模式识别课后作业三

2020E8013282019 李一帆

November 5, 2020

1 计算与证明

1.1 问题 1

将四个训练样本表示为规范化增广样本有：$(1,4,1)^T,(2,3,1)^T,(-4,-1,-1)^T,(-3,-2,-1)^T$。分别记作 x_1,x_2,x_3,x_4。根据批处理感知器算法，由于 $a^T\cdot x_1>0,a^T\cdot x_2>0,a^T\cdot x_3<0,a^T\cdot x_4<0$，因此第一次迭代可以表示为：

$$\begin{aligned}
a &:= a + 1\cdot\sum_{i=3}^{4} x_i \\
&= \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} -4 \\ -1 \\ -1 \end{pmatrix} + \begin{pmatrix} -3 \\ -2 \\ -1 \end{pmatrix} \\
&= \begin{pmatrix} -7 \\ -2 \\ -2 \end{pmatrix}
\end{aligned}$$

同理，由于 $a^T\cdot x_1<0,a^T\cdot x_2<0,a^T\cdot x_3>0,a^T\cdot x_4>0$，因此第二次迭代可以表示为如下形式。

$$\begin{aligned}
a &:= a + 1\cdot\sum_{i=1}^{2} x_i \\
&= \begin{pmatrix} -7 \\ -2 \\ -2 \end{pmatrix} + \begin{pmatrix} 1 \\ 4 \\ 1 \end{pmatrix} + \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} -4 \\ 5 \\ 0 \end{pmatrix}
\end{aligned}$$

因为此时 $a^T\cdot x_1>0,a^T\cdot x_2>0,a^T\cdot x_3>0,a^T\cdot x_4>0$，满足递归结束条件。因此，最终得到的权向量 $a=(-4,5,0)^T$。

1.2 问题 2

根据决策规则如果 $g_i(\boldsymbol{x}) > g_j(\boldsymbol{x})$，则 $\boldsymbol{x}$ 被划分为 ω_i 类别。

因此当 $\begin{cases} g_1(x) > g_2(x) \\ g_1(x) > g_3(x) \end{cases} \Rightarrow \begin{cases} x_1 < \frac{1}{2} \\ x_2 > \frac{x_1}{2} \end{cases}$ 时，属于 ω_1；

当 $\begin{cases} g_2(x) > g_1(x) \\ g_2(x) > g_3(x) \end{cases} \Rightarrow \begin{cases} x_1 > \frac{1}{2} \\ x_2 > \frac{1-x_1}{2} \end{cases}$ 时，属于 ω_2；

当 $\begin{cases} g_3(x) > g_1(x) \\ g_3(x) > g_2(x) \end{cases} \Rightarrow \begin{cases} x_2 < \frac{x_1}{2} \\ x_2 < \frac{1-x_1}{2} \end{cases}$ 时，属于 ω_3。

根据上述关系可以画出决策面如图 Figure 1 所示。

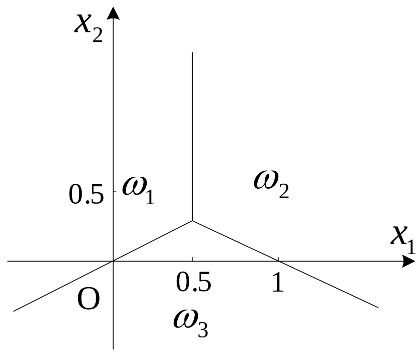

Figure 1: 决策面.

由 Figure 1 可以看出，由于所有区域都被划分成了三类中的某一类，不存在未被划分的区域，也不存在二义性区域。因此，此时不存在分类不确定区域。

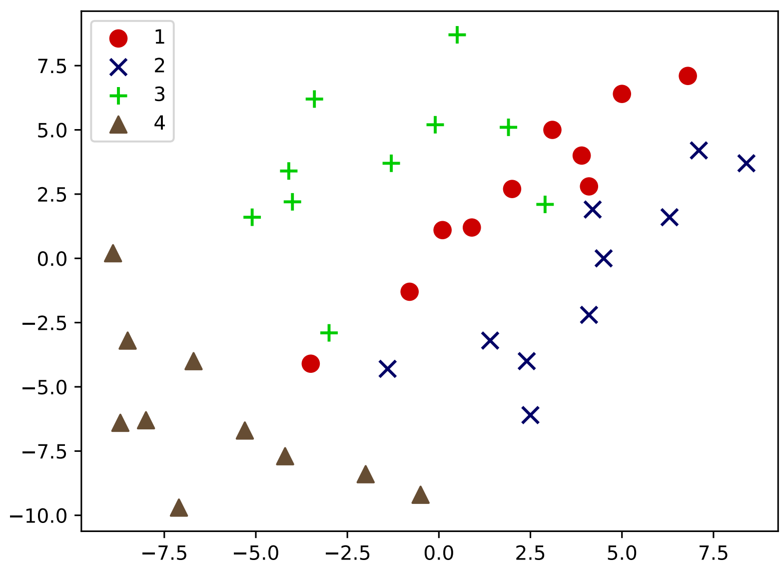

Figure 2: 数据可视化.

2 计算机编程

将数据进行可视化，如 Figure 2 所示。

2.1 问题 1

Batch Perception 的算法和 Python 代码如下所示。

Algorithm 1: Batch Perceptron

begin initialize: a, η, certain θ(small value), $k = 0$;
while $|\eta_k \sum y| < \theta, y \in Y_k$ **do**
  $k \leftarrow k + 1$;
  $a = a + \eta_k \sum_{y \in Y(k)} y$;
  return a;

```python
def batch_perceptron(lr=0.5, data=None):
    a = np.zeros([3, 1])
    threshold = 1e-9
    data[:, 2] = 1
    data[10:, :] = -data[10:, :]
    iter = 0
    for i in range(1000000):
        result = a.T.dot(data.T)
        _, inx = np.where(result <= 0)
        increments = np.sum(lr * data[inx], axis=0)
        a += increments.reshape([3, 1])
        iter += 1
        if abs(increments.sum()) < threshold:
            print('total iter:%d' % iter)
            return a
```

2.1.1 问题 a

对 ω_1 和 ω_2 的分类将结果如 Figure 3 所示。

通过运行上述算法，可以得到对 ω_1 和 ω_2 的迭代次数为 24。

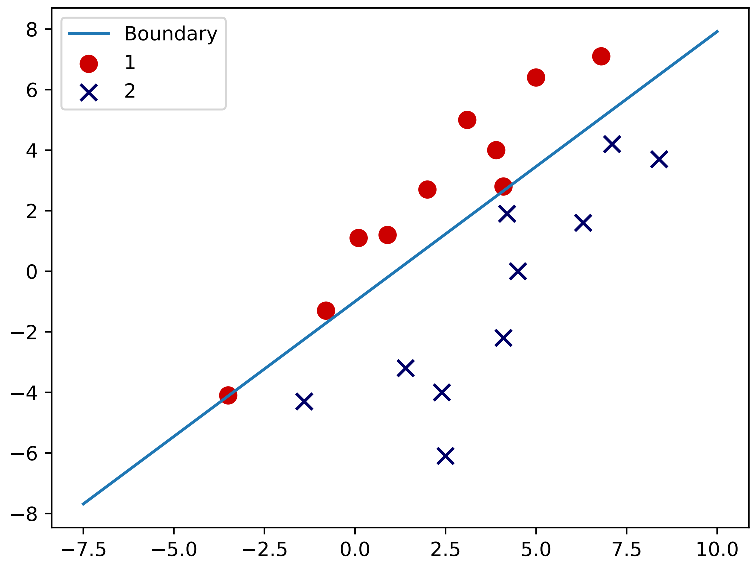

Figure 3: Batch perceptron 分类 ω_1 和 ω_2.

2.1.2 问题 b

对 ω_2 和 ω_3 的分类将结果如 Figure 4 所示，迭代次数为 17 次。

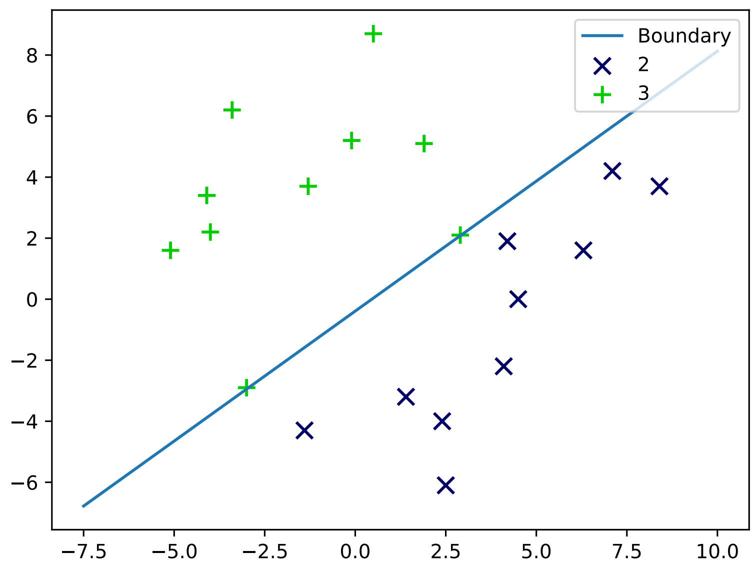

Figure 4: Batch perceptron 分类 ω_2 和 ω_3.

2.2 问题 2

Ho-Kashyap 的算法以及 Python 代码如下所示。

Algorithm 2: Ho-Kashyap

begin initialize: $a, b, \eta_0 < 1$, threshold $b_{min, k_{max}}$, $k = 0$;
while *True* **do**
    $k \leftarrow k + 1$;
    $a = a + \eta_k \sum_{y \in Y(k)} y$;
    $e \leftarrow Ya - b$;
    $e^+ \leftarrow 1/2(e + abs(e))$;
    $b \leftarrow b + 2\eta_k e^+$;
    $a = Y^+ b$;
    if $(e \leq b_{\text{min}})$ **then**
        return $a,\ b$;
print('No solutions!');

```
def Ho_Kashyap(lr=0.01, data=None, threshold=1e-10):
    a = np.ones([3, 1])
    b = np.ones([len(data), 1])
    data[:, 2] = 1
    data[10:, :] = -data[10:, :]
    for i in range(300000):
        error = data.dot(a) - b
        error_plus = 0.5 * (error + np.abs(error))
        b = b + 2 * lr * error_plus
        a = np.linalg.pinv(data).dot(b)
        if i % 100 == 0:
            print('iter:%d, the error is %f' % (i, error.sum()))
        if abs(error).sum() < threshold:
            print(error)
            return a
    print(error)
    return a
```

对 ω_1 和 ω_3 的分类结果如 Figure 5 所示。最终的 error 为:$[[-7.55684779e-01], [-1.50440675e-02], [-7.65558887e-01], [-5.17913983e-01], [5.32907052e-15], [-7.55691123e-01], [-4.61518484e-01], [-4.98780350e-01], [-4.63259450e-01], [-1.98151872e-01], [-1.19612264e+00], [5.32907052e-15], [-2.06583287e+00], [-1.54708533e-01], [5.55111512e-15], [-8.43128071e-02], [1.06581410e-14], [5.55111512e-15], [5.55111512e-15], [-9.30626139e-01]]$。由 Figure 5 所示，该样本是线性不可分的，由误差可以看出其中存在小于 0 的元素，因此也可以说明原样本线性不可分。

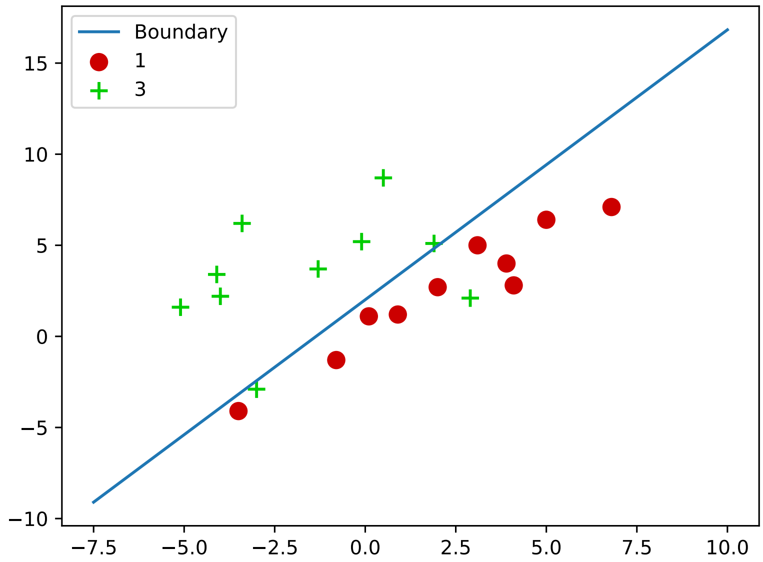

Figure 5: 利用 Ho-Kashyap 分类 ω_1 和 ω_3.

对 ω_2 和 ω_4 的分类将结果如 Figure 6 所示。最终的 error 为 $[[1.73194792e{-}12], [-4.66697792e{-}11], [1.12265752e-12], [1.41220369e-12], [1.25854882e-12], [5.65769653e-13], [5.87530025e-13], [4.10782519e{-}13], [1.79589676e{-}12], [8.99724739e{-}13], [1.83852933e{-}13], [-1.44970702e{-}11], [3.21520588e{-}13], [3.21520588e{-}13], [2.33146835e{-}13], [-2.57203148e{-}11], [3.42836870e{-}13], [6.11066753e-13], [7.58504370e-13], [5.54223334e-13]]$。由图 Figure 6 可以看出，该样本是线性可分的。并且 error 也全部为零（近似），因此也可以说明原样本线性可分。

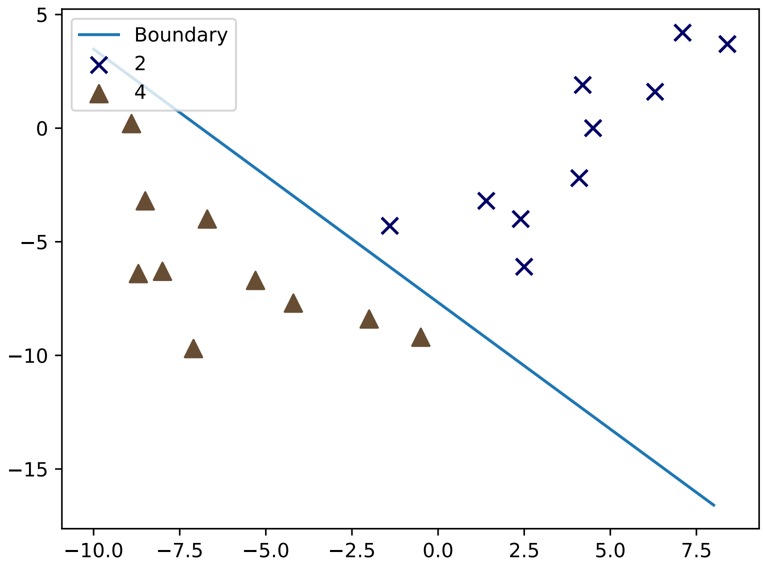

Figure 6: 利用 Ho-Kashyap 分类 ω_2 和 ω_4.

2.3 问题 3

MSE 多类扩展算法和 Python 的代码如下所示。

Algorithm 3: MSE 多类扩展

1 **Objective Function**: $\min_{\mathrm{W,b}} \sum_{\mathrm{i=1}}^{\mathrm{n}} \left\| \mathrm{W^T\widehat{X}} - \mathrm{Y} \right\|_2^2$;

2 $\mathrm{W} = \begin{pmatrix} \mathrm{W} \\ \mathrm{b^T} \end{pmatrix} \in R^{(d+1)\times c}, \widehat{\mathrm{x}} = \begin{pmatrix} \mathrm{x} \\ 1 \end{pmatrix} \in R^{d+1}, \hat{X} = (\widehat{\mathrm{x}}_1, \widehat{\mathrm{x}}_2, ..., \widehat{\mathrm{x}}_n) \in R^{(d+1)\times n}$;

3 $\widehat{\mathrm{W}} = (\widehat{\mathrm{X}}\widehat{\mathrm{X}}^T + \lambda \mathrm{I})^{-1}\widehat{\mathrm{X}}\mathrm{Y}^T \in R^{(d+1)\times c}$;

```python
def multi_class_mse(epsilon=0.01, data=None, label=None):
    W = np.linalg.inv(data.dot(data.T) + epsilon).dot(data).dot(
        label.T)
    return W
```

通过利用每类前 8 个样本来构造分类器，后两个样本作测试，可以得到最终的 $\hat{W}$ 为：

$$\begin{aligned} \hat{W} = &[[0.02049053, 0.06810353, -0.0408785, -0.04773576], \\ &[0.0162581, -0.0360417, 0.05968833, -0.03991593], \\ &[0.26737315, 0.27362052, 0.25018894, 0.20848962]] \end{aligned}$$

测试的准确率为 100%。

A 附录——代码 (Python)

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


def load_data(data_path):
    data = []
    with open(data_path) as data_file:
        for line in data_file.readlines():
            line_split = line.strip('\n').split(' ')
            line_split = [float(line_split[i]) for i in range(
               len(line_split))]
            data.append(line_split)
    return np.array(data)


def batch_perception(lr=0.5, data=None):
    a = np.zeros([3, 1])
    threshold = 1e-9
    data[:, 2] = 1
    data[10:, :] = -data[10:, :]
    iter = 0
    for i in range(1000000):
        result = a.T.dot(data.T)
        _, inx = np.where(result <= 0)
        increments = np.sum(lr * data[inx], axis=0)
        a += increments.reshape([3, 1])
        iter += 1
        if abs(increments.sum()) < threshold:
            print('total iter:%d' % iter)
            return a


def Ho_Kashyap(lr=0.01, data=None, threshold=1e-10):
    a = np.ones([3, 1])
```

```
    b = np.ones([len(data), 1])
    data[:, 2] = 1
    data[10:, :] = -data[10:, :]
    for i in range(300000):
        error = data.dot(a) - b
        error_plus = 0.5 * (error + np.abs(error))
        b = b + 2 * lr * error_plus
        a = np.linalg.pinv(data).dot(b)
        if i % 100 == 0:
            print('iter:%d, the error is %f' % (i, error.sum()))
        if abs(error).sum() < threshold:
            print(error)
            return a
    print(error)
    return a


def multi_class_mse(epsilon=0.01, data=None, label=None):
    W = np.linalg.inv(data.dot(data.T) + epsilon).dot(data).dot(label.T)
    return W


if __name__ == '__main__':
    data = load_data('data.txt')
    # show the data
    plt.figure()
    plt.scatter(data[:10, 0], data[:10, 1], marker='o', label='1', color=(0.8, 0., 0.), s=70)
    plt.scatter(data[10:20, 0], data[10:20, 1], marker='x', label='2', color=(0., 0., 0.4), s=70)
    plt.scatter(data[20:30, 0], data[20:30, 1], marker='+', label='3', color=(0., 0.8, 0.), s=80)
    plt.scatter(data[30:40, 0], data[30:40, 1], marker='^', label='4', color=(0.4, 0.3, 0.2), s=70)
```

```
    plt.legend(loc='upper left')

    #####batch_perception#####
    # train data of w1 and w2
    data_temp = np.zeros([20, 3])
    data_temp[:10] = data[:10]
    data_temp[10:20] = data[10:20]
    a = batch_perception(data=data_temp)
    a = a / a[1]
    a_x = np.linspace(-7.5, 10, 10)
    a_y = -a_x * a[0] - a[2]
    plt.figure()
    plt.scatter(data[:10, 0], data[:10, 1], marker='o', label='1', color=(0.8, 0., 0.), s=70)
    plt.scatter(data[10:20, 0], data[10:20, 1], marker='x', label='2', color=(0., 0., 0.4), s=70)
    plt.plot(a_x, a_y, '-', label='Boundary')
    plt.legend(loc='upper left')

    # train data of w2 and w3
    data_temp = np.zeros([20, 3])
    data_temp[:10] = data[10:20]
    data_temp[10:20] = data[20:30]
    a = batch_perception(data=data_temp)
    a = a / a[1]
    a_x = np.linspace(-7.5, 10, 10)
    a_y = -a_x * a[0] - a[2]
    plt.figure()
    plt.scatter(data[10:20, 0], data[10:20, 1], marker='x', label='2', color=(0., 0., 0.4), s=70)
    plt.scatter(data[20:30, 0], data[20:30, 1], marker='+', label='3', color=(0., 0.8, 0.), s=80)
    plt.plot(a_x, a_y, '-', label='Boundary')
    plt.legend(loc='upper right')

    #####Ho_Kashyap#####
```

```
        # train data of w1 and w3
        data_temp = np.zeros([20, 3])
        data_temp[:10] = data[:10]
        data_temp[10:20] = data[20:30]
        a = Ho_Kashyap(data=data_temp, threshold=1)
        a = a / a[1]
        a_x = np.linspace(-7.5, 10, 10)
        a_y = - a_x * a[0] - a[2]
        plt.figure()
        plt.scatter(data[:10, 0], data[:10, 1], marker='o', label='1', color=(0.8, 0., 0.), s=70)
        plt.scatter(data[20:30, 0], data[20:30, 1], marker='+', label='3', color=(0., 0.8, 0.), s=80)
        plt.plot(a_x, a_y, '-', label='Boundary')
        plt.legend(loc='upper left')

        # train data of w2 and w4
        data_temp = np.zeros([20, 3])
        data_temp[:10] = data[10:20]
        data_temp[10:20] = data[30:40]
        a = Ho_Kashyap(data=data_temp)
        a = a / a[1]
        a_x = np.linspace(-10, 8, 10)
        a_y = - a_x * a[0] - a[2]
        plt.figure()
        plt.scatter(data[10:20, 0], data[10:20, 1], marker='x', label='2', color=(0., 0., 0.4), s=70)
        plt.scatter(data[30:40, 0], data[30:40, 1], marker='^', label='4', color=(0.4, 0.3, 0.2), s=80)
        plt.plot(a_x, a_y, '-', label='Boundary')
        plt.legend(loc='upper left')

        #####Multi-Classification######
        train_data = np.zeros([3, 32])
        train_data[:, :8] = data[:8].T
        train_data[:, 8:16] = data[10:18].T
```

```python
        train_data[:, 16:24] = data[20:28].T
        train_data[:, 24:32] = data[30:38].T
        train_data[2, :] = 1

        train_label = np.zeros([4, 32])
        train_label[0, :8] = 1
        train_label[1, 8:16] = 1
        train_label[2, 16:24] = 1
        train_label[3, 24:32] = 1

        W = multi_class_mse(data=train_data, label=train_label)

        test_data = np.zeros([3, 8])
        test_data[:, :2] = data[8:10].T
        test_data[:, 2:4] = data[18:20].T
        test_data[:, 4:6] = data[28:30].T
        test_data[:, 6:8] = data[38:40].T
        test_data[2, :] = 1

        test_label = np.zeros([4, 8])
        test_label[0, :2] = 1
        test_label[1, 2:4] = 1
        test_label[2, 4:6] = 1
        test_label[3, 6:8] = 1

        predict = W.T.dot(test_data)
        inx = np.argmax(predict, axis=0)
        cnt = 0
        for i in range(8):
            if test_label[inx[i], i] == 1:
                cnt += 1
        acc = cnt / test_label.shape[1]
        print('The accuracy is: %f' % acc)

        plt.show()
```