机器学习 **Lecture 3**

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。

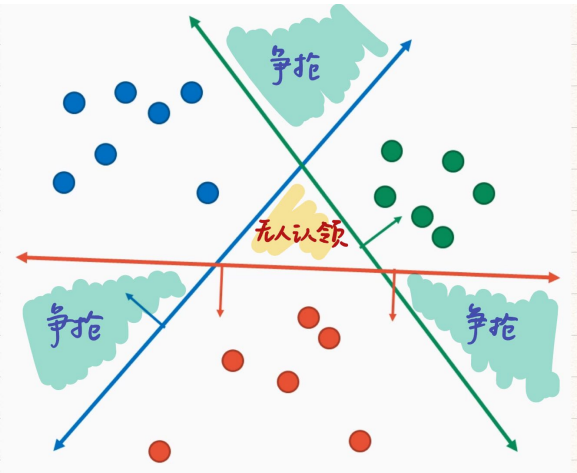

图 1: OVA 策略

2. 软间隔

同样是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。

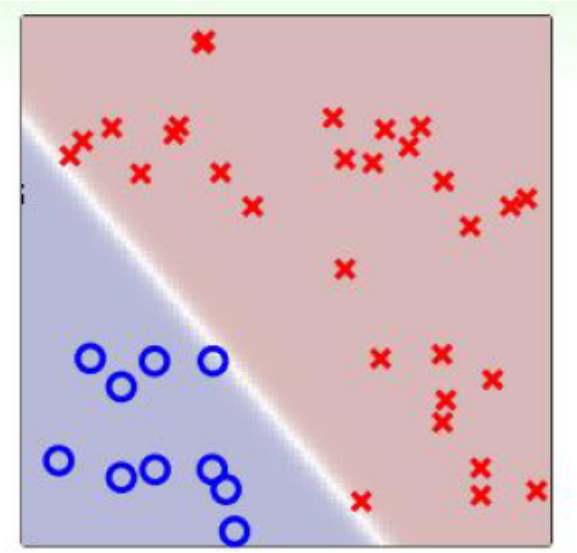

图 2: OVA 策略

Algorithm 1 Logistic Regression with SGD (OVA)

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

生成 num_classes 个 classifier
repeat
    随机在 $[1, m]$ 中选取一个 idx
    for classifier **in** classifiers **do**
        compute $g = \sigma(-y_i \mathbf{w}^\top \mathbf{x}_i)(-y_i \mathbf{x}_i)$
        update $\mathbf{w} \leftarrow \mathbf{w} - \alpha_t g$
    end for
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

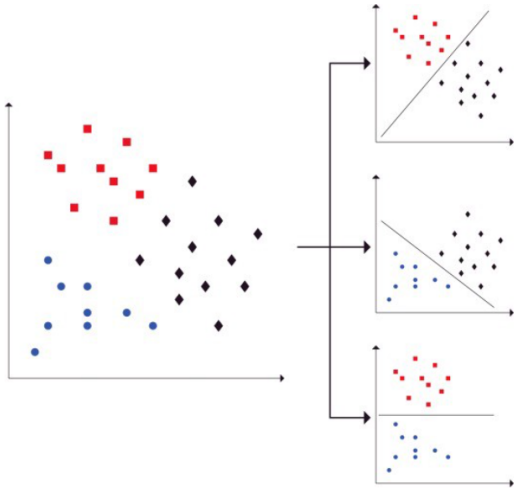

图 3: OVO 策略

Defination 1 (softmax)**.**

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

Defination 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 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

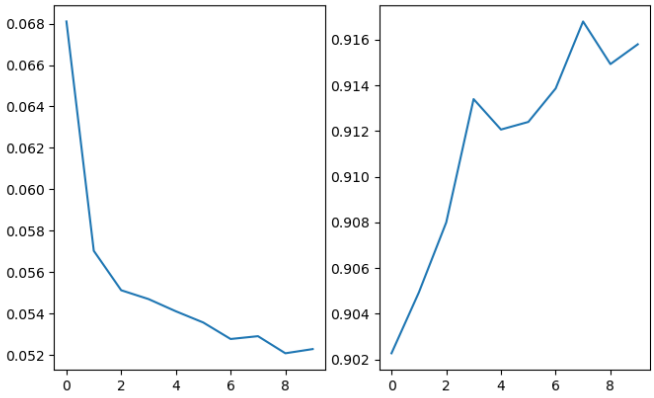

图 4: OVA loss 与 acc 变化

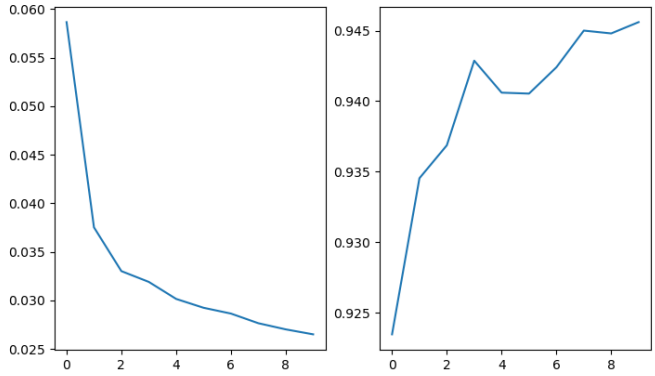

图 5: OVO loss 与 acc 变化

1. OVA 策略

```
import numpy as np
from array import array as pyarray
import struct
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.model_selection import train_test_split


def load_mnist(image_path, label_path):
    flabel = open(label_path, 'rb')
    magic_nr, size = struct.unpack('>II', flabel.read(8))
    label = pyarray('B', flabel.read())
    flabel.close()

```

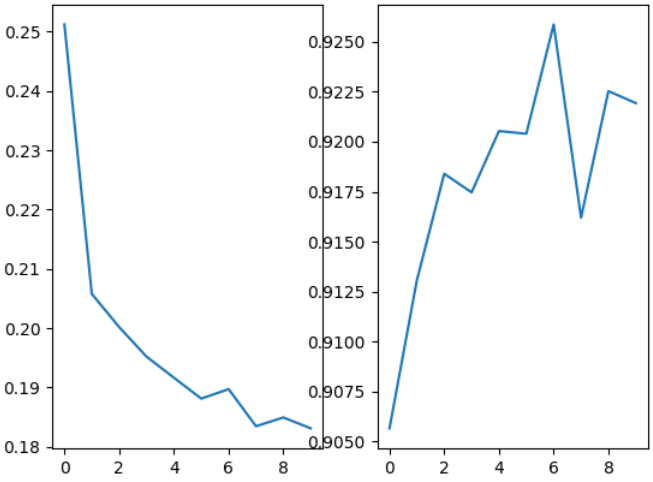

图 6: 交叉熵 loss 与 acc 变化

```
    fimg = open(image_path, 'rb')
    magic_nr, size, rows, cols = struct.unpack('>IIII', fimg.read(16))
    img = pyarray('B', fimg.read())
    fimg.close()

    digits = np.arange(10)
    ind = [k for k in range(size) if label[k] in digits]
    N = len(ind)

    images = np.zeros((N, rows * cols), dtype=np.uint8)
    labels = np.zeros((N, 1), dtype=np.int8)

    for i in range(N):
        images[i] = np.array(img[ind[i] * rows * cols:((ind[i] + 1) * rows *
            cols)]).reshape((1, rows * cols))
        labels[i] = label[ind[i]]
    return images, labels


class LG:
    def __init__(self, n, class_, lr=1e-2):
        self.w = np.zeros(n + 1)
        self.class_ = class_
        self.lr = lr

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def train(self, x, y):
        y = 1 if y == self.class_ else -1
```

```
        x = np.append(x, 1)
        g = self.sigmoid(-y * self.w.dot(x)) * (-y * x)
        self.w = self.w - self.lr * g
        return self.loss(x, y) # ,np.linalg.norm(g,2)

    def loss(self, x, y):
        return np.log(1 + np.exp(-y * self.w.dot(x)))

    def predict(self, x):
        x = np.append(x, 1)
        return self.sigmoid(self.w.dot(x))


def train_one_epoch(x, y, classifiers):
    total_loss = 0
    for i in range(len(x)):
        img = x[i] / 255.0
        label = y[i]
        loss = 0
        for classifier in classifiers:
            loss += classifier.train(img, label)
        total_loss += loss / num_classes
        if i % 10000 == 0:
            print(f'loss:{total_loss / (i + 1)}')
    return total_loss / len(x)


def acc(x, y, classifiers):
    cnt = 0
    for i in range(len(x)):
        img = x[i] / 255.0
        label = y[i]
        predict = []
        for classifier in classifiers:
            predict.append(classifier.predict(img))
        if label == np.argmax(np.array(predict)):
            cnt += 1
    return cnt / len(x)


def plot_loss_acc(history, acc):
    ax1 = plt.subplot(1, 2, 1)
    plt.plot(range(len(history)), history)
    ax2 = plt.subplot(1, 2, 2)
    plt.plot(range(len(acc)), acc)
    plt.show()


if __name__ == '__main__':
    train_image_path = 'MNIST/train-images.idx3-ubyte'
```

```
    train_label_path = 'MNIST/train-labels.idx1-ubyte'
    num_classes = 10
    images, labels = load_mnist(image_path=train_image_path, label_path=train_label_path)
    num_features = images.shape[1]
    lr = 1e-3
    epoch = 10

    history = []
    classifiers = [LG(n=num_features, class_=i) for i in range(num_classes)]
    acc_list = []
    for i in range(epoch):
        print(f'epoch {i + 1} ', end='')
        xtrain, xtest, ytrain, ytest = train_test_split(images, labels)
        history.append(train_one_epoch(xtrain, ytrain, classifiers))
        accuracy = acc(xtest, ytest, classifiers)
        acc_list.append(accuracy)
        print(f'accuracy {accuracy} ', end='')
    plot_loss_acc(history, acc_list)
```

2. OVO 策略

```
import numpy as np
from array import array as pyarray
import struct
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import seaborn as sns


def load_mnist(image_path, label_path):
    flabel = open(label_path, 'rb')
    magic_nr, size = struct.unpack('>II', flabel.read(8))
    label = pyarray('B', flabel.read())
    flabel.close()

    fimg = open(image_path, 'rb')
    magic_nr, size, rows, cols = struct.unpack('>IIII', fimg.read(16))
    img = pyarray('B', fimg.read())
    fimg.close()

    digits = np.arange(10)
    ind = [k for k in range(size) if label[k] in digits]
    N = len(ind)

    images = np.zeros((N, rows * cols), dtype=np.uint8)
    labels = np.zeros((N, 1), dtype=np.int8)

```

```
    for i in range(N):
        images[i] = np.array(img[ind[i] * rows * cols:((ind[i] + 1) * rows *
            cols)]).reshape((1, rows * cols))
        labels[i] = label[ind[i]]
    return images, labels


class LG:
    def __init__(self, n, class1, class2, lr=1e-2):
        self.w = np.zeros(n + 1)
        self.class1 = class1
        self.class2 = class2
        self.lr = lr

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def train(self, x, y):
        y = 1 if y == self.class1 else -1
        x = np.append(x, 1)
        g = self.sigmoid(-y * self.w.dot(x)) * (-y * x)
        self.w = self.w - self.lr * g
        return self.loss(x, y) # ,np.linalg.norm(g,2)

    def loss(self, x, y):
        return np.log(1 + np.exp(-y * self.w.dot(x)))

    def predict(self, x):
        x = np.append(x, 1)
        return self.sigmoid(self.w.dot(x))


def train_one_epoch(x, y, classifiers):
    total_loss = 0
    for i in range(len(x)):
        img = x[i] / 255.0
        label = y[i]
        loss = 0
        for classifier in classifiers:
            if label == classifier.class1 or label == classifier.class2:
                loss += classifier.train(img, label)
        total_loss += loss / (num_classes - 1)
        if i % 10000 == 0:
            print(f'loss:{total_loss / (i + 1)}')
    return total_loss / len(x)


def acc(x, y, classifiers):
    cnt = 0
    for i in range(len(x)):
```

```
        img = x[i] / 255.0
        label = y[i]
        predicts = []
        for classifier in classifiers:
            predict = classifier.predict(img)
            if predict > 0.5:
                predicts.append(classifier.class1)
            else:
                predicts.append(classifier.class2)
        predict = np.argmax(np.bincount(predicts))
        if label == predict:
            cnt += 1
    return cnt / len(x)


def plot_loss_acc(history, acc):
    ax1 = plt.subplot(1, 2, 1)
    plt.plot(range(len(history)), history)
    ax2 = plt.subplot(1, 2, 2)
    plt.plot(range(len(acc)), acc)
    plt.show()


if __name__ == '__main__':
    train_image_path = 'MNIST/train-images.idx3-ubyte'
    train_label_path = 'MNIST/train-labels.idx1-ubyte'
    num_classes = 10
    images, labels = load_mnist(image_path=train_image_path, label_path=train_label_path)
    num_features = images.shape[1]
    lr = 1e-3
    epoch = 10

    history = []
    classifiers = []
    for i in range(num_classes - 1):
        for j in range(i + 1, num_classes):
            classifiers.append(LG(n=num_features, class1=i, class2=j))
    print(len(classifiers))

    acc_list = []
    for i in range(epoch):
        print(f'epoch {i + 1} ', end='')
        xtrain, xtest, ytrain, ytest = train_test_split(images, labels)
        history.append(train_one_epoch(xtrain, ytrain, classifiers))
        accuracy = acc(xtest, ytest, classifiers)
        acc_list.append(accuracy)
        print(f'accuracy {accuracy} ', end='')

    plot_loss_acc(history,acc_list)
```

3. 交叉熵

```python
import numpy as np
from array import array as pyarray
import struct
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import seaborn as sns


def load_mnist(image_path, label_path):
    flabel = open(label_path, 'rb')
    magic_nr, size = struct.unpack('>II', flabel.read(8))
    label = pyarray('B', flabel.read())
    flabel.close()

    fimg = open(image_path, 'rb')
    magic_nr, size, rows, cols = struct.unpack('>IIII', fimg.read(16))
    img = pyarray('B', fimg.read())
    fimg.close()

    digits = np.arange(10)
    ind = [k for k in range(size) if label[k] in digits]
    N = len(ind)

    images = np.zeros((N, rows * cols), dtype=np.uint8)
    labels = np.zeros((N, 1), dtype=np.int8)

    for i in range(N):
        images[i] = np.array(img[ind[i] * rows * cols:((ind[i] + 1) * rows *
            cols)]).reshape((1, rows * cols))
        labels[i] = label[ind[i]]
    return images, labels


class LG:
    def __init__(self, n, num_classes, lr=1e-2):
        self.w = np.ones(shape=(n + 1, num_classes))
        self.lr = lr
        self.num_classes = num_classes

    def train(self, x, y):
        y = self.one_hot(y)
        x = np.append(x, 1)
        pi = np.exp(x.dot(self.w))
        pi = pi / np.sum(pi)
        g = np.outer(x, (pi - y))
        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
```

```python
if __name__ == '__main__':
    train_image_path = 'MNIST/train-images.idx3-ubyte'
    train_label_path = 'MNIST/train-labels.idx1-ubyte'
    num_classes = 10
    images, labels = load_mnist(image_path=train_image_path, label_path=train_label_path)
    num_features = images.shape[1]
    lr = 1e-2
    epoch = 10

    history = []
    classifier = LG(n=num_features, num_classes=num_classes, lr=lr)
    acc_list = []
    for i in range(epoch):
        print(f'epoch {i + 1} ')
        xtrain, xtest, ytrain, ytest = train_test_split(images, labels)
        history.append(train_one_epoch(xtrain, ytrain, classifier))
        accuracy = acc(xtest, ytest, classifier)
        acc_list.append(accuracy)
        print(f'accuracy {accuracy} ', end='')
        print('')

    plot_loss_acc(history,acc_list)
```