

高斯贝叶斯分类

1. 实验说明

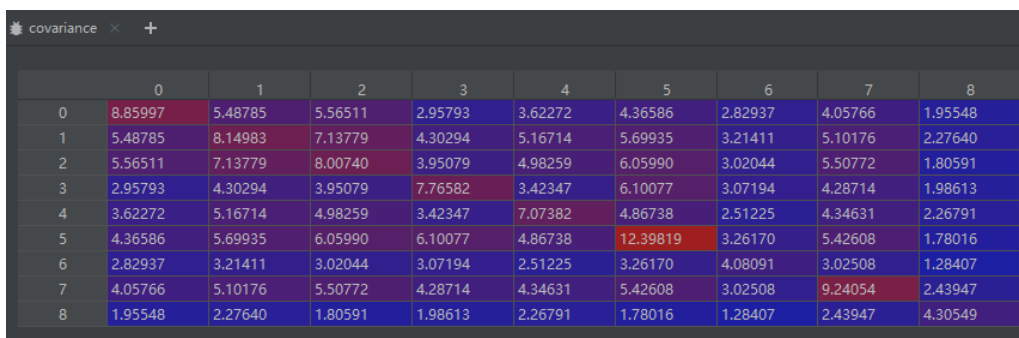
本实验先采用 PCA（主成分分析）对数据集进行降维处理，接着使用训练集训练高斯贝叶斯分类器，之后在测试集上进行测试，并根据测试结果计算 NMI 值。训练集为数据集的前 20%，其余为测试集。

2. 实验方法

2.1 PCA 降维

实验中首先采用主成分分析方法，对所有样本（包括训练集和测试集）进行降维处理，但降维所采用的变换矩阵，是根据训练集的数据分布得到。具体步骤为：

1. 根据训练集中的样本，计算每个属性的均值。
2. 将训练集中样本进行平移变换，使得平移之后属性的均值为零。
3. 将平移之后的样本矩阵的转置乘以该矩阵，注意到在样本矩阵中每一行代表一个数据样本，每一列对应一个属性，因此得到的矩阵即为协方差矩阵。计算所得到的协方差矩阵如图 1.1 所示。原始数据中，每行最后一个元素代表类别标签，因此，协方差矩阵为 9×9 的对称方阵。



	0	1	2	3	4	5	6	7	8
0	8.85997	5.48785	5.56511	2.95793	3.62272	4.36586	2.82937	4.05766	1.95548
1	5.48785	8.14983	7.13779	4.30294	5.16714	5.69935	3.21411	5.10176	2.27640
2	5.56511	7.13779	8.00740	3.95079	4.98259	6.05990	3.02044	5.50772	1.80591
3	2.95793	4.30294	3.95079	7.76582	3.42347	6.10077	3.07194	4.28714	1.98613
4	3.62272	5.16714	4.98259	3.42347	7.07382	4.86738	2.51225	4.34631	2.26791
5	4.36586	5.69935	6.05990	6.10077	4.86738	12.39819	3.26170	5.42608	1.78016
6	2.82937	3.21411	3.02044	3.07194	2.51225	3.26170	4.08091	3.02508	1.28407
7	4.05766	5.10176	5.50772	4.28714	4.34631	5.42608	3.02508	9.24054	2.43947
8	1.95548	2.27640	1.80591	1.98613	2.26791	1.78016	1.28407	2.43947	4.30549

图 2.1 训练集的协方差矩阵

实验中计算协方差矩阵的代码如下：

```
1. def calc_cov(data): # 传入 list, 返回协方差矩阵
2.     mean_list = np.mean(data, axis=0) # 计算每列平均值
```

```

3.     data_copy = np.array([[line[i]-
mean_list[i] for i in range(len(mean_list))] for line in data]) # 使均值为
零
4.     covariance = np.dot(data_copy.T, data_copy) / (len(data_copy)-1) # 计算
协方差矩阵
5.     return covariance

```

4. 在得到协方差矩阵之后，利用 numpy 中的 `linalg.eig()` 函数，可以直接得到特征值和特征向量。计算得到的特征值和特征向量如图 1.2 和图 1.3 所示。注意在图 1.3 中，每一列对应一个特征向量

	0
0	41.13164
1	7.34424
2	0.81653
3	5.16900
4	2.01041
5	2.33297
6	3.19330
7	4.02745
8	3.85644

图 2.2 计算得到的特征值

	0	1	2	3	4	5	6	7	8
0	0.33016	-0.51328	0.04008	0.42229	-0.25109	0.21876	-0.27910	0.44059	-0.25435
1	0.39194	-0.25139	0.66145	0.09245	0.13992	-0.39364	0.31397	-0.13935	0.21118
2	0.39176	-0.23092	-0.72201	0.13867	0.06637	-0.31680	0.26947	-0.27385	-0.01705
3	0.31276	0.42174	-0.07612	-0.16716	-0.41439	-0.06438	0.40453	0.56985	0.16231
4	0.31698	-0.09810	-0.01610	-0.16746	-0.27607	0.55577	-0.11908	-0.36878	0.57125
5	0.43240	0.65037	0.06600	0.41859	0.13118	-0.02398	-0.37500	-0.19389	-0.12029
6	0.21144	0.00254	-0.05665	-0.07752	0.76925	0.46242	0.23465	0.29173	0.01791
7	0.36268	-0.02781	0.11089	-0.63752	-0.09081	0.06495	-0.05844	-0.18827	-0.63059
8	0.14966	-0.09368	-0.11687	-0.38762	0.21871	-0.40642	-0.61155	0.30429	0.35401

图 2.3 计算得到的特征矩阵

PCA 中是选取前 k 个最大的特征值对应的特征向量为了选择保留的维度个数，但是 k 的个数可能要根据经验得到。在这里，通过给定一个比例系数 `ratio` 来决定 k 。该比例系数是从大到小选择 k 个特征值，这 k 个特征值占有所有特征值之和的百分比。即

$$\sum_{i=1}^{i=k} \lambda_i \geq ratio * \sum \lambda_i \quad \sum_{i=1}^{i=k-1} \lambda_i < ratio * \sum \lambda_i$$

用选定的特征向量组成投影矩阵。具体代码如下

```

1. def top_ratio(mat, ratio=0.95): # 从矩阵中选出最大的 k 个特征值对应的特征向量，
   并将这些特征向量组成矩阵，其中前 k 个特征值占有所有特征值之和的比例为 ratio
2.     e_vals, e_vecs = np.linalg.eig(mat) # 列向量才是特征向量
3.     sort_indices = np.argsort(e_vals) # 返回一个同等维度的 list，其中中存放可以
   将传入列表按从小到大排列时的下标
4.     ratio_list = np.cumsum(e_vals, axis=0) / np.cumsum(e_vals, axis=0)[-1]
5.     k = np.where(ratio_list == [e_val for e_val in ratio_list if e_val > rat
   io][0])[0][0] # 得到 k 值
6.     print("保留前 %d 个最大特征值对应的特征向量" % (k+1))
7.     return e_vals[sort_indices[-1:-k-2:-1]], e_vecs[:, sort_indices[-1:-k-
   2:-1]]

```

5. 用选定的特征向量组成的投影矩阵，对全部数据进行投影。此部分代码如下。

```

1. def dim_reduction(data, e_vecs):
2.     e_vecs_norm = [[t / np.sum(np.square(line)) for t in line] for line in
   e_vecs.T] # 标准化
3.     data = [np.dot(e_vecs_norm, data[i]) for i in range(len(data))] # 将数
   据样本进行投影
4.     return data

```

2.2 训练

训练的步骤如下：

1. 按样本类别进行分类，代码如下，传入的 data 为训练集，每一行对应一个训练样本，label 为一维标签列表，存放的是训练集中对应位置的标签

```

1. def separate_by_class(data, label): # 按标签进行分类
2.     separated_class = {}
3.     for i in range(len(data)):
4.         if label[i] not in separated_class:
5.             separated_class[label[i]] = []
6.             separated_class[label[i]].append(data[i])
7.     return separated_class

```

2. 在训练样本中，对于属于同一个标签的数据，计算各个属性的均值和标准差，返回存放均值列表和标准差列表。代码如下。

```

1. # 提取特征属性，对属于同一个标签的，计算均值和方差

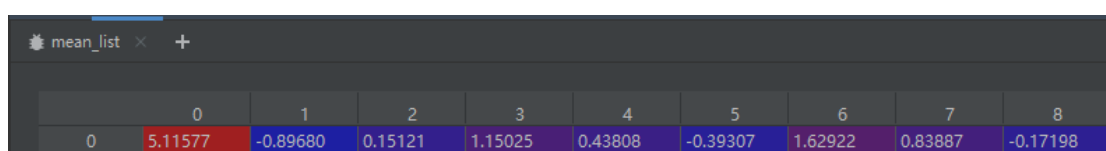
```

```

2. def cal_sta(data): # 此处传入的 data, 为训练集中, 属于同一个标签的数据
3.     # 将传入的 list 转化为 ndarray 数组
4.     mean_list = np.mean(np.array(data), axis=0)
5.     std_dev_list = np.sqrt(np.sum([np.square([data[line][i] - mean_list[i]
6.         for i in range(len(mean_list))]) for line in range(len(data))], axis=0) / (
7.         len(data) - 1))
8.     # print("mean_list: ", mean_list)
9.     # print("std_dev_list: ", std_dev_list)
10.    return mean_list, std_dev_list

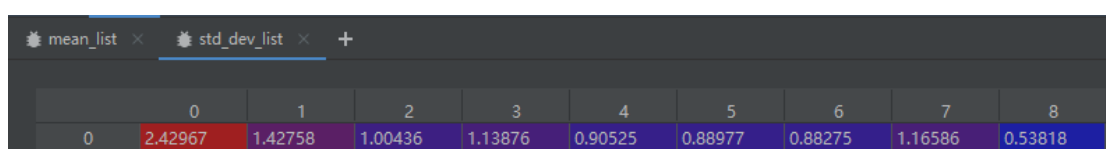
```

当选择的 ratio 为 0.95, 此时所有的特征向量都被保存下来, 此时 mean_list 和 std_dev_list 的值如图 2-4 和 2-5 所示。



	0	1	2	3	4	5	6	7	8
0	5.11577	-0.89680	0.15121	1.15025	0.43808	-0.39307	1.62922	0.83887	-0.17198

图 2.4 mean_list 取值



	0	1	2	3	4	5	6	7	8
0	2.42967	1.42758	1.00436	1.13876	0.90525	0.88977	0.88275	1.16586	0.53818

图 2.5 std_dev_list 取值

3. 计算训练样本集的各种属性, 对每个标签, 有属于该标签样本的各个属性的均值和方差。均值和方差均采用一维列表存储, 同时, 由均值和方差两个列表构成的二维列表, 为字典中对应该标签的 value 值。该部分代码如下。当 ratio 为 0.95 时, summary 的内容如图 2.6 所示。

```

1. def summarize(separated_class): # 返回一个字典, 字典的 key 为标签, summary 包
2.     summary = {}
3.     for key in separated_class:
4.         summary[key] = []
5.         mean_list, std_dev_list = cal_sta(separated_class[key])
6.         summary[key].append({"mean_list": mean_list})
7.         summary[key].append({"standard_deviation_list": std_dev_list})
8.     return summary

```

```
{-1: [{'mean_list': array([ 5.11576623, -0.89680125,  0.15121473,
 1.15024789,  0.43808272,
 -0.3930731 ,  1.62922185,  0.83887082, -0.17198122])},
{'standard_deviation_list': array([2.42966681, 1.42757707,
 1.00436474, 1.13876215, 0.90524815,
 0.88976809, 0.88274837, 1.16585718, 0.53818403])}]}, 1:
[{'mean_list': array([16.25656622, -1.19559606,  0.33166728,
 1.33977778,  0.11829503,
 -0.84610155,  1.7319716 ,  0.82006638, -0.24186929])},
{'standard_deviation_list': array([3.97670225, 3.74505668,
 3.2277954 , 2.7341168 , 2.76511139,
 2.47432092, 2.07389707, 1.68952261, 1.21804016])}]}
```

图 2.6 ratio=0.95 时 summary 的内容

2.3 预测

将训练得到的结果在测试集（数据集的后 80%部分）上进行测试，测试部分的代码如下。

```
1. def cal_class_label(summary, input_vector): # 对每个属性的每个取值，计算概率
2.     probability = {}
3.     key_list = list(summary.keys())
4.     for key in key_list:
5.         probability[key] = 1
6.         for i in range(len(input_vector)):
7.             probability[key] *= cal_gauss_prob(input_vector[i], summary[key]
8. [0]["mean_list"][i], summary[key][1]["standard_deviation_list"][i])
9.     label = max(probability, key=probability.get)
10.    return label
```

传入的参数为在 2.2 中计算得到的字典 summary 和单个的用于测试的样本（即 input_vector），返回的结果为预测的标签。此处调用了用来计算概率密度的函数（用概率密度来代替概率，因为只考虑相对之间概率的大小），该函数代码如下。

```
1. def cal_gauss_prob(x, mean, stdev): # 计算高斯概率密度
2.     exponent = math.exp(-math.pow(x-mean, 2) / (2 * math.pow(stdev, 2)))
3.     return exponent / math.sqrt(2 * math.pi * stdev)
```

对于所有测试集样本，得到预测标签之后，计算互信息值。该函数中，首先

对训练集预测产生的标签和实际的分类标签进行去重处理，生成新的标签列表，目的是对标签进行编号。

```
1. def get_nmi(test_label, calc_label): # 传入两个标签列表，根据这两个标签列表来计算互信息
2.     test_label_rm_duplicate = list(set(test_label)) # 测试训练集的标签去除重复
3.     calc_label_rm_duplicate = list(set(calc_label)) # 预测得到的表标签去除重复
4.     dic_test = dict(zip(test_label_rm_duplicate, range(len(test_label_rm_duplicate)))) # 生成字典，主要是想对标签进行编号
5.     dic_calc = dict(zip(calc_label_rm_duplicate, range(len(calc_label_rm_duplicate))))
6.     p_calc_test = np.array([[0 for i in range(len(calc_label_rm_duplicate)) for j in range(len(test_label_rm_duplicate))]) # 计算联合概率分布，列之和得到实际概率分布，行之和得到训练模型判断的结果概率分布
7.     for i in range(len(test_label)):
8.         p_calc_test[dic_calc[calc_label[i]]][dic_test[test_label[i]]] += 1 # 联合概率分布中，对应位置的次数加以
9.     p_calc_test = p_calc_test / len(test_label)
10.    p_calc = np.sum(p_calc_test, axis=1) # 行之和，对于每一个标签，得到预测结果为该标签的比例
11.    p_test = np.sum(p_calc_test, axis=0) # 列之和，对于每一个标签，得到测试集中该标签的比例
12.    h_calc = -sum([i * math.log(i, 2) for i in p_calc]) # 计算训练结果的信息熵
13.    h_test = -sum([i * math.log(i, 2) for i in p_test]) # 计算实际结果的信息熵
14.    tmp = sum([sum([p_calc_test[i][j] * (math.log(p_calc_test[i][j], 2) - math.log(p_calc[i] * p_test[j], 2)) for i in range(len(calc_label_rm_duplicate))]) for j in range(len(test_label_rm_duplicate))])
15.    nmi = 2 * tmp / (h_calc + h_test)
16.    return nmi
```

3. 实验结果

3.1 主函数

主函数中代码如下，实验中可以调节 ratio 进行调节，从而找到在合适值时，NMI 值可以达到最大，当两次计算所得 NMI 值不同时，打印输出信息。

```
1. if __name__ == "__main__":
```

```

2.     Data, Label = load_file("breast.txt") # 加载文件，得到数据样本和对应标签
3.     Train_Size = int(len(Data) * 0.2) # 表示训练集的数目
4.     Train_Data = Data[0:Train_Size] # 用于训练的样本
5.     Train_Label = Label[0:Train_Size] # 训练的样本标签
6.     Test_Label = Label[Train_Size+1:]
7.     Cov = calc_cov(Train_Data) # 计算协方差矩阵
8.     Last_NMI = 0
9.     for Ratio in [_/100 for _ in range(50, 100)]:
10.         E_Vals, E_Vecs = top_ratio(Cov, Ratio) # 根据特征值数据分布，得到前 k
            个最大特征值和对应的特征向量构成的特征矩阵，k 根据 ratio 计算得到
11.         Data_Pac = dim_reduction(Data, E_Vecs) # 得到降维之后的数据，包括训练
            集和测试集
12.         Train_Data_Pac = Data_Pac[0:Train_Size] # 降维之后的训练数据
13.         Test_Data = Data_Pac[Train_Size+1:] # 降维之后的测试数据
14.         Separated_Class = separate_by_class(Train_Data_Pac, Train_Label)
15.         Summary = summarize(Separated_Class)
16.         Calc_Label = np.array([cal_class_label(Summary, Input_Vector) for In
            put_Vector in Test_Data])
17.         NMI = get_nmi(Test_Label, Calc_Label)
18.         if Last_NMI != NMI:
19.             print("保留的特征值之和占有所有特征值之和的比例：%.2f    计算所得 NMI
                为：%.5f" % (Ratio, NMI))
20.         Last_NMI = NMI

```

3.2 运行结果

具体运行结果如图 3.1 所示。可以看出当 ratio 为 0.59–0.70 时，可以使 NMI 达到最大值，此时仅保留了两个特征向量。

保留的特征值之和占有所有特征值之和的比例：0.50	计算所得NMI为：0.78327
保留的特征值之和占有所有特征值之和的比例：0.59	计算所得NMI为：0.80319
保留的特征值之和占有所有特征值之和的比例：0.70	计算所得NMI为：0.78109
保留的特征值之和占有所有特征值之和的比例：0.71	计算所得NMI为：0.77381
保留的特征值之和占有所有特征值之和的比例：0.78	计算所得NMI为：0.76076
保留的特征值之和占有所有特征值之和的比例：0.81	计算所得NMI为：0.74669
保留的特征值之和占有所有特征值之和的比例：0.85	计算所得NMI为：0.73318
保留的特征值之和占有所有特征值之和的比例：0.89	计算所得NMI为：0.72017
保留的特征值之和占有所有特征值之和的比例：0.95	计算所得NMI为：0.70838

图 3.1 运行结果