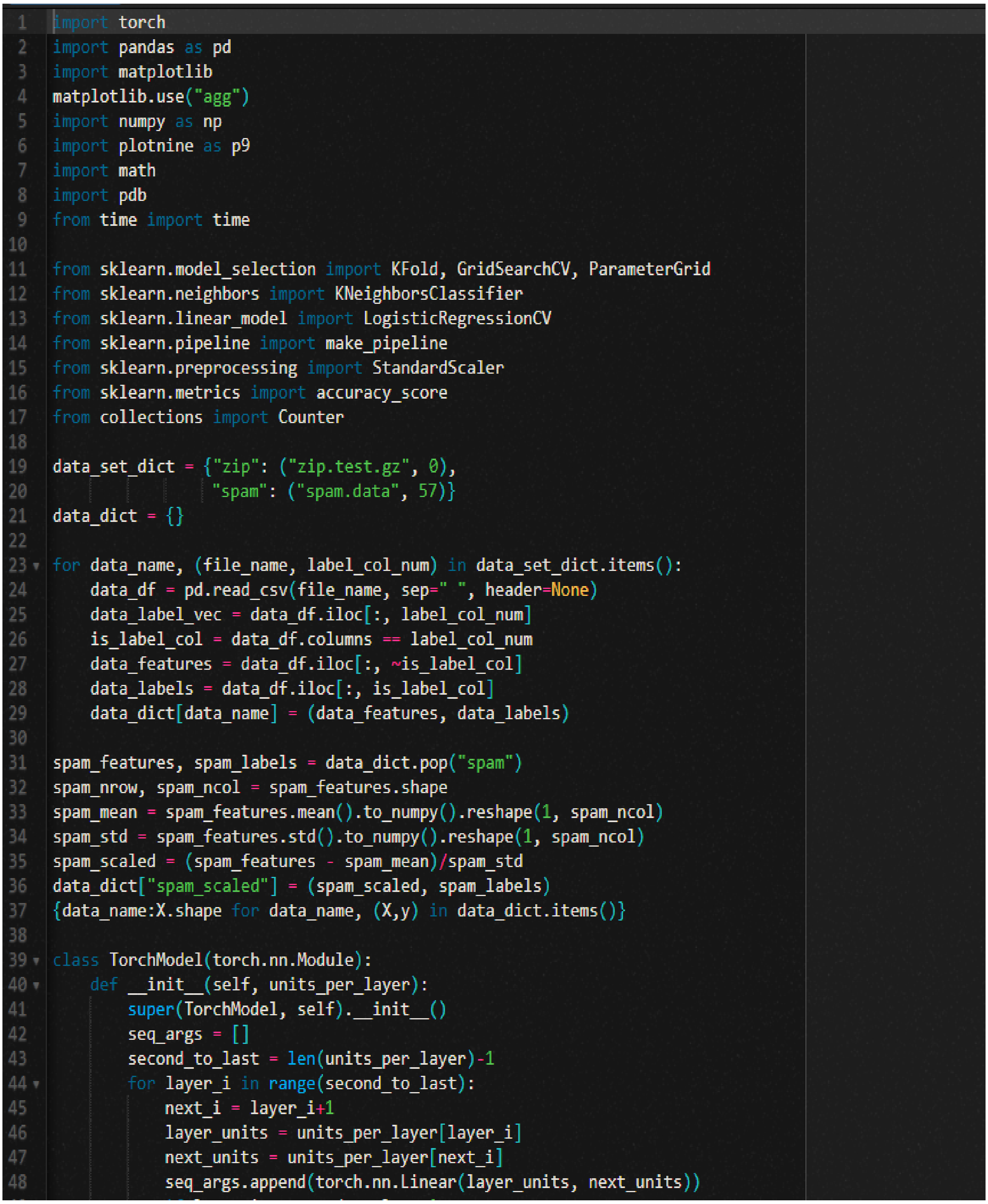
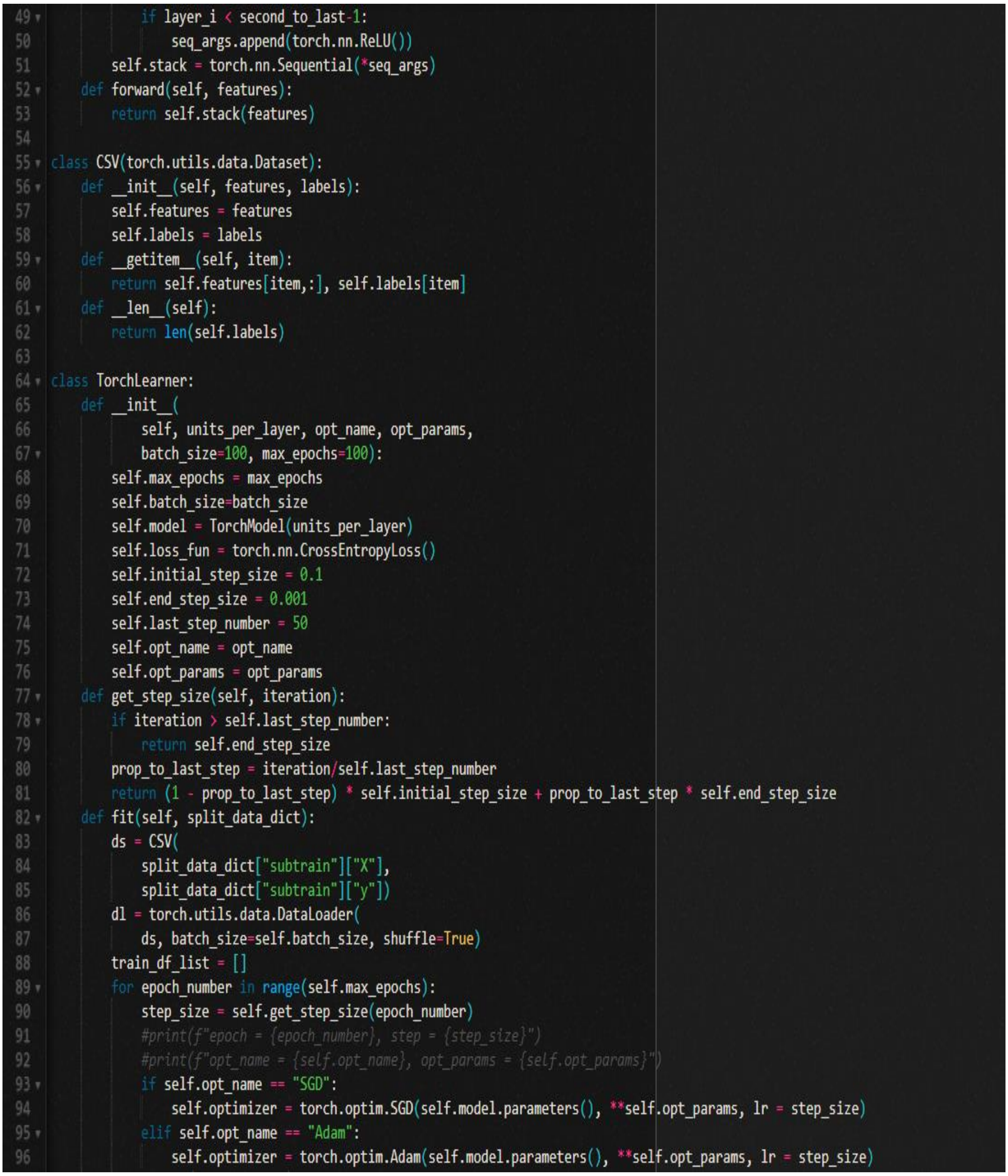
CS599 (Deep Learning)

Homework – 14

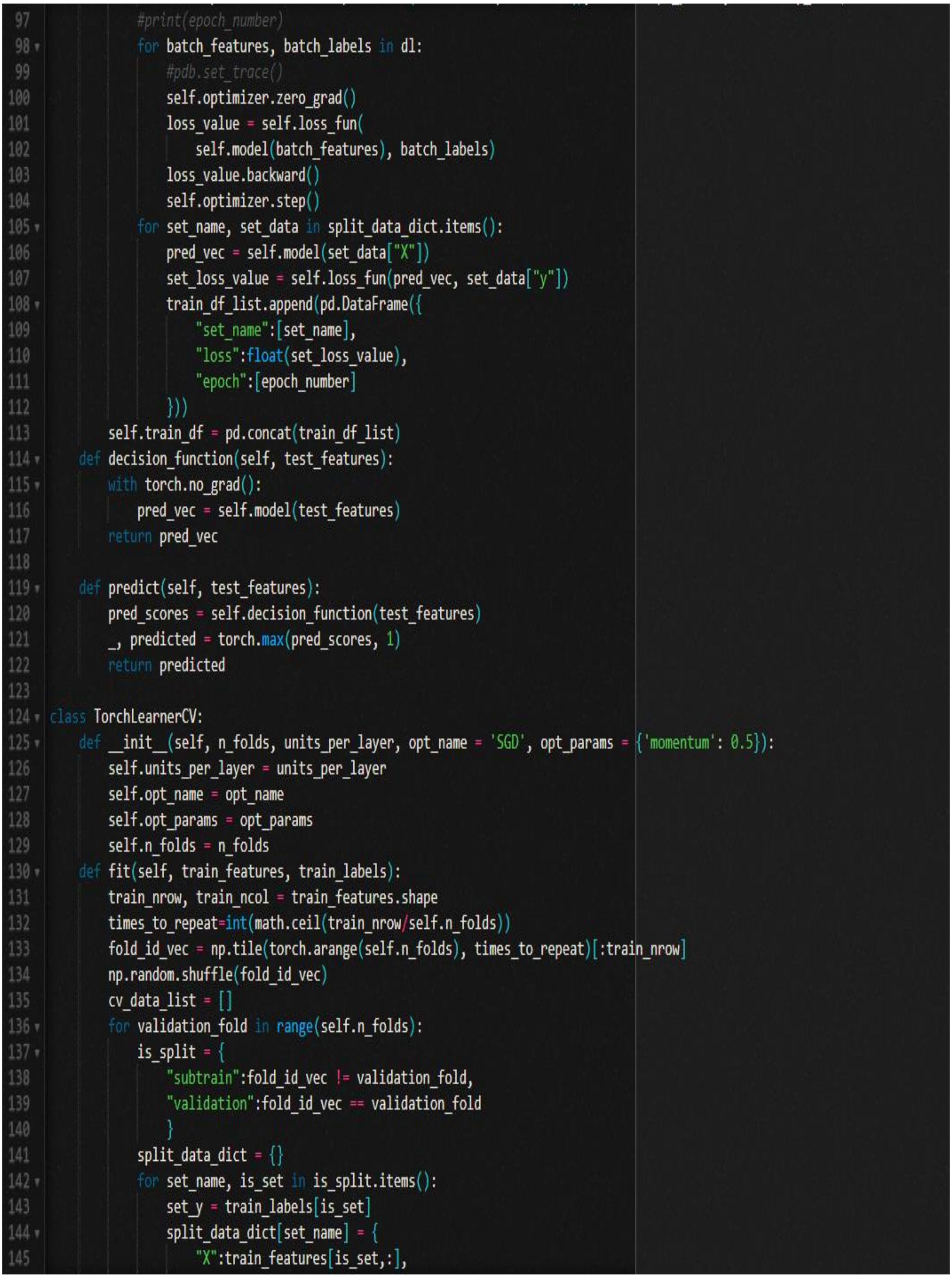
1. **Python Code:**



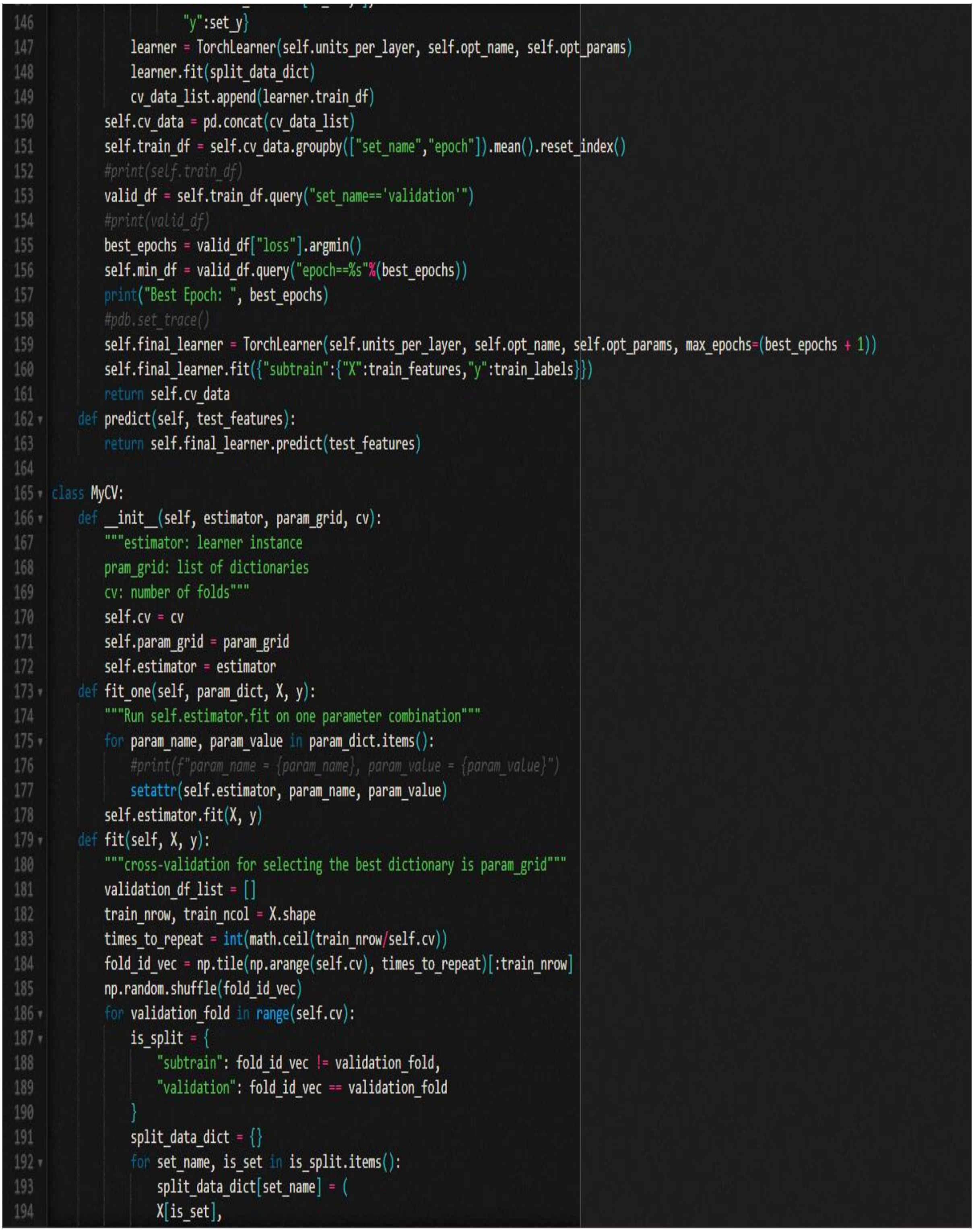
```python
import torch
import pandas as pd
import matplotlib
matplotlib.use("agg")
import numpy as np
import plotnine as p9
import math
import pdb
from time import time

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))
```



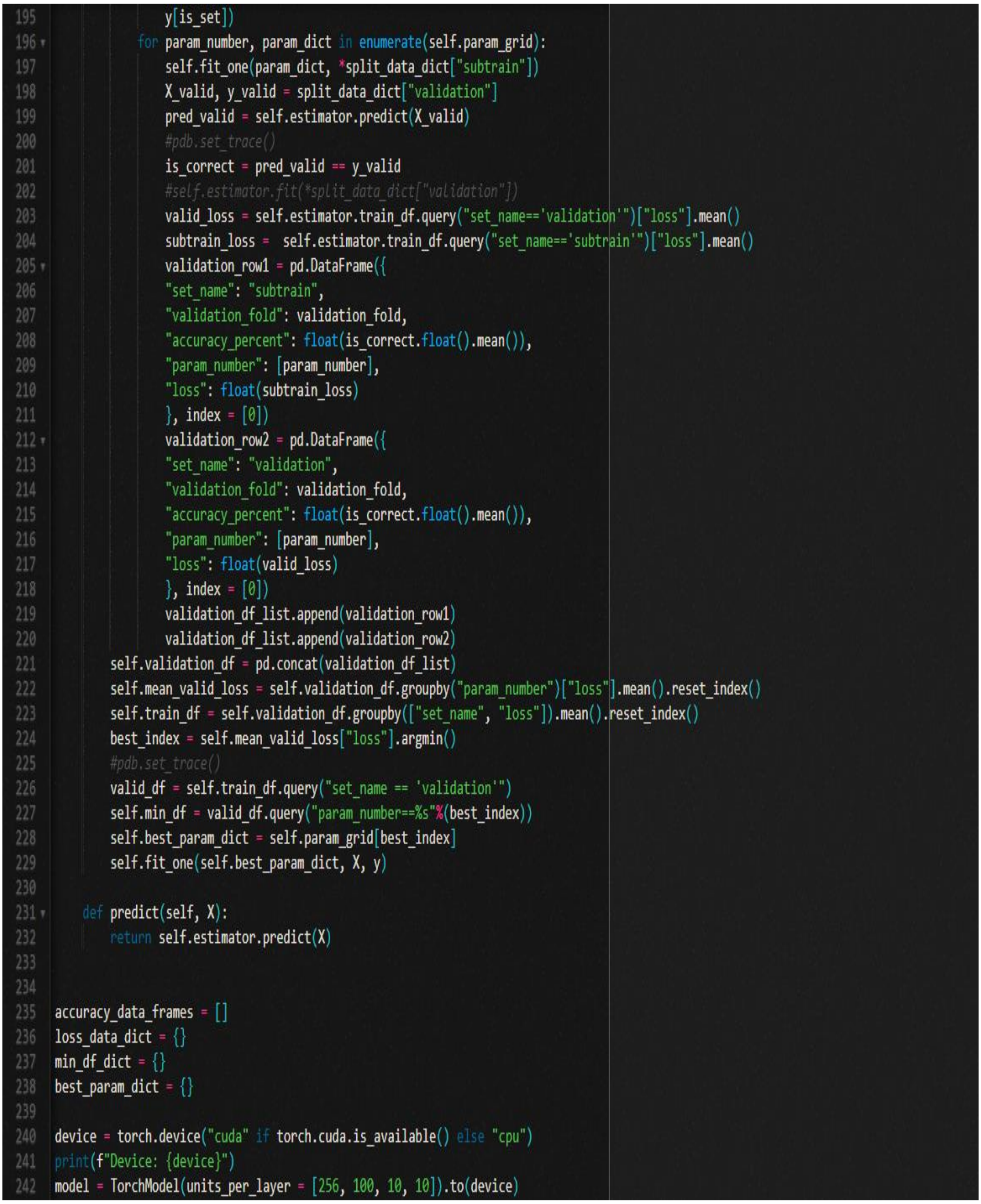
```python
            if layer_i < second_to_last-1:
                seq_args.append(torch.nn.ReLU())
        self.stack = torch.nn.Sequential(*seq_args)
    def forward(self, features):
        return self.stack(features)

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, opt_name, opt_params,
            batch_size=100, 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.initial_step_size = 0.1
        self.end_step_size = 0.001
        self.last_step_number = 50
        self.opt_name = opt_name
        self.opt_params = opt_params
    def get_step_size(self, iteration):
        if iteration > self.last_step_number:
            return self.end_step_size
        prop_to_last_step = iteration/self.last_step_number
        return (1 - prop_to_last_step) * self.initial_step_size + prop_to_last_step * self.end_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):
            step_size = self.get_step_size(epoch_number)
            #print(f"epoch = {epoch_number}, step = {step_size}")
            #print(f"opt_name = {self.opt_name}, opt_params = {self.opt_params}")
            if self.opt_name == "SGD":
                self.optimizer = torch.optim.SGD(self.model.parameters(), **self.opt_params, lr = step_size)
            elif self.opt_name == "Adam":
                self.optimizer = torch.optim.Adam(self.model.parameters(), **self.opt_params, lr = step_size)
```



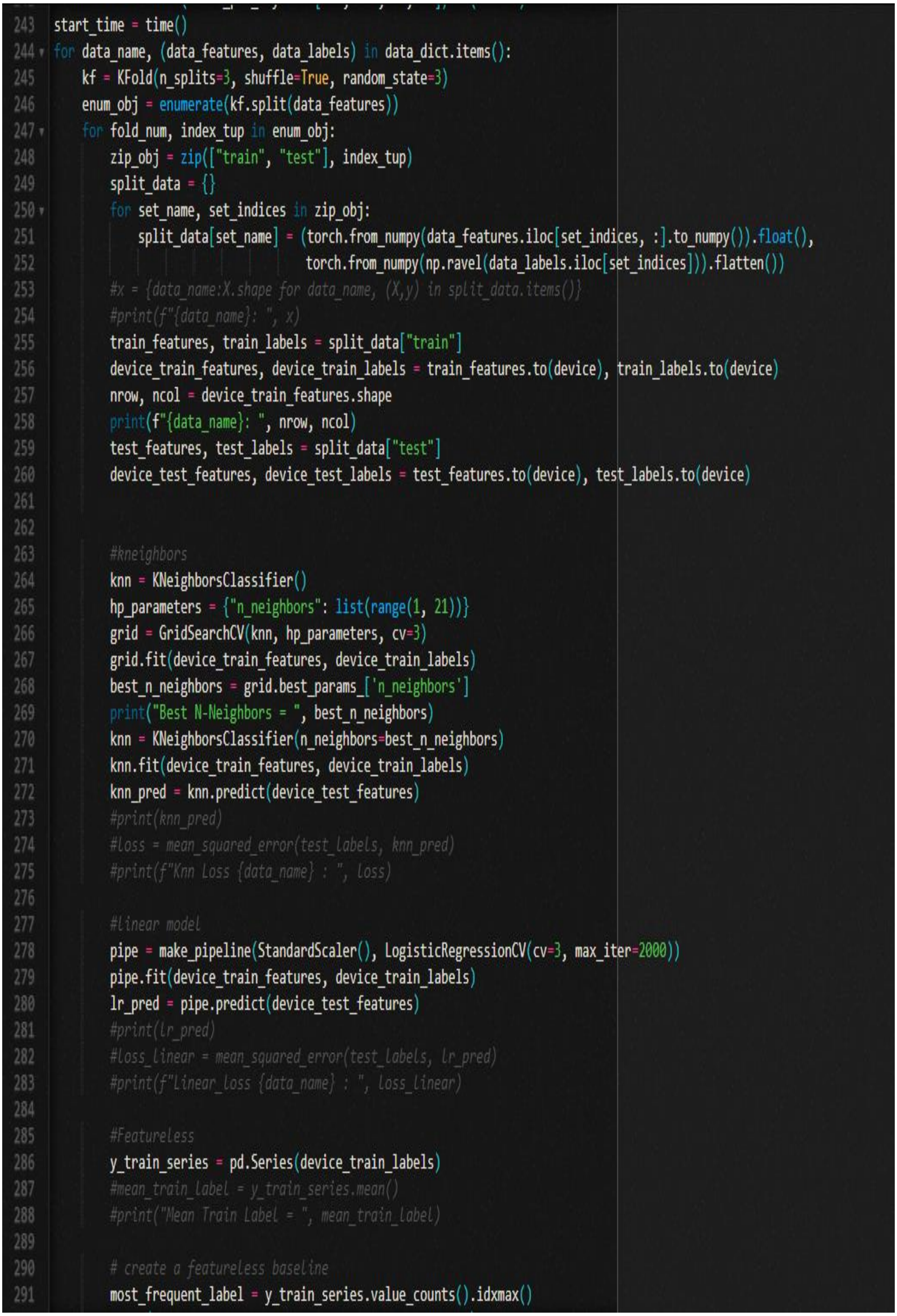
```
            #print(epoch_number)
            for batch_features, batch_labels in dl:
                #pdb.set_trace()
                self.optimizer.zero_grad()
                loss_value = self.loss_fun(
                    self.model(batch_features), batch_labels)
                loss_value.backward()
                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, opt_name = 'SGD', opt_params = {'momentum': 0.5}):
        self.units_per_layer = units_per_layer
        self.opt_name = opt_name
        self.opt_params = opt_params
        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,:],
```



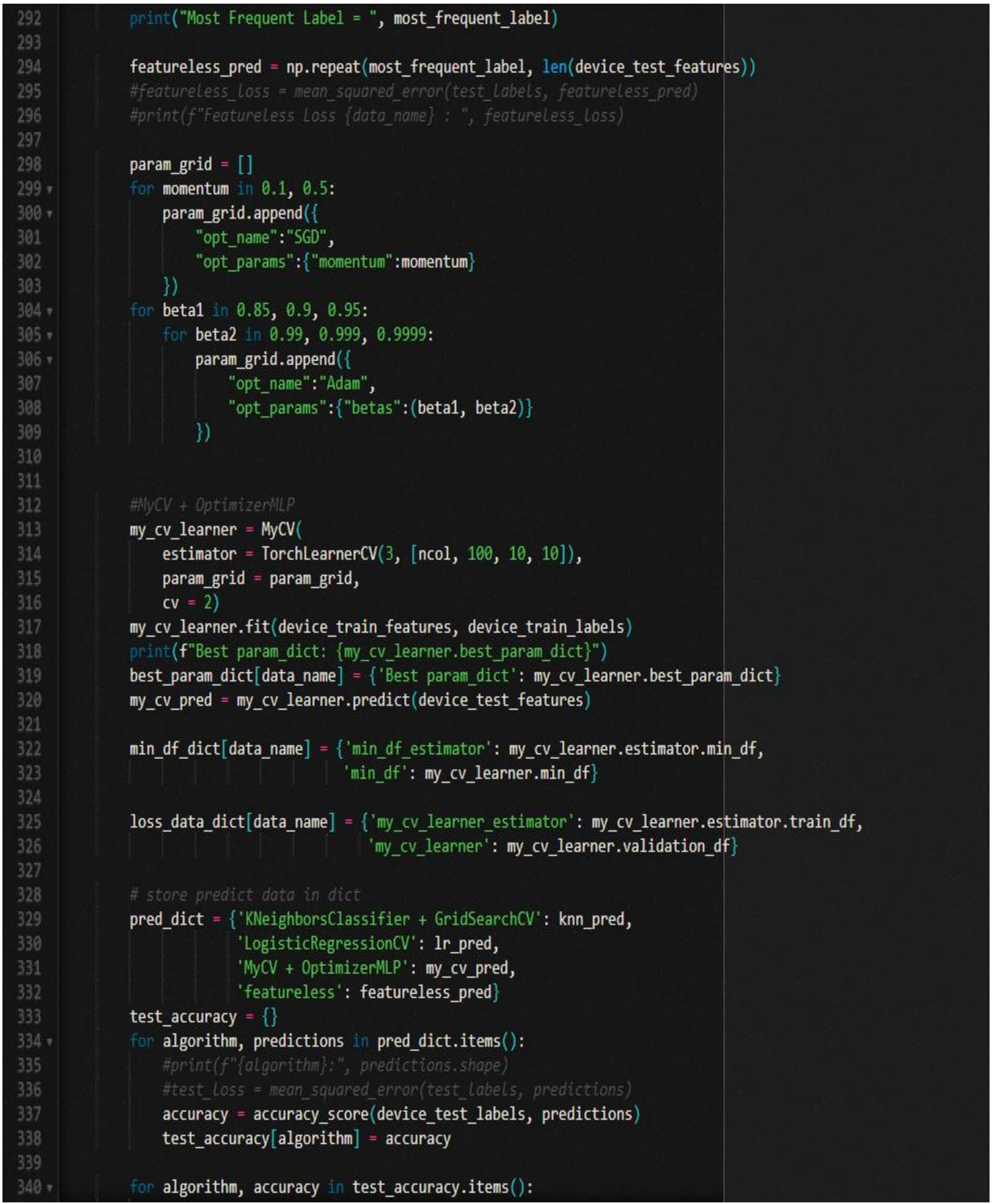
```python
                    "y":set_y}
            learner = TorchLearner(self.units_per_layer, self.opt_name, self.opt_params)
            learner.fit(split_data_dict)
            cv_data_list.append(learner.train_df)
        self.cv_data = pd.concat(cv_data_list)
        self.train_df = self.cv_data.groupby(["set_name","epoch"]).mean().reset_index()
        #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)
        #pdb.set_trace()
        self.final_learner = TorchLearner(self.units_per_layer, self.opt_name, self.opt_params, 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():
            #print(f"param_name = {param_name}, param_value = {param_value}")
            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],
```



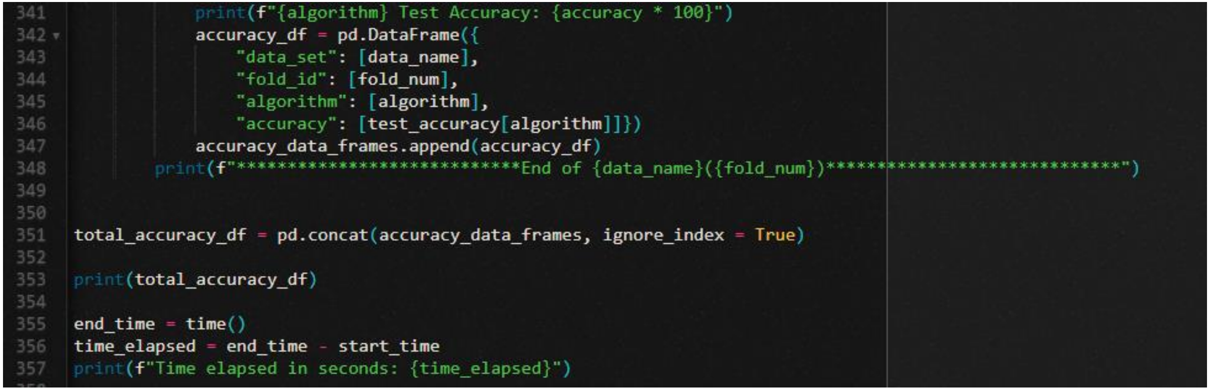
```python
                y[is_set])
            for param_number, param_dict in enumerate(self.param_grid):
                self.fit_one(param_dict, *split_data_dict["subtrain"])
                X_valid, y_valid = split_data_dict["validation"]
                pred_valid = self.estimator.predict(X_valid)
                #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 = {}
best_param_dict = {}

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Device: {device}")
model = TorchModel(units_per_layer = [256, 100, 10, 10]).to(device)
```



```
start_time = time()
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"]
        device_train_features, device_train_labels = train_features.to(device), train_labels.to(device)
        nrow, ncol = device_train_features.shape
        print(f"{data_name}: ", nrow, ncol)
        test_features, test_labels = split_data["test"]
        device_test_features, device_test_labels = test_features.to(device), test_labels.to(device)



        #kneighbors
        knn = KNeighborsClassifier()
        hp_parameters = {"n_neighbors": list(range(1, 21))}
        grid = GridSearchCV(knn, hp_parameters, cv=3)
        grid.fit(device_train_features, device_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(device_train_features, device_train_labels)
        knn_pred = knn.predict(device_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(device_train_features, device_train_labels)
        lr_pred = pipe.predict(device_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(device_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(device_test_features))
        #featureless_loss = mean_squared_error(test_labels, featureless_pred)
        #print(f"Featureless Loss {data_name} : ", featureless_loss)

        param_grid = []
        for momentum in 0.1, 0.5:
            param_grid.append({
                "opt_name":"SGD",
                "opt_params":{"momentum":momentum}
            })
        for beta1 in 0.85, 0.9, 0.95:
            for beta2 in 0.99, 0.999, 0.9999:
                param_grid.append({
                    "opt_name":"Adam",
                    "opt_params":{"betas":(beta1, beta2)}
                })


        #MyCV + OptimizerMLP
        my_cv_learner = MyCV(
            estimator = TorchLearnerCV(3, [ncol, 100, 10, 10]),
            param_grid = param_grid,
            cv = 2)
        my_cv_learner.fit(device_train_features, device_train_labels)
        print(f"Best param_dict: {my_cv_learner.best_param_dict}")
        best_param_dict[data_name] = {'Best param_dict': my_cv_learner.best_param_dict}
        my_cv_pred = my_cv_learner.predict(device_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 + OptimizerMLP': 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(device_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)

end_time = time()
time_elapsed = end_time - start_time
print(f"Time elapsed in seconds: {time_elapsed}")
```

2. **Output:**

```
(cs599fall2023) [sd2554@wind ~ ]$ time srun -t 1:00:00 --gres=gpu:tesla:1 --mem=8GB
python HW14.py
Device: cpu
zip:  1338 256
Best N-Neighbors =  1
Most Frequent Label =  0
Best Epoch:  17
Best Epoch:  10
Best Epoch:  15
Best Epoch:  16
Best Epoch:  13
Best Epoch:  12
Best Epoch:  22
Best Epoch:  0
Best Epoch:  11
Best Epoch:  37
Best Epoch:  12
Best Epoch:  11
Best Epoch:  11
Best Epoch:  40
Best Epoch:  10
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  2
Best Epoch:  26
Best Epoch:  0
Best Epoch:  29
Best Epoch:  6
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.1}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 90.5829596412556
LogisticRegressionCV Test Accuracy: 89.8355754857997
MyCV + OptimizerMLP Test Accuracy: 89.53662182361734
featureless Test Accuracy: 18.53512705530643
****************************End of zip(0)*****************************
zip:  1338 256
Best N-Neighbors =  1
Most Frequent Label =  0
Best Epoch:  8
Best Epoch:  7
Best Epoch:  0
Best Epoch:  22
Best Epoch:  14
Best Epoch:  3
Best Epoch:  25
Best Epoch:  2
Best Epoch:  21
Best Epoch:  36
Best Epoch:  4
Best Epoch:  12
Best Epoch:  10
Best Epoch:  3
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  3
Best Epoch:  1
Best Epoch:  7
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.1}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 91.18086696562034
LogisticRegressionCV Test Accuracy: 88.34080717488789
MyCV + OptimizerMLP Test Accuracy: 89.53662182361734
featureless Test Accuracy: 17.638266068759343
****************************End of zip(1)*****************************
zip:  1338 256
Best N-Neighbors =  1
Most Frequent Label =  0
Best Epoch:  13
Best Epoch:  10
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  2
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  8
Best Epoch:  9
Best Epoch:  11
Best Epoch:  3
Best Epoch:  7
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  34
Best Epoch:  5
Best Epoch:  12
Best Epoch:  2
Best Epoch:  7
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.5}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 89.98505231689087
LogisticRegressionCV Test Accuracy: 89.98505231689087
MyCV + OptimizerMLP Test Accuracy: 88.49028400597906
featureless Test Accuracy: 17.48878923766816
****************************End of zip(2)*****************************
spam_scaled:  3067 57
Best N-Neighbors =  4
Most Frequent Label =  0
Best Epoch:  7
Best Epoch:  3
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  0
Best Epoch:  0
Best Epoch:  10
Best Epoch:  8
Best Epoch:  1
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  2
Best Epoch:  3
Best Epoch:  1
Best Epoch:  0
Best Epoch:  0
Best Epoch:  4
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.1}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 88.72229465449804
LogisticRegressionCV Test Accuracy: 91.52542372881356
MyCV + OptimizerMLP Test Accuracy: 93.93741851368969
featureless Test Accuracy: 60.88657105606258
****************************End of spam_scaled(0)*****************************
spam_scaled:  3067 57
Best N-Neighbors =  5
Most Frequent Label =  0
Best Epoch:  3
Best Epoch:  3
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  0
Best Epoch:  1
Best Epoch:  8
Best Epoch:  7
Best Epoch:  0
Best Epoch:  1
Best Epoch:  1
Best Epoch:  2
Best Epoch:  2
Best Epoch:  1
Best Epoch:  5
Best Epoch:  0
Best Epoch:  1
Best Epoch:  10
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.1}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 90.80834419817471
LogisticRegressionCV Test Accuracy: 91.78617992177314
MyCV + OptimizerMLP Test Accuracy: 94.0677966101695
featureless Test Accuracy: 60.104302477183836
****************************End of spam_scaled(1)*****************************
spam_scaled:  3068 57
Best N-Neighbors =  9
Most Frequent Label =  0
Best Epoch:  7
Best Epoch:  3
Best Epoch:  0
Best Epoch:  1
Best Epoch:  2
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  3
Best Epoch:  0
Best Epoch:  0
Best Epoch:  3
Best Epoch:  3
Best Epoch:  1
Best Epoch:  0
Best Epoch:  1
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  2
Best Epoch:  1
Best Epoch:  6
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.1}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 90.54142204827136
LogisticRegressionCV Test Accuracy: 92.49836921069797
MyCV + OptimizerMLP Test Accuracy: 92.82452707110241
featureless Test Accuracy: 60.79582517938682
****************************End of spam_scaled(2)*****************************
     data_set        fold_id                  algorithm                     accuracy
0      zip              0        KNeighborsClassifier + GridSearchCV     0.905830
1      zip              0        LogisticRegressionCV                    0.898356
2      zip              0        MyCV + OptimizerMLP                     0.895366
3      zip              0        featureless                             0.185351
4      zip              1        KNeighborsClassifier + GridSearchCV     0.911809
5      zip              1        LogisticRegressionCV                    0.883408
6      zip              1        MyCV + OptimizerMLP                     0.895366
7      zip              1        featureless                             0.176383
8      zip              2        KNeighborsClassifier + GridSearchCV     0.899851
9      zip              2        LogisticRegressionCV                    0.899851
10      zip             2        MyCV + OptimizerMLP                     0.884903
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 + OptimizerMLP                     0.939374
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 + OptimizerMLP                     0.940678
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 + OptimizerMLP                     0.928245
23  spam_scaled         2        featureless                             0.607958
Time elapsed in seconds: 2025.400666475296

real    33m48.993s
user    0m0.011s
sys     0m0.010s

(cs599fall2023) [sd2554@wind ~ ]$ time srun -t 1:00:00 --mem=8GB  python HW14.py
srun: job 6702290 queued and waiting for resources
srun: job 6702290 has been allocated resources
Device: cpu
zip:  1338 256
Best N-Neighbors =  1
Most Frequent Label =  0
Best Epoch:  9
Best Epoch:  4
Best Epoch:  16
Best Epoch:  4
Best Epoch:  0
Best Epoch:  5
Best Epoch:  15
Best Epoch:  0
Best Epoch:  6
Best Epoch:  3
Best Epoch:  14
Best Epoch:  11
Best Epoch:  11
Best Epoch:  8
Best Epoch:  33
Best Epoch:  19
Best Epoch:  46
Best Epoch:  37
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  4
Best Epoch:  10
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.1}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 90.5829596412556
LogisticRegressionCV Test Accuracy: 89.8355754857997
MyCV + OptimizerMLP Test Accuracy: 89.53662182361734
featureless Test Accuracy: 18.53512705530643
****************************End of zip(0)*****************************
zip:  1338 256
Best N-Neighbors =  1
Most Frequent Label =  0
Best Epoch:  15
Best Epoch:  6
Best Epoch:  9
Best Epoch:  3
Best Epoch:  11
Best Epoch:  18
Best Epoch:  22
Best Epoch:  2
Best Epoch:  1
Best Epoch:  4
Best Epoch:  25
Best Epoch:  11
Best Epoch:  9
Best Epoch:  10
Best Epoch:  1
Best Epoch:  4
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  1
Best Epoch:  0
Best Epoch:  2
Best Epoch:  11
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.1}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 91.18086696562034
LogisticRegressionCV Test Accuracy: 88.34080717488789
MyCV + OptimizerMLP Test Accuracy: 89.98505231689087
featureless Test Accuracy: 17.638266068759343
****************************End of zip(1)*****************************
zip:  1338 256
Best N-Neighbors =  1
Most Frequent Label =  0
Best Epoch:  10
Best Epoch:  9
Best Epoch:  11
Best Epoch:  11
Best Epoch:  1
Best Epoch:  6
Best Epoch:  3
Best Epoch:  0
Best Epoch:  21
Best Epoch:  18
Best Epoch:  19
Best Epoch:  17
Best Epoch:  7
Best Epoch:  13
Best Epoch:  2
Best Epoch:  0
Best Epoch:  39
Best Epoch:  2
Best Epoch:  34
Best Epoch:  2
Best Epoch:  18
Best Epoch:  10
Best Epoch:  13
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.1}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 89.98505231689087
LogisticRegressionCV Test Accuracy: 89.98505231689087
MyCV + OptimizerMLP Test Accuracy: 91.03139013452915
featureless Test Accuracy: 17.48878923766816
****************************End of zip(2)*****************************
spam_scaled:  3067 57
Best N-Neighbors =  4
Most Frequent Label =  0
Best Epoch:  8
Best Epoch:  7
Best Epoch:  1
Best Epoch:  5
Best Epoch:  2
Best Epoch:  3
Best Epoch:  0
Best Epoch:  2
Best Epoch:  3
Best Epoch:  0
Best Epoch:  0
Best Epoch:  5
Best Epoch:  4
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  0
Best Epoch:  0
Best Epoch:  7
Best Epoch:  1
Best Epoch:  0
Best Epoch:  6
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.1}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 88.72229465449804
LogisticRegressionCV Test Accuracy: 91.52542372881356
MyCV + OptimizerMLP Test Accuracy: 93.67666232073012
featureless Test Accuracy: 60.88657105606258
****************************End of spam_scaled(0)*****************************
spam_scaled:  3067 57
Best N-Neighbors =  5
Most Frequent Label =  0
Best Epoch:  9
Best Epoch:  5
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  0
Best Epoch:  1
Best Epoch:  2
Best Epoch:  1
Best Epoch:  0
Best Epoch:  0
Best Epoch:  3
Best Epoch:  1
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  1
Best Epoch:  5
Best Epoch:  0
Best Epoch:  0
Best Epoch:  1
Best Epoch:  1
Best Epoch:  4
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.1}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 90.80834419817471
LogisticRegressionCV Test Accuracy: 91.78617992177314
MyCV + OptimizerMLP Test Accuracy: 93.08996088657105
featureless Test Accuracy: 60.104302477183836
****************************End of spam_scaled(1)******************************
spam_scaled:  3068 57
Best N-Neighbors =  9
Most Frequent Label =  0
Best Epoch:  4
Best Epoch:  4
Best Epoch:  0
Best Epoch:  1
Best Epoch:  2
Best Epoch:  1
Best Epoch:  0
Best Epoch:  1
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  8
Best Epoch:  6
Best Epoch:  1
Best Epoch:  0
Best Epoch:  1
Best Epoch:  0
Best Epoch:  2
Best Epoch:  1
Best Epoch:  0
Best Epoch:  0
Best Epoch:  0
Best Epoch:  8
Best param_dict: {'opt_name': 'SGD', 'opt_params': {'momentum': 0.1}}
KNeighborsClassifier + GridSearchCV Test Accuracy: 90.54142204827136
LogisticRegressionCV Test Accuracy: 92.49836921069797
MyCV + OptimizerMLP Test Accuracy: 91.78082191780823
featureless Test Accuracy: 60.79582517938682
****************************End of spam_scaled(2)******************************


   data_set   fold_id                    algorithm   accuracy
0       zip         0   KNeighborsClassifier + GridSearchCV   0.905830
1       zip         0   LogisticRegressionCV                  0.898356
2       zip         0   MyCV + OptimizerMLP                   0.895366
3       zip         0   featureless                           0.185351
```

```
4         zip            1         KNeighborsClassifier + GridSearchCV     0.911809
5         zip            1         LogisticRegressionCV                    0.883408
6         zip            1         MyCV + OptimizerMLP                     0.899851
7         zip            1         featureless                             0.176383
8         zip            2         KNeighborsClassifier + GridSearchCV     0.899851
9         zip            2         LogisticRegressionCV                    0.899851
10        zip            2         MyCV + OptimizerMLP                     0.910314
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 + OptimizerMLP                     0.936767
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 + OptimizerMLP                     0.930900
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 + OptimizerMLP                     0.917808
23  spam_scaled          2         featureless                             0.607958
Time elapsed in seconds: 2051.2660534381866

real    34m14.617s
user    0m0.011s
sys     0m0.009s
```

- Both CPU & GPU have similar accuracy.