EEE485 TERM PROJECT FIRTS REPORT – CREDIT CARD DEFAULTS

Name/Surname: Alp Dursunoğlu ID: 22102196
Name/Surname: Muhammet Melih Çelik ID: 22003836

1. INTRODUCTION

In recent years, according to financial personal data, there has been a clear increase in credit card defaults. Hence, banks aim to minimize capital loss by deploying machine learning algorithms to label potential customers that can have credit card default based on several different features of the customers. For this specific problem, a machine learning prediction system will be developed by using three different statistical learning algorithms for the project. In the project, the training and testing will be conducted with the data obtained from:

https://www.kaggle.com/code/gpreda/default-of-credit-card-clients-predictive-models/input

This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005. The dataset contains 23 distinct features related to customers. The features are given below:

1. LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
2. GENDER: (1=male, 2=female)
3. EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
4. MARRIAGE: Marital status (1=married, 2=single, 3=others)
5. AGE: Age in years
6. PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months, 9=payment delay for nine months and above)
7. PAY_2: Repayment status in August, 2005 (scale same as PAY_0)
8. PAY_3: Repayment status in July, 2005 (scale same as PAY_0)
9. PAY_4: Repayment status in June, 2005 (scale same as PAY_0)
10. PAY_5: Repayment status in May, 2005 (scale same as PAY_0)
11. PAY_6: Repayment status in April, 2005 (scale same as PAY_0)
12. BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
13. BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
14. BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
15. BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
16. BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
17. BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
18. PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
19. PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)

20. PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
21. PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
22. PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
23. PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)

2. ALGORITHMS TO BE USED FOR THE PROJECT

For this project, Logistic Regression, Support Vector Machine and Shallow Neural Network models will be used. Logistic Regression and Support Vector Machine algorithms were deployed using Python3 programming language. However, even though Support Vector Machine algorithm was implemented, the results were not promising. The accuracy of the algorithm is aimed to be improved in the final demo. And the Shallow Neural Network algorithm will be implemented in the final stages of the project.

2.1 LOGISTIC REGRESSION

Logistic regression is an algorithm used to classify data points binarily, as 0 and 1. The goal of the logistic regression is to model the probability that a given input belongs to one of the two classes based on one or more independent variables. The estimation of binary outcomes was done by fitting the data to a logistic curve, also known as the sigmoid function. The logistic function is defined as:

$$\pi(\underline{X}\,;\underline{\beta}) = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_p X_p}}, \qquad Y \sim Bernoulli\left(\pi\left(\underline{X}\,;\underline{\beta}\right)\right)$$

where βs are coefficients. The logistic regression coefficients were calculated by the Newton-Raphson Method.

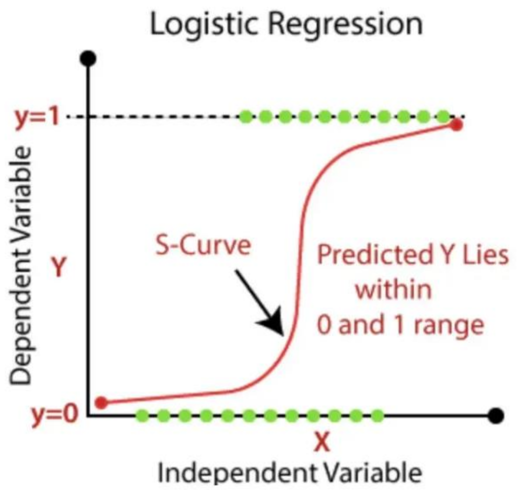


Fig.1: Visualization of Logistic Regression Method [4]

2.1.1 NEWTON-RAPHSON METHOD

Newton-Raphson method is a powerful iterative technique used to find successively better approximations to the roots (or zeroes) of a real-valued function. The method's procedure starts with an initial guess x_0 for the root of the function *f(x).* This guess can be any value, but it's often chosen to be close to the actual root for faster convergence. After computing the tangent line to the function *f(x)* for initial guess, the guess is updated with the point on which the x-axis and the tangent line intersects. Equation of this tangent line is given by:

$$y - f(x_0) = f'(x_0)(x - x_0)$$

where $f'(x_0)$ represents the derivative of *f(x)* evaluated at x_0. The updated guess can be indicated with x_1. Same procedure is applied for the x_1. The iterative process continues until convergence. Therefore, the method can be summarized by the general formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

2.1.2 IMPLEMENTATION OF LOGISTIC REGRESSION

In the project, first, the main design matrix **X** was obtained by the written module **LR_Pre.py**, which is given in Appendix B. Then, in the main module of the implementation, **LR_Main.py**, $\underline{\pi}$ vector and **W** matrices were initialized for calculations. **LR_Main.py** is also given in Appendix B.

$$\boldsymbol{X} = \begin{pmatrix} 1 & \cdots & X_{1p} \\ \vdots & \ddots & \vdots \\ 1 & \cdots & X_{np} \end{pmatrix}, \ \underline{\pi} = \begin{bmatrix} \pi(X_1; \underline{\beta_{old}}) \\ \pi(X_2; \underline{\beta_{old}}) \\ \vdots \\ \vdots \\ \pi(X_n; \underline{\beta_{old}}) \end{bmatrix}, \underline{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ \vdots \\ Y_n \end{bmatrix}$$

$$\boldsymbol{W} = \begin{bmatrix} \pi\left(X_1; \underline{\beta_{old}}\right)\left(1 - \pi\left(X_1; \underline{\beta_{old}}\right)\right) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \pi\left(X_n; \underline{\beta_{old}}\right)\left(1 - \pi\left(X_n; \underline{\beta_{old}}\right)\right) \end{bmatrix}$$

After initialization of necessary matrices, the MLE estimate of beta coefficients are found by the iterative Newton-Raphson formula given below:

$$\underline{\beta_{new}} = \underline{\beta_{old}} + (\boldsymbol{X^T W X})^{-1} \boldsymbol{X^T} (\underline{Y} - \underline{\pi}) \ [5]$$

Obtained results indicates a probability; hence, for binary results, a hyperparameter λ was used to determine an optimized threshold. The optimal lambda value was achieved via Cross-Validation. The module for Cross-Validation **LR_CV.py** is given in Appendix B.

2.1.3 IMPLEMENTATION RESULTS OF LOGISTIC REGRESSION

The optimal value of the threshold hyperparameter was found by K-fold Cross-Validation. It is identified that the highest accuracy rate was obtained when λ = 0.6, which is shown both in Table 1 and in Figure 2.

Lambda Value	Train Accuracy	Test Accuracy
0.1	0.222	0.23
0.2	0.220444444	0.216
0.3	0.223333333	0.218
0.35	0.246444444	0.242
0.4	0.297111111	0.304
0.5	0.570888889	0.584
0.6	0.802222222	0.804
0.7	0.785555556	0.802
0.8	0.780222222	0.784
0.9	0.778444444	0.784

Table 1: K-Fold Cross Validation Results for Hyperparameter Value

The results were obtained with 3000 sample from the dataset and 10000 number of iterations. Further accuracy improvements can be obtained via larger sample sizes. Moreover, for the logistic regression training CUDA and Cupy libraries were used for hardware acceleration but parallel computing didn't suit the computation flow of the modules.

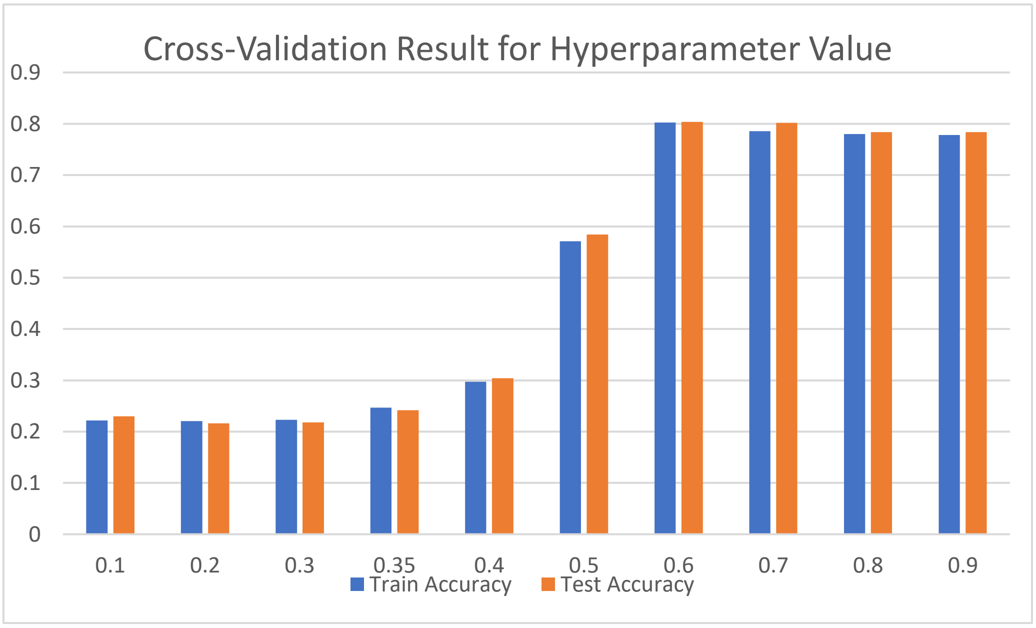


Fig.2: Cross-Validation Result for Hyperparameter Value

2.2 SUPPORT VECTOR MACHINE

Support Vector Machine (SVM in short) algorithm is a supervised learning method that can be used for regression and classification problems. In this project, we used SVM for classification purposes. When dealing with classification problems, SVM's aim is to find the best (optimal) hyperplane that separates the data points linearly regarding their classes. The hyperplane is positioned between the support vectors, the nearest data points of each class to the hyperplane. Then SVM tries to maximize this defined margin and achieves a better classification on the test data.

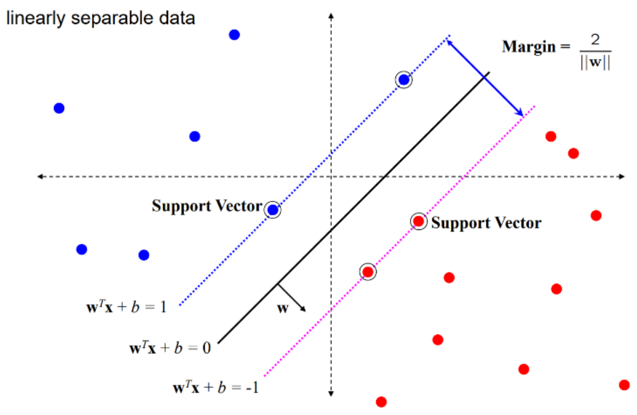

linearly separable data
Margin = $\frac{2}{||\mathbf{w}||}$
Support Vector
Support Vector
$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}^T\mathbf{x} + b = 0$
$\mathbf{w}^T\mathbf{x} + b = -1$
w

Fig.3: Visualization of the Support Vector Machine Model [3]

The logic behind the SVM is using a linear line equation:

$$y = ax + b$$

Rearrangement gives:

$$ax + b - y = 0.$$

By defining vectors $\underline{\mathrm{X}} = \begin{bmatrix} x \\ y \end{bmatrix}$ and $\underline{\mathrm{W}} = \begin{bmatrix} a \\ -1 \end{bmatrix}$, we can write the equation $ax + b - y = 0$ as:

$$\underline{\mathrm{X}} \cdot \underline{\mathrm{W}} + b = 0 \ (*)$$

In the Figure 2, the optimal hyperplane lies on $(*)$. For given vectors $\underline{\mathrm{X}}_i$ and $\underline{\mathrm{W}}_i$, if the result of $(*)$ is bigger than zero, it is classified as blue class; otherwise, the datapoint is classified as red class. For estimating purposes, we need the matrices **X** and **W**. We already have the main design matrix X. So, we need to find **W**. In this project, the matrix **W** was found using the gradient descent technique.

2.2.1 GRADIENT ASCENT/DESCENT

Gradient ascent/descent is a recursive algorithm that aims to optimize the parameters of a machine learning model by updating itself until it converges. This algorithm is used to optimize parameters of many models such as Neural Networks and Support Vector Machine, which are selected models for the project [1].

The gradient of a continuous function *f* at point P is defined as the vector that contains the partial derivatives of the function computed at that point P [1]. The gradient descent algorithm aims to minimize the loss function with respect to weights by simply updating weights in every single iteration. The weights are updated in the direction of the gradient. When the minimum of loss function is achieved (which also means that optimal value for weight hyperparameter is achieved) the algorithm converges and stops working. In the project we have successfully achieved the optimal hyperparameter for our model. The mathematical formula for gradient descent is:

$$W_{n+1} = W_n - \gamma \frac{\delta L}{\delta W}$$

where γ denotes the learning rate.

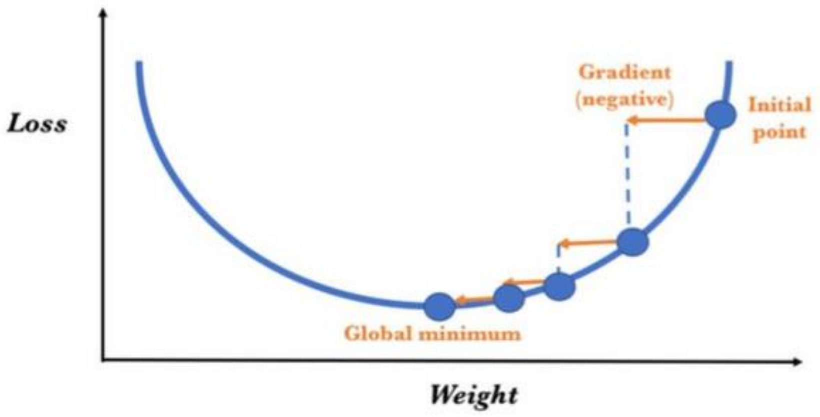


Fig.4: Visualization of the gradient descent method [2]

The gradient ascent algorithm works very similarly to the gradient descent algorithm with only one difference. In gradient ascent, the aim is to maximize a function with respect to a hyperparameter, instead of minimizing it. The mathematical expression for the gradient ascent algorithm is:

$$W_{n+1} = W_n - \gamma \frac{\delta L}{\delta W}$$

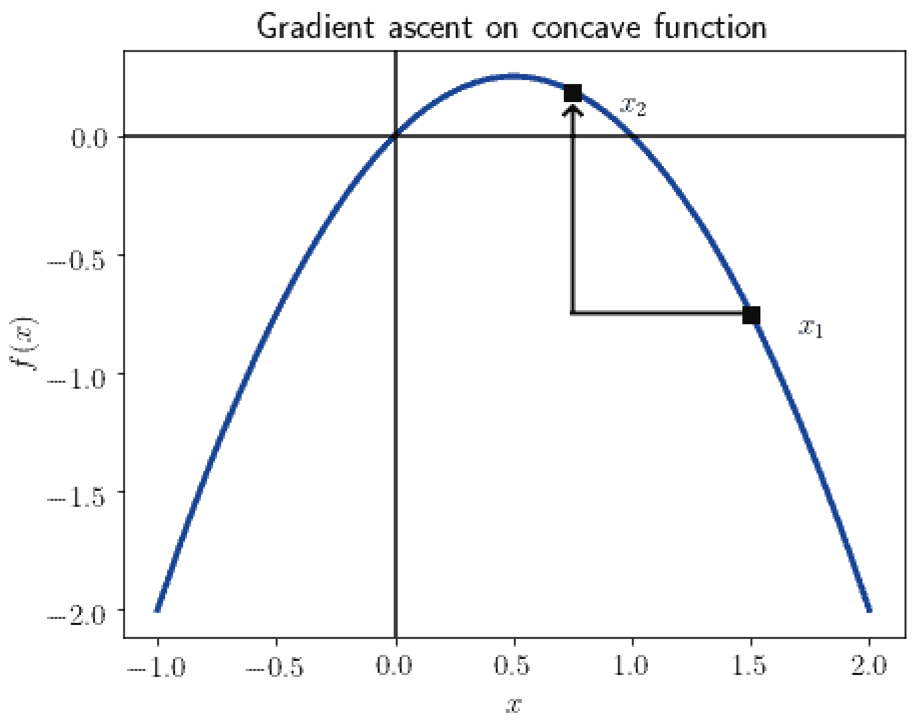


Fig.5: Visualization of the gradient ascent method [1]

2.2.2 GRADIENT DESCENT FOR SVM MODEL

The margin between support vectors can be expressed with $\frac{2}{w}$. Our goal is to maximize the margin, namely, we need to maximize $\frac{1}{||2w||}$ that is subject to constraints

$$y_i(w \cdot x_i + b) \geq 1 \ \forall \ i$$

To do that, this problem can be transformed to an unconstrained optimization problem using the Lagrange Multipliers method. The Lagrangian equation for this problem is:

$$L(w, b, \alpha) = \frac{1}{2} \parallel w \parallel^2 - \sum_{i=1}^{N} \alpha_i [y_i(w \cdot x_i + b) - 1]$$

Taking the derivative with respect to lambda and solving for (w) gives us the gradient:

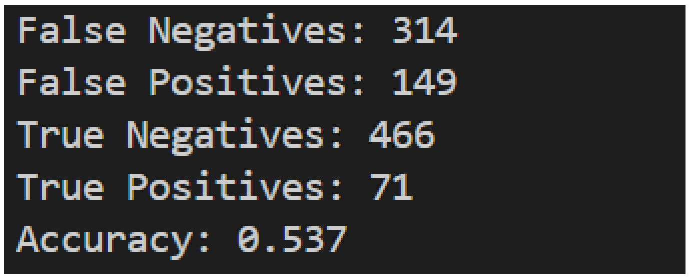
$$\nabla_w L = \frac{\delta L}{\delta w} = \sum_{i=1}^{N} \alpha_i \, y_i x_i$$

And the mathematical expression for the gradient ascent algorithm is:

$$w_{n+1} = w_n - \gamma \sum_{i=1}^{N} \alpha_i \, y_i x_i$$

2.2.3 IMPLEMENTATION RESULTS FOR SVM MODEL

The Support Vector Machine Algorithm was implemented in Python3 programming language by using the gradient descent optimization technique. However, the results were not promising. The raw data was trained and tested on 1000 data points (1000 for training, 1000 for testing) with parameters tolerance = $5 \cdot 10^{-6}$, learning rate = $2 \cdot 10^{-4}$, C = 45. Only %57.7 accuracy was achieved with only 71 true positives out of 220. The reasons for unsatisfying results may be lack of parameter optimization or a mistake in the code. The algorithm will be examined and analyzed. In addition to that, several algorithms for hyperparameter tuning will be implemented before the final demo to overcome the issues and improve the quality of the estimation of the SVM algorithm.



```
False Negatives: 314
False Positives: 149
True Negatives: 466
True Positives: 71
Accuracy: 0.537
```

Fig.6: Results of the SVM algorithm

3. WORK DONE BY EACH GROUP MEMBER

Name/Surname: Muhammet Melih Çelik ID:22003836

- Research About Logistic Regression Method
- Transferring the data from the .csv file to Python and storing it as matrixes (data import)
- Implementation of Logistic Regression Algorithm
- Cross-validation on Logistic Regression parameters to find optimum parameter values
- Visualization of the data and results

Name/Surname: Alp Dursunoğlu ID:22102196

- Research About Support Vector Machine Method
- Data preprocessing (normalization, separation of test sets and training sets)
- Implementation of Support Vector Machine Algorithm

4. WORK TO BE DONE BY EACH GROUP MEMBER

Name/Surname: Muhammet Melih Çelik ID:22003836

- Research of the Shallow Neural Network Method
- Implementation of the Shallow Neural Network Algorithm

Name/Surname: Alp Dursunoğlu ID:22102196

- Researching the Shallow Neural Network Method
- Improving the implemented SVM algorithm
- Cross-validation to find optimal hyperparameters of SVM
- Visualization of the final results
- Implementation of cross-validation algorithms

5. CONCLUSION AND DISCUSSION

In the first part of the project, Logistic Regression and SVM algorithms were researched and implemented with Python3 programming language. The results of the Logistic Regression algorithm were promising. However, the results from the SVM algorithm were not satisfying. In the second part of the project, the aim is to improve the accuracy of SVM algorithm. And the remaining method, Shallow Neural Network, will be researched and implemented for the second part of the project.

APPENDICES: CODES FOR IMPLEMENTED ALGORITHMS

APPENDIX A: SVM

```
"""
SUPPORT VECTOR MACHINE
WORK IN PROGRESS.............
"""
import numpy as np
import random
import math
import Logistic_Regression_Preprocess as LRP

def data_to_matrix_function():
    #this function will take the raw data and column size
    #and will return the main X decision matrix as an output.

    raw_data = open("UCI_Credit_Card.csv",'r')
    #the raw data obtained from kaggle.com as below:
    #https://www.kaggle.com/datasets/uciml/default-of-credit-card-
clients-dataset?resource=download

    column_size = int(input("Specified column size to be imported:
"))
    #column size will be input for model training with X which will
have variying column size
    #there are 23 number of fixed features.
    #X1,X2,X3,X4,...,X23.
    #decision_matrix_function(raw_data, column_size)

    predictor_matrix = []
    response_vector = []

    for i in range(1,column_size+2):                         #first
two line are predictor names.

        column_vector_str = raw_data.readline()
        """
        try:
            first = column_vector_str[0] #if a line starts with a
number it will
            first = int(first)           #it will be added as column
to the matrix.

            column_vector_lst = column_vector_str.split(',')
#splitting the line
                                                             # to
add it to a column vector.

            column_vector_lst[-1] = column_vector_lst[0:-2]  #last
element is \n
                                                             # which
is removed.
            column_vector_int_lst = []
            for i2 in range(0,len(column_vector_lst)-1):

column_vector_int_lst.append(int(column_vector_lst[i2]))

            X.append(column_vector_int_lst)

        except ValueError:
            continue
        """

        #due to the integer-float-string conversion the code above
generates
        #matrix with column dimension less than the input value.

        if column_vector_str[0].isdigit():
            column_vector_lst = column_vector_str.split(',')
            column_vector_lst[-1] = column_vector_lst[-1][0:1]
            response_vector.append(int(column_vector_lst[-1]))
            column_vector_float_lst = [1]
            for i2 in range(1,len(column_vector_lst)-1): # last
value is the y value of the regression.

column_vector_float_lst.append(float(column_vector_lst[i2]))

            predictor_matrix.append(column_vector_float_lst)

        else:
            continue


    #NORMALIZATION

    #print(np.mean(response_vector))
    # response_vector=random.shuffle(response_vector)
    # predictor_matrix=random.shuffle(predictor_matrix)
    trainset_length=math.floor(len(predictor_matrix)*0.7)

    X_test=predictor_matrix[trainset_length:]
    X_train=predictor_matrix[0:trainset_length]
    Y_test=response_vector[trainset_length:]
    Y_train=response_vector[:trainset_length]
    return predictor_matrix, response_vector, X_test, X_train,
Y_test, Y_train

data_to_matrix_function()

def gradient_descent(X,Y,C,learning_rate,tolerance,iterations):

    W=np.zeros(len(X[0]))

    for i in range (0,iterations):
        gradient=np.zeros(len(X[0]))

        for k in range(0,len(X)):
            if Y[k] * np.dot(W, X[k])<1:

                gradient+=C*Y[k]*X[k]


        W=W-learning_rate*gradient
        if np.all(abs(gradient*learning_rate) < tolerance):
            break


    return W

def convert_Y(Y_Train):
    for k in range (len(Y_Train)):
        if Y_Train[k]==0:
            Y_Train[k]=-1
    return Y_Train

def Support_Vector_Machine(X_Train,X_Test,Y_Train,iterations,tolerance,learning_rate,C):

    Y_Train2=convert_Y(Y_Train)

    print(np.mean(X_Train))


W=gradient_descent(X_Train,Y_Train2,C,learning_rate,tolerance,iterations)

    training_results=[]
    test_results=[]
```

```
#90000000000000000
    for X1 in X_Train:
        if np.dot(W,X1)>20000:
            #print(np.dot(W,X1))
            training_results.append(1)
        else:
            training_results.append(0)

    for X2 in X_Test:
        if np.dot(W,X2)>20000:
            test_results.append(1)
        else:
            test_results.append(0)


    return training_results,test_results

iterations=200
tolerance=5e-06
learning_rate=0.0002
C=45

(X, Y) = LRP.data_to_matrix_function()

X=(X-(np.mean(X)))/np.std(X)

X_Train=X[0:1000]
X_Test=X[1000:2000]
Y_Train=Y[0:1000]
Y_Test=Y[1000:2000]

training_results,test_results=Support_Vector_Machine(X_Train,X_Test,
Y_Train,iterations,tolerance,learning_rate,C)

summm=0
summm2=0
```

```python
for k in range(len(test_results)):
    if test_results[k]==Y_Test[k]:
        summm2+=1
        if Y_Test[k]==1:
            summm+=1

false_positive = 0
false_negative = 0
true_positive = 0
true_negative = 0

for i in range(0,len(test_results)):
    if (test_results[i] == Y_Test[i]) and (Y_Test[i] == 1):
        true_positive += 1
    elif (test_results[i] == Y_Test[i]) and (Y_Test[i] == 0):
        true_negative += 1
    elif (test_results[i] != Y_Test[i]) and (Y_Test[i] == 1):
        false_positive += 1
    elif (test_results[i] != Y_Test[i]) and (Y_Test[i] == 0):
        false_negative += 1



#print(training_results)

#print(test_results)
#print(Y_Test)
#print(summm2/len(test_results))
#print(summm2/summm)

accuracy = ((true_positive+true_negative)/len(test_results))

print("False Negatives:",false_negative)
print("False Positives:",false_positive)

print("True Negatives:",true_negative)
print("True Positives:",true_positive)
print("Accuracy:",accuracy)
```

APPENDIX B: LOGISTIC REGRESSION

LR_PRE.py:

```
"""
EEE 485 PROJECT: Credit Card Default

Module 1:This module imports raw data in a specified size and generates
         main decision matrix X and response vector Y.
"""

import numpy as np

def data_to_matrix_function():
    #this function will take the raw data and column size
    #and will return the main X decision matrix as an output.

    raw_data = open("UCI_Credit_Card.csv",'r')
    #the raw data obtained from kaggle.com as below:
    #https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-
dataset?resource=download

    column_size = int(input("Specified column size to be imported: "))
    #column size will be input for model training with X which will have
variying column size
    #there are 23 number of fixed features.
    #X1,X2,X3,X4,...,X23.

    predictor_matrix = []
    response_vector = []

    for i in range(1,column_size+2): #first two line are predictor names.

        row_vector_str = raw_data.readline()

        if row_vector_str[0].isdigit():
            row_vector_lst = row_vector_str.split(',')
            row_vector_lst[-1] = row_vector_lst[-1][0:1]
            response_vector.append(int(row_vector_lst[-1]))

            row_vector_float_lst =
[1]
            for i2 in range(1,len(row_vector_lst)-1): # last value is the y
value of the regression.
```

```
                row_vector_float_lst.append(float(row_vector_lst[i2]))

            predictor_matrix.append(row_vector_float_lst)
        else:
            continue

    return predictor_matrix, response_vector

```

LR_Main.py:

```
"""
EEE 485 PROJECT: Credit Card Default

Module 2:This module takes the decision matrix X and the response vector Y to
make prediction
         with logistic regression.
"""
import numpy as np
import time
import LR_Pre as LR_Pre

def Newton_Raphson(X,Y,iteration_number):

    #(predictor_matrix_numpy, response_vector) = LRP.data_to_matrix_function()
    #number_of_iteration = int(input("Iteration number: "))
    #X = np.array(predictor_matrix_numpy)        # X = [1,X1,X2,X3,...,X23]
the decision matrix
    #Y = np.array(response_vector)               # Y is the response vector

    Beta_coefficients = np.ones(24)*0.5          # B = [B0,B1,B2,B3,...B23],
unknown random coefficients which

    Beta_new = np.zeros(24)
    Beta_old = Beta_coefficients
    Beta_old2 = Beta_coefficients

    row_size = X.shape[0]

    #Pi vector generation:
    Pi_vector = []
    for i2 in range(0, row_size):
        Pi_vector.append(logistic_function(Beta_old,X[i2,:]))

    #W matrix generation:
    W = np.eye(X.shape[0])
    for i in range(0, X.shape[0]):
        W[i,i] = logistic_function(Beta_old,X[i,:])*(1-
logistic_function(Beta_old,X[i,:]))
```

```
    X_T = np.transpose(X)
    A = np.dot(np.dot(X_T,W), X)  #A = X_t*W*X
    A_inverse = np.linalg.inv(A)  #A^-1 = (X_t*W*X)^-1

    count = 0

    while count != iteration_number:

        A = np.dot(np.dot(X_T,W), X) #A = X_t*W*X
        A_inverse = np.linalg.inv(A) #A^-1 = (X_t*W*X)^-1
        C1 = np.dot(A_inverse,X_T)
        C2 = Y - Pi_vector
        C3 = np.dot(C1,C2)
        Beta_old2 = Beta_old
        Beta_new = Beta_old + C3

        #updating matrices and coefficients.
        Beta_old = Beta_new

        for i in range(0, X.shape[0]):
            Pi_vector[i] = (logistic_function(Beta_old,X[i,:]))

        for i in range(0, X.shape[0]):
            W[i,i] = logistic_function(Beta_old,X[i,:])*(1-
logistic_function(Beta_old,X[i,:]))

        count += 1

        print(count)

    return Beta_new


def logistic_function(beta_vector,X_i_colum_vector):

    logistic_result = np.longdouble()
    dot_product = np.longdouble()
    dot_product = np.dot(beta_vector,X_i_colum_vector)/10000

    logistic_result = 1/(1 + np.exp(-dot_product))

    return logistic_result
```

LR_CV.py:

```
"""
EEE 485 PROJECT: Credit Card Default
```

```
Module 3: K-Fold Cross-Validation for Logistic Regression
          K = 10

"""
import numpy as np
import LR_Pre as LR_Pre
import LR_Main as LRM_W

all_data, all_response = LR_Pre.data_to_matrix_function()
all_data = np.array(all_data)
all_response = np.array(all_response)

iteration = int(input("Iteration number: "))

fold_size = int(input("fold size: "))
lambda_parameter_list = [0.1,0.2,0.3,0.35,0.4,0.5,0.6,0.7,0.8,0.9]
j_lst = np.random.choice(np.arange(0, 9+1), size=10, replace=False)
f_matrix = []
f_matrix_response = []
fold_index = 0
accuracy_lst =[]

for i in range(0,10):

    jth_fold = all_data[j_lst[i]*fold_size:j_lst[i]*fold_size + fold_size,:]
    jth_fold_response = all_response[j_lst[i]*fold_size:j_lst[i]*fold_size +
fold_size]
    f_matrix.append(jth_fold)
    f_matrix_response.append(jth_fold_response)

while fold_index != 10:

    training_accuracy_elements      = [0,0,0,0] #1.TP, 2.TN, 3.FP, 4.FN
    fold_testing_accuracy_elements  = [0,0,0,0] #1.TP, 2.TN, 3.FP, 4.FN

    selected_lambda = lambda_parameter_list[fold_index]
    Y_test = f_matrix_response[fold_index]
    X_test = f_matrix[fold_index]

    #for X_train
    matrices_to_concatenate = []
    for fold in range(0,10):
        if fold != fold_index:
            matrices_to_concatenate.append(f_matrix[fold])
        else:
            continue
    X_train = np.concatenate(matrices_to_concatenate,axis=0)
    #for Y_train
    Y_train = []
    for fold in range(0,10):
        if fold != fold_index:
            Y_train.extend(f_matrix_response[fold])
        else:
            continue

    Y_train = np.array(Y_train)
    print(X_train.shape[0])
    print(Y_train.shape[0])
    Beta_hat = LRM_W.Newton_Raphson(X_train,Y_train,iteration)

    #---------------training accuracy part-----------
    log_estimate_train = []

    for row in range(0, len(X_train)):
        result = LRM_W.logistic_function(Beta_hat, X_train[row])
        if result >= selected_lambda:
            log_estimate_train.append(1)
        else:
            log_estimate_train.append(0)

    y = np.array(Y_train)
    log = np.array(log_estimate_train)

    for i in range(0,len(Y_train)):

        if (log_estimate_train[i] == Y_train[i]) and (Y_train[i] == 1):
            training_accuracy_elements[0] += 1
        elif (log_estimate_train[i] == Y_train[i]) and (Y_train[i] == 0):
            training_accuracy_elements[1] += 1
        elif (log_estimate_train[i] != Y_train[i]) and (Y_train[i] == 1):
            training_accuracy_elements[2] += 1
        elif (log_estimate_train[i] != Y_train[i]) and (Y_train[i] == 0):
            training_accuracy_elements[3] += 1

    training_accuracy = (training_accuracy_elements[0] +
training_accuracy_elements[1])/(all_data.shape[0]-fold_size)

    #---------------testing accuracy part------------
    log_estimate_test = []

    for row in range(0, len(X_test)):
        result = LRM_W.logistic_function(Beta_hat, X_test[row])
        if result >= selected_lambda:
            log_estimate_test.append(1)
        else:
            log_estimate_test.append(0)

    y = np.array(Y_test)
    log = np.array(log_estimate_test)

    for i in range(0,len(Y_test)):

        if (log_estimate_train[i] == Y_train[i]) and (Y_train[i] == 1):
            fold_testing_accuracy_elements[0] += 1
        elif (log_estimate_train[i] == Y_train[i]) and (Y_train[i] == 0):
            fold_testing_accuracy_elements[1] += 1
        elif (log_estimate_train[i] != Y_train[i]) and (Y_train[i] == 1):
            fold_testing_accuracy_elements[2] += 1
        elif (log_estimate_train[i] != Y_train[i]) and (Y_train[i] == 0):
            fold_testing_accuracy_elements[3] += 1

    test_accuracy = (fold_testing_accuracy_elements[0] +
fold_testing_accuracy_elements[1])/(fold_size)

    accuracy_tuple = (training_accuracy,test_accuracy)
    accuracy_lst.append(accuracy_tuple)

    fold_index += 1

for i in range(0,10):
    print(lambda_parameter_list[i],accuracy_lst[i],"\n")
```

REFERENCES

[1] W. by: G. D. Luca, “What is the difference between gradient descent and gradient ascent?,” Baeldung on Computer Science, https://www.baeldung.com/cs/gradient-descent-vs-ascent (accessed Apr. 19, 2024).

[2] L. Jiang, “A visual explanation of gradient descent methods (momentum, AdaGrad, RMSProp, adam),” Medium, https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c (accessed Apr. 19, 2024).

[3] Lecture 2: The SVM classifier, https://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf (accessed Apr. 19, 2024).

[4] F. Franco, “Logistic regression algorithm,” Medium, https://medium.com/@francescofranco_39234/logistic-regression-algorithm-6451c7928375#:~:text=Logistic%20regression%20is%20a%20statistical,or%20any%20other%20dichotomous%20categorization. (accessed Apr. 20, 2024).

[5] *Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Scholars Portal