人工智能第二次实验实验报告

舒文炫

2021 年 7 月 12 日

目录

Chapter 1

实验介绍

1.1 实验内容

本次实验主要分为两个部分，第一部分为传统机器学习，第二部分为深度学习。

- 传统机器学习，包括实现线性分类器，多分类朴素贝叶斯以及支持向量机，使用的数据为鲍鱼数据集，助教已经做完了数据预处理，类别为三种，需要用这三种方法对数据分类
- 深度学习，分为实现多层感知机和复现 MLP_mixer，多层感知机数据为随机生成，主要考察实现前向传播，自动求导，反向传播，梯度下降。时间关系，这部分我只做了多层感知机但结果不对，后面就不分析了。

1.2 实验环境

- 虚拟环境 Python3.6, 不过实测在我原环境 Python3.7 也能运行 (这主要是后面忘记在虚拟环境下运行，但是发现的时候并没有问题)
- Pytorch 版本:torch 1.9.0 torchvision 0.10.0

Chapter 2

实验设计

2.1 part1: 传统机器学习

线性分类器是考虑用线性模型

$$Y = Xw + b + e$$

拟合数据得出结果，这里 X 为数据，Y 为数据类别标签,w,b 为需要训练的参数,e 是误差项，一般我们认为其为正态分布

误差函数为

$$L = \frac{1}{2n}(Xw - b)^2 + \frac{1}{2}\lambda w^2$$

n 为数据个数,λ 为惩罚项，这一项的添加是防止出现共线性的情况，导致模型预测不准，如果使用回归分析里面岭回归的知识，可以直接求得闭式解. 在这里，我们运用梯度下降法求解，默认参数学习率 lr=0.05，迭代 1000 次，$\lambda = 0.001$，在这个误差函数下，梯度为

$$dL = \frac{1}{n}(X^T XY - X^T b) + \lambda w$$

梯度下降法，每次向负梯度方向更新 w 为 $w - lr * dL$, 在学习率取的适当的情况下，可以保证损失函数递减，损失函数越小，这个模型在训练集上表现越好。预测则使用训练出的模型在测试集上判断类别，这里要分类，考虑分到每一个类别是的损失，我们取损失最小类别的为预测结果

朴素贝叶斯考虑用概率模型训练数据，主要用到贝叶斯公式，最简单的情况是

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|A^C)P(A^C)}$$

, 这个公式使得我们在得到每个类别的出现概率，已经在该类别下，每个特征出现的概率之后，可以预测出已知这些特征的情况后，该数据类别是哪一类的概率。故我们的训练，即通过训练数据，给出训练数据的每个类别的概率，由于我们假设各个特征是在给的类别条件独立的故我们只要得到每个特征在每一类下的概率即可。同时考虑到泛化性能，对离散数据使用了拉普拉斯平滑，这个保证了，即使在测试集该特征出现了训练集没有的情况，也能处理、对连续数据，我假设其为正态分布，使用极大似然估计，求出该分布期望和方差的估计，用这个作为连续数据的分布。期望的极大似然估计为样本均值，方差的极大似然估计为 $\frac{n-1}{n}Var(x)$, 不过这不是无偏估计，一般在大样本下我们直接用样本方差来估计分布方差。最后的概率即将这些乘起来，但是考虑到精度问题，概率一般很小，乘起来结果在 python 里面可能为 0. 我们对每一项取了对数，再相加，这是考虑到有 $\log(xy) = \log(x) + \log(y)$. 预测即用我们得出的条件概率计算，概率最高值对应类别为预测类别。

SVM 是考虑用一个分离超平面来分类数据，如果数据线性可分，SVM 的准确率可以很高，可以达到百分之百，这里数据不一定会线性可分，我们要实现的是有软间隔的，即允许一些数据跨过分隔超平面。原型的 SVM 往往在实现时比较困难，这里我们采用其对偶形式，并使用核方法，实现了线性核，高斯核以及多项式核。考虑到会有精度问题，对很小的参数，我们直接置为 0，即不将它对应的数据视为支持向量。对偶形式如下：

$$min\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K(x_i,x_j)+\sum_{i=1}^{N}\alpha_i$$

$$s.t.\sum_{i=1}^{N}\alpha_i y_i=0$$

$$0\le\alpha_i\le C$$

$$i=1,2,3,...,N$$

其中 $K(x_i,x_j)$ 为第 i 个和第 j 个数据算出的核，我们会得到一个 N*N 的核矩阵，α_i 为我们需要求的参数，C 为惩罚参数，C 越大，对误分类的惩罚越大。由上形式可知，此为一个关于 α 的约束二次规划问题。

2.2 part2: 深度学习

这一部分基本没做出来，代码写了但是结果不对，不知道为啥 orz

Chapter 3

实验实现

下面我将展示我的代码来解释具体实现：

3.1 part1: 传统机器学习

3.1.1 线性分类器

```
14      def __init__(self,lr=0.05,Lambda= 0.001,epochs = 1000):
15          self.lr=lr
16          self.Lambda=Lambda
17          self.epochs =epochs
18          self.w=np.array([0 for i in range(8)])
19
20      '''根据训练数据train_features,train_labels计算梯度更新参数W'''
21      def fit(self,train_features,train_labels):
22          self.w=self.w.reshape(-1,1)
23          qwq=np.dot(train_features.T,train_features)
24          qaq=np.dot(train_features.T,train_labels)
25          for i in range(0,self.epochs):
26              df=(np.dot(qwq,self.w)-qaq)/len(train_labels)+self.Lambda*self.w
27              self.w=self.w-self.lr*df
28              '''print(self.w.reshape(1,-1))
29              print(np.linalg.norm(np.dot(train_features,self.w)-train_labels)+self.Lambda*np.linalg.norm(self.w))'''
30
```

图 3.1: linearclassification fit

这里创建一个 LinearClassfication 类，init 方法里面助教并没有给出参数 w 的初始化，这在 fit 训练 w 时将结果传给 predict 有点麻烦，故我在此添加了 w 的初始化这对于这个类也是合理的，因为我们的模型就是要得到 w，需要将其保存起来。在 fit 方法里面，为防止重复计算，我将 X^TX, X^TY 的值分别用 qwq 与 qaq 保存了起来，后面可以直接使用，第 25 到 27 行用 for 循环实现了梯度下降，迭代 self.epochs 次。梯度如前面实验设计里面所示。

```
33
34      '''根据训练好的参数对测试数据test_features进行预测，返回预测结果
35      预测结果的数据类型应为np数组，shape=(test_num,1) test_num为测试数据的数目'''
36      def predict(self,test_features):
37          result=np.dot(test_features,self.w)
38          pred=np.array([0 for i in range(len(result[:,0]))])
39          for i in range(len(result[:,0])):
40              l1=(result[i,0]-1)*(result[i,0]-1)
41              l2=(result[i,0]-2)*(result[i,0]-2)
42              l3=(result[i,0]-3)*(result[i,0]-3)
43              if(l1<l2):
44                  if(l1<l3):
45                      pred[i]=1
46                  else:
47                      pred[i]=3
48              else:
49                  if(l2<l3):
50                      pred[i]=2
51                  else:
52                      pred[i]=3
53          return(pred.reshape(-1,1))
54
```

图 3.2: linearclassification predict

在这个 predict 方法里面, 第 37 行用训练到的 w 在测试机算出每个数据的结果，但这个结果显然不会是 1，2，3 这样的离散值，故我在后面使用 loss 函数判断，如果分到某一类的 loss 函数最小，我就将其分到这一类，但是考虑到 loss 函数中有很多相等的常量，比如惩罚项，在给定了 w 后就都一样了。最重要的项就是计算预测结果到类别的距离这一项，故我只保留了该项，简化计算。39 到 52 行用循环，一个一个分类，分类主要写了一个判断大小模块，l1,l2,l3 分别表示到 1，2，3 的距离，选择距离最小的分到该类。

3.1.2 多分类朴素贝叶斯

```
55  def predict(self,features,featuretype):
56      pred=np.array([0 for x in range(len(features[:,0]))])
57
58      for i in range(len(features[:,0])):
59          k=0
60          p1=float('-inf')
61          p2=0
62          for key1 in self.Pc:
63              p2=math.log(self.Pc[key1],2)
64              for j in range(len(featuretype)):
65                  if(featuretype[j]==1):
66                      p2=p2-1/2*math.log(2*math.pi*self.Pxc[j][key1]["var"],math.e)-math.pow(features[i,j]-self.Pxc[j][key1][
67                  if(featuretype[j]==0):
68                      p2=p2+math.log(self.Pxc[j][key1][features[i,j]])
69              if(p2>p1):
70                  k=key1
71                  p1=p2
72          pred[i]=k
73      return(pred.reshape(-1,1))
```

图 3.3: nbayes fit

这里某些行代码写的比较长，截不全，完整的可以直接看源代码。这个 fit 方法里面，我们先用 counter 方法，统计每个类别的个数，保存的数据类型类似字典，然后计算每个类别出现的概率，存到 Pc 字典里面。

后面对每个 feature，通过 featuretype 来区分时离散型还是连续型，离散型 featuretype=0，此时使用拉普拉斯光滑，将结果存入 Pxc 字典中，这里储存是 Pxcq1"1","2","3" 两个字典嵌套，q1 是该 feature 对应的字典，里面 1,2,3 对应储存了该 feature 在每个类别之中的概率，对连续型 featuretype=1, 此时用正态分布去估计，同理也是存在 Pxc 字典里面，这个结构与离散型稍有不同，Pxcq1"mean","var",q1 也是储存了对应 feature 的字典，里面由于是连续型，只需要保存均值和方差即可。这个 predict 方法，我

```
55  def predict(self,features,featuretype):
56      pred=np.array([0 for x in range(len(features[:,0]))])
57
58      for i in range(len(features[:,0])):
59          k=0
60          p1=float('-inf')
61          p2=0
62          for key1 in self.Pc:
63              p2=math.log(self.Pc[key1],2)
64              for j in range(len(featuretype)):
65                  if(featuretype[j]==1):
66                      p2=p2-1/2*math.log(2*math.pi*self.Pxc[j][key1]["var"],math.e)-math.pow(features[i,j]-self.Pxc[j][key1][
67                  if(featuretype[j]==0):
68                      p2=p2+math.log(self.Pxc[j][key1][features[i,j]])
69              if(p2>p1):
70                  k=key1
71                  p1=p2
72          pred[i]=k
73      return(pred.reshape(-1,1))
74
```

图 3.4: nbayes predict

先初始化了一个 pred 数组，用来储存预测出来的类别，后面一个 for 循环，对每组数据我们去计算这个数据在拥有对应特征下在每个类别里面的概率，去找最大值，这里我就直接用 k 暂时储存最大概率类别的类别，p1 为最大的概率，p2 为当前类别的概率。计算概率是取过了 log 的值，具体原因在实验设计里面说过了。SVM 只需要实现 fit 方法，不过里面的函数返回值就是预测类别。由于使用核方法，需要将这个核矩阵储存起来这里我使用 K1 来储存它，这里的 K 矩阵维数与 K1 一样，元素是 K1 对应元素乘上对应 label，是为了简化后面的代码创建的。这里由于是求解二次规划问题，调用了 cvxopt 包，使用了

```
    def fit(self,train_data,train_label,test_data):
        K=np.array([0.0 for x in range(len(train_data[:,0])*len(train_data[:,0]))])
        K=K.reshape(-1,len(train_data[:,0]))
        K1=np.array([0.0 for x in range(len(train_data[:,0])*len(train_data[:,0]))])
        K1=K1.reshape(len(train_data[:,0]),len(train_data[:,0]))

        for i in range(len(train_data[:,0])):
            for j in range(len(train_data[:,0])):
                K1[i,j]=self.KERNEL(train_data[i,:],train_data[j,:],kernel=self.kernel)
                K[i,j]=train_label[i]*train_label[j]*K1[i,j]



        p=np.array([-1.0 for x in range(len(train_data[:,0]))])
        G=np.array([0.0 for x in range(2*len(train_data[:,0])*len(train_data[:,0]))])
        G=G.reshape(2*len(train_data[:,0]),len(train_data[:,0]))
        h=np.array([0.0 for x in range(2*len(train_data[:,0]))])
        A=train_label.flatten()
        b=0

        for i in range(len(train_data[:,0])):
            G[i,i]=-1.0
            G[i+len(train_data[:,0]),i]=1.0
            h[i]=0.0
            h[i+len(train_data[:,0])]=float(self.C)

        K=cvxopt.matrix(K,(len(train_data[:,0]),len(train_data[:,0])),'d')
        b=cvxopt.matrix(b,(1,1),'d')
        p=cvxopt.matrix(p,(len(train_data[:,0]),1),'d')
        G=cvxopt.matrix(G,(2*len(train_data[:,0]),len(train_data[:,0])),'d')
        h=cvxopt.matrix(h,(2*len(train_data[:,0]),1),'d')
        A=cvxopt.matrix(A,(1,len(train_data[:,0])),'d')

```

图 3.5: SVM 1

cvxopt.solvers.qp(Q,p,G,h,A,b) 函数。对应二次规划问题

$$min\ \frac{1}{2}x^TQx + px \tag{3.1}$$

$$s.t.Gx \leq h \tag{3.2}$$

$$Ax = b \tag{3.3}$$

那么对应到该出，Q=K,p 为全为-1 的向量，若记数据个数为 n，维数为 n，G 的话由于约束问题对每个 α 两边都有约束，G 的行数为 2n，列数为 n。前 n 行与后 n 行都是一个 n 维单位阵，h 前 n 行为 0，后 n 行为 C。约束里面的等式约束为 $\sum_{i=1}^{N}\alpha_i y_i = 0$, 所以转化为 A 为数据的标签，b 为 0，然后比较搞的一点，直接用 numpy 还不行，所以后面我将 numpy 转为了 cvxopt 里面的矩阵。解出来的结果，我们还要处理一下，将过于接近 0 的 α 去掉，和过于接近 C 的 α 都置为 C，这步是因为 python 自身的精度问题，过于小的数和接近 C 的数，最好是不视为支持向量，之前没有考虑这点，得出的准确度很低。后面的部分就是在测试集上预测了，我们使用公式

$$w^* = \sum_{i=1}^{N}\alpha_i^* y_i x_i \tag{3.4}$$

$$b^* = y_j - \sum_{i=1}^{N} y_i\alpha_i K(x_i, x_j) \tag{3.5}$$

其中 α_i^* 为二次规划的解，且第 j 个元素满足 $0 < \alpha_j < C$, 即表明第 j 个为支持向量，当然若考虑鲁棒性，我们可以使用所以支持向量算出来的平均值，不考虑的话，这里用一个支持向量也没有大问题。qwq 是我为了方便创建的一个数组，就是除掉核以外的那些系数，后面要预测，在测试集上我们需要算出新的核矩阵，记为 K2，用这个和 qwq 点积即可。最后调整一下结果的维数，变成列向量输出。

```
85          sol=cvxopt.solvers.qp(K,p,G,h,A,b)
86          for i in range(len(train_data[:,0])):
87              if(sol['x'][i]<self.Epsilon):
88                  sol['x'][i]=0
89              if(sol['x'][i]>self.C-self.Epsilon):
90                  sol['x'][i]=1
91
92
93          b=0.0
94          qwq=np.array([0.0 for x in range(len(train_label))])
95          for i in range(len(train_label)):
96              qwq[i]=train_label[i]*sol['x'][i]
97          count=0
98          for i in range(len(train_data[:,0])):
99              if(sol['x'][i]>0 and sol['x'][i]<self.C):
100                 count=count+1
101                 b=b+train_label[i]-np.dot(qwq,K1[:,i])
102         b=b/count
103         K2=np.array([0.0 for x in range(len(train_data[:,0])*len(test_data[:,0]))])
104         K2=K2.reshape(len(train_data[:,0]),-1)
105         for i in range(len(train_data[:,0])):
106             for j in range(len(test_data[:,0])):
107                 K2[i,j]=self.KERNEL(train_data[i,:],test_data[j,:],kernel=self.kernel)
108         result=np.dot(qwq,K2)+b
109
110         result=result.reshape(-1,1)
111
112         return result
113
```

图 3.6: SVM 1

Chapter 4

实验结果与分析

4.1 part1: 传统机器学习

这一部分我会贴上我的输出

4.1.1 线性分类器

```
(base) G:\绸带\LAB2_for_students\src1>python linearclassification.py
train_num: 3554
test_num: 983
train_feature's shape:(3554, 8)
test_feature's shape:(983, 8)
Acc: 0.6174974567650051
0.6214099216710182
0.6032786885245902
0.6347305389221557
macro-F1: 0.6198063830392546
micro-F1: 0.6174974567650051
```

图 4.1: linearclassification output

这是线性分类器的输出结果，可以看到总的预测准确度达到了 0.61，对每个类别的准确度也有 0.6 朝上，micro-F1 和 macro-F1 也有 0.6 以上，说明模型实现没有问题，结果比较理想

4.1.2 多分类朴素贝叶斯

```
(base) G:\绸带\LAB2_for_students\src1>python nBayesClassifier.py
train_num: 3554
test_num: 983
train_feature's shape:(3554, 8)
test_feature's shape:(983, 8)
Acc: 0.5198372329603256
0.33834586466165417
0.5924006908462867
0.45387453874538747
macro-F1: 0.46154036475110943
micro-F1: 0.5198372329603256
```

图 4.2: nbayes output

这是朴素贝叶斯的输出结果，可以看到总的预测准确度达到 0.5，比线性分类器稍差，也有可能还是精度有一定影响，或者假设为正态分布没有特别合理，但是结果也还可以。总体实现没有太大问题。

4.1.3 SVM

```
12: -1.8757e+03 -1.8783e+03  3e+00  6e-07  5e-14
13: -1.8760e+03 -1.8779e+03  2e+00  3e-07  4e-14
14: -1.8763e+03 -1.8776e+03  1e+00  2e-07  4e-14
15: -1.8765e+03 -1.8774e+03  9e-01  1e-07  4e-14
16: -1.8767e+03 -1.8772e+03  5e-01  3e-08  5e-14
17: -1.8768e+03 -1.8771e+03  3e-01  1e-08  5e-14
18: -1.8769e+03 -1.8769e+03  1e-01  1e-09  5e-14
19: -1.8769e+03 -1.8769e+03  5e-02  2e-10  5e-14
20: -1.8769e+03 -1.8769e+03  7e-03  2e-11  5e-14
21: -1.8769e+03 -1.8769e+03  2e-04  1e-13  5e-14
Optimal solution found.
Acc: 0.6561546286876907
0.7539503386004515
0.5740498034076016
0.6815789473684212
macro-F1: 0.6698596964588247
micro-F1: 0.6561546286876907
```

图 4.3: SVM gauss output

```
14: -1.9271e+03 -1.9271e+03  4e-03  2e-09  5e-13
15: -1.9271e+03 -1.9271e+03  4e-05  2e-11  5e-13
Optimal solution found.
Acc: 0.602238046795524
0.6967509025270758
0.21973094170403587
0.7246376811594203
macro-F1: 0.547039841796844
micro-F1: 0.602238046795524
```

图 4.4: SVM linear output

这三张图分别是选择高斯核，线性核，多项式核的结果，前面是二次规划迭代过程，很长就没有一一截下来，截下来也没有太大意义，可以看到高斯核准确率 0.65，各个类别的 f1-score 也很不错，线性核低一点只有 0.60，多项式核准确率 0.64，比线性核好，比高斯核差一点。总体而言这三种核的准确率都很好，所以 SVM 实现也没有问题。

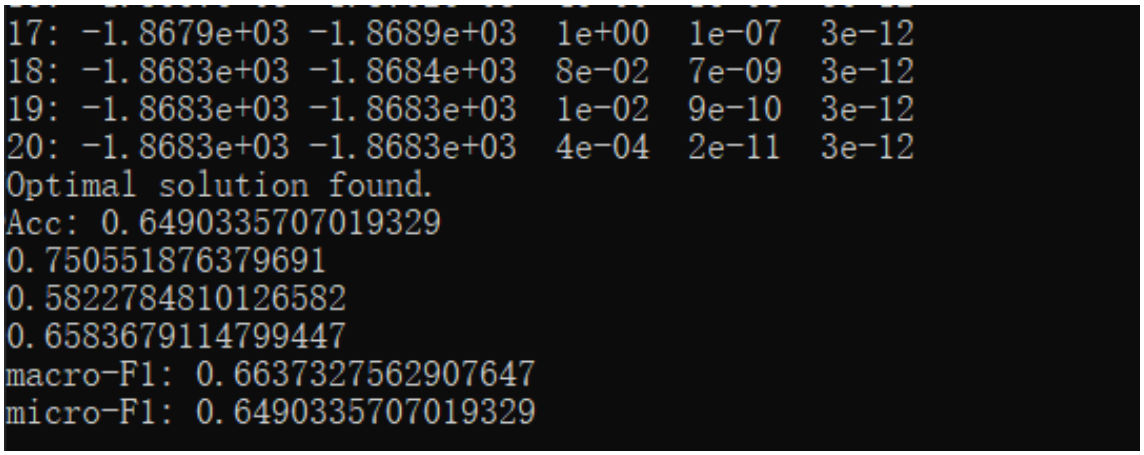

图 4.5: SVM poly output

Chapter 5

实验总结

本次实验，我接触到了传统机器学习的各种经典方法，并尝试实现了他们，这让我对机器学习理解更为深刻，深度学习过于困难，难以完成 orz，我好菜 www