

華東理工大學

模式识别大作业

学 院	信息科学与工程
专 业	控制科学与工程
组 员	杨涛 （学号：Y30180688）
指导教师	赵海涛

完成日期： 2018 年 10 月 24 日

模式识别作业报告

组员：杨涛

经过一段时间的课上学习以及课下的自我补充让我对机器学习有了初步的了解，对机器学习的一些算法也有了一定的认识。目前为止我们主要学过 PCA、SVD 和最小二乘法等数据的处理方法以及贝叶斯、朴素贝叶斯和 Logistic 回归等机器学习中主流算法，课下自己有自学决策树算法。本次大作业我采用课下自学的决策树算法来实现一个二分类，以此来巩固学习过的算法。

1 Titanic 生存预测：

本次报告按照要求在 Lintcode 网站上人工智能题库十道题中选择一题的规则，我选择了 Titanic 生存预测，实验数据均来自于题库，其中给了我们用于训练模型的训练样本集和测试样本集。通过训练样本集训练生成我们的决策模型，那训练好的模型测试测试样本集得到模型的测试结果，将结果输出生成 CSV 文件格式上传 Lintcode 可查看自己所做模型的好坏。Titanic 生存预测中一共给了我们 12 个属性特征，分别是：survival(是否生存)、pclass(船仓等级)、sex(性别)、age(年龄)、sibsp(船上的兄弟姐妹或配偶的人数)、parch(船上父母子女同行的人数)、ticket(票号)、fare(票价)、cabin(客舱号)、embarked(登船口)。本次大作业我通过训练样本集来建立决策树。

1.1 决策树基本概念知识：

决策树是一类常见的机器学习方法，分类目标为二分类。通过对给定的训练样本集进行学习得到一个模型用以对新事物进行分类，看这个新事物是属于哪一类，这个对新事物做划分的过程就叫“决策”或“判定”。例如判断一个西瓜是否为好瓜时，我们需要从它的色泽、根蒂、敲声等方面来判断。在这个过程中我们通常会经过一系列的“子决断”。判断过程大致如下图所示：

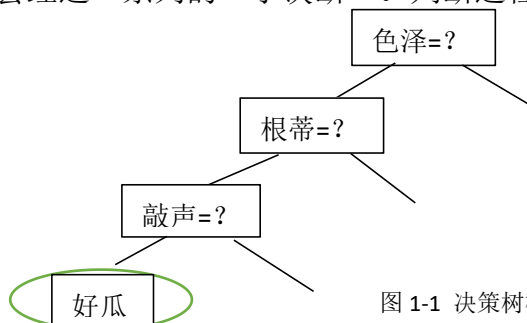


图 1-1 决策树模型

一般来说，一棵决策树包含一个根节点、若干个内部节点和若干个叶子结点，叶子结点对应于决策结果，其他节点则对应于一个特征属性。每个节点包含的样本集合根据属性测试的结果被划分到叶子结点中，根节点包含样本全集。从根节点到每个叶子节点的路径对应了一个判定测试序列。决策树学习的目的是为了长生一棵泛化能力强，即处理未见实例能力强的决策树，其基本流程遵循简单且直观的“分而治之”策略。决策树的生成是一个递归过程，在决策树的算法中有三种情况会导致递归返回：（1）当前节点包含的样本全属于同一类，无需划分；（2）当前特征属性集为空，或是所有样本在所有属性上取值相同，无法划分；（3）当前节点包含的样本集合为空，不能划分。

在第（2）种情形下，我们把当前节点标记为叶子结点，并将其类别设定为该节点所含样本最多的类别；在第（3）种情形下，同样把当前节点标记为叶子结点，但将其类别设定为其父节点所含样本最多的类别。注意这两种情形处理的实质不同：情形（2）是在利用当前节点的后验分布，而情形（3）则是把父节点的样本分布欧威当前节点的先验分布。

1.2 划分选择：

利用实物的多种特征属性对实物进行决策的关键问题在于如何从多种特征属性中找出最佳判决属性。一般而言，随着划分过程的不断进行，我们希望决策树的分支节点所包含的样本尽可能属于同一类，即节点的“纯度”（purity）越来越高。

信息增益：

“信息熵”（information entropy）是度量样本集合纯度最常用的一种指标。假定当前集合 D 中第 k 类样本所占的比例为 P_k ($k=1,2,\dots,|y|$)，则 D 的信息熵定义为

$$Ent(D) = -\sum_{k=1}^{|y|} P_k \log_2 P_k \quad (1.1)$$

$Ent(D)$ 的值越小，则 D 的纯度越高。

假定属性 a 有 v 个可能的取值 $\{a^1, a^2, \dots, a^v\}$ ，若使用 a 来对样本集 D 进行划分，则会产生 v 个分支节点，其中第 v 个分支节点包含了 D 中所有在属性 a 上

取值为 a^v 的样本，记为 D^v 。我们可根据式 (1.1) 计算出 D^v 的信息熵，再考虑到不同的分支节点所包含的样本数不同，给分支节点赋予权重 $|D^v|/|D|$ ，即样本数越多的分支节点的影响越大，于是可计算出用属性 a 对样本集 D 进行划分所获得的“信息增益” (information gain)

$$Gain(D, a) = Ent(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} Ent(D^v) \quad (1.2)$$

一般而言，信息增益越大，则意味着使用属性 a 来进行划分所获得的“纯度提升”越大。一次我们可用信息增益来进行决策树的划分属性选择，即选择属性 $a^* = \arg \max Gain(D, a)$ 。著名的 ID3 算法决策树学习算法就是以信息增益为准则来选择划分属性，本次大作业亦是采用 ID3 算法来建立决策树的。

2 Titanic 训练样本集特征属性处理

在建立决策树之前我们需要首先对训练样本集的特征属性进行分析。此处我们采用 python 中的数据透视功能来分别对各特征属性与生存之间的关系进行查看，看哪些属性对存活率有着比较大的影响。在此之前我们先通过 python 查看一下训练样本集的数据情况。此处我们只读入前五元素。

```
In [2]: import pandas as pd
import numpy as np
from pandas import Series, DataFrame
titanic_survival=pd.read_csv("C:/Users/YT/Desktop/train(1).csv")
colus=titanic_survival.columns
titanic_survival.head(5)
#print(colus)
#print(heads)
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

图 2-1: 读入训练样本集前五行数据

从上图我们可以清楚看到训练样本集中包含哪些特征属性。对每个特征属性中有多少缺失值和各特征属性的一些特性我们可查看到如下：

```
In [3]: titanic_survival.info()
#查看每列属性的未缺失个数。

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived        891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

图 2-2：各特征属性缺失值情况

```
In [4]: titanic_survival.describe()

Out[4]:
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

图 2-3：特征属性细节描述

样本数据集中 891 人存活率为：

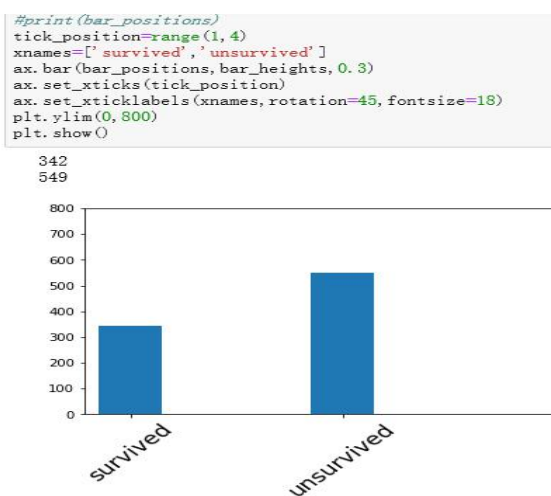


图 2-4：存活率

以下我们分别对几个主要的特征属性与存活率之间的关系进行透视。

2.1 年龄与存活率的关系：

首先我们获取到年龄的全部数据，如下：

```

In [6]: titanic_survival['Age']

Out[6]: 0      22.0
        1      38.0
        2      26.0
        3      35.0
        4      35.0
        5       NaN
        6      54.0
        7       2.0
        8      27.0
        9      14.0
       10       4.0
       11      58.0
       12      20.0
       13      39.0
       14      14.0
       15      55.0
       16       2.0
       17       NaN
       18      31.0
       19       NaN

```

图 2-5：年龄特征属性

从上面的图我们可以看到年龄特征属性存在很多的缺失值，我们在分析年龄对生存的影响时需要先对缺失值进行处理。在 ID3 算法中我们通常有两种方法对确实值进行处理，一是利用平均值填充缺失值，而是舍弃缺失值。此次大作业我们采取舍弃年龄的缺失值。如下图：

```

In [7]: titanic_survival=titanic_survival.dropna(axis=0, subset=['Age'])
        titanic_survival['Age']

Out[7]: 0      22.0
        1      38.0
        2      26.0
        3      35.0
        4      35.0
        6      54.0
        7       2.0
        8      27.0
        9      14.0
       10       4.0
       11      58.0
       12      20.0
       13      39.0
       14      14.0
       15      55.0
       16       2.0
       18      31.0
       20      35.0
       21      34.0
       22      15.0
       23      28.0

```

图 2-6：舍弃缺失值

由于年龄是连续的特征属性，我们需要先对其进行分段，然后进行透视，透

视情况如下：

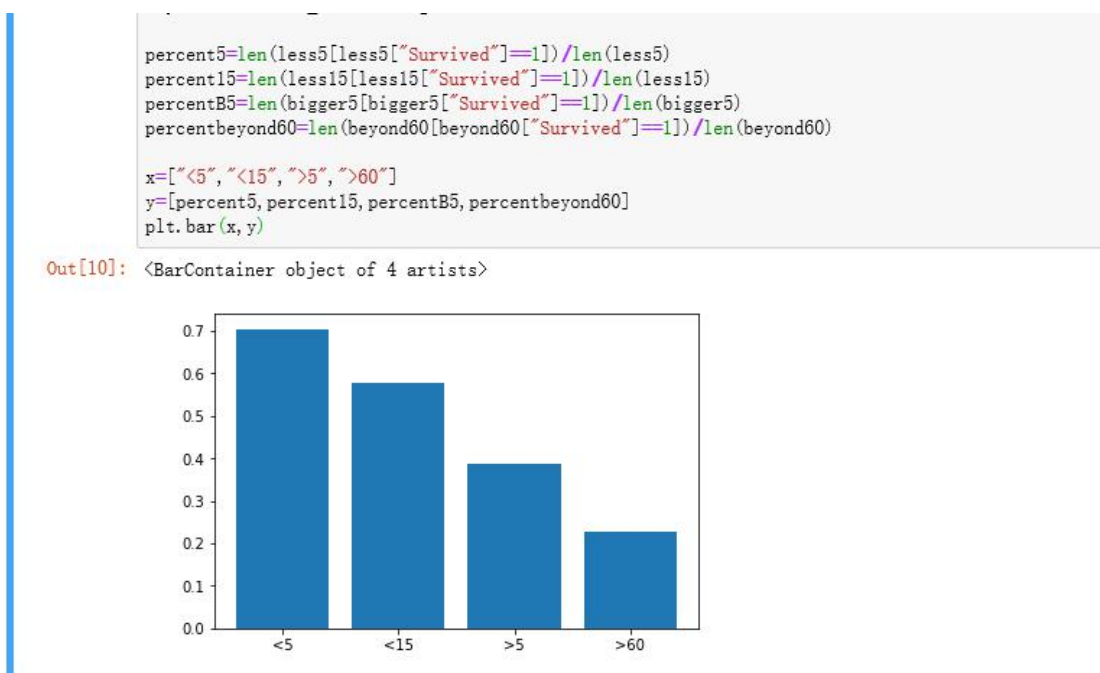


图 2-7：年龄与存活的情况透视

从透视所得的柱形图我们可以观察到，年龄小于 5 岁的存活率非常高，要远远大于年龄大于 60 的，也就是说年龄对是否存活有着一定的影响。

2.2 船舱等级与存活率

船舱等级越高费用也就越高，也就是有钱人越多，我们会想这是否花钱越多存活下来的几率越大呢？通过透视我们得到一等舱、二等舱和三等舱的存活率如下：

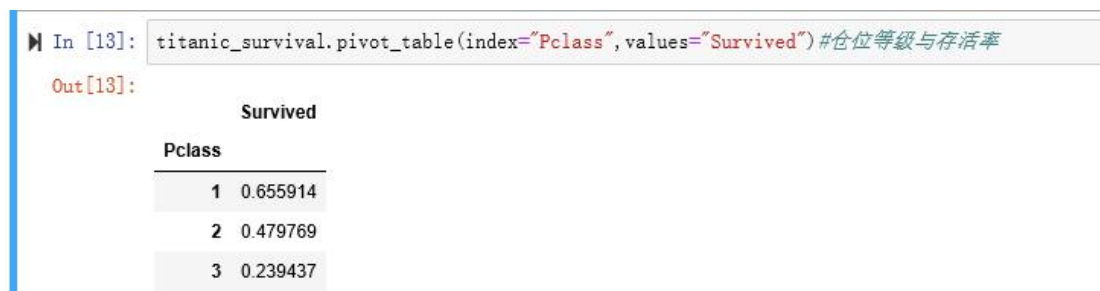


图 2-8：船舱等级存活率

从上面的数据分析我们可知，船舱等级对是否能够生存下来也有着一定影响。

2.3 性别与存活率

在生活中我们经常会看到男士礼让女士的情况，在面临生死抉择时是否依然存在这种情况呢？通过数据透视，我们得到如下结果：

```
In [15]: titanic_survival.pivot_table(index="Sex", values="Survived") #性别与存活率
```

Out[15]:

	Survived
female	0.754789
male	0.205298

图 2-9: 性别与存活率

从上面的数据我们发现女性的存活率远远大于男性的存活率,说明性别这一特性对是否存活有着很大的影响,我们猜测该属性可能信息增益也是最大的,后面我们在进行特性选择时也会发现确实是性别的信息增益最大,故性别是最好的特征属性。以下不在分别透视每一个特征属性,本次大作业我们主要选择了 Pclass、Sex、SibSp、Parch、Embarked、Fare、Age 这些特征属性来构建决策树的。

3 决策树的构建:

决策树的构建主要包含数据集的熵值的计算、按给定特征属性划分数据集(将某一属性按其取值的可能分类)、计算属性的信息增益(选择最佳属性进行分类)、对分类后的结点做判断,若该节点多有样本属于同一类则停止下一步划分数据集,并将该节点作为叶子结点确定节点的类别,如其中样本不属于同一类但所有属性以判断完则将该节点定位叶子结点并将叶子结点类别定位其中包含数据集较多的类别,若为上述两种情况则进行下一步分类、最后创建决策树。

在利用 python 创建决策树需要注意一些问题。我们在决策树中的各种处理一定要记得是对 DataFrame 的数据类进行处理,我们利用 python 读取到的 csv 文件的数据类型是 DataFrame 的类型,在创建决策树时我们很容易陷入 DataFrame 的数据处理方式与 Series 数据类型的处理方式的不同指点,会常出现错误,故需要注意。

3.1 决策树创建过程:

3.1.1 数据的读入与与处理:

```
#引入所需要的模块
from math import log
from pandas import DataFrame, Series
import pandas as pd
import operator
import numpy as np
import matplotlib.pyplot as plt
```



```

dataSet = pd.read_csv('C:/Users/YT/Desktop/train(1).csv') #读取训练样本集数据集
#dataSet.columns #查看所有特征属性
#dataSet.head(5) #查看前五个元素，看读取情况
dataSet=dataSet.drop(['PassengerId'],axis=1) #删除无用属性 PassengerId
#print(dataSet.head(5)) #查看删除后的样本数据集

#对连续变量 Age、Fare、Parch、SibSp 进行处理
s = dataSet['Fare']
for i in range(len(dataSet)):
    if s[i]<7.910400:
        dataSet.ix[i, 'Fare2']=0
    elif s[i]<14.454200:
        dataSet.ix[i, 'Fare2']=1
    elif s[i]<31.000000:
        dataSet.ix[i, 'Fare2']=2
    else:
        dataSet.ix[i, 'Fare2']=3
    if dataSet.ix[i, 'Parch']>2:
        dataSet.ix[i, 'Parch'] = 2
    if dataSet.ix[i, 'SibSp']>3:
        dataSet.ix[i, 'SibSp'] = 3
k=dataSet['Age']
for i in range(len(dataSet)):
    if k[i]=='NaN':
        dataSet=dataSet.dropna(axis=0,subset=['Age'])
    elif k[i]<5.000000:
        dataSet.ix[i,'Age2']=0
    elif s[i]<15.000000:
        dataSet.ix[i,'Age2']=1
    elif s[i]<60.000000:
        dataSet.ix[i,'Age2']=2
    else:
        dataSet.ix[i,'Age2']=3
#删除无用变量
dataSet=dataSet.drop(['Age'],axis=1)
dataSet=dataSet.drop(['Cabin'],axis=1)
dataSet=dataSet.drop(['Name'],axis=1)
dataSet=dataSet.drop(['Ticket'],axis=1)
dataSet=dataSet.drop(['Fare'],axis=1)
print(dataSet.head(3))

dataSet=dataSet.fillna({'Embarked':'S'})

```

处理完数据集后我们查看一下处理后的前三行元素看看处理后是否是否达

到我们所想的结果，查看结果如下：

See the documentation here:
<http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated>

	Survived	Pclass	Sex	SibSp	Parch	Embarked	Fare2	Age2
0	0	3	male	1	0	S	0.0	1.0
1	1	1	female	1	0	C	3.0	3.0
2	1	3	female	0	0	S	1.0	1.0

图 3-1: 数据处理结果

3.1.2 决策树的创建结果:

此次报告中我就不对如何一步步创建决策树进行详细的讲解，具体的可以看后面的代码，在看代码时需要注意的是注意我们处理的 DataFrame 类型的数据集，所以其中有很多对 DataFrame 数据类型的操作我转换成较为简单的 Series 类型的操作。如下方这个地方就是我所处理的地方。

```

In [100]: #计算数据集的熵值确定最佳判断的属性

def sulEnt(data):
    numEnt=len(data)
    labelsCount={}
    l=data.columns.tolist()
    #print(l)
    #print(data.head(3))
    k=l.index('Survived')
    #print(k)
    for i in range(len(data)):
        featVec=data.loc[i]

```

图 3-2: DataFrame 数据类型的处理

另外，在对某个特征属性按其不同取值进行划分时我们要注意要对 index 进行处理，因为我们在对特征进行取值划分时会保留原始的 index，我们后期再对划分好的子类进行熵值进行计算时会用其 index 按从大到小进行检索，然而划分后我们的 index 会出现缺失，故会产生错误，此处是很容易产生错误的地方。再多次通过 pycharm 进行单独断点调试才发现错误所在，故在此作出强调。面对这个问题我所采取的方法是对 index 进行重新设置。具体方法如下：

```

featSplit=featSplit.reset_index()
featSplit=featSplit.drop(['index'],axis=1)
#print(featSplit)

```

图 3-3: index 处理

首先重置 index，这会产生一个新的顺序的 index，原始的 index 还依然存在，然后通过 drop 命令删除原始的 index，这样我们就成功重设了样本集的 index。

查看一下我们处理后的结果，如下：

	index	Survived
0	1	1
1	3	1
2	6	0
3	11	1
4	23	1
5	27	0
6	30	0
7	31	1
8	34	0
9	35	0
10	52	1
11	54	0
12	55	1
13	61	1
14	62	0
15	64	0
16	83	0
17	88	1

图 3-4：处理前的 index

	Survived
0	1
1	1
2	0
3	1
4	1
5	0
6	0
7	1
8	0
9	0
10	1
11	0
12	1
13	1
14	0
15	0
16	0
17	1

图 3-5：处理后的 index

数据处理及基本注意事项解决后我们开始来利用这些基础来创建我们的决策树，在创建决策树时我们依然需要注意上诉所提到数据类型和 index 方面的问题。在决策树的创建过程中我们会用到决策树的递归返回，在此处我们对代码稍做一些解释，代码具体如下：

```
def createTree(data,labels):
    #此处，创建一个类别列表用于对叶子结点的类别进行判断
    classList=list(data['Survived'].values)
    #首先判断节点内样本集是否属于同一类，若是则放回样本类型，否则判断所有属性是否判别完，如是返回样本集内样本属于同类数目较多的类别作为该叶子节点的类型。
    if classList.count(classList[0])==len(classList):
        return classList[0]
    if len(data.columns)==1:
        return majorityClass(classList)
    #若上述两个条件均不成立，则进行下一次的最佳特征属性的选择进行下一次划分数据集。
    bestFeature=chooseBestFeatureToSplit(data)
    #以字典的形式来建立决策树格式
    myTree={bestFeature:{}}
```


分类。训练代码与结果如下：

```
myTree=createTree(dataSet,dataSet.columns)
featLables=dataSet.columns.tolist()
for i in range(len(dataSet)):
    a_test=dataSet.ix[i]
    r_p=classify(myTree,featLables,a_test)
    print(r_p)
```

```
0
1
0
1
0
0
0
0
1
1
1
1
0
0
0
1
0
0
0
1
1
1
```

图 3-7：训练结果

从结果可以看出，我们所得到的结果正是模型训练所得到的，模型训练好后我们接下来只需对测试样本集进行与训练样本集相同的特征属性处理，然后拿训练好的树进行测试即可得到预测结果。测试代码与结果如下所示：

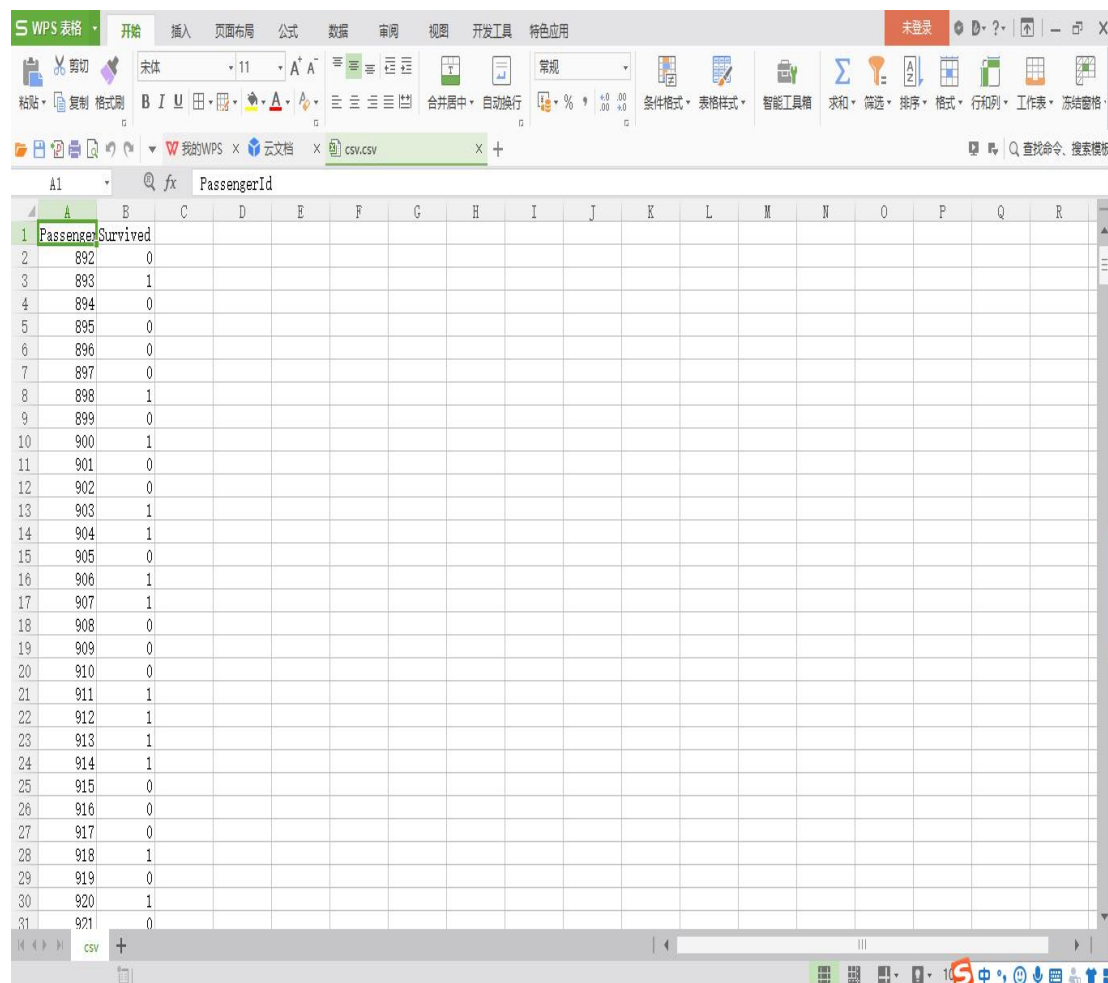
```
In [36]: import csv
csvfile=open('C:/Users/YT/Desktop/csv.csv','w',newline='')
writer=csv.writer(csvfile)
writer.writerow(['PassengerId','Survived'])
result=[]
#myTree=createTree(dataTest,dataTest.columns)
featLables=dataTest.columns.tolist()

for i in range(len(dataTest)):
    Id=PassengerId[i]
    a_test=dataTest.ix[i]
    r_p=classify(myTree,featLables,a_test)
    print(r_p)
    current_result=(Id,r_p)
    result.append(current_result)
#print(result)
writer.writerows(result)
csvfile.close()
```

```
0
1
1
1
1
0
0
0
1
0
1
```

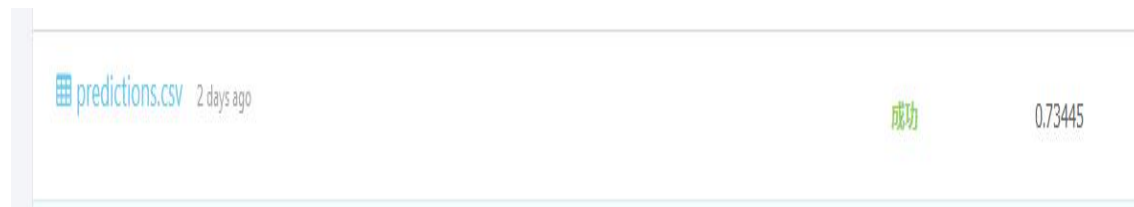
图 3-8：测试结果

如图 3-8 所示,我们在将最后的结果一 CSV 文件格式输出得到如下图所示的结果,将结果提交到 Lintcode 网站评分,准确率为 73.445%,对于二分类问题这个分类效果明显不是特别好,我们在一些特征属性的处理上面还可以进一步处理,或者可以考虑将 ID3 算法换成 C45 算法,对缺失值 ID 算法的处理太过暴力,多少会带来一些影响,C45 算法对缺失值的处理更为妥当。此次报告暂不讨论 C45 算法,后面我们在进行完善。



PassengerId	Survived
892	0
893	1
894	0
895	0
896	0
897	0
898	1
899	0
900	1
901	0
902	0
903	1
904	1
905	0
906	1
907	1
908	0
909	0
910	0
911	1
912	1
913	1
914	1
915	0
916	0
917	0
918	1
919	0
920	1
921	0

图 3-9: 测试结果 CSV 格式输出



File Name	Status	Accuracy
predictions.csv 2 days ago	成功	0.73445

图 3-10: 网上测评结果

4 总结

通过这次大作业，我对老师上课所讲的知识得到了进一步的巩固，同时自己学习了一些其他的算法。模式识别这门课让我深深喜欢上了机器学习，喜欢上那种遇到问题解决问题，到最后完成整个作业后满满的收获感。在模式识别这门课之前我对机器学习并未有多少了解，对 `python` 更是不熟悉。但通过这次作业后我对这些都有了一定的了解，了解到它的魅力所在。在这次大作业中我也深感到自己还有很多不足之处，所欠缺的东西很多，要学的东西也很多。因为自己的基础不好浪费了很多时间去巩固基础，以至于未能将算法完善，正确率不是很理想，希望后期能够继续的系统去学习。

最后非常感谢赵老师不论课上还是课下对我们的耐心指导，让我们能够在培养兴趣的同时为我们指明问题所在。

附页

```
from math import log
from pandas import DataFrame, Series
import pandas as pd
import operator
import numpy as np
import matplotlib.pyplot as plt

dataSet = pd.read_csv('C:/Users/YT/Desktop/train(1).csv')
#print(dataSet.columns)
#print(dataSet.head(5))
dataSet=dataSet.drop(['PassengerId'],axis=1)
#print(dataSet.head(5))
s = dataSet['Fare']
for i in range(len(dataSet)):
    if s[i]<7.910400:
        dataSet.ix[i, 'Fare2']=0
    elif s[i]<14.454200:
        dataSet.ix[i, 'Fare2']=1
    elif s[i]<31.000000:
        dataSet.ix[i, 'Fare2']=2
    else:
        dataSet.ix[i, 'Fare2']=3
    if dataSet.ix[i, 'Parch']>2:
        dataSet.ix[i, 'Parch'] = 2
    if dataSet.ix[i, 'SibSp']>3:
        dataSet.ix[i, 'SibSp'] = 3
#删除无用变量
dataSet=dataSet.drop(['Age'],axis=1)
dataSet=dataSet.drop(['Cabin'],axis=1)
dataSet=dataSet.drop(['Name'],axis=1)
dataSet=dataSet.drop(['Ticket'],axis=1)
dataSet=dataSet.drop(['Fare'],axis=1)
#print(dataSet.head(3))

dataSet=dataSet.fillna({'Embarked':'S'})

#dataSet=np.array(dataSet)
#print(dataSet)
```



```

def sulEnt(data):
    numEnt=len(data)
    labelsCount={}
    l=data.columns.tolist()
    #print(l)
    #print(data.head(3))
    k=l.index('Survived')
    #print(k)
    for i in range(len(data)):
        featVec=data.loc[i]
        #print(featVec)
        #featVec=list(featVec)
        currentLabel=featVec[k]
        #print(currentLabel)
        if currentLabel not in labelsCount.keys():
            labelsCount[currentLabel]=0

        labelsCount[currentLabel]+=1
    resultEnt=0.0
    for key in labelsCount:
        p=float(labelsCount[key])/numEnt
        resultEnt-=p*log(p, 2)
    return resultEnt
# mydata=sulEnt(dataSet)
# print(mydata)

#按照给定的属性划分数数据集,将同一属性的各种不同取值进行分类,以便于计算增益熵中的权重
def splitData(data,axis,value):
    retData=data[data[axis]==value]
    retData=retData.drop([axis],axis=1)
    return retData
#mydata=splitData(dataSet,'Pclass',1)
#print(mydata)

#计算不同属性的增益熵,决定最有分类属性进行分类
def chooseBestFeatureToSplit(data):
    baseEnt=sulEnt(data)
    bestEnt=0.0
    bestFeature=data.columns[1]
    for i in data.columns:
        if i=='Survived':continue
        featList=data[i]
        #print(featList)

```

```

        uniqueVals=set(featsList)
        #print(uniqueVals)
        newEnt=0.0
        for value in uniqueVals:
            featSplit=splitData(data,i,value)
            # print(featSplit)
            r=len(data[data[i]==value])/float(len(data))
            # print(r)
            featSplit=featSplit.reset_index()
            featSplit=featSplit.drop(['index'],axis=1)
            #print(featSplit)
            newEnt+=r*ent(featSplit)
            #print(newEnt)
        infoGain=baseEnt-newEnt
        if i != 'Survived':
            if (infoGain>bestEnt):
                bestEnt=infoGain
                bestFeature=i
        return bestFeature

#判断叶子结点中所有样本是否属于同一类, 如不属于同一类则统计其中类别个数最多的哪一类作为该叶子
#结点的分类, 并返回
import operator
#判断叶子结点中所有样本是否属于同一类, 如不属于同一类则统计其中类别个数最多的哪一类作为该叶子
#结点的分类, 并返回
def majorityClass(classList):
    classCount={}
    for i in classList:
        if i not in classCount.keys():
            classCount[i]=0
        classCount[i]+=1
    sortClass=sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)
    return sortClass[0][0]

#利用递归方法创建决策树
def createTree(data, labels):
    classList=list(data['Survived'].values)
    if classList.count(classList[0])==len(classList):
        return classList[0]
    if len(data.columns)==1:
        return majorityClass(classList)
    bestFeature=chooseBestFeatureToSplit(data)
    myTree={bestFeature: {}}
    featVals=data[bestFeature]

```

```
uniqueVals=set(featsVals)
for valus in uniqueVals:
    subLabes=labels[:]
    new_splitData=splitData(data,bestFeature, valus)
    new_splitData=new_splitData.reset_index()
    new_splitData=new_splitData.drop(['index'],axis=1)
    myTree[bestFeature][valus]=createTree(new_splitData, labels)
return myTree

#myTree=createTree(dataSet,dataSet.columns)
#print(myTree)

#使用决策树的分类函数
def classify(inputTree, featLabels, testVec):
    global classLabel
    firstStr=list(inputTree.keys())
    firstStr=firstStr[0]
    #print(firstStr)
    secondDict=inputTree[firstStr]
    #print(secondDict)
    featIndex=featLabels.index(firstStr)
    for key in secondDict.keys():
        if testVec[featIndex]==key:
            if type(secondDict[key]).__name__=='dict':
                classLabel=classify(secondDict[key], featLabels, testVec)
            else:
                classLabel=secondDict[key]
    # print(classLabel)
    return classLabel

from math import log
from pandas import DataFrame, Series
import pandas as pd
import operator
import numpy as np
import matplotlib.pyplot as plt

dataTest = pd.read_csv('C:/Users/YT/Desktop/test(2).csv')
#print(dataTest.columns)
#print(dataTest.head(5))
```

```

PassengerId=dataTest['PassengerId']
#print(PassengerId)
dataTest=dataTest.drop(['PassengerId'],axis=1)
#print(dataTest.head(5))
s = dataTest['Fare']
for i in range(len(dataTest)):
    if s[i]<7.910400:
        dataTest.ix[i, 'Fare2']=0
    elif s[i]<14.454200:
        dataTest.ix[i, 'Fare2']=1
    elif s[i]<31.000000:
        dataTest.ix[i, 'Fare2']=2
    else:
        dataTest.ix[i, 'Fare2']=3
    if dataTest.ix[i, 'Parch']>2:
        dataTest.ix[i, 'Parch'] = 2
    if dataTest.ix[i, 'SibSp']>3:
        dataTest.ix[i, 'SibSp'] = 3

#删除无用变量
dataTest=dataTest.drop(['Age'],axis=1)
dataTest=dataTest.drop(['Cabin'],axis=1)
dataTest=dataTest.drop(['Name'],axis=1)
dataTest=dataTest.drop(['Ticket'],axis=1)
dataTest=dataTest.drop(['Fare'],axis=1)
#print(dataTest.head(3))

dataTest=dataTest.fillna({'Embarked':'S'})
#dataTest=np.array(dataTest)
#print(dataTest)

myTree=createTree(dataSet,dataSet.columns)
featLables=dataSet.columns.tolist()
#RCount=0
#asum=0
for i in range(len(dataSet)):
    a_test=dataSet.ix[i]
    r_p=classify(myTree,featLables,a_test)
    #print(r_p)

import csv

csvfile=open('C:/Users/YT/Desktop/csv.csv','w',newline='')

```

```
writer=csv.writer(csvfile)
writer.writerow(['PassengerId','Survived'])
result=[]
#myTree=createTree(dataTest,dataTest.columns)
featLables=dataTest.columns.tolist()
#RCount = 0
#asum = 0
for i in range(len(dataTest)):
    Id=PassengerId[i]
    a_test = dataTest.ix[i]
    r_p = classify(myTree,featLables,a_test)
    #print(r_p)
    current_result=(Id,r_p)
    result.append(current_result)
#print(result)
writer.writerows(result)
csvfile.close()
```