CS599 (Deep Learning)

Homework – 4

1. **Python Code:**

```
import pandas as pd
import numpy as np
import matplotlib
#matplotlib.use("agg")

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_df.iloc[:, ~is_label_col]
   data_labels = data_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)

from sklearn.model_selection import KFold, GridSearchCV
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
```

```
class MyKNN:
    def __init__(self, n_neighbors):
        """store n_neighbors as attribute"""
        self.n_neighbors = n_neighbors

    def fit(self, X, y):
        """store data"""
        self.X_train = X
        self.y_train = y



    def decision_function(self, X):
        """Compute vector of predicted scores.
        Larger values mean more likely to be in positive class."""
        scores = []
        for x in X:
            distances = np.sqrt(np.sum((x - self.X_train) ** 2, axis = 1))
            n_neighbor_indices = np.argsort(distances)[:self.n_neighbors]
            n_neighbor_labels = [self.y_train[i] for i in n_neighbor_indices]

            most_common_label = Counter(n_neighbor_labels).most_common(1)[0][0]
            scores.append(most_common_label)
        return scores

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

class MyCV:
    def __init__(self, estimator, param_grid, cv):
        self.estimator = estimator
        self.param_grid = param_grid
        self.cv = cv

    def fit_one(self, param_dict, X, y):
        self.estimator.__init__(param_dict)
        self.estimator.fit(X, y)

    def fit(self, X, y):
        validation_df_list = []
        kf = KFold(n_splits=self.cv, shuffle=True, random_state=3)
        for validation_fold, (train_index, test_index) in enumerate(kf.split(X)):
```

```
        train_data = {"X": X[train_index], "y": y[train_index]}
        test_data = {"X": X[test_index], "y": y[test_index]}

        for param_dict in self.param_grid:
          self.fit_one(param_dict, **train_data)
          y_pred = self.estimator.predict(test_data["X"])
          accuracy = np.mean(y_pred == test_data["y"])
          validation_row = pd.DataFrame({
            "validation_fold": [validation_fold],
            "accuracy": [accuracy],
            "param_value": [param_dict]
          })
          validation_df_list.append(validation_row)
      validation_df = pd.concat(validation_df_list)
      best_param_dict = validation_df.groupby("param_value")["accuracy"].mean().idxmax()
      self.fit_one(best_param_dict, X, y)
  def predict(self, X):
      return self.estimator.predict(X)

class Featureless:
  def fit(self, X_train, y_train):
      y_train_series = pd.Series(y_train)
      self.most_freq_labels = y_train_series.value_counts().idxmax()

  def predict(self, x_test):
      test_nrow, test_ncol = x_test.shape
      return np.repeat(self.most_freq_labels, test_nrow)



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)

#KNN
knn1 = MyKNN(n_neighbors = 3)

#KNN + gridCV
gridcv = MyCV(estimator = knn1, param_grid= [n_neighbors for n_neighbors in
range(1,21)], cv=5)
gridcv.fit(X_train, y_train)
knn1_pred = gridcv.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)
my_learner_instance = Featureless()
my_learner_instance.fit(X_train, y_train)
featureless_pred = my_learner_instance.predict(X_test)

# store predict data in dict
pred_dict = {'gridSearch + nearest neighbors': knn_pred,
        'KNN + CV': knn1_pred,
        'linear_model': lr_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)
```

2. **Output:**

```
>>> for data_name, (data_features, data_labels) in data_dict.items():
...    kf = KFold(n_splits=3, shuffle=True, random_state=3)
... …
...
...        # K-nearest neighbors
...        knn = KNeighborsClassifier()
...        hp_parameters = {"n_neighbors": list(range(1, 21))}
... ...
...
...        for algorithm, accuracy in test_accuracy.items():
...            print(f"{algorithm} Test Accuracy: {accuracy * 100}")
... …

...        print(f"****************************End of
{data_name}({fold_num})*****************************")

Best N-Neighbors =  1
gridSearch + nearest neighbors Test Accuracy: 100.0
KNN + CV Test Accuracy: 100.0
linear_model Test Accuracy: 99.51923076923077
featureless Test Accuracy: 58.65384615384615
```

***************************End of zip(0)*****************************
Best N-Neighbors = 1
gridSearch + nearest neighbors Test Accuracy: 99.51923076923077
KNN + CV Test Accuracy: 99.51923076923077
linear_model Test Accuracy: 99.03846153846155
featureless Test Accuracy: 57.21153846153846
***************************End of zip(1)*****************************
Best N-Neighbors = 3
gridSearch + nearest neighbors Test Accuracy: 99.03381642512076
KNN + CV Test Accuracy: 100.0
linear_model Test Accuracy: 99.03381642512076
featureless Test Accuracy: 57.00483091787439
***************************End of zip(2)*****************************
Best N-Neighbors = 2
gridSearch + nearest neighbors Test Accuracy: 100.0
KNN + CV Test Accuracy: 100.0
linear_model Test Accuracy: 99.51923076923077
featureless Test Accuracy: 58.65384615384615
***************************End of zip_scaled(0)*****************************
Best N-Neighbors = 1
gridSearch + nearest neighbors Test Accuracy: 99.03846153846155
KNN + CV Test Accuracy: 99.03846153846155
linear_model Test Accuracy: 99.03846153846155
featureless Test Accuracy: 57.21153846153846
***************************End of zip_scaled(1)*****************************
Best N-Neighbors = 1
gridSearch + nearest neighbors Test Accuracy: 99.03381642512076
KNN + CV Test Accuracy: 99.03381642512076
linear_model Test Accuracy: 99.03381642512076
featureless Test Accuracy: 57.00483091787439
***************************End of zip_scaled(2)*****************************
Best N-Neighbors = 3
gridSearch + nearest neighbors Test Accuracy: 79.85658409387223
KNN + CV Test Accuracy: 82.13820078226858
linear_model Test Accuracy: 91.39504563233378
featureless Test Accuracy: 60.88657105606258
***************************End of spam(0)*****************************
Best N-Neighbors = 5
gridSearch + nearest neighbors Test Accuracy: 77.90091264667535
KNN + CV Test Accuracy: 79.92177314211213
linear_model Test Accuracy: 92.63363754889178
featureless Test Accuracy: 60.104302477183836
***************************End of spam(1)*****************************
Best N-Neighbors = 1
gridSearch + nearest neighbors Test Accuracy: 81.99608610567515

KNN + CV Test Accuracy: 81.60469667318982
linear_model Test Accuracy: 92.8897586431833
featureless Test Accuracy: 60.79582517938682
***************************End of spam(2)*****************************
Best N-Neighbors = 5
gridSearch + nearest neighbors Test Accuracy: 90.28683181225554
KNN + CV Test Accuracy: 91.13428943937419
linear_model Test Accuracy: 91.39504563233378
featureless Test Accuracy: 60.88657105606258
***************************End of spam_scaled(0)*****************************
Best N-Neighbors = 6
gridSearch + nearest neighbors Test Accuracy: 89.4393741851369
KNN + CV Test Accuracy: 90.67796610169492
linear_model Test Accuracy: 92.63363754889178
featureless Test Accuracy: 60.104302477183836
***************************End of spam_scaled(1)*****************************
Best N-Neighbors = 5
gridSearch + nearest neighbors Test Accuracy: 90.01956947162427
KNN + CV Test Accuracy: 90.93281148075668
linear_model Test Accuracy: 92.8897586431833
featureless Test Accuracy: 60.79582517938682
***************************End of spam_scaled(2)*****************************

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

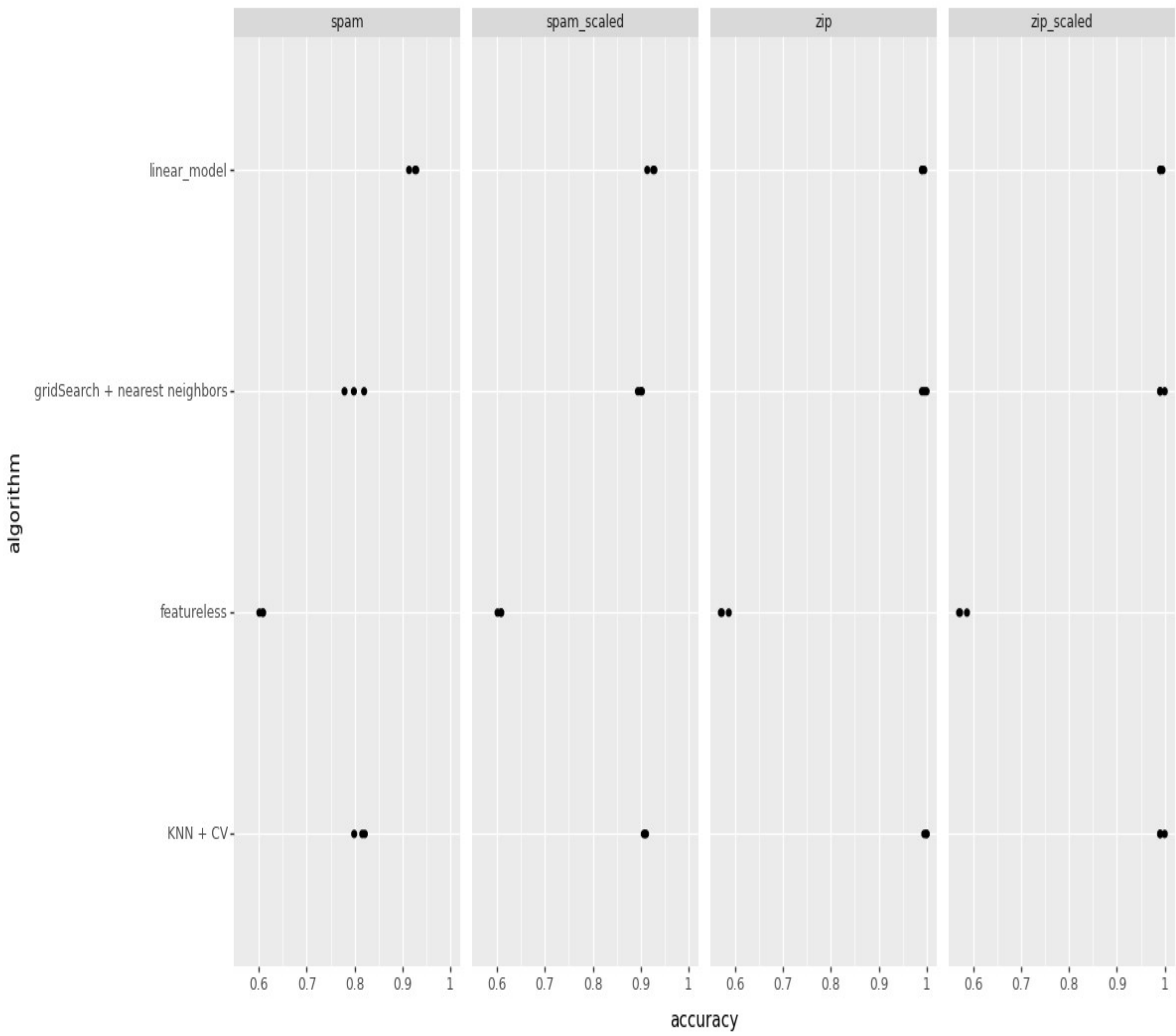
>>> print(total_accuracy_df)

	data_set	fold_id	algorithm	accuracy
0	zip	0	gridSearch + nearest neighbors	1.000000
1	zip	0	KNN + CV	1.000000
2	zip	0	linear_model	0.995192
3	zip	0	featureless	0.586538
4	zip	1	gridSearch + nearest neighbors	0.995192
5	zip	1	KNN + CV	0.995192
6	zip	1	linear_model	0.990385
7	zip	1	featureless	0.572115
8	zip	2	gridSearch + nearest neighbors	0.990338
9	zip	2	KNN + CV	1.000000
10	zip	2	linear_model	0.990338
11	zip	2	featureless	0.570048
12	zip_scaled	0	gridSearch + nearest neighbors	1.000000
13	zip_scaled	0	KNN + CV	1.000000
14	zip_scaled	0	linear_model	0.995192

15	zip_scaled	0	featureless	0.586538
16	zip_scaled	1	gridSearch + nearest neighbors	0.990385
17	zip_scaled	1	KNN + CV	0.990385
18	zip_scaled	1	linear_model	0.990385
19	zip_scaled	1	featureless	0.572115
20	zip_scaled	2	gridSearch + nearest neighbors	0.990338
21	zip_scaled	2	KNN + CV	0.990338
22	zip_scaled	2	linear_model	0.990338
23	zip_scaled	2	featureless	0.570048
24	spam	0	gridSearch + nearest neighbors	0.798566
25	spam	0	KNN + CV	0.821382
26	spam	0	linear_model	0.913950
27	spam	0	featureless	0.608866
28	spam	1	gridSearch + nearest neighbors	0.779009
29	spam	1	KNN + CV	0.799218
30	spam	1	linear_model	0.926336
31	spam	1	featureless	0.601043
32	spam	2	gridSearch + nearest neighbors	0.819961
33	spam	2	KNN + CV	0.816047
34	spam	2	linear_model	0.928898
35	spam	2	featureless	0.607958
36	spam_scaled	0	gridSearch + nearest neighbors	0.902868
37	spam_scaled	0	KNN + CV	0.911343
38	spam_scaled	0	linear_model	0.913950
39	spam_scaled	0	featureless	0.608866
40	spam_scaled	1	gridSearch + nearest neighbors	0.894394
41	spam_scaled	1	KNN + CV	0.906780
42	spam_scaled	1	linear_model	0.926336
43	spam_scaled	1	featureless	0.601043
44	spam_scaled	2	gridSearch + nearest neighbors	0.900196
45	spam_scaled	2	KNN + CV	0.909328
46	spam_scaled	2	linear_model	0.928898
47	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=10)
```



3. Summary:

- Similar to the HW3, we need to perform binary classification using KNN.
- Here, we need to scale the both datasets.
- Need to create MyKNN class, MyCV class from scratch.
- Need to plot the graph with all 4 datasets i.e., normal and scaled, with the algorithms MyKNN + MyCV, gridsearch + k-neighbors, linear model and featureless.