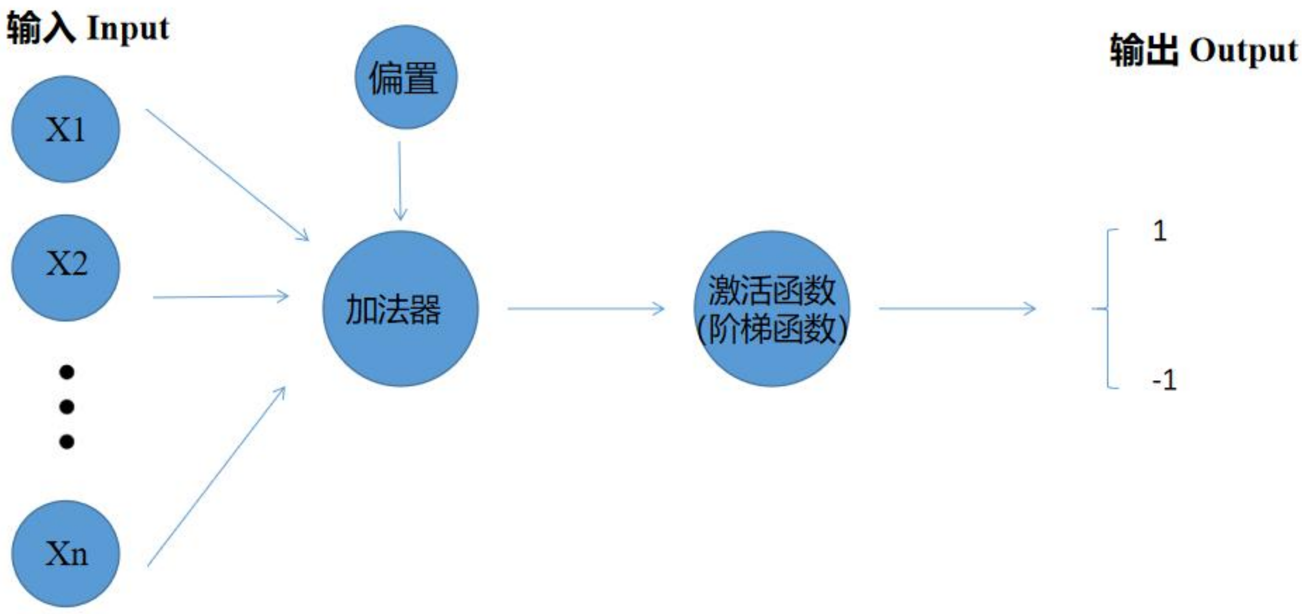
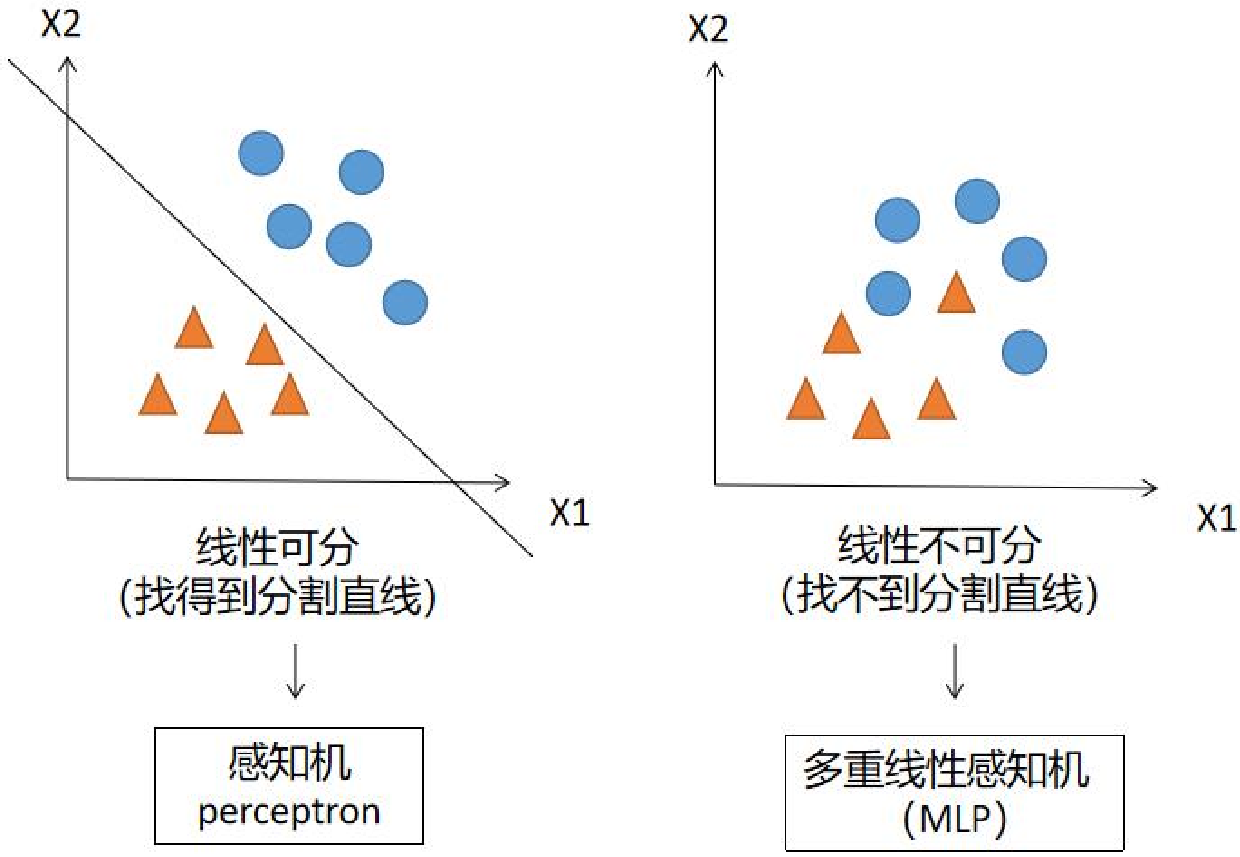
Table of Contents

1 什么是感知机？

- 简单定义：二分类的线性分类器



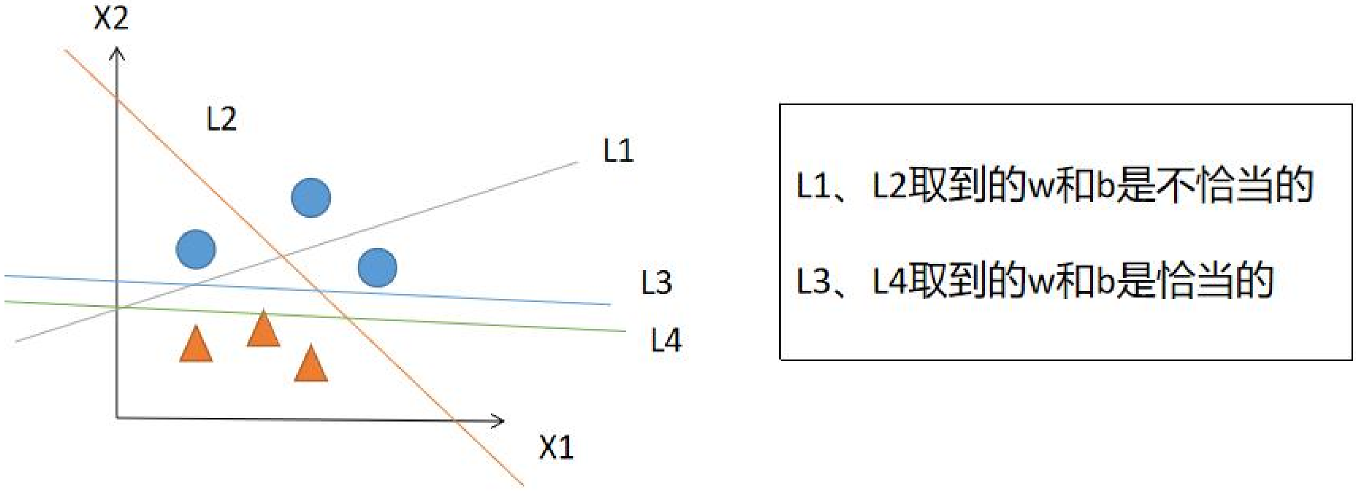
2 线性可分性



- 理论上讲，任何复杂的非线性分类问题均可以用 MLP （给感知机加层）去近似分类

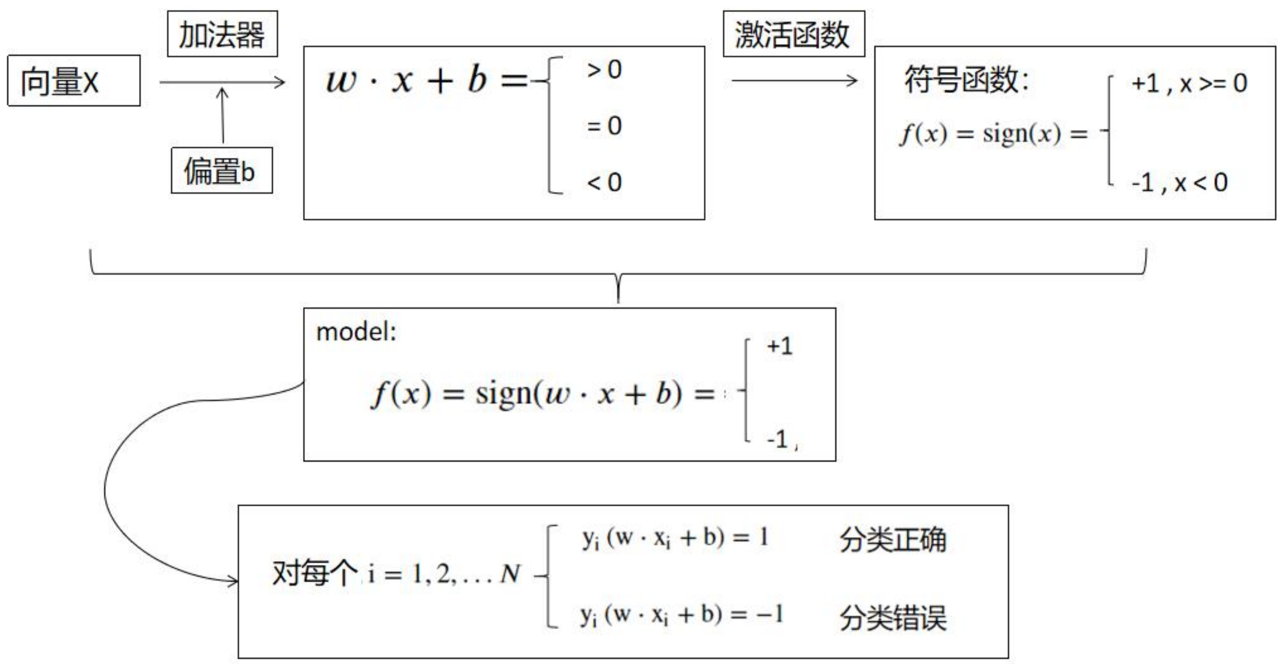
3 待解决的问题：寻找参数 w 和 b

- 已知 $Data=\{(x_i,y_i),i=1,\dots N\}$,$x_i\in\mathbb{R}^d$ 是向量，$y_i\in\{+1,-1\}$ 为标注，若 $Data$ 是线性可分的，则感知机 perceptron 要寻找 w 和 b 使得
$$w\cdot x+b=0$$
成为分割的超平面



- 显然，感知机存在且不唯一！！！

4 模型 model



5 策略 strategy

- ## 5.1 损失函数 --- > 误分类点个数 决定

  - ### 5.1.1 定性描述：

$$M=\{i\mid -y_i(w\cdot x_i+b)>0\}$$

其中 M 为误分类点集合。**上式说明预测试与实际值相反，分类错误**

- **5.1.2 定量描述:**

$$f(x)=\begin{cases} d=\frac{1}{\|w\|}|\mathrm{w}\cdot\mathrm{x_i}+\mathrm{b}|=\frac{1}{\|w\|}y_i(\mathrm{w}\cdot\mathrm{x_i}+\mathrm{b}), i\in M \\ \\ \\ 0 \end{cases}$$

其中 M 为误分类点集合。

- **5.1.3 感知机损失函数 & 经验风险函数:**

$$L(w,b)=-\sum_{x_i\in M}y_i(\mathrm{w}\cdot\mathrm{x_i}+\mathrm{b})$$

其中 M 为误分类点集合。如果没有**误分类点**，损失函数值为0.

6 算法 algorithm

- ## 6.1 随机梯度下降算法：——误分类点驱动

  - **6.1.1 基本思路：用任意点处的值减去步长（学习率）乘以在该点的导数，来最小化函数值。**
    - 更新函数 $w\cdot x+b$ 中的 w 和 b :

$$w^{k+1}=w^k-\eta\frac{\partial Loss}{\partial w}|_{w=w^k}=w^k-\eta\sum_{i\in M}y_ix_i$$
$$b^{k+1}=b^k-\eta\frac{\partial Loss}{\partial b}|_{b=b^k}=b^k-\eta\sum_{i\in M}y_i$$

其中 M 为误分类点集合。

- ## 6.2 感知机学习算法——原始状态：

==

输入: 训练数据集 $\mathrm{T}=\left\{\left(\mathrm{x_1},\mathrm{y_1}\right),\left(\mathrm{x_2},\mathrm{y_2}\right),\ldots,\left(\mathrm{x_N},\mathrm{y_N}\right)\right\}$, 其中
$\mathrm{x_i}\in\mathrm{R^n},\quad \mathrm{y_i}\in\mathcal{Y}=\{-1,+1\},\quad \mathrm{i}=1,2,\ldots N;\quad$ 学习率$\eta(0<\eta<=1)$

输出:感知机模型 $f(x)=\mathrm{sign}(w\cdot x+b)$ 的 w,b

(1) 选取初值 w_0,b_0

(2) 在训练集中选取数据 $\left(\mathrm{x_i},\mathrm{y_i}\right)$

(3) 如果 $\mathrm{y_i}\left(\mathrm{w}\cdot\mathrm{x_i}+\mathrm{b}\right)\leq 0$*，则*

$$w\leftarrow w+\eta y_ix_i$$
$$b\leftarrow b+\eta y_i$$

(4) 转至 (2) , 直至训练集中没有误分类点。

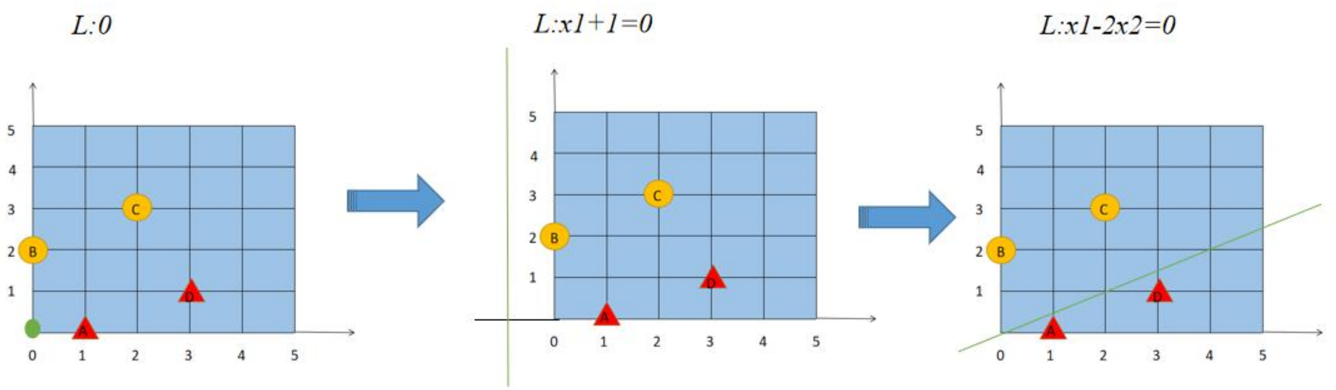
==

- 这种学习算法直观上有如下解释: 当一个实例点被误分类, 即位于分离超平面的错误 一侧时, 则调整 w,b 的值, 使分离超平面向该误分类点的一侧移动, 以减少该误分类点与超平面间的距离, 直至超平面越过该

误分类点使其被正确分类。

- ## 6.3 算法的注释：
  - *(1) 原始算法一定**收敛**（对于**线性可分集合**而言），即一定能找到一组 w, b*
  - *(2) 若**不是线性可分集合**，在迭代时一定会出现**振荡**现象，不收敛*
  - *(3) 感知机**解不唯一**，由 初始点 w_0, b_0 ，学习率 η,误 分类点不唯一时的改进选择 **三个因素**决定*

- ## 6.4 手算经典感知机
  - *假设二维平面坐标系上有四个点：A(1,0),B(0,2),C(2,3),D(3,1)，其中B和C点是正类，A 和D点是负类。现要求用感知机的原始算法，手算上述样本点的分割线。相应的初始值和参数自行设定，并作图表示迭代过程。*
    - 解：
      - (1) 令 $w_0 = 0, b_0 = 0, \eta = 1$，并且设正类为 $y = 1$，负类为 $y = -1$
      - (2) 对A点 $x_1 = (1, 0)^T$ 而言，$w \cdot x + b = 0$, 所以 $y(w \cdot x + b) = 0$ 其未被正确分类 此时选择A点为误分点进行更新参数 w, b 。
        $$w_1 = w_0 + \eta y_a x_a = 0 + 1 \times 1 \times (1, 0)^T = (1, 0)^T$$
        $$b_1 = b_0 + \eta y_a = 0 + 1 \times 1 = 1$$
        得到线性模型
        $$w_1 \cdot x + b_1 = x^{(1)} + 1$$
      - (3) 对B点 $x_2 = (0, 2)^T$ 而言，$y(w \cdot x + b) < 0$ 其未被正确分类 此时选择B点为误分点进行更新参数
        $$w_2 = w_1 + \eta y_b x_b = (1, 0)^T + 1 \times -1 \times (0, 2)^T = (1, -2)^T$$
        $$b_2 = b_0 + \eta y_b = 1 + 1 \times -1 = 0$$
        得到线性模型
        $$w_2 \cdot x + b_2 = x^{(1)} - 2x^{(2)}$$
      - (4) 此时所有点都被正确分类，损失函数达到最小，算法停止。 因此分离超平面为 $x^{(1)} - 2x^{(2)} = 0$ ，感知机模型为 $f(x) = \text{sign}(x^{(1)} - 2x^{(2)})$



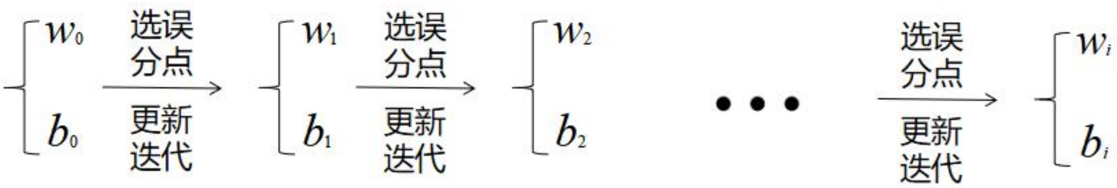
- ## 6.5 感知机 原始状态 python代码实现

In [1]:

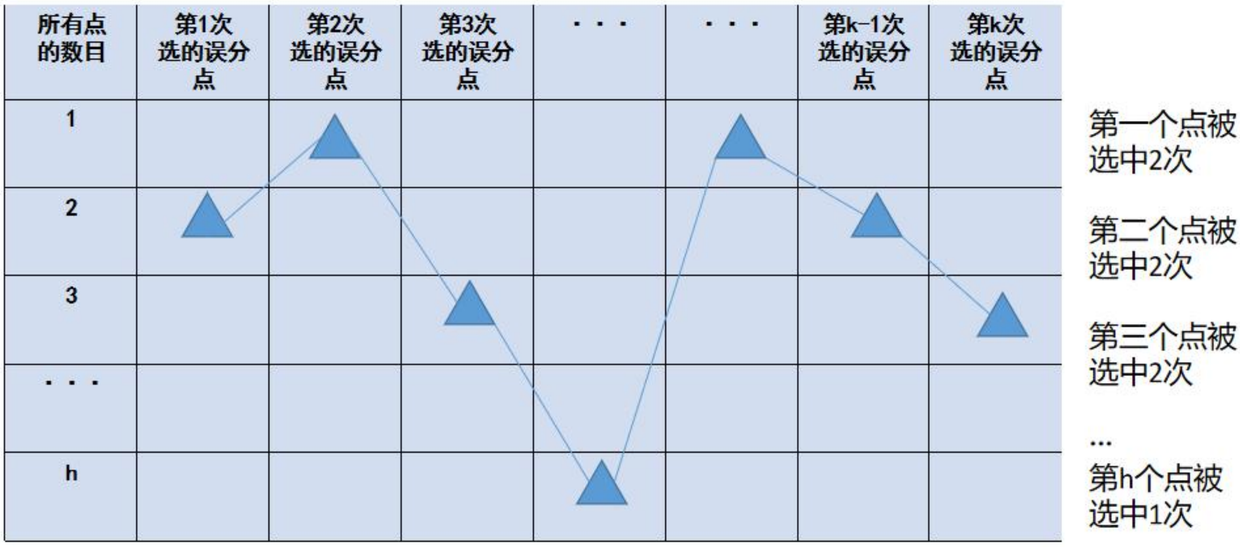
```
import numpy as np   #导入相关包
import matplotlib.pyplot as plt   #导入相关包
import pandas as pd     #导入相关包

##
x1=np.random.uniform(-5,5,10)   #随机生成10个-5到5之间的整数赋给x1
x2=np.random.uniform(-5,5,10)   #随机生成10个-5到5之间的整数赋给x2
x=pd.DataFrame({'x1':x1,'x2':x2})  #用x1和x2创建一个数据框
w=[1,-1];b=0.5   #将w的初始值赋为向量[1, -1], b的初始值赋为0.5
y=np.sign([np.dot(w,x.iloc[i,:])+b for i in x.index])  #对数据框内所有的x的列和w做点积再加上

##
def perceptron(x,y,w,b):      #定义感知机函数
    eta=0.1                   #学习率（步长）eta值取0.1
    d=len(x.index)            #d为引索为x的向量的长度
    n=0                       #n为迭代次数，将n的值赋为0
    while d!=0:               #当d不等于0时进入循环
        M=[i for i in x.index if (np.dot(w,x.iloc[i,:])+b)*y[i]<0]
        d=len(M)              #d为M的长度
        if d>0:               #如果d>0
            w=w+eta*y[M[0]]*x.iloc[M[0],:]      #w 更新为 w+eta*yi*xi
            b=b+eta*y[M[0]]                     #b 更新为 b+eta*yi
            n=n+1                               #n 更新为 n+1
    return w,b,n              #返回参数w,b以及迭代次数n
```

7 感知机算法的对偶形式 ——> 提升算法效率，在误分点判断、寻找时，通过内外双重循环来实现

- ## 7.1 算法提升的突破口？某个点可能会被选为误分点多次！！



在进行更新 w 和 b 时，对误分点的选择可能会出现重复现象，即某误分点可能会被选中多次



- ## 7.2 对偶算法效率提升的方法？内、外循环！！
  - ### 7.2.1 内循环：

$$w=\sum_{i=1}^{N} a_i y_i x_i$$

$$b=\sum_{i=1}^{N} a_i y_i$$

其中 $a_i=n_i\eta$， n_i 是点 (x_i, y_i) 被误分类的次数

  - ### 7.2.2 外循环：

$$\left(\sum_{j=1}^{N}(a_j y_j x_j)\cdot x_i+b\right)\leq 0, i=1,2,\ldots,N$$

- ## 7.3 感知机对偶算法：

==

输入: 训练数据集 $\mathrm{T}=\left\{\left(\mathrm{x}_1, \mathrm{y}_1\right),\left(\mathrm{x}_2, \mathrm{y}_2\right), \ldots,\left(\mathrm{x}_{\mathrm{N}}, \mathrm{y}_{\mathrm{N}}\right)\right\}$, 其中
$\mathrm{x}_{\mathrm{i}} \in \mathrm{R}^{\mathrm{n}}, \quad \mathrm{y}_{\mathrm{i}} \in \mathcal{Y}=\{-1,+1\}, \quad \mathrm{i}=1,2, \ldots N ; \quad$ 学习率$\eta(0<\eta<=1)$

输出:感知机模型 $f(x)=\operatorname{sign}(\sum_{j=1}^{N}(a_j y_j x_j)\cdot x+b)$ 的 a, b 其中 $a=(a_1, a_2, \ldots, a_N)$

(1) $a \leftarrow 0, b \leftarrow 0$

(2) 在训练集中选取数据 $\left(\mathrm{x}_{\mathrm{i}}, \mathrm{y}_{\mathrm{i}}\right)$

(3) 如果 $\mathrm{y}_{\mathrm{i}}\left(\sum_{j=1}^{N}(a_j y_j x_j)\cdot x_i+b\right)\leq 0$*，则*

$$a_i \leftarrow a_i+\eta$$
$$b \leftarrow b+\eta y_i$$

(4) 转至 (2)，直至训练集中没有误分类点。

==

  - *为了方便，可以预先将训练集中实例间的内积计算出来并以矩阵的形式储存，这个矩阵就是所谓的 Gram 矩阵：*

$$G = [x_i \cdot x_j]_{N \times N}$$

- ## 7.4 感知机 对偶状态 python代码实现

In [2]:

```
import numpy as np      #导入相关包

sample_n=20           #样本个数设置为20
x1=np.random.uniform(-5,5,sample_n)  #随机生成sample_n个-5到5之间的整数赋给x1
x2=np.random.uniform(-5,5,sample_n)  #随机生成sample_n个-5到5之间的整数赋给x2

x=np.array(np.transpose(np.mat((x1,x2),dtype=float))) #令x1,x2分别为行然后形成矩阵，矩阵元素关
w=[1,-1];b=0.5          #将w的初始值赋为向量[1，-1]，b的初始值赋为0.5
y=np.sign([np.dot(w,x[i,:])+b for i in range(x.shape[0])])     #对数据框内所有的x和w做点积再加

trainingSet=x                  #将x的值赋给训练集trainingset
# 计算Gram矩阵
m = np.shape(trainingSet)[0]      #m值为trainingset的行数
GramMatrix = [None] * m          #将行数与空矩阵相乘
for i in range(m):                #取遍每一行
    GramMatrix[i] = [0] * m       #将每一行元素用0代替
    for j in range(m):           #取遍每一行
        GramMatrix[i][j] = int(np.dot(trainingSet[i], trainingSet[j].T))#每个元素等于training

# 参数初始化
#alpha = [0] * m
alpha=np.random.normal(0,0.5,sample_n) #随机生成sample_n个服从均值为0，标准差为0.5的正态分布数
b = 0     #b的初始值为0
eta = 1   #学习率（步长）的值为1

n=0       #迭代次数初始值为0
# 开始训练
isFound = False                #最优解未找到
while not isFound:             #当最优解未找到时进入循环
    for i in range(m):            #取遍每一行
        temp = 0               #temp初值为0
        for j in range(m):       #取遍每一行
            temp += alpha[j] * y[j] * GramMatrix[j][i]   #temp更新为temp+alpha[j]* y[j] * Gra

        if y[i] * (temp + b) <= 0: #若y[i] * (temp + b) <= 0
            alpha[i] += 1 * eta  #alpha 更新为 alpha+eta*1
            b += y[i] * eta      #b 更新为 b+eta*yi
            n +=1                #n 更新为 n+1
            break                #跳出循环
        elif i == m - 1:         #否则i==m-1
            isFound = True       #此事最优解找到

w = [0, 0]                      #w初始值为向量[0,0]
for i in range(m):              #整体做m次循环
    w += alpha[i] * trainingSet[i] * y[i]  #w 更新为 w+ai*trainingset i*yi
print(w, b, n)  #打印出参数w,b和迭代次数n
```

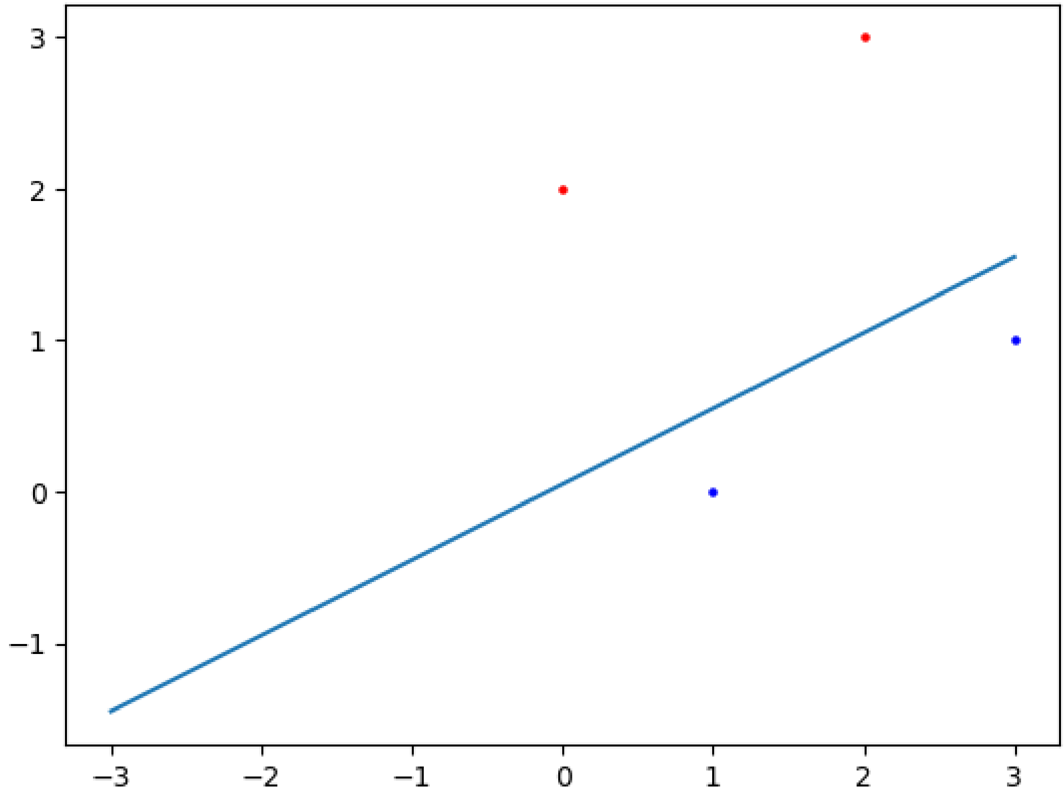
```
[ 3.74989265 -3.21877674] 0 0
```

8 原始算法代码实现

In [3]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

x1 = [1, 0, 2, 3]
x2 = [0, 2, 3, 1]
x = pd.DataFrame({'x1':x1,'x2':x2})
y = [1,-1,-1,1]

def perceptron(x,y,w,b):
    eta = 1
    d = len(x.index)
    n = 0  #迭代次数
    while d!= 0:  #d表示误分点的个数
        M = [i for i in x.index if (np.dot(w,x.iloc[i,:])+b)*y[i]<0]  #误分点判别
        d = len(M)
        if d > 0:
            w = w + eta*y[M[0]]*x.iloc[M[0],:]
            b = b + eta*y[M[0]]
            n = n+1
    return w,b,n

w0 = [0,0];b0=0.1
result = perceptron(x,y,w0,b0)
ww = result[0]
bb = result[1]
n = result[2]
xx = np.linspace(-3,3,3)
yy =-(ww[0]*xx+bb)/ww[1]


plt.scatter(x1[0],x2[0],s=20,c='b',marker='.')
plt.scatter(x1[3],x2[3],s=20,c='b',marker='.')
plt.scatter(x1[1],x2[1],s=20,c='r',marker='.')
plt.scatter(x1[2],x2[2],s=20,c='r',marker='.')
plt.plot(xx,yy)
print("w为")
print(ww)
print("           ")
print("b为")
print(bb)
print("           ")
print("迭代次数为：")
print(n)
```

```
w为
x1    1
x2   -2
dtype: int64

b为
0.09999999999999998

迭代次数为：
2
```



9 对偶形式代码实现

In [4]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

sample_n=4
x1 = [1, 0, 2, 3]
x2 = [0, 2, 3, 1]
x=np.array(np.transpose(np.mat((x1,x2),dtype=float)))
w = [0,0];b=0
y = [1,-1,-1,1]

trainingSet=x
# 计算Gram矩阵
m = np.shape(trainingSet)[0]
GramMatrix = [None] * m
for i in range(m):
    GramMatrix[i] = [0] * m
    for j in range(m):
        GramMatrix[i][j] = int(np.dot(trainingSet[i], trainingSet[j].T))

# 参数初始化
#alpha = [0] * m
alpha=np.random.normal(0,0.5,sample_n)
b = 0.1
eta = 1
n=0
# 开始训练
isFound = False
while not isFound:
    for i in range(m):
        temp = 0
        for j in range(m):
            temp += alpha[j] * y[j] * GramMatrix[j][i]

        if y[i] * (temp + b) <= 0:
            alpha[i] += 1 * eta
            b += y[i] * eta
            n +=1
            break
        elif i == m - 1:
            isFound = True

w = [0, 0]
for i in range(m):
    w += alpha[i] * trainingSet[i] * y[i]
print("w为")
print(w)
print("           ")
print("b为")
print(b)
print("           ")
print("迭代次数为：")
print(n)

xx = np.linspace(-3,3,3)
yy = -(w[0]*xx+b)/w[1]

plt.scatter(x1[0],x2[0],s=20,c='b',marker='.')
plt.scatter(x1[3],x2[3],s=20,c='b',marker='.')
```

```python
plt.scatter(x1[1],x2[1],s=20,c='r',marker='.')
plt.scatter(x1[2],x2[2],s=20,c='r',marker='.')
plt.plot(xx,yy)
```

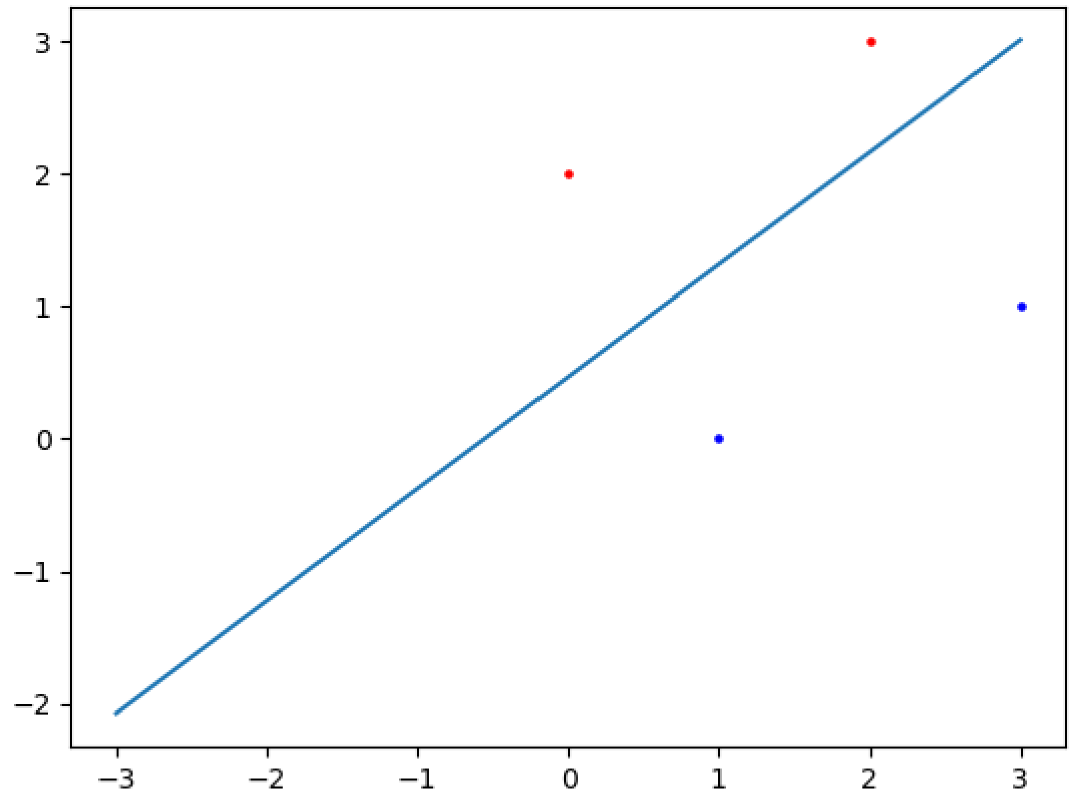
w为
[1.99150615 -2.3508184]

b为
1.1

迭代次数为:
3

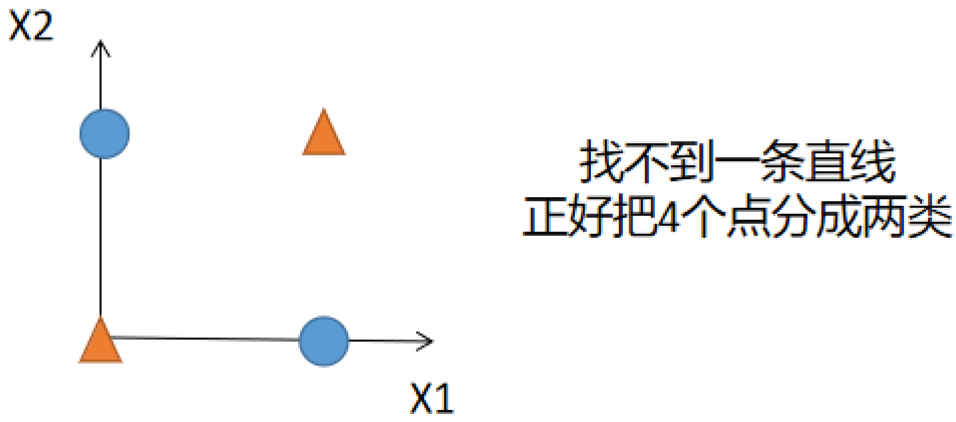
Out[4]:

[<matplotlib.lines.Line2D at 0x1fe5b6afeb0>]

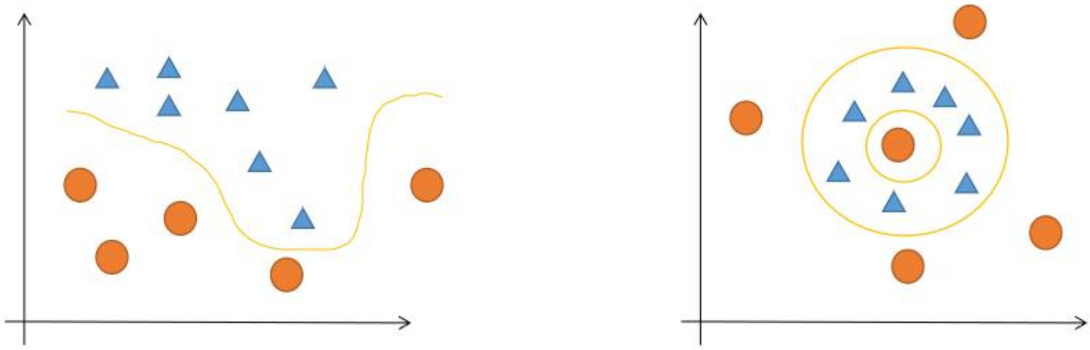


10 拓展问题：非线性分类器——MLP多层感知机

- **10.1 异或问题（XOR）——>两层感知机 解决**



- **10.2 曲线分割——>两层感知机也失效**



11 单层感知机对“异或”问题失效（迭代上限设置为1000）

In [5]:

```python
import numpy as np


M, input_size = 100, 2
x = np.random.uniform(-5, 5, [M, input_size])
w = [1.2, 0.8]
b1 = -3
b2 = 3
y1 = [np.dot(x[i], w)+b1 for i in range(x.shape[0])]
y11 = [1 if x > 0 else 0 for x in y1]
y2 = [np.dot(x[i], w)+b2 for i in range(x.shape[0])]
y22 = [1 if x > 0 else 0 for x in y2]
y3 = [1 if y11[i]+y22[i] == 1 else 0 for i in range(len(y2))]
X = np.array(x)
y = np.array(y3).reshape(-1,)

trainingSet = X

m = np.shape(trainingSet)[0]
GramMatrix = [None] * m
for i in range(m):
    GramMatrix[i] = [0] * m
    for j in range(m):
        GramMatrix[i][j] = int(np.dot(trainingSet[i], trainingSet[j].T))

Gram = np.array(GramMatrix)


def perceptron_daul(x, y, alpha, b):
    eta = 0.05
    n = 0
    isFound = False
    while (not isFound) & (n < 1000):#自行设置迭代次数上限，以免死循环。
        M = [i for i in range(len(y)) if y[i] *
              (np.dot(alpha*y, Gram[:, i])+b) <= 0]
        # print(M,n)
        if len(M) == 0:
            isFound = True
            # break
        else:
            alpha[M[0]] += eta
            b += y[M[0]]*eta
            n += 1
            # print(alpha,b)

    return alpha, b, n


alpha0 = np.random.normal(0, 0.5, 100)
b0 = 0
alpha, b, n = perceptron_daul(x, y, alpha0, b0)
w = [0, 0]
for i in range(m):
    w += alpha[i] * trainingSet[i] * y[i]
print("w为")
print(w)
print("           ")
print("b为")
print(b)
```

```
print("            ")
print("迭代次数为：")
print(n)
```

```
w为
[-5.37753938  3.61940535]

b为
0.0

迭代次数为：
1000
```

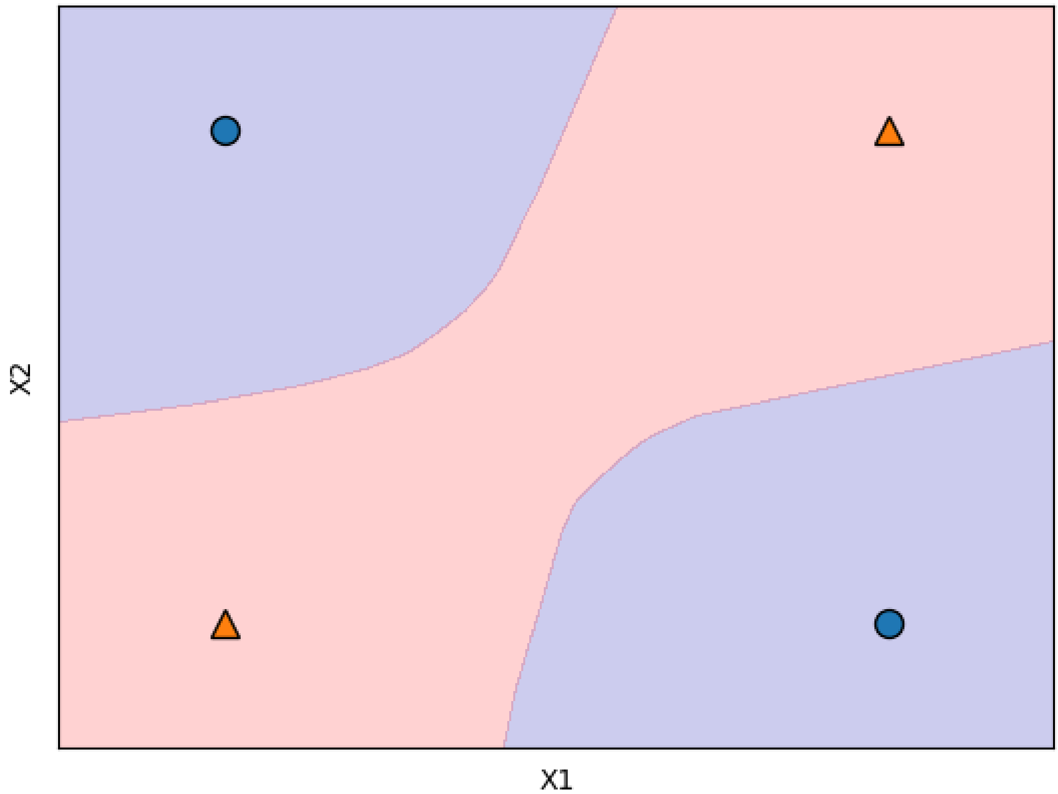
12 异或问题代码实现

In [7]:

```python
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import mglearn
import numpy as np

x1 = [1, 0, 0, 1]
x2 = [0, 1, 0, 1]
X=np.array(np.transpose(np.mat((x1,x2),dtype=float)))
y=[0,0,1,1]
mlp=MLPClassifier(solver='lbfgs',random_state=44).fit(X,y)
mglearn.plots.plot_2d_separator(mlp,X,fill=True,alpha=0.2)
mglearn.discrete_scatter(X[:,0],X[:,1],y)
plt.xlabel("X1")
plt.ylabel("X2")
```

Out[7]:

```
Text(0, 0.5, 'X2')
```



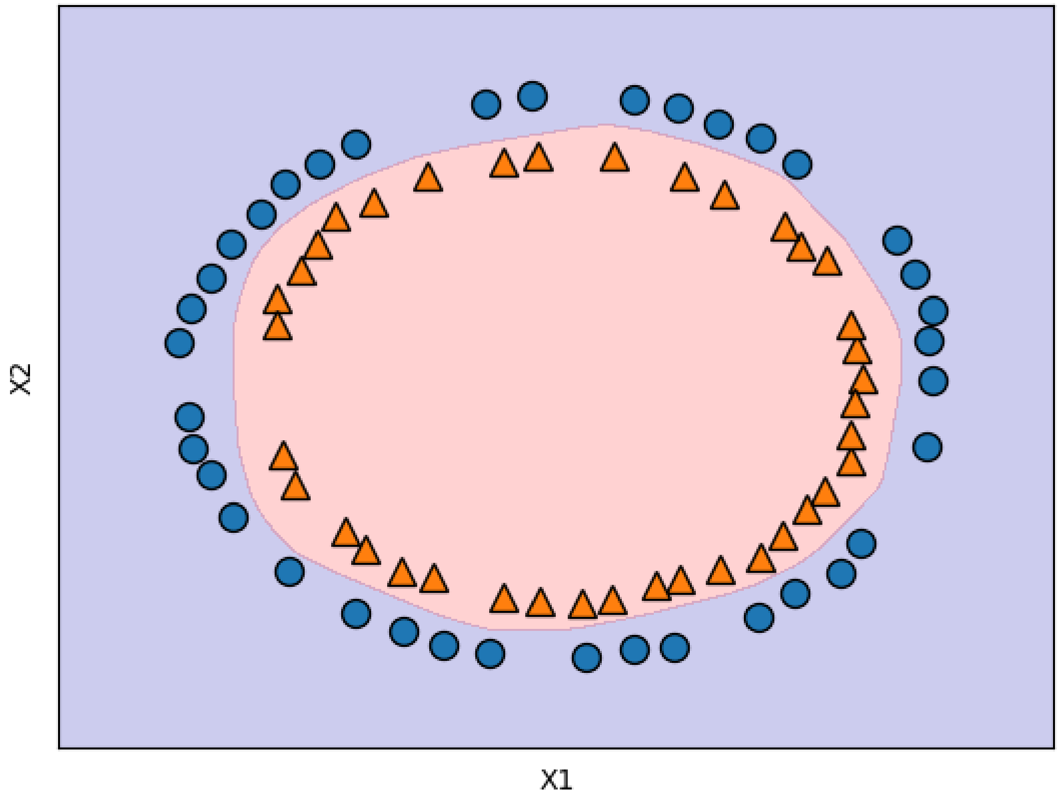
13 曲线分割代码实现

In [8]:

```
from sklearn.neural_network import MLPClassifier #多层感知机分割器
from sklearn.datasets import make_circles
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import mglearn

X,y=make_circles(n_samples=100,noise=0.01,random_state=42)#sklearn库里本身就有可以创造圆形数扌
X_train,X_test,y_train,y_test=train_test_split(X,y,stratify=y,random_state=42)#stratify 依据札
mlp=MLPClassifier(solver='lbfgs',random_state=0).fit(X_train,y_train)
mglearn.plots.plot_2d_separator(mlp,X_train,fill=True,alpha=0.2)#alpha是颜色参数
mglearn.discrete_scatter(X_train[:,0],X_train[:,1],y_train)
plt.xlabel("X1")
plt.ylabel("X2")
```

Out[8]:

Text(0, 0.5, 'X2')

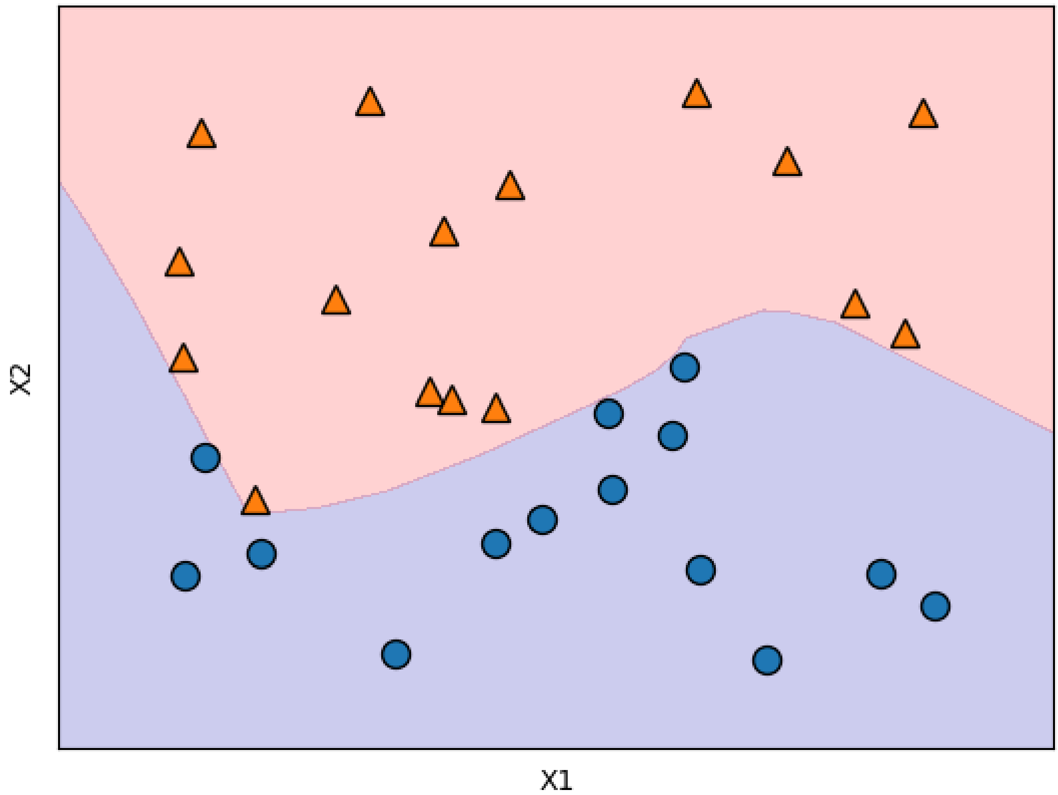


In [9]:

```
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import mglearn
import numpy as np

M, input_size=40,2
x=np.random.uniform(-1,1,[M, input_size])
x1=x[:,0]/(0.85*max(abs(x[:,0])))
yy=[1 if x[i,1]>0.75*np.sin(np.pi*x1[i]) else 0 for i in range(x.shape[0])]
X=np.array(x)
y=np.array(yy)
X_train,X_test,y_train,y_test=train_test_split(X,y,stratify=y,random_state=42)
mlp=MLPClassifier(solver='lbfgs',random_state=0).fit(X_train,y_train)
mglearn.plots.plot_2d_separator(mlp,X_train,fill=True,alpha=0.2)
mglearn.discrete_scatter(X_train[:,0],X_train[:,1],y_train)
plt.xlabel("X1")
plt.ylabel("X2")
```

Out[9]:

```
Text(0, 0.5, 'X2')
```

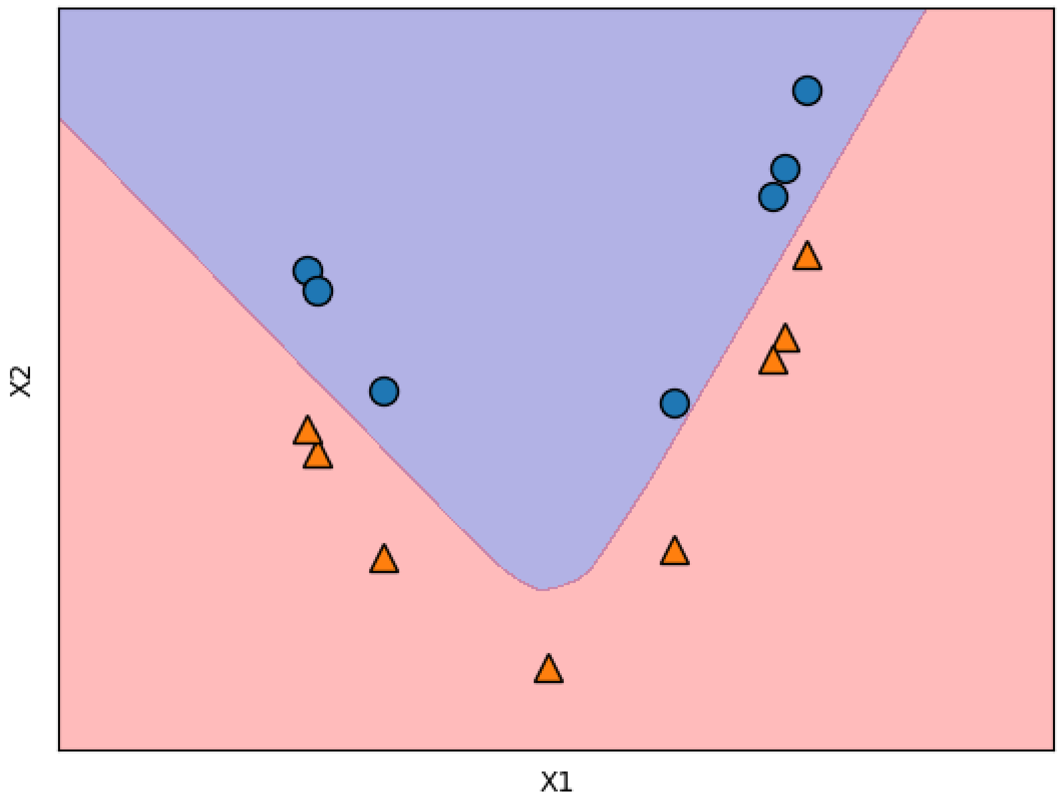


In [10]:

```
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import mglearn
import numpy as np
import pandas as pd


np.random.seed(12)
x0 = list(np.random.uniform(-5,5,10))
x1 = [i**2 + np.random.uniform(3,5) for i in x0]
x2 = [i**2 - np.random.uniform(3,5) for i in x0]
x11 = x0+x0
x22 = x1+x2
x = pd.DataFrame({"x1":x11,'x2':x22})
y1 = [-1]*len(x2) + [1]*len(x1)

X=np.array(x)
y=np.array(y1).reshape(-1,)

X_train,X_test,y_train,y_test=train_test_split(X,y,stratify=y,random_state=42)
mlp=MLPClassifier(solver='lbfgs',random_state=0).fit(X_train,y_train)
mglearn.plots.plot_2d_separator(mlp,X_train,fill=True,alpha=0.3)
mglearn.discrete_scatter(X_train[:,0],X_train[:,1],y_train)
plt.xlabel("X1")
plt.ylabel("X2")
```

Out[10]:

```
Text(0, 0.5, 'X2')
```



In [ ]:

```

```