

人工智能Lab2实验报告

PB19030888 张舒恒

决策树

决策树建立

采用ID3决策树算法，为数据的特征添加记号label1-label9，数据的标签值添加记号value。如果当前数据集的value只有一类则直接返回value。如果当前数据集的特征值1-9完全一样则选择数据样本最多的类的value返回。选择当前数据集的一个特征计算按这个特征划分所带来的信息增益，根据最大信息增益选择最优划分特征。去除最优划分特征后递归调用生成函数来生成子结点，由此递归地进行ID3决策树建立。因为python没有指针类型，所以考虑使用字典的多层嵌套来生成中间节点，最内层字典即为叶子节点。

```
def fit(self, train_features, train_labels):
    """
    TODO: 实现决策树学习算法。
    train_features是维度为(训练样本数,属性数)的numpy数组
    train_labels是维度为(训练样本数,)的numpy数组
    """
    f = open('test.csv', 'w', encoding='utf-8', newline='')
    csv_writer = csv.writer(f)
    csv_writer.writerow(["label1", "label2", "label3", "label4", "label5",
"label6", "label7", "label8", "label9", "value"])
    for i in range(train_features.shape[0]):
        temp = list(train_features[i])
        temp.append(train_labels[i])
        csv_writer.writerow(temp)
    f.close()
    data = pd.read_csv('test.csv', engine='python')
    self.tree = set_dic_tree(data)
    print(self.tree)

def set_dic_tree(data_list):
    fea_list = []
    for fea in data_list.iloc[:, :-1].columns:
        val_list = list(pd.unique(data_list[fea]))
        fea_list.append((fea, val_list))
    fea_dic = dict(fea_list)
    labels = data_list.iloc[:, -1]
    # print(labels)
    # for i in data_list.iloc[:, :-1].columns:
    #     print(data_list[i].value_counts())
    labels_count = len(labels.value_counts())
    if labels_count == 1:
        # print(labels.values[0])
        return labels.values[0]

    sign = 1
    for item in data_list.iloc[:, :-1].columns:
        if len(data_list[item].value_counts()) != 1:
            sign = 0
    if sign:
```

```

        # print("1111111")
        return sel_label(data_list)
    best = set_feature(data_list)
    dic_tree = {best: {}}
    values = pd.unique(data_list[best])

    # print(best)
    # print(values)

    if len(values) != len(fea_dic[best]):
        # print(best)
        # print(data)
        other_vals = set(fea_dic[best]) - set(values)
        for val in other_vals:
            dic_tree[best][val] = sel_label(data_list)

    for item in del_feature(data_list, best):
        dic_tree[best][item[0]] = set_dic_tree(item[1])
    return dic_tree

```

信息熵和信息增益

通过以下公式计算节点的信息熵和特征划分的信息增益。

$$Ent(D) = - \sum_{k=1}^{|y|} p_k \log_2 p_k$$

$$Gain(D, a) = Ent(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} Ent(D^v)$$

```

def info_func(data):
    info = 0
    labels = data.iloc[:, -1]
    label_count = labels.value_counts()
    label_num = len(labels)
    for k in label_count.keys():
        temp = label_count[k] / label_num
        info += -temp * np.log(temp)
    return info

def info_gain(data, feature):
    info = info_func(data)
    feature_count = data[feature].value_counts()
    info_2 = 0
    data_num = data.shape[0]
    for key in feature_count.keys():
        key_w = feature_count[key] / data_num
        data_loc = data.loc[data[feature] == key]
        info_key = info_func(data_loc)
        info_2 += key_w * info_key
    gain = info - info_2
    return gain

```

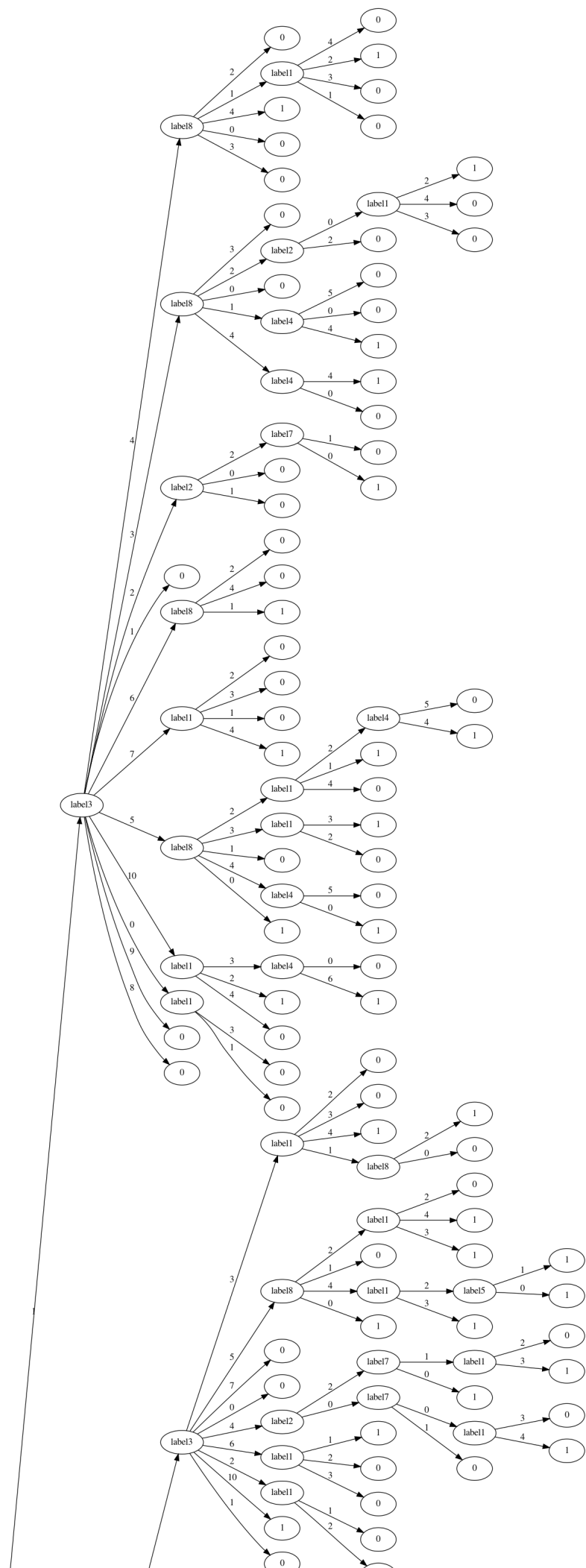
决策树预测

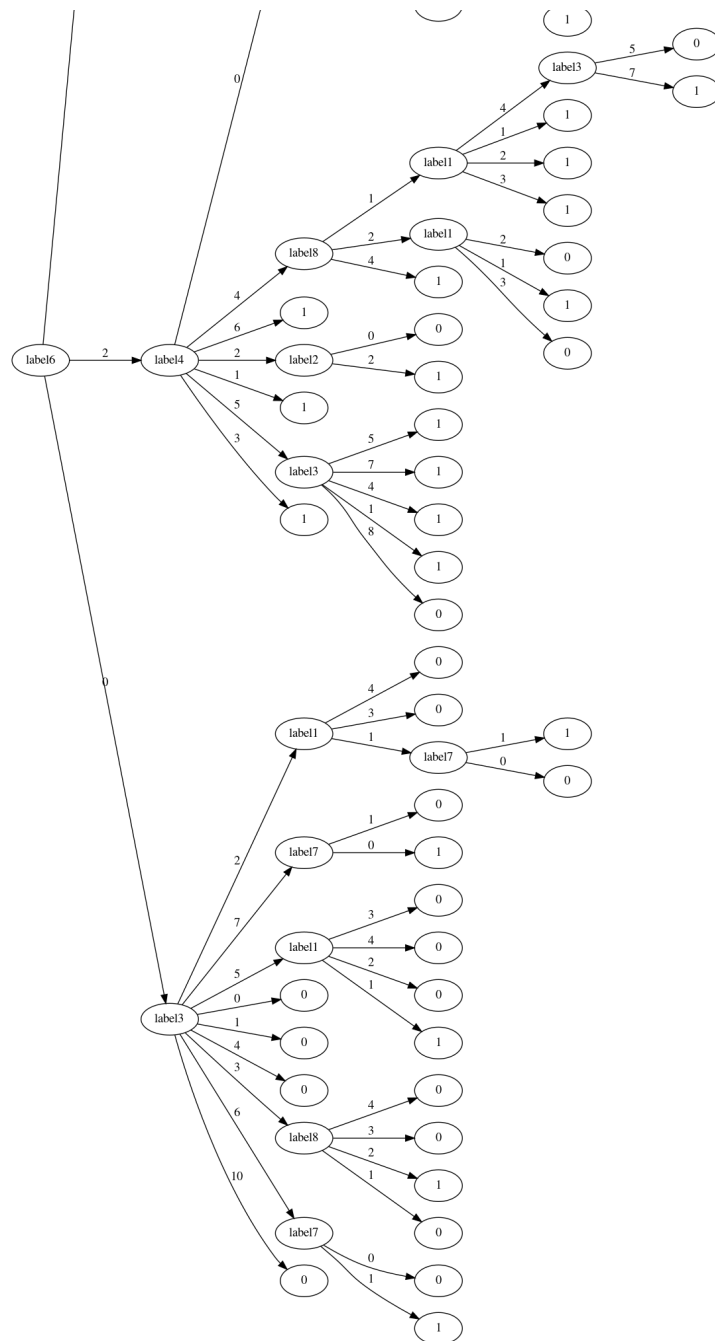
在字典树的二级字典中查找和数据特征值一样的key，判断是否是字典，若是则递归查找，否则输出value。

```
def prediction(dic_tree, test_data):  
    f_1 = list(dic_tree.keys())[0]  
    dic_2 = dic_tree[f_1]  
    in_1 = test_data[f_1]  
    in_value = dic_2[in_1]  
    if isinstance(in_value, dict):  
        return prediction(in_value, test_data)  
    else:  
        return in_value
```

模型测试

输出的决策树为：





决策树预测准确率为64.29%，在一些较为极端的测试用例时模型预测错误，这主要是训练数据不足导致模型无法学习到更多的决策策略。

```

运行: main
C:\Users\凝雨\AppData\Local\Programs\Python\Python39\python.exe C:/Users/凝雨/Desktop/人工智能/EXP2_files/src1/main.py
[1. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0.
1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 1. 1.
1. 1. 0. 0. 0. 1. 1. 0.]
[1 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 1 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
DecisionTree acc: 64.29%

进程已结束,退出代码0

```

支持向量机

算法思路

软间隔支持向量机模型算法如下(引自《机器学习》周志华):

算法 7.4 (非线性支持向量机学习算法)

输入: 训练数据集 $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, 其中 $x_i \in \mathcal{X} = \mathbf{R}^n$, $y_i \in$

$\mathcal{Y} = \{-1, +1\}$, $i = 1, 2, \dots, N$;

输出: 分类决策函数。

(1) 选取适当的核函数 $K(x, z)$ 和适当的参数 C , 构造并求解最优化问题

$$\min_{\alpha} \quad \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^N \alpha_i \quad (7.95)$$

$$\text{s.t.} \quad \sum_{i=1}^N \alpha_i y_i = 0 \quad (7.96)$$

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N \quad (7.97)$$

求得最优解 $\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*)^T$ 。

(2) 选择 α^* 的一个正分量 $0 < \alpha_j^* < C$, 计算

$$b^* = y_j - \sum_{i=1}^N \alpha_i^* y_i K(x_i, x_j)$$

(3) 构造决策函数:

$$f(x) = \text{sign} \left(\sum_{i=1}^N \alpha_i^* y_i K(x, x_i) + b^* \right) \quad \blacksquare$$

当 $K(x, z)$ 是正定核函数时, 问题 (7.95)~(7.97) 是凸二次规划问题, 解是存在的。

模型初始化

记录支持向量机的软间隔参数C, 归零参数epsilon, 核函数kernel, 支持向量拉格朗日参数sv_alpha, 支持向量数positive_num, 超平面偏移参数b, 训练特征集train_data, 训练标签集train_label

```
def __init__(self, C=1, kernel='Linear', epsilon=1e-4):
    self.C = C
    self.epsilon = epsilon
    self.kernel = kernel
    self.sv_alpha = None
    self.positive_num = None
    self.b = None
    self.train_data = None
    self.train_label = None
```

模型训练

二次凸优化求解函数 `cvxopt.solvers.qp(P,q,G,h,A,b)` 的标准形式为:

$$\begin{aligned} \min \quad & \frac{1}{2} x^T P x + q^T x \\ \text{s.t.} \quad & G x \leq h \\ & A x = b \end{aligned}$$

将软间隔支持向量机模型的优化目标与约束代入上述标准形式构造矩阵 P, q, G, h, A, b ，再统一精度类型为 `np.double`，将矩阵转换为 `cvxopt.matrix` 类型。使用二次凸优化包求解参数 `sv_alpha`，最后利用 `sv_alpha` 和核函数求解 `b`。

```
def fit(self, train_data, train_label):
    '''
    TODO: 实现软间隔SVM训练算法
    train_data: 训练数据，是(N, 7)的numpy二维数组，每一行为一个样本
    train_label: 训练数据标签，是(N,)的numpy数组，和train_data按行对应
    '''

    self.train_data = train_data
    self.train_label = train_label
    num = train_data.shape[0]
    q = -1 * np.ones((num, 1))
    q = q.astype(np.double)
    q_cvx = cvxopt.matrix(q)

    a = train_label.reshape(1, num)
    a = a.astype(np.double)
    a_cvx = cvxopt.matrix(a)

    k = np.zeros((num, num))
    for i in range(num):
        for j in range(num):
            k[i][j] = self.KERNEL(train_data[i], train_data[j]) * train_label[i]
    * train_label[j]
    k = k.astype(np.double)
    k_cvx = cvxopt.matrix(k)

    h = np.zeros((num, 1))
    for i in range(num):
        h[i] = self.c
    h = h.astype(np.double)
    h_cvx = cvxopt.matrix(h)

    b = np.zeros(1)
    b = b.astype(np.double)
    b_cvx = cvxopt.matrix(b)

    ig = np.eye(num, dtype=int)
    ig = ig.astype(np.double)
    ig_cvx = cvxopt.matrix(ig)

    x1 = cvxopt.solvers.qp(k_cvx, q_cvx, ig_cvx, h_cvx, a_cvx, b_cvx)
    self.sv_alpha = np.array(x1['x'])
    self.positive_num = np.where(self.sv_alpha > self.epsilon)[0]
    sum_b = 0
    for j in self.positive_num:
        sum_tmp = 0
        for i in range(num):
            sum_tmp += train_label[i] * self.sv_alpha[i] *
self.KERNEL(train_data[i], train_data[j])
        sum_tmp += train_label[j] - sum_tmp
    self.b = sum_b / self.positive_num.shape[0]
```

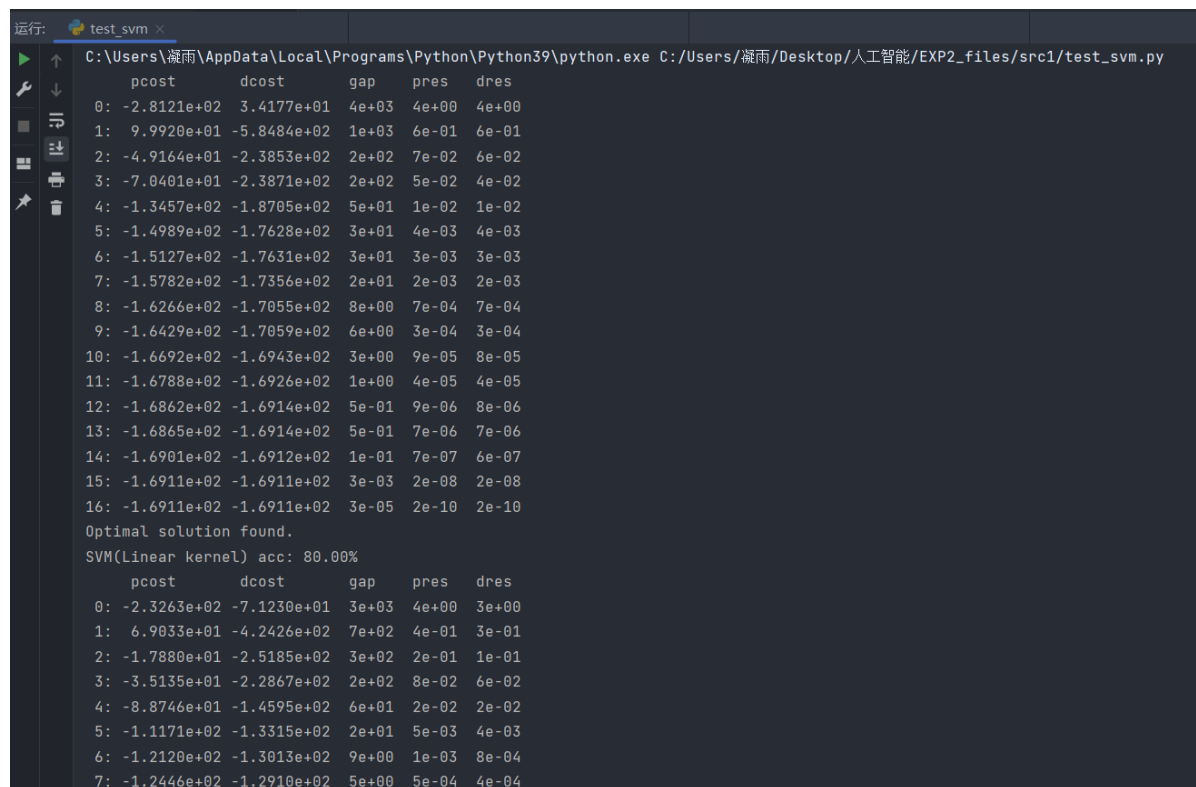
模型预测

根据上述软间隔支持向量机模型算法构造决策函数，若决策结果为正则预测结果为正例点，否则预测结果为负例点。

```
def predict(self, test_data):  
    """  
    TODO: 实现软间隔SVM预测算法  
    train_data: 测试数据，是(M, 7)的numpy二维数组，每一行为一个样本  
    必须返回一个(M,)的numpy数组，对应每个输入预测的标签，取值为1或-1表示正负例  
    """  
    num = test_data.shape[0]  
    preds = []  
    for j in range(num):  
        sum_tmp = 0  
        for i in self.positive_num:  
            sum_tmp += self.train_label[i] * self.sv_alpha[i] *  
self.KERNEL(test_data[j], self.train_data[i])  
        pred = self.b + sum_tmp  
        if pred <= 0:  
            preds.append(-1)  
        else:  
            preds.append(1)  
    result = np.array(preds).reshape(num)  
    return result
```

模型测试

可以看到Linear核函数，Poly 核函数，Gauss 核函数预测准确率分别为80.00%，86.67%，80.00%。如果想进一步提高准确率应该适当调整模型的参数或者更换更合适的核函数。



The screenshot shows a terminal window with the following output:

```
运行: test_svm x  
C:\Users\凝雨\AppData\Local\Programs\Python\Python39\python.exe C:/Users/凝雨/Desktop/人工智能/EXP2_files/src1/test_svm.py  
    pcost    dcost    gap    pres    dres  
0: -2.8121e+02  3.4177e+01  4e+03  4e+00  4e+00  
1:  9.9920e+01 -5.8484e+02  1e+03  6e-01  6e-01  
2: -4.9164e+01 -2.3853e+02  2e+02  7e-02  6e-02  
3: -7.0401e+01 -2.3871e+02  2e+02  5e-02  4e-02  
4: -1.3457e+02 -1.8705e+02  5e+01  1e-02  1e-02  
5: -1.4989e+02 -1.7628e+02  3e+01  4e-03  4e-03  
6: -1.5127e+02 -1.7631e+02  3e+01  3e-03  3e-03  
7: -1.5782e+02 -1.7356e+02  2e+01  2e-03  2e-03  
8: -1.6266e+02 -1.7055e+02  8e+00  7e-04  7e-04  
9: -1.6429e+02 -1.7059e+02  6e+00  3e-04  3e-04  
10: -1.6692e+02 -1.6943e+02  3e+00  9e-05  8e-05  
11: -1.6788e+02 -1.6926e+02  1e+00  4e-05  4e-05  
12: -1.6862e+02 -1.6914e+02  5e-01  9e-06  8e-06  
13: -1.6865e+02 -1.6914e+02  5e-01  7e-06  7e-06  
14: -1.6901e+02 -1.6912e+02  1e-01  7e-07  6e-07  
15: -1.6911e+02 -1.6911e+02  3e-03  2e-08  2e-08  
16: -1.6911e+02 -1.6911e+02  3e-05  2e-10  2e-10  
Optimal solution found.  
SVM(Linear kernel) acc: 80.00%  
    pcost    dcost    gap    pres    dres  
0: -2.3263e+02 -7.1230e+01  3e+03  4e+00  3e+00  
1:  6.9033e+01 -4.2426e+02  7e+02  4e-01  3e-01  
2: -1.7880e+01 -2.5185e+02  3e+02  2e-01  1e-01  
3: -3.5135e+01 -2.2867e+02  2e+02  8e-02  6e-02  
4: -8.8746e+01 -1.4595e+02  6e+01  2e-02  2e-02  
5: -1.1171e+02 -1.3315e+02  2e+01  5e-03  4e-03  
6: -1.2120e+02 -1.3013e+02  9e+00  1e-03  8e-04  
7: -1.2446e+02 -1.2910e+02  5e+00  5e-04  4e-04
```



```
运行: test_svm x
1: 6.9033e+01 -4.2426e+02 7e+02 4e-01 3e-01
2: -1.7880e+01 -2.5185e+02 3e+02 2e-01 1e-01
3: -3.5135e+01 -2.2867e+02 2e+02 8e-02 6e-02
4: -8.8746e+01 -1.4595e+02 6e+01 2e-02 2e-02
5: -1.1171e+02 -1.3315e+02 2e+01 5e-03 4e-03
6: -1.2120e+02 -1.3013e+02 9e+00 1e-03 8e-04
7: -1.2446e+02 -1.2910e+02 5e+00 5e-04 4e-04
8: -1.2639e+02 -1.2860e+02 2e+00 2e-04 1e-04
9: -1.2772e+02 -1.2826e+02 5e-01 2e-05 2e-05
10: -1.2808e+02 -1.2819e+02 1e-01 3e-06 2e-06
11: -1.2815e+02 -1.2817e+02 2e-02 5e-07 4e-07
12: -1.2817e+02 -1.2817e+02 1e-03 2e-08 2e-08
13: -1.2817e+02 -1.2817e+02 1e-05 2e-10 2e-10
Optimal solution found.
SVM(Poly kernel) acc: 86.67%
      pcost      dcost      gap      pres      dres
0: -1.1633e+02 -1.8910e+02 2e+03 3e+00 3e+00
1: -8.2766e+00 -2.5004e+02 2e+02 1e-14 1e-15
2: -7.6076e+01 -1.2905e+02 5e+01 1e-14 5e-16
3: -9.2722e+01 -1.0242e+02 1e+01 2e-14 6e-16
4: -9.6094e+01 -9.7557e+01 1e+00 2e-14 7e-16
5: -9.6625e+01 -9.6759e+01 1e-01 7e-15 8e-16
6: -9.6678e+01 -9.6684e+01 7e-03 1e-14 8e-16
7: -9.6680e+01 -9.6681e+01 4e-04 1e-15 7e-16
8: -9.6681e+01 -9.6681e+01 2e-05 2e-14 7e-16
Optimal solution found.
SVM(Gauss kernel) acc: 80.00%

进程已结束,退出代码0
```

手写感知机模型并进行反向传播

模型初始化

初始化输入层，隐层1，隐层2，隐层3，输出层神经元数目，学习率，以及各个隐层的参数。

```
def __init__(self):
    # layer size = [10, 8, 8, 4]
    # 初始化所需参数
    self.num_1 = 10
    self.num_2 = 10
    self.num_3 = 8
    self.num_4 = 8
    self.num_5 = 4
    self.w1 = np.random.rand(10, 10)
    self.w2 = np.random.rand(8, 10)
    self.w3 = np.random.rand(8, 8)
    self.w4 = np.random.rand(4, 8)
    self.b1 = np.random.rand(10)
    self.b2 = np.random.rand(8)
    self.b3 = np.random.rand(8)
    self.b4 = np.random.rand(4)
    self.lr = 0.05
```

前向传播

前向传播算法(引自实验文档):

$$\begin{aligned} \mathbf{h}_1 &= s_1(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \\ \mathbf{h}_2 &= s_2(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2) \\ \mathbf{h}_3 &= s_3(\mathbf{W}_3 \mathbf{h}_2 + \mathbf{b}_3) \\ \hat{\mathbf{y}} &= s_4(\mathbf{W}_4 \mathbf{h}_3 + \mathbf{b}_4) \\ l(\hat{\mathbf{y}}, \mathbf{y}) &= \text{CrossEntropyLoss}(\hat{\mathbf{y}}, \mathbf{y}) = -\log(\hat{y}_t) \end{aligned}$$

依次将输入通过四层处理，再进行softmax归一化，最后求出交叉熵。

```
def forward(self, x):
    # 前向传播

    h1_out = np.dot(self.w1, x) + np.dot(np.diag(self.b1), np.ones((self.num_2,
100)))
    h1_out = act_func(h1_out)
    h2_out = np.dot(self.w2, h1_out) + np.dot(np.diag(self.b2),
np.ones((self.num_3, 100)))
    h2_out = act_func(h2_out)
    h3_out = np.dot(self.w3, h2_out) + np.dot(np.diag(self.b3),
np.ones((self.num_4, 100)))
    h3_out = act_func(h3_out)
    h4_out = np.dot(self.w4, h3_out) + np.dot(np.diag(self.b4),
np.ones((self.num_5, 100)))

    for i in range(h4_out.shape[0]):
        for j in range(h4_out.shape[1]):
            h4_out[i, j] = np.exp(h4_out[i, j])

    out_col = np.sum(h4_out, axis=0)

    for i in range(h4_out.shape[0]):
        for j in range(h4_out.shape[1]):
            h4_out[i][j] = h4_out[i][j] / out_col[j]

    loss = loss_func(h4_out)
    return h4_out, h3_out, h2_out, h1_out, loss
```

反向传播和梯度下降

反向传播和梯度下降算法(引自实验文档):

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{W}_4} &= (l' s_4') \mathbf{h}_3^T, \frac{\partial L}{\partial \mathbf{b}_4} = l' s_4' \\ \frac{\partial L}{\partial \mathbf{W}_3} &= (\mathbf{W}_4^T (l' s_4') \odot s_3') \mathbf{h}_2^T, \frac{\partial L}{\partial \mathbf{b}_3} = \mathbf{W}_4^T (l' s_4') \odot s_3' \\ \frac{\partial L}{\partial \mathbf{W}_2} &= (\mathbf{W}_3^T (\mathbf{W}_4^T (l' s_4') \odot s_3') \odot s_2') \mathbf{h}_1^T, \frac{\partial L}{\partial \mathbf{b}_2} = \mathbf{W}_3^T (\mathbf{W}_4^T (l' s_4') \odot s_3') \odot s_2' \\ \frac{\partial L}{\partial \mathbf{W}_1} &= (\mathbf{W}_2^T (\mathbf{W}_3^T (\mathbf{W}_4^T (l' s_4') \odot s_3') \odot s_2') \odot s_1') \mathbf{x}^T, \frac{\partial L}{\partial \mathbf{b}_1} = \mathbf{W}_2^T (\mathbf{W}_3^T (\mathbf{W}_4^T (l' s_4') \odot s_3') \odot s_2') \odot s_1' \end{aligned}$$

$$\mathbf{W}_i = \mathbf{W}_i - \eta \frac{\partial L}{\partial \mathbf{W}_i}$$

$$\mathbf{b}_i = \mathbf{b}_i - \eta \frac{\partial L}{\partial \mathbf{b}_i}$$

依次求出交叉熵L关于各个隐层参数的偏导数，再将偏导数乘学习率后更新隐层参数。

```
def backward(self, a4, a3, a2, a1, x): # 自行确定参数表
    # 反向传播
    ones = np.ones((100, 1))

    l_z4 = a4 - labels.T
    l_w4 = np.dot(l_z4, a3.T)
    l_b4 = np.dot(l_z4, ones)

    # print(self.w3.T.shape)
    # print(l_z4.shape)

    l_z3 = np.multiply(np.dot(self.w4.T, l_z4), np.ones((a3.shape[0],
a3.shape[1]))) - np.multiply(a3, a3))
    l_w3 = np.dot(l_z3, a2.T)
    l_b3 = np.dot(l_z3, ones)

    l_z2 = np.multiply(np.dot(self.w3.T, l_z3), np.ones((a2.shape[0],
a2.shape[1]))) - np.multiply(a2, a2))
    l_w2 = np.dot(l_z2, a1.T)
    l_b2 = np.dot(l_z2, ones)

    l_z1 = np.multiply(np.dot(self.w2.T, l_z2), np.ones((a1.shape[0],
a1.shape[1]))) - np.multiply(a1, a1))
    l_w1 = np.dot(l_z1, x)
    l_b1 = np.dot(l_z1, ones)

    self.w1 -= self.lr * l_w1
    self.w2 -= self.lr * l_w2
    self.w3 -= self.lr * l_w3
    self.w4 -= self.lr * l_w4
    # print(l_b1)
    # print(self.b1)
    self.b1 -= self.lr * l_b1.reshape(self.num_2)
    self.b2 -= self.lr * l_b2.reshape(self.num_3)
    self.b3 -= self.lr * l_b3.reshape(self.num_4)
    self.b4 -= self.lr * l_b4.reshape(self.num_5)
```

模型训练

模型一共迭代epochs次，每次先正向传播，记录交叉熵即损失函数，再反向传播。最后输出模型隐层参数。

```
def train():
    """
    mlp: 传入实例化的MLP模型
    epochs: 训练轮数
```

```

lr: 学习率
inputs: 生成的随机数据
labels: 生成的one-hot标签
"""
loss_list = []
for i in range(epochs):
    h4_out, h3_out, h2_out, h1_out, loss = mlp.forward(inputs.T)
    loss_list.append(loss)
    mlp.backward(h4_out, h3_out, h2_out, h1_out, inputs)
print("w1: ")
print(mlp.w1)
print("w2: ")
print(mlp.w2)
print("w3: ")
print(mlp.w3)
print("w4: ")
print(mlp.w4)
print("b1: ")
print(mlp.b1)
print("b2: ")
print(mlp.b2)
print("b3: ")
print(mlp.b3)
print("b4: ")
print(mlp.b4)

return loss_list

```

损失函数曲线绘制

这里使用python的可交互式绘图模块plotly绘制损失函数曲线，集成了手写MLP模型和自动pytorch模型损失函数画迹。

```

epochs_list = np.zeros(epochs)
for i in range(epochs):
    epochs_list[i] = i
trace1 = go.Scatter(
    x=epochs_list,
    y=loss_list,
    mode='lines',
    name='MLP'
)
trace2 = go.Scatter(
    x=epochs_list,
    y=loss_list_2,
    mode='lines',
    name='pytorch'
)
data = [trace1, trace2]
layout = go.Layout(
    title='损失函数与迭代次数关系折线图',
    xaxis=dict(title="epoch"),
    yaxis=dict(title="loss"),
)

fig = go.Figure(data=data, layout=layout)
py.plot(fig, filename='MLP')

```

模型测试

学习率为0.05，迭代次数为1000时最终的模型隐层参数 W_i ， b_i 如下所示：

```
C:\Users\凝雨\AppData\Local\Programs\Python\Python39\python.exe C:/Users/凝雨/Desktop/人工智能/EXP2_files/src2/MLP_manual.py
```

w1:

```
[[ -3.28243250e+00  5.64860799e-01 -1.80661257e+00 -1.33285374e+01
   1.00821442e+01  1.95291812e-02 -3.40232550e+00 -1.74070406e-01
   1.08400321e+01 -1.42290042e-01]
 [ 7.30595908e+00  7.69381880e-01  6.06871408e-01  1.64942947e+00
   9.35754388e-01  1.41687654e+00 -4.05756284e-01 -8.59345759e+00
   5.37580770e+00  5.25168502e-01]
 [ 5.38921363e+00 -1.67591395e+00 -5.52303170e+00  6.29610752e-01
   3.02406050e+00  1.12669340e+01  5.10003722e+00 -1.65311512e+00
  -2.64751978e+00 -9.26560992e+00]
 [ 2.77066416e+00 -6.74508214e+00  1.68305976e+00 -6.62665526e-01
   1.20797348e+01 -7.15634236e-01  5.80144581e-01 -1.22792121e-01
   4.03537897e-01  4.46839612e+00]
 [ 1.01685317e+01 -6.21706632e-01 -2.28400782e+00 -1.05830766e-01
   3.60861681e+00 -2.84546704e-01  3.95042964e+00 -4.91750093e+00
   4.80938669e+00 -1.30899273e+00]
 [-3.57238302e+00  3.39675114e+00 -1.80580568e+00  3.66813894e+00
   8.19997503e-01 -3.19099312e+00  4.41917313e+00 -1.60743054e+00
   4.80577087e+00  6.78638576e+00]
 [-1.06630581e+00  3.57117349e+00  3.31816964e+00 -4.26791280e+00
  -4.45264629e+00 -1.12989423e+00  1.05096504e+01  1.11651297e+00
  -4.05565165e-01 -1.26216925e+00]
 [-1.58467978e+00 -1.82027497e+00  1.01023710e+00  4.74173273e+00
   5.81598258e+00  6.18838960e-03 -1.23241350e+00  4.54656266e+00
   4.44509132e+00 -1.65364036e-01]
 [ 9.08637001e-01  2.09002441e+00  8.71281189e-01  5.16321306e+00
   3.12038512e+00  2.35772021e+00 -3.91990605e+00  2.30051466e+00
  -2.66677372e+00 -2.75549077e+00]
 [-6.00051319e+00  6.39733004e+00  4.21073356e+00  6.04293345e+00
   2.03971807e+00 -3.60077841e+00 -1.85022723e+00  4.16066545e+00
   6.80866871e+00  3.92053028e+00]]
```

w2:

```
[[ 2.08948880e+00 -6.22791415e+00  5.49428331e-01  6.59516162e+00
   2.58247424e-02 -2.70323652e+00 -6.69078007e+00  3.47770535e+00
  -2.69118601e-01 -1.15946944e+00]
 [-3.06196143e+00  2.28034380e+00  9.13237952e+00  3.27850328e+00
   3.23505452e+00  3.29118999e-03  2.63287071e+00  1.37229498e+00
   1.74523884e-01 -2.06478708e+00]
 [-2.77739180e+00 -6.08299837e+00 -3.56000255e+00  2.67774200e+00
   1.96992139e+00 -1.65947565e+00 -5.77862309e+00  1.33347226e+00
   2.82672405e+00 -4.88973363e+00]
 [ 1.82593116e+00  3.36265235e+00 -5.15689990e-01  6.24374536e-01
  -7.79005571e-01 -3.88302349e+00 -1.50172626e+00 -3.38066055e+00
  -2.55243240e+00 -6.14822465e+00]
 [ 8.83231247e-01 -9.37065258e-02 -4.68435655e+00  5.16927665e+00
   9.80318217e-01 -4.51132528e+00 -1.11407474e+00  4.07445281e+00
   5.61045770e+00  2.46442108e+00]
 [-3.22664748e+00  8.09615778e+00 -5.28935068e+00  7.35825473e-01
   3.65610598e+00 -4.35704273e+00 -1.96414180e+00 -1.73424427e+00
   1.59473403e-01  5.98740587e+00]
 [-5.70209721e+00  1.59383376e+00  3.90392783e-03  3.17640253e-01
```

```

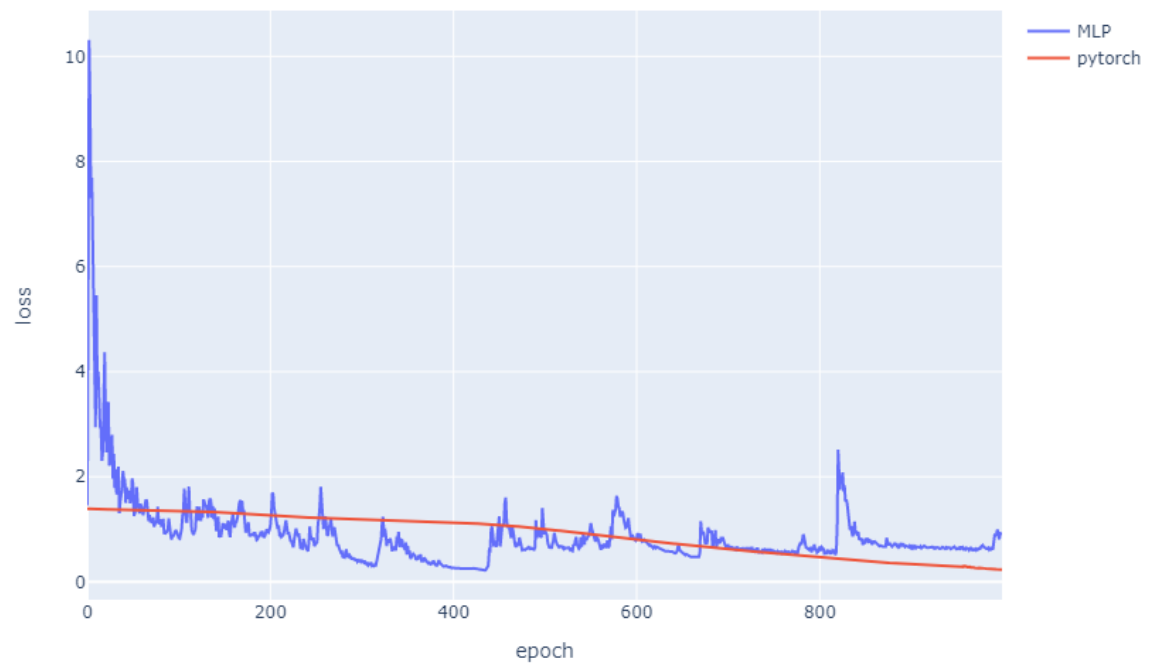
2.52548841e+00 -6.03772106e+00 4.26017955e-01 1.99967333e-01
-6.44039096e-01 -2.51840833e+00]
[ 5.16519353e+00 5.00950319e+00 3.61297540e+00 1.34275772e+00
4.41379010e+00 1.76119072e+00 4.01923497e+00 1.42306358e+00
-1.28448983e+00 -8.38252431e-01]]
w3:
[[ 1.88221755 -2.77938388 -2.51538624 -2.47879862 -2.89687751 -2.57304925
-5.00650712 -3.42750817]
[-4.0865764 4.38464308 -4.17087314 -1.35173501 1.87222497 0.19874815
2.81656074 -0.23901395]
[ 0.39235063 3.75975098 3.70084268 5.05759057 2.15478484 3.05328217
0.06873308 1.53424649]
[-1.8246859 -4.29814368 3.86878357 -1.34331715 -1.21176236 -4.34292816
1.85558943 -2.47802638]
[-0.1272371 -2.79755368 -0.91913084 4.12232172 -5.01323351 1.47902279
1.64208667 1.7686348 ]
[ 2.54954298 3.46857561 -5.47029388 -0.57685333 1.86922517 3.29169784
3.34133403 2.84606761]
[ 2.68288287 -0.85599924 0.51447612 -5.57519239 -3.99974039 -1.71266476
3.84468622 -4.39985025]
[-5.74434016 -1.67309074 -2.65772481 -3.45267148 -2.64797252 -4.59952184
-2.84269366 -2.38570283]]
w4:
[[ 4.46317652 -2.53396325 0.04631763 -1.32909028 1.95648595 2.84491748
3.86407813 0.06070073]
[ 1.8408349 3.91885477 -1.96459285 2.65233586 1.16392416 -1.15797396
-1.92986739 -2.9314586 ]
[-1.78327678 1.37401725 5.57016887 0.38050904 -2.66631185 -1.03096539
0.05506517 1.83574591]
[-2.22410058 0.09603113 -1.83670651 0.56072607 2.28608087 1.64362812
-0.05870673 3.72727927]]
b1:
[ 2.54530813 0.19525162 -6.85867692 -3.72854327 -5.53741245 3.49782868
1.03058918 0.23819914 -3.29770073 -1.87150592]
b2:
[-1.57666124 3.8707203 2.11288406 3.52705609 -2.79483527 2.25366883
-2.82388133 -0.56876664]
b3:
[ 4.1053479 -4.52296775 4.62859927 -3.83423143 6.3216477 -1.53652425
-1.94962526 -1.3371373 ]
b4:
[-1.61161578 3.00538058 -1.00279612 1.49479045]

```

进程已结束,退出代码0

手写MLP模型和**自动pytorch模型**损失函数随迭代次数关系曲线如下，可以看到手写模型loss曲线比较陡峭，且收敛性较弱，而自动pytorch模型收敛性较强，曲线较为顺滑。

损失函数与迭代次数关系折线图



卷积神经网络

根据要求我的学号后两位是88，应选择第5个模型，通过计算可得模型参数如下：

layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8	激活函数
Conv2d(3, 12, 5)	MaxPool2d(2)	Conv2d(12, 24, 3)	MaxPool2d(2)	Conv2d(24, 32, 3)	Linear(512, 108)	Linear(108, 84)	Linear(84, 10)	relu

模型初始化

设置各个卷积层，池化层和线性层，在每个卷积层和线性层后加入激活函数，在进入线性层阵列前加入Flatten()将向量展开。

```
def __init__(self):
    super(MyNet, self).__init__()
    #####
    # 这里需要写MyNet的卷积层、池化层和全连接层
    self.my_net = nn.Sequential(
        nn.Conv2d(3, 12, 5),
        nn.ReLU(),
        nn.MaxPool2d(2),
        nn.Conv2d(12, 24, 3),
        nn.ReLU(),
        nn.MaxPool2d(2),
        nn.Conv2d(24, 32, 3),
        nn.ReLU(),
        nn.Flatten(),
        nn.Linear(512, 108),
```

```

        nn.ReLU(),
        nn.Linear(108, 84),
        nn.ReLU(),
        nn.Linear(84, 10),
        nn.ReLU(),
    )

```

前向传播

将x代入模型计算结果再返回

```

def forward(self, x):
    #####
    #这里需要写MyNet的前向传播
    x = self.my_net(x)
    return x

```

优化器和损失函数定义

定义优化器为Adam，损失函数为交叉熵。

```

# 自己设定优化器和损失函数
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)

```

计算测试集损失函数和准确率

计算每个测试用例的损失后进行累加，准确率是正确的预测数除以测试用例数。

```

for inputs, labels in test_loader:
    inputs, labels = inputs.to(device), labels.to(device)
    #####
    # 需要计算测试集的loss和accuracy
    prediction = net(inputs)
    print(prediction.shape)
    test_loss += loss_function(prediction, labels)
    prediction = np.argmax(prediction, axis=1)
    num_correct += (prediction == labels).sum()
length = len(test_loader.dataset)
accuracy = num_correct / length

```

模型测试

可以看到模型预测的正确率为57%，说明该模型有一定的预测能力但仍有待提高。如果想进一步提高准确率应该适当调整模型的参数和结构。


```
运行: MyNet x
C:\Users\凝雨\AppData\Local\Programs\Python\Python39\python.exe C:/Users/凝雨/Desktop/人工智能/EXP2_files/src2/MyNet.py
Train Epoch: 0/5 [0/50000] Loss: 2.303592
Train Epoch: 0/5 [12800/50000] Loss: 1.860245
Train Epoch: 0/5 [25600/50000] Loss: 1.899082
Train Epoch: 0/5 [38400/50000] Loss: 1.642756
Train Epoch: 1/5 [0/50000] Loss: 1.690813
Train Epoch: 1/5 [12800/50000] Loss: 1.545214
Train Epoch: 1/5 [25600/50000] Loss: 1.542243
Train Epoch: 1/5 [38400/50000] Loss: 1.515969
Train Epoch: 2/5 [0/50000] Loss: 1.407736
Train Epoch: 2/5 [12800/50000] Loss: 1.474881
Train Epoch: 2/5 [25600/50000] Loss: 1.393515
Train Epoch: 2/5 [38400/50000] Loss: 1.369667
Train Epoch: 3/5 [0/50000] Loss: 1.318228
Train Epoch: 3/5 [12800/50000] Loss: 1.255145
Train Epoch: 3/5 [25600/50000] Loss: 1.234399
Train Epoch: 3/5 [38400/50000] Loss: 1.370554
Train Epoch: 4/5 [0/50000] Loss: 1.141724
Train Epoch: 4/5 [12800/50000] Loss: 1.141157
Train Epoch: 4/5 [25600/50000] Loss: 1.183843
Train Epoch: 4/5 [38400/50000] Loss: 1.300333
Finished Training
Test set: Average loss: 2.4243 Acc 0.57

进程已结束,退出代码0
```

实验总结

通过本次实验我学会了传统机器学习和深度学习算法的基本思路，也熟悉了python处理数据的常用方法。