Heaven's Light is Our Guide

Computer Science and Engineering
Rajshahi University of Engineering and Technology

Course No: CSE 4204

Course Title: Sessional Based on CSE 4203

No of Experiment:02

Name of the Experiment: Design and implementation of single layer perceptron learning algorithm.

Course Outcomes:CO1

Learning Domain with Level: Cognitive (Applying, Analyzing,Evaluating and Creating)

Submitted To

Rizoan Toufiq

Assistant Professor

Department of Computer Science and Engineering

Rajshahi University of Engineering and Technology

Submitted By

Name:Rafi Ahammed Songram

Roll:1803095

Department:Computer Science and Engineering

Rajshahi University of Engineering and Technology

Design and implementation of Single Layer Perceptron Learning Algorithm.

Content Page

Single layer Perceptron Algorithm:

The single layer perceptron is a type of artificial neural network that can be used for binary classification problems. It consists of a single layer of neurons, each of which takes in a set of input values, applies a weighted sum to them, and outputs a binary classification decision based on the weighted sum.

A feed-forward neural network, such as a single layer perceptron, is one in which the input signals travel forward through the network, going via the neurons in each layer before generating an output.The decision is typically either 0 or 1, which corresponds to one of the two classes in a binary classification problem. In the case of the single layer perceptron, there is only one layer of neurons, and each neuron computes a weighted sum of its inputs, applies a threshold function to the sum, and outputs a binary decision based on the result.

The weights of the single layer perceptron are adjusted during training process to minimizes the errror process on training data. Once the network is trained, it can be used to classify new inputs into one of the two output categories.

Steps of Perceptron Learning Algorithm:

Step-1: Initialize weights and threshold. Define w_i(t)(0 $\leq$ i $\leq$ n)be the weight from input i and time t and θ to be the threshold value in the output node.Set w_0 to be -θ , the bias and x_0 to be always 1. Set w_0 to small random values, thus initializing all the weights and the threshold.

Step-2: Present input (x_0,x_1.....,x_n) and desired output d(t)
Step-3: Calculate actual output:

$$y(t) = f_h \sum_{i=0}^{n} w_i(t)x_i(t)$$

Step-4:

- Adapt Weights:
  If correct W_i(t+1)= w_i(t)
  If output 0, should be 1 (class A) W_i(t+1)= w_i(t) +x_i(t)
  If output 1, should be 0 (class B) W_i(t+1)= w_i(t) -x_i(t)

- Adapt weights – modified version
  If correct W_i(t+1)= w_i(t)
  If output 0, should be 1 (class A) W_i(t+1)= w_i(t) + ηx_i(t)
  If output 1, should be 0 (class B) W_i(t+1)= w_i(t) - ηx_i(t)
  η positive gain term that controls the adaption rate.

- Adapt weights – Widrow-Hoff delta rule
  $\triangle$ = d(t) -y(t)
  W_i(t+1)= w_i(t) + ηx_i(t)
  d(t)= +1 if input from class A
  d(t)= -1 if input from class B

Dataset:

The name of the dataset is **Diabetes** and csv file diabetes.csv

Table 1: Attribute of Dataset

Column	**Count**	**Type**
1 pregnencies	768	int64
2.Glucose	768	int64
3.BloodPressure	768	int64
4.SkinThickness	768	int64
5.Insulin	768	int64
6.BMI	768	float64
7.Diabetespedigrefunction	768	float64
8.Age	768	int64
9.Outcome	768	int64

Here are the all attribute of full dataset and we apply on this single layer perceptron algorithm .

Outcome plot of dataset:

Code-snippet

```
ax = sns.countplot(df["diagnosis"],label="Count")
f, ax = plt.subplots(figsize=(2, 2))
sns.countplot(df, x="Outcome", ax=ax)
```

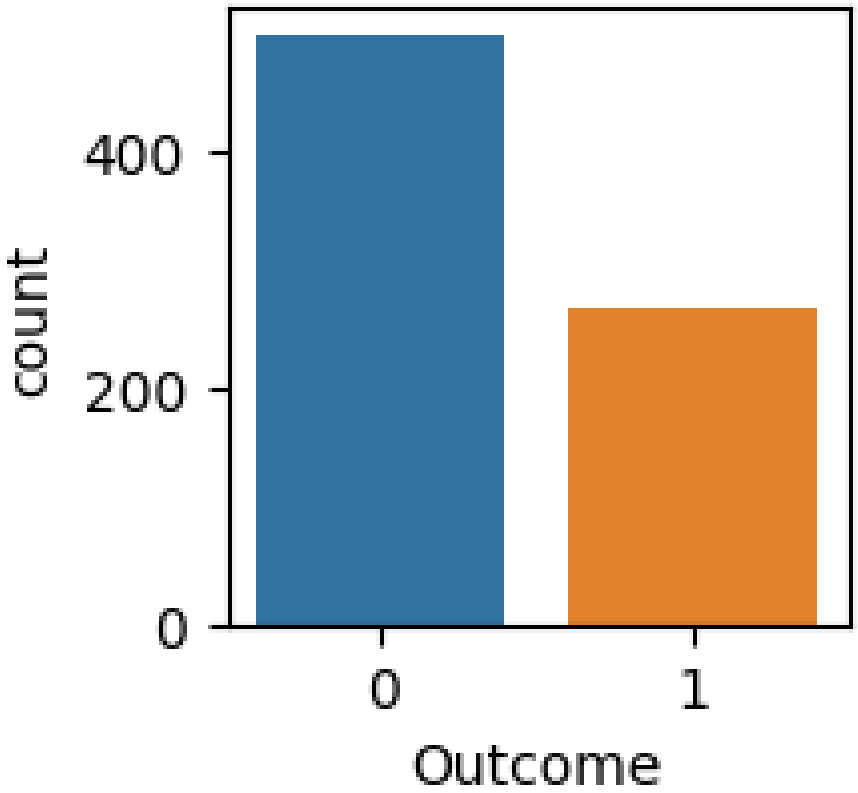

Figure-1: Dataset Outcome Count Plot

Correlation Matrix :

A correlation matrix shows the correlation between different variables in a matrix setting. However, because these matrices have so many numbers on them, they can be difficult to follow. Heatmap coloring of the matrix, where one color indicates a positive correlation, another indicates a negative correlation, and the shade indicates the strength of correlation, can make these matrices easier for the reader to understand. Code-snippet

```
corr = df.corr(method = 'pearson')
f, ax = plt.subplots(figsize=(5, 5))
cmap = sns.diverging_palette(10, 275, as_cmap=True)
sns.heatmap(corr, cmap=cmap, square=True, linewidths=0.5, cbar_kws={"s
```

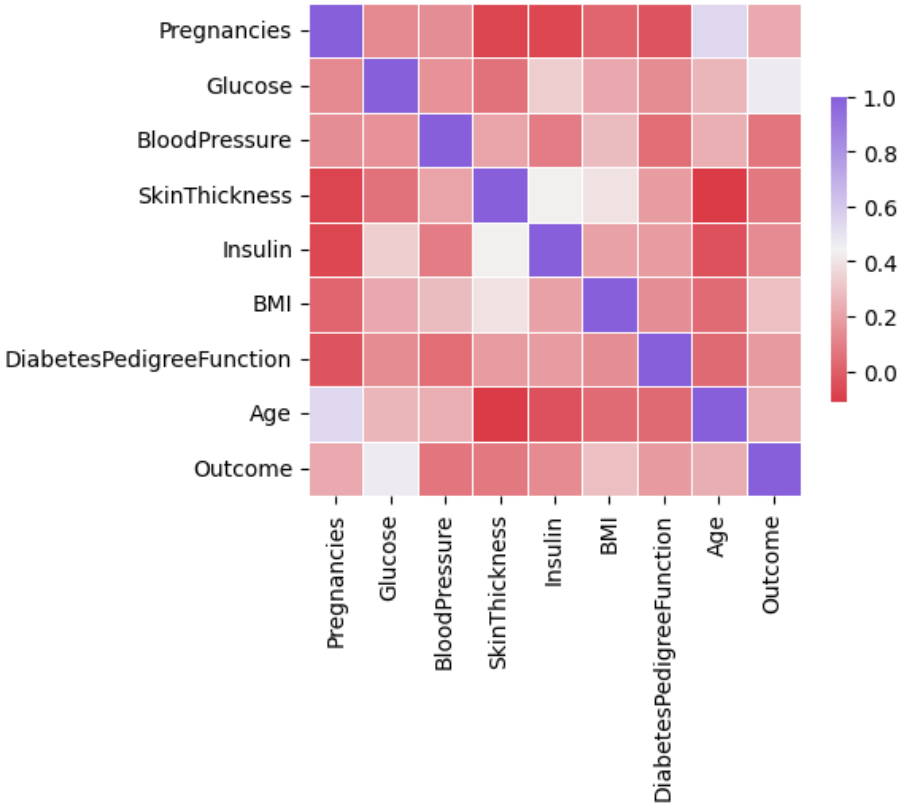

Figure-2: Correlation Heatmap

Split Dataset

Separates the features and target variable, and then splits them into training and testing sets. The training set (Xtrain and Ytrain) is used to train a machine learning model, while the testing set (Xtest and Ytest) is used to evaluate the model's performance.Finally single lair perceptron algorithm will be applied .Here test size = 0.1

```
X_train, X_test,
y_train,y_test = train_test_split(X,d, test_size=0.1)
```

Single Layer Perceptron Model

In the perceptron classs Intitial weight ,threshold value ,epoch ,learning rate declare then update weight factor in three way and finally the result will come.
Code Snippet:

```
from sklearn.metrics import accuracy_score
from random import randint
class Perceptron:
def __init__(self,input_size,epochs=100,alpha=0.02,method='m3'):
self.finals = []
self.epochs = epochs
self.alpha = alpha
self.input_size = input_size
weight = [randint(1, 5) for i in range(input_size + 1)]
weight[0] = 1.0
self.weight = weight
#print("initial weight is ")
print(weight)
self.method = method
#self.weight = np.zeros(input_size+1)
def activation(self,x):
return 1 if x>=0 else 0
ef predict(self,x):
z = np.dot(self.weight, x)
a = self.activation(z)
return a
```

Weight Update : Method 1

Here every time weight and input multiple each iteration and finally shows result of learned weghts. Code Snippet:

```
def learn(self,X,d):
for j in range(self.epochs):
sum = 0
```

```
for i in range(d.shape[0]):
x = np.insert(X[i],0,1)
y = self.predict(x)
if self.method == 'm1':
if y == 0 and d[i] == 1:
self.weight = self.weight + x
elif y == 1 and d[i] == 0:
self.weight = self.weight - x
```

Weight Update : Method 2 (Modified version)

Here every time learning rate will be multiplied finally shows result of learned weights.A positive gain term that controls the adaption rate.. Code Snippet:

```
def learn(self,X,d):
for j in range(self.epochs):
sum = 0
for i in range(d.shape[0]):
x = np.insert(X[i],0,1)
y = self.predict(x)
e = (d[i] - y)
elif self.method == 'm2':
if y == 0 and d[i] == 1:
self.weight = self.weight + self.alpha*x
elif y == 1 and d[i] == 0:
self.weight = self.weight - self.alpha*x
self.finals.append(abs(sum))
```

Weight Update : Method 3 (Widrow-Hoff delta rule)

Here every time learning rate will be multiplied finally shows result of learned weights.calculates The difference between the weighted sum and the required output, and calls that the error. Weight adjustment is then carried out in proportion to that error. d(i) is the desired response and y(i) is the actual response and there difference is here declared as e.
Code Snippet:

```
def learn(self,X,d):
final = []
```

```
for j in range(self.epochs):
sum = 0
for i in range(d.shape[0]):
x = np.insert(X[i],0,1)
y = self.predict(x)
e = (d[i] - y)
self.weight = self.weight + self.alpha*e*x
sum = sum + e
self.finals.append(abs(sum))
```

Model Fit

This Code section seems to be training the perceptron learning algorithm on some data and evaluating its performance on both training and testing sets.we use this perceptron learning algorithm using 3 method of weight updation.From Each method the outcome is leaned weight ,testing and training accuracy, which can be helpful for better understanding

Code-Snippet:

```
learning_rate = 0.02
iterations = 20
perceptron = Perceptron(input_size=input_size,
alpha = learning_rate,
epochs =iterations, method=f'm{i}')
perceptron.learn(X_train, y_train)
```

Evaluate Model and Accuracy

Perceptron learning algorithm on some data and evaluating its performance on both training and testing sets.we use this perceptron learning algorithm using 3 method of weight updation.From Each method the outcome is leaned weight ,testing and training accuracy, which can be helpful for better understanding.

Code-Snippet:

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
from random import randint
if __name__ == '__main__':
input_size = 2
learning_rate = 0.02
iterations = 10
for i in range(1, 4):
perceptron = Perceptron(input_size=input_size, alpha = learning_rate,
epochs = iterations, method=f'm{i}')
perceptron.learn(X_train, y_train)
print("The learned weight is ")
print(perceptron.weight)
print("Learning Rate = "+str(learning_rate))
y_pred_train = []
for i in range(X_train.shape[0]):
x= np.insert(X_train[i],0,1)
y_pred_train.append(perceptron.predict(x))
print("Training data Accuracy = "
+str(accuracy_score(y_train, y_pred_train)))
conftrain = confusion_matrix(y_train,y_pred_train)
y_pred_test = []
finals2 = []
for i in range(X_test.shape[0]):
x= np.insert(X_test[i],0,1)
y_pred_test.append(perceptron.predict(x))
print("Testing data Accuracy = "
+str(accuracy_score(y_test, y_pred_test)))
conftest = confusion_matrix(y_test, y_pred_test)
```

Initial weight, learned weight ,Training data accuracy and Testing data accuracy of each weight update method given below.

Result:
initial weight is
[1.0, 4, 1]
The learned weight is
[-413. 271. -57.]
Learning Rate = 0.02
Training data Accuracy = 0.7226772793053546
Testing data Accuracy = 0.7063636363636364

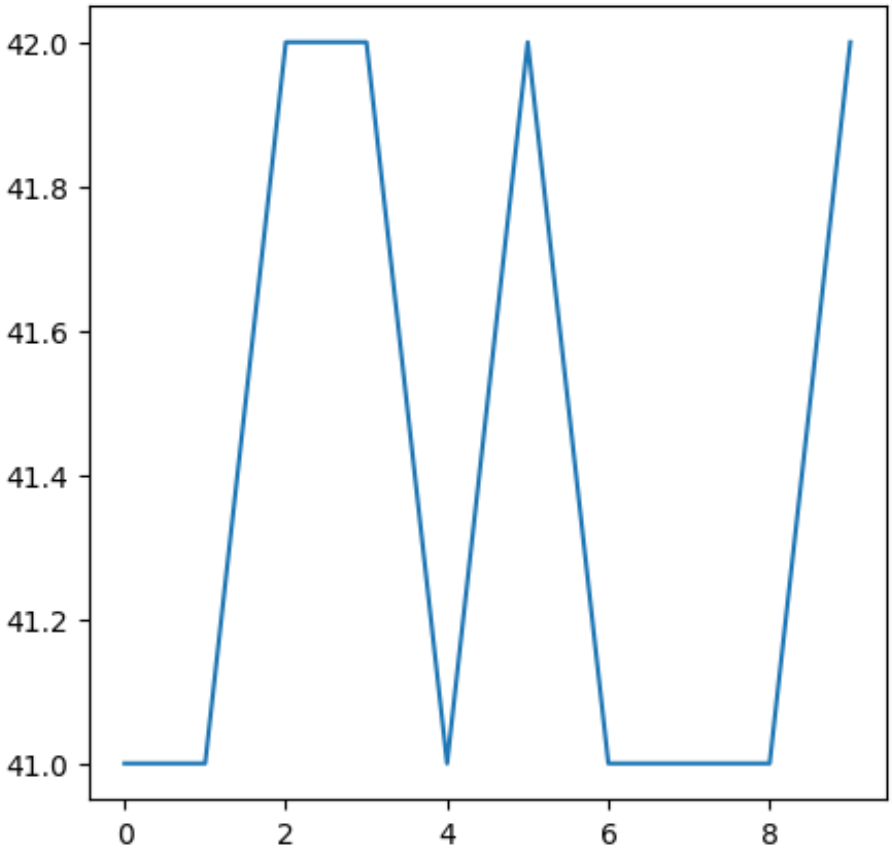

Figure-3:Train and Test Accuracy according to weight update Method-1

Result:
initial weight is
[1.0, 1, 1]
The learned weight is
[-7.34 4.54 -1.12]
Learning Rate = 0.02
Training data Accuracy = 0.742677279305372
Testing data Accuracy = 0.7363636363636311

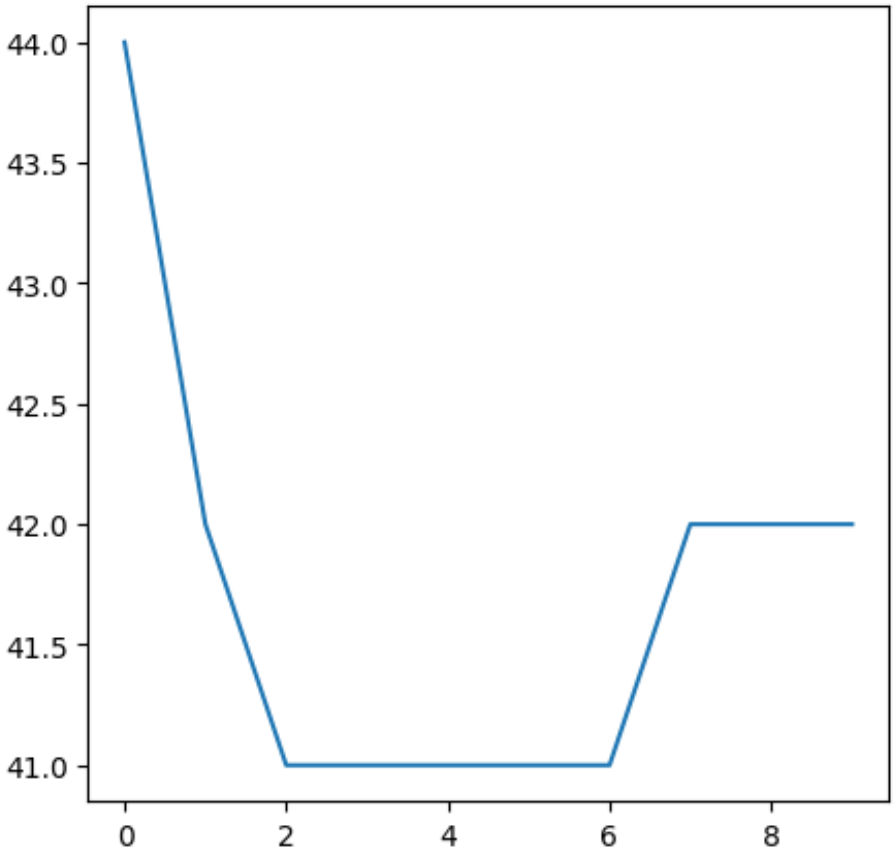

Figure-4:Train and Test Accuracy according to Weight update Method-2

Result:
initial weight is
initial weight is [1.0, 2, 2]
The learned weight is
[-7.28 5.38 -0.68]
Learning Rate = 0.02
Training data Accuracy = 0.742671079305372
Testing data Accuracy = 0.7363635463636311

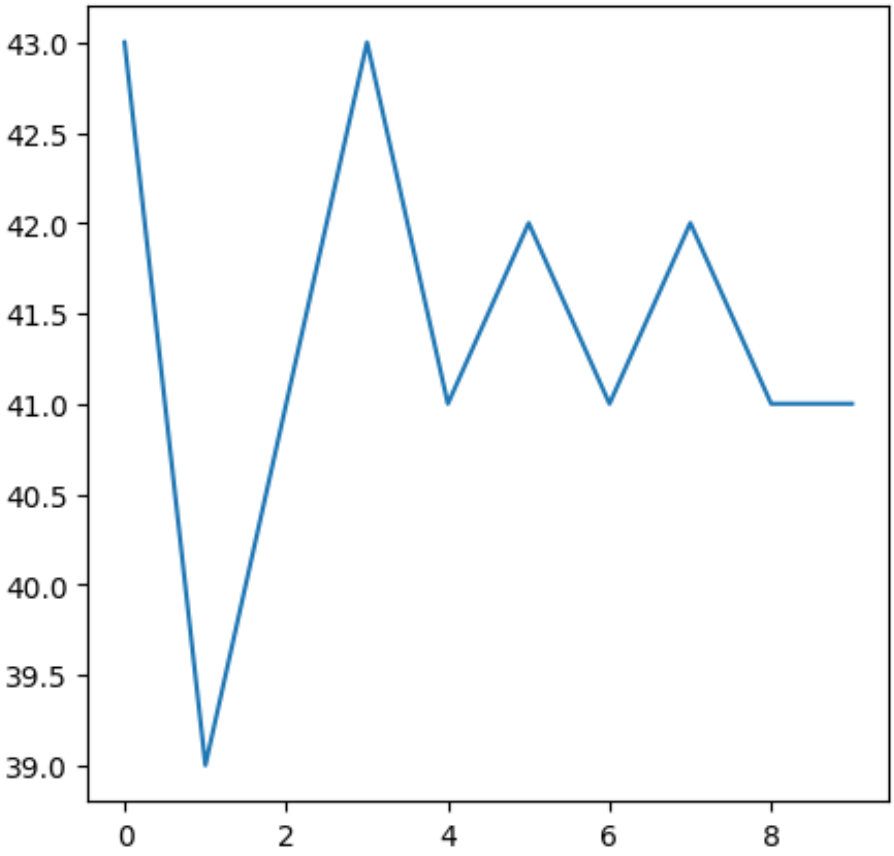

Figure-5:Train and Test Accuracy according to Weight update Method-3

Evaluation(Confusion) Matrics

confusion matrix visualize and summarize the performance of this trained model performance on trainning and testing and code seems correct and should display a heatmap with annotations for True Positives, True Negatives, False Positives, and False Negatives.
Confusion Matrix:Training Data

```
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
print("Confusion Matrix:Training Data")
df_cm = pd.DataFrame(conftrain, index = [i for i in ("True","False")],
columns = [i for i in ("True","False")])
plt.figure(figsize = (5,5))
```

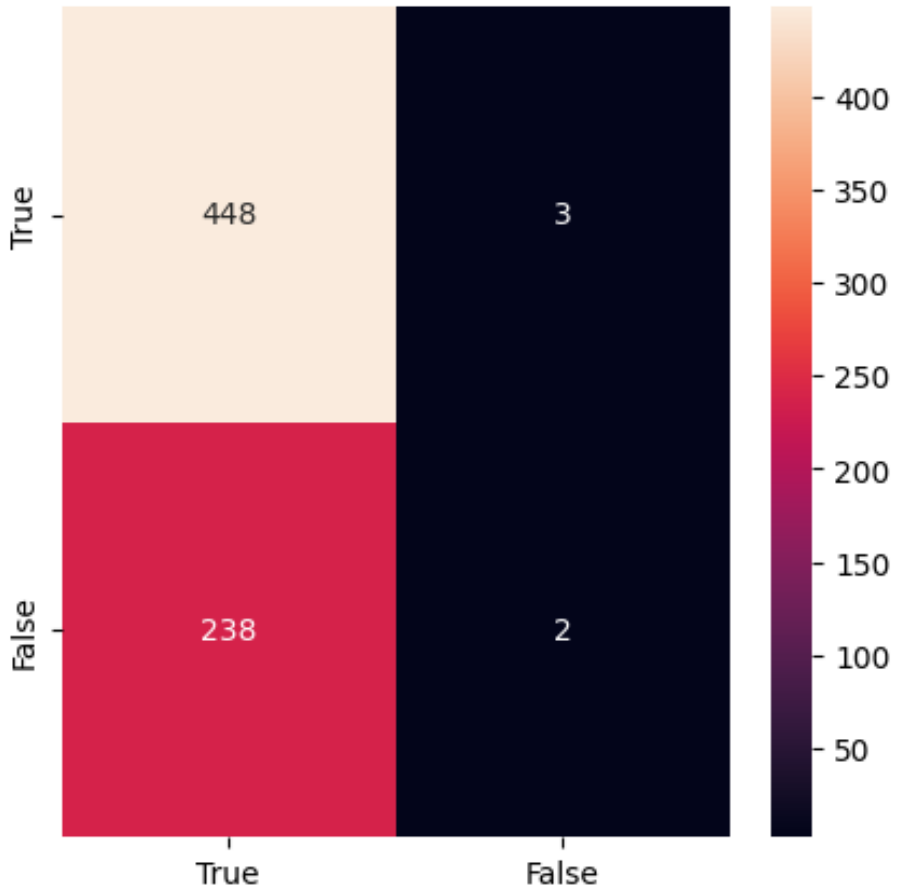

Figure-6:Confusion Matrix:Training Data

Confusion Matrix:Testing Data

```
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
print("Confusion Matrix:Testing Data")
df_cm = pd.DataFrame(conftest, index = [i for i in ("True","False")],
columns = [i for i in ("True","False")])
plt.figure(figsize = (5,5))
sn.heatmap(df_cm, annot=True, fmt='.0f')
```

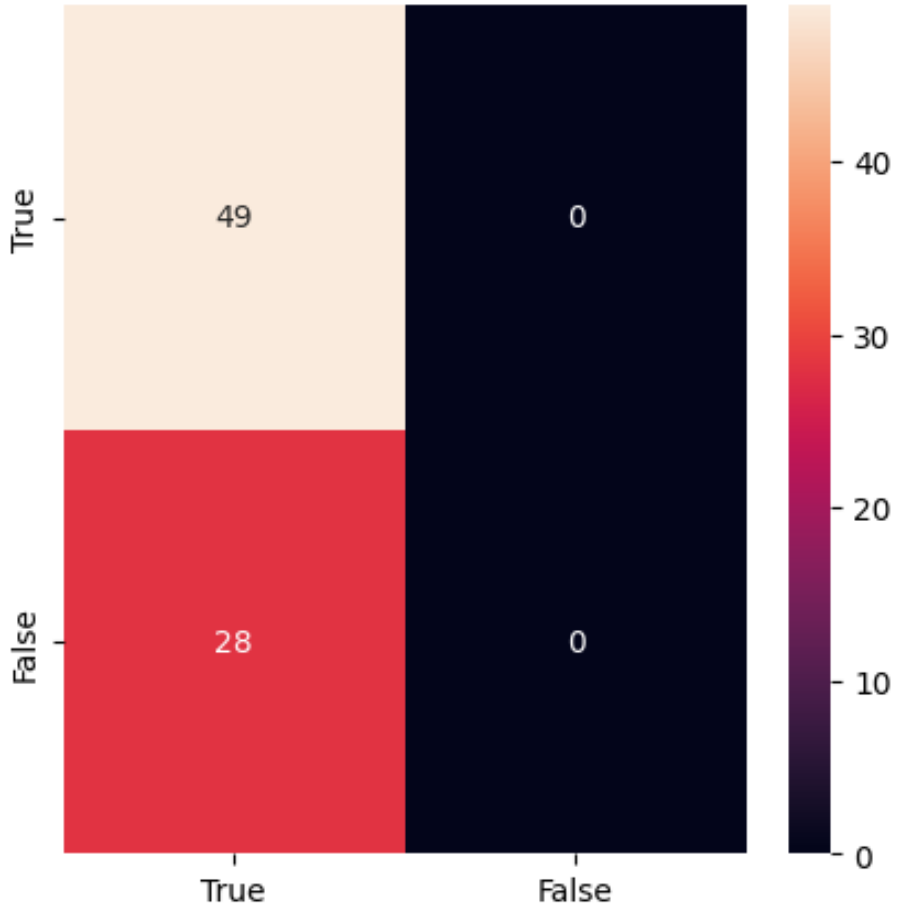

Figure-7:Confusion Matrix:Testing Data

Top 5 features:

```
import seaborn as sn
import pandas as pd
print(df_tmp)
d = df['Outcome'].values
X = df.drop('Outcome',axis=1).values
print(X.shape)
from sklearn.feature_selection import SelectKBest, chi2
X_new = SelectKBest(chi2, k=3).fit_transform(X,d)
X_new.shape
features_columns = df.columns
fs = SelectKBest(k=5)
fs.fit(X,d)
top_features = zip(fs.get_support(),features_columns)
print("The top 5 features are: ")
pp = pprint.PrettyPrinter(depth=4)
for i,j in top_features:
if i==True:
pp.pprint(j)
```

Outcome:

The top 5 features are:
'Pregnancies'
'Glucose'
'BloodPressure'
'SkinThickness'
'BMI'

Plot of Top 2 Features with the target value:

```
plt.scatter(df[ df['Outcome']==0.0 ]['Pregnancies'],
df[ df['Outcome']==0.0 ]['Glucose'], marker='o', label=0, color='y')
plt.scatter(df[ df['Outcome']==1.0 ]['Pregnancies'],
df[ df['Outcome']==1.0 ]['Glucose'], marker='o', label=1, color='r')
plt.xlabel('P')
plt.ylabel('G')
plt.show()
```

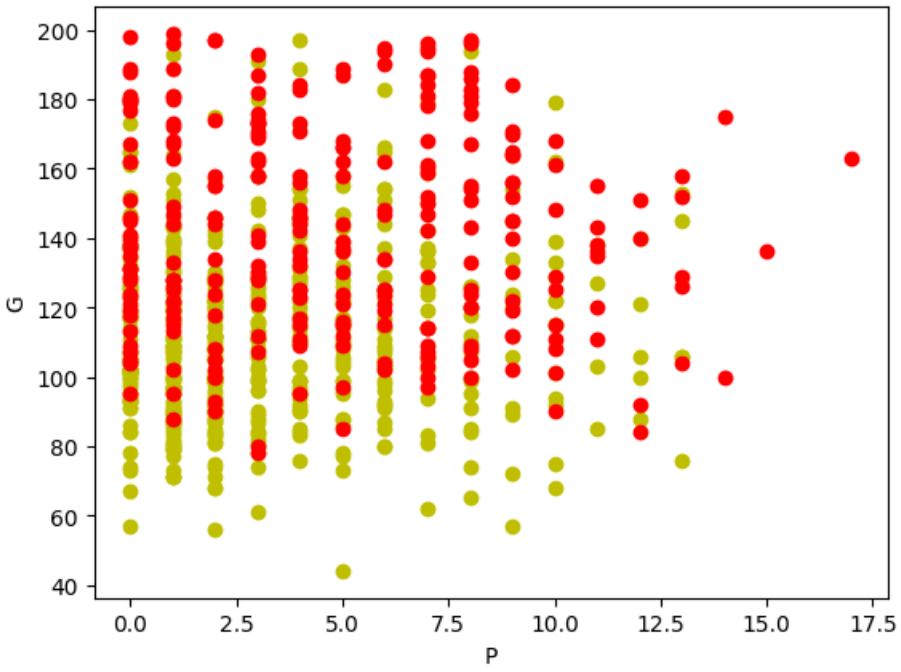

Figure-8:Plot of Top 2 Features with the target value

Limitations:

1.The biggest limitation is it can be used only to solve binary classification problems that are linearly separable. It can't learn non-linear boundaries.

2.This algorithm can only solve binary classification problems. But it can't solve multiclass problems.

3.The performance of a single-layer perceptron is highly sensitive to the initial weights which has seen in various tests.

4.Single-layer perceptrons do not have hidden layers, which limits their ability to model complex relationships between inputs.

Improvements

Here are some improvements and considerations for enhancing the performance and capabilities of a single-layer perceptron:
1. Dataset is not enough good. Better dataset can give better result.

2.Non-linear Activation Function: Replace the linear activation function with a non-linear one, such as the sigmoid, hyperbolic tangent (tanh), or rectified linear unit (ReLU). This allows the perceptron to model non-linear relationships in the data.

3.Multiple Epochs:Train the model for multiple epochs to allow it to see the entire training dataset multiple times. This is particularly important when using iterative training algorithms.

4.Learning Rate Adjustment: Experiment with different learning rates or adaptive learning rates. Too high a learning rate may cause overshooting, while too low a learning rate may result in slow convergence.

Conclusion

The single-layer perceptron (SLP) is very useful to linearly separable binary classification but struggles with non-linear data. Its simplicity and interpretability make it suitable for basic tasks.. For more intricate tasks, adopting multi-layer perceptrons or other sophisticated neural network architectures is essential to overcome the constraints and enhance the model's capability.

References

[1] Neural Computing: An Introduction by R-Beale and T-Jackson

[2] An Optimized Single Layer Perceptron-based Approach for Cardiotocography Data Classification((IJACSA) International Journal of Advanced Computer Science and Applications,Vol. 13, No. 10, 2022)