Applications Of Machine Learning

Assignment 4

-by Srilalith Nampally

# Part 1:

## Code:

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn import preprocessing as preproc

from sklearn.metrics import accuracy\_score, roc\_auc\_score, precision\_score, recall\_score, confusion\_matrix

import time

from sklearn.preprocessing import PolynomialFeatures

def getDF(path):

df = pd.read\_excel(path)

return df

def getNormalized\_and\_train\_test(df):

# Separate features and target

X = df.iloc[:, :-1] # All rows, exclude the last column

y = df.iloc[:, -1] # All rows, just the last column

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Step 2: Fit the scaler on the training data

scaler = preproc.MinMaxScaler(feature\_range=(-1, 1))

scaler.fit(X\_train) # Compute the min and max values to be used for scaling

# Step 3: Transform both the training and test data with the fitted scaler

X\_train\_scaled = scaler.transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

return X\_train\_scaled, X\_test\_scaled, y\_train, y\_test

def getPolyTransform(X\_train, X\_test):

poly = PolynomialFeatures(degree=2, include\_bias=False)

X\_train\_poly = poly.fit\_transform(X\_train)

X\_test\_poly = poly.transform(X\_test)

return X\_train\_poly, X\_test\_poly

def getLogTransform(X\_train, X\_test):

X\_train\_log = np.log(X\_train + 1 - X\_train.min())

X\_test\_log = np.log(X\_test + 1 - X\_test.min())

return X\_train\_log, X\_test\_log

def getCombinationTransform(X\_train\_poly, X\_test\_poly, X\_train\_log, X\_test\_log):

X\_train\_combo = np.hstack([X\_train\_poly, X\_train\_log])

X\_test\_combo = np.hstack([X\_test\_poly, X\_test\_log])

return X\_train\_combo, X\_test\_combo

def logisticRegression(X\_train, Y\_train):

logistic\_model = LogisticRegression()

logistic\_model.fit(X\_train, Y\_train)

return logistic\_model

def getMetrics(model, X\_test, Y\_test):

# Predictions

y\_pred = model.predict(X\_test)

y\_pred\_proba = model.predict\_proba(X\_test)[:, 1] # Probabilities for the positive class

# Evaluation Metrics

n\_iterations = model.n\_iter\_[0]

accuracy = accuracy\_score(Y\_test, y\_pred)

roc\_auc = roc\_auc\_score(Y\_test, y\_pred\_proba)

precision = precision\_score(Y\_test, y\_pred)

recall = recall\_score(Y\_test, y\_pred)

conf\_matrix = confusion\_matrix(Y\_test, y\_pred)

# Confusion matrix components

tn, fp, fn, tp = conf\_matrix.ravel()

tpr = tp / (tp + fn) # True Positive Rate

fnr = fn / (fn + tp) # False Negative Rate

fpr = fp / (fp + tn) # False Positive Rate

tnr = tn / (tn + fp) # True Negative Rate

# Print metrics

print(f'Classification test: [{n\_iterations}] iterations', end=', ')

print(f'accuracy: {accuracy:.4f}', end=', ')

print(f'AUC: {roc\_auc:.4f}')

print(f'Precision: {precision:.6f}', end=', ')

print(f'Recall: {recall:.6f}')

print(f'Confusion Matrix:\n{conf\_matrix}')

print(f'TPR: {tpr:.4f}, FNR: {fnr:.4f}, FPR: {fpr:.4f}, TNR: {tnr:.4f}')

if \_\_name\_\_ == '\_\_main\_\_':

start = time.time()

df1 = getDF('VWXYZ.xlsx')

end = time.time()

print('Time it took to read the excel file: ', end - start)

# print(df1)

X\_train\_scaled, X\_test\_scaled, y\_train, y\_test = getNormalized\_and\_train\_test(df1)

originalModel = logisticRegression(X\_train\_scaled, y\_train)

print('\n\nMetrics for Original Dataset\n')

getMetrics(originalModel, X\_test\_scaled, y\_test)

X\_poly\_train, X\_poly\_test = getPolyTransform(X\_train\_scaled, X\_test\_scaled)

polyModel = logisticRegression(X\_poly\_train, y\_train)

print('\n\nMetrics for Polynomial Deg 2 Transformed Dataset\n')

getMetrics(polyModel, X\_poly\_test, y\_test)

X\_log\_train, X\_log\_test = getLogTransform(X\_train\_scaled, X\_test\_scaled)

logModel = logisticRegression(X\_log\_train, y\_train)

print('\n\nMetrics for Log Transformation Dataset\n')

getMetrics(logModel, X\_log\_test, y\_test)

X\_combi\_train, X\_combi\_test = getCombinationTransform(X\_poly\_train, X\_poly\_test, X\_log\_train, X\_log\_test)

comboModel = logisticRegression(X\_combi\_train, y\_train)

print('\n\nMetrics for Combination Transformation\n')

getMetrics(comboModel, X\_combi\_test, y\_test)

## Output:

PS C:\Users\srico\OneDrive\Desktop\Applications of Machine Learning\Assignment\_4> & C:/Users/srico/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/srico/OneDrive/Desktop/Applications of Machine Learning/Assignment\_4/Part1.py"

Time it took to read the excel file: 4.046921730041504

Metrics for Original Dataset

Classification test: [11] iterations, accuracy: 0.8450, AUC: 0.9268

Precision: 0.853507, Recall: 0.849206

Confusion Matrix:

[[12038 2285]

[ 2364 13313]]

TPR: 0.8492, FNR: 0.1508, FPR: 0.1595, TNR: 0.8405

Metrics for Polynomial Deg 2 Transformed Dataset

Classification test: [22] iterations, accuracy: 0.8459, AUC: 0.9274

Precision: 0.854786, Recall: 0.849333

Confusion Matrix:

[[12061 2262]

[ 2362 13315]]

TPR: 0.8493, FNR: 0.1507, FPR: 0.1579, TNR: 0.8421

Metrics for Log Transformation Dataset

Classification test: [14] iterations, accuracy: 0.8437, AUC: 0.9255

Precision: 0.862649, Recall: 0.833705

Confusion Matrix:

[[12242 2081]

[ 2607 13070]]

TPR: 0.8337, FNR: 0.1663, FPR: 0.1453, TNR: 0.8547

Metrics for Combination Transformation

Classification test: [64] iterations, accuracy: 0.8456, AUC: 0.9274

Precision: 0.854484, Recall: 0.849142

Confusion Matrix:

[[12056 2267]

[ 2365 13312]]

TPR: 0.8491, FNR: 0.1509, FPR: 0.1583, TNR: 0.8417

PS C:\Users\srico\OneDrive\Desktop\Applications of Machine Learning\Assignment\_4>

## Discussion:

The polynomial transformation seems to offer a slight improvement in model performance without a significant increase in complexity compared to the combination transformation.

The log transformation might not be as effective for this dataset based on the metrics observed.

The combination transformation increases model complexity (as seen in the number of iterations) without a proportional improvement in performance metrics.

Depending on the specific application and the cost of FP vs FN, you might opt for one transformation over another. For example, if precision is more critical than recall, the log transformation might be preferable despite its lower recall.

Continuous monitoring and validation on new data are essential to ensure the model's performance remains consistent over time.

# Part 2:

## Code:

import pandas as pd

import numpy as np

from sklearn import preprocessing as preproc

from sklearn.impute import KNNImputer, SimpleImputer

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn import model\_selection as modelsel

from sklearn import neural\_network as ann

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

import matplotlib.pyplot as plt

import time

def getData(path):

data\_frame = pd.read\_excel(path)

return data\_frame

def doPreprocessing(data\_frame):

# 1. Separate features and target

features = data\_frame.drop(columns=['AGI']) # Assuming 'AGI' is the target

target = data\_frame['AGI']

# 2. Remove ID columns

features = features.drop(columns=['HSUP\_WGT', 'MARSUPWT', 'FSUP\_WGT'])

remaining\_features = features.columns.tolist()

binary\_features = ['A\_SEX', 'HAS\_DIV'] # impute missing values by knn

ordinal\_features = ['PEINUSYR'] # impute missing values by knn

categorical\_features = ['PAW\_YN', 'A\_MARITL', 'PENATVTY'] # one hot encoding and impute missing values by knn

numeric\_features = set(remaining\_features) - set(binary\_features) - set(ordinal\_features) - set(categorical\_features)

numeric\_features = list(numeric\_features)

# print(numeric\_features)

# Preprocessing for numeric features

numeric\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='mean'))

])

# Preprocessing for ordinal and binary features: KNN imputation

knn\_transformer = Pipeline(steps=[

('imputer', KNNImputer())

])

# Preprocessing for categorical features: One-hot encoding followed by KNN imputation

categorical\_transformer = Pipeline(steps=[

('onehot', preproc.OneHotEncoder(handle\_unknown='ignore', sparse\_output=False)),

('imputer', KNNImputer())

])

# Bundle preprocessing for numeric and categorical data

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numeric\_features),

('knn\_b', knn\_transformer, binary\_features),

('knn\_o', knn\_transformer, ordinal\_features),

('cat', categorical\_transformer, ['PAW\_YN', 'A\_MARITL', 'PENATVTY'])

])

# Create a preprocessing pipeline

pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Fit and transform the features

features\_processed = pipeline.fit\_transform(features)

# Convert back into pandas DF

features\_processed\_df = pd.DataFrame(features\_processed)

return features\_processed\_df, target

def doNormalize(X):

scalerX = preproc.MinMaxScaler(feature\_range=(-1,1))

scalerX.fit(X)

X\_scaled = scalerX.transform(X)

return X\_scaled

def getMetrics(hl, clf, trainX, testX, trainY, testY):

# i. Architecture

print(f"\nArchitecture (hidden layer sizes): {hl}")

# ii. Number of epochs

print(f"Number of epochs: {clf.n\_iter\_}")

# iii. Training set metrics

train\_score = clf.score(trainX, trainY)

train\_mse = mean\_squared\_error(trainY, clf.predict(trainX))

train\_mae = mean\_absolute\_error(trainY, clf.predict(trainX))

print(f"Training Set - Coefficient of determination (R^2): {train\_score:.4f}, MSE: {train\_mse:.4f}, MAE: {train\_mae:.4f}")

# iv. Test set metrics

test\_score = clf.score(testX, testY)

test\_mse = mean\_squared\_error(testY, clf.predict(testX))

test\_mae = mean\_absolute\_error(testY, clf.predict(testX))

print(f"Test Set - Coefficient of determination (R^2): {test\_score:.4f}, MSE: {test\_mse:.4f}, MAE: {test\_mae:.4f}")

# v. Generalization gap (using R^2 for illustration)

generalization\_gap = train\_score - test\_score

print(f"Generalization gap (R^2): {generalization\_gap:.4f}\n")

def getPlot(hl, clf):

trainingLoss = np.asarray(clf.loss\_curve\_)

validation\_loss = np.sqrt(1 - np.asarray(clf.validation\_scores\_))

factor = trainingLoss[1] / validation\_loss[1]

validation\_loss = validation\_loss\*factor

# Plot setup

xlabel = "epochs (hl=" + str(hl) + ")"

fig, ax = plt.subplots()

# Plot training loss on the primary y-axis

ax.plot(trainingLoss, color="blue", label='Training Loss')

ax.set\_xlabel(xlabel, fontsize=10)

ax.set\_ylabel("Training Loss", color="blue", fontsize=10)

# Create a secondary y-axis for validation loss

ax2 = ax.twinx()

ax2.plot(validation\_loss, color="red", label='Validation Loss')

ax2.set\_ylabel("Validation Loss", color="red", fontsize=10)

# Set y-axis scale

ax.set\_yscale('log')

ax2.set\_yscale('log')

# Show plot with proper layout

fig.tight\_layout()

plt.show()

def doANNRegression(feature, target):

trainX, testX, trainY, testY = modelsel.train\_test\_split(feature, target, test\_size=0.3, random\_state=241)

hidden\_layers =[(4,4), (10,6), (32,16), (8,3,5), (12,9,10)]

for hl in hidden\_layers:

clf = ann.MLPRegressor(hidden\_layer\_sizes=hl, activation='relu', early\_stopping=True, tol=0.0005, alpha=0.0001, max\_iter=1000)

clf.fit(trainX, trainY)

getMetrics(hl, clf, trainX, testX, trainY, testY)

getPlot(hl, clf)

if \_\_name\_\_ == '\_\_main\_\_':

start = time.time()

df = getData('Census\_Supplement\_Data.xlsx')

end = time.time()

print('Time taken to read excel file: {:.4f} seconds'.format(end - start))

# print(f'\nData:\n{df.head(10)}')

# Preprocessing

X, Y = doPreprocessing(df)

# print(type(X), type(Y))

# Normalization

X\_norm = doNormalize(X)

# print(type(X\_norm))

# ANN

doANNRegression(X\_norm, Y)

## Output:

PS C:\Users\srico\OneDrive\Desktop\Applications of Machine Learning\Assignment\_4> & C:/Users/srico/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/srico/OneDrive/Desktop/Applications of Machine Learning/Assignment\_4/Part2.py"

Architecture (hidden layer sizes): (4, 4)

Number of epochs: 783

Training Set - Coefficient of determination (R^2): 0.7239, MSE: 2370905320.3842, MAE: 24599.8883

Test Set - Coefficient of determination (R^2): 0.7034, MSE: 2407847888.0077, MAE: 25009.1249

Generalization gap (R^2): 0.0205

Architecture (hidden layer sizes): (10, 6)

Number of epochs: 563

Training Set - Coefficient of determination (R^2): 0.7345, MSE: 2280056266.9017, MAE: 24134.2532

Test Set - Coefficient of determination (R^2): 0.7137, MSE: 2324753134.1979, MAE: 24534.0171

Generalization gap (R^2): 0.0208

Architecture (hidden layer sizes): (32, 16)

Number of epochs: 311

Training Set - Coefficient of determination (R^2): 0.7141, MSE: 2455452077.1516, MAE: 25225.3384

Test Set - Coefficient of determination (R^2): 0.6906, MSE: 2511714570.3482, MAE: 25674.9411

Generalization gap (R^2): 0.0234

Architecture (hidden layer sizes): (8, 3, 5)

Number of epochs: 12

Training Set - Coefficient of determination (R^2): -0.1765, MSE: 10103426843.5239, MAE: 38948.5744

Test Set - Coefficient of determination (R^2): -0.1924, MSE: 9680795223.1014, MAE: 39532.8486

Generalization gap (R^2): 0.0158

Architecture (hidden layer sizes): (12, 9, 10)

Number of epochs: 319

Training Set - Coefficient of determination (R^2): 0.7703, MSE: 1972277394.6740, MAE: 21667.1000

Test Set - Coefficient of determination (R^2): 0.7541, MSE: 1996162442.3780, MAE: 22004.0245

Generalization gap (R^2): 0.0162

PS C:\Users\srico\OneDrive\Desktop\Applications of Machine Learning\Assignment\_4>

## Validation Loss and Training Loss vs Epoch Plots:

A graph of a number of epops

Description automatically generatedA graph of a line

Description automatically generatedA graph of a number of numbers and a line

Description automatically generated with medium confidenceA graph of a function

Description automatically generated with medium confidenceA graph of a line

Description automatically generated with medium confidence

## Discussion:

Analyzing the performance of various hidden layer architectures in an artificial neural network (ANN) regressor reveals a spectrum of outcomes. The simplest (4, 4) architecture displayed respectable scores with a moderate gap between training and testing. Expanding to a (10, 6) configuration slightly enhanced performance but also slightly widened the generalization gap. Surprisingly, the more complex (32, 16) setup did not yield better results, showing lower effectiveness and the largest gap in generalization, which could suggest overfitting. On the other hand, the (8, 3, 5) structure significantly underperformed, with negative R^2 scores indicating a possible mismatch for the problem at hand. Conversely, the (12, 9, 10) architecture emerged as the top performer, achieving the highest R^2 values with a minimal generalization gap, suggesting it as the most apt model for capturing and predicting the data patterns effectively among the tested configurations.