CS 599 (Deep Learning)

Homework – 06

1. **Python Code:**

```
import torch
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)

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 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:
        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, max_epochs, batch_size, step_size):
    self.max_epochs = max_epochs
    self.batch_size = batch_size
    self.step_size = step_size
    self.model = TorchModel(units_per_layer)
    self.optimizer = torch.optim.SGD(self.model.parameters(), lr = self.step_size)
    self.loss_fun = torch.nn.BCEWithLogitsLoss()

  def take_step(self, X, y):
    self.optimizer.zero_grad()
    pred_tensor = self.model(X)
    loss_value = self.loss_fun(pred_tensor, y)
    loss_value.backward()
    self.optimizer.step()
    return loss_value.item()

  def fit(self, X, y):
    ds = CSV(X, y)
    dl = torch.utils.data.DataLoader(
      ds, batch_size = self.batch_size, shuffle = True)
    for epoch in range(self.max_epochs):
      for batch_features, batch_labels in dl:
        loss = self.take_step(batch_features, batch_labels)

  def decision_function(self, X):
```

```
        with torch.no_grad():
            pred_vec = self.model(X)
        return pred_vec.numpy()

    def predict(self, X):
        pred_scores = self.decision_function(X)
        return np.where(pred_scores > 0, 1, 0)



class TorchLearnerCV:
    def __init__(self, units_per_layer, max_epochs, batch_size, step_size, n_splits):
        self.units_per_layer = units_per_layer
        self.max_epochs = max_epochs
        self.batch_size = batch_size
        self.step_size = step_size
        self.n_splits = n_splits

    def fit(self, X, y):
        kf = KFold(n_splits = self.n_splits, shuffle = True, random_state = 5)

        best_loss = float('inf')
        best_model = None
        best_epochs = 0
        valid_loss = []
        sub_train_loss = []
        loss_val = []

        scores = pd.DataFrame(columns = ["validation_fold", "setname", "loss_value", "epoch"])
        for max_epoch in range(1, self.max_epochs + 1):
            loss_values = []
            for fold_num, (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]}

                learner = TorchLearner(self.units_per_layer, max_epoch, self.batch_size,
self.step_size)
                learner.fit(subtrain_data["X"], subtrain_data["y"])
                val_loss = learner.take_step(val_data["X"], val_data["y"])
                subtrain_loss = learner.take_step(subtrain_data["X"], subtrain_data["y"])

                loss_values.append(val_loss)
                loss_dict = {"validation": val_loss, "subtrain": subtrain_loss}
                for setname, loss_value in loss_dict.items():
                    loss_val_row = pd.DataFrame({
                              "validation_fold": [fold_num],
                              "setname": [setname],
                              "loss_value": [loss_value],
```

```
                        "epoch" : [max_epoch]})
                loss_val.append(loss_val_row)
            loss_df = pd.concat(loss_val)
            loss_values_mean = np.mean(loss_values)

            if loss_values_mean < best_loss:
                best_loss = loss_values_mean
                best_model = learner
                best_epochs = max_epoch


        print("best_epoch: ", best_epochs)
        self.best_model = best_model
        self.best_model.fit(X,y)
        return loss_df
    def predict(self, X):
        return self.best_model.predict(X)




zip_features, zip_labels = data_dict["zip"]
input_tensor = torch.from_numpy(zip_features.to_numpy()).float()
output_tensor = torch.from_numpy(zip_labels.to_numpy()).float()


accuracy_data_frames = []
loss_data_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))
    data_nrow, data_ncol = data_features.shape

    for fold_num, (train_index, test_index) in enum_obj:
        X_train, X_test = torch.from_numpy(data_features.iloc[train_index].to_numpy()).float(),
torch.from_numpy(data_features.iloc[test_index].to_numpy()).float()
        y_train, y_test = torch.from_numpy(data_labels.iloc[train_index].to_numpy()).float(),
torch.from_numpy(data_labels.iloc[test_index].to_numpy()).float()

        input_tensor = torch.from_numpy(data_features.to_numpy()).float()
        output_tensor = torch.from_numpy(data_labels.to_numpy()).float()
        # K-nearest neighbors
        knn = KNeighborsClassifier()
        hp_parameters = {"n_neighbors": list(range(1, 21))}
        grid = GridSearchCV(knn, hp_parameters, cv=3)
        grid.fit(X_train, y_train.ravel())
        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.ravel())
knn_pred = knn.predict(X_test)

# Logistic Regression
pipe = make_pipeline(StandardScaler(), LogisticRegressionCV(cv=3, max_iter=2000))
pipe.fit(X_train, y_train.ravel())
lr_pred = pipe.predict(X_test)
y_train_series = pd.Series(y_train.ravel())

#TorchLearnerCV
torch_learner = TorchLearnerCV([data_ncol, 100, 1], 15, 100, 0.15, 3)
loss = torch_learner.fit(X_train, y_train)
tl_pred = torch_learner.predict(X_test)

#TorchLearnerCV + deep
torch_learner_deep = TorchLearnerCV([data_ncol, 100, 10, 5, 1], 10, 100, 0.5, 3)
deep_loss = torch_learner_deep.fit(X_train, y_train)
tl_deep_pred = torch_learner_deep.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.ravel(), most_frequent_class)

# store predict data in dict
pred_dict = {'gridSearch + nearest neighbors': knn_pred,
        'linear_model': lr_pred,
        'TorchLearnerCV': tl_pred,
        'TorchLearnerCV + Deep': tl_deep_pred,
        'featureless': featureless_pred}
test_accuracy = {}

loss_data_dict[data_name] = {'TorchLearnerCV': loss,
        'TorchLearnerCV + Deep': deep_loss}


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)


zip_loss = loss_data_dict["zip"]
spam_loss = loss_data_dict["spam_scaled"]

gg1 = p9.ggplot(zip_loss["TorchLearnerCV"], p9.aes(x ='epoch', y = 'loss_value', fill =
'setname')) + p9.facet_grid('.~validation_fold') + p9.geom_point() + p9.ggtitle("Zip
Data(TorchLearner): Subtrain/Validation vs Epoch")
gg1.save("valid_graph1.png", height = 8, width = 12)

gg2 = p9.ggplot(zip_loss["TorchLearnerCV + Deep"], p9.aes(x ='epoch', y = 'loss_value', fill =
'setname')) + p9.facet_grid('.~validation_fold') + p9.geom_point() + p9.ggtitle("Zip
Data(TorchLearner + Deep): Subtrain/Validation vs Epoch")
gg2.save("valid_graph2.png", height = 8, width = 12)

gg3 = p9.ggplot(spam_loss["TorchLearnerCV"], p9.aes(x ='epoch', y = 'loss_value', fill =
'setname')) + p9.facet_grid('.~validation_fold') + p9.geom_point() + p9.ggtitle("Spam_scaled
Data(TorchLearner): Subtrain/Validation vs Epoch")
gg3.save("valid_graph3.png", height = 8, width = 12)

gg4 = p9.ggplot(spam_loss["TorchLearnerCV + Deep"], p9.aes(x ='epoch', y = 'loss_value', fill
= 'setname')) \
 + p9.facet_grid('.~validation_fold') + p9.geom_point() + p9.ggtitle("spam_scaled
Data(TorchLearner + Deep): Subtrain/Validation vs Epoch")
gg4.save("valid_graph4.png", height = 8, width = 12)
```

2. **Outputs:**

```
>>> 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))
...     data_nrow, data_ncol = data_features.shape
```

```
...
...    for fold_num, (train_index, test_index) in enum_obj:
...        X_train, X_test =
torch.from_numpy(data_features.iloc[train_index].to_numpy()).float(),
torch.from_numpy(data_features.iloc[test_index].to_numpy()).float()
...        y_train, y_test =
torch.from_numpy(data_labels.iloc[train_index].to_numpy()).float(),
torch.from_numpy(data_labels.iloc[test_index].to_numpy()).float()
...
...        input_tensor = torch.from_numpy(data_features.to_numpy()).float()
... ...
...
...        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
best_epoch:  11
best_epoch:  9
Most Frequent Class =  0.0
gridSearch + nearest neighbors Test Accuracy: 100.0
linear_model Test Accuracy: 99.51923076923077
TorchLearnerCV Test Accuracy: 98.5576923076923
TorchLearnerCV + Deep Test Accuracy: 99.03846153846155
featureless Test Accuracy: 58.65384615384615
****************************End of zip(0)******************************
Best N-Neighbors =  1
best_epoch:  14
best_epoch:  5
Most Frequent Class =  0.0
gridSearch + nearest neighbors Test Accuracy: 99.51923076923077
linear_model Test Accuracy: 99.03846153846155
TorchLearnerCV Test Accuracy: 96.63461538461539
TorchLearnerCV + Deep Test Accuracy: 96.63461538461539
featureless Test Accuracy: 57.21153846153846
****************************End of zip(1)******************************
Best N-Neighbors =  4
best_epoch:  14
best_epoch:  7
Most Frequent Class =  0.0
gridSearch + nearest neighbors Test Accuracy: 99.03381642512076
```

linear_model Test Accuracy: 99.03381642512076
TorchLearnerCV Test Accuracy: 97.58454106280193
TorchLearnerCV + Deep Test Accuracy: 98.55072463768117
featureless Test Accuracy: 57.00483091787439
***************************End of zip(2)****************************
Best N-Neighbors = 4
best_epoch: 14
best_epoch: 10
Most Frequent Class = 0.0
gridSearch + nearest neighbors Test Accuracy: 88.72229465449804
linear_model Test Accuracy: 91.52542372881356
TorchLearnerCV Test Accuracy: 91.98174706649283
TorchLearnerCV + Deep Test Accuracy: 93.08996088657105
featureless Test Accuracy: 60.88657105606258
***************************End of spam_scaled(0)****************************
Best N-Neighbors = 5
best_epoch: 15
best_epoch: 7
Most Frequent Class = 0.0
gridSearch + nearest neighbors Test Accuracy: 90.80834419817471
linear_model Test Accuracy: 91.78617992177314
TorchLearnerCV Test Accuracy: 92.69882659713168
TorchLearnerCV + Deep Test Accuracy: 91.85136897001304
featureless Test Accuracy: 60.104302477183836
***************************End of spam_scaled(1)****************************
Best N-Neighbors = 9
best_epoch: 15
best_epoch: 10
Most Frequent Class = 0.0
gridSearch + nearest neighbors Test Accuracy: 90.54142204827136
linear_model Test Accuracy: 92.49836921069797
TorchLearnerCV Test Accuracy: 91.71559034572732
TorchLearnerCV + Deep Test Accuracy: 92.75929549902152
featureless Test Accuracy: 60.79582517938682
***************************End of spam_scaled(2)****************************

>>> total_accuracy_df = pd.concat(accuracy_data_frames, ignore_index = True)

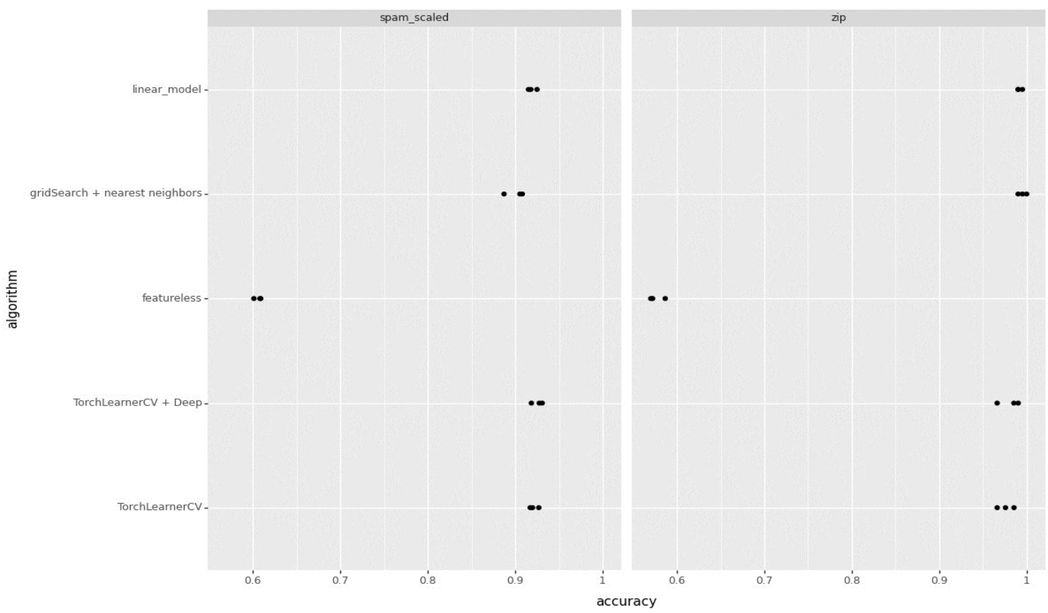
>>> print(total_accuracy_df)

	data_set	fold_id	algorithm	accuracy
0	zip	0	gridSearch + nearest neighbors	1.000000
1	zip	0	linear_model	0.995192
2	zip	0	TorchLearnerCV	0.985577
3	zip	0	TorchLearnerCV + Deep	0.990385
4	zip	0	featureless	0.586538
5	zip	1	gridSearch + nearest neighbors	0.995192

6	zip	1	linear_model	0.990385
7	zip	1	TorchLearnerCV	0.966346
8	zip	1	TorchLearnerCV + Deep	0.966346
9	zip	1	featureless	0.572115
10	zip	2	gridSearch + nearest neighbors	0.990338
11	zip	2	linear_model	0.990338
12	zip	2	TorchLearnerCV	0.975845
13	zip	2	TorchLearnerCV + Deep	0.985507
14	zip	2	featureless	0.570048
15	spam_scaled	0	gridSearch + nearest neighbors	0.887223
16	spam_scaled	0	linear_model	0.915254
17	spam_scaled	0	TorchLearnerCV	0.919817
18	spam_scaled	0	TorchLearnerCV + Deep	0.930900
19	spam_scaled	0	featureless	0.608866
20	spam_scaled	1	gridSearch + nearest neighbors	0.908083
21	spam_scaled	1	linear_model	0.917862
22	spam_scaled	1	TorchLearnerCV	0.926988
23	spam_scaled	1	TorchLearnerCV + Deep	0.918514
24	spam_scaled	1	featureless	0.601043
25	spam_scaled	2	gridSearch + nearest neighbors	0.905414
26	spam_scaled	2	linear_model	0.924984
27	spam_scaled	2	TorchLearnerCV	0.917156
28	spam_scaled	2	TorchLearnerCV + Deep	0.927593
29	spam_scaled	2	featureless	0.607958

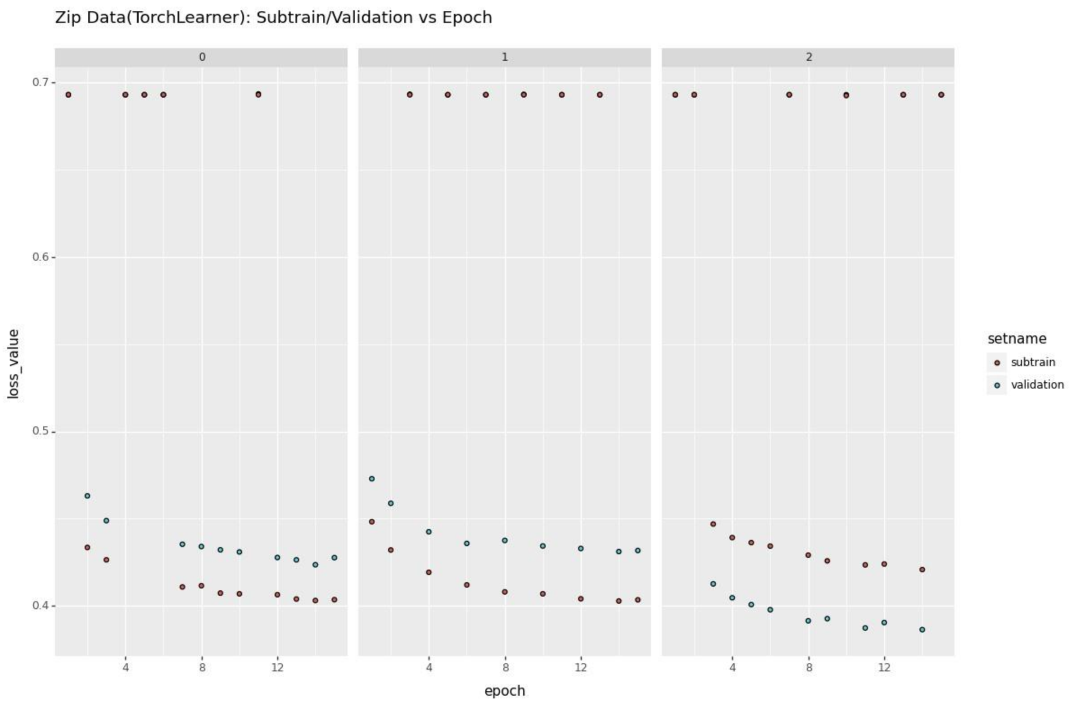
```
>>> gg = p9.ggplot(total_accuracy_df, p9.aes(x ='accuracy', y = 'algorithm'))+\
...     p9.facet_grid('.~data_set') + p9.geom_point()

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



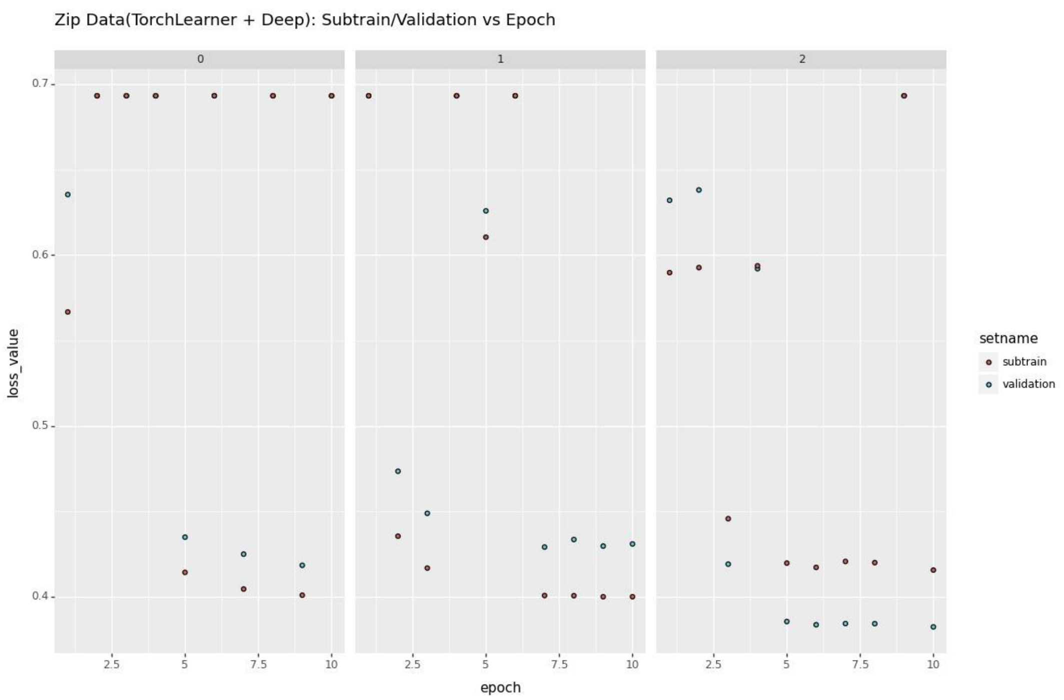
>>> gg1 = p9.ggplot(zip_loss["TorchLearnerCV"], p9.aes(x ='epoch', y = 'loss_value', fill = 'setname')) + p9.facet_grid('.~validation_fold') + p9.geom_point() + p9.ggtitle("Zip Data(TorchLearner): Subtrain/Validation vs Epoch") #p9.geom_line(p9.aes(fill = 'setname'))

>>> gg1.save("valid_graph1.png", height = 8, width = 12)



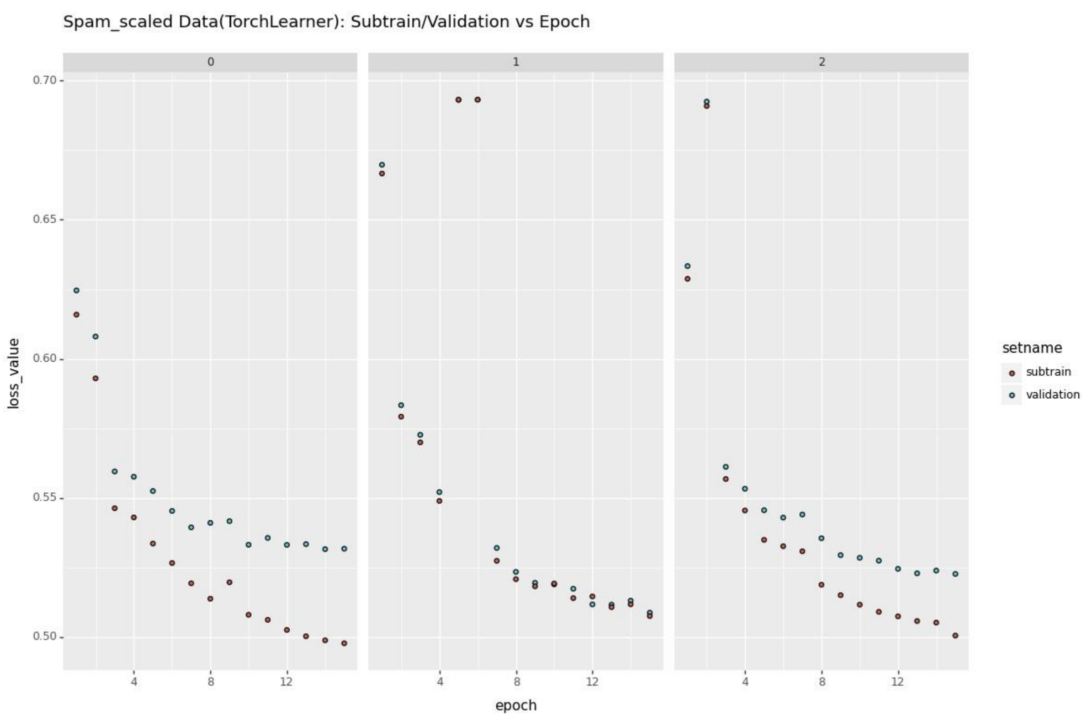
>>> gg2 = p9.ggplot(zip_loss["TorchLearnerCV + Deep"], p9.aes(x ='epoch', y = 'loss_value', fill = 'setname')) + p9.facet_grid('.~validation_fold') + p9.geom_point() + p9.ggtitle("Zip Data(TorchLearner + Deep): Subtrain/Validation vs Epoch") #p9.geom_line(p9.aes(fill = 'setname'))

>>> gg2.save("c:/Users/Anudeep Kumar/OneDrive/Desktop/Fall 2023/CS599-Deep Learning/Homework/HW06/valid_graph2.png", height = 8, width = 12)



>>> gg3 = p9.ggplot(spam_loss["TorchLearnerCV"], p9.aes(x ='epoch', y = 'loss_value', fill = 'setname')) + p9.facet_grid('.~validation_fold') + p9.geom_point() + p9.ggtitle("Spam_scaled Data(TorchLearner): Subtrain/Validation vs Epoch") #p9.geom_line(p9.aes(fill = 'setname'))

>>> gg3.save("valid_graph3.png", height = 8, width = 12)



**>>> gg4 = p9.ggplot(spam_loss["TorchLearnerCV + Deep"], p9.aes(x ='epoch', y = 'loss_value', fill = 'setname')) **
... + p9.facet_grid('.~validation_fold') + p9.geom_point() + p9.ggtitle("spam_scaled Data(TorchLearner + Deep): Subtrain/Validation vs Epoch") #p9.geom_line(p9.aes(fill = 'setname'))

>>> gg4.save("valid_graph4.png", height = 8, width = 12)