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)
```

2. **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)

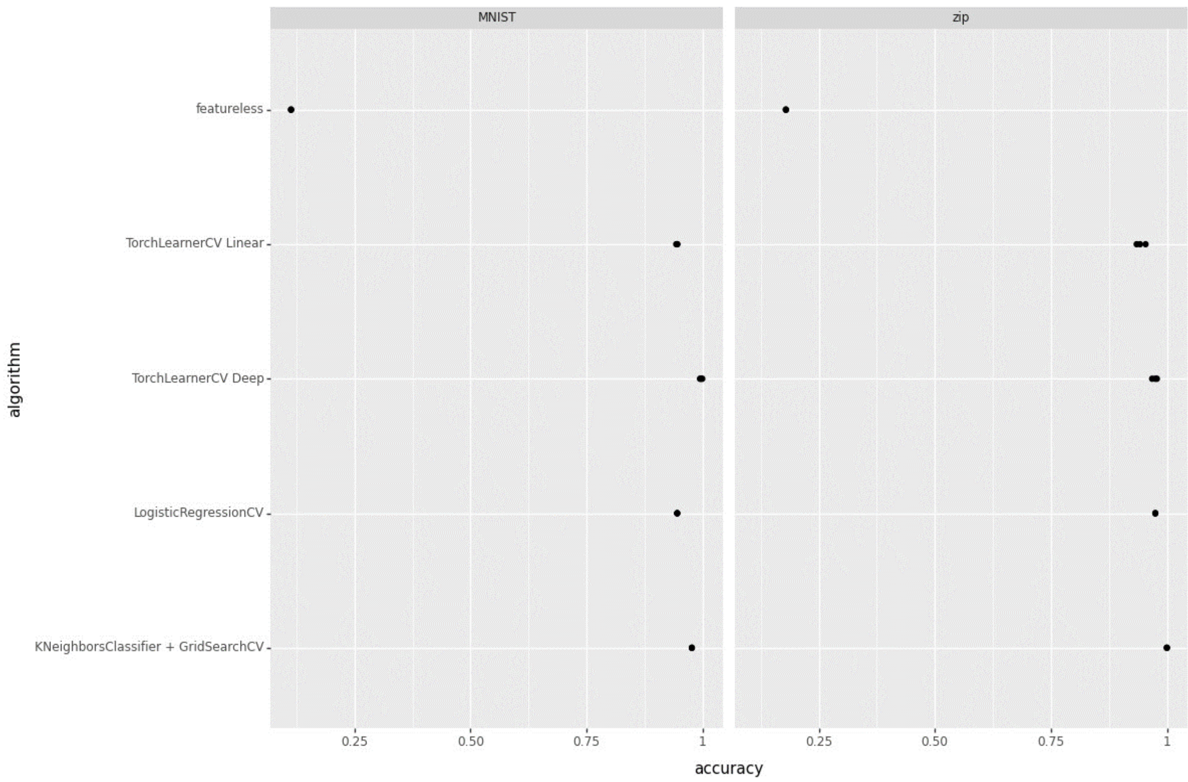
>>> print(total_accuracy_df)

	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:

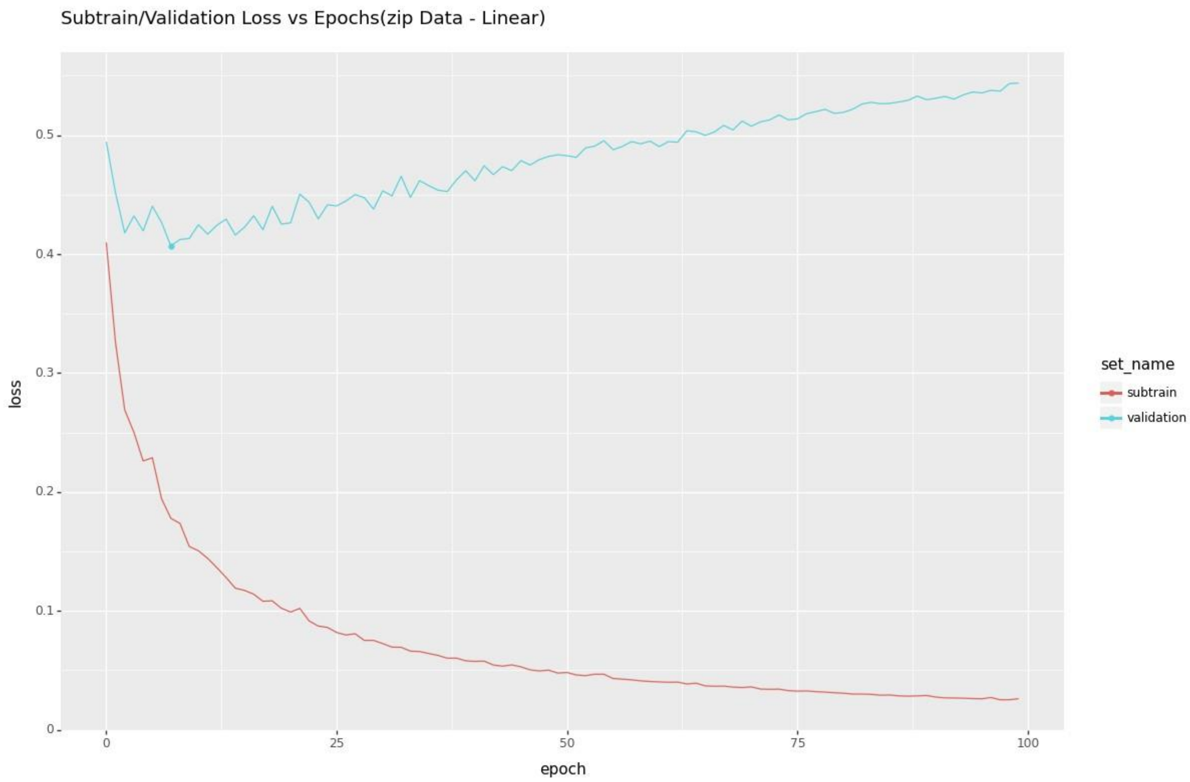
```
>>> gg = p9.ggplot(total_accuracy_df, p9.aes(x ='accuracy', y = 'algorithm'))+\
...     p9.facet_grid('.~data_set') + p9.geom_point()

>>> 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)")

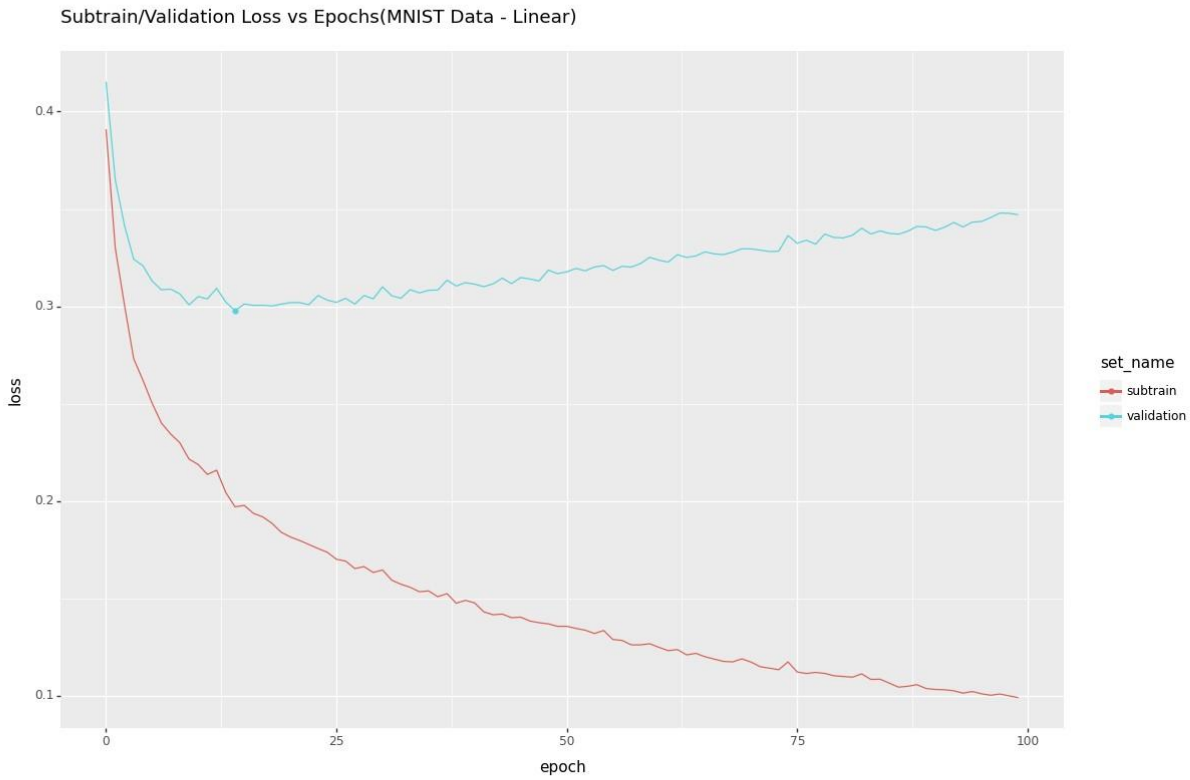
>>> gg1.save("Torch_validation_graph1.png", height = 8, width = 12)



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)")

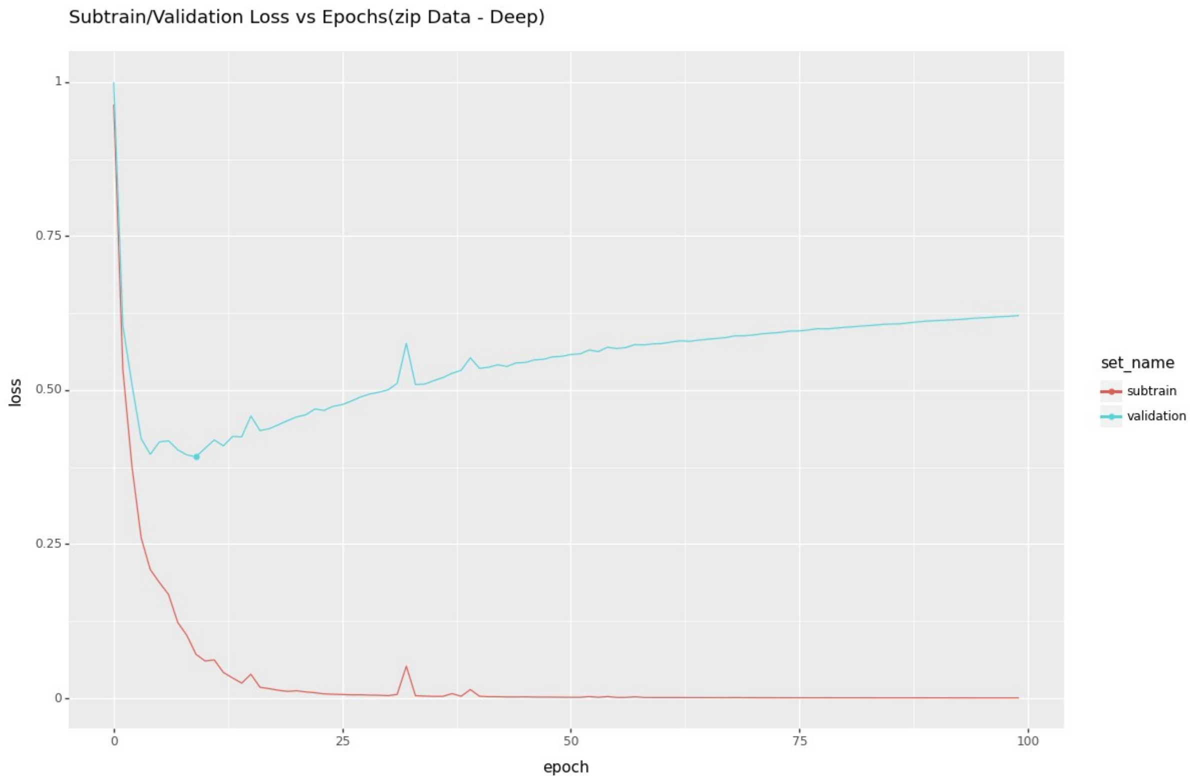
>>> gg3.save("Torch_validation_graph3.png", height = 8, width = 12)



Deep subtrain/validation loss graph (zip):

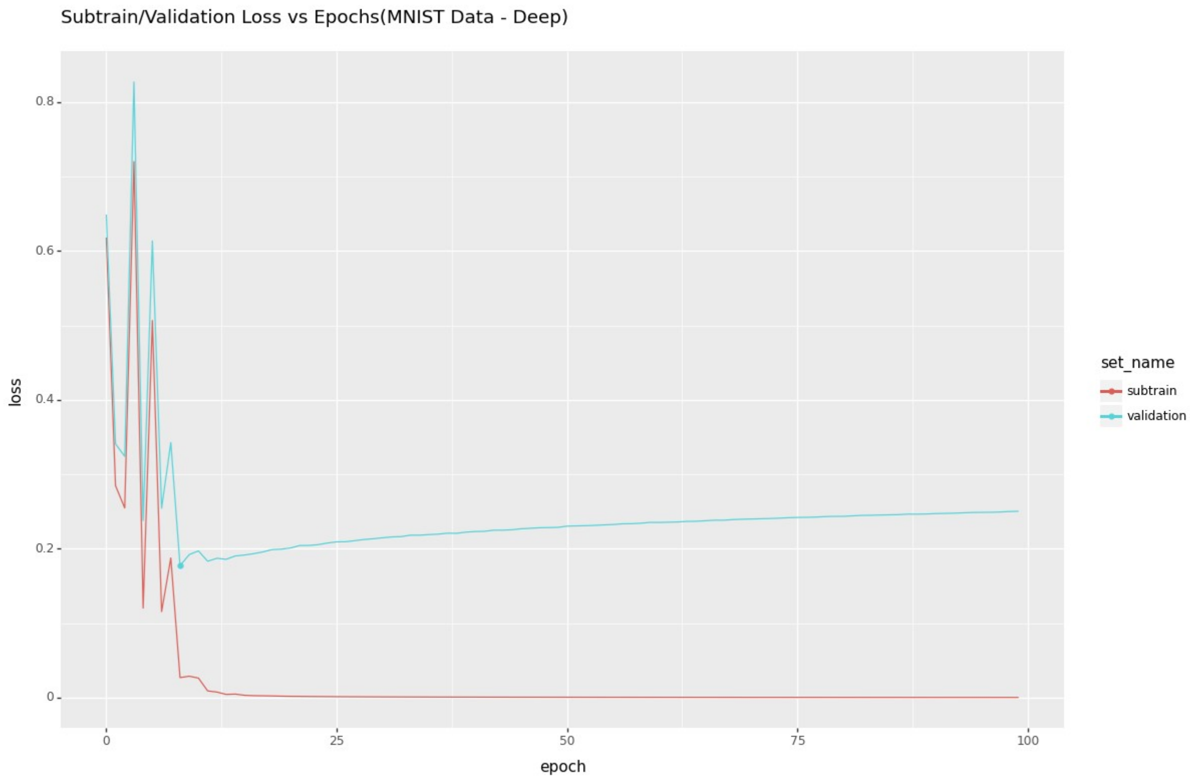
>>> gg2 = p9.ggplot() + p9.geom_line(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data = zip_loss["TorchLearnerCV Deep"])
... + p9.geom_point(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data = zip_min["min_df deep"]) + p9.ggtitle("Subtrain/Validation Loss vs Epochs(zip Data - Deep)")

>>> 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)
```



Subtrain/Validation Loss vs Epochs(MNIST Data - Deep)
loss
0.8
0.6
0.4
0.2
0
0
25
50
75
100
epoch
set_name
subtrain
validation