人工智能基础课程实验报告 2

姓名：章东泉　　**学号：**PB17121706

问题一　监督学习

Part1　KNN

按照 7:3 随机划分数据集，以预测数学成绩(**mat**)为例，我们在应用不同的 K 值时得到的准确率和召回率如下。

使用 G1,G2 属性时，

表 1-1　KNN 算法性能评估 1

K	Precision	Recall	Score
6	0.8500	0.8947	0.8718
8	0.8987	0.9342	0.9161
10	0.8636	0.9744	0.9157
12	0.8925	0.9881	0.9379
14	0.8409	0.9487	0.8916

由于数据集是随机划分的，在多次测试后可以发现，KNN 算法在 K=10 附近的效果最优。整个数据集有 395 条记录，划分后训练集(277)：测试集(118)。参考的数据点个数约占训练集的 3%-4%。

在 K 达到 50 时，查全率 R 达到了 1.0，但是显然的，查准率直线下降到不到 80%。综合考虑在 10 附近 90%左右的查准和 95%以上的查全是可以接受的。

而不使用 G1,G2 属性时：

表 1-2 KNN 算法性能评估 2

K	Precision	Recall	Score
6	0.6636	0.9342	0.7760
8	0.6973	0.9383	0.8000
10	0.6762	0.9103	0.7760
12	0.6545	0.9351	0.7700
14	0.6930	0.9753	0.8103

很明显，在失去了 G1,G2 属性后，预测的准确率有大幅的下降，说明最终成绩与 G1,G2 有很高的相关性。

Part2 SVM

支持向量机的二次规划部分用了 SMO 算法实现。

以预测葡萄牙语成绩(**por**)为例，使用 G1,G2 属性的预测结果如下：

表 2-1 SVM 算法性能评估 1

Soft-margin	kernel	Precision	Recall	Score
50	linear	1.0000	0.5337	0.6960
100	linear	0.9901	0.6211	0.7633
200	linear	1.0000	0.5917	0.7435
200	polynomial	0.8711	1.0000	0.9311
100	polynomial	0.8660	1.0000	0.9282

可以看出，支持向量机的性能十分优秀。使用线性核时，预测的查准率相当高，只有在少数几次测试中出现了假正例数量不为 0 的情况，大多数时候查准率都是 100%，但是查全率相对较低，在软间隔

变化的情况下，查全率一般只有不到 0.6。

使用多项式核时查全率显著上升，但是查准相对下降，但是运行速度提升了很多。多项式核使得 SMO 算法做二次规划的收敛速度显著上升。

而在不使用 G1,G2 属性的条件下：

表 2-2 SVM 算法性能评估 2

Soft-margin	kernel	Precision	Recall	Score
50	linear	0.8490	0.5625	0.6767
100	linear	0.8834	0.8521	0.8675
200	linear	0.8405	0.7205	0.7759
200	polynomial	0.8608	1.0000	0.9252
100	polynomial	0.8402	1.0000	0.9132

明显的变化是，使用线性核的查准率有显著下降，而不管是查准还是查全对多项式核都没有太大的影响。说明在缺少了 G1，G2 属性的情况下，线性核无法将数据点很好的用超平面线性划分，更可能出现的情况是无法线性可分，在迭代次数超过设置的 max_iter 后自动跳出循环，但是多项式核仍然能够有效分割数据集。

经过上面分析，使用 200 软间隔和多项式核在性能上比较好，也比较稳定。

SMO 算法的主要思想就是，通过选定一对参数，固定其他参数进行高效的优化。关于参数的选择，这里没有使用启发式，只是通过随机选择优化的参数。详见代码注释。

Part3 Naive-Bayes

监督学习自由完成的 other.py 选择采用朴素贝叶斯实现。

以预测数学成绩(**mat**)为例：

表 3-1 朴素贝叶斯算法性能评估

Precision	Recall	Score
0.9506	0.9333	0.9419
0.9379	0.9151	0.9264
0.8981	0.8981	0.8981
0.9758	0.9253	0.9499

朴素贝叶斯的主要实现思路就是通过训练集中的各个属性项的出现频率计算 P(x|Label)，测试时计算 P(Label|x)通过先验概率和似然函数乘积得到预测的标签。性能尚可。

Part4 代码展示

代码 4-1 KNN 标签预测

```
for data in TestSet:
    DistanceList = []
    for i in range(len(TrainSet)):
        distance = Calculate_Distance(data, TrainSet[i])
        CaledData = {'Data': TrainLabel[i], 'Value': distance}
        DistanceList.append(CaledData)
    DistanceList = sorted(DistanceList, key=lambda x:x['Value'])
    passNum = 0
    failNum = 0
    for i in range(K):
        if(DistanceList[i]['Data'] == 1):
            passNum = passNum + 1
        else:
            failNum = failNum + 1
    if(passNum >= failNum):
        predict_label.append(1)
```

```
    else:
        predict_label.append(-1)
```

代码 4-2 SMO 算法部分代码示意

```
for j in range(0, n):
    # 生成两个不同的随机数 i, j
    i = get_random(0, n-1, j)
    x_i, x_j = trainSet[i][:], trainSet[j][:]
    y_i, y_j = trainLabel[i], trainLabel[j]
    # 计算核
    K_ij = KernelTrans(x_i, x_i, kernel) + KernelTrans(x_j, x_j, kernel) - 2 * Kernel
Trans(x_i, x_j, kernel)
    if(K_ij == 0):
        continue
    alpha_prime_i, alpha_prime_j = alpha[i], alpha[j]
    (L, H) = get_L_H(C, alpha_prime_j, alpha_prime_i, y_j, y_i)
    # 计算模型参数
    w = cal_w(alpha, trainSet, trainLabel)
    b = cal_b(trainSet, trainLabel, w)
    E_i = E(x_i, y_i, w, b)
    E_j = E(x_j, y_j, w, b)
    # 新的 alpha
    alpha[j] = alpha_prime_j + float(y_j * (E_i - E_j)) / K_ij
    alpha[j] = max(alpha[j], L)
    alpha[j] = min(alpha[j], H)
    alpha[i] = alpha_prime_i + y_i * y_j * (alpha_prime_j - alpha[j])
```

代码 4-3 朴素贝叶斯学习过程

```
P_x_c = []
for i in range(len(dataType)):
    probability = []
    for data in encode[i]:
        tmp_positive = 0
        tmp_negative = 0
        for j in range(len(trainSet)):
            if(trainSet[j][i] == data and trainLabel[j] == 1):
                tmp_positive = tmp_positive + 1
            if(trainSet[j][i] == data and trainLabel[j] == -1):
                tmp_negative = tmp_negative + 1
        probability.append((tmp_positive / positive, tmp_negative / negative))
    P_x_c.append(probability)
```