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)
```

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.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)

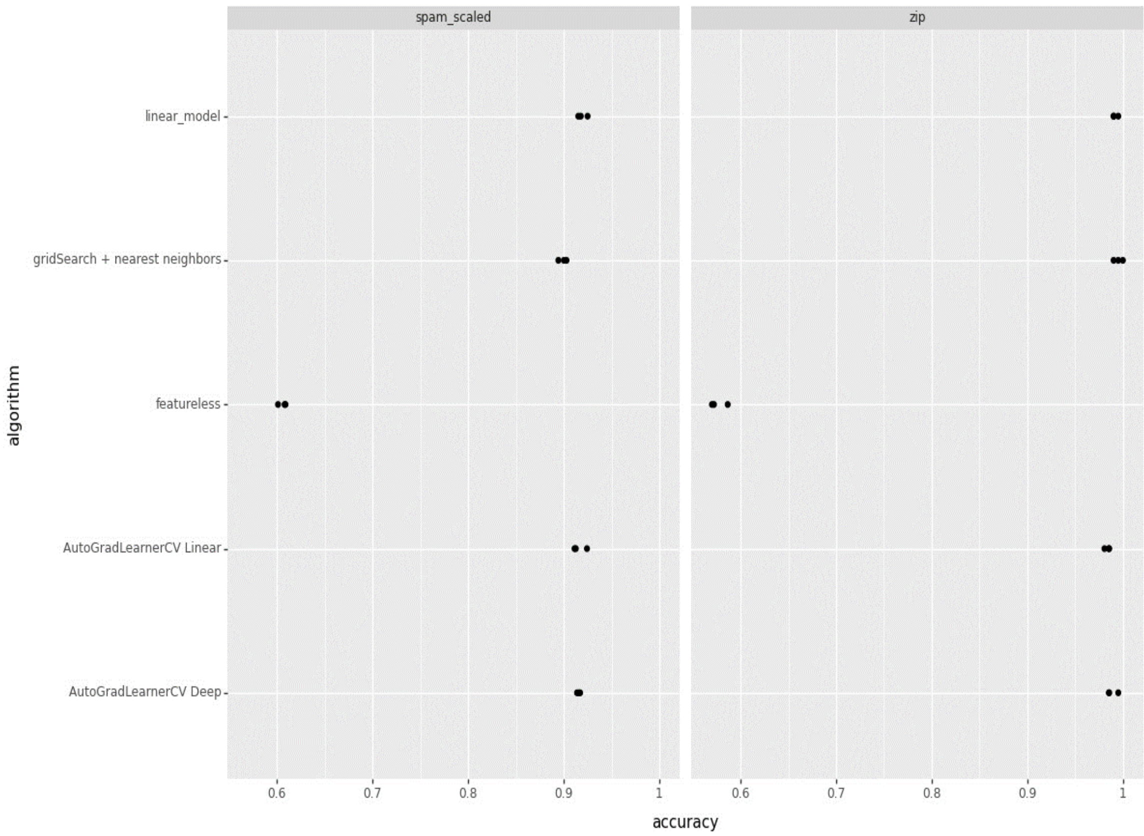
>>> print(total_accuracy_df)

	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:

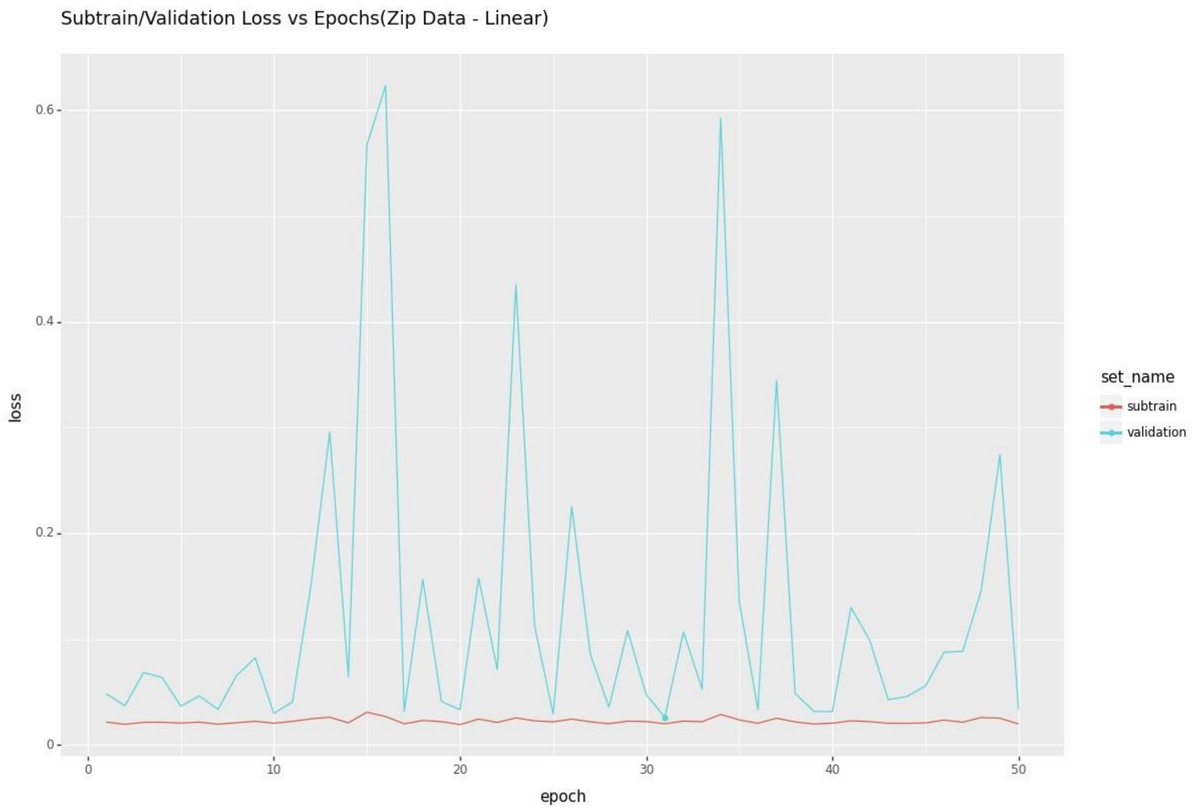
```
>>> gg4 = p9.ggplot(total_accuracy_df, p9.aes(x ='accuracy', y = 'algorithm'))+\
...     p9.facet_grid('.~data_set') + p9.geom_point()
>>> gg4.save("Accuracy_graph.png", height = 8, width = 12)
```



Linear subtrain/validation Loss graph (Zip):

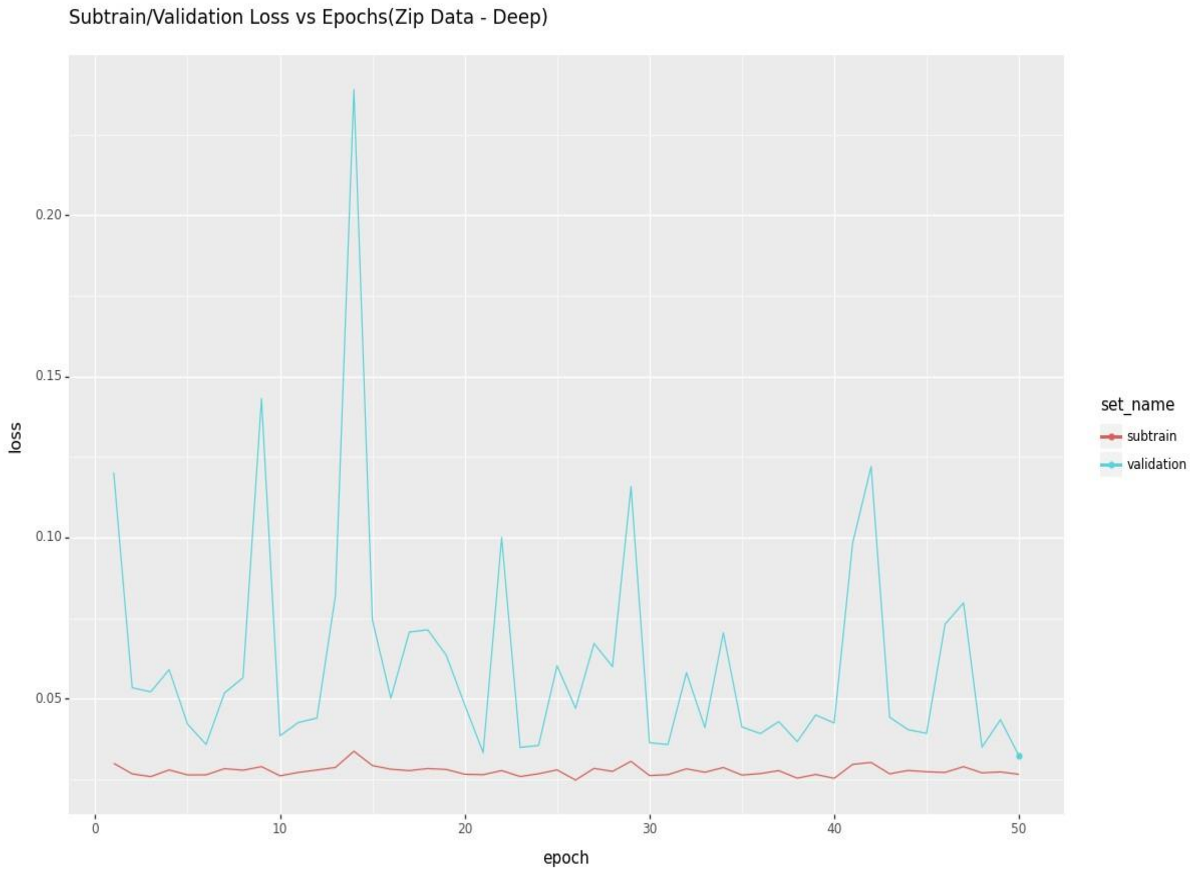
```
>>> gg = p9.ggplot() +\
...     p9.geom_line(
...         p9.aes(
...             x = "epoch",
...             y= "loss",
...             color = "set_name"
...         ),
...         data = zip_loss["AutoGradLearnerCV Linear"]) +\
...     p9.geom_point(
...         p9.aes(
...             x = "epoch",
...             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)
```



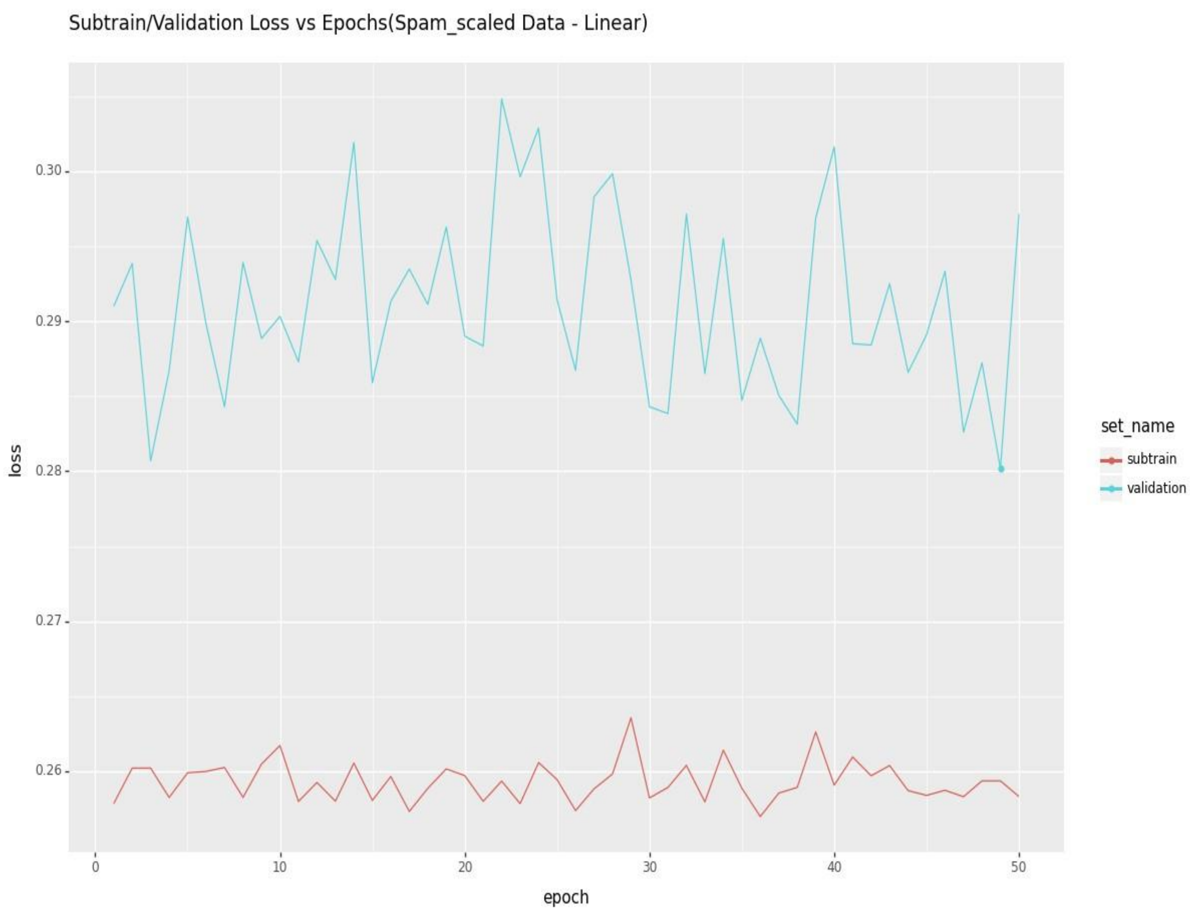
Subtrain/Validation Loss Graph (Zip Data – Deep Model):

```
>>> 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):