

## 多分类问题

## 一、 Introduction

机器学习中遇到的许多模型都是二分类模型，如感知机、逻辑回归、SVM 等等，但生活中的问题往往是多分类问题，所以需要一定的策略将其转换为多分类问题。策略总体可以分为两类：一类是训练多个二分类模型解决多分类问题（OVA 与 OVO），一类是训练一个模型解决多分类问题（softmax）。其方法具体如下。

## 二、 OVA 策略

## 1. 硬间隔

对于每一个类别训练一个分类器，输出标签  $\{+1, -1\}$  判断是否为该类。预测时遍历所有分类器，判断  $\mathbf{x}_i$  是否属于类别  $k$

缺陷：不能满足以下两种情况：

- (1) 类与类之间是互斥的：如果是猫，就不能是狗
- (2) 只有若干类：不是男性，就是女性，不是数字 1-9，就是数字 0

对于情况 1，硬间隔的方法可能有多个分类器同时认为某个样本属于正类，对于情况 2，可能所有分类器都认为某个样本属于负类。

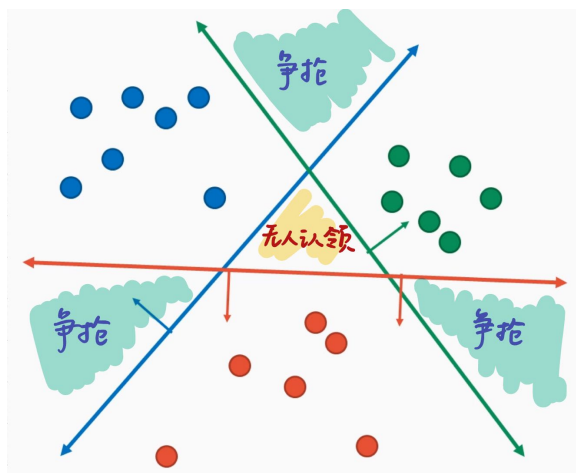


图 1: OVA 策略

## 2. 软间隔

同样对于每一个类别训练一个分类器，但是将输出的标签从  $\{+1, -1\}$  转换为  $p(y_i|\mathbf{x}_i)$ ，在预测时则选择概率最大的进行输出。这样就可以保证有且仅有一个输出，从而避免硬间隔的两个问题。

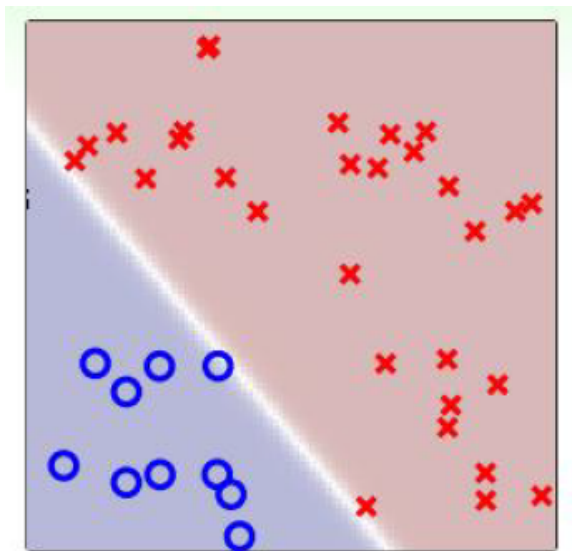


图 2: OVA 策略

---

### Algorithm 1 Logistic Regression with SGD (OVA)

---

**Input:**  $\mathbf{x}, y, \alpha_0, T, num\_classes$

- 1: 生成  $num\_classes$  个 classifier
  - 2: **repeat**
  - 3:   随机在  $[1, m]$  中选取一个  $idx$
  - 4:   **for** classifier **in** classifiers **do**
  - 5:     **compute**  $g = \sigma(-y_i \mathbf{w}^\top \mathbf{x}_i)(-y_i \mathbf{x}_i)$
  - 6:     **update**  $\mathbf{w} \leftarrow \mathbf{w} - \alpha_t g$
  - 7:   **end for**
  - 8: **until**  $step \geq T$
- 

## 三、 OVO 策略

每次训练两个类别的所有样本，输出  $\{+1, -1\}$  判断样本是 A 类还是 B 类。

## 四、 交叉熵

与逻辑回归类似，将 sigmoid 推广到多元得到 softmax，输出每一个类别的相应概率，同时通过交叉熵损失函数进行求解

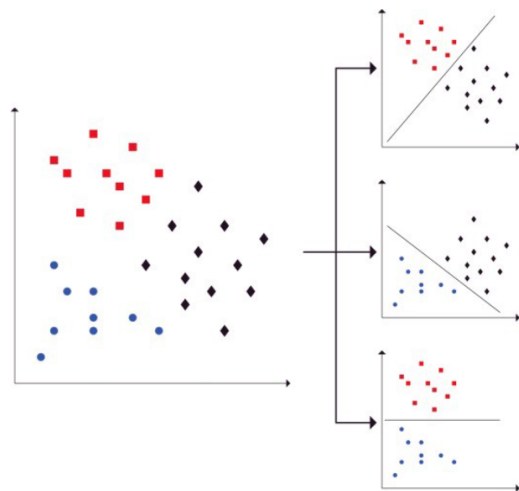


图 3: OVO 策略

**Definition 1** (softmax).

$$p_{ik} = \frac{e^{\mathbf{w}_k \mathbf{x}_{ik}}}{\sum_{c=1}^K e^{\mathbf{w}_c \mathbf{x}_{ic}}}$$

**Definition 2** (交叉熵).

$$\text{cross entropy} = - \sum_{c=1}^K y_c \ln(p_c(\mathbf{x}))$$

---

**Algorithm 2** softmax with SGD

---

**Input:**  $\mathbf{x} \in \mathbb{R}^{m \times n}, y, \mathbf{w} \in \mathbb{R}^{K \times n}, s_t, T, \text{num\_classes}$

- 1: **repeat**
- 2:   随机在  $[1, m]$  中选取一个  $\text{idx}$
- 3:   **compute**

$$p_i = \frac{e^{\mathbf{w} \mathbf{x}_i}}{\sum_{c=1}^K e^{\mathbf{w}_c \mathbf{x}_{ic}}}$$

- 4:   **compute**  $g = (p_i - \text{one\_hot}(y_i)) \mathbf{x}_i^\top$
  - 5:   **Update**  $\mathbf{w} \leftarrow \mathbf{w} - s_t g$
  - 6: **until**  $g < \varepsilon$  或  $t \geq T$
- 

## 五、 代码实现

本次实验中在 MNIST 数据集上分别用 OVO 和 OVA 和交叉熵三种方法进行训练。其结果如图4所示

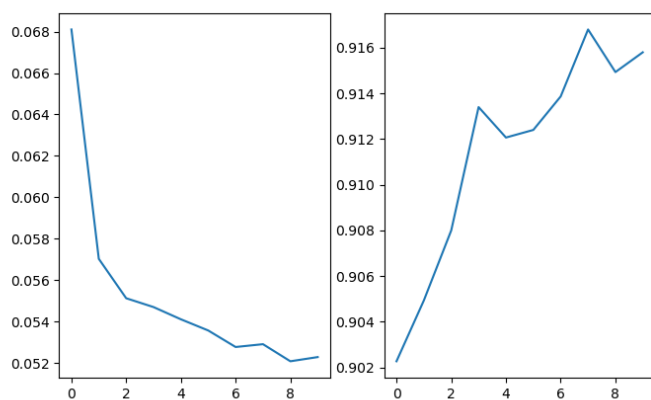


图 4: OVA loss 与 acc 变化

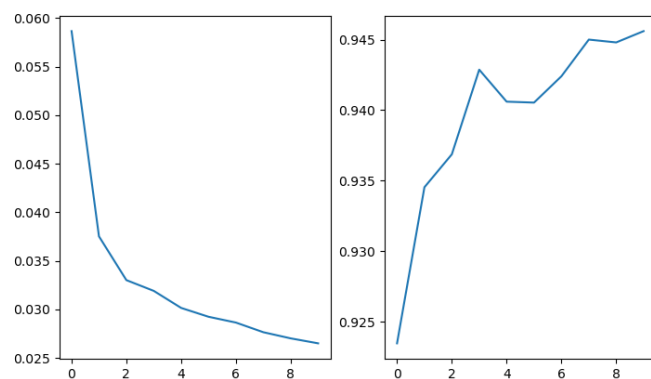


图 5: OVO loss 与 acc 变化

## 1. OVA 策略

```
1 import numpy as np
2 from array import array as pyarray
3 import struct
4 import matplotlib.pyplot as plt
5 from tqdm import tqdm
6 from sklearn.model_selection import train_test_split
7
8
9 def load_mnist(image_path, label_path):
10     flabel = open(label_path, 'rb')
11     magic_nr, size = struct.unpack('>II', flabel.read(8))
12     label = pyarray('B', flabel.read())
13     flabel.close()
14
```

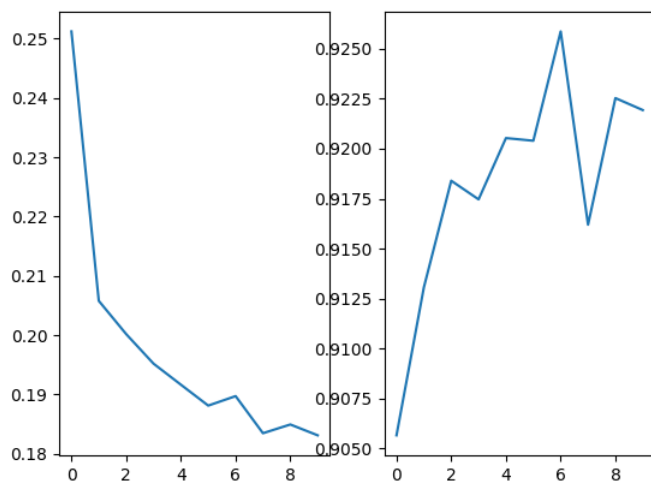


图 6: 交叉熵 loss 与 acc 变化

```

15  fimg = open(image_path, 'rb')
16  magic_nr, size, rows, cols = struct.unpack('>IIII', fimg.read(16))
17  img = pyarray('B', fimg.read())
18  fimg.close()
19
20  digits = np.arange(10)
21  ind = [k for k in range(size) if label[k] in digits]
22  N = len(ind)
23
24  images = np.zeros((N, rows * cols), dtype=np.uint8)
25  labels = np.zeros((N, 1), dtype=np.int8)
26
27  for i in range(N):
28      images[i] = np.array(img[ind[i] * rows * cols:(ind[i] + 1) * rows *
29      cols]).reshape((1, rows * cols))
29      labels[i] = label[ind[i]]
30  return images, labels
31
32
33  class LG:
34      def __init__(self, n, class_, lr=1e-2):
35          self.w = np.zeros(n + 1)
36          self.class_ = class_
37          self.lr = lr
38
39      def sigmoid(self, x):
40          return 1 / (1 + np.exp(-x))
41
42      def train(self, x, y):
43          y = 1 if y == self.class_ else -1

```

```
44     x = np.append(x, 1)
45     g = self.sigmoid(-y * self.w.dot(x)) * (-y * x)
46     self.w = self.w - self.lr * g
47     return self.loss(x, y) # ,np.linalg.norm(g,2)
48
49     def loss(self, x, y):
50         return np.log(1 + np.exp(-y * self.w.dot(x)))
51
52     def predict(self, x):
53         x = np.append(x, 1)
54         return self.sigmoid(self.w.dot(x))
55
56
57 def train_one_epoch(x, y, classifiers):
58     total_loss = 0
59     for i in range(len(x)):
60         img = x[i] / 255.0
61         label = y[i]
62         loss = 0
63         for classifier in classifiers:
64             loss += classifier.train(img, label)
65         total_loss += loss / num_classes
66         if i % 10000 == 0:
67             print(f'loss:{total_loss / (i + 1)}')
68     return total_loss / len(x)
69
70
71 def acc(x, y, classifiers):
72     cnt = 0
73     for i in range(len(x)):
74         img = x[i] / 255.0
75         label = y[i]
76         predict = []
77         for classifier in classifiers:
78             predict.append(classifier.predict(img))
79         if label == np.argmax(np.array(predict)):
80             cnt += 1
81     return cnt / len(x)
82
83
84 def plot_loss_acc(history, acc):
85     ax1 = plt.subplot(1, 2, 1)
86     plt.plot(range(len(history)), history)
87     ax2 = plt.subplot(1, 2, 2)
88     plt.plot(range(len(acc)), acc)
89     plt.show()
90
91
92 if __name__ == '__main__':
93     train_image_path = 'MNIST/train-images.idx3-ubyte'
```

```
94 train_label_path = 'MNIST/train-labels.idx1-ubyte'
95 num_classes = 10
96 images, labels = load_mnist(image_path=train_image_path, label_path=train_label_path)
97 num_features = images.shape[1]
98 lr = 1e-3
99 epoch = 10
100
101 history = []
102 classifiers = [LG(n=num_features, class_=i) for i in range(num_classes)]
103 acc_list = []
104 for i in range(epoch):
105     print(f'epoch {i + 1} ', end='')
106     xtrain, xtest, ytrain, ytest = train_test_split(images, labels)
107     history.append(train_one_epoch(xtrain, ytrain, classifiers))
108     accuracy = acc(xtest, ytest, classifiers)
109     acc_list.append(accuracy)
110     print(f'accuracy {accuracy} ', end='')
111 plot_loss_acc(history, acc_list)
```

## 2. OVO 策略

```
1 import numpy as np
2 from array import array as pyarray
3 import struct
4 import matplotlib.pyplot as plt
5 from tqdm import tqdm
6 from sklearn.model_selection import train_test_split
7 import seaborn as sns
8
9
10 def load_mnist(image_path, label_path):
11     flabel = open(label_path, 'rb')
12     magic_nr, size = struct.unpack('>II', flabel.read(8))
13     label = pyarray('B', flabel.read())
14     flabel.close()
15
16     fimg = open(image_path, 'rb')
17     magic_nr, size, rows, cols = struct.unpack('>IIII', fimg.read(16))
18     img = pyarray('B', fimg.read())
19     fimg.close()
20
21     digits = np.arange(10)
22     ind = [k for k in range(size) if label[k] in digits]
23     N = len(ind)
24
25     images = np.zeros((N, rows * cols), dtype=np.uint8)
26     labels = np.zeros((N, 1), dtype=np.int8)
27
```

```
28     for i in range(N):
29         images[i] = np.array(img[ind[i] * rows * cols:(ind[i] + 1) * rows *
30                                cols]).reshape((1, rows * cols))
31         labels[i] = label[ind[i]]
32     return images, labels
33
34 class LG:
35     def __init__(self, n, class1, class2, lr=1e-2):
36         self.w = np.zeros(n + 1)
37         self.class1 = class1
38         self.class2 = class2
39         self.lr = lr
40
41     def sigmoid(self, x):
42         return 1 / (1 + np.exp(-x))
43
44     def train(self, x, y):
45         y = 1 if y == self.class1 else -1
46         x = np.append(x, 1)
47         g = self.sigmoid(-y * self.w.dot(x)) * (-y * x)
48         self.w = self.w - self.lr * g
49         return self.loss(x, y) # ,np.linalg.norm(g,2)
50
51     def loss(self, x, y):
52         return np.log(1 + np.exp(-y * self.w.dot(x)))
53
54     def predict(self, x):
55         x = np.append(x, 1)
56         return self.sigmoid(self.w.dot(x))
57
58
59 def train_one_epoch(x, y, classifiers):
60     total_loss = 0
61     for i in range(len(x)):
62         img = x[i] / 255.0
63         label = y[i]
64         loss = 0
65         for classifier in classifiers:
66             if label == classifier.class1 or label == classifier.class2:
67                 loss += classifier.train(img, label)
68         total_loss += loss / (num_classes - 1)
69         if i % 10000 == 0:
70             print(f'loss:{total_loss / (i + 1)}')
71     return total_loss / len(x)
72
73
74 def acc(x, y, classifiers):
75     cnt = 0
76     for i in range(len(x)):
```



```
77     img = x[i] / 255.0
78     label = y[i]
79     predicts = []
80     for classifier in classifiers:
81         predict = classifier.predict(img)
82         if predict > 0.5:
83             predicts.append(classifier.class1)
84         else:
85             predicts.append(classifier.class2)
86     predict = np.argmax(np.bincount(predicts))
87     if label == predict:
88         cnt += 1
89     return cnt / len(x)
90
91
92 def plot_loss_acc(history, acc):
93     ax1 = plt.subplot(1, 2, 1)
94     plt.plot(range(len(history)), history)
95     ax2 = plt.subplot(1, 2, 2)
96     plt.plot(range(len(acc)), acc)
97     plt.show()
98
99
100 if __name__ == '__main__':
101     train_image_path = 'MNIST/train-images.idx3-ubyte'
102     train_label_path = 'MNIST/train-labels.idx1-ubyte'
103     num_classes = 10
104     images, labels = load_mnist(image_path=train_image_path, label_path=train_label_path)
105     num_features = images.shape[1]
106     lr = 1e-3
107     epoch = 10
108
109     history = []
110     classifiers = []
111     for i in range(num_classes - 1):
112         for j in range(i + 1, num_classes):
113             classifiers.append(LG(n=num_features, class1=i, class2=j))
114     print(len(classifiers))
115
116     acc_list = []
117     for i in range(epoch):
118         print(f'epoch {i + 1} ', end='')
119         xtrain, xtest, ytrain, ytest = train_test_split(images, labels)
120         history.append(train_one_epoch(xtrain, ytrain, classifiers))
121         accuracy = acc(xtest, ytest, classifiers)
122         acc_list.append(accuracy)
123         print(f'accuracy {accuracy} ', end='')
124
125     plot_loss_acc(history, acc_list)
```

### 3. 交叉熵

```
1 import numpy as np
2 from array import array as pyarray
3 import struct
4 import matplotlib.pyplot as plt
5 from tqdm import tqdm
6 from sklearn.model_selection import train_test_split
7 import seaborn as sns
8
9
10 def load_mnist(image_path, label_path):
11     flabel = open(label_path, 'rb')
12     magic_nr, size = struct.unpack('>II', flabel.read(8))
13     label = pyarray('B', flabel.read())
14     flabel.close()
15
16     fimg = open(image_path, 'rb')
17     magic_nr, size, rows, cols = struct.unpack('>IIII', fimg.read(16))
18     img = pyarray('B', fimg.read())
19     fimg.close()
20
21     digits = np.arange(10)
22     ind = [k for k in range(size) if label[k] in digits]
23     N = len(ind)
24
25     images = np.zeros((N, rows * cols), dtype=np.uint8)
26     labels = np.zeros((N, 1), dtype=np.int8)
27
28     for i in range(N):
29         images[i] = np.array(img[ind[i] * rows * cols:(ind[i] + 1) * rows *
30             cols]).reshape((1, rows * cols))
31         labels[i] = label[ind[i]]
32     return images, labels
33
34 class LG:
35     def __init__(self, n, num_classes, lr=1e-2):
36         self.w = np.ones(shape=(n + 1, num_classes))
37         self.lr = lr
38         self.num_classes = num_classes
39
40     def train(self, x, y):
41         y = self.one_hot(y)
42         x = np.append(x, 1)
43         pi = np.exp(x.dot(self.w))
44         pi = pi / np.sum(pi)
45         g = np.outer(x, (pi - y))
46         self.w = self.w - self.lr * g
```

```
47         return self.loss(x, y)
48
49     def loss(self, x, y):
50         pi = np.exp(x.dot(self.w))
51         pi = pi / np.sum(pi)
52         return -np.sum(y * np.log(pi))
53
54     def predict(self, x):
55         x = np.append(x, 1)
56         pi = np.exp(x.dot(self.w))
57         pi = pi / np.sum(pi)
58         res = np.argmax(pi)
59         return res
60
61     def one_hot(self, y):
62         y_ = np.zeros(self.num_classes)
63         y_[y] = 1
64         return y_
65
66
67     def train_one_epoch(x, y, classifier):
68         total_loss = 0
69         for i in range(len(x)):
70             img = x[i] / 255.0
71             label = y[i]
72             loss = classifier.train(img, label)
73             total_loss += loss
74             if i % 10000 == 0:
75                 print(f'loss:{total_loss / (i + 1)}')
76         return total_loss / len(x)
77
78
79     def acc(x, y, classifier):
80         cnt = 0
81         for i in range(len(x)):
82             img = x[i] / 255.0
83             label = y[i]
84             if label == classifier.predict(img):
85                 cnt += 1
86         return cnt / len(x)
87
88
89     def plot_loss_acc(history, acc):
90         ax1 = plt.subplot(1, 2, 1)
91         plt.plot(range(len(history)), history)
92         ax2 = plt.subplot(1, 2, 2)
93         plt.plot(range(len(acc)), acc)
94         plt.show()
95
96
```

```
97 if __name__ == '__main__':
98     train_image_path = 'MNIST/train-images.idx3-ubyte'
99     train_label_path = 'MNIST/train-labels.idx1-ubyte'
100     num_classes = 10
101     images, labels = load_mnist(image_path=train_image_path, label_path=train_label_path)
102     num_features = images.shape[1]
103     lr = 1e-2
104     epoch = 10
105
106     history = []
107     classifier = LG(n=num_features, num_classes=num_classes, lr=lr)
108     acc_list = []
109     for i in range(epoch):
110         print(f'epoch {i + 1} ')
111         xtrain, xtest, ytrain, ytest = train_test_split(images, labels)
112         history.append(train_one_epoch(xtrain, ytrain, classifier))
113         accuracy = acc(xtest, ytest, classifier)
114         acc_list.append(accuracy)
115         print(f'accuracy {accuracy} ', end='')
116         print('')
117
118     plot_loss_acc(history, acc_list)
```