Lab2实验报告

实验内容

本次实验包含传统机器学习与深度学习两部分。在传统机器学习部分，需要实现`linearclassification`，`nBayesClassifier`，`SVM`三类分类器。而深度学习部分， 要求手写感知机模型并进行反向传播。然后复现MLP-Mixer

实验环境

实验部分需要使用python=3.6，深度学习部分要求使用pytorch=1.8.1，torchvision=0.9.1完成，实验部分使用CPU足够训练，如果想体验GPU的速度可以使用colab。

传统机器学习

linearclassification

这部分只需实现训练函数和预测函数。

训练函数 。 可以用最小二乘法的办法， 也可以用梯度下降的方法来求w* 。我采用了最小二乘法的办法 。 直接解如下方程。

$$min_w(Xw-y)^2+\lambda\|w\|^2$$

核心代码如下

```
        ## 最小二乘法 计算参数
        assert train_features.shape[0] == train_labels.shape[0], "数据集有问题"
        X = np.hstack([np.ones((len(train_features), 1)), train_features]) # 前一列加上1
        tmp = X.T.dot(X) + self.Lambda * np.identity(X.shape[1])
        self.w = (np.linalg.inv(tmp)).dot(X.T).dot(train_labels)
```

预测函数 。

这个只需要，用训练的参数， 计算测试数据集。 将计算结果映射到预测结果即可。

```
def predict(self,test_features):
        ''''
        需要你实现的部分
        '''
        features_mean = []
        features_var = []
        for i in range(0,test_features.shape[1]):
            features_mean.append(np.mean(test_features[:,i]))
            features_var.append(np.var(test_features[:,i]))
        for i in range(0,test_features.shape[0]):
            for j in range(0,test_features.shape[1]):
                test_features[i][j] = float((test_features[i][j]-
features_mean[j])/features_var[j])
        test_X = np.hstack([np.ones((len(test_features), 1)), test_features]) # 前一列加
上1
        pred_init = test_X.dot(self.w)
        print(pred_init)
        for i in range(0,pred_init.shape[0]):
            if pred_init[i] < 1.5 :
                pred_init[i] = 1
            elif pred_init[i] > 2.5 :
                pred_init[i] = 3
            else:
                pred_init[i] = 2
        return pred_init
```

运行结果：

```
>>> runfile('/Users/zhangwanlin/PycharmProjects/pythonProject/AILAB2/src1/linearclassification.py',
  wdir='/Users/zhangwanlin/PycharmProjects/pythonProject/AILAB2/src1')
train_num: 3554
test_num: 983
train_feature's shape:(3554, 8)
test_feature's shape:(983, 8)
Acc: 0.6378433367243134
0.7300771208226221
0.6203904555314534
0.6076335877862595
macro-F1: 0.6527003880467784
micro-F1: 0.6378433367243134
```

nBayesClassifier

训练部分

计算先验概率和条件概率。

但是数据分为连续数据和离散数据 。 对于离散数据，统计个数计算条件概率 。 对于连续数据， 计算值μ和方差σ^2 。

核心代码

```
        ## 构建 每种先验的条件概率
        for key in self.Pc.keys():
            # 计算每种类别在数据集中出现的概率
            self.Pc[key] /= row_num
            # 构建self.condition_prob中的key
            self.Pxc[key] = {}
            for i in range(col_num):
                #### 每个属性一个字典
                self.Pxc[key][i] = {}
                if featuretype[i] == 0:  ## 如果是离散的 , 则每个
                    for k in np.unique(traindata[:, i], axis=0):
                        ### 统计数字
                        self.Pxc[key][i][k] = 0
                if featuretype[i] == 1:  ## 如果是连续的 , 则每个求sigma ,
                    ## 求值为 key 的集合
                    traindata_key = []
                    for ith in range (0,trainlabel.shape[0]):
                        if trainlabel[ith][0] == key:
                            traindata_key.append(traindata[ith][i])
                    ## 求mu,sigma
                    mu = np.mean(traindata_key)
                    sigma = np.std(traindata_key)
                    self.Pxc[key][i][0] = mu
                    self.Pxc[key][i][1] = sigma
```

预测部分

分别计算每种预测， 出现该结果的概率 。 选取最大的即可 。

```
        #对每条测试数据都进行预测
        for ith,f in enumerate(features):
            #可能的类别的概率
            prob = np.zeros(len(self.Pc.keys()))
            ii = 0
            for label,label_prob in self.Pc.items():
                #计算概率
                prob[ii] = math.log(label_prob)
                for j in range(0,len(features[0])): ## j个属性
                    if featuretype[j] == 0:
                        last_prob = self.Pxc[label][j][f[j]]
                    else:
```

```
                        last_prob = NaiveBayes.gauss(f[j],self.Pxc[label][j]
[0],self.Pxc[label][j][1])
                    prob[ii] += math.log(last_prob)
                ii += 1
            #取概率最大的类别作为结果
            result.append(list(self.Pc.keys())[np.argmax(prob)])
        print(np.array(result).astype(int).reshape(-1,1))
        return np.array(result).astype(int).reshape(-1,1)
```

实验结果。

```
>>> runfile('/Users/zhangwanlin/PycharmProjects/pythonProject/AILAB2/src1/nBayesClassifier.py',
 wdir='/Users/zhangwanlin/PycharmProjects/pythonProject/AILAB2/src1')
train_num: 2581
test_num: 983
train_feature's shape:(2581, 8)
test_feature's shape:(983, 8)
Acc: 0.6103763987792472
0.7112810707456978
0.45921450151057397
0.6709346991037132
macro-F1: 0.6138100904533283
micro-F1: 0.6103763987792472
```

SVM

只需找到支持向量 。 而支持向量直接使用cvxopt库可以求解 。系数矩阵参考助教提供的博客即可。 然后预测部分，直接计算$w^* * x + b$ 的值 。 不过点乘需要用核函数计算。

求解支持向量

```
        P = cvxopt.matrix(P)
        # q
        q = cvxopt.matrix(np.ones((row_num, 1)) * -1)
        # G
        # I 单位阵
        I = np.diag(np.ones((row_num)))
        G = cvxopt.matrix(np.vstack((-1 * I, I)))
        h = cvxopt.matrix(np.vstack((np.zeros((row_num, 1)), self.C *
np.ones((row_num, 1)))))
        # A
        y = cvxopt.matrix(np.array(train_label, float), (row_num, 1))
        A = y.T
        # b为0
        b = cvxopt.matrix(0.0)
        opt = cvxopt.solvers.qp(P, q, G, h, A, b)
        # 求出x
        x = np.ravel(opt['x'])
```

```
            # 求支持向量
            ## 支持向量的序号
            sv = (x > self.Epsilon) * (x < self.C)
```

预测部分的计算。

```
            pred = np.zeros(test_data.shape[0])
            for j in range(test_data.shape[0]):
                sum = 0.0
                for i in range(len(x[sv])):
                    sum += x[sv][i] * train_label[sv][i] * self.KERNEL(train_data[sv][i],
test_data[j], self.kernel)
                pred[j][0] = sum + b

            return pred
```

由于训练较慢，减少了一些训练集数据 。

这个是线性核的结果。

```
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0: -1.6260e+03 -7.2467e+03  4e+04  3e+00  5e-13
 1: -1.0993e+03 -5.0406e+03  5e+03  2e-01  6e-13
 2: -1.1653e+03 -1.7962e+03  7e+02  2e-02  5e-13
 3: -1.3145e+03 -1.5447e+03  2e+02  6e-03  5e-13
 4: -1.3358e+03 -1.5170e+03  2e+02  4e-03  5e-13
 5: -1.3549e+03 -1.4889e+03  1e+02  3e-03  5e-13
 6: -1.3740e+03 -1.4606e+03  9e+01  2e-03  5e-13
 7: -1.3904e+03 -1.4369e+03  5e+01  7e-04  5e-13
 8: -1.3979e+03 -1.4261e+03  3e+01  4e-04  5e-13
 9: -1.4047e+03 -1.4168e+03  1e+01  1e-04  5e-13
10: -1.4071e+03 -1.4136e+03  7e+00  6e-05  5e-13
11: -1.4091e+03 -1.4110e+03  2e+00  1e-05  6e-13
12: -1.4094e+03 -1.4107e+03  1e+00  5e-06  5e-13
13: -1.4099e+03 -1.4102e+03  3e-01  1e-06  6e-13
14: -1.4100e+03 -1.4100e+03  1e-02  5e-14  6e-13
15: -1.4100e+03 -1.4100e+03  1e-03  8e-14  6e-13
Optimal solution found.
Acc: 0.6541200406917599
0.7589285714285714
0.5561643835616439
0.6852791878172588
macro-F1: 0.6667907142691579
micro-F1: 0.6541200406917599
```

多项式核的结果。

```
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0: -1.0848e+03 -5.6450e+03  3e+04  4e+00  3e-12
 1: -7.1542e+02 -4.2084e+03  6e+03  4e-01  3e-12
 2: -7.1752e+02 -1.3603e+03  7e+02  2e-02  2e-12
 3: -8.1513e+02 -1.0824e+03  3e+02  5e-03  2e-12
 4: -8.3665e+02 -1.0450e+03  2e+02  4e-03  2e-12
 5: -8.5418e+02 -1.0168e+03  2e+02  6e-04  3e-12
 6: -8.6877e+02 -9.8226e+02  1e+02  3e-04  2e-12
 7: -8.8064e+02 -9.5579e+02  8e+01  2e-04  2e-12
 8: -8.8928e+02 -9.3637e+02  5e+01  8e-05  3e-12
 9: -8.9430e+02 -9.2669e+02  3e+01  4e-05  2e-12
10: -8.9849e+02 -9.1768e+02  2e+01  9e-06  3e-12
11: -9.0088e+02 -9.1392e+02  1e+01  4e-06  3e-12
12: -9.0108e+02 -9.1354e+02  1e+01  3e-06  2e-12
13: -9.0389e+02 -9.0969e+02  6e+00  1e-06  3e-12
14: -9.0506e+02 -9.0805e+02  3e+00  4e-07  3e-12
15: -9.0570e+02 -9.0720e+02  1e+00  6e-08  3e-12
16: -9.0621e+02 -9.0661e+02  4e-01  9e-09  3e-12
17: -9.0638e+02 -9.0642e+02  4e-02  7e-10  3e-12
18: -9.0640e+02 -9.0640e+02  2e-03  2e-11  3e-12
19: -9.0640e+02 -9.0640e+02  3e-05  2e-13  3e-12
Optimal solution found.
Acc: 0.6480162767039674
0.7467248908296943
0.5718015665796344
0.6657681940700808
macro-F1: 0.6614315504931365
micro-F1: 0.6480162767039674
```

深度学习

手写感知机模型并进行反向传播

代码框架

```
class MLP4:
    def __init__(self, input_size, hidden_num1, hidden_num2, num_classes):
        # 隐层参数设置 ， 初始化为随机值
    ## 激活函数
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def softmax(self, x):
        return np.exp(x) / np.sum(np.exp(x), axis=0)

    def dsigmoid(self, s):
        return s * (1 - s)

    # 正向传播 ， 按步骤计算 ,并计算前向信息
    def forward(self, data):

    def predict(self, data):
        return self.forward(data).argmax(0)

    ## 静态函数 ， 计算交叉熵
    def cross_entrop(self, a1, a2):

    def loss(self, data, label):
        return self.cross_entrop(label, self.forward(data))

    ## BP算法
    def BP(self, data, label):
        # 从最后一层开始更新参数
        ## 计算，每一层的新值

    ## 梯度下降算法
    def GradientDescent(self, eta=0.1):
        return
```

主要核心部分是实现 BP和梯度下降 。

BP的算法逻辑如下

function BACK-PROP-LEARNING (*examples, network, hyperparameters*) **returns** a neural network
inputs: *examples*, a set of examples with input vector **x** and output vector **y**
network, a neural network with layers, weights, biases, and activation functions
hyperparameters, consist of learning rate **LR**, number of **epochs**, batch size **BS**, and momentum **α**
local variables: *velocity*, the starting velocity for each weight and bias
gradientSum, a matrix that collects the gradients for each bias and weight matrix

velocity ← *set to 0*
for each *epoch* **do**

/* *Shuffle examples*/
/* *Segment examples into batches of size* ***BS*** */

gradientSum ← *set to 0*
for each example (**x**, **y**) **in** batch **do**
/* *Propagate the inputs forward and compute the output*/
for each *weight and bias* **in** *network* **do**
/* *Propagate error backward and compute gradients*/
gradientSum ← *collect gradients*

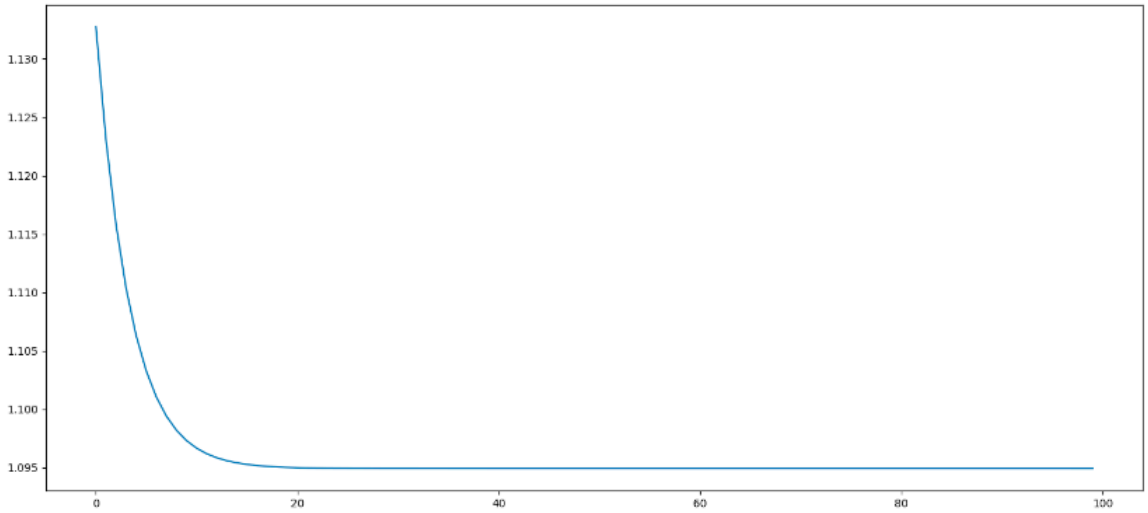
for each weight and bias **in** network **do**
velocity ← *update using gradientSum, LR, and α*
weight ← *update using velocity*
bias ← *update using velocity*

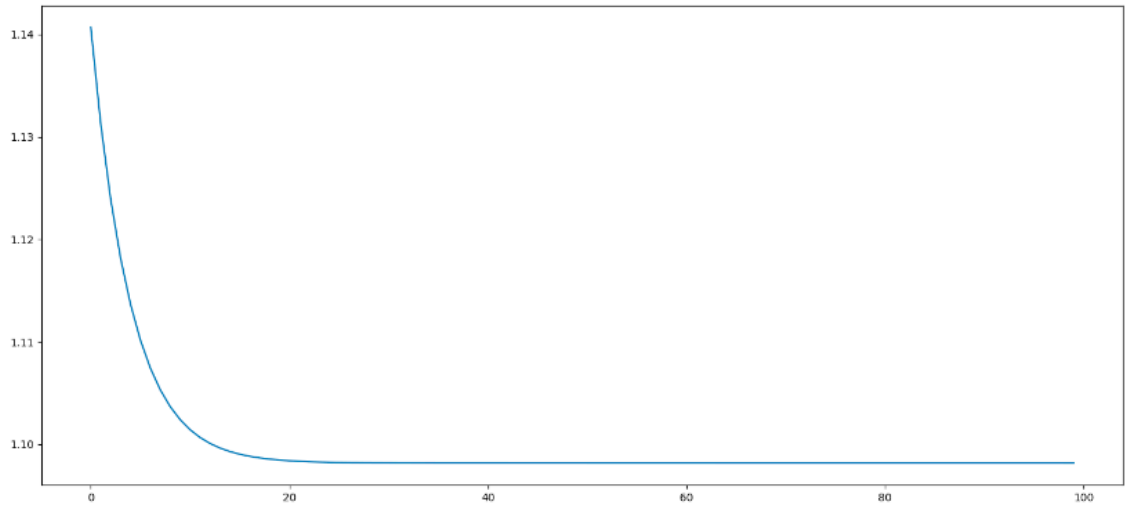
梯度下降的算法。只需按照博客给出的公式计算即可 。

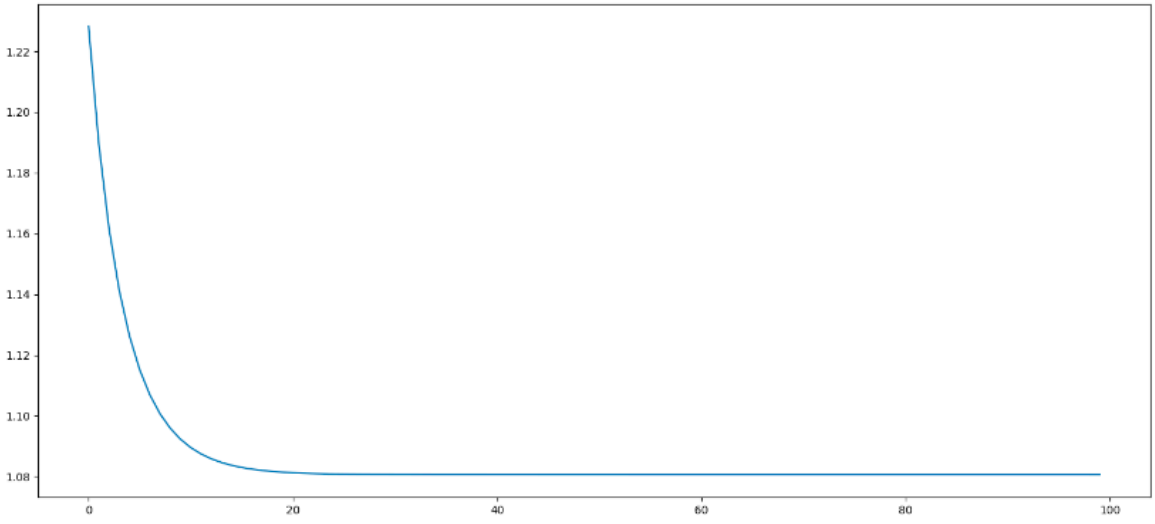
代码

```
        for i in range(3):
            s = "delta_b" + str(i + 1)
            self.b[i] = self.b[i] - eta * self.diff_info[s]
        for i in range(3):
            s = "delta_w" + str(i + 1)
            self.w[i] = self.w[i] - eta * self.diff_info[s]
```

训练的学习曲线


1.130
1.125
1.120
1.115
1.110
1.105
1.100
1.095
0
20
40
60
80
100


1.14
1.13
1.12
1.11
1.10
0
20
40
60
80
100


1.22
1.20
1.18
1.16
1.14
1.12
1.10
1.08
0
20
40
60
80
100

实验总结

本实验是第一次，进行机器学习的实验。学到的很多东西。 包括机器学习的代码框架结构， 用于机器学习的很多库， 机器学习的准确率问题， 参数设置调整。当然这中间也遇到很多的问题。包括库不熟悉， 模型准确率只有0.6等等的问题。但是毕竟是第一次，收获还是很大的。