**CS599 (Deep Learning)**

**Homework – 09**

1. **Python Code:**

import torch

import pandas as pd

import matplotlib

import numpy as np

import math

matplotlib.use("agg")

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

from sklearn.neighbors import KNeighborsRegressor

from sklearn.linear\_model import LassoCV

from sklearn.pipeline import make\_pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error

from collections import Counter

data\_set\_dict = {"forest\_fires": ("forestfires.csv", ",", True),

"air\_foil": ("airfoil\_self\_noise.tsv", "\t", False)}

data\_dict = {}

hist\_df\_list = []

for data\_name, (file\_name, sep, log\_trans) in data\_set\_dict.items():

data\_df = pd.read\_csv(file\_name, sep = sep, header = 0)

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

label\_col\_num = data\_ncol - 1

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

if log\_trans:

data\_label\_vec = np.log(data\_label\_vec + 1)

label\_sd = math.sqrt(data\_label\_vec.var())

standard\_label\_vec = (

data\_label\_vec - data\_label\_vec.mean()

)/ label\_sd

is\_feature\_col = (

np.arange(data\_ncol) != label\_col\_num

) & (

data\_df.dtypes != "object"

)

data\_features = data\_df.loc[:, is\_feature\_col]

feature\_nrow, feature\_ncol = data\_features.shape

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

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

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

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

input\_tensor = torch.from\_numpy(

feature\_scaled.to\_numpy()

).float()

output\_tensor = torch.from\_numpy(

standard\_label\_vec.to\_numpy()

).float().reshape(data\_nrow, 1)

data\_dict[data\_name] = (feature\_scaled, standard\_label\_vec)

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

"data\_name": data\_name,

"label": standard\_label\_vec})

)

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

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=50):

self.max\_epochs = max\_epochs

self.batch\_size=batch\_size

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

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

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.numpy()

def predict(self, test\_features):

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

return pred\_scores

class TorchLearnerCV:

def \_\_init\_\_(self, n\_folds, units\_per\_layer=[data\_ncol,1]):

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)

test\_loss\_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] = (torch.from\_numpy(data\_features.iloc[set\_indices, :].to\_numpy()).float(),

torch.from\_numpy(np.ravel(data\_labels.iloc[set\_indices])).float())

#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 = KNeighborsRegressor()

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 = KNeighborsRegressor(n\_neighbors=best\_n\_neighbors)

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

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

#print(test\_labels)

#print(knn\_pred)

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

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

#linear model

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

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

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

#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

featureless\_pred = np.repeat(mean\_train\_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, 1])

#print("ncol:", ncol)

linear\_loss = linear\_learner.fit(train\_features, train\_labels.reshape(nrow,1))

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, 1])

deep\_loss = deep\_learner.fit(train\_features, train\_labels.reshape(nrow, 1))

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

#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 = {'KNeighborsRegressor + GridSearchCV': knn\_pred,

'LassoCV': lr\_pred,

'TorchLearnerCV Linear': ll\_pred,

'TorchLearnerCV Deep': dl\_pred,

'featureless': featureless\_pred}

test\_square\_loss = {}

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

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

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

#accuracy = np.mean(test\_labels == predictions)

test\_square\_loss[algorithm] = test\_loss

for algorithm, test\_loss in test\_square\_loss.items():

print(f"{algorithm} Test Square Loss: {test\_loss}")

test\_loss\_df = pd.DataFrame({

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

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

"algorithm": [algorithm],

"test square loss": [test\_square\_loss[algorithm]]})

test\_loss\_data\_frames.append(test\_loss\_df)

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

total\_test\_loss\_df = pd.concat(test\_loss\_data\_frames, ignore\_index = True)

print(total\_test\_loss\_df)

import plotnine as p9

gg = p9.ggplot(total\_test\_loss\_df, p9.aes(x ='test square loss', y = 'algorithm'))+\

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

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

forest\_fires\_loss = loss\_data\_dict["forest\_fires"]

air\_foil\_loss = loss\_data\_dict["air\_foil"]

forest\_fires\_min = min\_df\_dict["forest\_fires"]

air\_foil\_min = min\_df\_dict["air\_foil"]

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

+ p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = forest\_fires\_min["min\_df linear"]) #p9.geom\_line(p9.aes(fill = 'setname'))

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 = forest\_fires\_loss["TorchLearnerCV Deep"])\

+ p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = forest\_fires\_min["min\_df deep"]) #p9.geom\_line(p9.aes(fill = 'setname'))

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 = air\_foil\_loss["TorchLearnerCV Linear"])\

+ p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = air\_foil\_min["min\_df linear"]) #p9.geom\_line(p9.aes(fill = 'setname'))

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 = air\_foil\_loss["TorchLearnerCV Deep"])\

+ p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = air\_foil\_min["min\_df deep"]) #p9.geom\_line(p9.aes(fill = 'setname'))

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] = (torch.from\_numpy(data\_features.iloc[set\_indices, :].to\_numpy()).float(),**

**... torch.from\_numpy(np.ravel(data\_labels.iloc[set\_indices])).float())**

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

**... ...**

**...**

**... for algorithm, test\_loss in test\_square\_loss.items():**

**... print(f"{algorithm} Test Square Loss: {test\_loss}")**

**... test\_loss\_df = pd.DataFrame({**

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

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

**... "algorithm": [algorithm],**

**... "test square loss": [test\_square\_loss[algorithm]]})**

**... test\_loss\_data\_frames.append(test\_loss\_df)**

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

Best N-Neighbors = 19

Mean Train Label = 0.0049645514

Best Epoch: 0

Best Epoch: 2

KNeighborsRegressor + GridSearchCV Test Square Loss: 1.0609387159347534

LassoCV Test Square Loss: 1.0739297224352429

TorchLearnerCV Linear Test Square Loss: 1.2798919677734375

TorchLearnerCV Deep Test Square Loss: 1.073811411857605

featureless Test Square Loss: 1.0739296674728394

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of forest\_fires(0)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 20

Mean Train Label = -0.019172536

Best Epoch: 0

Best Epoch: 3

KNeighborsRegressor + GridSearchCV Test Square Loss: 0.988243818283081

LassoCV Test Square Loss: 0.9922808202492903

TorchLearnerCV Linear Test Square Loss: 1.0052711963653564

TorchLearnerCV Deep Test Square Loss: 0.989805281162262

featureless Test Square Loss: 0.9922808408737183

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of forest\_fires(1)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 17

Mean Train Label = 0.014222353

Best Epoch: 10

Best Epoch: 0

KNeighborsRegressor + GridSearchCV Test Square Loss: 1.0026485919952393

LassoCV Test Square Loss: 0.9305249905775753

TorchLearnerCV Linear Test Square Loss: 1.1973176002502441

TorchLearnerCV Deep Test Square Loss: 0.9264524579048157

featureless Test Square Loss: 0.9305248856544495

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of forest\_fires(2)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 13

Mean Train Label = -0.034039237

Best Epoch: 42

Best Epoch: 44

KNeighborsRegressor + GridSearchCV Test Square Loss: 0.31644493341445923

LassoCV Test Square Loss: 0.47843713543411764

TorchLearnerCV Linear Test Square Loss: 0.5696210265159607

TorchLearnerCV Deep Test Square Loss: 0.1302846521139145

featureless Test Square Loss: 0.9787271022796631

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of air\_foil(0)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 9

Mean Train Label = 0.00056366913

Best Epoch: 44

Best Epoch: 49

KNeighborsRegressor + GridSearchCV Test Square Loss: 0.2813766300678253

LassoCV Test Square Loss: 0.48135759164756325

TorchLearnerCV Linear Test Square Loss: 0.6965237259864807

TorchLearnerCV Deep Test Square Loss: 0.2059951275587082

featureless Test Square Loss: 0.9960943460464478

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of air\_foil(1)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 11

Mean Train Label = 0.033475567

Best Epoch: 42

Best Epoch: 45

KNeighborsRegressor + GridSearchCV Test Square Loss: 0.3124180734157562

LassoCV Test Square Loss: 0.5237005567324898

TorchLearnerCV Linear Test Square Loss: 0.5518736839294434

TorchLearnerCV Deep Test Square Loss: 0.3650369644165039

featureless Test Square Loss: 1.0345804691314697

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of air\_foil(2)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**>>> total\_test\_loss\_df = pd.concat(test\_loss\_data\_frames, ignore\_index = True)**

**>>> print(total\_test\_loss\_df)**

data\_set fold\_id algorithm test square loss

0 forest\_fires 0 KNeighborsRegressor + GridSearchCV 1.060939

1 forest\_fires 0 LassoCV 1.073930

2 forest\_fires 0 TorchLearnerCV Linear 1.279892

3 forest\_fires 0 TorchLearnerCV Deep 1.073811

4 forest\_fires 0 featureless 1.073930

5 forest\_fires 1 KNeighborsRegressor + GridSearchCV 0.988244

6 forest\_fires 1 LassoCV 0.992281

7 forest\_fires 1 TorchLearnerCV Linear 1.005271

8 forest\_fires 1 TorchLearnerCV Deep 0.989805

9 forest\_fires 1 featureless 0.992281

10 forest\_fires 2 KNeighborsRegressor + GridSearchCV 1.002649

11 forest\_fires 2 LassoCV 0.930525

12 forest\_fires 2 TorchLearnerCV Linear 1.197318

13 forest\_fires 2 TorchLearnerCV Deep 0.926452

14 forest\_fires 2 featureless 0.930525

15 air\_foil 0 KNeighborsRegressor + GridSearchCV 0.316445

16 air\_foil 0 LassoCV 0.478437

17 air\_foil 0 TorchLearnerCV Linear 0.569621

18 air\_foil 0 TorchLearnerCV Deep 0.130285

19 air\_foil 0 featureless 0.978727

20 air\_foil 1 KNeighborsRegressor + GridSearchCV 0.281377

21 air\_foil 1 LassoCV 0.481358

22 air\_foil 1 TorchLearnerCV Linear 0.696524

23 air\_foil 1 TorchLearnerCV Deep 0.205995

24 air\_foil 1 featureless 0.996094

25 air\_foil 2 KNeighborsRegressor + GridSearchCV 0.312418

26 air\_foil 2 LassoCV 0.523701

27 air\_foil 2 TorchLearnerCV Linear 0.551874

28 air\_foil 2 TorchLearnerCV Deep 0.365037

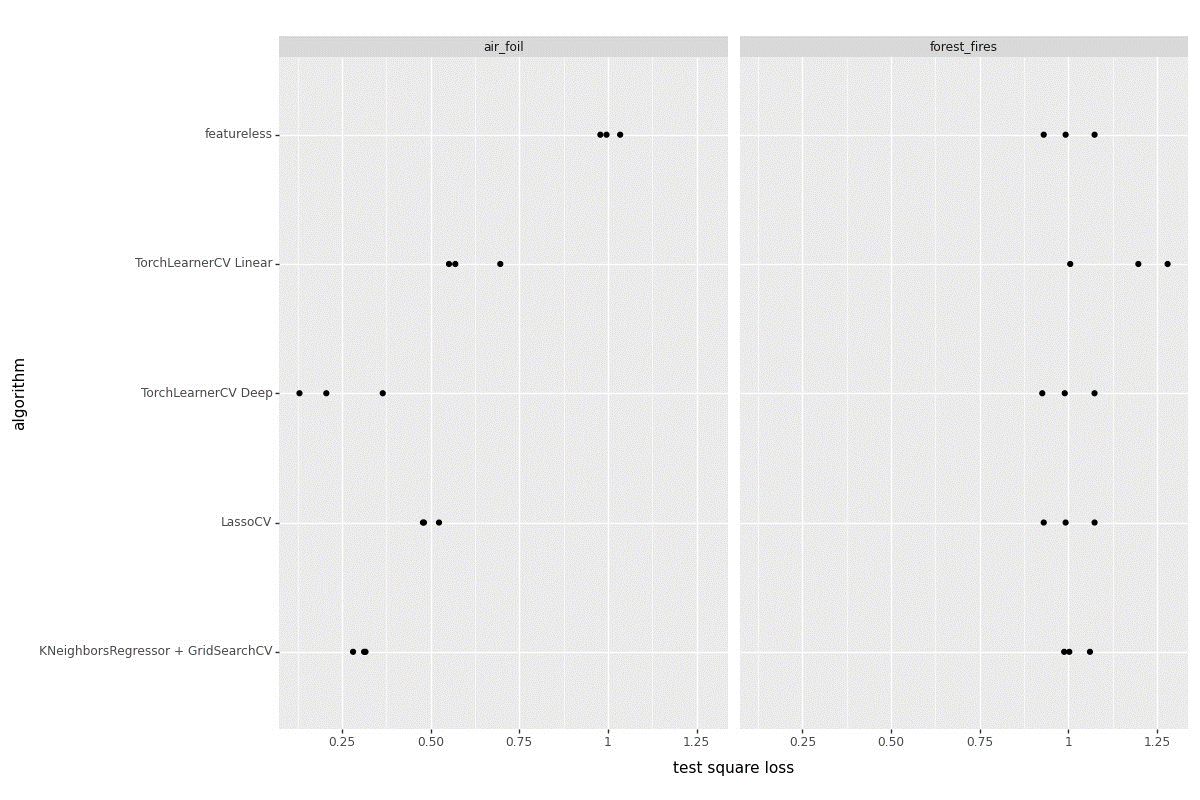
29 air\_foil 2 featureless 1.034580

**Test Loss Square Graph:**

**>>> gg = p9.ggplot(total\_test\_loss\_df, p9.aes(x ='test square loss', y = 'algorithm'))+\**

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

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

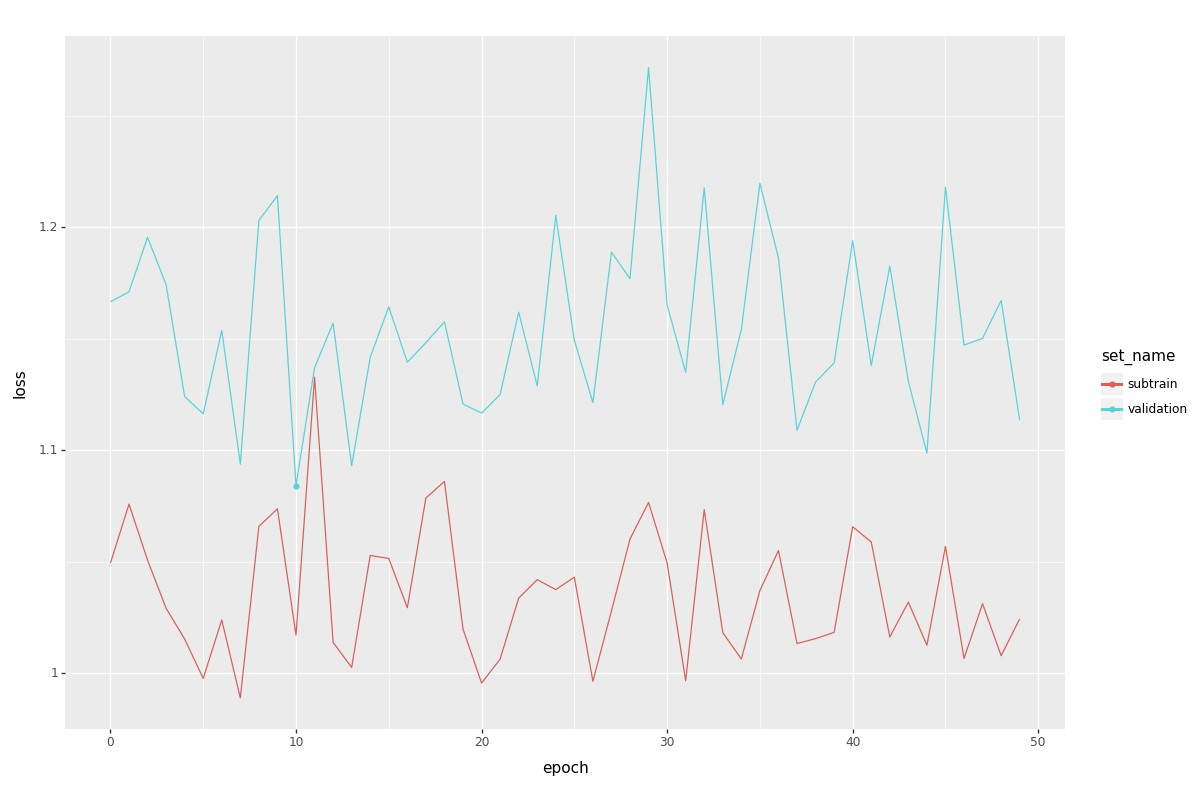


**Linear subtrain/validation loss graph (Forest Fires):**

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

**... + p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = forest\_fires\_min["min\_df linear"]) #p9.geom\_line(p9.aes(fill = 'setname'))**

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

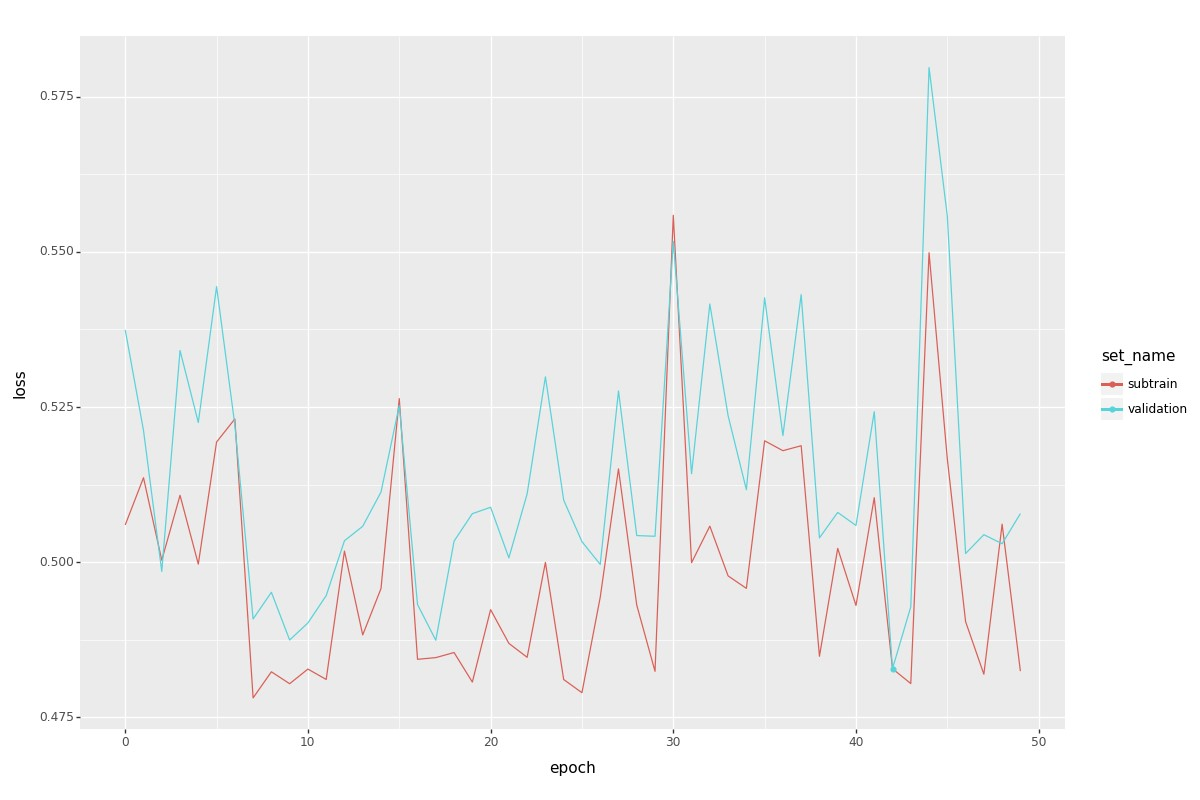


**Linear subtrain/validation loss graph (Air Foil):**

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

**... + p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = air\_foil\_min["min\_df linear"]) #p9.geom\_line(p9.aes(fill = 'setname'))**

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

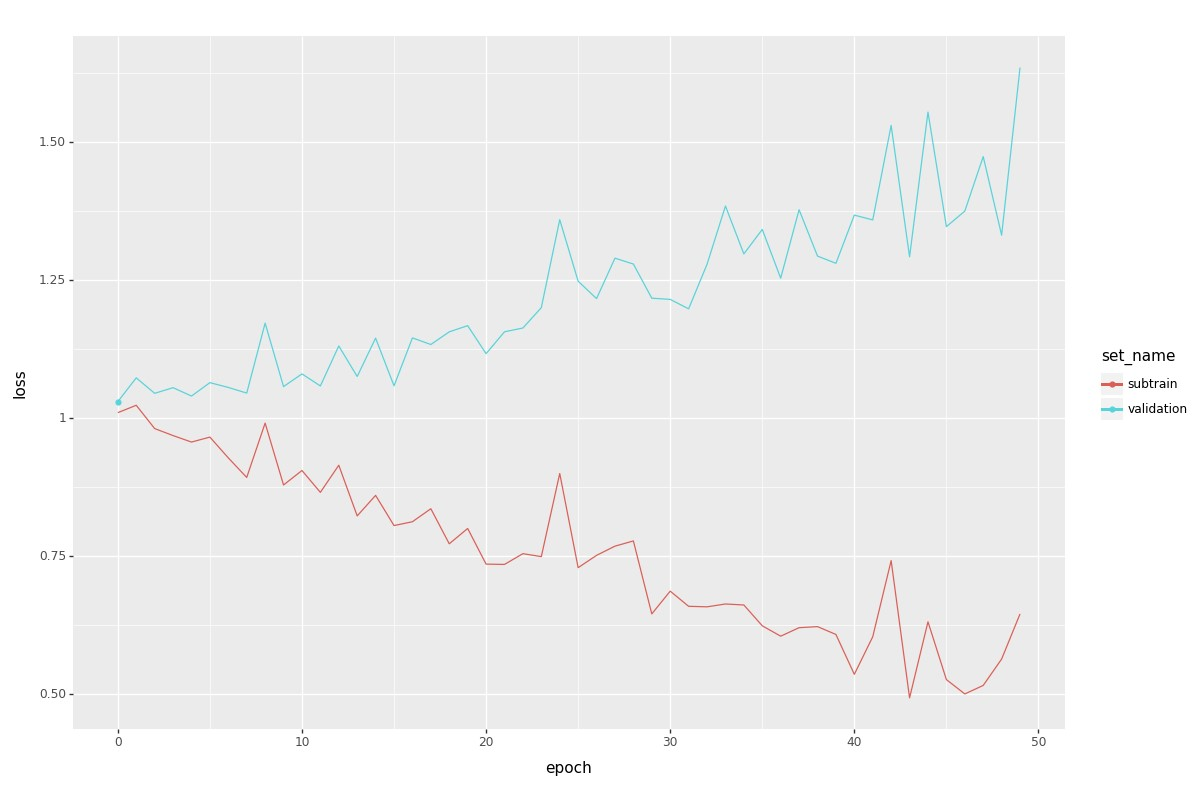


**Deep subtrain/validation loss graph (Forest Fires):**

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

**... + p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = forest\_fires\_min["min\_df deep"]) #p9.geom\_line(p9.aes(fill = 'setname'))**

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



**Deep subtrain/validation loss graph (Air Foil):**

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

**... + p9.geom\_point(p9.aes(x ='epoch', y = 'loss', color = 'set\_name'), data = air\_foil\_min["min\_df deep"]) #p9.geom\_line(p9.aes(fill = 'setname'))**

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

