

人工智能实验二

PB18000221 袁一玮

0.1 传统机器学习

0.1.1 最小二乘分类

参考 Slides 上的做法，可以使用梯度下降法求解（后面需加上 λw^2 正规化项）

Solving Least Squares Classification

Let

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ \vdots & & & \\ 1 & x_{N1} & \cdots & x_{Nd} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} b \\ \vdots \\ w_d \end{bmatrix}$$

$$\begin{aligned} \text{Loss} &= \min_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^2 = \min_{\mathbf{w}} (\mathbf{X}\mathbf{w} - \mathbf{y})^2 \\ &= \min_{\mathbf{w}} (\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) \end{aligned}$$

图 1: 1.1

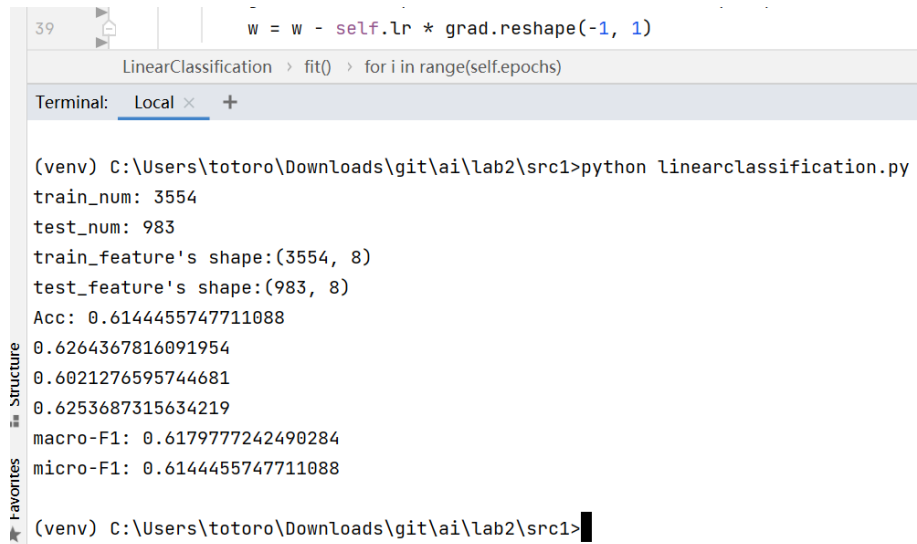
得到迭代式: $\mathbf{w}' = \mathbf{w} - \text{LearningRate} * \text{Partial}(\text{Loss}/\mathbf{w})$

输出结果如下

Solving for w

$$\begin{aligned}\frac{\partial \text{Loss}}{\partial \mathbf{w}} &= 2(\mathbf{X}\mathbf{w} - \mathbf{y})^\top \mathbf{X} = 0 \\ \mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{y} &= 0 \\ \mathbf{w}^* &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}\end{aligned}$$

图 2: 1.2



The screenshot shows a Jupyter Notebook interface. At the top, a code cell contains the line `w = w - self.lr * grad.reshape(-1, 1)`. Below the code cell, the notebook's execution history is visible, showing the output of the `fit()` method. The output includes the number of training and testing samples, the shapes of the feature matrices, and various performance metrics.

```

LinearClassification > fit() > for i in range(self.epochs)

Terminal: Local x +

(venv) C:\Users\totoro\Downloads\git\ai\lab2\src1>python linearclassification.py
train_num: 3554
test_num: 983
train_feature's shape:(3554, 8)
test_feature's shape:(983, 8)
Acc: 0.6144455747711088
0.6264367816091954
0.6021276595744681
0.6253687315634219
macro-F1: 0.6179777242490284
micro-F1: 0.6144455747711088

(venv) C:\Users\totoro\Downloads\git\ai\lab2\src1>

```

图 3: 1.out

0.1.2 朴素贝叶斯

对 feature[0] 使用离散计算频率, feature[1..7] 采用使用正态分布拟合

初始化时遍历整个训练数据集, 统计 feature[0] 为 1,2,3 时各个类别的数量和 feature[1..7] 不同分类下的子集 subset_dict

拟合阶段, 对离散变量 feature[0] 计算先验概率 $P(c)$ 和条件概率 $P(x_i|c)$, 对连续性变量 feature[1..7] 计算各个 subset 的平均值和标准差, 得到对应的 $P_{xc}(i, j)$ 参数分布, 通过寻找最大式来得到预测的值

$$h_{nb}(x) = \operatorname{argmax}_{c \in \mathcal{Y}} P(c) \prod_{i=1}^d P(x_i|c)$$

图 4: 2.1

输出结果如下

0.1.3 支持向量机

根据瓜书和 Slides 上的方法, 根据对偶方程解出一组 α_i , 然后在这里使用 cvxopt 来求解 α_i

输出结果如下

上面是使用 Linear 核的结果

若使用 Poly 核, 效果略差于 Linear:

Gauss 核, 效果和 Linear 相当:

0.2 深度学习

```

139         print("Acc: " + str(get_acc(test_label, pred)))
140         print("macro-F1: " + str(get_macro_F1(test_label, pred)))
141         print("micro-F1: " + str(get_micro_F1(test_label, pred)))

```

NaiveBayes > predict() > for k in range(test_num) > for c in range(1,4)

Terminal: Local × +

```

--use-feature <feature>      Enable new functionality, that may be backward
--use-deprecated <feature>  Enable deprecated functionality, that will be

(venv) C:\Users\totoro\Downloads\git\ai\lab2\src1>python nBayesClassifier.p
train_num: 3554
test_num: 983
train_feature's shape:(3554, 8)
test_feature's shape:(983, 8)
Acc: 0.6134282807731435
0.7137404580152672
0.4725111441307578
0.6684005201560468
macro-F1: 0.6182173741006906
micro-F1: 0.6134282807731435

```

图 5: 2.out

The Optimization Problem

- The dual of this new constrained optimization problem is

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^{\top} \mathbf{x}_j)$$

subject to $\forall i, 0 \leq \alpha_i \leq C, \sum_{i=1}^n \alpha_i y_i = 0$

- This is very similar to the optimization problem in the linear separable case, except that there is an upper bound C on α_i now
- Once again, a QP solver can be used to find α_i

图 6: 3.1

```

145         print("Acc: " + str(get_acc(test_label, pred)))
146         print("macro-F1: " + str(get_macro_F1(test_label, pred)))
147         print("micro-F1: " + str(get_micro_F1(test_label, pred)))

```

SupportVectorMachine > fit()

Terminal: Local × +

```

14: -1.9271e+03 -1.9271e+03 4e-03 2e-09 7e-13
15: -1.9271e+03 -1.9271e+03 4e-05 2e-11 7e-13
Optimal solution found.
Acc: 0.6581892166836215
0.7678571428571428
0.568733153638814
0.6804123711340206
macro-F1: 0.6723342225433259
micro-F1: 0.6581892166836215

```

图 7: 3.out.Linear

```

110
111     kernel = 'Poly'
112     C = 1
113     Epsilon = 10e-5
114     # 生成SVM分类器

```

main()

Terminal: Local × +

```

18: -1.8683e+03 -1.8684e+03 8e-02 7e-09 4e-12
19: -1.8683e+03 -1.8683e+03 1e-02 9e-10 4e-12
20: -1.8683e+03 -1.8683e+03 4e-04 2e-11 4e-12
Optimal solution found.
Acc: 0.6449643947100712
0.750551876379691
0.5717948717948718
0.6575716234652115
macro-F1: 0.6599727905465914
micro-F1: 0.6449643947100712

```

图 8: 3.out.Poly

```
110 # Epsilon为拉格朗日乘子阈值，低于此阈值时将该乘子设置为0
111 kernel = 'Gauss'
112 C = 1
113 Epsilon = 10e-5
114 # 生成SVM分类器

main()

Terminal: Local × +
19: -1.8769e+03 -1.8769e+03 5e-02 2e-10 6e-14
20: -1.8769e+03 -1.8769e+03 7e-03 2e-11 6e-14
21: -1.8769e+03 -1.8769e+03 2e-04 1e-13 6e-14
Optimal solution found.
Acc: 0.6561546286876907
0.755056179775281
0.570673712021136
0.6832460732984293
macro-F1: 0.6696586550316154
micro-F1: 0.6561546286876907
```

图 9: 3.out.Gauss

0.2.1 手写感知机模型

我在网上搜寻到的 BP 指的是 w 的迭代公式: <https://zhuanlan.zhihu.com/p/45190898>

五、BP算法

我们知道, 给定一个输出, 通过一次正向传播, 我们就能获得输出。但是这是假设已经训练好了神经网络的情况下。然而训练网络的过程才是最难的。

下面就来介绍最经典最常用的训练网络的算法, BP算法。这个算法算是机器学习入门的一大门槛之一, 估计劝退了不少人。我也是折腾了好久才搞明白这个算法的原理。下面我们来慢慢解析这个大名鼎鼎的BP算法。

BP算法是一种更新权重的方法, 我们知道每一层都有一个权重 \mathbf{W}_l 在BP算法中, 权重的更新依据是这样的:

$$\mathbf{W}_l = \mathbf{W}_l - \eta \frac{\partial C}{\partial \mathbf{W}_l}$$

其中 C 是我们定义的损失函数, η 是我们设定的学习率常数。对于回归问题, 通常定义损失函

图 10: 4.BP

而给的实验指导上是梯度下降公式, 我不是很分得清后两点小分的具体要求

程序最后输出的结果如下

输出的 $\log(\text{Loss})$ 如下

0.2.2 MLP-Mixer

这个视频帮我理解了 MLP-Mixer: <https://www.youtube.com/watch?v=Ilg3gGewQ5U>

但是实验最后并没有成功:

1. 我不知道该如何处理 NN 使得其满足 128 channels 的 input[128, 1, 28, 28] (或者在 Conv 卷积上入手)
2. 对于原论文使用的 Dense 层, 我不知道怎么用几个 Linear 层完成 fully connected

$$\frac{\partial L}{\partial \mathbf{W}_1} = (\mathbf{W}_2^T (\mathbf{W}_3^T (\ell' \mathbf{s}'_3) \odot \mathbf{s}'_2) \odot \mathbf{s}'_1) \mathbf{x}^T$$

$$\frac{\partial L}{\partial \mathbf{W}_2} = (\mathbf{W}_3^T (\ell' \mathbf{s}'_3) \odot \mathbf{s}'_2) \mathbf{h}_1^T$$

$$\frac{\partial L}{\partial \mathbf{W}_3} = (\ell' \mathbf{s}'_3) \mathbf{h}_2^T$$

梯度下降算法

$$\begin{aligned} \mathbf{s}_1 &= \mathbf{s}_2 = \boldsymbol{\sigma} \\ \boldsymbol{\sigma}' &= \boldsymbol{\sigma}(\mathbf{1} - \boldsymbol{\sigma}) \end{aligned}$$

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

图 11: 4.grad

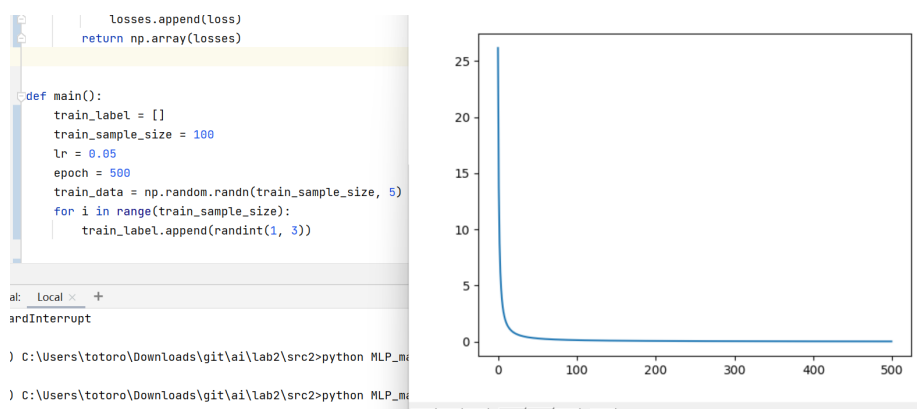


图 12: 4.out

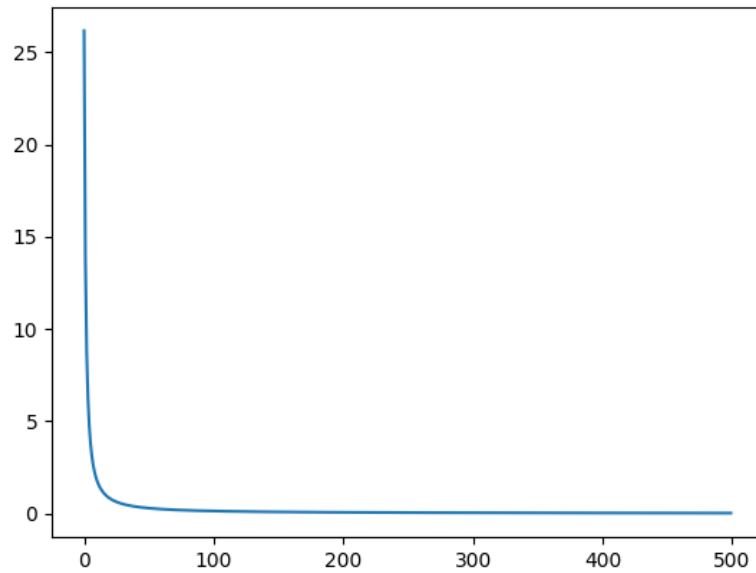


图 13: MLP.out

```

46 self.pre_patch_fc = nn.Conv2d(hidden_dim, hidden_dim, stride=patch_size, kernel_size=patch_size * patch_size,
47                                 torch.nn.modules.conv.Conv2d
48                                 def __init__(self,
49                                     in_channels: int,
50                                     out_channels: int,
51                                     kernel_size: tuple[int, ...],
52                                     stride: tuple[int, ...] = 1,
53                                     padding: str = 0,
54                                     dilation: tuple[int, ...] = 1,
55                                     groups: int = 1,
56                                     bias: bool = True,
57                                     padding_mode: str = 'zeros',
58                                     device: Any = None,
59                                     dtype: Any = None) -> None
60
61 #####
62
63 def forward(self, data):
64     #####
65     # 注意维度的变化
66     mlp1_data = self.pre_patch_fc(data)
67     mlp1_data = self.mlp(mlp1_data)
68
69 #####
70
71 MlpMixer -> __init__
72
73 File "C:\Users\totoro\Downloads\git\
74 mlp1_data = self.pre_patch_fc(data)
75 File "C:\Users\totoro\Downloads\git\
76 return forward_call(*input, **kwargs)
77 File "C:\Users\totoro\Downloads\git\
78 return self._conv_forward(input, se
79 File "C:\Users\totoro\Downloads\git\
80 return F.conv2d(input, weight, bias
81
82 RuntimeError: Given groups=128, weight of size [128, 1, 44, 44], expected input[128, 1, 28, 28] to have 128 channels, but got 1 channel

```

图 14: 5.conv

```

123 # 这里需要调用optimizer.criterion(交叉熵)
124 model = MLP Mixer(patch_size=7, hidden_dim=128, depth=3).to(device) # 参数自己设定, 其中depth必须大于1
125 optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
126 criterion = nn.CrossEntropyLoss()
127 #####

if __name__ == '__main__':

Terminal: Local x +
File "C:\Users\totoro\Downloads\git\ai\lab2\src2\MLP_Mixer.py", line 57, in forward
    mlp1_data = self.pre_patch_fc(data)
File "C:\Users\totoro\Downloads\git\ai\lab2\venv\lib\site-packages\torch\nn\modules\module.py", line 1051, in _call_impl
    return forward_call(*input, **kwargs)
File "C:\Users\totoro\Downloads\git\ai\lab2\venv\lib\site-packages\torch\nn\modules\conv.py", line 443, in forward
    return self._conv_forward(input, self.weight, self.bias)
File "C:\Users\totoro\Downloads\git\ai\lab2\venv\lib\site-packages\torch\nn\modules\conv.py", line 439, in _conv_forward
    return F.conv2d(input, weight, bias, self.stride,
RuntimeError: Given groups=128, weight of size [128, 1, 49, 49], expected input[128, 1, 28, 28] to have 128 channels, but got 1 channel

```

图 15: 5.nn

参考的复现 repo: <https://github.com/lucidrains/mlp-mixer-pytorch>