CS599 (Deep Learning)

Homework – 5

1. **Python Code:**

```
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)
    # scaling the data
    n_data_features = data_features.shape[1]
    data_mean = data_features.mean().to_numpy().reshape(1, n_data_features)
    data_std = data_features.std().to_numpy().reshape(1, n_data_features)
    data_scaled = (data_features - data_mean) / data_std
    data_name_scaled = data_name + "_scaled"
    data_scaled = data_scaled.dropna(axis="columns")
    data_dict[data_name_scaled] = (data_scaled, data_labels)
    #print(data_scaled)

data_dict.pop("zip_scaled")
data_dict.pop("spam")
```

```
class MyLogReg:

    def __init__(self, max_iterations, step_size):
        self.max_iterations = max_iterations
        self.step_size = step_size



    def fit(self, X, y):
        self.X_train = X
        self.y_train = y
        data_nrow, data_ncol = X.shape
        self.intercept_ = 0.0
        self.weight_vec = np.repeat(0.0, data_ncol).reshape(data_ncol,1)  # Initialize weight
vector with zeros
        for i in range(self.max_iterations):
            data_mat = X
            pred_vec = np.matmul(data_mat, self.weight_vec) + self.intercept_
            label_pos_neg_vec = np.where(y == 1, 1, -1).reshape(data_nrow,1)
            grad_loss_wrt_pred = -label_pos_neg_vec / (1 + np.exp(label_pos_neg_vec *
pred_vec))
            loss_vec = np.log(1 + np.exp(-label_pos_neg_vec * pred_vec))
            grad_loss_wrt_weight = np.matmul(data_mat.T, grad_loss_wrt_pred)/data_nrow
            self.weight_vec -= self.step_size * grad_loss_wrt_weight
            self.intercept_ -= self.step_size * grad_loss_wrt_pred.mean()
        return loss_vec.mean()

    def decision_function(self, X):

        return np.matmul(X, self.weight_vec).reshape(X.shape[0], 1) + self.intercept_

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




class MyLogRegCV:

    def __init__(self, max_iterations, step_size, num_splits):

        self.max_iterations = max_iterations
        self.step_size = step_size
        self.num_splits = num_splits

    def fit(self, X, y):
```

```
        kf = KFold(n_splits=self.num_splits, shuffle=True, random_state=3)

        self.scores_ = pd.DataFrame(columns = ["iteration", "set_name", "loss_value"])
        best_loss = float("inf")
        best_lr = None
        for i in range(1, self.max_iterations + 1):

            for validation_fold, (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]}

                lr = MyLogReg(i, self.step_size)
                subtrain_loss = lr.fit(subtrain_data["X"], subtrain_data["y"])
                y_pred = lr.predict(val_data["X"])
                #subtrain_loss = lr.loss_function(subtrain_data["X"], subtrain_data["y"])
                val_loss = lr.fit(val_data["X"], val_data["y"])

                self.scores_ = pd.concat([self.scores_, pd.DataFrame({"iteration": [i], "setname": ["subtrain"], "loss_value": [subtrain_loss]})], ignore_index = True, sort = False)
                self.scores_ = pd.concat([self.scores_, pd.DataFrame({"iteration": [i], "setname": ["validation"], "loss_value": [val_loss]})], ignore_index = True, sort = False)
                accuracy = np.mean(y_pred == val_data["y"])

                subtrain_loss_values = self.scores_[self.scores_["setname"] == "subtrain"]["loss_value"].values
                validation_loss_values = self.scores_[self.scores_["setname"] == "validation"]["loss_value"].values


                #print("Val Loss: ", val_loss)
                #print("Best Loss: ", best_loss)

                if val_loss < best_loss:
                    best_loss = val_loss
                    best_lr = lr
                    self.best_iterations = i
                    #print(self.best_iterations)
        self.lr = MyLogReg(self.best_iterations, self.step_size)
        self.lr.fit(X,y)

        return subtrain_loss_values, validation_loss_values

    def predict(self, X):
        return self.lr.predict(X)
```

```
accuracy_data_frames = []
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, (train_index, test_index) in enum_obj:
      X_train, X_test = np.array(data_features.iloc[train_index]),
np.array(data_features.iloc[test_index])
      y_train, y_test = np.ravel(data_labels.iloc[train_index]),
np.ravel(data_labels.iloc[test_index])

      # K-nearest neighbors
      knn = KNeighborsClassifier()
      hp_parameters = {"n_neighbors": list(range(1, 21))}
      grid = GridSearchCV(knn, hp_parameters, cv=5)
      grid.fit(X_train, y_train)
      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)
      knn_pred = knn.predict(X_test)

      # Logistic Regression
      pipe = make_pipeline(StandardScaler(), LogisticRegressionCV(cv=5, max_iter=2000))
      pipe.fit(X_train, y_train)
      lr_pred = pipe.predict(X_test)
      y_train_series = pd.Series(y_train)

      #MyLogReg + MyLogRegCV
      mylogreg = MyLogRegCV(200, 0.1, 5)
      subtrain_loss, validation_loss = mylogreg.fit(X_train, y_train)
      lr_cv_pred = mylogreg.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, most_frequent_class)

      # store predict data in dict
      pred_dict = {'gridSearch + nearest neighbors': knn_pred,
               'linear_model': lr_pred,
               'MyLogRegCV': lr_cv_pred,
               'featureless': featureless_pred}
      test_accuracy = {}
```

```
        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)

import matplotlib.pyplot as plt
# Number of iterations
iterations = range(1, len(subtrain_loss) + 1)

# Plot subtrain loss
plt.figure(figsize=(8, 6))
plt.subplot(1, 2, 1)
plt.plot(iterations, subtrain_loss, label='Subtrain Loss', color='red')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Subtrain Loss vs. Iterations')
plt.grid(True)

# Plot validation loss
plt.subplot(1, 2, 2)
plt.plot(iterations, validation_loss, label='Validation Loss', color='blue')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Validation Loss vs. Iterations')
plt.grid(True)
```

plt.tight_layout()
plt.savefig("test.png")

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, (train_index, test_index) in enum_obj:
... ...
...   "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
MyLogRegCV Test Accuracy: 99.03846153846155
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
MyLogRegCV 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
MyLogRegCV Test Accuracy: 99.03381642512076
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: 91.39504563233378
MyLogRegCV Test Accuracy: 90.74315514993481
featureless Test Accuracy: 60.88657105606258
****************************End of spam_scaled(0)*****************************
Best N-Neighbors =  6
Most Frequent Class =  0
```

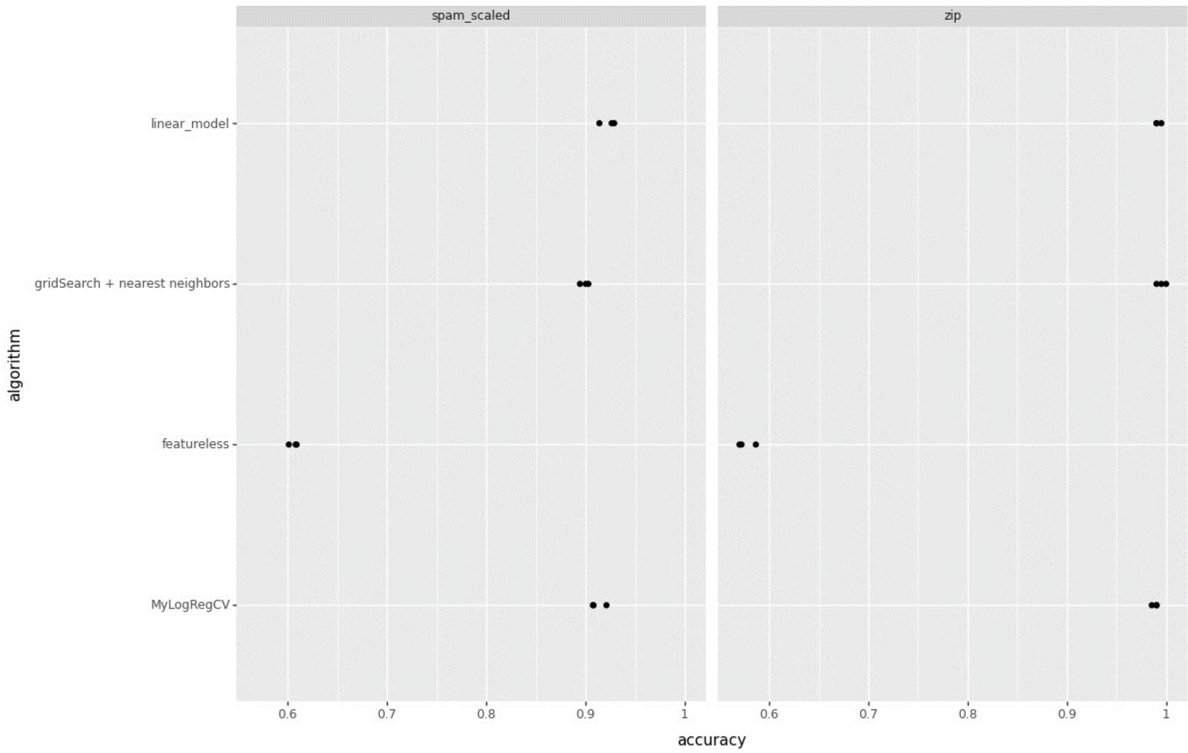
gridSearch + nearest neighbors Test Accuracy: 89.4393741851369
linear_model Test Accuracy: 92.63363754889178
MyLogRegCV Test Accuracy: 90.80834419817471
featureless Test Accuracy: 60.104302477183836
***************************End of spam_scaled(1)*****************************
Best N-Neighbors = 5
Most Frequent Class = 0
gridSearch + nearest neighbors Test Accuracy: 90.01956947162427
linear_model Test Accuracy: 92.8897586431833
MyLogRegCV Test Accuracy: 92.10697977821265
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)

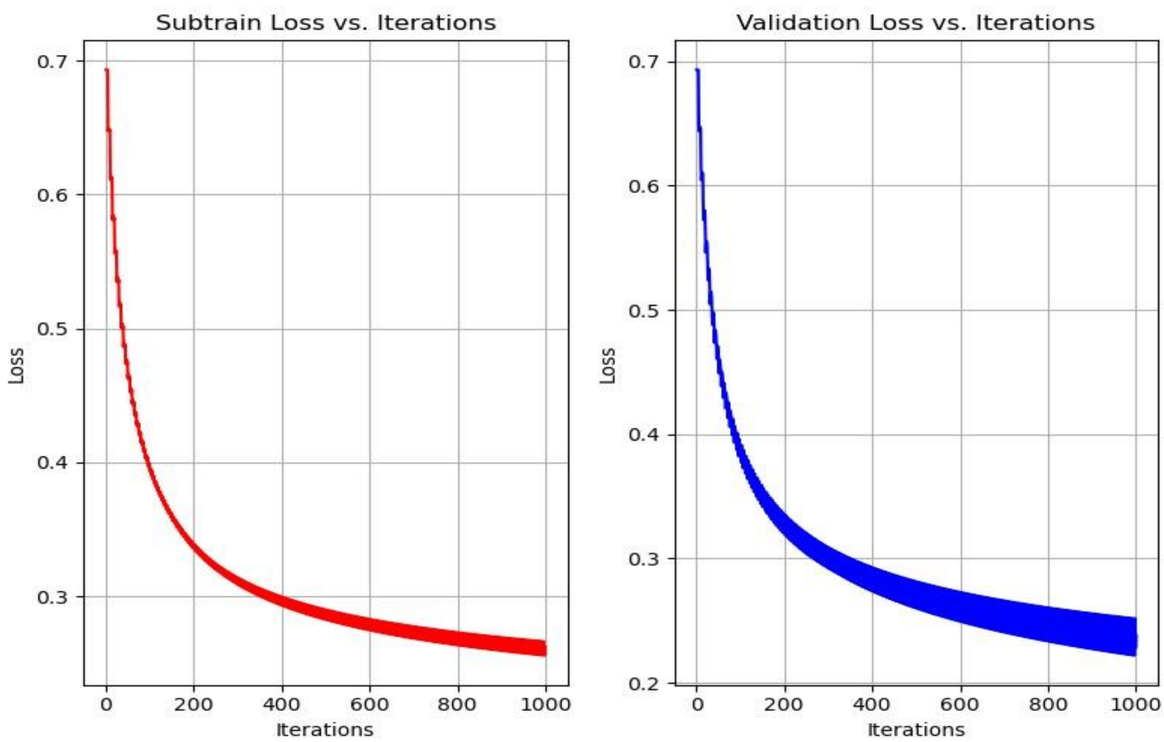
	data_set	fold_id	algorithm	accuracy
0	zip	0	gridSearch + nearest neighbors	1.000000
1	zip	0	linear_model	0.995192
2	zip	0	MyLogRegCV	0.990385
3	zip	0	featureless	0.586538
4	zip	1	gridSearch + nearest neighbors	0.995192
5	zip	1	linear_model	0.990385
6	zip	1	MyLogRegCV	0.985577
7	zip	1	featureless	0.572115
8	zip	2	gridSearch + nearest neighbors	0.990338
9	zip	2	linear_model	0.990338
10	zip	2	MyLogRegCV	0.990338
11	zip	2	featureless	0.570048
12	spam_scaled	0	gridSearch + nearest neighbors	0.902868
13	spam_scaled	0	linear_model	0.913950
14	spam_scaled	0	MyLogRegCV	0.907432
15	spam_scaled	0	featureless	0.608866
16	spam_scaled	1	gridSearch + nearest neighbors	0.894394
17	spam_scaled	1	linear_model	0.926336
18	spam_scaled	1	MyLogRegCV	0.908083
19	spam_scaled	1	featureless	0.601043
20	spam_scaled	2	gridSearch + nearest neighbors	0.900196
21	spam_scaled	2	linear_model	0.928898
22	spam_scaled	2	MyLogRegCV	0.921070
23	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)
```




```
>>> plt.savefig("test.png")
```



3. Summary:

- Create the MyLogReg and MyLogRegCV functions with the given requirements.
- Scale the spam dataset since zip dataset is already scaled.
- Need to plot the graph comparing KNeighbors, Linear Model, Featureless and created MyLogRegCV.
- Plot the subtrain loss & validation loss with respect to iterations