Assignment Two

CS 499

Richard McCormick (RLM443)

Python Program:

```
# <-- BEGIN IMPORTS / HEADERS -->
import os
import urllib
import urllib.request
import pandas as pd
import numpy as np
import plotnine as p9

import sklearn
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

from statistics import mode
import inspect
import warnings
# <-- END IMPORTS / HEADERS -->

# <-- BEGIN INITIALIZATION -->
# FILE VARIABLES
download_directory = "."

# - Spam data variables
spam_data_url = "https://hastie.su.domains/ElemStatLearn/datasets/spam.data"
spam_data_file = "spam.data"
spam_file_path = os.path.join(download_directory, spam_data_file)

# - Zip data (Training) variables
ziptrain_url = "https://hastie.su.domains/ElemStatLearn/datasets/zip.train.gz"
ziptrain_file = "zip.train.gz"
ziptrain_file_path = os.path.join(download_directory, ziptrain_file)

# - Zip data (Test) variables
ziptest_url = "https://hastie.su.domains/ElemStatLearn/datasets/zip.test.gz"
```

```python
ziptest_file = "zip.test.gz"
ziptest_file_path = os.path.join(download_directory, ziptest_file)

# CONSTANT VARIABLES
spam_label_col = 57
zip_empty_col = 257
MyKNN_N_NEIGHBORS_VAL = 20
CV_VAL = 5

# MISC. VARIABLES
kf = KFold(n_splits=3, shuffle=True, random_state=1)
test_acc_df_list = []
pipe = make_pipeline(StandardScaler(), LogisticRegression(max_iter=1000))

#CLASS VARIABLES
class MyKNN():
    def __init__(self, n_neighbors):
        # Initialize the class with number of desired neighbors
        self.n_neighbors = n_neighbors
        self.train_features = []
        self.train_labels = []

    def fit(self, X, y):
        # Stores training data
        self.train_features = X
        self.train_labels = y

    def predict(self, test_features):
        # Create array to hold prediction
        predicted_labels = []

        # Loop over data entries
        for test_index in range(len(test_features)):
            # Create array for holding best neighbors for this entry
            nearest_labels = []

            # Calculate best neighbors using code from class
            test_i_features = test_features[test_index,:]
            diff_mat = self.train_features - test_i_features
            squared_diff_mat = diff_mat ** 2
            squared_diff_mat.sum(axis=0) # sum over columns, for each row.
            distance_vec = squared_diff_mat.sum(axis=1) # sum over rows
            sorted_indices = distance_vec.argsort()
            nearest_indices = sorted_indices[:self.n_neighbors]
```

```python
            # Get the output values for selected neighbors
            for entry_index in nearest_indices:
                nearest_labels.append(self.train_labels[entry_index])

            # Add this entry's predicted outcome to the list
            predicted_labels.append(mode(nearest_labels))

        # Return list of predicted outcomes
        return(predicted_labels)

class MyCV():
    def __init__(self, estimator, param_grid, cv):
        # Initialize parameters and setup variables
        self.train_features = []
        self.train_labels = []
        self.training_data = None
        self.estimator = estimator
        self.param_grid = param_grid
        self.num_folds = cv
        self.kf = KFold(n_splits=self.num_folds, shuffle=True, random_state=1)
        self.best_model = None

    def fit(self, training_data):
        # Populate internal data structures
        self.training_data = training_data
        self.train_features = training_data["X"]
        self.train_labels = training_data["y"]

        # Create a dataframe to temporarily hold results from each fold
        best_paramter_df = pd.DataFrame()

        # Loop over the folds
        for foldnum, indicies in enumerate(self.kf.split(self.train_features)):
            print("(MyCV) Subfold #" + str(foldnum))

            # Get indicies of data chosen for this fold
            index_dict = dict(zip(["train", "test"], indicies))
            param_dicts = [self.param_grid]
            set_data_dict = {}

            # Dictionary for test and train data
            for set_name, index_vec in index_dict.items():
                set_data_dict[set_name] = {
                    "X":self.train_features[index_vec],
                    "y":self.train_labels.iloc[index_vec].reset_index(drop=True)
```

```
            }

        # Create a dictionary to hold the results of the fitting
        results_dict = {}

        # Loop over each parameter in the param_grid
        for parameter in self.param_grid:
            # Get current param
            parameter_cv = list(parameter.items())[0][1][0]

            # Pass it into estimator for construction
            estimator = self.estimator(parameter_cv)

            # Fit fold data to estimator
            estimator.fit(**set_data_dict["train"])

            # Make a prediction of current fold's test data
            prediction = estimator.predict(set_data_dict["test"]['X'])

            # Determine accuracy of the prediction
            results_dict[parameter_cv] = \
                (prediction == set_data_dict["test"]["y"]).mean()*100

        # Store the results of this fold into dataframe
        best_paramter_df = best_paramter_df.append(results_dict,
                                                   ignore_index=True)

    # Get the average results across all folds
    averaged_results = dict(best_paramter_df.mean())

    # From the averaged data, get the single best model
    best_result = max(averaged_results, key = averaged_results.get)

    # Store best model for future reference
    self.best_model = best_result

def predict(self, test_features):
    estimator = self.estimator(self.best_model)
    estimator.fit(**self.training_data)
    prediction = estimator.predict(test_features)

    return(prediction)

# <-- END INITIALIZATION -->
```

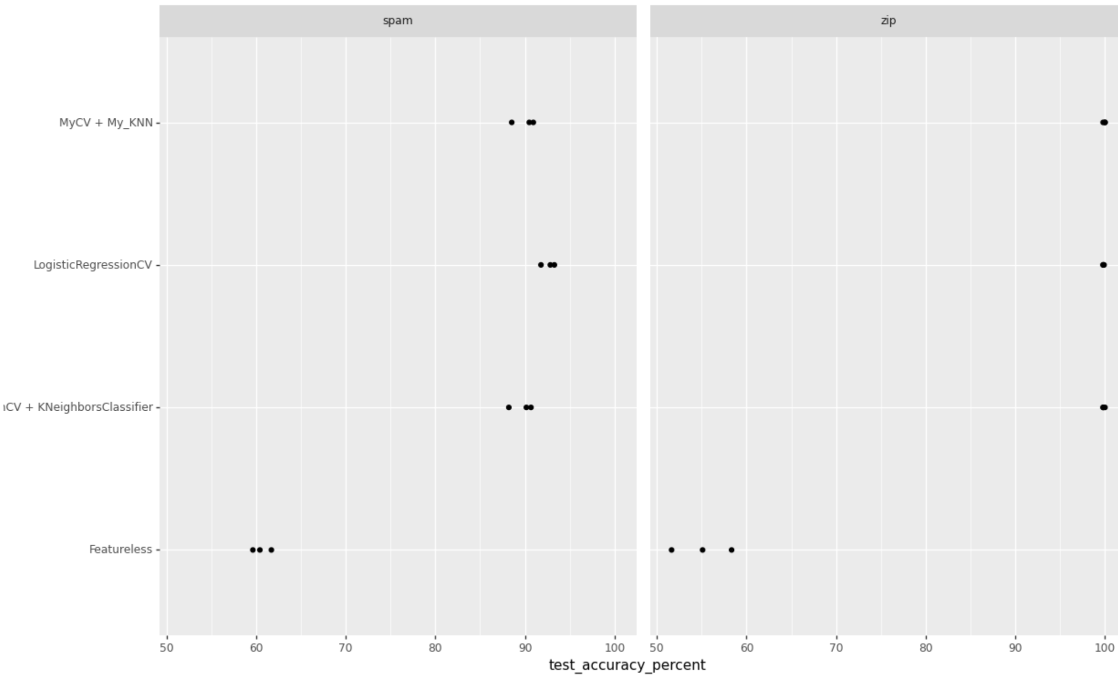
```python
# <-- BEGIN FUNCTIONS -->
# FUNCTION: MAIN
#   Description  : Main driver for Assignment Three
#   Inputs       : None
#   Outputs      : PlotNine graphs saved to program directory
#   Dependencies : build_image_df_from_dataframe
def main():
    # Display the title
    print("\nCS 499: Homework 3 Program Start")
    print("================================\n")

    # Suppress annoying plotnine warnings
    warnings.filterwarnings('ignore')

    # Download data files
    download_data_file(spam_data_file, spam_data_url, spam_file_path)
    download_data_file(ziptrain_file, ziptrain_url, ziptrain_file_path)
    download_data_file(ziptest_file, ziptest_url, ziptest_file_path)

    # Open each dataset as a pandas dataframe
    spam_df = pd.read_csv(spam_data_file, header=None, sep=" ")
    zip_train_df = pd.read_csv(ziptrain_file, header=None, sep=" ")
    zip_test_df = pd.read_csv(ziptest_file, header=None, sep=" ")
    # Concat the two zip dataframes together
    zip_df = pd.concat([zip_train_df, zip_test_df])

    # Drop rows of dataframes where the label is not ( 0 or 1)
    zip_df[0] = zip_df[0].astype(int)
    zip_df = zip_df[zip_df[0].isin([0, 1])]
    # Drop empty col from zip dataframe
    zip_df = zip_df.drop(columns=[zip_empty_col])

    # Create label vectors
    zip_labels = zip_df[0]
    spam_labels = spam_df[spam_label_col]

    # Create numpy data
    zip_data = zip_df.iloc[:, 1:256].to_numpy()
    spam_data = spam_df.iloc[:, :56].to_numpy()

    # Create data dictionary
    print("Data dictionary initialized and populated.\n")
    data_dict = {
        'spam' : [spam_data, spam_labels],
        'zip' : [zip_data, zip_labels]
```

```
    }

    # Loop through each data set
    for data_set, (input_data, output_array) in data_dict.items():
        # Output message for logging
        print("Working on set: " + str(data_set))
        current_set = str(data_set)
        # Scale the data set
        pipe.fit(input_data, output_array)

        # Loop over each fold for each data set
        for foldnum, indicies in enumerate(kf.split(input_data)):
            print("Fold #" + str(foldnum))
            # Set up input data structs
            index_dict = dict(zip(["train", "test"], indicies))
            param_dicts = [{'n_neighbors':[x]} for x in range(1, 21)]

            # Establish different models
            clf = GridSearchCV(KNeighborsClassifier(), param_dicts)
            linear_model = sklearn.linear_model.LogisticRegressionCV(cv=5)
            #my_knn = MyKNN(MyKNN_N_NEIGHBORS_VAL)
            my_cv = MyCV(MyKNN, param_dicts, CV_VAL)

            # Creating dictionary with input and outputs
            set_data_dict = {}
            for set_name, index_vec in index_dict.items():
                set_data_dict[set_name] = {
                    "X":input_data[index_vec],
                    "y":output_array.iloc[index_vec].reset_index(drop=True)
                }

            # Train the models with given data
            clf.fit(**set_data_dict["train"])
            linear_model.fit(**set_data_dict["train"])
            #my_knn.fit(**set_data_dict["train"])
            my_cv.fit(set_data_dict["train"])

            # Get most common output from outputs for featureless set
            most_common_element = mode(output_array)

            # Get results
            cv_df = pd.DataFrame(clf.cv_results_)
            cv_df.loc[:, ["param_n_neighbors", "mean_test_score"]]
            pred_dict = {
                "GridSearchCV + KNeighborsClassifier": \
```

```
                clf.predict(set_data_dict["test"]["X"]),
            "LogisticRegressionCV": \
                linear_model.predict(set_data_dict["test"]["X"]),
            "MyCV + My_KNN":my_cv.predict(set_data_dict["test"]["X"]),
            "Featureless":most_common_element
        }

        # Build results dataframe for each algo/fold
        for algorithm, pred_vec in pred_dict.items():
            test_acc_dict = {
                "test_accuracy_percent":(
                    pred_vec == set_data_dict["test"]["y"]).mean()*100,
                "data_set":data_set,
                "fold_id":foldnum,
                "algorithm":algorithm
            }
            test_acc_df_list.append(pd.DataFrame(test_acc_dict, index=[0]))

    # Build accuracy results dataframe
    test_acc_df = pd.concat(test_acc_df_list)

    # Print results
    print("\n")
    print(test_acc_df)

    # Plot results
    plot = (p9.ggplot(test_acc_df,
                    p9.aes(x='test_accuracy_percent',
                    y='algorithm'))
               + p9.facet_grid('. ~ data_set')
               + p9.geom_point())

    print(plot)

    print("\nCS 499: Homework 3 Program End")
    print("==============================\n")

# FUNCTION : DOWNLOAD_DATA_FILE
#   Description: Downloads file from source, if not already downloaded
#   Inputs:
#       - file      : Name of file to download
#       - file_url  : URL of file
#       - file_path : Absolute path of location to download file to.
#                     Defaults to the local directory of this program.
#   Outputs: None
```

```python
def download_data_file(file, file_url, file_path):
    # Check for data file. If not found, download
    if not os.path.isfile(file_path):
        try:
            print("Getting file: " + str(file) + "...\n")
            urllib.request.urlretrieve(file_url, file_path)
            print("File downloaded.\n")
        except(error):
            print(error)
    else:
        print("File: " + str(file) + " is already downloaded.\n")

# Launch main
if __name__ == "__main__":
    main()
```

Program Output:



```
CS 499: Homework 3 Program Start
================================

File: spam.data is already downloaded.

File: zip.train.gz is already downloaded.

File: zip.test.gz is already downloaded.

Data dictionary initialized and populated.

Working on set: spam
Fold #0
(MyCV) Subfold #0
(MyCV) Subfold #1
(MyCV) Subfold #2
(MyCV) Subfold #3
(MyCV) Subfold #4
Fold #1
(MyCV) Subfold #0
(MyCV) Subfold #1
(MyCV) Subfold #2
(MyCV) Subfold #3
(MyCV) Subfold #4
Fold #2
(MyCV) Subfold #0
(MyCV) Subfold #1
(MyCV) Subfold #2
(MyCV) Subfold #3
(MyCV) Subfold #4
Working on set: zip
Fold #0
(MyCV) Subfold #0
(MyCV) Subfold #1
(MyCV) Subfold #2
(MyCV) Subfold #3
(MyCV) Subfold #4
Fold #1
(MyCV) Subfold #0
(MyCV) Subfold #1
(MyCV) Subfold #2
(MyCV) Subfold #3
(MyCV) Subfold #4
Fold #2
(MyCV) Subfold #0
(MyCV) Subfold #1
(MyCV) Subfold #2
(MyCV) Subfold #3
(MyCV) Subfold #4
```

```
   test_accuracy_percent data_set  fold_id                          algorithm
0              90.156454     spam        0  GridSearchCV + KNeighborsClassifier
0              92.829205     spam        0                 LogisticRegressionCV
0              90.482399     spam        0                        MyCV + My_KNN
0              59.647979     spam        0                          Featureless
0              88.200782     spam        1  GridSearchCV + KNeighborsClassifier
0              91.786180     spam        1                 LogisticRegressionCV
0              88.526728     spam        1                        MyCV + My_KNN
0              60.430248     spam        1                          Featureless
0              90.671885     spam        2  GridSearchCV + KNeighborsClassifier
0              93.281148     spam        2                 LogisticRegressionCV
0              90.932811     spam        2                        MyCV + My_KNN
0              61.709067     spam        2                          Featureless
0              99.787460      zip        0  GridSearchCV + KNeighborsClassifier
0              99.787460      zip        0                 LogisticRegressionCV
0              99.787460      zip        0                        MyCV + My_KNN
0              58.342189      zip        0                          Featureless
0              99.787460      zip        1  GridSearchCV + KNeighborsClassifier
0              99.787460      zip        1                 LogisticRegressionCV
0             100.000000      zip        1                        MyCV + My_KNN
0              51.647184      zip        1                          Featureless
0             100.000000      zip        2  GridSearchCV + KNeighborsClassifier
0              99.893617      zip        2                 LogisticRegressionCV
0             100.000000      zip        2                        MyCV + My_KNN
0              55.106383      zip        2                          Featureless


CS 499: Homework 3 Program End
==============================
```

Question Answers / Commentary:

In this assignment, I was able to create an implementation of both the KNeighbors classifier and the CVGridSearch algorithm. My nearest neighbors algorithm was similar to the SciKit nearest neighbors algorithm, although when tested independently it showed results roughly 3% lower in accuracy than the SciKit nearest neighbors algorithm. However, when combined with my implementation of the Grid Search algorithm, I was able to achieve accuracy roughly 0.3% higher than the combined SciKit Nearest Neighbors and Grid Search process, for both the Zip and Spam data sets.

Overall, my attempt at this assignment appears to be a success. Functionally the classes I have created work very similarly to the existing SciKit tools. When the two tools created in this assignment are combined, they lead to a negligible increase in accuracy over existing SciKit modules.