CS599 (Deep Learning)

Homework – 11

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 pdb

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_label_col = data_df.columns == label_col_num
  data_features = data_df.iloc[:, ~is_label_col]
  data_labels = data_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-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

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)

class MyCV:
    def __init__(self, estimator, param_grid, cv):
        """estimator: learner instance
        pram_grid: list of dictionaries
        cv: number of folds"""
        self.cv = cv
        self.param_grid = param_grid
        self.estimator = estimator
    def fit_one(self, param_dict, X, y):
        """Run self.estimator.fit on one parameter combination"""
        for param_name, param_value in param_dict.items():
            setattr(self.estimator, param_name, param_value)
        self.estimator.fit(X, y)
    def fit(self, X, y):
        """cross-validation for selecting the best dictionary is param_grid"""
        validation_df_list = []
        train_nrow, train_ncol = X.shape
        times_to_repeat = int(math.ceil(train_nrow/self.cv))
        fold_id_vec = np.tile(np.arange(self.cv), times_to_repeat)[:train_nrow]
        np.random.shuffle(fold_id_vec)
        for validation_fold in range(self.cv):
            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():
                split_data_dict[set_name] = (
                X[is_set],
                y[is_set])
            for param_number, param_dict in enumerate(self.param_grid):
                self.fit_one(param_dict, *split_data_dict["subtrain"])
                X_valid, y_valid = split_data_dict["validation"]
                pred_valid = self.estimator.predict(X_valid)
                #pdb.set_trace()
                is_correct = pred_valid == y_valid
                #self.estimator.fit(*split_data_dict["validation"])
                valid_loss = self.estimator.train_df.query("set_name=='validation'")["loss"].mean()
                subtrain_loss =
self.estimator.train_df.query("set_name=='subtrain'")["loss"].mean()
                validation_row1 = pd.DataFrame({
                "set_name": "subtrain",
```

```
            "validation_fold": validation_fold,
            "accuracy_percent": float(is_correct.float().mean()),
            "param_number": [param_number],
            "loss": float(subtrain_loss)
            }, index = [0])
            validation_row2 = pd.DataFrame({
            "set_name": "validation",
            "validation_fold": validation_fold,
            "accuracy_percent": float(is_correct.float().mean()),
            "param_number": [param_number],
            "loss": float(valid_loss)
            }, index = [0])
            validation_df_list.append(validation_row1)
            validation_df_list.append(validation_row2)
        self.validation_df = pd.concat(validation_df_list)
        self.mean_valid_loss =
self.validation_df.groupby("param_number")["loss"].mean().reset_index()
        self.train_df = self.validation_df.groupby(["set_name", "loss"]).mean().reset_index()
        best_index = self.mean_valid_loss["loss"].argmin()
        #pdb.set_trace()
        valid_df = self.train_df.query("set_name == 'validation'")
        self.min_df = valid_df.query("param_number==%s"%(best_index))
        self.best_param_dict = self.param_grid[best_index]
        self.fit_one(self.best_param_dict, X, y)

    def predict(self, X):
        return self.estimator.predict(X)


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] = (torch.from_numpy(data_features.iloc[set_indices,
:].to_numpy()).float(),
                        torch.from_numpy(np.ravel(data_labels.iloc[set_indices])).flatten())
        #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)

hyper_params = []
n_classes = 10
for n_layers in range(5):
    hyper_params.append({
        "units_per_layer": [
            ncol
        ] + [10 for layer_num in range(n_layers)] + [n_classes]
    })

#MyCV + regularizedMLP
my_cv_learner = MyCV(
    estimator = TorchLearnerCV(3, [ncol, 1]),
```

```
            param_grid = hyper_params,
            cv = 2)
        my_cv_learner.fit(train_features, train_labels)
        my_cv_pred = my_cv_learner.predict(test_features)

        min_df_dict[data_name] = {'min_df_estimator': my_cv_learner.estimator.min_df,
                          'min_df': my_cv_learner.min_df}

        loss_data_dict[data_name] = {'my_cv_learner_estimator':
my_cv_learner.estimator.train_df,
                           'my_cv_learner': my_cv_learner.validation_df}

        # store predict data in dict
        pred_dict = {'KNeighborsClassifier + GridSearchCV': knn_pred,
                'LogisticRegressionCV': lr_pred,
                'MyCV + RegularizedMLP': my_cv_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("output.png", height = 8, width = 12)


gg1 = p9.ggplot() +\
```

```
    p9.geom_line(
    p9.aes(
    x = "epoch",
    y= "loss",
    color = "set_name"
    ),
    data = loss_data_dict["zip"]["my_cv_learner_estimator"]) +\
    p9.geom_point(
    p9.aes(
    x = "epoch",
    y = "loss",
    color = "set_name"
    ),
    data = min_df_dict["zip"]["min_df_estimator"]) +\
    p9.ggtitle("Subtrain/Validation Loss vs Epochs(Zip - Data)")

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 = loss_data_dict["spam_scaled"]["my_cv_learner_estimator"]) +\
    p9.geom_point(
    p9.aes(
    x = "epoch",
    y = "loss",
    color = "set_name"
    ),
    data = min_df_dict["spam_scaled"]["min_df_estimator"]) +\
    p9.ggtitle("Subtrain/Validation Loss vs Epochs(Spam_scaled - Data)")

gg2.save("Torch_validation_graph2.png", height = 8, width = 12)

gg3 = p9.ggplot() +\
    p9.geom_line(
    p9.aes(
    x = "param_number",
    y= "loss",
    color = "set_name"
    ),
    data = loss_data_dict["zip"]["my_cv_learner"]) +\
    p9.geom_point(
    p9.aes(
    x = "param_number",
```

```
    y = "loss",
    color = "set_name"
    ),
    data = min_df_dict["zip"]["min_df"]) +\
    p9.facet_grid('.~validation_fold') +\
    p9.ggtitle("Subtrain/Validation Loss vs Number of Hidden Layers(Zip - Data)")

gg3.save("01loss_graph1.png", height = 8, width = 12)

gg4 = p9.ggplot() +\
    p9.geom_line(
    p9.aes(
    x = "param_number",
    y= "loss",
    color = "set_name"
    ),
    data = loss_data_dict["spam_scaled"]["my_cv_learner"]) +\
    p9.geom_point(
    p9.aes(
    x = "param_number",
    y = "loss",
    color = "set_name"
    ),
    data = min_df_dict["spam_scaled"]["min_df"]) +\
    p9.facet_grid('.~validation_fold') +\
    p9.ggtitle("Subtrain/Validation Loss vs Number of Hidden Layers(Spam_scaled - Data)")

gg4.save("01loss_graph2.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] = (torch.from_numpy(data_features.iloc[set_indices,
:].to_numpy()).float(),
...                                     torch.from_numpy(np.ravel(data_labels.iloc[set_indices])).flatten())
...         #x = {data_name:X.shape for data_name, (X,y) in split_data.items()}
... ...
...
...         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:  1338 256
Best N-Neighbors =  1
Most Frequent Label =  0
Best Epoch:  8
Best Epoch:  8
Best Epoch:  20
Best Epoch:  19
Best Epoch:  52
Best Epoch:  9
Best Epoch:  14
Best Epoch:  12
Best Epoch:  14
Best Epoch:  29
Best Epoch:  8
KNeighborsClassifier + GridSearchCV Test Accuracy: 90.5829596412556
LogisticRegressionCV Test Accuracy: 89.8355754857997
MyCV + RegularizedMLP Test Accuracy: 89.68609865470853
featureless Test Accuracy: 18.53512705530643
***************************End of zip(0)****************************
zip:  1338 256
Best N-Neighbors =  1
Most Frequent Label =  0
Best Epoch:  13
Best Epoch:  11
Best Epoch:  17
Best Epoch:  25
Best Epoch:  55
Best Epoch:  8
Best Epoch:  12
Best Epoch:  15
Best Epoch:  15
Best Epoch:  63
Best Epoch:  8
KNeighborsClassifier + GridSearchCV Test Accuracy: 91.18086696562034
LogisticRegressionCV Test Accuracy: 88.34080717488789
MyCV + RegularizedMLP Test Accuracy: 87.29446935724962
featureless Test Accuracy: 17.638266068759343
***************************End of zip(1)****************************
zip:  1338 256
Best N-Neighbors =  1
Most Frequent Label =  0
```

Best Epoch: 14
Best Epoch: 16
Best Epoch: 16
Best Epoch: 25
Best Epoch: 27
Best Epoch: 11
Best Epoch: 5
Best Epoch: 9
Best Epoch: 23
Best Epoch: 44
Best Epoch: 11
KNeighborsClassifier + GridSearchCV Test Accuracy: 89.98505231689087
LogisticRegressionCV Test Accuracy: 89.98505231689087
MyCV + RegularizedMLP Test Accuracy: 88.34080717488789
featureless Test Accuracy: 17.48878923766816
***************************End of zip(2)****************************
spam_scaled: 3067 57
Best N-Neighbors = 4
Most Frequent Label = 0
Best Epoch: 66
Best Epoch: 11
Best Epoch: 5
Best Epoch: 5
Best Epoch: 14
Best Epoch: 95
Best Epoch: 17
Best Epoch: 12
Best Epoch: 10
Best Epoch: 7
Best Epoch: 12
KNeighborsClassifier + GridSearchCV Test Accuracy: 88.72229465449804
LogisticRegressionCV Test Accuracy: 91.52542372881356
MyCV + RegularizedMLP Test Accuracy: 93.48109517601043
featureless Test Accuracy: 60.88657105606258
***************************End of spam_scaled(0)****************************
spam_scaled: 3067 57
Best N-Neighbors = 5
Most Frequent Label = 0
Best Epoch: 15
Best Epoch: 10
Best Epoch: 5
Best Epoch: 13
Best Epoch: 13
Best Epoch: 28
Best Epoch: 13
Best Epoch: 10
Best Epoch: 9
Best Epoch: 11

Best Epoch: 16
KNeighborsClassifier + GridSearchCV Test Accuracy: 90.80834419817471
LogisticRegressionCV Test Accuracy: 91.78617992177314
MyCV + RegularizedMLP Test Accuracy: 92.95958279009126
featureless Test Accuracy: 60.104302477183836
***************************End of spam_scaled(1)*****************************
spam_scaled: 3068 57
Best N-Neighbors = 9
Most Frequent Label = 0
Best Epoch: 34
Best Epoch: 6
Best Epoch: 10
Best Epoch: 14
Best Epoch: 22
Best Epoch: 96
Best Epoch: 11
Best Epoch: 14
Best Epoch: 7
Best Epoch: 14
Best Epoch: 12
KNeighborsClassifier + GridSearchCV Test Accuracy: 90.54142204827136
LogisticRegressionCV Test Accuracy: 92.49836921069797
MyCV + RegularizedMLP Test Accuracy: 92.3679060665362
featureless Test Accuracy: 60.79582517938682
***************************End of spam_scaled(2)*****************************

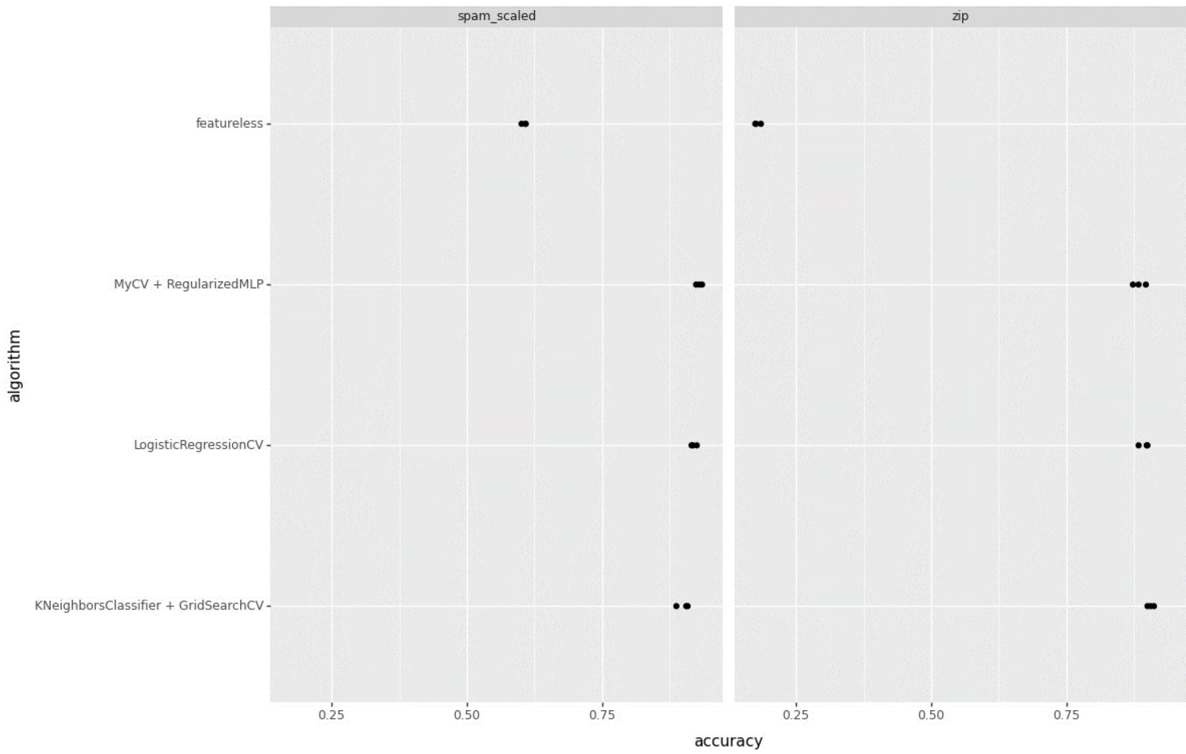
>>> total_accuracy_df = pd.concat(accuracy_data_frames, ignore_index = True)

>>> print(total_accuracy_df)

	data_set	fold_id	algorithm	accuracy
0	zip	0	KNeighborsClassifier + GridSearchCV	0.905830
1	zip	0	LogisticRegressionCV	0.898356
2	zip	0	MyCV + RegularizedMLP	0.896861
3	zip	0	featureless	0.185351
4	zip	1	KNeighborsClassifier + GridSearchCV	0.911809
5	zip	1	LogisticRegressionCV	0.883408
6	zip	1	MyCV + RegularizedMLP	0.872945
7	zip	1	featureless	0.176383
8	zip	2	KNeighborsClassifier + GridSearchCV	0.899851
9	zip	2	LogisticRegressionCV	0.899851
10	zip	2	MyCV + RegularizedMLP	0.883408
11	zip	2	featureless	0.174888
12	spam_scaled	0	KNeighborsClassifier + GridSearchCV	0.887223
13	spam_scaled	0	LogisticRegressionCV	0.915254
14	spam_scaled	0	MyCV + RegularizedMLP	0.934811
15	spam_scaled	0	featureless	0.608866
16	spam_scaled	1	KNeighborsClassifier + GridSearchCV	0.908083

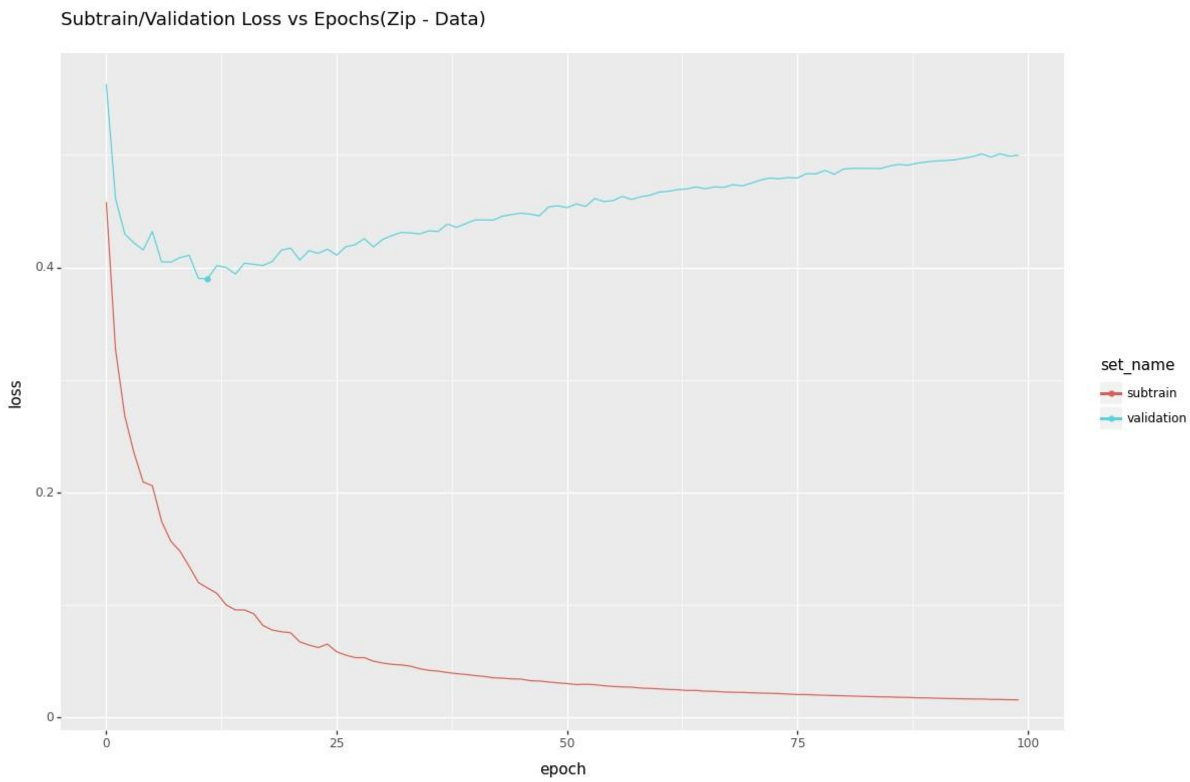
17	spam_scaled	1	LogisticRegressionCV	0.917862
18	spam_scaled	1	MyCV + RegularizedMLP	0.929596
19	spam_scaled	1	featureless	0.601043
20	spam_scaled	2	KNeighborsClassifier + GridSearchCV	0.905414
21	spam_scaled	2	LogisticRegressionCV	0.924984
22	spam_scaled	2	MyCV + RegularizedMLP	0.923679
23	spam_scaled	2	featureless	0.607958

```
>>> 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)
```

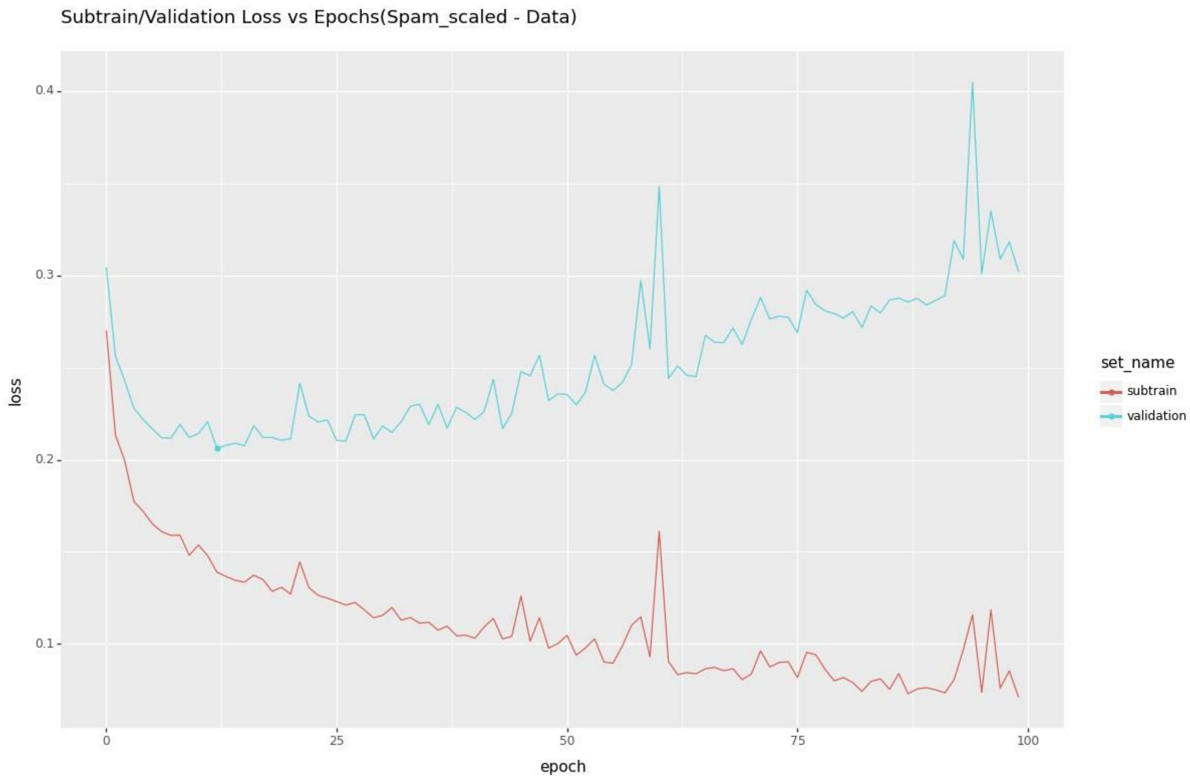



```
>>> gg1 = p9.ggplot() +\
...     p9.geom_line(
...     p9.aes(
...     x = "epoch",
...     y= "loss",
...     color = "set_name"
...     ),
...     data = loss_data_dict["zip"]["my_cv_learner_estimator"]) +\
...     p9.geom_point(
...     p9.aes(
...     x = "epoch",
```

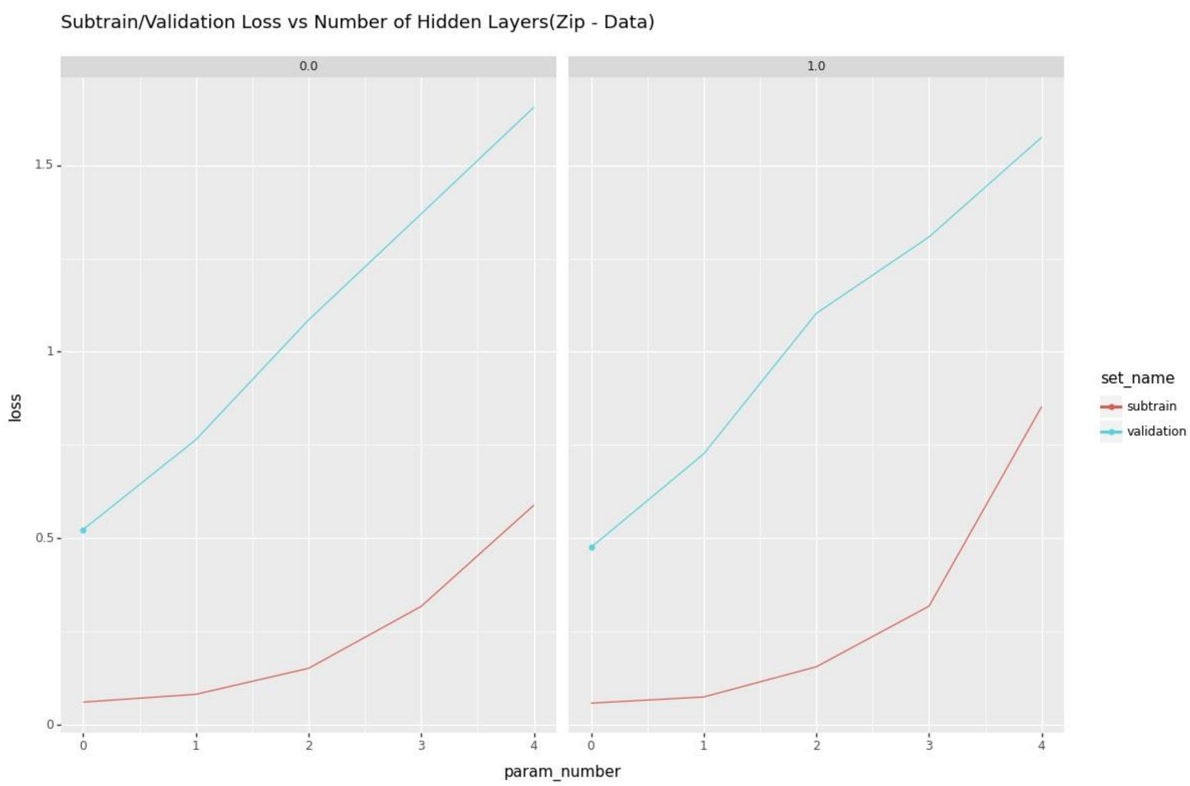
```
...     y = "loss",
...     color = "set_name"
...     ),
...     data = min_df_dict["zip"]["min_df_estimator"]) +\
...     p9.ggtitle("Subtrain/Validation Loss vs Epochs(Zip - Data)")

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



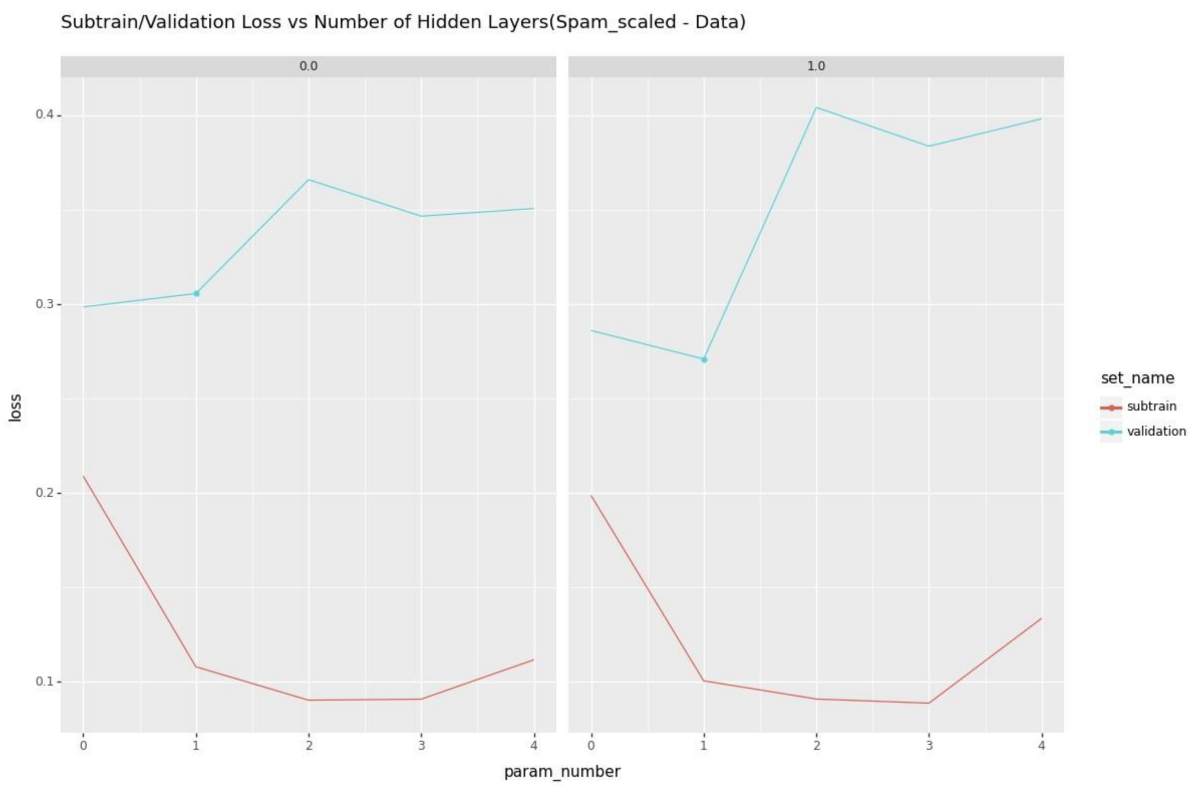

```
>>> gg2 = p9.ggplot() +\
...     p9.geom_line(
...     p9.aes(
...     x = "epoch",
...     y= "loss",
...     color = "set_name"
...     ),
...     data = loss_data_dict["spam_scaled"]["my_cv_learner_estimator"]) +\
...     p9.geom_point(
...     p9.aes(
...     x = "epoch",
...     y = "loss",
...     color = "set_name"
...     ),
...     data = min_df_dict["spam_scaled"]["min_df_estimator"]) +\
...     p9.ggtitle("Subtrain/Validation Loss vs Epochs(Spam_scaled - Data)")

>>> gg2.save("Torch_validation_graph2.png", height = 8, width = 12)
```




```
>>> gg3 = p9.ggplot() +\
...     p9.geom_line(
...     p9.aes(
...     x = "param_number",
...     y= "loss",
...     color = "set_name"
...     ),
...     data = loss_data_dict["zip"]["my_cv_learner"]) +\
...     p9.geom_point(
...     p9.aes(
...     x = "param_number",
...     y = "loss",
...     color = "set_name"
...     ),
...     data = min_df_dict["zip"]["min_df"]) +\
...     p9.facet_grid('.~validation_fold') +\
...     p9.ggtitle("Subtrain/Validation Loss vs Number of Hidden Layers(Zip - Data)")

>>> gg3.save("01loss_graph1.png", height = 8, width = 12)
```




```
>>> gg4 = p9.ggplot() +\
...     p9.geom_line(
...     p9.aes(
...     x = "param_number",
...     y= "loss",
...     color = "set_name"
...     ),
...     data = loss_data_dict["spam_scaled"]["my_cv_learner"]) +\
...     p9.geom_point(
...     p9.aes(
...     x = "param_number",
...     y = "loss",
...     color = "set_name"
...     ),
...     data = min_df_dict["spam_scaled"]["min_df"]) +\
...     p9.facet_grid('.~validation_fold') +\
...     p9.ggtitle("Subtrain/Validation Loss vs Number of Hidden Layers(Spam_scaled - Data)")

>>> gg4.save("01loss_graph2.png", height = 8, width = 12)
```


Subtrain/Validation Loss vs Number of Hidden Layers(Spam_scaled - Data)
0.0
1.0
0.4
0.3
0.2
0.1
loss
0
1
2
3
4
param_number
set_name
subtrain
validation