Checkmate 实验报告

PB16060674-归舒睿

Checkmate 实验报告
PB16060674-归舒睿
代码间的处理思路
伪代码
knn
DecisionTree
SVM
评估结果
KNN
决策树
SVM
三者比较

代码间的处理思路

直接训练：原因是曾经对三个棋子的两两距离当作训练属性，但是发现结果并不理想，实际上是因为只看重两两距离，而忽视了棋子与边界的距离，也就是丢失了信息，因此效果不尽人意。由此我们可以想到，能由一组属性推出来的属性，信息量只可能等于或者小于。

伪代码

knn

```
输入：k，训练数据
输出：评估结果
无须训练
测试：
    对于每个点v，knear = check.distance(k, v)
    返回的knear为最近的k个节点
    distance的实现为计算每个点与v的距离（棋盘距离），排序，返回前n个
    对knear中的节点的class进行统计，选出投票最高的class作为ypred
    ypred与真实结果比较，计算评估参数
```

DecisionTree

```
输入：训练数据 D， 属性集 A
输出：以node为根节点的一颗决策树
函数：TreeGenerate(D, A)
生成节点node
```

```
if D中样本全属于同一类别C then
    将node标记为C类叶节点；  return
end if
if A = 空集 or D中样本在A上取值相同 then
    将node标记为叶节点，其类型标记为D中样本数最多的类；  return
end if
从A中选择最优划分属性 Bestfeature
对于每个 Bestfeature的取值 a:
    为node生成一个分支；令Dv表示D中在Bestfeature上取值为a的样本子集；
    if Dv为空 then
        将分支节点标记为叶节点，其类别标记为D中样本最多的类；  return
    else
        以 TreeGenerate(Dv, A除去Bestfeature)为分支节点
    end if
end
```

SVM

```
输入：训练集数据，测试集数据，sigma，C
输出：测试结果
对于 Min 1/2 * x.T * P * x + q.T * x
    l < A * x < u
    的解
P的计算：
    P = yl[SVMclass] * yr[SVMclass] * np.exp(- (np.sum(np.square(Kl - Kr), axis=2)) / sigma
** 2)
    Kl和Kr为每行相同，每列x1~xn和每列相同，每行x1~xn的矩阵
q的计算：
    q = - np.ones((length))
A的计算：
    A = np.identity((length + 1))
    A = np.delete(A, -1, axis=1)
    A[-1] = yr[SVMclass][-1]
    A = sparse.csc_matrix(A)
    最后一行的特殊是\sum{yi * xi} = 0的条件
l和u的计算：
    l = np.zeros((length + 1))

    u = np.ones((length + 1)) * C
    u[-1] = 0
    最后一行的特殊是和A的配合意在：  0 <= \sum {yi * xi} <= 0

代入qp求解器求解
求解出17个SVM
对于每个测试数据：
    对每个SVM，对此数据进行测试：
        计算b: b -= np.sum(a_np * yr[SVMclass][0] * np.exp(- (np.sum(np.square(Kl[0] -
Kr[0]), axis=1)) / sigma ** 2))
        计算f: f = np.sum(a_np * yr[SVMclass][0] * np.exp(- (np.sum(np.square(x - Kr[0]),
axis=1)) / sigma ** 2)) + b
    统计每个SVM得出的f的值，选出最大者作为预测的类ypred
    评估结果
```

评估结果

KNN

详情参考knn.out

k参数的取值从1取到10：

交叉验证，得到的10*5（参数*fold）的矩阵

```
[[0.52740642 0.52295009 0.52473262 0.5176025  0.50846702]
 [0.52740642 0.52295009 0.52473262 0.5176025  0.50846702]
 [0.64438503 0.625      0.62811943 0.61452763 0.6073975 ]
 [0.69050802 0.67424242 0.69184492 0.67713904 0.67892157]
 [0.73217469 0.71702317 0.73328877 0.71323529 0.71590909]
 [0.74977718 0.73618538 0.74710339 0.73284314 0.73395722]
 *[0.75735294 0.73729947 0.75713012 0.73707665 0.73618538]*
 [0.75155971 0.73663102 0.7486631  0.7337344  0.73418004]
 [0.74220143 0.73061497 0.74131016 0.72704991 0.73128342]
 [0.73596257 0.72615865 0.73729947 0.71925134 0.72771836]]
 best= 7
```

得到最好的k的取值为7

在k=7时，在测试集上得到的结果为

```
total: 5611 Accuracy: 0.7720548921760827 Macro F1: 0.7110160901831112 Micro F1:
0.7720548921760827
Train finished after: 358.5605636
```

达到了77%的正确率

决策树

决策树的评估结果为：

```
total: 5611 Accuracy: 0.5781500623774728 Macro F1: 0.5303016749907119 Micro F1:
0.5781500623774728
Train finished after: 1.6695634000000001
```

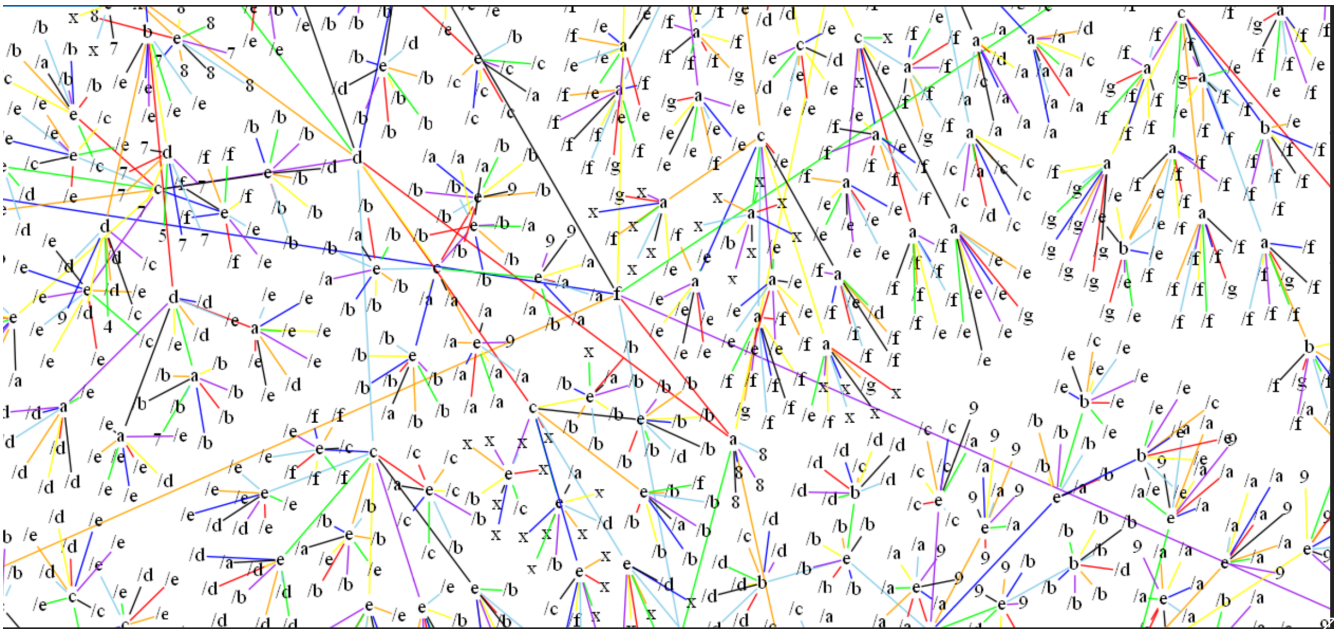
达到了58%的正确率

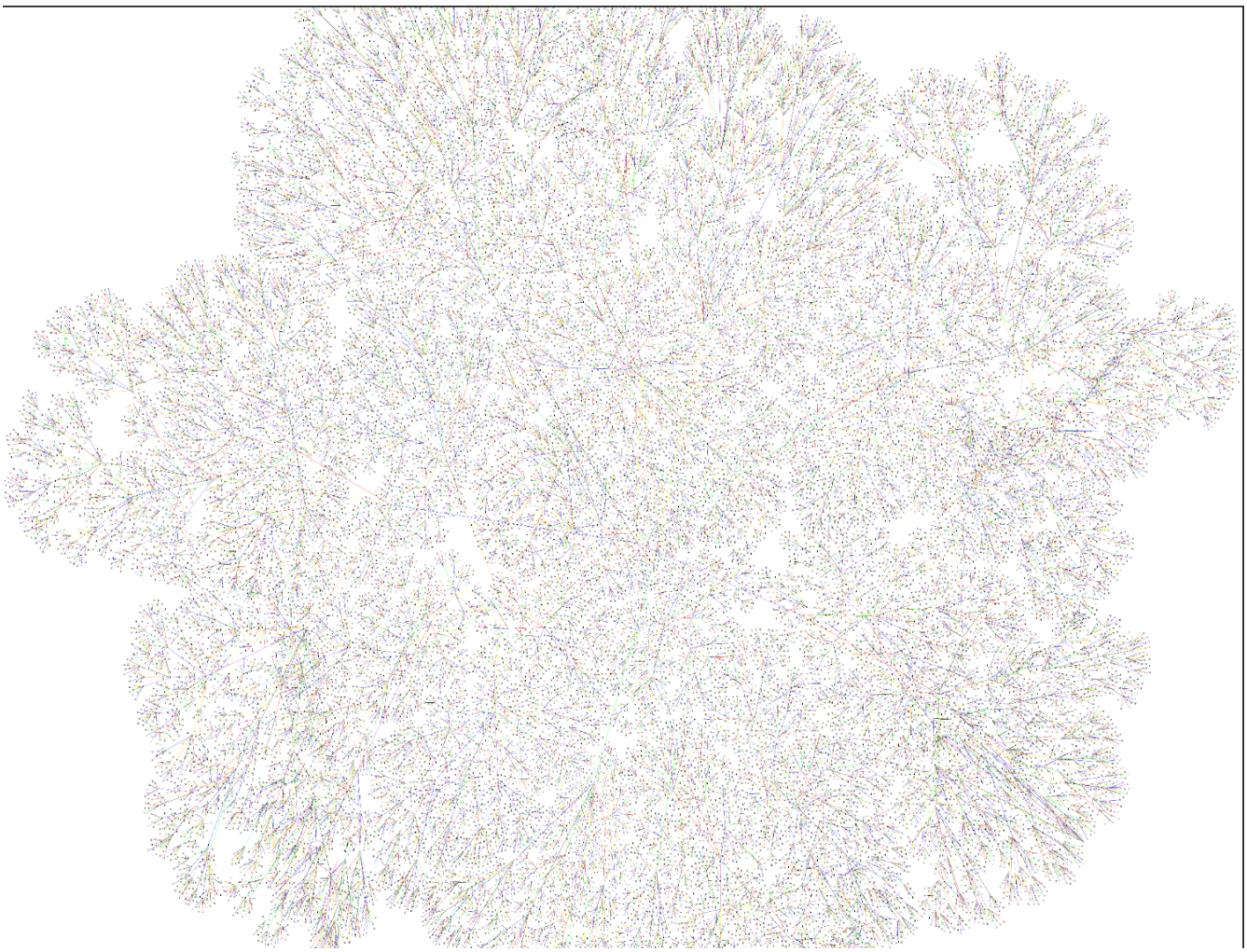
决策树的可视化调了特别久，因为一旦图片过大，就会报段错误，但是压缩不消除重叠又看不清，而且线与字之间有空白距离，而这空白距离实际又由多个参数控制。使用graphviz相关大图文档较少，一些个别的推荐参数又容易直接报段错误，因此以下记录最终参数与可视化结果。**原图请在checkmate/Graph.gv.png 21MB查看，可以手动走完决策树（从最中心的f开始），可视化程度为：全部节点可视化。**若想运行一遍带可视化的程序，请把第286行的dot.view()注释还原回代码（预计时间几分钟）

参数：

```
dot = Graph(comment='Tree', format='png', engine='sfdp')
```

```
dot.graph_attr['nodesep'] = str(0.02)
dot.graph_attr['ranksep'] = str(0.02)
dot.attr('node', shape='plaintext')
dot.attr('node', color='none')
dot.attr('node', margin='0')
dot.attr('node', width='0')
dot.attr('node', height='0')
dot.graph_attr['overlap'] = 'prism10' # 5 去重叠指标
dot.graph_attr['overlap_shrink'] = 'true'
dot.graph_attr['concentrate'] = 'true'
# dot.graph_attr['splines'] = 'curved'
dot.graph_attr['fontsize'] = str(1.0)
key2color = {1: 'red', 2: 'orange', 3: 'yellow', 4: 'green', 5: 'blue', 6: 'skyblue', 7:
'purple', 8: 'black'}
class2num = {'draw': 'x', 'zero': '0', 'one': '1', 'two': '2', 'three': '3', 'four': '4',
'five': '5', 'six': '6',
             'seven': '7', 'eight': '8', 'nine': '9', 'ten': '/a', 'eleven': '/b',
'twelve': '/c', 'thirteen': '/d',
             'fourteen': '/e', 'fifteen': '/f', 'sixteen': '/g'}
```





SVM

详情参考SVM.out

SVM是在5000的训练集上跑的，10参数 5fold需要跑约9个小时

10组参数对应是：

```
sigma = [0, 0.45, 4.5, 45, 2, 0, 0.45, 4.5, 45, 2]
C = [2, 2, 2, 2, 2, 10, 10, 10, 10, 10]
```

10参5fold的交叉验证矩阵为：

```
[[0.09893048 0.09893048 0.09901873 0.09901873 0.09901873]
 [0.18449198 0.18270945 0.18198037 0.18733274 0.19000892]
 [0.33778966 0.32976827 0.31489741 0.35860839 0.34522748]
 [0.09893048 0.09893048 0.09901873 0.09901873 0.09901873]
 *[0.51158645 0.52049911 0.52096343 0.56199822 0.54504906]*
 [0.09893048 0.09625668 0.1043711  0.09901873 0.09901873]
 [0.18449198 0.18270945 0.18198037 0.18733274 0.19000892]
 [0.35026738 0.42156863 0.38804639 0.40677966 0.38358608]
 [0.09893048 0.09893048 0.09901873 0.09901873 0.09901873]
 [0.42067736 0.49376114 0.48438894 0.53434434 0.53612846]]
----------------------best= 4
```

为当sigma取2，C取2时验证效果最好：

实际测试结果为：

```
total: 5611 Accuracy: 0.5897344501871324 Macro F1: 0.4964574089709358 Micro F1:
0.5897344501871324
finished train in 1525.5922045000007 s
finished all in 31283.818305 s
```

达到了59%的正确率

三者比较

类型	数据量	有交叉验证?	Accuracy	Macro F1	Micro F1	Time
KNN	100%	有	0.7720548921760827	0.7110160901831112	0.7720548921760827	359
决策树	100%	无	0.5781500623774728	0.5303016749907119	0.5781500623774728	1.6
SVM	25%	有	0.5897344501871324	0.4964574089709358	0.5897344501871324	31284

三者比较来说KNN时最好的，但是决策树是最快的，在不需要很精确的情况下，决策树能够给出一个可接受解。

而SVM是潜力最大的，第一它只用了25%的数据，其次它的高维分离的特性是knn难以达到的，虽然它所花费的时间是最长的，和神经网络有得一拼。