Softmax

赵海涛

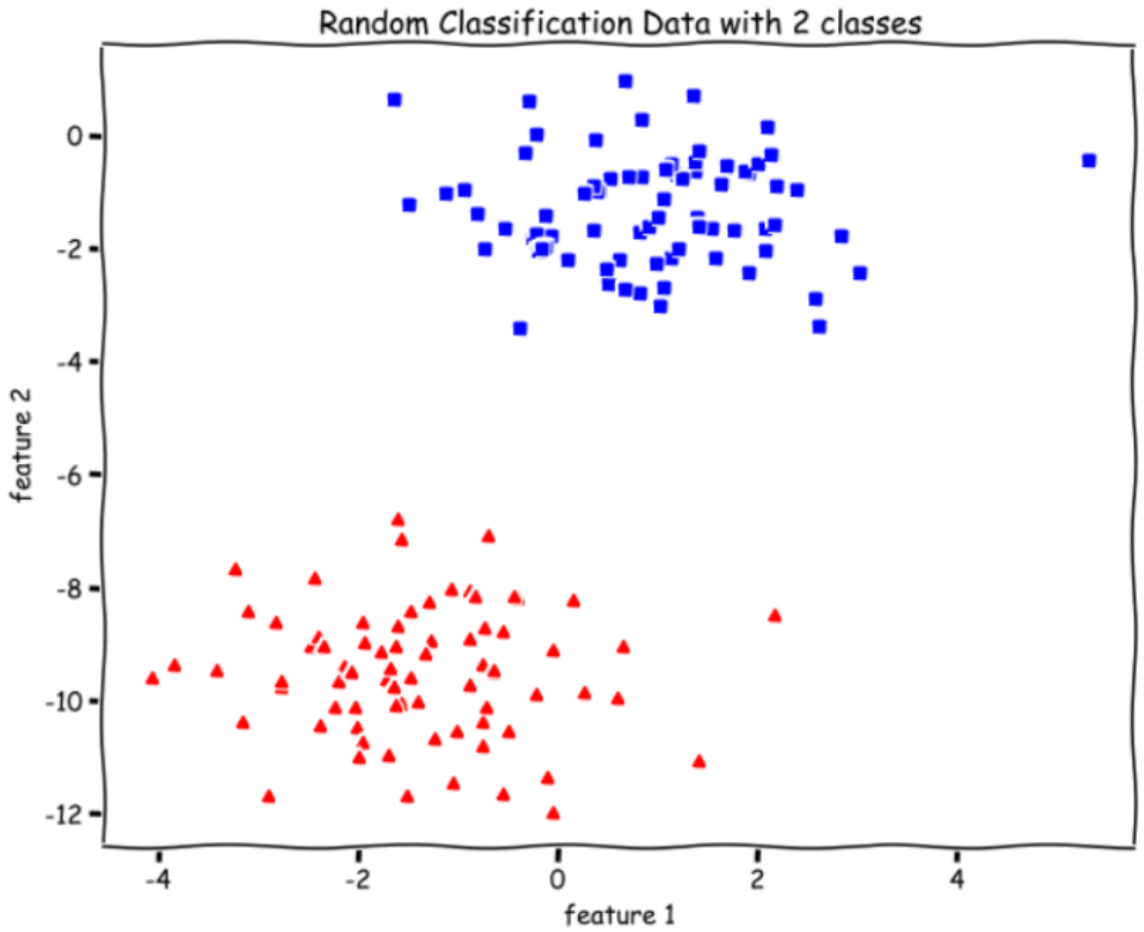
haitaozhao@ecust.edu.cn

大纲

- 二分类与多分类问题
- Softmax推导
- Softmax推导的向量形式
- Softmax例子

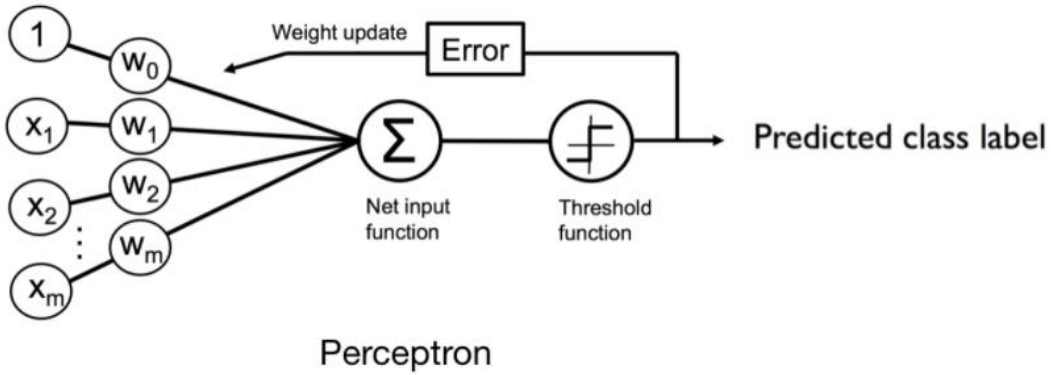
二分类问题

- 两类分类器：感知机，线性回归，逻辑斯蒂回归

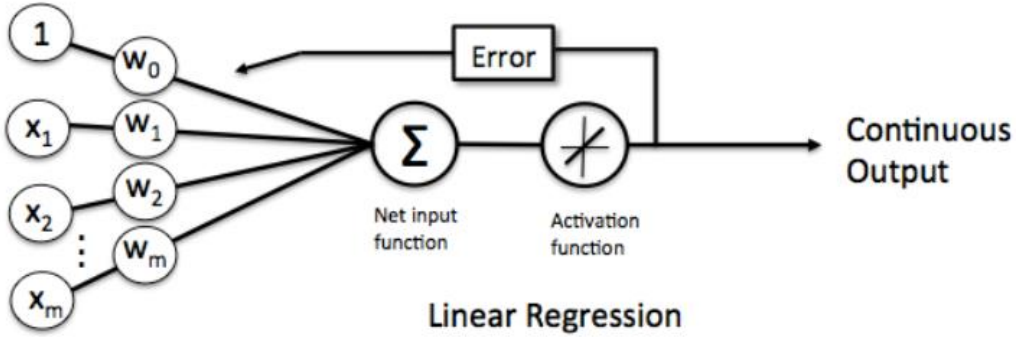


二分类

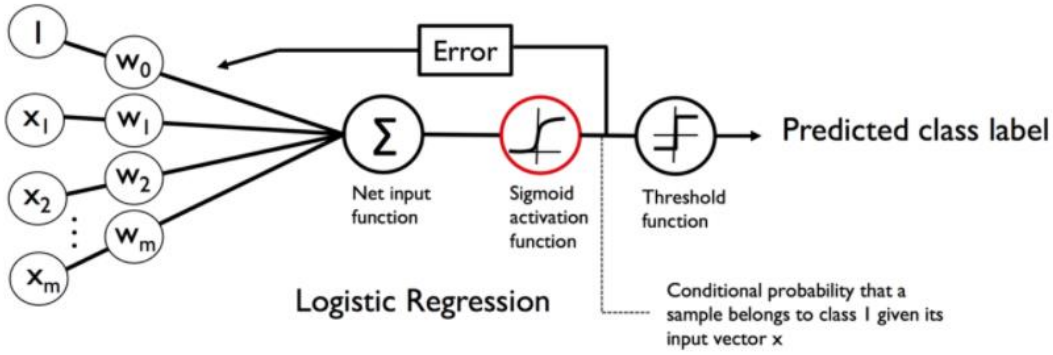
- 两类分类器：感知机，线性回归，逻辑斯蒂回归



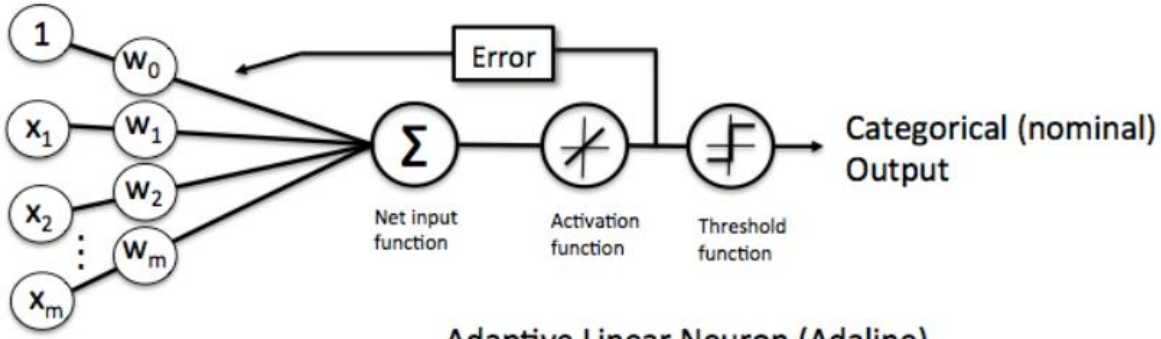
Perceptron



Linear Regression



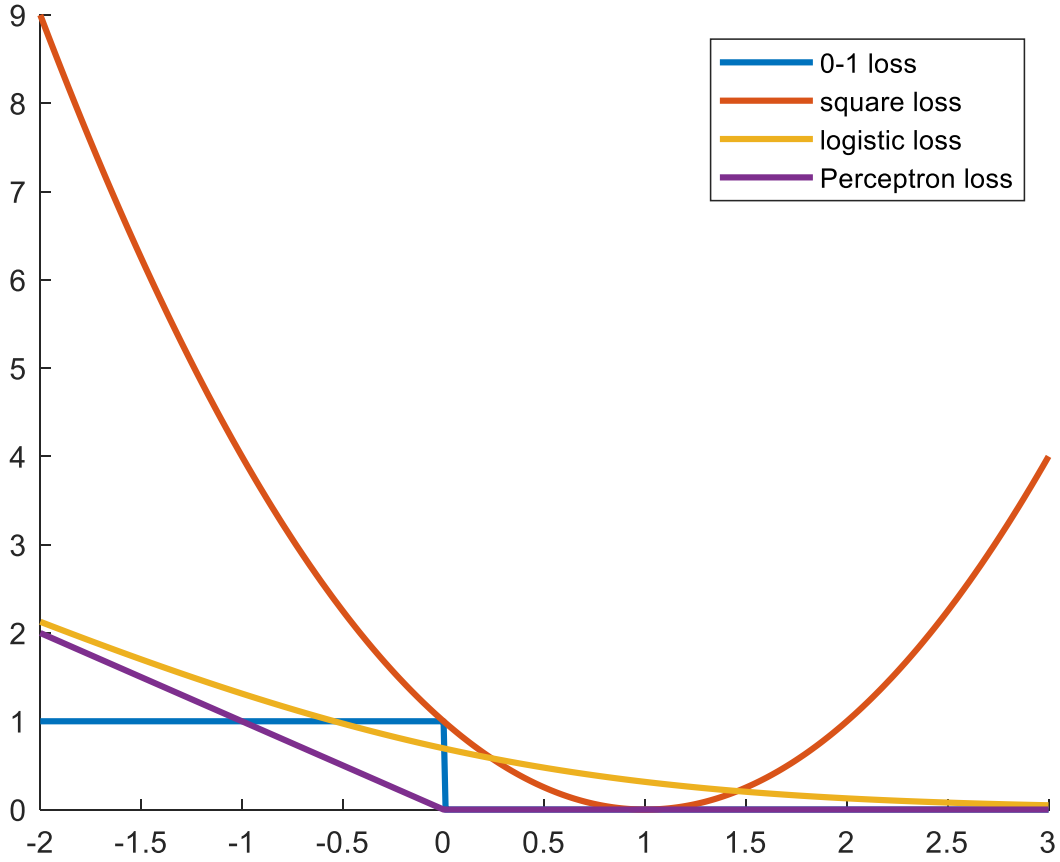
Logistic Regression



Adaptive Linear Neuron (Adaline)

二分类问题

- 两类分类器：感知机，线性回归，逻辑斯蒂回归

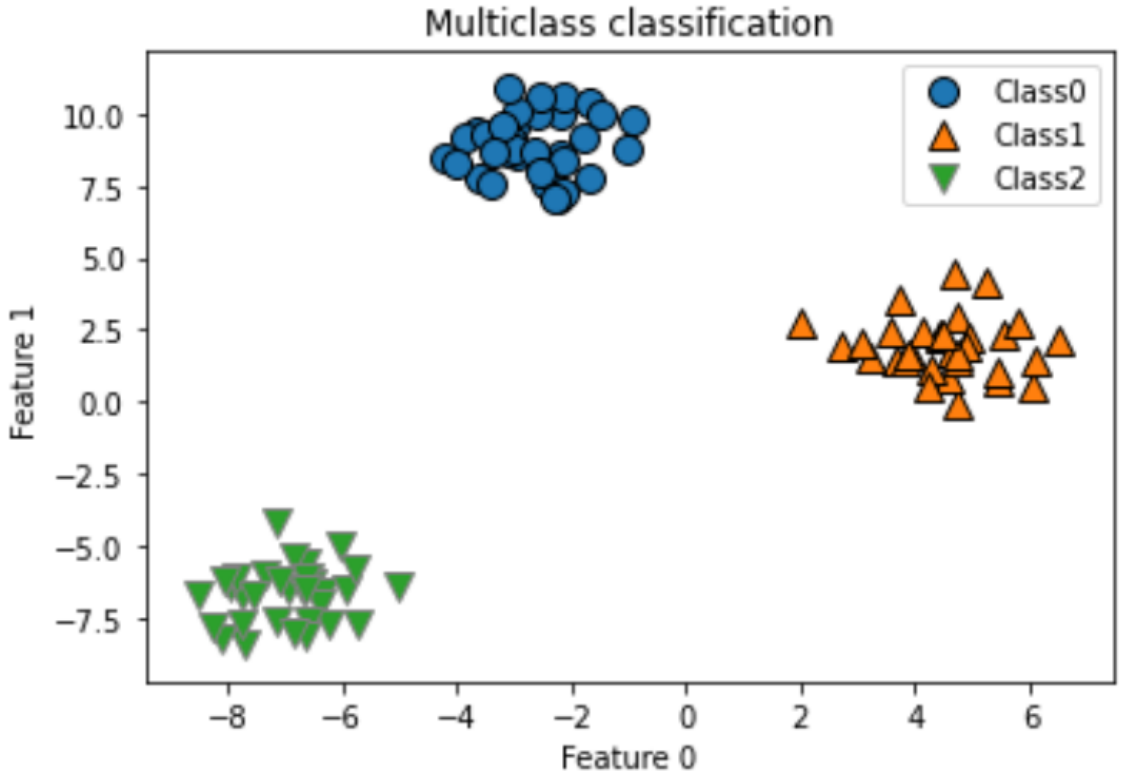


二分类与多分类问题

- 感知机，线性回归，逻辑斯蒂回归的共同优点
  - 无需对数据的分布进行假设
  - 易于训练
  - 对测试数据进行快速分类
- 逻辑斯蒂回归的独特优点
  - 对分类结果具有概率视角
  - 在很多简单数据集上能够取得好的分类结果
- 共同缺点
  - 线性决策边界 （线性可分与线性不可分）

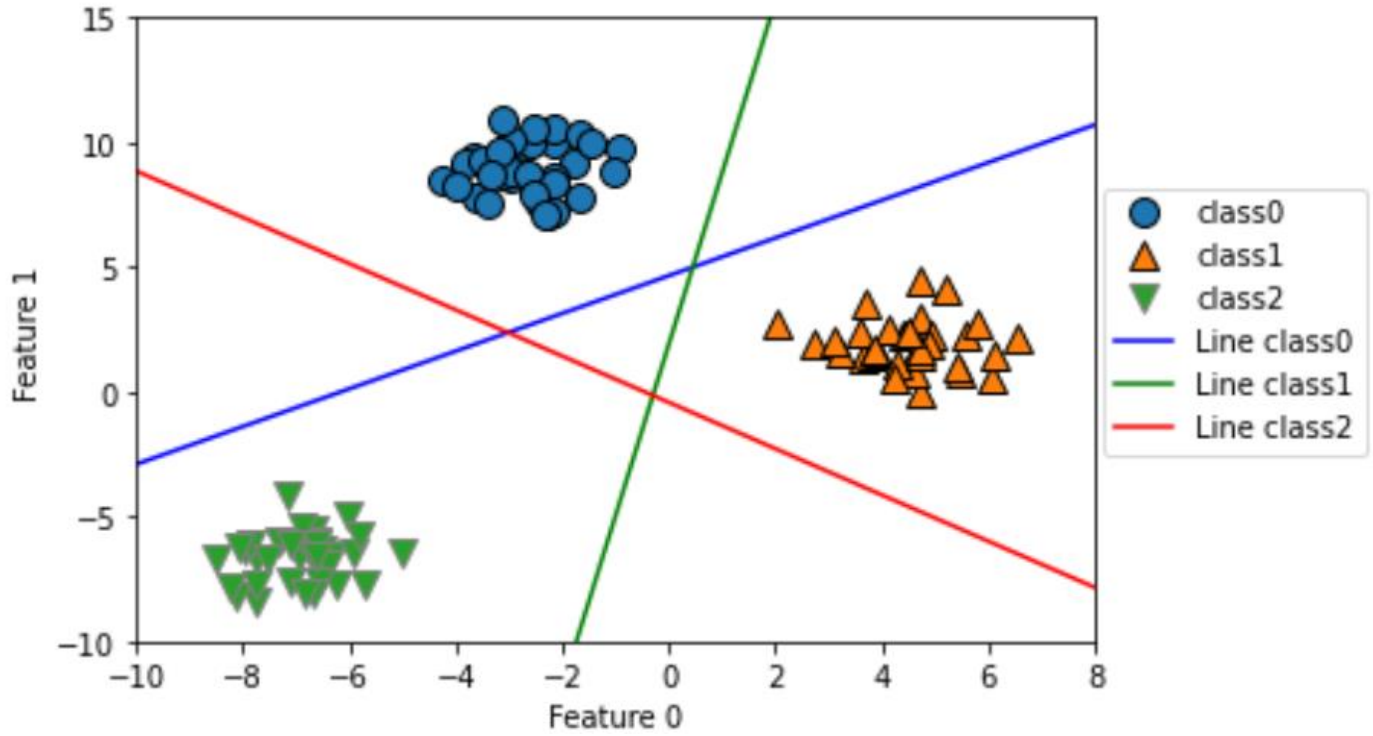
二分类与多分类问题

- 两类分类器：感知机，线性回归，逻辑斯蒂回归
- 如何实现多分类？



二分类与多分类问题

- 两类分类器：感知机，线性回归，逻辑斯蒂回归
- 如何用两类分类器实现多分类？
  - 一对多 (one-vs-rest, one –vs-all, one-over-all)



二分类与多分类问题

- 两类分类器：感知机，线性回归，逻辑斯蒂回归
- 如何用两类分类器实现多分类？
  - 一对一 (one-vs-one, pairwise classification)

decision_function_shape : *{'ovo', 'ovr'}, default='ovr'*

Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2). However, one-vs-one ('ovo') is always used as multi-class strategy. The parameter is ignored for binary classification.

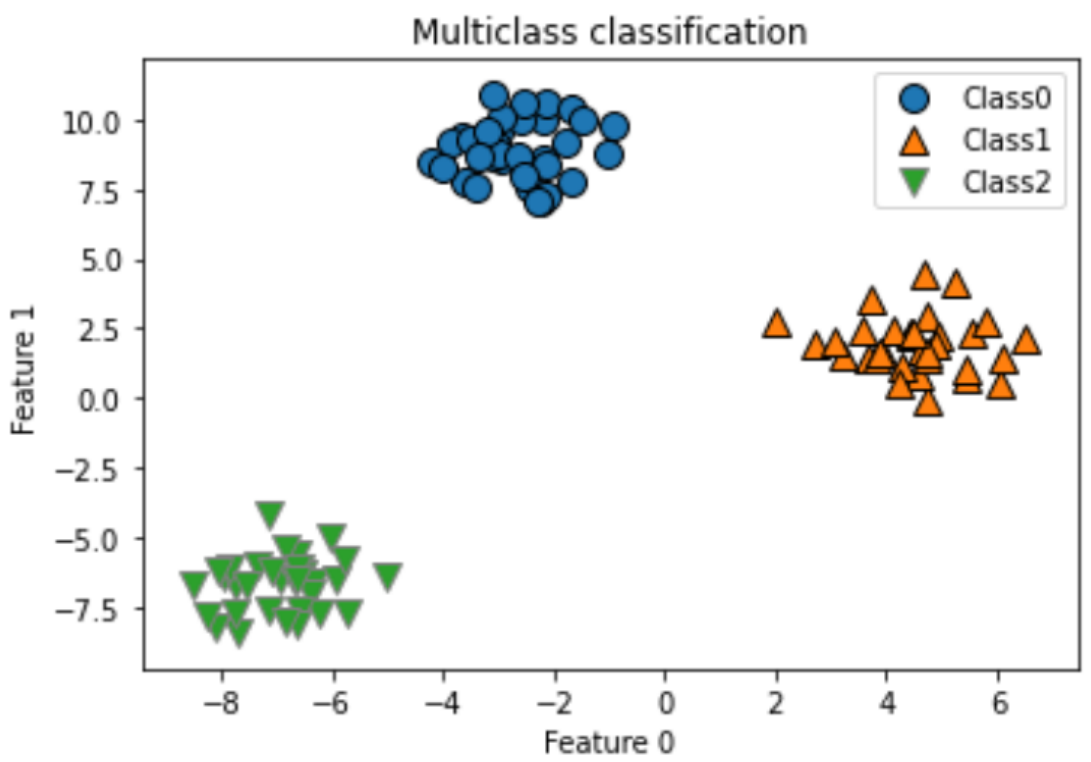
Changed in version 0.19: decision_function_shape is 'ovr' by default.

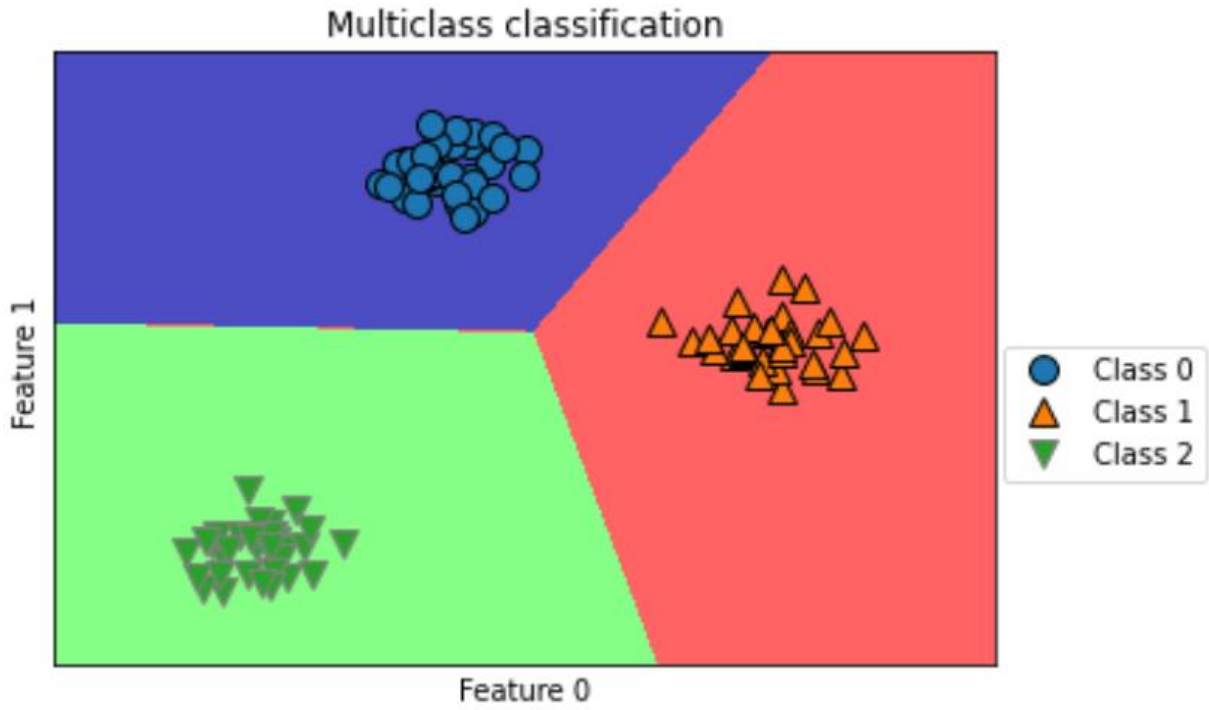
New in version 0.17: decision_function_shape='ovr' is recommended.

Changed in version 0.17: Deprecated *decision_function_shape='ovo' and None.*

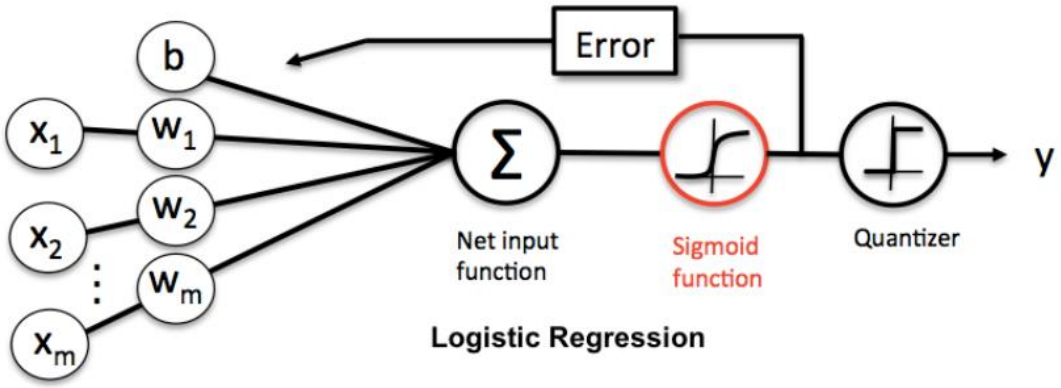
二分类与多分类问题

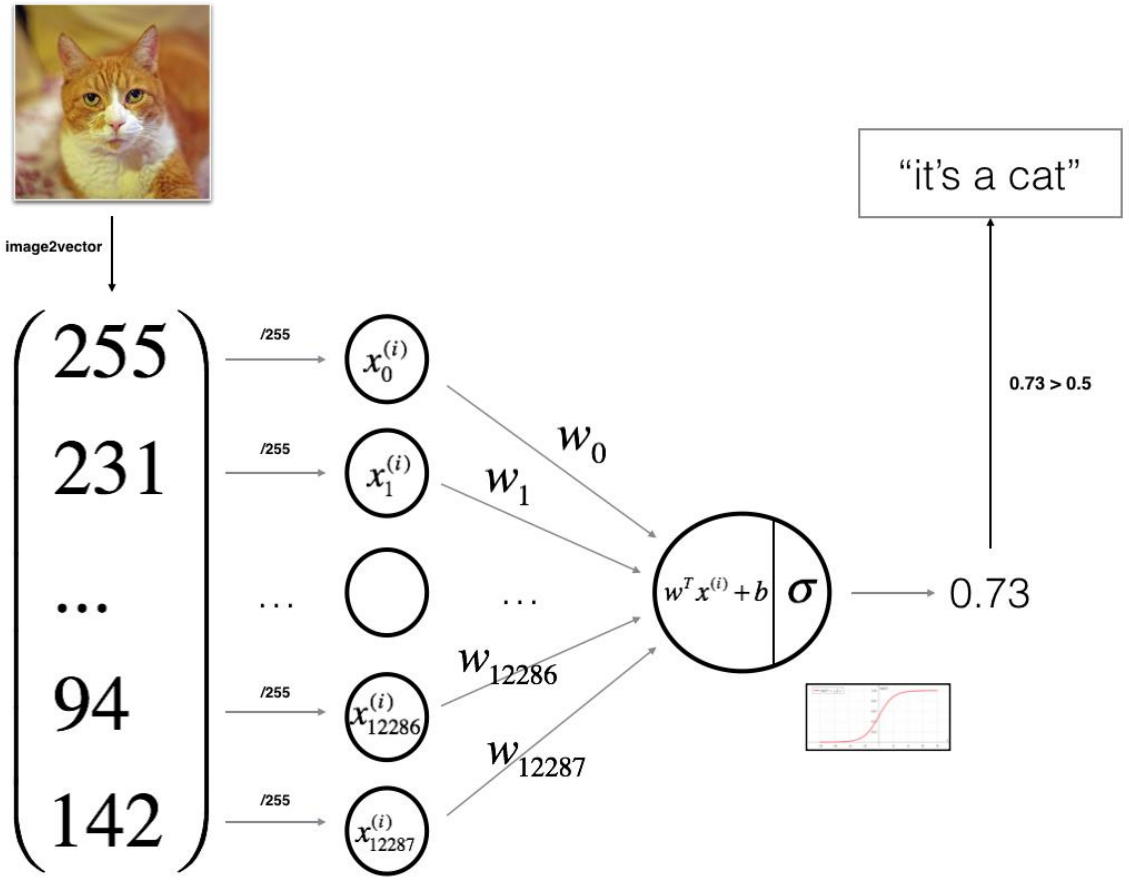
- 两类分类器：感知机，线性回归，逻辑斯蒂回归
- 如何实现多分类？



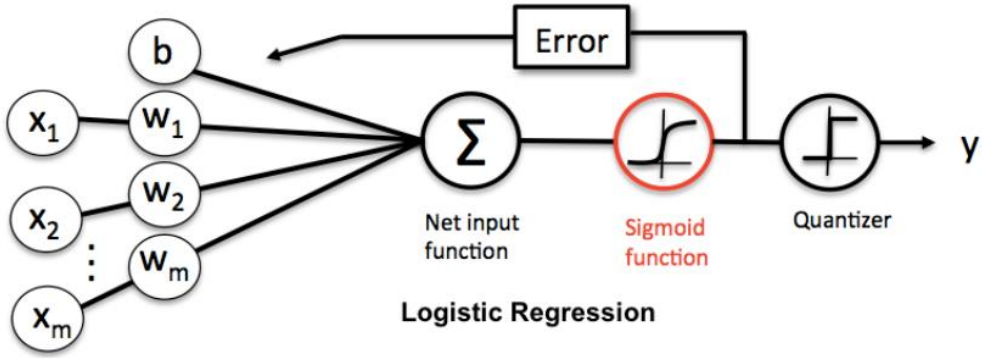


Logistic Regression 示意图

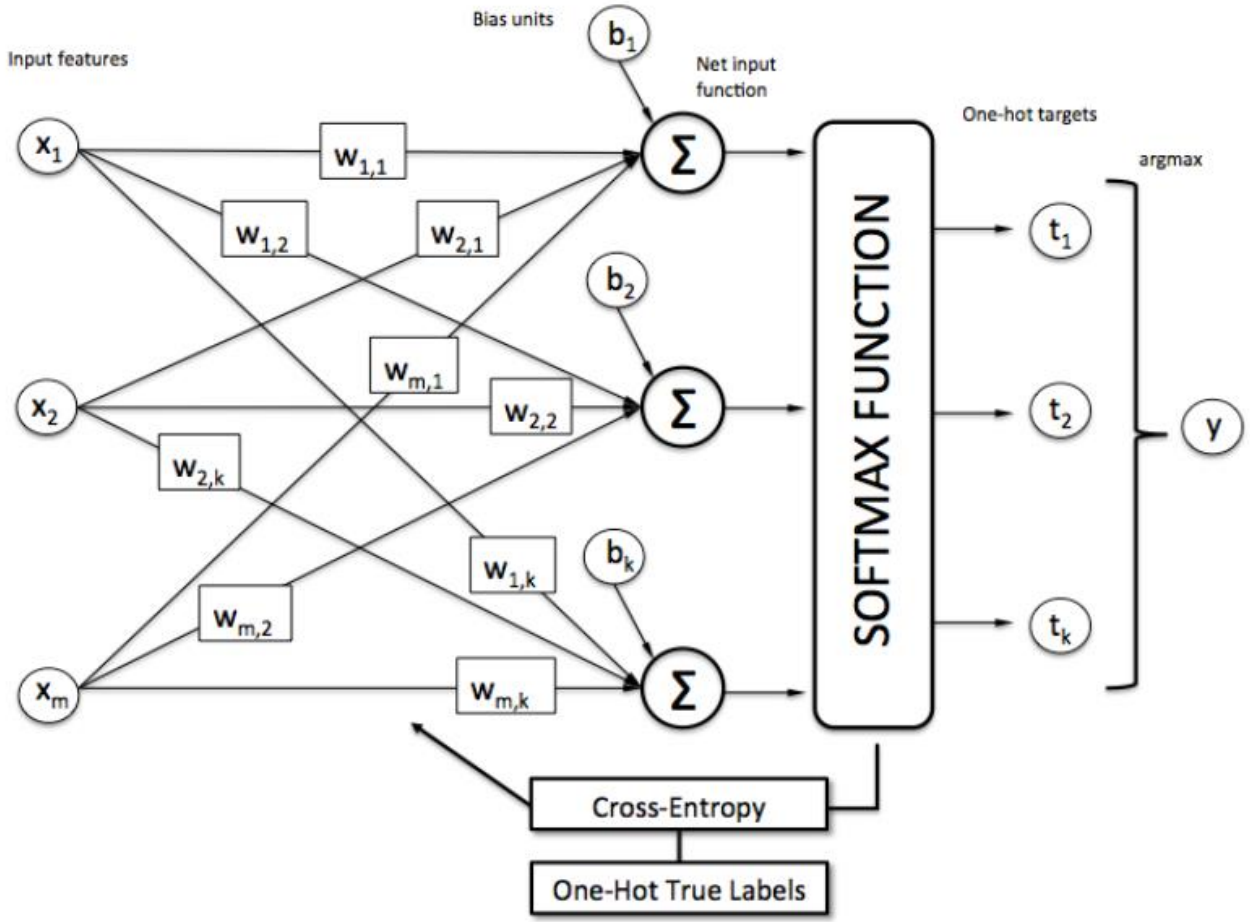




Logistic Regression 示意图

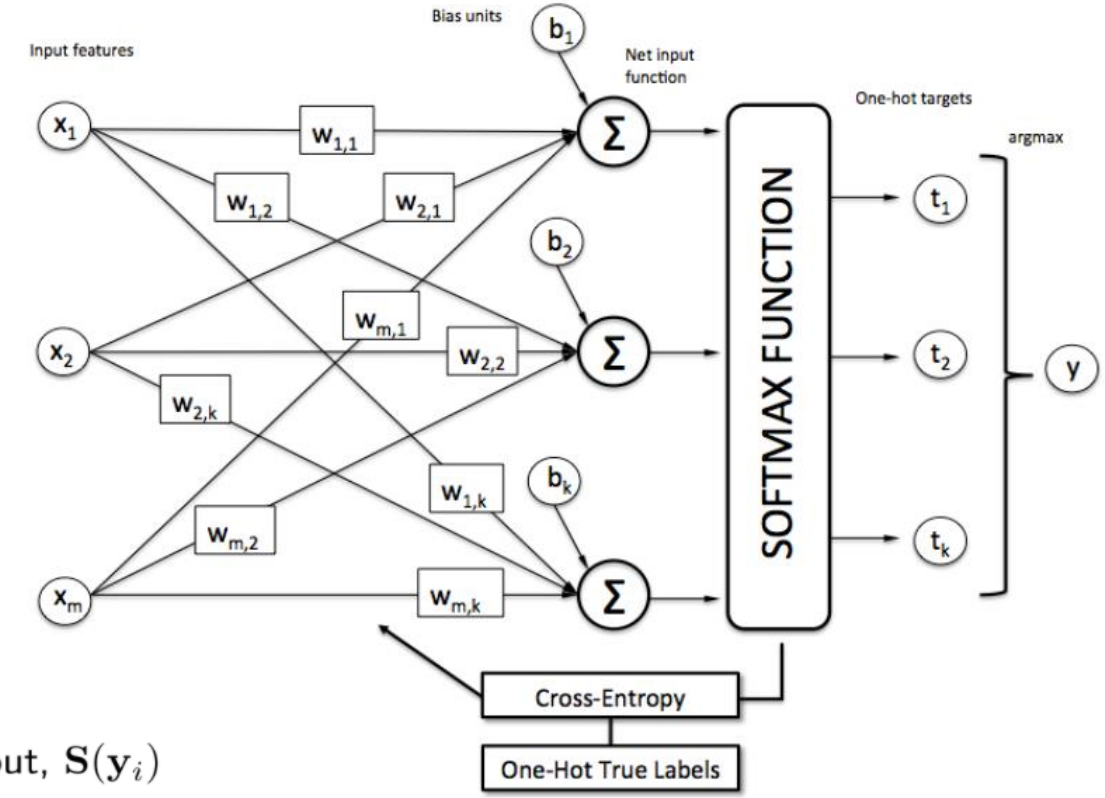


Logistic Regression



Softmax Regression

Softmax示意图



Input pixels, $\mathbf{x}$ Feedforward output, $\mathbf{y}_i$ Softmax output, $\mathbf{S}(\mathbf{y}_i)$

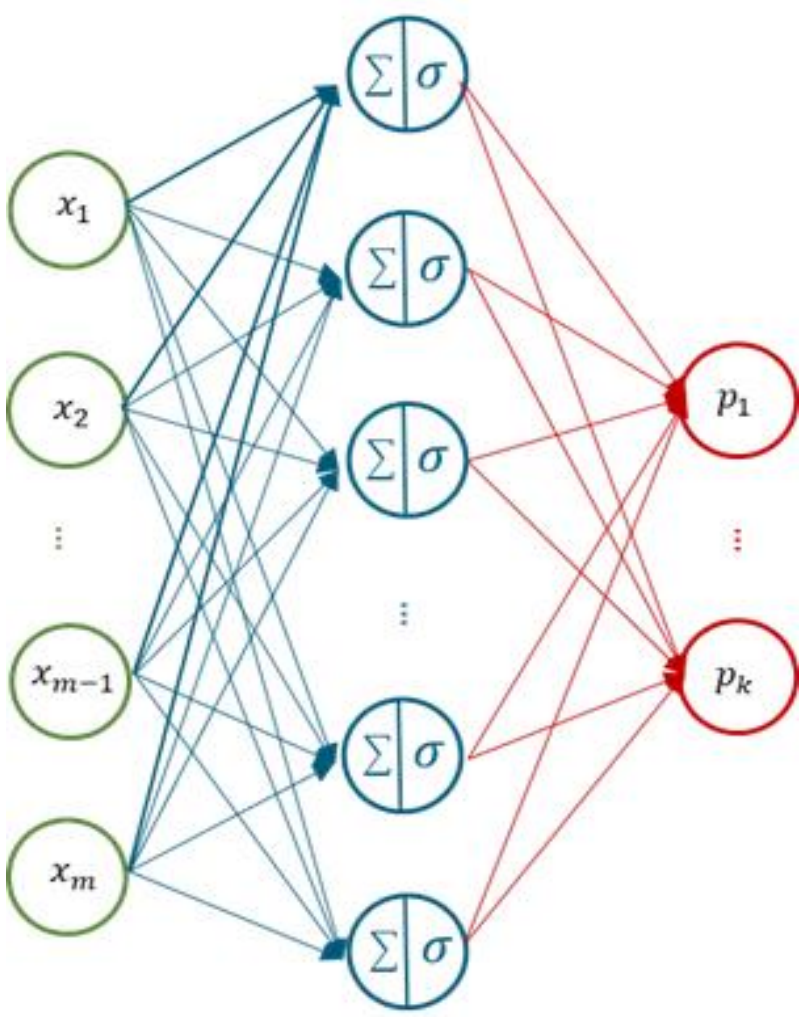
Forward propagation →

cat	dog	horse
5	4	2
4	2	8
4	4	1

Softmax function →

cat	dog	horse
0.71	0.26	0.04
0.02	0.00	0.98
0.49	0.49	0.02

Softmax示意图



- Softmax回归模型是Logistic回归模型在多分类问题上的推广，采用随机梯度下降法的方式对单一样本进行梯度的计算。
- 考虑随机训练样本$\boldsymbol{x}$和标签$\boldsymbol{y}$，其中$y_i = 1$且$y_{j\neq i} = 1$，如果样本$\boldsymbol{x}$属于第i类。

Softmax的计算

- 如果样本假定Softmax回归将输入数据$\boldsymbol{x}$归属于第i类别的概率p_i，要求使用这些样本，估计参数$\boldsymbol{W}$的值。

$$z_i = \sum_{j=1}^{m} w_{ij}\, x_j (i = 1,2,\cdots,k) \rightarrow \boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$$

$$p_i = \frac{e^{z_i}}{\sum_{s=1}^{k} e^{z_s}} (i = 1,2,\cdots,k) = \sigma_i(z_1, z_2, \cdots, z_k) \rightarrow \boldsymbol{p} = \sigma(z_1, z_2, \cdots, z_k)$$

其中$\boldsymbol{x} \in \mathcal{R}^{m\times 1}$，$\boldsymbol{W} \in \mathcal{R}^{k\times m}$，$\boldsymbol{z} \in \mathcal{R}^{k\times 1}$，$\boldsymbol{p} \in \mathcal{R}^{k\times 1}$。显然，$\sum_{i=1}^{k} p_i = 1$。

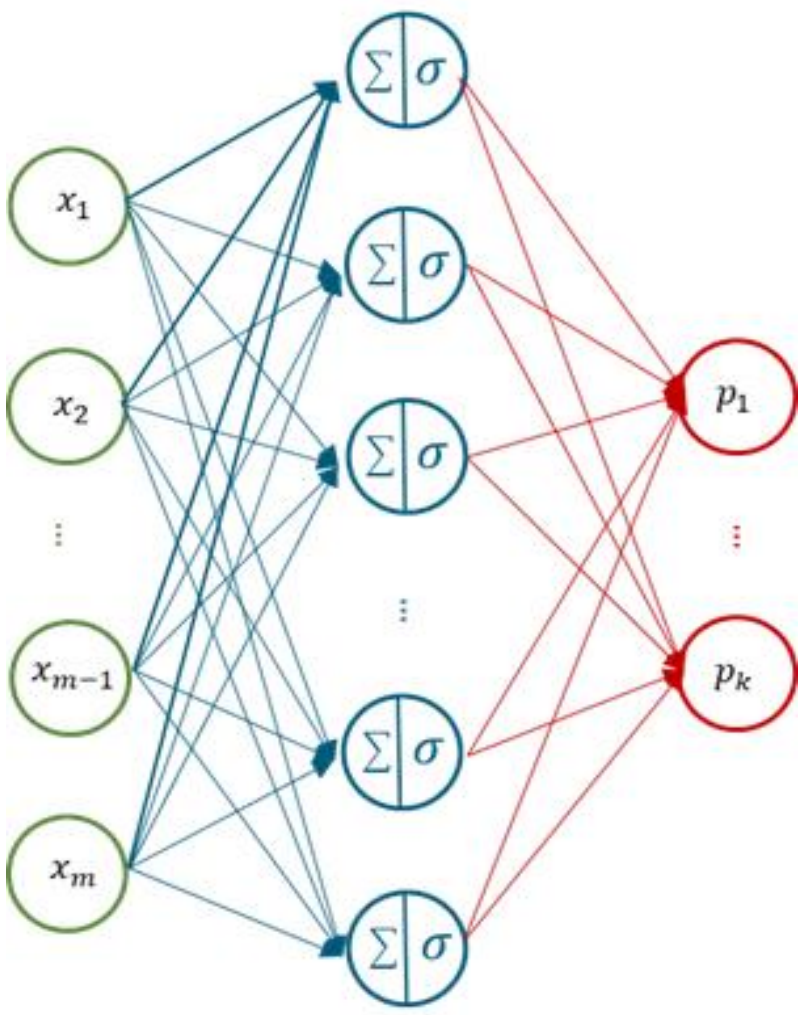
Softmax的目标函数

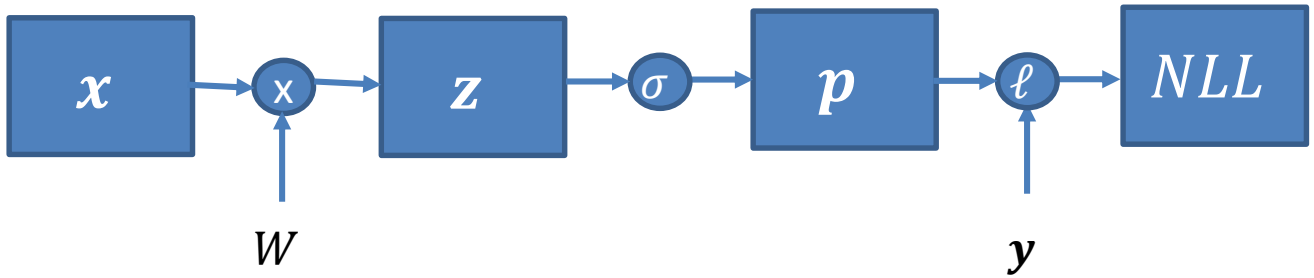
- 针对训练数据$\boldsymbol{x}$，如果$y_i = 1$且$y_{j\neq i} = 1$，我们希望其对应的输出概率p_i最大，$p_i \geq p_j$。使用似然函数的负对数函数(Negative Log Likelihood, NLL)作为损失函数（只考虑某个随机的$\boldsymbol{x} \in \mathcal{R}^{m\times 1}$，即考虑随机梯度下降法中梯度的计算问题）：

$$NLL(\boldsymbol{W}) = -\sum_{i=1}^{k}(y_i \ln p_i)$$

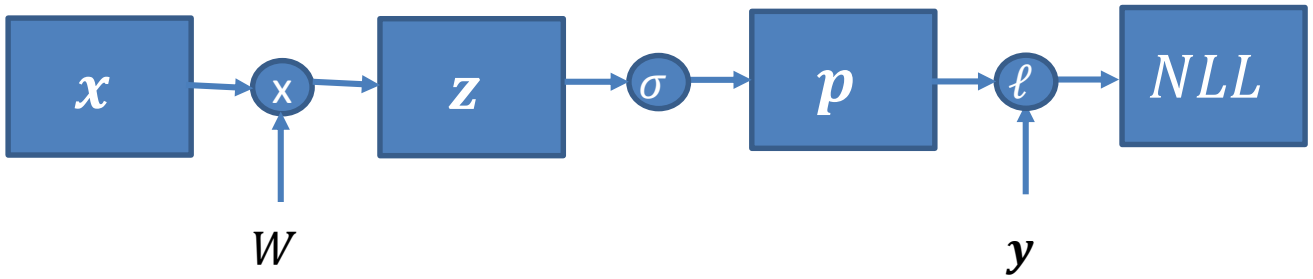
Softmax 就是要求$\boldsymbol{W}$，使得$NLL(\boldsymbol{W})$最小。

Softmax的求解





Softmax的求解



$$\frac{\partial NLL}{\partial p_i} = \frac{\partial(-\sum_{i=1}^{k}[y_i \ln p_i])}{\partial p_i} = -\frac{y_i}{p_i}$$

$$\frac{\partial NLL}{\partial z_i} = \sum_{l=1}^{k} \frac{\partial NLL}{\partial p_l} \frac{\partial p_l}{\partial z_i} (i = 1,2,\cdots,k)$$

Softmax的求解

每一个p_l与所有的$z_i(i=1,2,\cdots,k)$都有关系，因此求导时需要分$l=i$和$l\neq i$讨论

$l=i$时：

$$\frac{\partial p_l}{\partial z_i}=\frac{\partial p_l}{\partial z_i}=\frac{\partial \frac{e^{z_l}}{\sum_k e^{z_k}}}{\partial z_i}=\frac{e^{z_i}\sum_k e^{z_k}-(e^{z_i})^2}{(\sum_k e^{z_k})^2}=\frac{e^{z_i}}{\sum_k e^{z_k}}\left(1-\frac{e^{z_i}}{\sum_k e^{z_k}}\right)=p_i(1-p_i)$$

$l\neq i$时：

$$\frac{\partial p_l}{\partial z_i}=\frac{\partial \frac{e^{z_l}}{\sum_k e^{z_k}}}{\partial z_i}=-\frac{e^{z_i}e^{z_l}}{(\sum_k e^{z_k})^2}=-p_ip_l$$

Softmax的求解

把两部分相结合，可以得到:

$$\frac{\partial NLL}{\partial z_i} = \sum_{l=1}^{k}(-\frac{y_l}{p_l})\frac{\partial p_l}{\partial z_i} = -\frac{y_i}{p_i}p_i(1-p_i) + \sum_{l\neq i} -\frac{y_l}{p_l}p_i p_l$$

$$= -y_i + y_i p_i + \sum_{l\neq i} y_l\, p_i = -y_i + p_i \sum_{l} y_l = p_i - y_i$$

Softmax的求解

由 $\frac{\partial z_i}{\partial w_{ij}} = x_j$

可得

$$\frac{\partial NLL}{\partial w_{ij}} = \frac{\partial NLL}{\partial z_i}\frac{\partial z_i}{\partial w_{ij}} = (p_i - y_i)x_j$$

$$\frac{\partial NLL}{\partial \boldsymbol{W}} = \begin{bmatrix} \frac{\partial NLL}{\partial w_{11}} & \frac{\partial NLL}{\partial w_{12}} & \cdots & \frac{\partial NLL}{\partial w_{1m}} \\ \frac{\partial NLL}{\partial w_{21}} & \frac{\partial NLL}{\partial w_{22}} & \cdots & \frac{\partial NLL}{\partial w_{2m}} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial NLL}{\partial w_{k1}} & \frac{\partial NLL}{\partial w_{k2}} & \cdots & \frac{\partial NLL}{\partial w_{km}} \end{bmatrix}$$

$$= (\boldsymbol{p} - \boldsymbol{y})\boldsymbol{x}^T$$

Softmax的求解

由 $\frac{\partial z_i}{\partial w_{ij}} = x_j$

可得

$$\frac{\partial NLL}{\partial w_{ij}} = \frac{\partial NLL}{\partial z_i}\frac{\partial z_i}{\partial w_{ij}} = (p_i - y_i)x_j$$

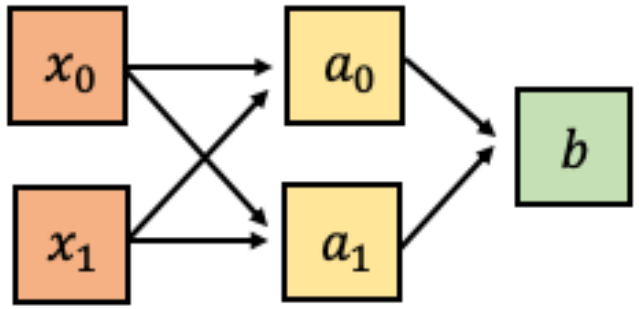
$$\frac{\partial NLL}{\partial \boldsymbol{W}} = \begin{bmatrix} \frac{\partial NLL}{\partial w_{11}} & \frac{\partial NLL}{\partial w_{12}} & \cdots & \frac{\partial NLL}{\partial w_{1m}} \\ \frac{\partial NLL}{\partial w_{21}} & \frac{\partial NLL}{\partial w_{22}} & \cdots & \frac{\partial NLL}{\partial w_{2m}} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial NLL}{\partial w_{k1}} & \frac{\partial NLL}{\partial w_{k2}} & \cdots & \frac{\partial NLL}{\partial w_{km}} \end{bmatrix}$$

$$= (\boldsymbol{p} - \boldsymbol{y})\boldsymbol{x}^T$$

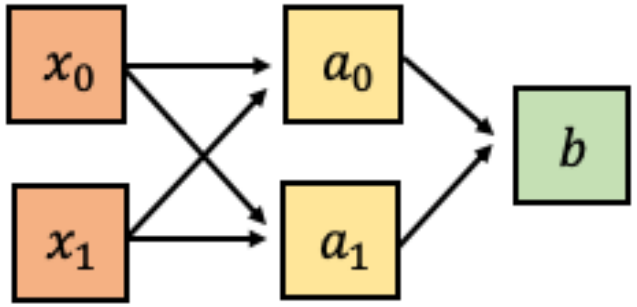
向量形式求导的链式法则

- The chain rule is a formula for computing the derivatives of the composition of two or more functions. To understand the chain rule, consider a simple example of a composite function. Let

$$b(x_0, x_1) = f\big(a_0(x_0, x_1), a_1(x_0, x_1)\big)$$



向量形式求导的链式法则

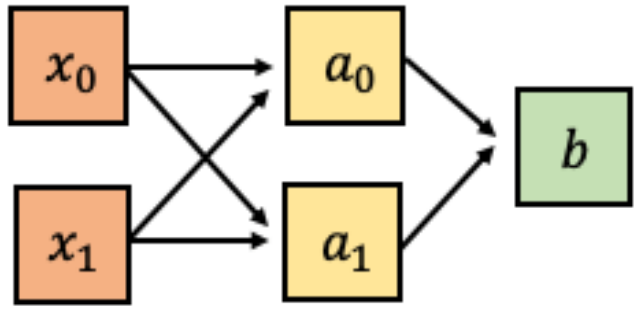


The goal is to compute the gradient of the final output value b with respect to each variable of (x_0, x_1). Using the chain rule, we have

$$\frac{db}{dx_0} = \frac{db}{da_0}\frac{da_0}{dx_0} + \frac{db}{da_1}\frac{da_1}{dx_0} \tag{10.14}$$

$$\frac{db}{dx_1} = \frac{db}{da_0}\frac{da_0}{dx_1} + \frac{db}{da_1}\frac{da_1}{dx_1} \tag{10.15}$$

向量形式求导的链式法则

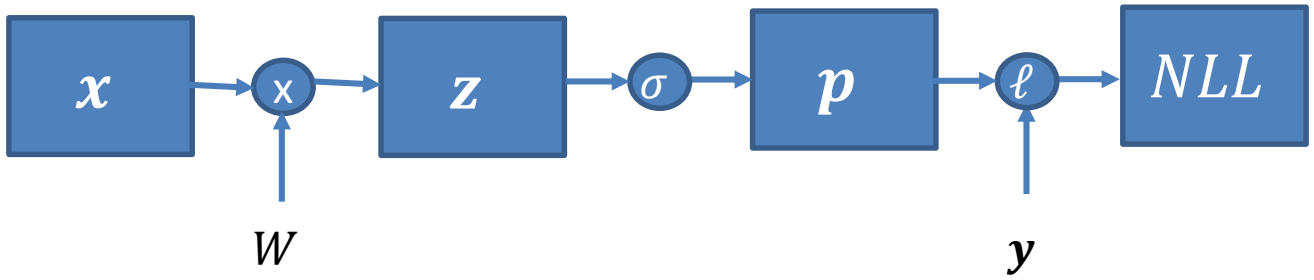


Let $\boldsymbol{x} = [x_0, x_1]^T$ and $\boldsymbol{a} = [a_0, a_1]^T$, then the derivation of the gradient can be written into vector form as:

$$\frac{\partial b}{\partial \boldsymbol{x}^T} = \left[\frac{db}{dx_0}, \frac{db}{dx_1}\right] = \left[\frac{db}{da_0}, \frac{db}{da_1}\right] \begin{bmatrix} \frac{da_0}{dx_0} & \frac{da_0}{dx_1} \\ \frac{da_1}{dx_0} & \frac{da_1}{dx_1} \end{bmatrix} = \frac{\partial b}{\partial \boldsymbol{a}^T} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}^T} \tag{10.16}$$

The vector form is simple and easy to use, which we will use in the derivation throughout this book.

向量形式求导的链式法则



$$\frac{\partial NLL}{\partial vec(\boldsymbol{W})^T} = \frac{\partial NLL}{\partial \boldsymbol{p}^T}\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{z}^T}\frac{\partial \boldsymbol{z}}{\partial vec(\boldsymbol{W})^T}$$

$$\frac{\partial NLL}{\partial \boldsymbol{p}^T} = -(\mathrm{diag}(\boldsymbol{p})^{-1}\boldsymbol{y})^T$$

$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{z}^T} = \mathrm{diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^T$$

$$\frac{\partial \boldsymbol{z}}{\partial vec(\boldsymbol{W})^T} = \boldsymbol{x}^T \otimes \boldsymbol{I}_k$$

$$\frac{\partial NLL}{\partial vec(\boldsymbol{W})^T} = -(\boldsymbol{y} - \boldsymbol{p})^T \boldsymbol{x}^T \otimes \boldsymbol{I}_k$$

向量形式求导的链式法则

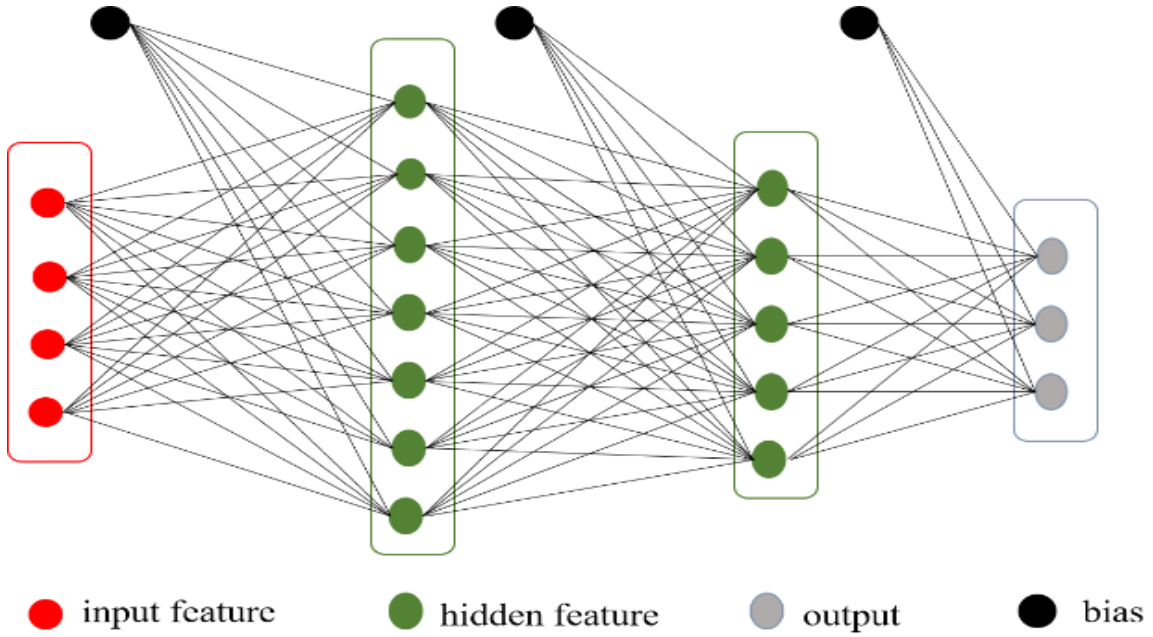
$$\frac{\partial NLL}{\partial vec(\boldsymbol{W})^T} = -(\boldsymbol{y}-\boldsymbol{p})^T \boldsymbol{x}^T \otimes \boldsymbol{I}_k$$

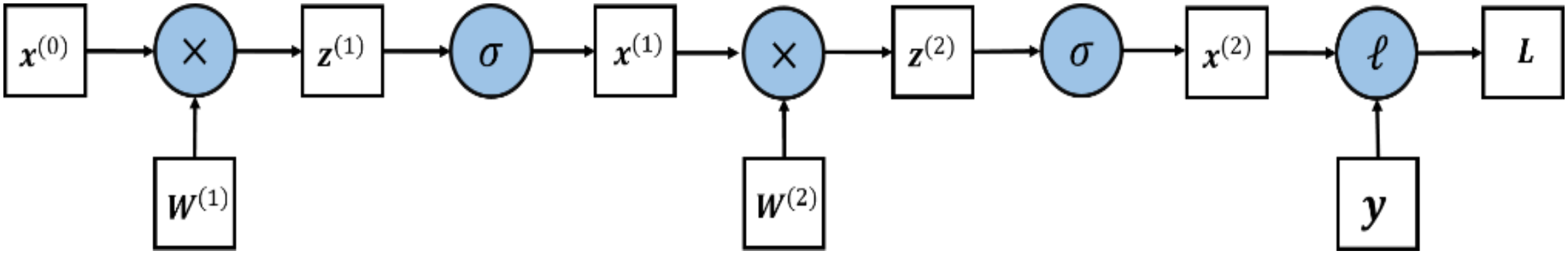
$$\frac{\partial NLL}{\partial vec(\boldsymbol{W})} = -\boldsymbol{x} \otimes \boldsymbol{I}_k (\boldsymbol{y}-\boldsymbol{p})$$

由 $vec(\boldsymbol{ABC}) = (\boldsymbol{C}^T \otimes \boldsymbol{A})\mathrm{vec}(\boldsymbol{B})$

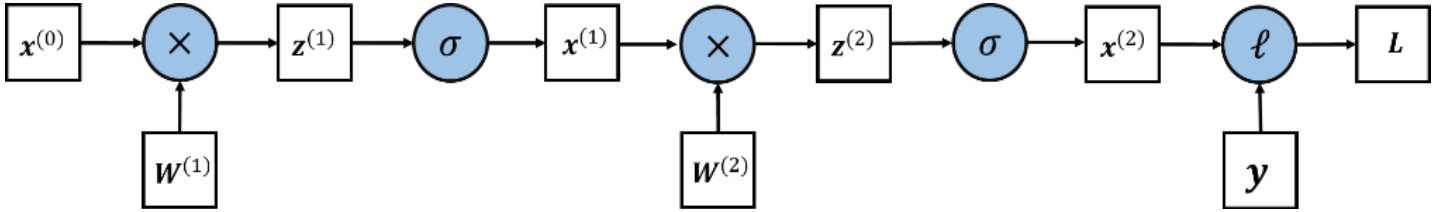
可得 $\frac{\partial NLL}{\partial \boldsymbol{W}} = -(\boldsymbol{y}-\boldsymbol{p})\boldsymbol{x}^T$

多层感知机

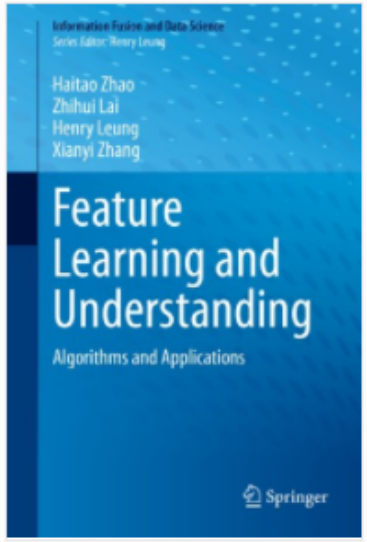

input feature
hidden feature
output
bias


$x^{(0)}$
$\times$
$W^{(1)}$
$z^{(1)}$
σ
$x^{(1)}$
$\times$
$W^{(2)}$
$z^{(2)}$
σ
$x^{(2)}$
ℓ
y
L

广告时间



Springer Link



Feature Learning and Understanding

Algorithms and Applications

Authors (view affiliations)

Haitao Zhao, Zhihui Lai, Henry Leung, Xianyi Zhang

Offers advanced feature learning methods, such as sparse learning, and deep-learning-based feature learning

Includes also traditional and cutting-edge feature learning methods

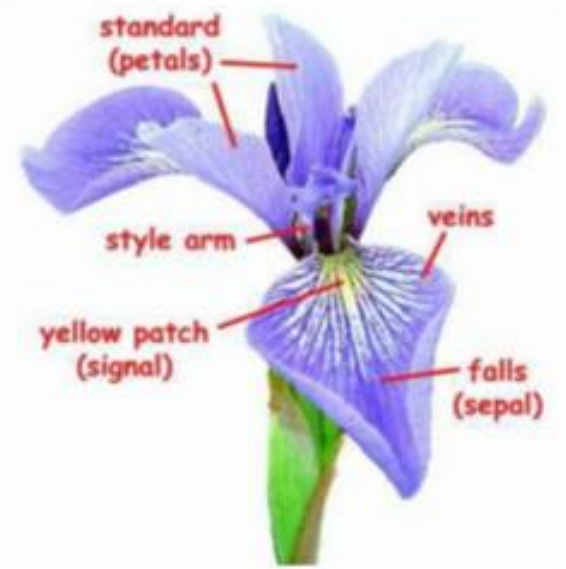
Contains the detailed theoretical analysis of each feature learning method

https://link.springer.com/book/10.1007/978-3-030-40794-0

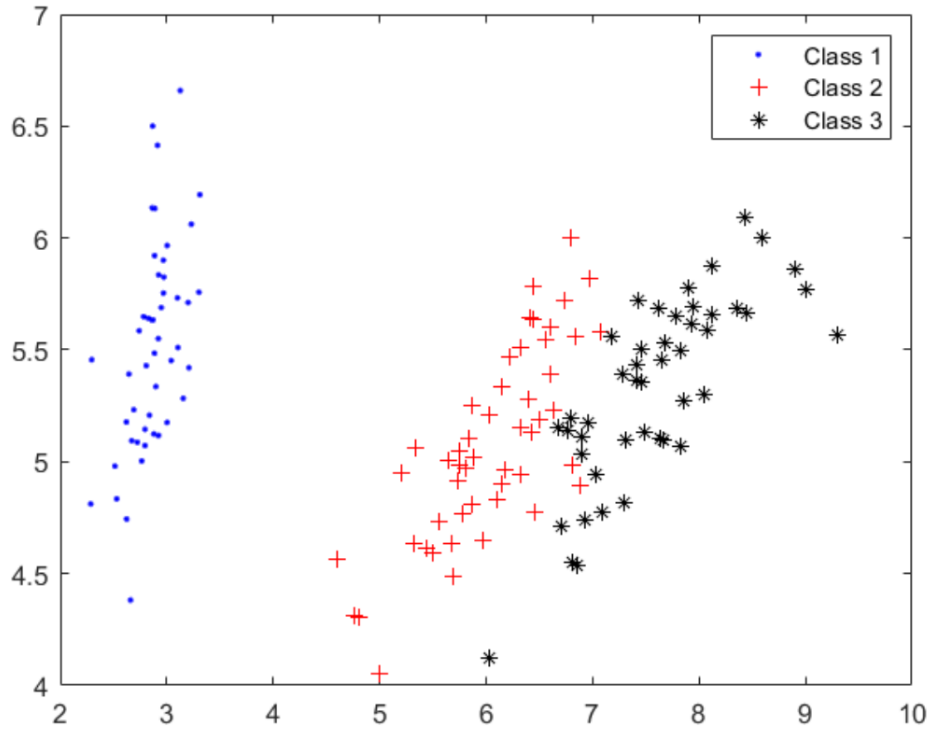
Softmax实验

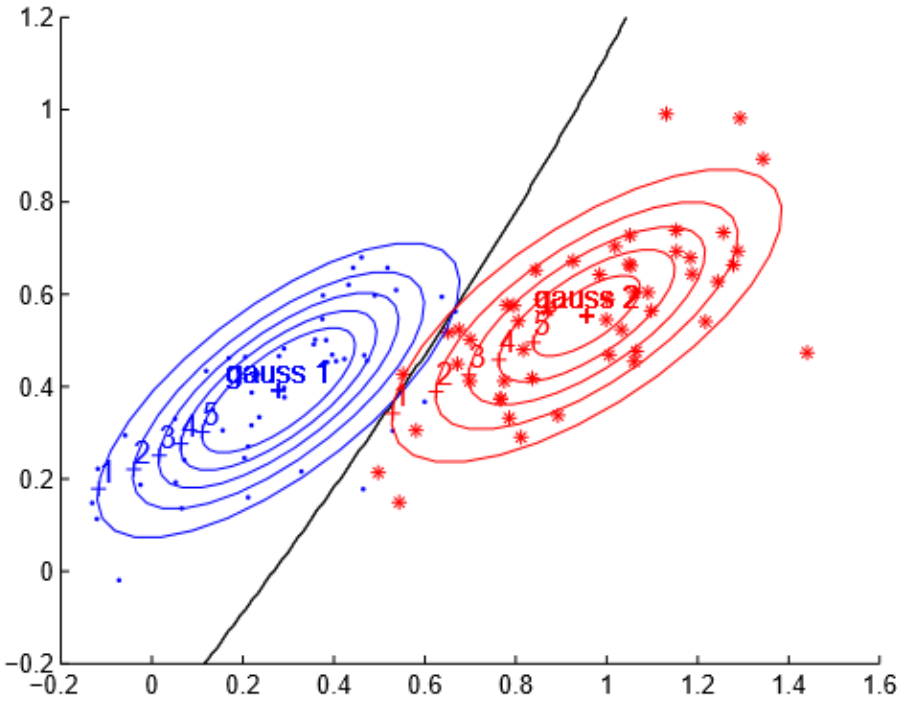
Iris Data Set

- **Contents**: 150 Objects, 4 Variables
- **Source**: E. Anderson, The irises of the Gaspé peninsula, Bulletin of the American Iris Society, 59 (1935) 2-5
- **Description**: there are a family of Iris flowers, this data set measure only three of possible variance. Perhaps this is the more famous date set. It is famous for its intrinsic simplicity but also for the difficulty to find a statistic method that carefully separates the three classes. Unfortunately there are in the wild MANY badly iris data set! Notation can be done of R.A. Fisher works on iris data set.
- **Variables**: sepal-length, sepal-width, petal-length, petal-width
- **Classes**: 50 Iris setosa canadensis, 50 Iris versicolor, 50 Iris virginica



Softmax实验


Class 1
Class 2
Class 3
7
6.5
6
5.5
5
4.5
4
2
3
4
5
6
7
8
9
10


gauss 1
gauss 2
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.2
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6

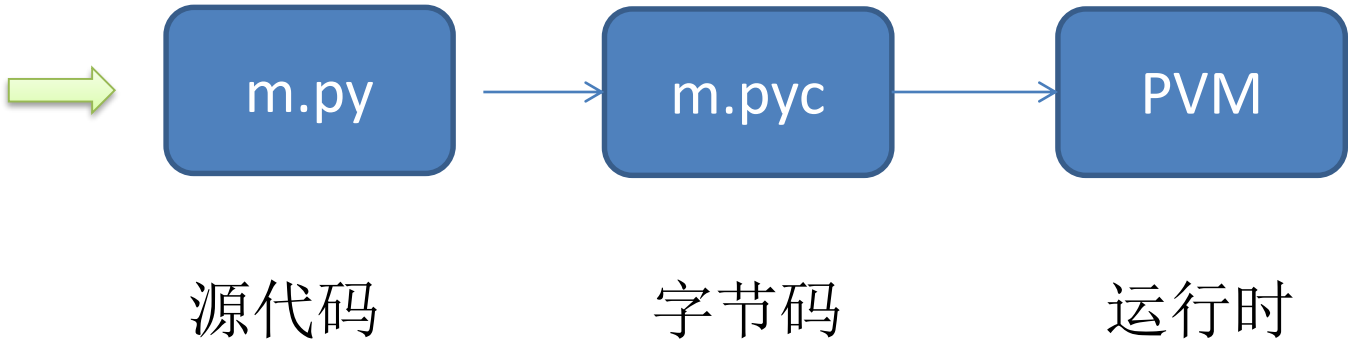
Python程序调试

- Python传统运行模式：

  Python解释器： 运行Python程序的程序

  Python字节码： Python将程序编译后所得到的底层形式

  Python自动将字节码保存为名为.pyc的文件中



- 录入的源码转换为字节码->字节码在PVM（Python虚拟机）中运行->代码自动被编译，之后再解释

Python程序调试

- Python无“build”和“make”的步骤，代码写好后立即运行
- Python字节码不是机器的二进制代码（so 不能像C++运行速度那么快，其速度介于传统编译语言和传统解释语言之间）

Python模块

- 每一个.py文件都是一个模块，其他文件可以通过导入一个模块读取这个模块的内容，相当于C中的include……
- 一个大型程序往往呈现出多模块的形式。
- 其中一个模块文件被设计为主文件（or顶层文件）。

- 模块是Python程序最大的程序结构
- 每个模块文件是一个独立完备的变量包装，即一个命名空间

Python模块

模块的导入

- import，from 和 reload

  import语句将模块作为一个整体引用，相当于引入一个类的object。

  From 增加了对变量名的额外赋值，也就是拷贝模块的属性，因此能够以主模块导入，而不是原来的对象

Softmax实验

1. iris数据读入

```
from sklearn import datasets
import numpy as np

# 读入 iris数据
iris_df = datasets.load_iris()
x = iris_df.data
x = x/np.max(x)
y = iris_df.target
```

```
# encode y into onehot format
nb_classes = 3
targets = train_y.reshape(-1)
y_onehot=
np.eye(nb_classes)[targets]
```

Softmax实验

2. 划分测试集和验证集

```
from sklearn.model_selection import train_test_split


# 划分训练集和测试集
train_X,test_X, train_y, test_y = train_test_split(x,
                                                   y,
                                                   test_size = 0.3,
                                                   random_state = 0)
```

Softmax实验

3. 梯度下降法求解参数W

```
# 初始化W
num_train = train_y.shape[0]
W0 = np.random.random([3,4])

# 批处理方法求解W
iter = 0
r =0.01
while  iter < 1000:
 #   myrand = np.random.permutation(range(num_train))
    deltaW = np.zeros([3,1])
    temp_p = mysoft(W0.dot(train_X.T))
    deltaW = (y_onehot.T-temp_p).dot(train_X)
    W0 = W0 + r * deltaW
    iter = iter + 1
```

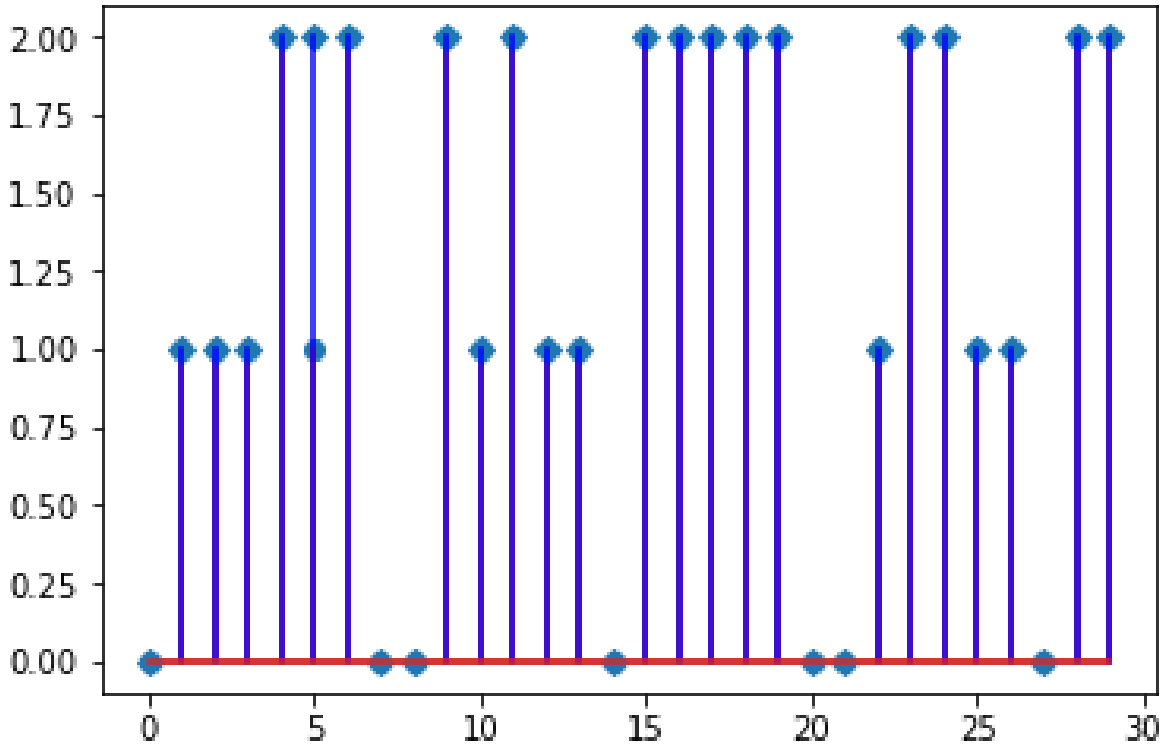
Softmax实验

4. 在验证集上进行验证

```
#对验证集进行测试，并得到识别率
tmp_pre = mysoft(W0.dot(test_X.T))
indice = np.argmax(tmp_pre,axis=0)
pre = indice

fig = plt.figure(num=1)
plt.stem(test_y,linefmt='red')
plt.stem(pre,linefmt='blue', markerfmt='D')

correct = pre.reshape(-1) == test_y.reshape(-1)
acc = sum(correct.astype(int))/test_y.shape[0]
```



Softmax实验

5. 定义的softmax函数

```
def mysoft(x):
    y = np.exp(x)
    y = y/np.sum(y,axis=0)
    return y
```

谢谢各位同学！