**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 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"

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

kf = KFold(n\_splits=3, shuffle=True, random\_state=1)

test\_acc\_df\_list = []

pipe = make\_pipeline(StandardScaler(), LogisticRegression(max\_iter=1000))

# <-- END INITIALIZATION -->

# <-- BEGIN FUNCTIONS -->

# FUNCTION: MAIN

# Description : Main driver for Assignment Two

# Inputs : None

# Outputs : PlotNine graphs saved to program directory

# Dependencies : build\_image\_df\_from\_dataframe

def main():

# Display the title

print("\nCS 499: Homework 2 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(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)

# 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]

}

# Train the models with given data

clf.fit(\*\*set\_data\_dict["train"])

linear\_model.fit(\*\*set\_data\_dict["train"])

# Get most common output from outputs for featureless set

most\_common\_element = mode(output\_array)

#print(clf.cv\_results\_)

# Get results

cv\_df = pd.DataFrame(clf.cv\_results\_)

cv\_df.loc[:, ["param\_n\_neighbors", "mean\_test\_score"]]

pred\_dict = {

"nearest\_neighbors":clf.predict(set\_data\_dict["test"]["X"]),

"linear\_model":linear\_model.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]))

test\_acc\_df = pd.concat(test\_acc\_df\_list)

print(test\_acc\_df)

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 2 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

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:**

Chart, scatter chart

Description automatically generated

**Text

Description automatically generated**

**Question Answers / Commentary:**

From these results, it is clear that there is a difference between the two algorithms, as well as between the two data sets. For the spam data set, the nearest neighbors algorithm performed slightly better than the linear model, with roughly 3% more total accuracy.

For the zip data set, there is nearly 100% accuracy total for both algorithms. This may be due to the fact that I combined both the zip training data and the zip test data into a single data frame which was used to train the algorithms. Regardless, it seems that in this case, both algorithms performed very well.

The featureless data set scored roughly between 50 and 60 percent accuracy for both data sets. This was calculated by simply taking the mode of the output data. We would expect it to be at least 50% for a binary data set, while also being significantly less accurate than the trained algorithms. This is indeed the case.