Perceptron

张栋玮 19373703 Open in GITHUB

Abstract

In this experiment, I first design and compile the codes of perceptron learning toward the linear classification of two classes. Main parts of the codes including datasets generation, visualization and Class of $Perceptron$ algorithm. I analysed the effect of the initialization of $weight$,$iterations$ and $learning\_rate$, and create new datasets to study the performance of the algorithm. I find $Perceptron$ useful in classification of linearly separable datasets with two classes. However, $Perceptron$ is just one of the bases of machine learning, and it is constrained strictly by the datasets. So there are more algorithms have been proposed which need further study. It's really fun to see the plots of random datasets!

1. Algorithm

1.1 Datasets Generation Function

First, generate one dataset with N points using $random.uniform()$ function. Then use random w and b to create a line to separate the dataset into two categories using $sign()$ function. By doing so, we get two linearly separable datasets.

I set the default random seed as 2, which is carefully chosen and gives a relatively good separation. The parameter $seed$ can be changed to get different datasets. Since w and b are randomly chosen, the two classes may have unequal number of items, which makes choosing a proper $seed$ important.

Function $y = sgn(wx + b)$ ensures that points above the line is classified as 1, while points below the line is classified as -1.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
def generate_data(N, seed=2):
    np.random.seed(seed)
    x = np.random.uniform(-1, 1, [N, 2])
    w = np.random.uniform(-1, 1, 2)
    b = np.random.uniform(-0.5, 0.5)
```

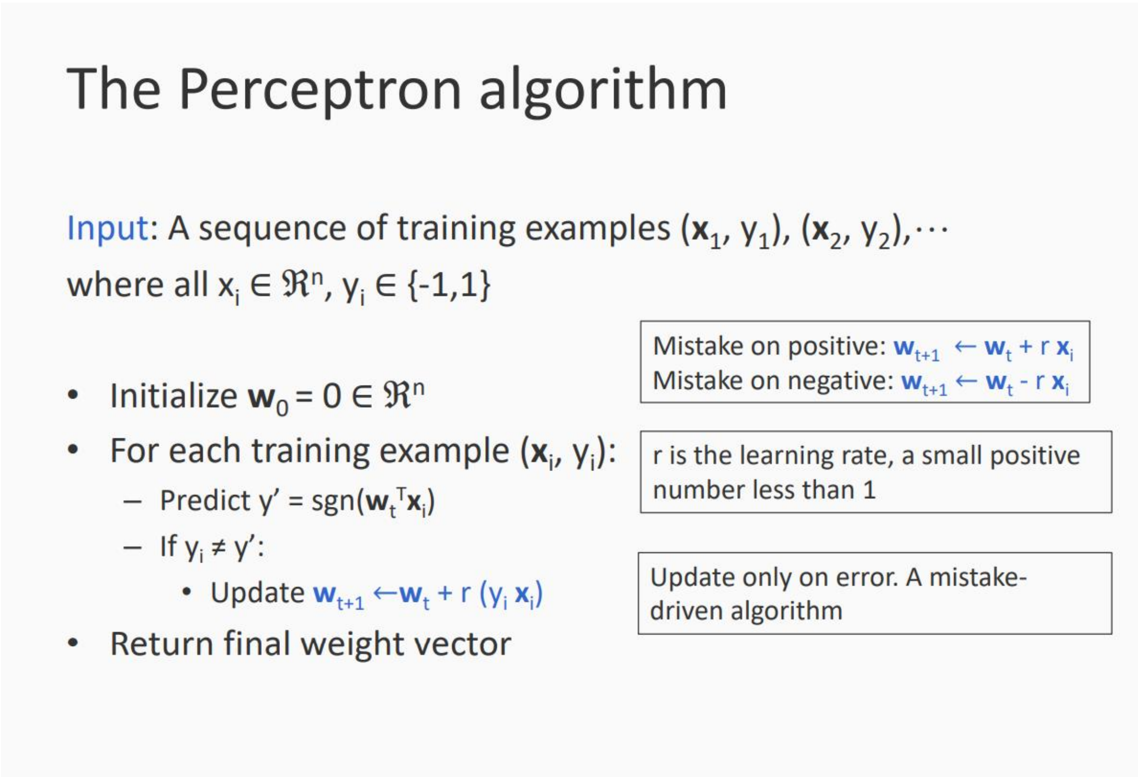
```
    y = np.sign(np.inner(w, x)+b)
    return x,y,w,b
```

1.2. Data visualization

In [3]:
```
def plot_data(x, y, w, b,title=None,save=None) :
    fig = plt.figure(figsize=(10,10))
    plt.title(title)
    plt.xlim(-1.1,1.1)
    plt.ylim(-1.1,1.1)

    # 1. Scatter Plot
    classes = {1: 'b', -1: 'r'}
    for i in range(len(x)):
        plt.plot(x[i][0], x[i][1], classes[y[i]]+'.')

    # 2. Separation Line
    lx = np.linspace(-1,1)
    a = -w[0]/w[1]
    plt.plot(lx, a*lx-(b/w[1]))

    # 3. Save it if need
    if(save):
        plt.savefig("./pic/"+save+".jpg")

    plt.show()
```

1.3. Perceptron

The perceptron algorithm is a mistake-driven algorithm that works as follows (b is seen as a part of w): Perceptron



In [4]:

```
class Perceptron :

    def __init__(self, max_iter=1000, lr=0.1) :

        self.w = np.zeros(3)
        self.max_iter = max_iter
        self.lr = lr
        self.records = []
        self._converge = False
        self._iteration = 0
        self.x=[]
        self.y=[]
        self.x1=[]

    def fit(self, x, y) :

        # Data preprocess
        self.x = x
        self.y = y
        self.x1 = np.column_stack((self.x,np.ones([len(self.x),1])))

        # Fit
        while (not self._converge and self._iteration <= self.max_iter ) :
            self._converge = True
            for i in range(len(self.x)) :
                # Update W if there is one misclassification.

                if self.y[i] * self.y_pre(self.w,self.x1[i]) <= 0 :
                    self.w = self.w + self.lr * self.y[i] * self.x1[i]
                    self._converge = False
            self._iteration += 1

            # Record W every 10 iterations
            if(self._iteration%10==0):
                self.records.append(self.w)

        # Check convergence
        if (not self._converge):
            print(f"Not converged!")
        else:
            print(f"Converged in {self._iteration} iterations!")

        print(f"Current w={self.w}")
        # Return final weight vector
        return self.w

    # Change inital weight
    def init_w(self,w):
        self.w = w


    def y_pre(self,x,w):
        y_pre = np.dot(x,w)
        return y_pre

    def record(self):
        return self.records
```

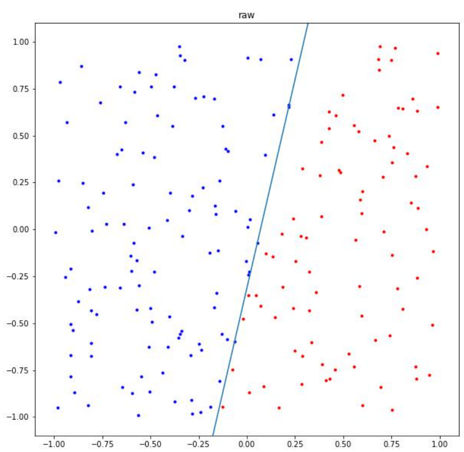
2. Test

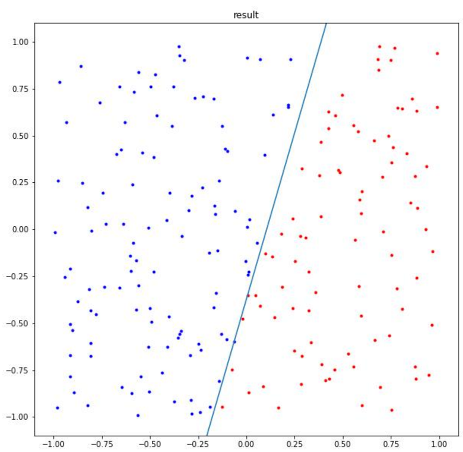
You can change $seed$ here and uncomment the $plot\_data()$ function to plot the result. Note here b is integrated in w as $w[2]$.

In [5]:
```
seed = 2
num_of_data=200
lr = 0.1
max_iter = 100

x,y,w,b = generate_data(num_of_data,seed) # here w,b is the raw separation line para
p = Perceptron(max_iter,lr)
weight = p.fit(x,y)
#plot_data(x,y,weight[:2],weight[2])
#plot_data(x,y,w,b,title="raw",save="raw")
#plot_data(x,y,weight[:2],weight[2],title="result",save="result")
```

```
Converged in 20 iterations!
Current w=[-0.94910184  0.26775336  0.1        ]
```

Here are the raw and result figures. The classification is done great, though the separation line has changed a little after the fit process.





3. Analysis of Parameters

The initial weight is zero which is set in the $Perceptron.\_\_init\_\_()$. We can change w using $init\_w()$.

In [6]:
```
# 1. Raw
lr = 0.1
max_iter = 100
p = Perceptron(max_iter,lr)
weight = p.fit(x,y)
```

```
Converged in 20 iterations!
Current w=[-0.94910184  0.26775336  0.1        ]
```

In [7]:
```
# 2. Change learning rate
lr = 1
max_iter = 100
p = Perceptron(max_iter,lr)
weight = p.fit(x,y)
```

```
Converged in 20 iterations!
Current w=[-9.49101842  2.67753357  1.         ]
```

In [8]:
```
# 3. Add init_w()
lr = 0.1
max_iter = 100
w=[100,100,100]   # Randomly choose
p = Perceptron(max_iter,lr)
p.init_w(w)
weight = p.fit(x,y)
```

```
Not converged!
Current w=[-42.0868654   13.96353553   5.5        ]
```

In [9]:
```
# 4. Change learning rate again (corresponding to 3)
lr = 1
max_iter = 100
w=[100,100,100]   # Randomly choose
p = Perceptron(max_iter,lr)
p.init_w(w)
weight = p.fit(x,y)
```

```
Converged in 24 iterations!
Current w=[-43.84290692  11.25932544   4.         ]
```

In [10]:
```
# 5. Add iterations (corresponding to 3)
lr = 0.1
max_iter = 1000
w=[100,100,100]   # Randomly choose
p = Perceptron(max_iter,lr)
p.init_w(w)
weight = p.fit(x,y)
```

```
Converged in 183 iterations!
Current w=[-42.62197525  12.46761728   4.7        ]
```

By comparing the above experiments, I get conclusions as follows:

- When `weight` is initialized as zero, it always needs same number of iterations to convergence whatever the `learning rate` is. However the `weight` result will change at a same proportion as `learning rate'.
- `weight` is changing proportionally because the parameters of linear function can be mutiplied by any number while stay the same line.
- `Learning rate` can accelerate the iteration if `weight` is large at beginning.

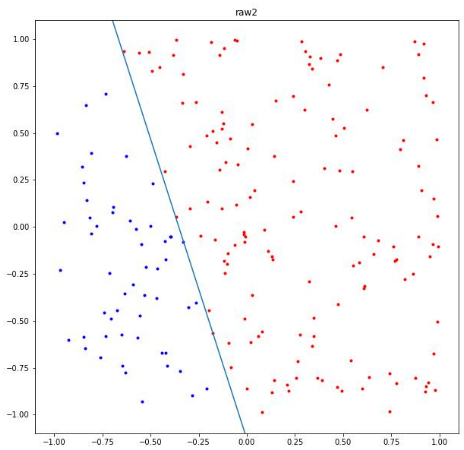
4. Test on New Datasets

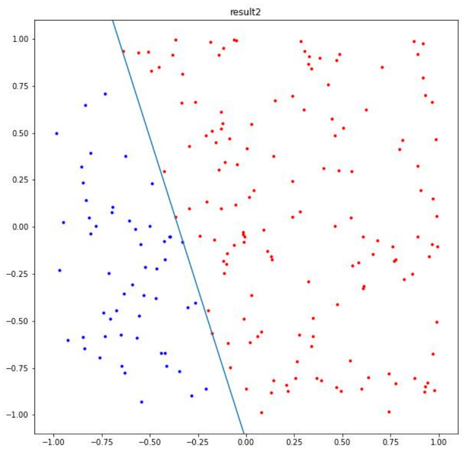
Uncomment the $plot\_data()$ and play with it !

In [11]:
```
seed = 14
num_of_data=200
lr = 0.1
max_iter = 1000

x,y,w,b = generate_data(num_of_data,seed) # here w,b is the raw separation line para
p = Perceptron(max_iter,lr)
weight = p.fit(x,y)


#plot_data(x,y,w,b)
#plot_data(x,y,weight[:2],weight[2])
```

```
Converged in 193 iterations!
Current w=[-1.70280639 -0.52832915 -0.6        ]
```

Here are comparisons with $seed = 14$. The two are almost the same.





Reference

[1]. Statistical Pattern Recognition Lab, SASEE

[2]. 机器学习笔记——感知机（Perceptron）

[3]. 最简单的神经网络——感知器算法