人工智能基础 Lab2

PB19111675 德斯别尔

传统机器学习

决策树

决策树生成算法：

输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
属性集 $A = \{a_1, a_2, \ldots, a_d\}$.

过程： 函数 TreeGenerate(D, A)

```
生成结点 node;
if D 中样本全属于同一类别 C then
    将 node 标记为 C 类叶结点; return
end if
if A = ∅ OR D 中样本在 A 上取值相同 then
    将 node 标记为叶结点, 其类别标记为 D 中样本数最多的类; return
end if
从 A 中选择最优划分属性 a_*;
for a_* 的每一个值 a_*^v do
    为 node 生成一个分支; 令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
    if D_v 为空 then
        将分支结点标记为叶结点, 其类别标记为 D 中样本最多的类; return
    else
        以 TreeGenerate(D_v, A \ {a_*})为分支结点
    end if
end for
```

输出： 以 node 为根结点的一棵决策树

图 4.2　决策树学习基本算法

其中给定了类`DecisionTree`以及其中的两个接口：

- `fit(train_features, train_labels)`,在这个函数中根据训练数据生成一棵决策树。
- `predict(test_features)`,在这个函数中你需要根据已经生成的决策树来预测标签。

选择最优属性划分以及节点生成：

```python
def TreeGenerate(self,train_features,train_labels,A):
        node = Treenode()
        if len(A) == 0 or len(set(train_labels)) <= 1 or len(train_features) <= 1:   # label均相同，或无可划分的属性集
            node.label = np.argmax(np.bincount(train_labels))    # 返回train_labels 的众数
            return node
        # 选择期望剩余熵最小的作为最优划分
        av = A[np.argmin([Remainder(i,train_features,train_labels) for i in A])]
        # 属性a的值域
        range_av = set([i[av] for i in train_features])
```

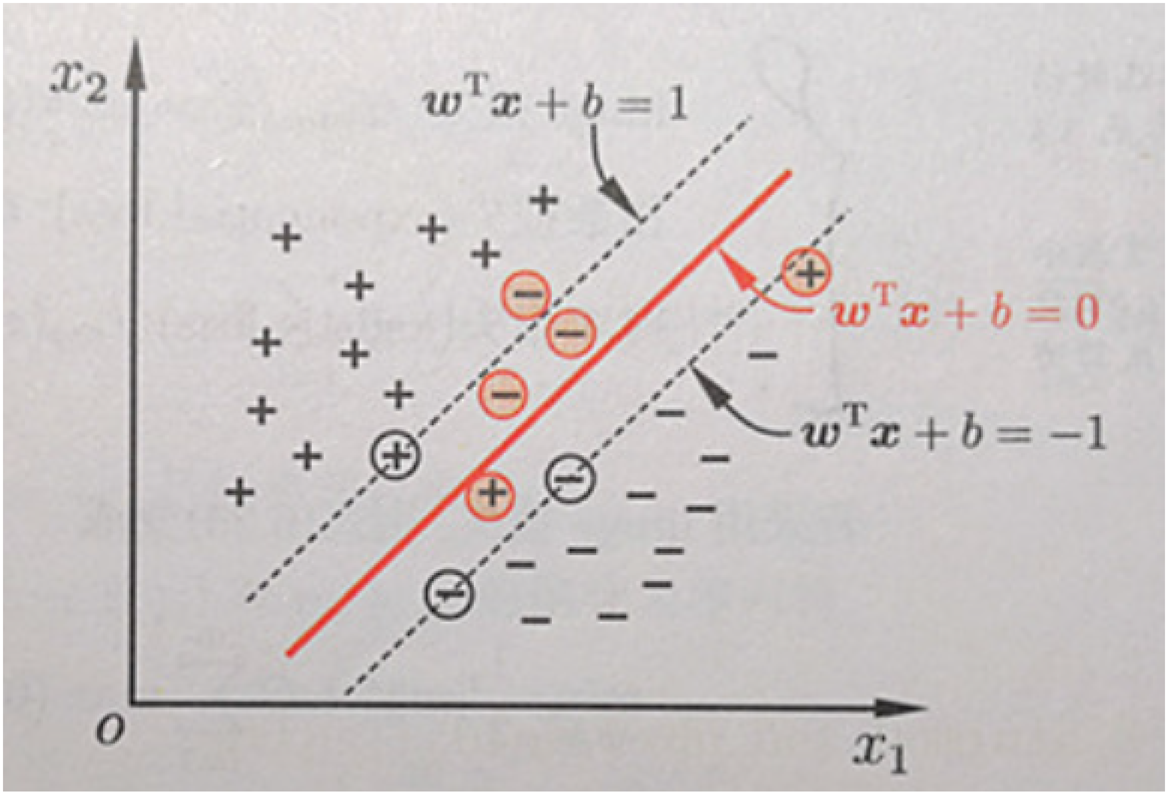
```
        for l in range_av:
            Dvfeatures = [ feat for feat in train_features if feat[av] == l]
            Dvlabels   = [ train_labels[i] for i in range(len(train_labels)) if train_features[i][av] == l]
            # Dv不会是空集
            node.child.append([ l, self.TreeGenerate(Dvfeatures,Dvlabels,[a for a in A if a != av])])

        node.leaf = False # 改为不是叶子
        node.attr = av      # 该节点是依据attr划分的
        return node
```

运行结果：

```
(base) D:\AI\lab\exp2\src1>python main.py
DecisionTree acc: 69.64%
SVM(Linear kernel) acc: 86.67%
SVM(Poly kernel) acc: 86.67%
SVM(Gauss kernel) acc: 86.67%
```

支持向量机

实验要求使用软间隔SVM来完成实验。



$$min_w \frac{1}{2} + C\sum_{N=1}^{N} |[y_n \neq sign(w^T z_n + b)]|$$

$$s.t. y_n(w^T x_n + b) \geq 1 - \xi_i, \xi_i > 0$$

二次型

```
# 二次型规划
        P = np.array([[train_labels[i]*train_labels[j]*self.KERNEL(train_features[i],train_features[j]) for j in range(m)] for i in range(m)])
        q = np.array([-1]*m)
        G = np.array(list(-1*np.eye(m)) + list(np.eye(m)))
        h = np.array([0]*m + [self.C]*m)
        A = np.array([train_labels])
        b = np.array([0])
        # cvxopt求解二次型规划
        Pc = matrix(P,tc='d')
        qc = matrix(q,tc='d')
        Gc = matrix(G,tc='d')
        hc = matrix(h,tc='d')
        Ac = matrix(A,tc='d')
        bc = matrix(b,tc='d')
        sol = solvers.qp(Pc,qc,Gc,hc,Ac,bc)
        a = sol['x']
```

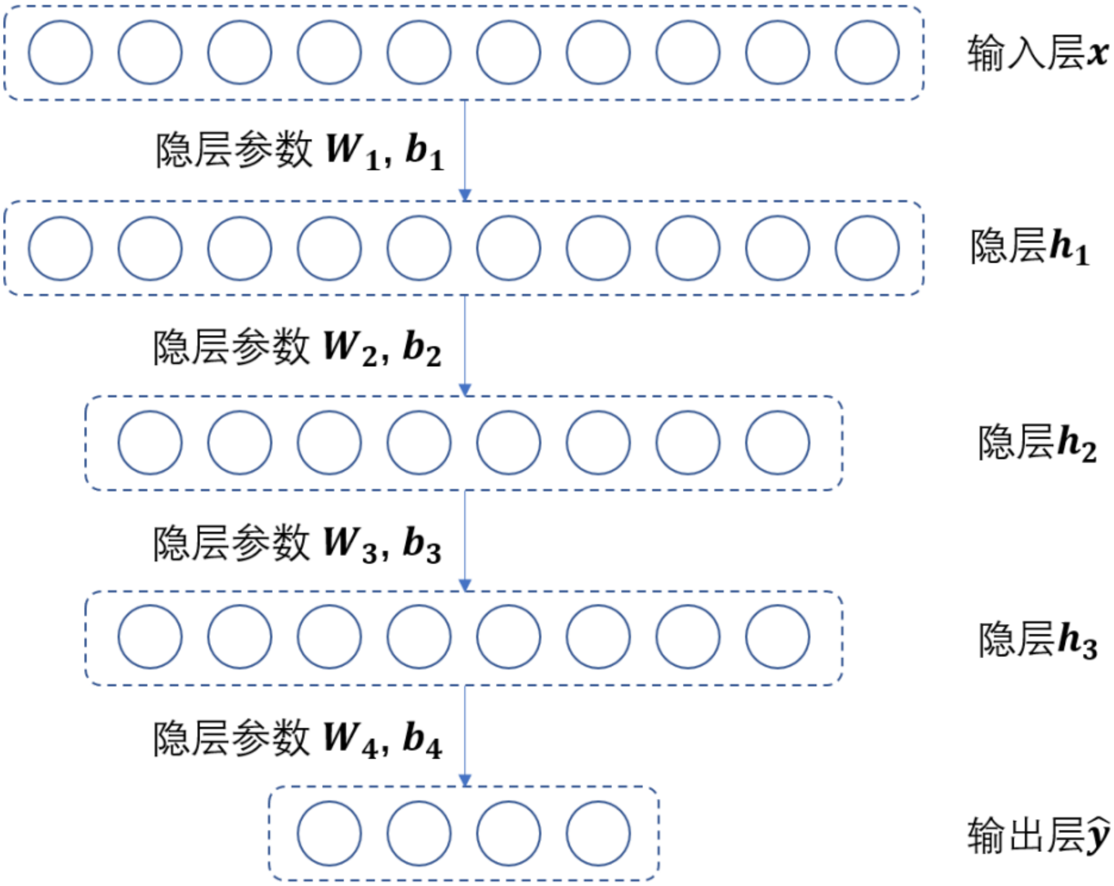
运行结果:

```
(base) D:\AI\lab\exp2\src1>python main.py
DecisionTree acc: 69.64%
SVM(Linear kernel) acc: 86.67%
SVM(Poly kernel) acc: 86.67%
SVM(Gauss kernel) acc: 86.67%
```

深度学习

感知机模型

感知机模型:



前向传播:

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

其中，$\mathbf{y}$是样本类别的one-hot向量表示，t是样本所处的类别，$\hat{y_t}$是$\hat{\mathbf{y}}$的第t个分量。

反向传播:

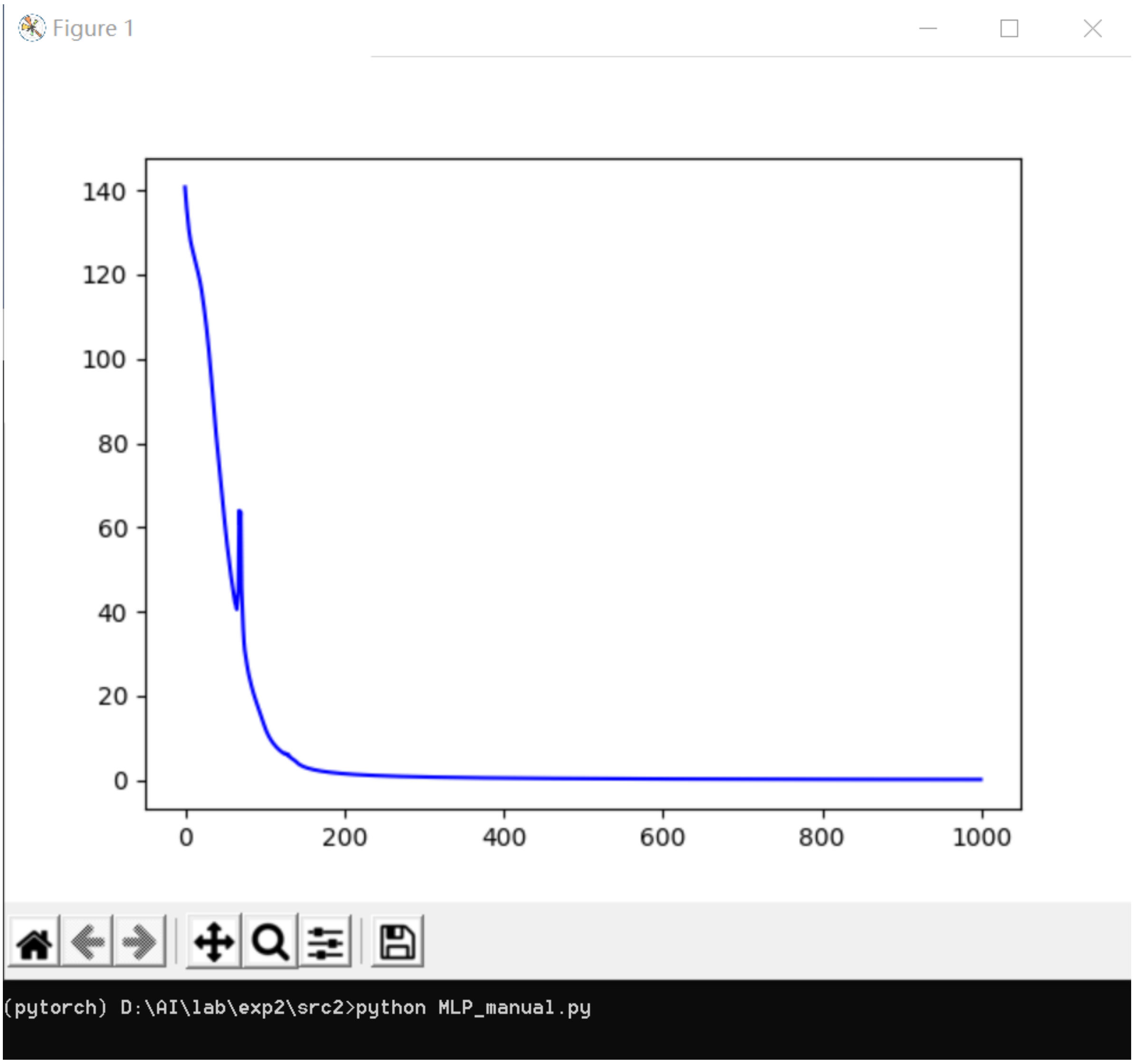
链式法则的展开形式：

$$
\begin{aligned}
\frac{\partial L}{\partial \mathbf{W_4}} &= (l' s_4{}')\mathbf{h}_3^{\mathrm{T}}, \frac{\partial L}{\partial \mathbf{b_4}} = l' s_4{}' \\
\frac{\partial L}{\partial \mathbf{W_3}} &= (\mathbf{W}_4^{\mathrm{T}}(l' s_4{}') \odot s_3{}')\mathbf{h}_2^{\mathrm{T}}, \frac{\partial L}{\partial \mathbf{b_3}} = \mathbf{W}_4^{\mathrm{T}}(l' s_4{}') \odot s_3{}' \\
\frac{\partial L}{\partial \mathbf{W_2}} &= (\mathbf{W}_3^{\mathrm{T}}(\mathbf{W}_4^{\mathrm{T}}(l' s_4{}') \odot s_3{}') \odot s_2{}')\mathbf{h}_1^{\mathrm{T}}, \frac{\partial L}{\partial \mathbf{b_2}} = \mathbf{W}_3^{\mathrm{T}}(\mathbf{W}_4^{\mathrm{T}}(l' s_4{}') \odot s_3{}') \odot s_2{}' \\
\frac{\partial L}{\partial \mathbf{W_1}} &= (\mathbf{W}_2^{\mathrm{T}}(\mathbf{W}_3^{\mathrm{T}}(\mathbf{W}_4^{\mathrm{T}}(l' s_4{}') \odot s_3{}') \odot s_2{}') \odot s_1{}')\mathbf{x}^{\mathrm{T}}, \frac{\partial L}{\partial \mathbf{b_1}} = \mathbf{W}_2^{\mathrm{T}}(\mathbf{W}_3^{\mathrm{T}}(\mathbf{W}_4^{\mathrm{T}}(l' s_4{}') \odot s_3{}') \odot s_2{}') \odot s_1{}'
\end{aligned}
$$

其中$\odot$表示按位乘，并且：

$$
\begin{gathered}
s_4(x_1, x_2, x_3, x_4) = Softmax(x_1, x_2, x_3, x_4) = \frac{1}{e^{x_1} + e^{x_2} + e^{x_3} + e^{x_4}}(e^{x_1}, e^{x_2}, e^{x_3}, e^{x_4}), \\
s_1 = s_2 = s_3 = \tanh(\cdot) \\
s_1{}' = s_2{}' = s_3{}' = 1 - \tanh^2 \\
(l' s_4{}')_i = \begin{cases} \hat{y}_i - 1 & i = t \\ \hat{y}_i & i \neq t \end{cases}
\end{gathered}
$$

运行结果：



卷积神经网络

对卷积神经网络的初步掌握，实现图像分类。

学号为 PB19111675， 最后两位相加然后模6再加1计算得出自己的模型。

$$(7+5)\%6+1=1$$

既选择列表中第一个模型。

编号	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8	激活函数
--	2d卷积	池化	2d卷积	池化	卷积	线性层	线性层	线性层	--
1	16，5	最大池化	32，5	最大池化	-	120	84	10	tanh

表格说明：

- 2d卷积（a,b) a:kernel个数，b:kernel size为（b,b)
- 默认池化大小为2
- 线性层 b:output channel的大小 (32 * 5 * 5)
- 激活函数 在每个卷积和线性层后都加入激活函数，池化层无需添加
- -表示没有

运行结果：

```
D:\Anaconda\envs\pytorch\lib\site-packages\torch\nn\functional.py:1795: UserW
 instead.
  warnings.warn("nn.functional.tanh is deprecated. Use torch.tanh instead.")
Train Epoch: 0/5 [0/50000]      Loss: 2.298599
Train Epoch: 0/5 [12800/50000]  Loss: 1.926111
Train Epoch: 0/5 [25600/50000]  Loss: 1.728492
Train Epoch: 0/5 [38400/50000]  Loss: 1.625720
Train Epoch: 1/5 [0/50000]      Loss: 1.653885
Train Epoch: 1/5 [12800/50000]  Loss: 1.595533
Train Epoch: 1/5 [25600/50000]  Loss: 1.499322
Train Epoch: 1/5 [38400/50000]  Loss: 1.576571
Train Epoch: 2/5 [0/50000]      Loss: 1.544436
Train Epoch: 2/5 [12800/50000]  Loss: 1.507099
Train Epoch: 2/5 [25600/50000]  Loss: 1.476817
Train Epoch: 2/5 [38400/50000]  Loss: 1.442285
Train Epoch: 3/5 [0/50000]      Loss: 1.486340
Train Epoch: 3/5 [12800/50000]  Loss: 1.481541
Train Epoch: 3/5 [25600/50000]  Loss: 1.453739
Train Epoch: 3/5 [38400/50000]  Loss: 1.456928
Train Epoch: 4/5 [0/50000]      Loss: 1.311592
Train Epoch: 4/5 [12800/50000]  Loss: 1.367461
Train Epoch: 4/5 [25600/50000]  Loss: 1.412440
Train Epoch: 4/5 [38400/50000]  Loss: 1.336563
Finished Training
Test set: Average loss: 2.8066   Acc 0.62
```