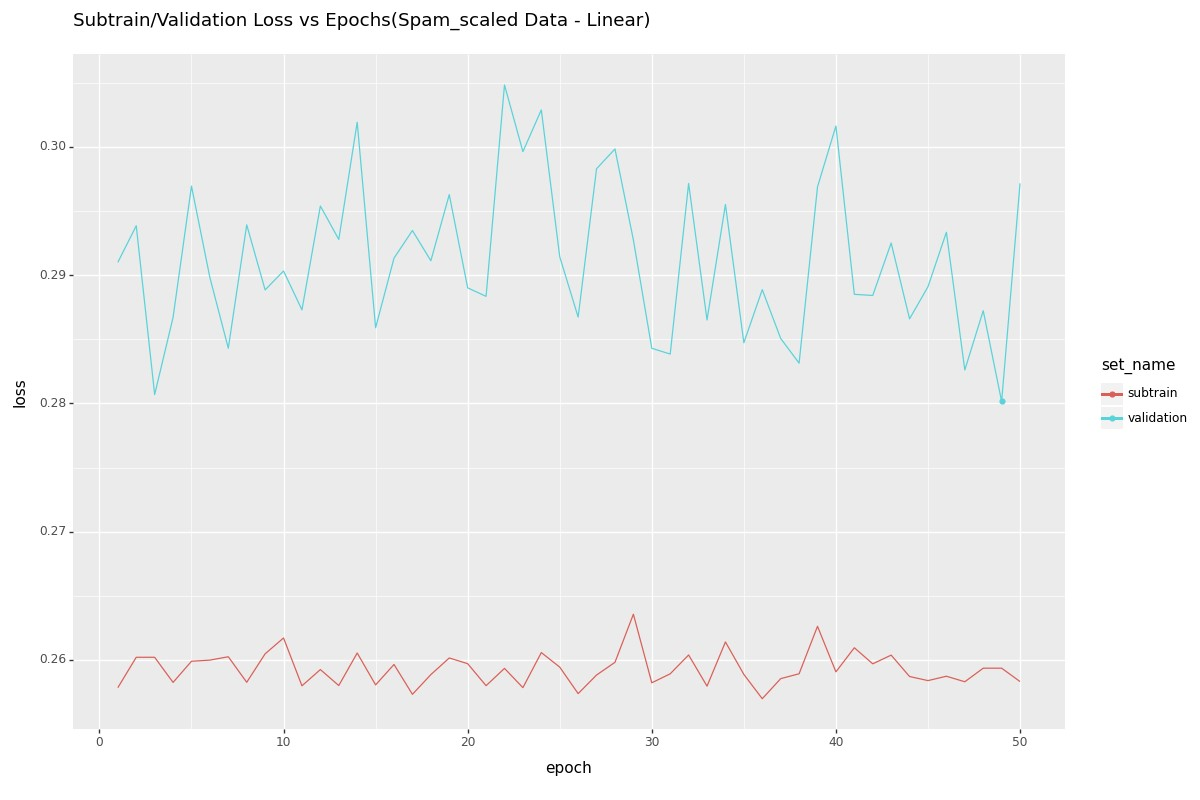
**... y = "loss",**

**... color = "set\_name"**

**... ),**

**... data = spam\_min["min\_df linear"]) +\**

**... p9.ggtitle("Subtrain/Validation Loss vs Epochs(Spam\_scaled Data - Linear)")**



**Subtrain/Validation Loss Graph (Spam\_scaled Data – Deep Model):**

