Project 1 NeuroComputing

Israel yacobovich, 212403679 ziv morgan, 209904606

part A

we built and trained an Adaline to distinguish between dots above or beneath the y = 1 line as requested using python.

Chart, line chart

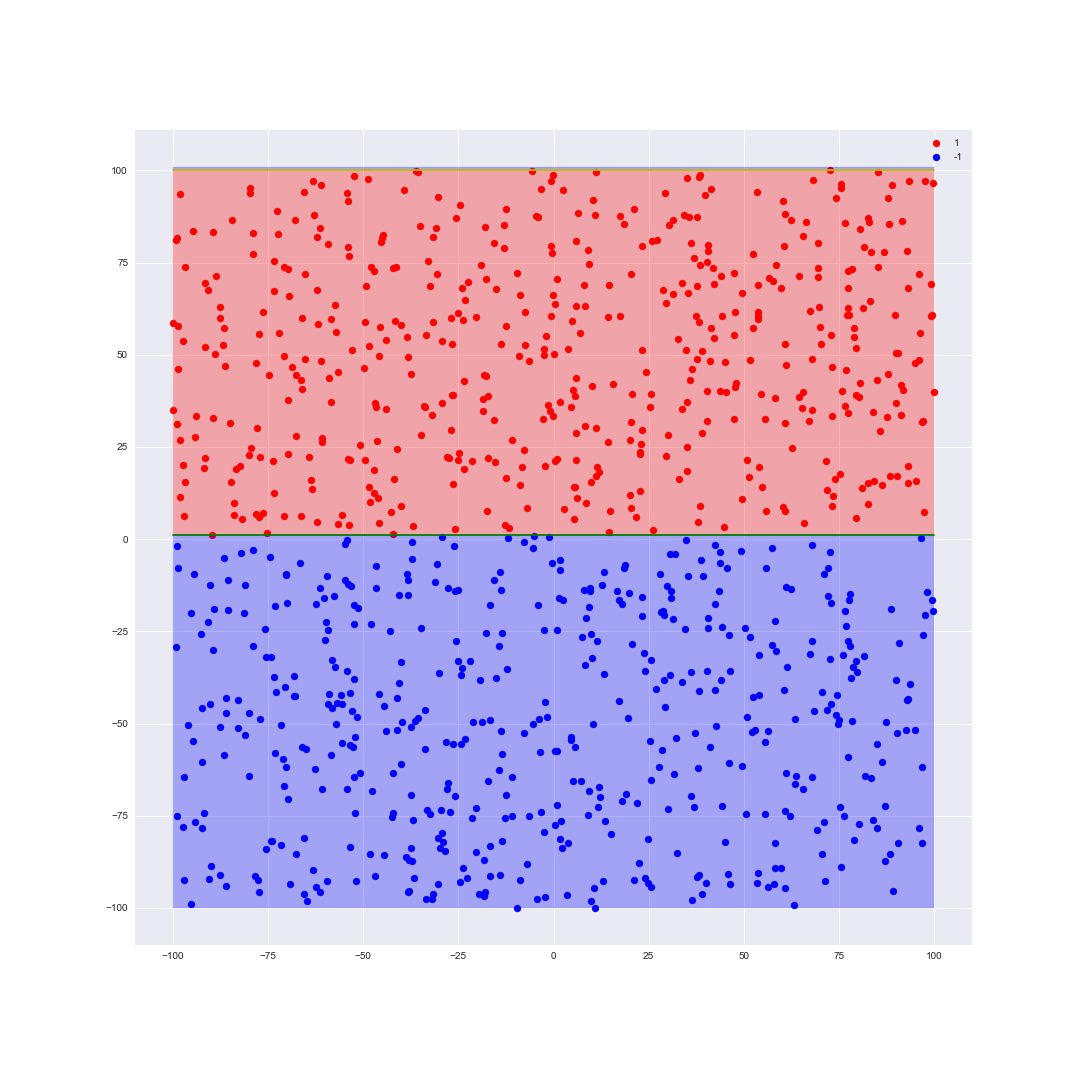
Description automatically generatedChart, treemap chart

Description automatically generated

Graphical user interface, chart

Description automatically generatedhere are relevant graphs and other forms of data we gathered from the program:

we can observe the successful learning process. The error rate goes down to almost 0.

This is no surprise cause the Adaline model should be able to classify linearly separable input sets

**Our code:**

class Adaline:

    def \_\_init\_\_(self,seed:int = None, learning\_rate:float=0.00001, bias:float=0.2, repeats:int=10000, possible\_targets=None,epsilon=0.00000001, activationFunction=lambda x: 1 if x>= 0 else -1):

        self.rng = np.random.RandomState(seed)

        self.repeats = repeats

        self.learning\_rate = learning\_rate

        self.bias = bias

        self.possible\_targets = possible\_targets

        self.epsilon = epsilon

        self.activationFunction = activationFunction

    def calc(self, sample):

        return self.bias + np.dot(self.weights,sample)

    def fit(self, data, target, verbose:int=0, precision:int=4, max\_step:int=10, logRate:int=5) -> dict:

        #make sure we are using a numpy array of shape (fetures, samples)

        if isinstance(data,pd.DataFrame):

            data = data.to\_numpy().T

        self.weights = self.rng.rand(data.shape[0])

        #logging

        past\_MSE = [float("inf")]

        #save each epoch

        stats = {"Epoch":[],"bias":[],"MSE":[]}

        for feature\_i in range(data.shape[0]):

            stats["w\_"+str(feature\_i)] = []

            stats["deltaW\_"+str(feature\_i)] = []

        #make sure we don't enter an infinite loop if our model doesn't converge

        for i in range(self.repeats):

            errors = []

            #repeat for all data samples

            for x\_i in range(data.shape[1]):

                #sum(w\_i\*x\_i) for all features

                Y = self.calc(data[:,x\_i])

                pred = 0

                pred = self.activationFunction(Y)

                delta = 0

                #adjust weights and bias if we guessed wrong

                if pred != target[x\_i]:

                    delta = (target[x\_i]-Y)

                    deltaW = self.learning\_rate\*delta\*data[:,x\_i]

                    self.weights = self.weights + deltaW

                    self.bias = self.bias+self.learning\_rate\*delta

                #squared error

                errors.append((delta\*\*2))

            if len(errors) == 0:

                break

            mse = sum(errors)/len(errors)

            past\_MSE.append(mse)

            if i%logRate == 0:

                #save stats of each value in our data

                stats["bias"].append(round(self.bias,precision))

                stats["MSE"].append(round(mse,precision))

                stats["Epoch"].append(i)

                for feature\_i in range(data.shape[0]):

                    stats["w\_"+str(feature\_i)].append(round(self.weights[feature\_i],precision))

                    stats["deltaW\_"+str(feature\_i)].append(round(deltaW[feature\_i],precision))

            #print stats

            if verbose>=1 and i%logRate == 0:

                print("Epoch",i)

                print(mse)

                print("errors:",len([err for err in errors if err>0]))

                if verbose>=2:

                    df = pd.DataFrame(stats)

                    print(df)

                print("--------")

            #stop if our mse is not changing at all

            if len(past\_MSE) > max\_step and (all(abs(past\_MSE[-1] - x)<self.epsilon for x in past\_MSE[-max\_step:]) or np.all(np.diff( np.array(past\_MSE[-max\_step:])) >= 0)):

                break

        model\_performance = pd.DataFrame(stats)

        return model\_performance

    def predict(self,data):

        #make sure we are using a numpy array of shape (fetures, samples)

        if isinstance(data,pd.DataFrame):

            data = data.to\_numpy().T

        #get vector of predicted target values

        target = []

        for x\_i in range(data.shape[1]):

            target.append(self.activationFunction(self.calc(data[:,x\_i])))

        return np.array(target)

code for generating data:

def generate\_points(size=1000, seed=None,\_min=-10000, \_max=10001, data:any=[]):

    #valid input is an array of shape (2,n) for any positive n or dictionary/dataframe with x and y as keys

    seed = get\_seed(seed)

    out = []

    rng = np.random.RandomState(seed)

    if len(data) != 0:

        if isinstance(data,(dict,pd.DataFrame)):

            out = np.array( [data["x"],data["y"]])

        else:

            out = np.array(data[:2])

        #slice out to the correct size

        if size != None and out.shape[1] > size:

            out = out[:,0:size]

        elif size != None and out.shape[1] < size:

            extra = rng.randint(\_min,\_max,(2,size-out.shape[1]))

            out = np.concatenate((out,extra),axis=1)

    else:

        out = rng.randint(\_min,\_max,(2,size))

    return out

#generate a DataSet for Part A

def create\_A(size=1000,raw\_data:np.ndarray=[],seed=None,\_min=-10000,\_max=10001):

    seed = get\_seed(seed)

    raw\_data = generate\_points(size,seed,\_min,\_max,raw\_data)

    #true iff y>100 (implies y/100 > 1)

    target = np.ma.masked\_greater(raw\_data[1],100).mask

    #turn change from 0 and 1 to -1 and 1

    return pd.DataFrame({"x":raw\_data[0]/100, "y":raw\_data[1]/100, "target":target\*2-1})

#generate a data set for Part B

def create\_B(size=1000,raw\_data:np.ndarray=[],seed=None,\_min=-10000,\_max=10001):

    seed = get\_seed(seed)

    raw\_data = generate\_points(size,seed,\_min,\_max,raw\_data)

    conditionArg = raw\_data[0]\*\*2+raw\_data[1]\*\*2

    #true iff 40000 <= x^2 + y^2 <= 90000 (implies 4 <= x^2 + y^2 <= 9)

    target = np.ma.masked\_where((40000<=conditionArg) & (conditionArg<=90000),conditionArg).mask

    #turn change from 0 and 1 to -1 and 1

    return pd.DataFrame({"x":raw\_data[0]/100, "y":raw\_data[1]/100, "target":target\*2-1})

Part B

This time we were tasked on training the Adaline to determined whether or not a point was in a ring

that is: outside of a small circle and inside a bigger one.

This time we weren’t able to produce results as impressive as in part A. again this is shouldn’t come as a surprise

since as mentioned: the Adaline can only solve linearly separable sets.

However we did switch to a different activation function that helped. Yet of course one shouldn’t confuse

this with a successful model for this mission because over all it wasn’t able to find a good fit, to 'understand the problem' if you will.

here is relevant information from this part:

Chart, box and whisker chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart

Description automatically generatedhere are relevant graphs and other forms of data we gathered from the program:

Chart

Description automatically generated

We tried again with a bigger data-set and we saw some improvement:

Chart

Description automatically generated

Chart, line chart

Description automatically generated

here are relevant graphs and other forms of data we gathered from the program:

Graphical user interface, chart

Description automatically generated

תמונה שמכילה טקסט, מכשירי כתיבה, מעטפה

התיאור נוצר באופן אוטומטי

We also tried to manually input some points that should classify as 1 to get a more balanced dataset

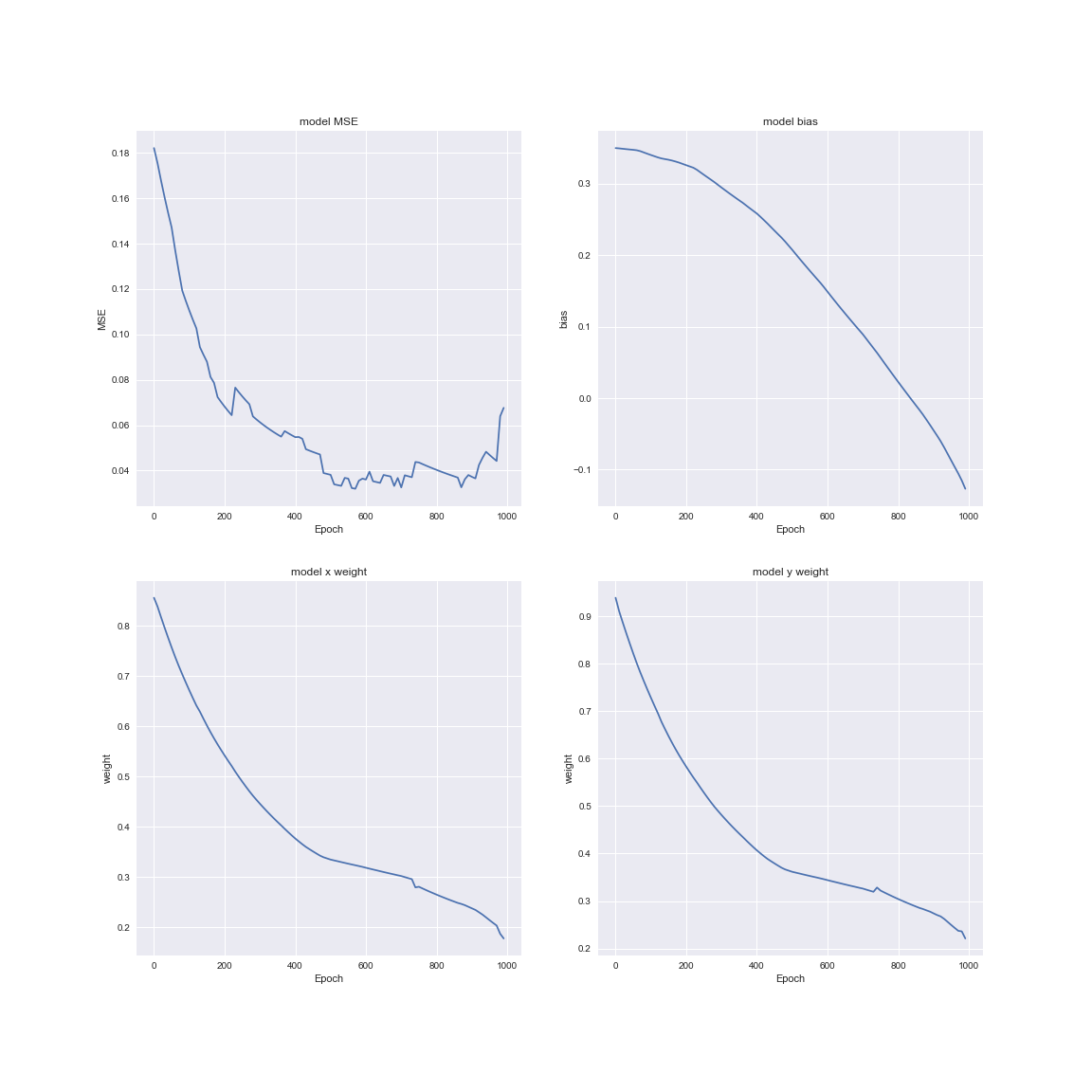
Chart, treemap chart

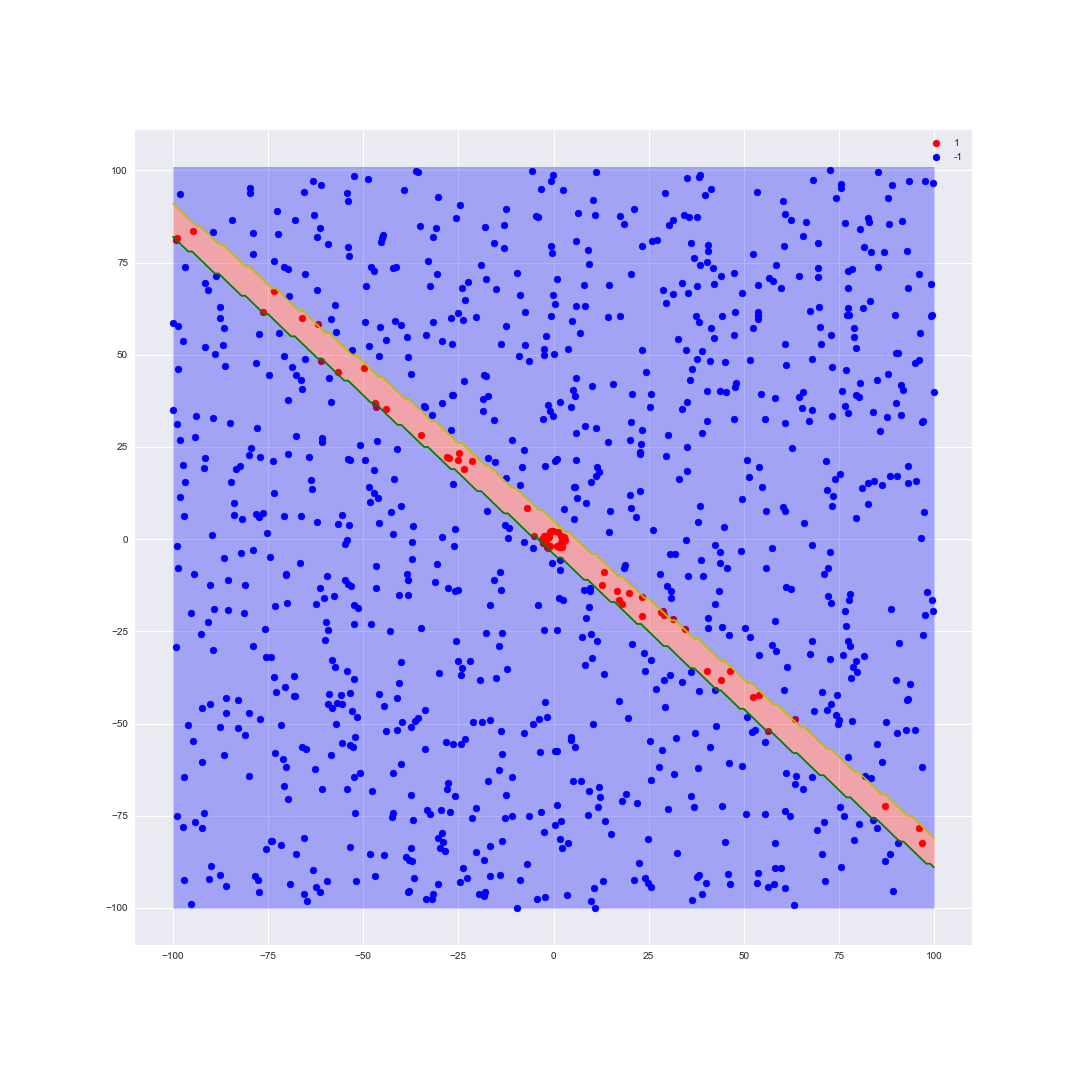
Description automatically generated

Chart, line chart

Description automatically generated

here are relevant graphs and other forms of data we gathered from the program:





From this example we can clearly see that the line captures the borders of the circle we should classify as 1