בס"ד

PROJECT 1

Parts A and B

Taliya shitreet 314855099

Renana Rimon 207616830

Talia Meizlish 207883430

Esther Binnes 211328109

~All members of the group are on evening class~

29.04.2022

Adeline Algorithm

ADALINE is a single-layer artificial neural network. It consists of a weight, a bias, and a summation function.

In ADALINE Algorithm the weights are adjusted according to the weighted sum of the inputs (the net), but it uses a linear division.

The algorithm consists of three parts:

1. Feedforward - Learning input system.
2. Backpropagation -Calculation of the learning error
3. Correcting the weights according to the calculation of the error

1.

$$y_{out} = \sum_{j=1}^{n} x_j w_j + bias$$

weight vector, constant bias, input vector

2.

$$Error = (y_{target} - y_{out})^2$$

3.

$$W_{(new)} = W_{(old)} + \mu \cdot (error) \cdot X_i$$

Input Vector, Learning Rate

create data set

```
import random
import numpy as np

def createDataSetA(num: int) -> (np.ndarray, np.ndarray):
    dataSet = np.ndarray((num, 3))
    for i in range(num):
        m = random.randint(-10000, 10000)
        n = random.randint(-10000, 10000)
        dataSet[i, 0] = m / 100
        dataSet[i, 1] = n / 100
        dataSet[i, 2] = 1 if n / 100 > 1 else -1
    return dataSet[:, 0:2], dataSet[:, 2]


def createDataSetB(num: int) -> (np.ndarray, np.ndarray):
    dataSet = np.zeros((num, 3))
    for i in range(num):
        m = random.randint(-10000, 10000)
        n = random.randint(-10000, 10000)
        x = m/ 100
        y = n/100
        dataSet[i, 0] = x
        dataSet[i, 1] = y
        dataSet[i, 2] = 1 if 4 <= ((x ** 2) + (y ** 2)) <= 9 else -1
    return dataSet[:, 0:2], dataSet[:, 2]

def createDataSetBLargeRadius(num: int) -> (np.ndarray, np.ndarray):
    dataSet = np.zeros((num, 3))
    for i in range(num):
        m = random.randint(-10000, 10000)
        n = random.randint(-10000, 10000)
        x = m/ 100
        y = n/100
        dataSet[i, 0] = x
        dataSet[i, 1] = y
        dataSet[i, 2] = 1 if 4 <= ((x ** 2) + (y ** 2)) <= 3000 else -1
    return dataSet[:, 0:2], dataSet[:, 2]
```

plotting figures

```
def build_df(X, y):
    x_df = pd.DataFrame(X,columns=['x','y'])
    y_df = pd.DataFrame(y,columns=['target'])
    df = x_df.join(y_df)
    return df

def build_df_pred(adaline, X):
    x_df = pd.DataFrame(X,columns=['x','y'])
    pred = adaline.predict(X)
    y_pred_df = pd.DataFrame(pred,columns=['target'])
    df_pred = x_df.join(y_pred_df)
    return df_pred

def plot_real_and_predicted(adaline, X, y):
    df = build_df(X, y)
    df_pred = build_df_pred(adaline, X)
    f, ax = plt.subplots(1,2)
    ax[0].set_title("train set value")
    ax[1].set_title("train set predicted")
    for index, row in df.iterrows():
        if row['target']==1:
            ax[0].plot(row['x'], row['y'] , markersize=1,marker="o" , color="green")
        if row['target']==-1:
            ax[0].plot(row['x'], row['y'] , markersize=1,marker="o" , color="pink")
    for index, row in df_pred.iterrows():
        if row['target']==1:
            ax[1].plot(row['x'], row['y'] , markersize=1,marker="o" , color="green")
        if row['target']==-1:
            ax[1].plot(row['x'], row['y'] , markersize=1,marker="o" , color="pink")
    plt.show()

def my_confusion_matrix(adaline, X_test, y_test):
    score = adaline.score(X_test, y_test)
    print("score : ", score)
    y_pred = adaline.predict(X_test)
    print(classification_report(y_test, y_pred))
    sns.set(font_scale=1.3)
    fig, ax = plt.subplots(figsize=(4, 4))
    ax = sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cbar=True , cmap="YlGnBu")
    plt.xlabel("True Labels")
    plt.ylabel("Predicted Labels")
```

main

```
def partA(X_train, y_train, X_test, y_test):
    a = Adaline(50, 0.0001)
    a.fit(X_train, y_train)
    plot_real_and_predicted(a, X_train, y_train)
    my_confusion_matrix(a, X_test, y_test)

def partB(X_train, y_train, X_test, y_test):
    a = Adaline(20, 0.1)
    a.fit(X_train, y_train)
    plot_real_and_predicted(a, X_train, y_train)
    my_confusion_matrix(a, X_test, y_test)

def partBLargeRadius():
    X_train, y_train = createDataSetBLargeRadius(1000)
    X_test, y_test = createDataSetBLargeRadius(1000)
    a = Adaline(20, 0.1)
    a.fit(X_train, y_train)
    plot_real_and_predicted(a, X_train, y_train)
    my_confusion_matrix(a, X_test, y_test)

def quantities(X_train, y_train):
    df = build_df(X_train, y_train)
    df.target.value_counts().plot(kind='bar',  color=['orange' , 'mediumseagreen'] , figsize=(6, 5));
    plt.xticks([0, 1], ["1", "-1"], rotation=0);
    plt.title("Points Values")
    plt.ylabel("Amount")

    print("The number of points with value 1: ", len(df[df['target'] == 1]))
    print("The number of points with value -1: ", len(df[df['target'] == -1]))

def my_heat_map(X_train, y_train):
    df = build_df(X_train, y_train)
    plt.subplots(figsize=(5, 5))
    sns.heatmap(df.corr(), linewidths=0.4 , square=True, cmap = 'Oranges')
    plt.title('Heatmap for the Dataset')
    plt.show()
```

Adaline Algorithm

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import math
from sklearn.metrics import confusion_matrix, classification_report
```

```
import numpy as np


class Adaline:

    def __init__(self, n_iter=100, learning_rate=0.01, random_state=42):
        self.n_iter = n_iter
        self.random_state = random_state
        self.learning_rate = learning_rate

    def net_input(self, X: np.ndarray):
        """
        sum(Xi*Wi)+Bi
        (Bi is the last col in X)
        :param X:
        :return:
        """
        return np.dot(X/100, self.w)

    def random_weights(self, X: np.ndarray):
        """
        generate random weights for each feature.
        weight[-1]: bias
```

```
    def fit(self, X: np.ndarray, y: np.ndarray):
        """
        train the model - find the best weights to predict.
        in each iteration,
        fix weight according to each sample error
        :param X:
        :param y:
        :return:
        """
        bias = np.ones([X.shape[0], 1])  # add bias column
        np.append(X, bias, axis=1)
        self.random_weights(X)
        for i in range(self.n_iter):
            for xi, target in zip(X, y):
                output = self.net_input(xi)
                error = target - output
                self.w += self.learning_rate * (xi/100).dot(error)

    def predict(self, X: np.ndarray):
        """
        predict X_test, using the new weights.
        if output >= threshold --> 1
        else: -1
        :param X:
        :return:
        """
        return np.where(self.net_input(X) >= 0.0, 1.0, -1.0)

    def score(self, X: np.ndarray, y: np.ndarray):
        """
        present of true prediction
        :param X:
        :param y:
        :return:
        """
        counter = 0
        for x, target in zip(X, y):
            p = self.predict(x)
            if p == target:
                counter += 1

        return counter / y.shape[0]
```

PART A

We have created a data set as follows:

The data is all data points where:

- x is of the form m/100
- m: an integer between -10000 and +10000.
- y: is of the form n/100
- n: an integer between - 10000 and +10000.

Suppose that all data points with y >1 have the value 1; all other points have the value -1.

The test and training sets are below:

```
df = build_df(X_train_A,y_train_A)
df
```

	x	y	target
0	-18.77	-53.84	-1.0
1	-68.28	-83.42	-1.0
2	-4.91	25.87	1.0
3	37.01	-18.65	-1.0
4	-47.67	7.29	1.0
...	...	...	...
995	16.41	-26.82	-1.0
996	0.62	37.78	1.0
997	8.56	21.79	1.0
998	-14.62	91.87	1.0
999	-3.65	51.96	1.0

1000 rows × 3 columns

1st random train data

```
df = build_df(X_test_A1,y_test_A1)
df
```

	x	y	target
0	-61.41	54.18	1.0
1	-82.42	91.22	1.0
2	90.15	-76.67	-1.0
3	54.73	-36.26	-1.0
4	63.99	-60.74	-1.0
...	...	...	...
995	-40.79	-61.48	-1.0
996	-56.89	-33.53	-1.0
997	-98.85	-17.07	-1.0
998	-96.00	50.28	1.0
999	63.06	63.39	1.0

1000 rows × 3 columns

1st random test data

```
df = build_df(X_test_A2,y_test_A2)
df
```

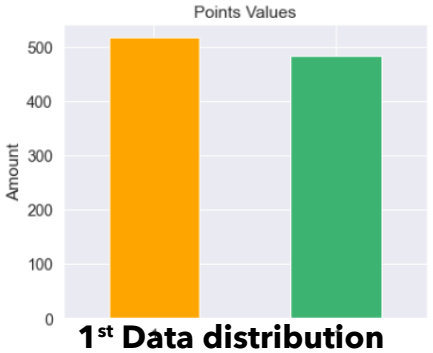
	x	y	target
0	78.09	-98.12	-1.0
1	99.22	61.69	1.0
2	-22.06	-28.94	-1.0
3	92.95	55.88	1.0
4	48.68	34.39	1.0
...	...	...	...
995	41.86	-53.54	-1.0
996	-14.68	-61.72	-1.0
997	2.65	-85.53	-1.0
998	-58.10	63.58	1.0
999	-7.93	18.15	1.0

1000 rows × 3 columns

2nd random test data

```
quantities(X_train_A, y_train_A)
```

```
The number of points with value 1:  517
The number of points with value -1:  483
```

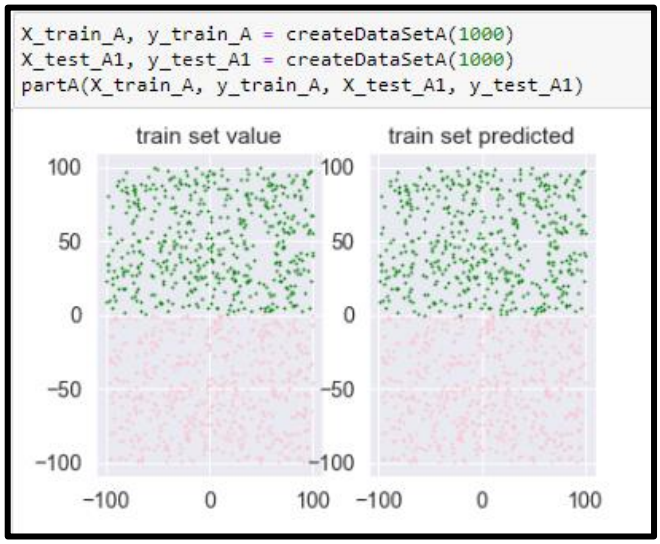


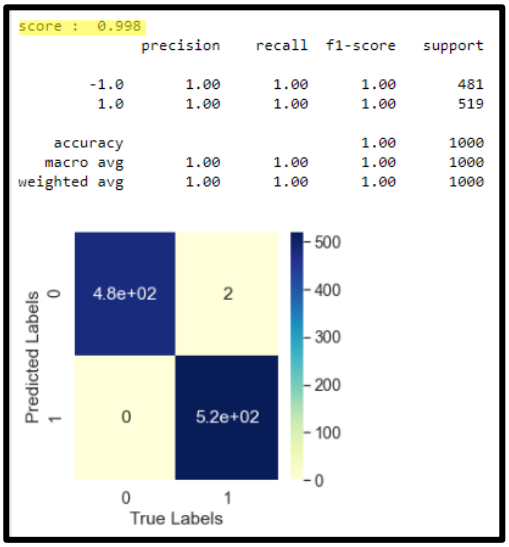
1st Data distribution

The classification of points in Part A is a linear classification (n/100), and therefore it is observed that the ADALINE model will be able to classify many of the points correctly and obtain a high score. Because we will use a linear division, the model adjusts itself quickly and does not require many iterations to achieve a good result.

We tried to use different test set sizes and a different number of iterations or learning-rate, but we did not see a difference as there is a good fit anyway.

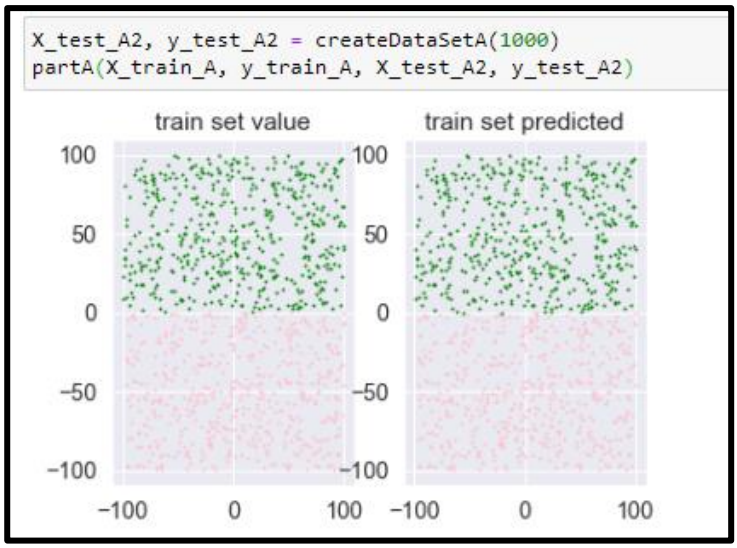
Results of the first Data set - Score: 0.998

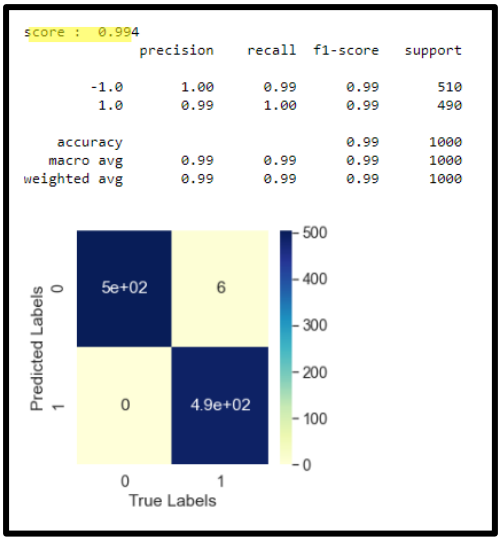




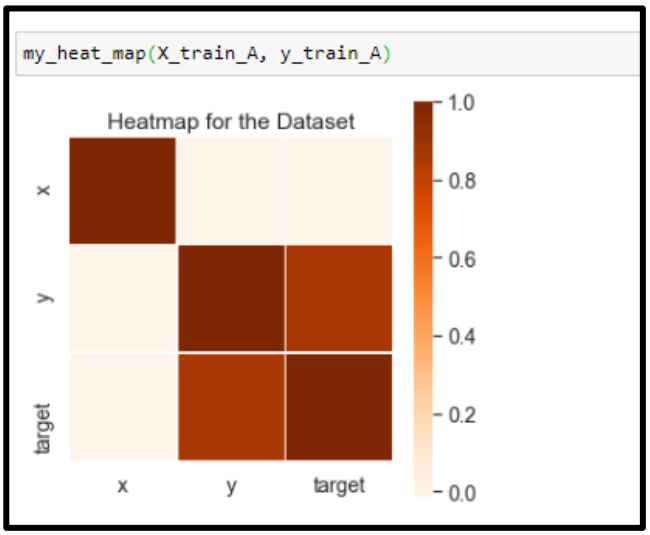
As we can see, the score obtained is high and consistent on additional test sets.

Results for the second Data set - Score: 0.994





Below we can see a heat map describing the relations between x, y, and target. Indeed, there is a high correlation, this correlation has significantly affected the high score.



PART B

We have created the dataset as follows:

points such that has value 1 only if 4 <= x^2 + y^2 <= 9

```
df = build_df(X_train_B,y_train_B)
df
```

	x	y	target
0	-17.77	-59.31	-1.0
1	99.19	88.57	-1.0
2	-59.96	-91.92	-1.0
3	-43.24	4.90	-1.0
4	12.64	96.80	-1.0
...	...	...	...
995	-17.53	-35.34	-1.0
996	72.92	-21.92	-1.0
997	-14.03	33.98	-1.0
998	18.36	-61.73	-1.0
999	-97.61	-21.29	-1.0

1000 rows × 3 columns

1st train data

```
df = build_df(X_test_B1,y_test_B1)
df
```

	x	y	target
0	-93.98	30.49	-1.0
1	-20.12	-46.39	-1.0
2	-86.62	67.65	-1.0
3	-30.03	-46.35	-1.0
4	40.81	-22.25	-1.0
...	...	...	...
995	39.47	-19.04	-1.0
996	25.05	8.69	-1.0
997	11.53	97.95	-1.0
998	-81.51	36.05	-1.0
999	-90.85	38.32	-1.0

1000 rows × 3 columns

1st test data

```
df = build_df(X_test_B2,y_test_B2)
df
```

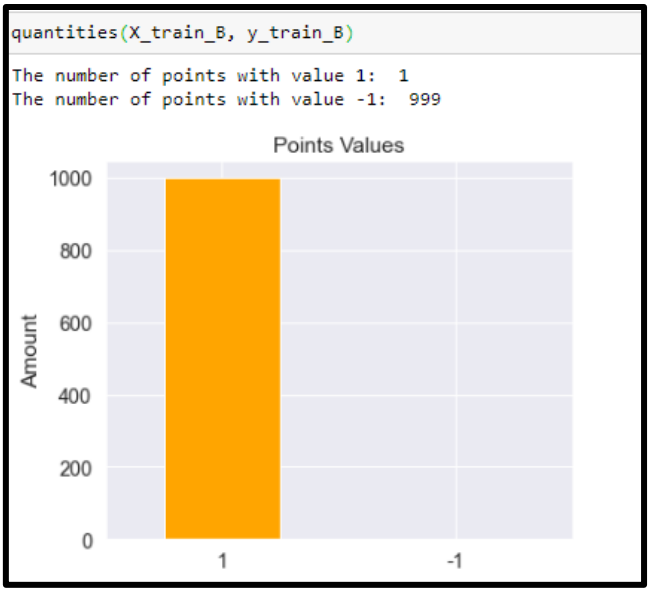
	x	y	target
0	42.28	-26.02	-1.0
1	88.91	37.14	-1.0
2	78.89	-8.44	-1.0
3	59.95	-6.98	-1.0
4	-18.67	-96.78	-1.0
...	...	...	...
995	-36.49	-34.32	-1.0
996	-91.08	43.32	-1.0
997	11.85	-5.66	-1.0
998	-19.89	31.70	-1.0
999	26.04	-78.14	-1.0

1000 rows × 3 columns

2nd test data

```
quantities(X_train_B, y_train_B)
```

```
The number of points with value 1:  1
The number of points with value -1:  999
```



1st data distribution

Compared to Part A, the points in Part B are not classified in a linear division, they are classified as a circle. (4 <= x^2 + y^2 <= 9).

At first, we didn't understand why the model doesn't work on this dataset, especially after the great results of part A.

Here is our explanation:

Because the desired radius is very small, few points (maybe even just one) are obtained within the desired radius. Adaline is a **linear** model, and therefore its accuracy percentages are very low.

As can be seen, the values predicted linearly in contrast to the original classification of points.

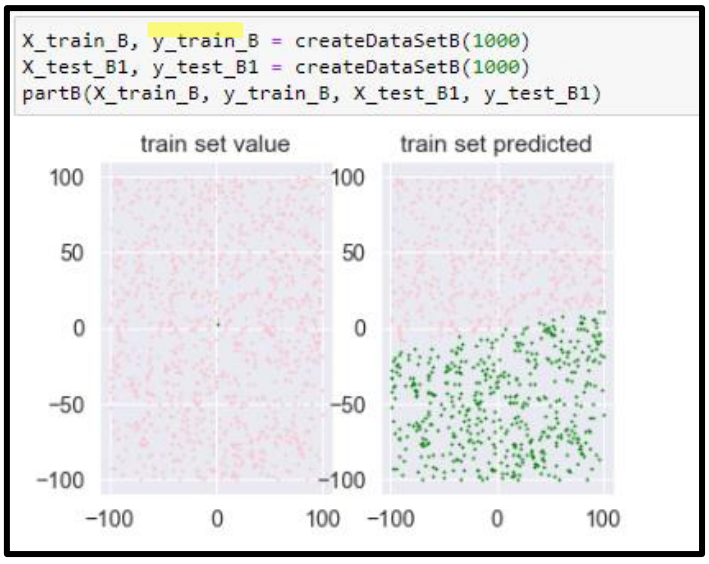
That is, the ADALINE model does not fit the classification in Part B.

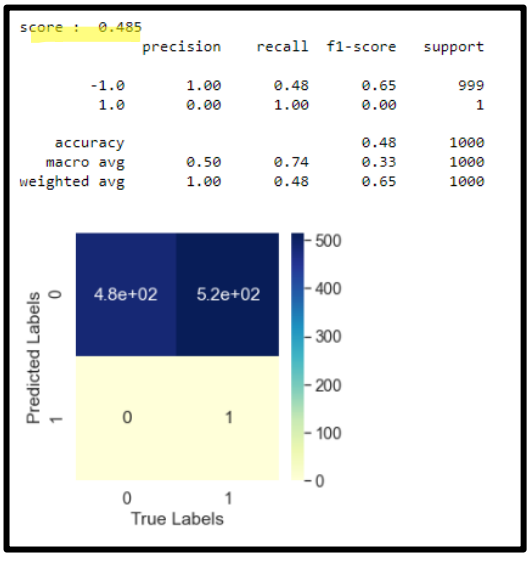
To test our model well, we test it with different parameters:

1. Learning rate
2. Number of iteration (n_iter)

Due to the explanation above, there is no significant difference in the result.This can be seen on several different test sets.

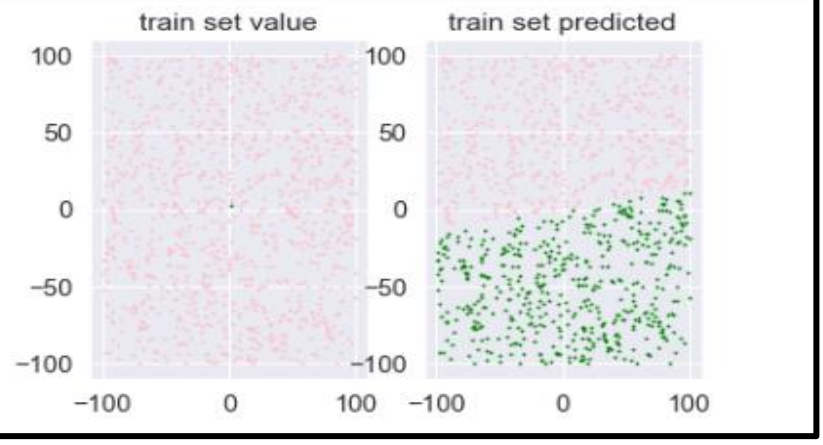
Results for the first Data set - Score: 0.485



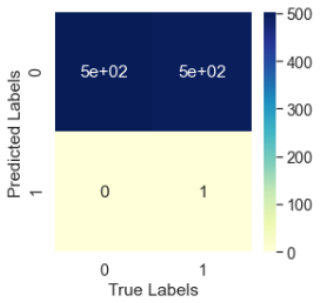


Results for the second Data set - Score: 0.503

```
X_test_B2, y_test_B2 = createDataSetB(1000)
partB(X_train_B, y_train_B, X_test_B2, y_test_B2)
```



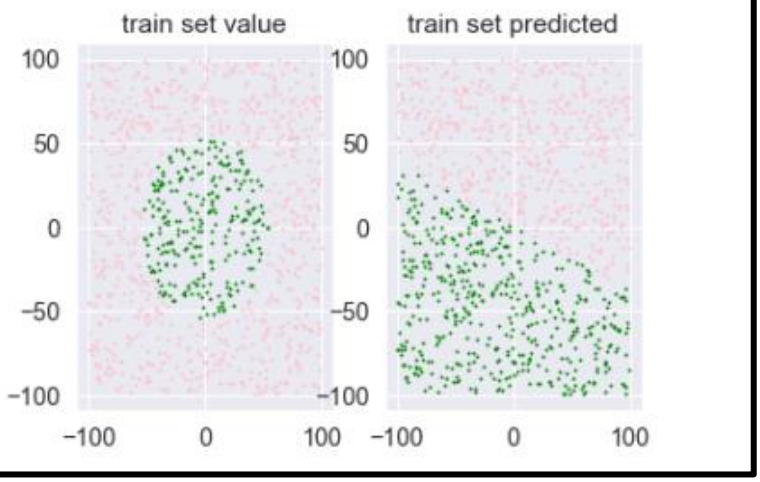

```
score :  0.503
              precision    recall  f1-score   support

        -1.0       1.00      0.50      0.67       999
         1.0       0.00      1.00      0.00         1

    accuracy                           0.50      1000
   macro avg       0.50      0.75      0.34      1000
weighted avg       1.00      0.50      0.67      1000
```



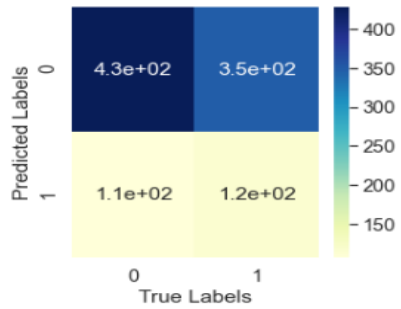
In addition, it seems that only a few points were classified as 1, this is because the desired radius is small.

Using a larger radius shows that we have indeed obtained a circuit that does not correspond to the efficiency of ADALINE.

```
partBLargeRadius()
```




```
score :  0.544
              precision    recall  f1-score   support

        -1.0       0.80      0.55      0.65       777
         1.0       0.25      0.52      0.34       223

    accuracy                           0.54      1000
   macro avg       0.52      0.54      0.49      1000
weighted avg       0.68      0.54      0.58      1000
```



Below we can see a heat map describing the relations between x, y, and target. Indeed, there is a low correlation, this correlation has significantly affected the low score.