数据挖掘分类大作业

2016.5.22

朱青蓥 5130369057
许翰云 5130369055

一：数据分析和探索

1. 数据集选择

此次老师总共提供了两个数据集，分别为

- **Default of credit card clients Data Set**
- **Poker Hand Data Set**

综合分析后，我们小组选择了第一个数据集，即**"default of credit card clients data set"**作为本次挖掘的数据集，原因如下：

- 该数据的最终目的是预测某个信用卡用户是否会在下个月违约，而老师在讲解分类算法时对类似的数据集进行过讲解，因此我们已经有了相应的基础，处理起来可以更加得心应手。
- 该数据集较大共有 30000 条数据，属性也比较多，众所周知，数据越多最终分类效果会更好。
- 该数据集有较强的应用背景，可以应用到实际的信用卡场景中。
- 小组成员都是女生，对扑克牌原理不大理解，所以就放弃了该数据集。

2. 数据集分析

首先看网站对数据集的介绍：

Data Set Characteristics:	Multivariate	**Number of Instances:**	30000	**Area:**	Business
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	24	**Date Donated**	2016-01-26
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	26274

可以发现该数据集共有 30000 条数据，24 个属性。其中一个为类别属性。

具体属性介绍如下：

Attribute Information:

This research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable. This study reviewed the literature and used the following 23 variables as explanatory variables:
X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
X2: Gender (1 = male; 2 = female).
X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
X4: Marital status (1 = married; 2 = single; 3 = others).
X5: Age (year).
X6 - X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.
X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.

X1:信用额度，连续值
X2:性别 离散值
X3:教育程度 离散值
X4:婚姻状态 离散值
X5:年龄 离散值
X6-X11:从 2005 年 4 月-9 月该用户拖欠月份长度，离散值
X12-X17:从 2005 年 4 月-9 月该用户账单总额，连续值
X18-X23:从 2005 年 4 月-9 月该用户换钱总额，连续值。

Default.payment.next.month:1 代表违约，0 代表不违约

从以上信息，可见该数据集属性较多，处理起来比较复杂，有必要探寻数据之间的关系以便进行预处理。

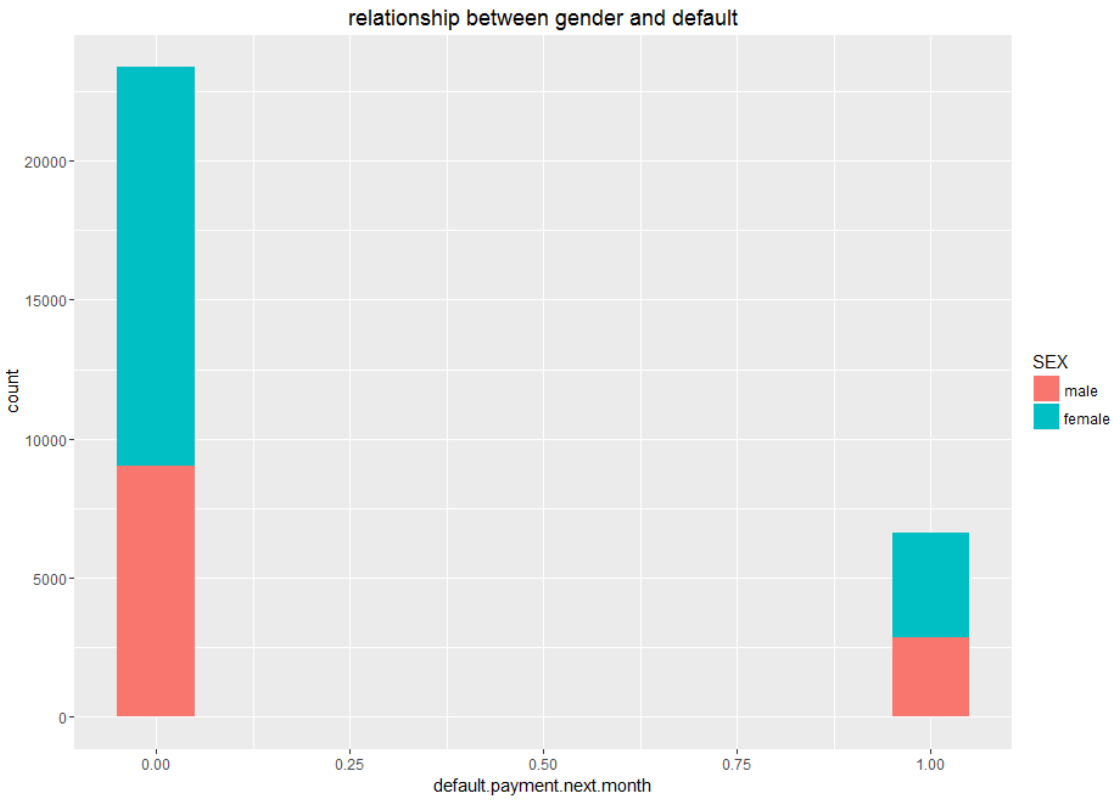
由于我们小组成员之前接触过 R 语言，R 语言的数据处理能力十分强大，因此此次我们选择使用 R 语言来探索数据。

1. 由于数据属性过多，我们首先检验属性与结果之间的**关联性**。

- 以**性别**为例，我们用 R 语言展示男性与女性在违约与不违约上的总数，为了更好的体现性别的特点，我们用柱状图把男性与女性画在一起，代码如下：

```
library(gdata)
credit_data=read.xls("mydata.xls",sheet=1)//读入数据
credit_data=tbl_df(credit_data)
sex_desc=c('male','female')
sec_vec=c('1','2')
credit_data$SEX=factor(x=credit_data$SEX,levels=sex_vec,labels=sex_desc)// 把SEX 这个属性转为因子，以便后面画图处理
ggplot(data=credit_data)+geom_histogram(aes(x=default.payment.next.month, fill=SEX),binwidth  = 0.1)//画出柱状图
```

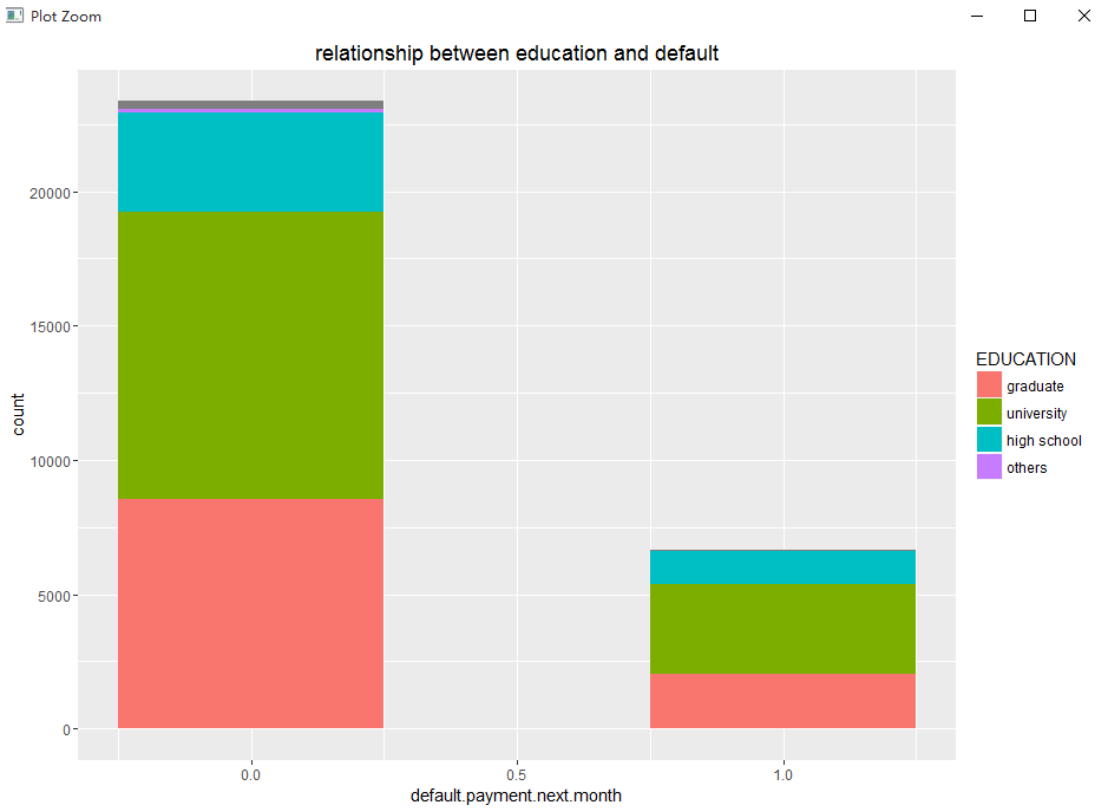
结果如下：



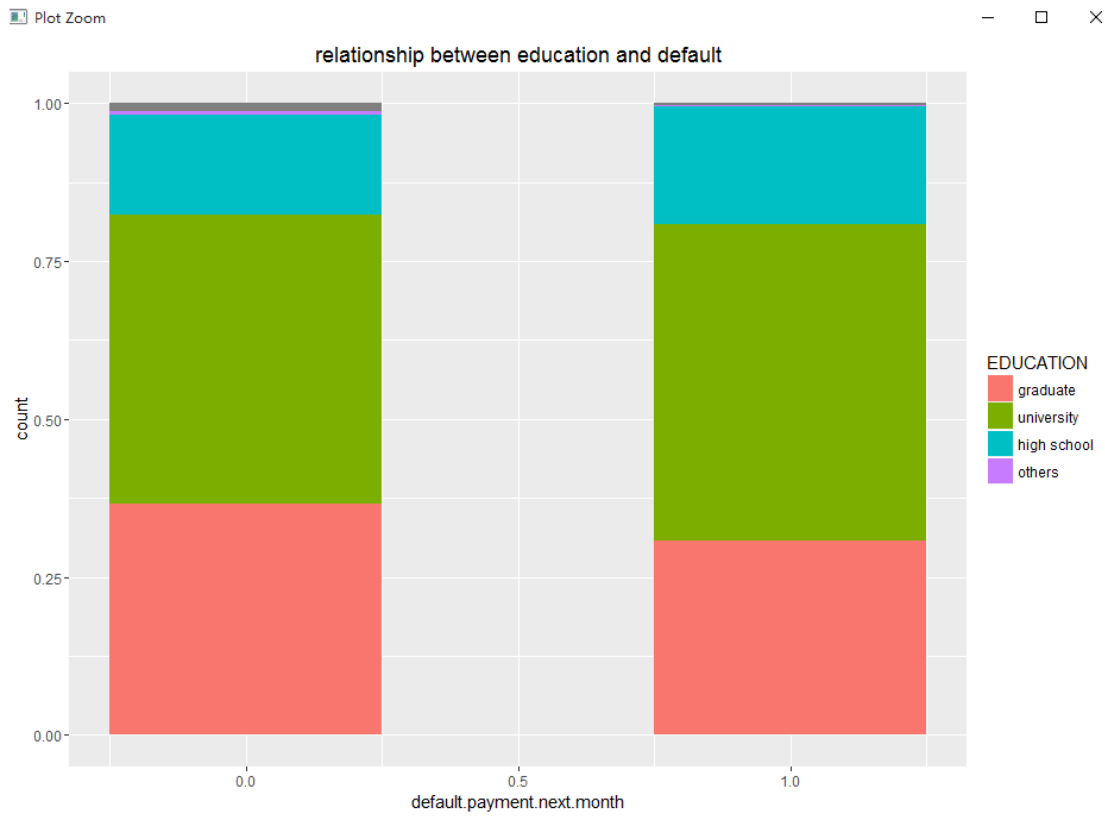
可以看出女性总数多于男性，但是女性违约的比例明显低于男性，可见性别对最终结果有一定的影响。

- **教育属性**

因为除了 graduate,university,highschool 外其余数量较少，我把他们都归到了 other 一类:



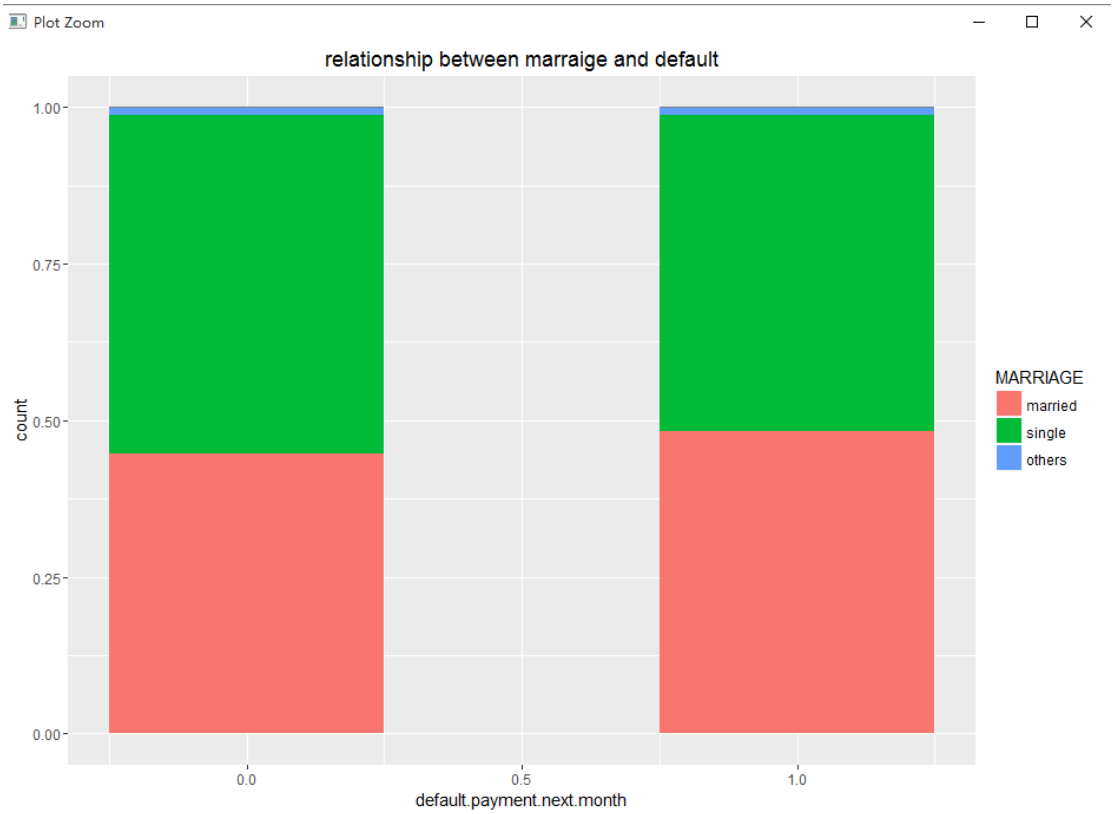
然而这时属性区分不是很明显，因此我们把它继续用**相对比例**展示：



这时，可以看出，学历越高，违约比例越小。可见教育程度也对结果有一

定影响。

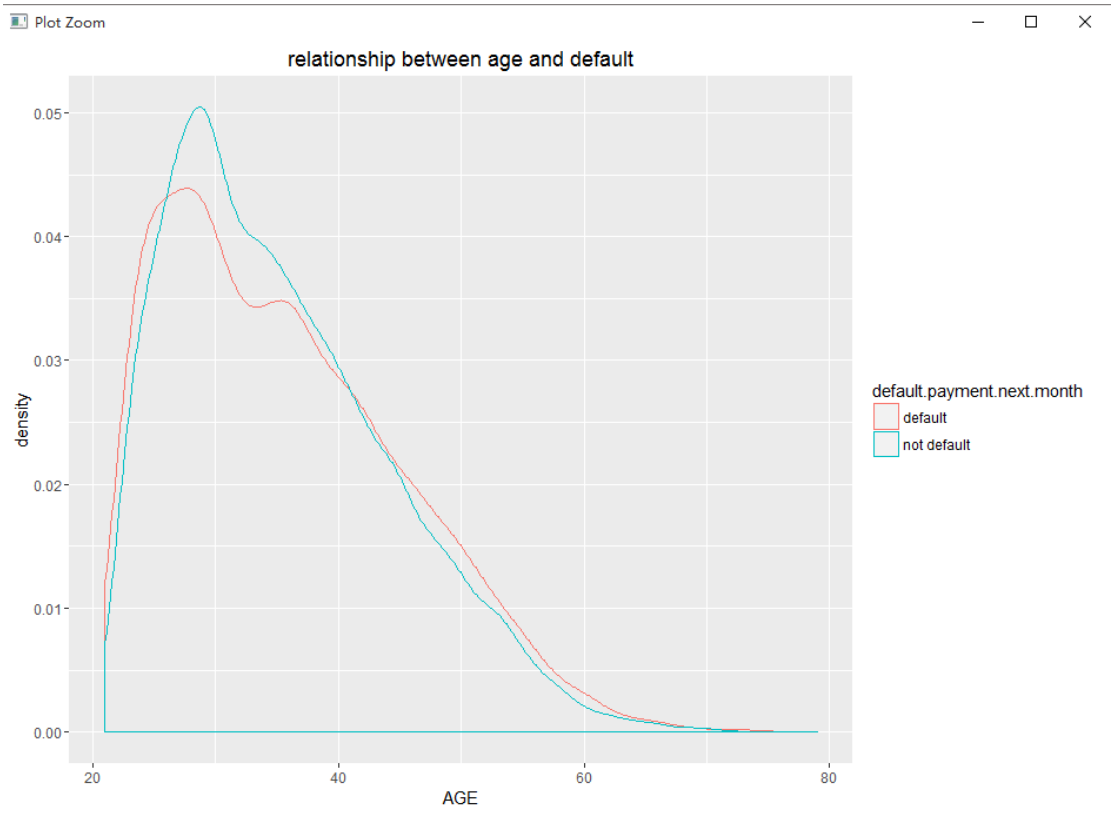
- **婚姻属性：**



可见结婚的人违约率比未婚的人高。

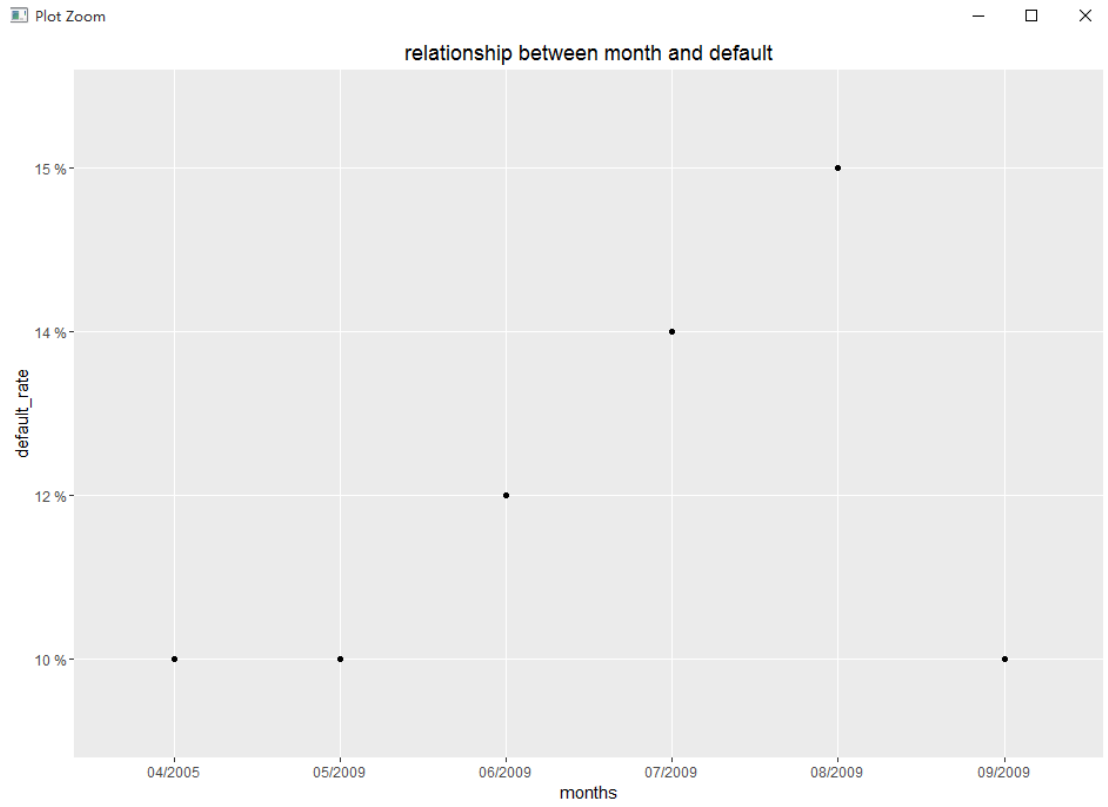
- **年龄属性：**

由于年龄为连续值，我们改用曲线来描述特性：



可见从 20-28 岁左右是违约率最高的年龄。

- **月份属性：**



我们计算了 2005 年 4 月到 9 月，违约时间超过一个月的人数所占比例，发

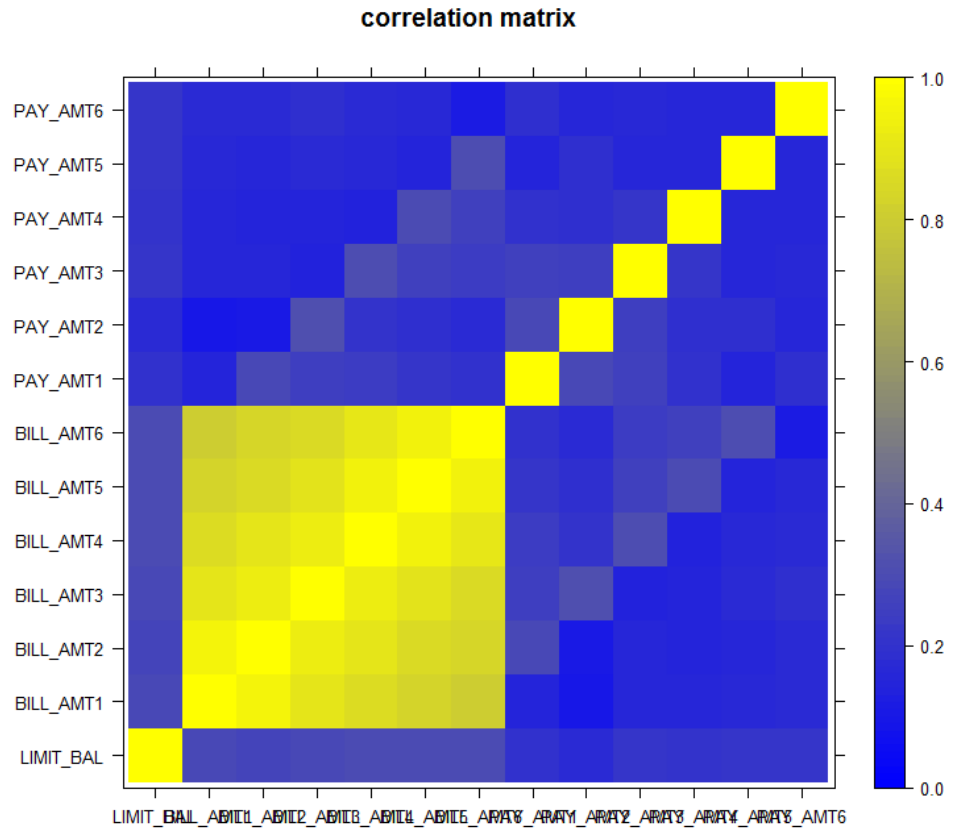
现 8 月比例相对最高。

我们可以发现，年龄，性别，婚姻状况，以及该月违约时间都与最终结果有一定的联系，因此不能随意舍弃。

因此思考后，我们决定从寻找属性的相关性着手，如果属性之间相关性非常强，那自然而然可以舍弃部分属性精简数据集。
我们计算 LIMIT_BAL,BILL_AMT1-BILL_AMT6.PAY_AMT1-PAY_AMT6 之间的**关联矩阵**，并以图表的形式给出，越黄代表关联性最强，代码如下：

```
cor=cor(credit_data) #计算属性相关性
levelplot(cor, col.regions=rgb.palette(120), cuts=13*13, at=seq(0,1,0.01))
```

结果如下：(越黄代表关联度越高)



可见 LIMIT_BAL 与 BILL_AMT1-6 之间的关联性非常强，而 PAY_AMT1-6 与其他属性没什么关联性，比较独立。

因此综上考虑，我们决定舍弃 BILL_AMT1-6 这六个属性，剩余 17 个属性进行训练。虽然属性依旧很多，但为了保持数据之间的关联性，这也是必然选择的一步路。至于预处理，请见第三部分。

二：分类算法的选择

由于网站上给出了一篇论文，作者分别对 KNN，逻辑回归，判别分析，朴素贝叶斯，

神经网络，决策树六种方法进行了训练。从结果可以看出，KNN，神经网络与决策树结果**相对**较为理想。

Table 1
Classification accuracy

Method	Error rate		Area ratio	
	Training	Validation	Training	Validation
K-nearest neighbor	0.18	0.16	0.68	0.45
Logistic regression	0.20	0.18	0.41	0.44
Discriminant analysis	0.29	0.26	0.40	0.43
Naïve Bayesian	0.21	0.21	0.47	0.53
Neural networks	0.19	0.17	0.55	0.54
Classification trees	0.18	0.17	0.48	0.536

从以上结果可以看出，整体分类准确率并不是特别理想，我们组猜测是因为该作者没有对数据进行预处理直接把所有属性数据扔到模型中进行训练

其实分析看来已经可以预测分类结果：
对于朴素贝叶斯：朴素贝叶斯最大的特点就是需要各个属性之间相互独立，由于作者并没有提到他对数据进行预处理，可以假定他并没有去除某些属性，而从刚才的关联图可以看出属性之间的关联性还是很强的，所以朴素贝叶斯训练得到的结果不如人意。
对于逻辑回归：逻辑回归适用于特征数目较少的数据集。而此次的数据集特征空间的维度较大，并且部分特征是非线性的，还需要对该些特性进行转换。因此逻辑回归也不在我们此次的考虑范围之内。

因此我们小组此次决定选择 **KNN，神经网络和决策树**来分别进行训练学习，在下一部分对数据进行预处理，训练后对三种模型做出相应的评估。
注意点：
由于该数据集比较特殊，要预测的是用户下一个月是否会违约，所以我们在之后评估模型时不能只看预测的准确率，而是要看我们是否准确预测了下一个月会违约的用户。这一点老师上课也提到过，因为这才是银行最关注的部分。所以在之后的处理中，我们会给出两个准确率，一个是预测成功准确率，另一个是准确预测下一个月会违约的用户的准确率，即假设实际上下一个月分类为 1 的用户共有 x 个，而我们预测他们的确是 1 的用户有 y 个，我们计算 y/x，即为结果。

三：数据预处理

- 首先再次说明，我们之后所有的数据只有 17 个属性。
- 我们先检查数据是否有缺失和错误。

使用 R 语言的 table 函数，可以计算每个属性中相应值的个数。
比如要计算 SEX 属性的分布：

```
table(credit_data$SEX);
```

计算可得：

```
> table(data$SEX)

    1     2
11888 18112
```
可见该属性没有缺失。

用其余相同方法对属性数据进行检查，发现属性没有缺失值，也没有重复。但是我们最后一列类别的属性名为 default payment next month.为了之后处理方便，我们把空格变成了.，变成 default.payment.next.month。

再次研究数据后，我们发现第一列是用户的信用额度，额度越高，说明这个用户越不会违约。然而由于每个用户的信用额度不同，自然而然其每个月的消费与归还金额也不同。从数据也可以发现，信用额度越高，消费的也越多，一般归还的金额相应的也越大。因此我们对数据做以下处理：

1. 对 EDUCATION 属性进行处理，原属性取值为 0，1，2，3，4，5，6.。1 代表研究生，2 代表大学生，3 代表高中生，4 属于其他，网站没有对 5，6，0 代表的含义进行说明，并且这三类数量也较少，因此我把 0，5，6 都归到了 4—other 这一类别，这样属性取值就只剩下 1，2，3，4。
2. 对 MARRIAGE 属性进行处理，原属性取值为 0，1，2，3，1 代表结婚，2 代表单身，3 为其他。网站没有对 0 进行说明，因此我把 0 归类到了 3 其他一类，这样属性取值就只剩下 1，2，3。
3. 对 PAY_AMT1-6 除以 LIMIT_BAL 做另一种概念的”归一化”处理，这样计算出的就是他每个月还钱的程度。
4. 归一化: 虽然归一化操作是数据挖掘必定要考虑的一步，但由于对于不同的训练算法是否选择这一步是不同的，因此这一步我们在之后具体情况具体分析。

遇到问题：

我们在预处理数据时，在是否要对六个月的违约时间进行平均化有了产生了犹豫。因为我们一开始考虑数据集属性实在太多，如果可以对违约时间进行平均那就可以再减少五个属性，但之后讨论后我们认为不能进行该步操作，原因是我们要预测的是用户**下一个月**是否会违约，而在第一部分我们看到每个月的违约率其实是会有较大的差异的，因此决策后我们还是保留了这六个属性，后续还是以 17 个属性进行训练。

四：模型生成

说明：我们挑选 27000 条作为 traing set，3000 条作为 test set。观察后发现 default 的分布较为均匀，因此不需要再做随机化数据的处理。

1.模型 1：KNN

在学习 KNN 时，老师就提到他最大的缺点就是在计算时需要计算所有属性之间的距离，时间耗费较长，但是由于这算是理论上最简单的一种模型，因此我们首选它作为我们的第一个模型。

KNN 数据预处理:

由于 KNN 是要计算属性间的距离，因此不可避免要进行数据的归一化：
我们采用 sklearn 包中的 preprocessing.MinMaxScaler 进行全部属性的归一化，十分方便快捷，代码如下：

```
min_max_scaler = preprocessing.MinMaxScaler()
data= min_max_scaler.fit_transform(array(data))
```

KNN 最大的问题在于确定 K。因此我们等间隔供选择了 7 个 k，分别为[4,7,10,14,20,24]。由于 knn 是没有训练的过程，因此可以直接拿 3000 条测试数据去与 27000 条数据计算距离确定类别。我们把对 3000 条预测的结果与实际对应的分类写入 txt 文件中，结果中的 result4.txt 代表 k=4 的预测结果。

代码如下：

```
def createDataSet():
    data = xlrd.open_workbook('data.xls')  #读取 excel 数据
    table = data.sheets()[0]
    data=[]
    labels=[]
    for i in range(1,30001):
        data.append(table.row_values(i)[0:17])
        labels.append(table.row_values(i)[17])
    min_max_scaler = preprocessing.MinMaxScaler() #进行归一化
    characters=array(data)
    return characters,labels
def classify(testSet,tlabels,trainSet,labels,k):
    Size=trainSet.shape[0]
    filename="Result"+str(k)+".txt" #将预测分类结果写入 result.txt 中
    f=open(filename,"w")
    for m in range(0,3000):
        diff=(tile(testSet[m],(Size,1))-trainSet)**2 #计算距离
        distances=(diff.sum(axis=1))**0.5
        sortedDistances=distances.argsort() #对距离进行排序
        Count={}
        for i in range(k):#
            vote=labels[sortedDistances[i]]
            Count[vote]=Count.get(vote,0)+1

        sortedClass=sorted(Count.items(),key=operator.itemgetter(1),reverse=Tr
        ue)#选择 k 个中个数最多的那个分类
        f.write(str(sortedClass[0][0])+","+str(tlabels[m])+"\n")#将结果写入文件
    f.close()
def predict(k):
    dataSet,labels=createDataSet()
    testSet=dataSet[27000:30000] #后 3000 条数据作为测试集
    testLabel=labels[27000:30000]
    trainSet=dataSet[0:27000]#前 27000 条数据作为标准集
    trainLabel=labels[0:27000]
    label=classify(testSet,testLabel,trainSet,trainLabel,k)
```

对结果进行分析统计：原本 3000 条数据中，共有 663 个类别为 1 的数据。

预测成功(即 0=0，1=1)的结果：这里以 error rate 作为结果，即预测失败的比率：

k	4	7	10	14	17	20	24
预测成功	2411	2436	2447	2468	2470	2459	2463
error rate	0.196333	0.188	0.184333	0.177333	0.176667	0.180333	0.179

可见 k 对于预测成功率的影响并不大，error rate 均在 18%-20%之间。

但是，就像之前提到的一样，该类数据集的关键不在于预测是否成功，关键在于原本是 default 即原本分类为 1 的有多少预测的确是 1，因为只有原本分类为 1 却预测错误这样的代价才是最大的。

因此我们继续计算原本为 1 预测类别也为 1 的概率：

之前提到 3000 条数据中，共有 633 个类别为 1。统计结果如下：

k	4	7	10	14	17	20	24
预测成功	177	237	198	211	228	204	205
rate	0.266968	0.357466	0.298643	0.31825	0.343891	0.307692	0.309201

可以看出成功率还是很低的，这时候 k 的作用就比较明显了，k=4 时准确率最低，只有 26.7%;k=7 时，准确率最高，达到了 35.7%.

花费时间：

Knn 由于计算量较大，平均花费 85s 左右的时间预测某个特定 k 下的分类。不同的 k 之间花费时间相近。

具体模型评估请见第五部分。

2.模型二决策树：

由于决策树老师上课时具体只讲了基本的理论与概念，因此我们上网搜寻后，发现决策树实际上有非常多的种类，例如 CART,IDS,C4.5,C5.0 等等。进行比对后我们了解到：

ID3 是最基本的决策树算法，但是他存在诸多问题：

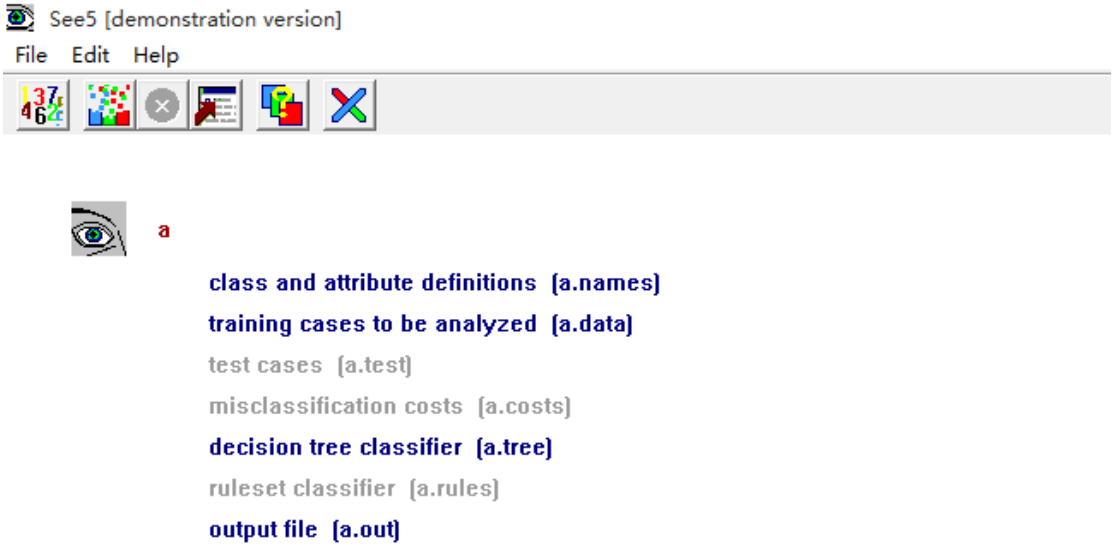
1. 在选择分支属性时，采用信息增益作为评价标准，倾向于选择取值的属性，而这些属性并不一定最多。
2. 对于我们此次数据集来说，ID3 最大的缺点在于它只能处理离散性描述属性。

因此，我们发现目前有了一种在 ID3 算法基础上实现的 **C4.5 决策树算法**，他利用信息增益率来选择属性，克服了 ID3 的第一个缺点；同时，它可以自动对连续属性进行离散化处理，克服了 ID3 的第二个缺点，比较适合我们此次的数据集。

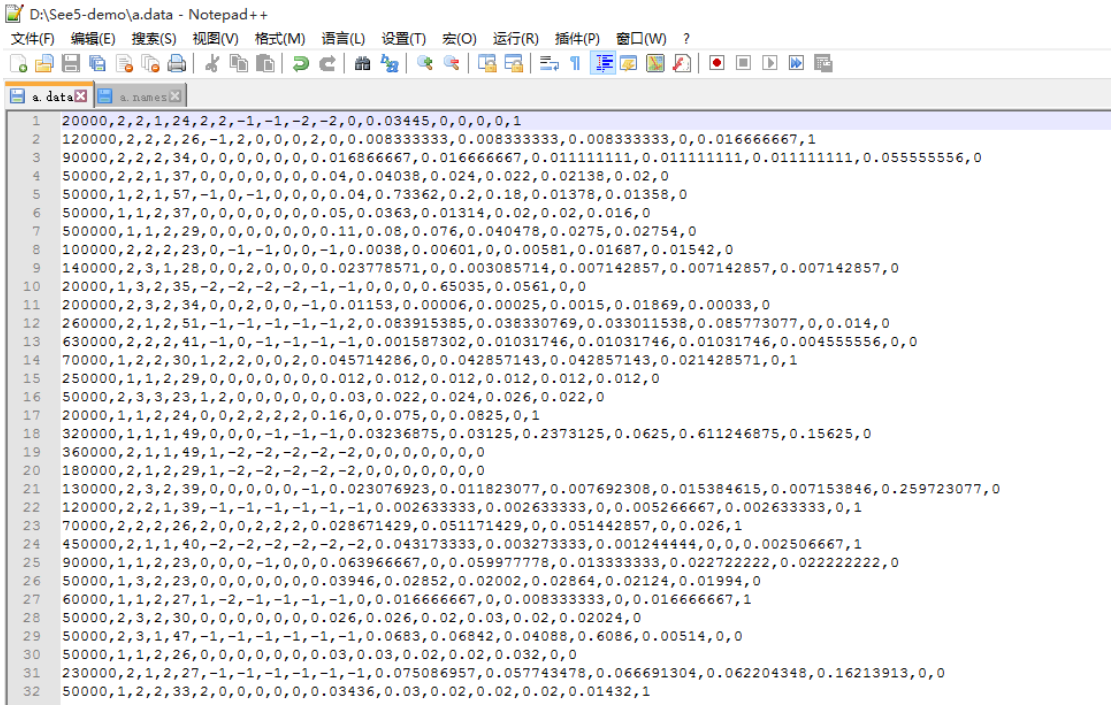
但是，C4.5 最大的缺点在于只适用于小的数据集，它在构造树的时候需要进行多次的顺序扫描和排序，算法比较低效，因此又与我们此次的数据集背道而驰。

继续努力搜索后，我们发现在 ID3 的基础上，还有一种新的决策树算法，即 **C5.0 决策树算法**，它在执行效率和内存使用上有了很强的改进，即能对连续值进行处理，也适用于大的数据集。

搜寻 C5.0 官网后，我们下载了 see5.0demo.exe 进行训练，这是一个 GUI 工具，用户无需自己编写代码就能对数据集进行分析处理。



阅读手册后发现，要用 see5demo.exe 进行训练，只需要一个.data 文件和一个.names 文件。.data 就是各个属性与分类结果，以,分割。.names 是对属性的介绍，是离散值呢还是连续值等等。结果如下：



D:\See5-demo\a.names - Notepad++

文件(F) 编辑(E) 搜索(S) 视图(V) 格式(M) 语言(L) 设置(T) 宏(O) 运行(R) 插件(P) 窗口(W) ?

a.data a.names

```
0, 1.           | classes to be predicted

LIMIT_BAL:      continuous.
SEX :   1,2
EDUCATION:  1,2,3,4.
MARRIAGE:   1,2,3.
AGE:            continuous.
PAY_1:  -2,-1,0,1,2,3,4,5,6,7,8.
PAY_2:  -2,-1,0,1,2,3,4,5,6,7,8.
PAY_3:  -2,-1,0,1,2,3,4,5,6,7,8.
PAY_4:  -2,-1,0,1,2,3,4,5,6,7,8.
PAY_5:  -2,-1,0,1,2,3,4,5,6,7,8.
PAY_6:  -2,-1,0,1,2,3,4,5,6,7,8.
PAY_AMT1:   continuous.
PAY_AMT2:   continuous.
PAY_AMT3:   continuous.
PAY_AMT4:   continuous.
PAY_AMT5:   continuous.
PAY_AMT6:   continuous.
```

用该工具进行处理后，发现很好的输出了决策树的分支结果。

Results for a

File Edit

```
PAY_1 = 3:
:...AGE <= 37: 1 (2)
:   AGE > 37: 0 (4)
PAY_1 = -1:
:...PAY_3 in {1,3,4,5,6,7,8}: 0 (0)
:   PAY_3 in {-2,-1}: 0 (56/5)
:   PAY_3 = 0:
:   :...PAY_AMT1 <= 0.01473: 1 (4)
:   :   PAY_AMT1 > 0.01473: 0 (7/1)
:   PAY_3 = 2:
:   :...AGE <= 40: 0 (8)
:       AGE > 40: 1 (3)
PAY_1 = -2:
:...PAY_AMT5 > 0.000676667: 0 (18)
:   PAY_AMT5 <= 0.000676667:
:   :...SEX = 1: 1 (2)
:       SEX = 2:
:       :...LIMIT_BAL <= 300000: 0 (5)
:           LIMIT_BAL > 300000: 1 (2)
PAY_1 = 2:
:...MARRIAGE in {0,3}: 1 (0)
:   MARRIAGE = 1: 1 (15/3)
:   MARRIAGE = 2:
:   :...PAY_5 in {-2,1,3,4,5,6,7,8}: 0 (0)
:       PAY_5 in {-1,0}: 0 (12/4)
:       PAY_5 = 2:
:       :...PAY_AMT6 <= 0.03538788: 1 (7)
:           PAY_AMT6 > 0.03538788: 0 (3)
PAY_1 = 1:
:...PAY_3 in {1,5,6,7,8}: 0 (0)
    PAY_3 in {-1,3,4}: 0 (19/5)
    PAY_3 = 0:
    :...PAY_AMT4 <= 0.00353125: 1 (3)
    :   PAY_AMT4 > 0.00353125: 0 (8)
    PAY_3 = 2:
    :...PAY_AMT3 <= 0.01268824: 0 (2)
    :   PAY_AMT3 > 0.01268824: 1 (9/1)
    PAY_3 = -2:
    :...PAY_6 in {0,1,2,3,4,5,6,7,8}: 0 (0)
        PAY_6 = -1: 0 (3)
        PAY_6 = -2:
        :...MARRIAGE in {0,3}: 0 (0)
            MARRIAGE = 1: 0 (7/2)
            MARRIAGE = 2:
            :...LIMIT_BAL <= 210000: 0 (6/2)
                LIMIT_BAL > 210000: 1 (4)
```

但是非常可惜的是，这个工具只能处理不超过 400 条的数据。如果需要处理大数据则需要购买完整版。

但我们小组不想放弃对 C5.0 模型的使用，因此继续搜寻后发现已经有人将 C5.0 的核心代码集成到了 R 语言中。因此花了一段时间研究后，我们继续在 R 平台上使用 C50 这个 libaray 进行对数据进行学习处理。因此我们继续转战 R 语言。

代码如下：

```
Library(C50);#载入 C50 包
train=credit_data[c(1:27000),]#取前 27000 条为训练数据
test=credit_data[c(27001:30000),]#取后 3000 条未测试数据
treeModel=C5.0(x=train[,-18],y=train$default.payment.next.month)#用 C5.0 决策树进行训练
summary(treeModel)#输出该 model 的情况
p=predict(treeModel,test)#进行预测
postResample(p,test$default.payment.next.month)
```

结果如下：可以看到很方便的输出了决策树的分支结果。

```
C5.0 [Release 2.07 GPL Edition]      Sun May 22 21:56:59 2016
-------------------------------

Class specified by attribute `outcome'

Read 27000 cases (18 attributes) from undefined.data

Decision tree:

PAY_1 <= 1:
:...PAY_2 > 1:
:   :...PAY_6 <= 0: 0 (1442/535)
:   :   PAY_6 > 0:
:   :   :...SEX <= 1: 1 (269/108)
:   :       SEX > 1: 0 (327/154)
:   PAY_2 <= 1:
:   :...PAY_1 <= 0: 0 (20448/2729)
:       PAY_1 > 0:
:       :...AGE <= 55: 0 (1620/401)
:           AGE > 55:
:           :...PAY_5 <= -2: 1 (35/8)
:               PAY_5 > -2: 0 (27/7)
PAY_1 > 1:
:...PAY_3 <= -1: 0 (170/74)
    PAY_3 > -1:
    :...PAY_5 > 0: 1 (1177/287)
        PAY_5 <= 0:
        :...EDUCATION <= 2: 1 (1145/355)
            EDUCATION > 2:
            :...EDUCATION > 3: 0 (19/4)
                EDUCATION <= 3:
                :...AGE <= 51: 1 (265/88)
                    AGE > 51: 0 (56/24)
```

```
Evaluation on training data (27000 cases):

        Decision Tree
      ----------------
      Size      Errors

        13 4774(17.7%)   <<


       (a)   (b)    <-classified as
      ----  ----
     20181   846    (a): class 0
      3928  2045    (b): class 1


    Attribute usage:

    100.00% PAY_1
     89.51% PAY_2
     10.49% PAY_3
     10.09% PAY_5
      7.55% PAY_6
      7.42% AGE
      5.50% EDUCATION
      2.21% SEX


Time: 1.8 secs
```

可以看到，一共花费 1.8s 的时间生成决策树。决策树共使用了 9 个属性进行分类，分支结果在上图最前面也有所体现。

在进行上述实验时，发现有一个选项为 trial，查询后发现 C50 这个库可以直接进行十折交叉验证。这是用来测试算法准确性的常用的测试方法。它将数据集分成十分，轮流将其中 9 份作为训练数据，1 份作为测试数据，进行试验。每次试验都会得出相应的正确率（或差错率）。10 次的结果的正确率（或差错率）的平均值作为对算法精度的估计。

进行十折交叉验证：

```
treeModel=C5.0(x=train[,-18],y=train$default.payment.next.month,trials = 10)
p1=predict(treeModel,test)
postResample(p1,test$default.payment.next.month)
```

得到结果:

```
C5.0 [Release 2.07 GPL Edition]      Sun May 22 22:06:19 2016
-------------------------------

Class specified by attribute `outcome'

Read 27000 cases (18 attributes) from undefined.data

-----  Trial 0:  -----

Decision tree:

PAY_1 <= 1:
:...PAY_2 > 1:
:   :...PAY_6 <= 0: 0 (1442/535)
:   :   PAY_6 > 0:
:   :   :...SEX <= 1: 1 (269/108)
:   :       SEX > 1: 0 (327/154)
:   PAY_2 <= 1:
:   :...PAY_1 <= 0: 0 (20448/2729)
:       PAY_1 > 0:
:       :...AGE <= 55: 0 (1620/401)
:           AGE > 55:
:           :...PAY_5 <= -2: 1 (35/8)
:               PAY_5 > -2: 0 (27/7)
PAY_1 > 1:
:...PAY_3 <= -1: 0 (170/74)
    PAY_3 > -1:
    :...PAY_5 > 0: 1 (1177/287)
        PAY_5 <= 0:
        :...EDUCATION <= 2: 1 (1145/355)
            EDUCATION > 2:
            :...EDUCATION > 3: 0 (19/4)
                EDUCATION <= 3:
                :...AGE <= 51: 1 (265/88)
                    AGE > 51: 0 (56/24)
```

```
-----  Trial 1:  -----

Decision tree:

PAY_2 <= 1:
:...PAY_4 <= 0: 0 (20936/5847.7)
:   PAY_4 > 0: 1 (1359.4/655.6)
PAY_2 > 1:
:...PAY_AMT6 <= 0.273: 1 (4616.7/2037.1)
    PAY_AMT6 > 0.273: 0 (87.8/30.3)

-----  Trial 2:  -----

Decision tree:

EDUCATION > 3: 0 (327.9/53.3)
EDUCATION <= 3:
:...PAY_1 <= 0: 0 (19154.5/6377)
    PAY_1 > 0:
    :...PAY_4 <= -2: 1 (1146.6/450.8)
        PAY_4 > -2:
        :...PAY_2 <= 0: 1 (1948.2/901.2)
            PAY_2 > 0: 0 (4422.8/1964.6)
```

.....

```
Evaluation on training data (27000 cases):

Trial       Decision Tree
-----     ----------------
          Size      Errors

   0        13 4774(17.7%)
   1         4 5843(21.6%)
   2         5 6450(23.9%)
   3         8 6770(25.1%)
   4        13 6415(23.8%)
   5        10 6935(25.7%)
   6         4 5465(20.2%)
   7         8 6019(22.3%)
   8         3 4873(18.0%)
   9        11 5136(19.0%)
boost          4828(17.9%)   <<


           (a)   (b)    <-classified as
          ----  ----
         19950  1077    (a): class 0
          3751  2222    (b): class 1
```

```
Attribute usage:

100.00% EDUCATION
100.00% PAY_1
100.00% PAY_2
 98.92% PAY_4
 98.62% PAY_6
 98.60% PAY_3
 98.51% LIMIT_BAL
 94.67% PAY_5
 79.25% PAY_AMT4
 77.07% PAY_AMT1
 43.07% PAY_AMT2
 30.02% MARRIAGE
 20.44% AGE
 17.49% PAY_AMT3
 14.72% PAY_AMT6
  2.21% SEX


Time: 13.9 secs
```

结果：

共花费 13.9s 时间，共使用了 16 个属性，十次的决策树太庞大，我将结果放在了 C5.0 决策树目录下的 result.txt 下。

进行预测后，准确率对比：

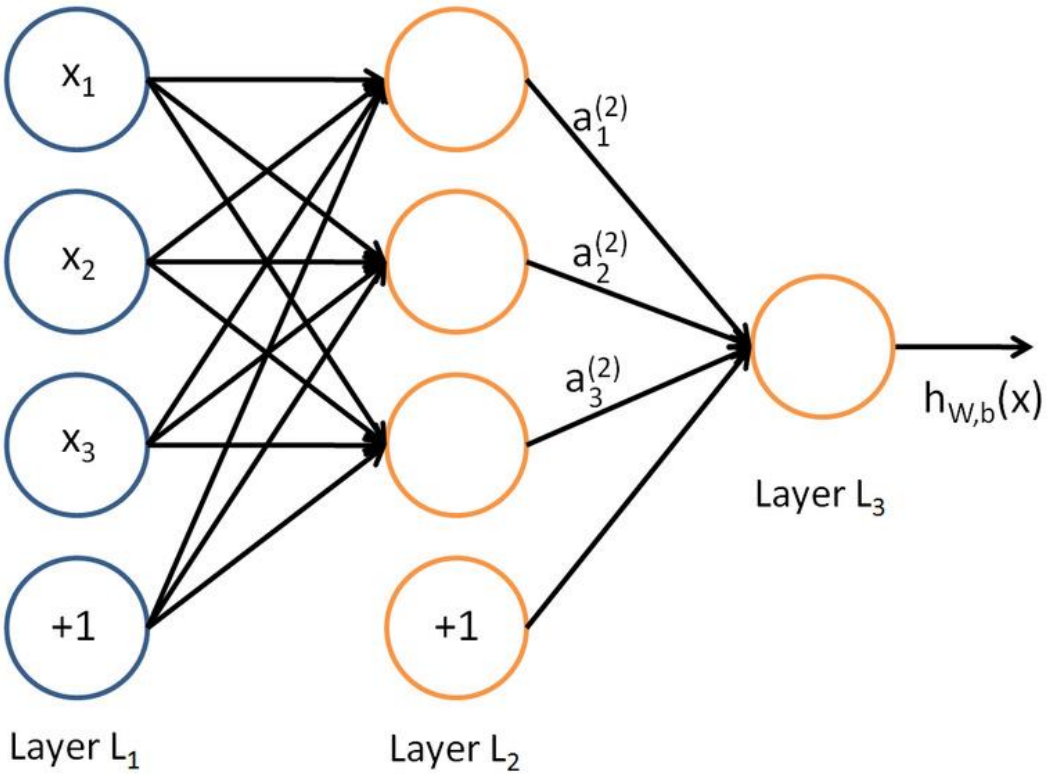
	C5.0	C5.0加十折交叉验证
error rate	17.80%	17.40%
预测1的准确率	32.30%	36.30%

具体模型评估请见第五部分。

3. **模型三 神经网络**

最后我们以论文中认为最佳的方法神经网络进行学习训练。由于对神经网络不是太了解，我们选择了最普及的 BP 神经网络来进行学习。

BP 神经网络简单介绍:



一共有三层，分别为输入层，隐层，输出层。

输出层即为最后的结果，神经网络最后不是直接输出 0 或 1，而是输出 0-1 中的某个值，需要我们自己设定一个阈值。

输入层的每个节点为相应的属性。

+1 是偏置节点，偏置节点不接受输入，输出总是+1。

BP 可以分为前向传播与后向传播两个过程，后向传播是在训练过程中，用误差来调整各层之间的权重，前向传播是利用训练得到的模型，即各层参数来进行实际的预测。

在前向传播即预测时：

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

f()为 Sigmoid 函数，即把结果归到 0-1 之间。

在后向传播即训练时，利用按误差函数梯度下降的方向进行收敛，即将误差结果对参数进行求导然后进行调整。梯度下降算法是一种最优化算法，比如我们要求一个函数的最小值，先求它的导数，即梯度，然后更新 x=x-n*delta，n 为学习率，delta 为梯度，这样如果步长取得比较合适，可以保证每一次迭代结果都会减少最后收敛。因此我们这里可以构造误差函数，对其进行梯度下降算法进行训练。

总的步骤如下(推导过程省略)：

1.根据参数，计算输出节点的 delta,O 代表节点输出值，t 代表实际即目标值。

$$\delta_k = \mathcal{O}_k(1-\mathcal{O}_k)(\mathcal{O}_k - t_k)$$

2.计算隐藏节点的 delta:

$$\delta_j = \mathcal{O}_j(1-\mathcal{O}_j)\sum_{k\in K}\delta_k W_{jk}$$

3.计算参数改变值，n 代表学习率

$$\Delta W = -\eta\delta_\ell \mathcal{O}_{\ell-1}$$

$$\Delta\theta = -\eta\delta_\ell$$

4.更新权值参数即可。

在此基础上，我们改进了学习率，设置了 N,M 两个学习率，我们考虑最近两次梯度，这里 N 是该次梯度的学习率，M 是上次训练梯度更新的学习率。在查询相关资料后，我们设定 N=0.1,M=0.08，在单一学习率的基础上有了改进。

在**确定隐层节点**个数上，我们参考了网上的资料：

$m = \sqrt{n+l} + \alpha$

$m = \log_2 n$

$m = \sqrt{nl}$

m：隐含层节点数

n：输入层节点数

l：输出层节点数

α：1--10 之间的常数。

因此我们确定隐层节点在 4-5 之间，我们选取 4 作为隐层节点个数。

代码如下：这里的代码参考了 **Neil Schemenauer 所写的 BP 神经网络样例**，给出关键代码：

```
def runNN(self, inputs):#前向预测
        if len(inputs) != self.ni - 1:
            print 'incorrect number of inputs'
        for i in range(self.ni - 1):
            self.ai[i] = inputs[i]
        for j in range(self.nh):
            sum = 0.0
            for i in range(self.ni):
                sum += ( self.ai[i] * self.wi[i][j] )
            self.ah[j] = sigmoid(sum) #得到隐层节点值
        for k in range(self.no):
            sum = 0.0
```

```
            for j in range(self.nh):
                sum += ( self.ah[j] * self.wo[j][k] )
            self.ao[k] = sigmoid(sum) #得到输出层节点值
        return self.ao #得到预测结果
    def backPropagate(self, targets, N, M):#后向传播
        output_deltas = [0.0] * self.no
        for k in range(self.no):
            error = targets[k] - self.ao[k]
            output_deltas[k] = error * dsigmoid(self.ao[k])#  计算输出层梯度
        for j in range(self.nh):#  更新输出层权值
            for k in range(self.no):
                change = output_deltas[k] * self.ah[j]
                self.wo[j][k] += N * change + M * self.co[j][k]
                self.co[j][k] = change
        hidden_deltas = [0.0] * self.nh #  计算隐藏层 deltas
        for j in range(self.nh):
            error = 0.0
            for k in range(self.no):
                error += output_deltas[k] * self.wo[j][k]
            hidden_deltas[j] = error * dsigmoid(self.ah[j])
        for i in range(self.ni):#  更新输入层权值
            for j in range(self.nh):
                change = hidden_deltas[j] * self.ai[i]
                self.wi[i][j] += N * change + M * self.ci[i][j]
                self.ci[i][j] = change
        error = 0.0
        for k in range(len(targets)): #计算误差值
            error = 0.5 * (targets[k] - self.ao[k]) ** 2
        return error
```

最终将最后 3000 个数据作为测试集，程序输出实际分类和预测概率到 result.txt 中。
结果统计如下：

我们先对进行 1000 次迭代和 2000 次迭代进行了对比，均取 0.5 作为阈值。

迭代次数	1000次	2000次
error rate	17.60%	17.70%
预测为1的成功率	36.80%	36.80%

我们发现 1000 次与 2000 次结果基本相似，说明已经收敛。
之后我们将迭代次数固定为 1000 次，取不同的阈值进行比较。

阈值	0.5	0.4	0.3	0.25	0.2
error rate	17.60%	17.80%	18.60%	19.90%	21.80%
预测为1的成功率	36.80%	40.90%	47.40%	52.60%	57.40%

1000 次迭代共花费时间：8765s,约 2.4 个小时。

在与其余模型进行对比后，我们选择 0.4 作为阈值。具体分析请见请见第五部分。

五：模型评估

纵观 KNN，决策树，神经网络这三个模型，我们发现其实对于 error rate 这一项指标来说，三种模型都在 80%-82%之间，相差并不是很大，这一点与前面的论文给出的结果也比较类似。看来的确没有上升的空间。

然而在对于 1 这个分类，即是否对违约的用户预测准确上，KNN<决策树<神经网络。而在时间花费上，决策树<knn<神经网络。以下我们分别对这三个模型进行评估：

1.KNN

优点：

1. 对于 KNN 来说，我们小组公认它是逻辑最简单处理最方便的一种数据挖掘算法。
2. 它不需要进行训练就可以直接进行预测。

缺点：

1. 它最大的弊端就是计算量较大，虽然相比神经网络来说它的处理时间还是比较少的。但我们还要考虑的一个问题就是**所占内存**的大小，对于 KNN 来说，每来一条新的数据就要把所有标准样本放到内存中跟它去计算距离来进行预测，所以内存开销还是比较大的。
2. KNN 还是更加适合多分类。我们看到这里结果中，KNN 在预测违约用户的准确率上是最低的，因为相比来说标准样本中 0 的个数还是远多于 1 的，所以较难预测准确 1 这个分类。

进行分析后，我们认为，此次在进行 KNN 处理时，我们并没有对属性赋予相应的权重，因为我们把这一内容分到了后面的模型中进行处理。但我们猜想对于 KNN 来说，如果我们对属性赋予权重的话，结果应该可以更好。

2.C5.0 决策树

决策树对于我们这个数据集是一个比较良好的模型。可以看到不管是在他的准确率，还是预测违约的用户的准确率，以及训练时间上都比较良好。并且它的处理速度更快，内存也占用的较少。

同时我们看到了十折交叉验证的重要性，经过十折交叉验证后，两项准确率都有所提高。

但这里还要讨论另一个问题，先看一下这张结果图：(前面曾出现过)

```
Evaluation on training data (27000 cases):

        Decision Tree
      ----------------
      Size      Errors

        13 4774(17.7%)   <<


       (a)   (b)    <-classified as
      ----  ----
     20181   846    (a): class 0
      3928  2045    (b): class 1


    Attribute usage:

    100.00% PAY_1
     89.51% PAY_2
     10.49% PAY_3
     10.09% PAY_5
      7.55% PAY_6
      7.42% AGE
      5.50% EDUCATION
      2.21% SEX


Time: 1.8 secs
```

训练完这个后我们发现，决策树其实只用了 8 个属性，于是我们猜想是否后 6 个属性其实对结果影响不大。因此我们又用 R 语言使用**主成分分析**的方法，得到以下结果：

```
$values
 [1] 4.4793614 1.7349172 1.4903628 1.0708473 0.9920855 0.9
304677 0.9201997 0.8946190 0.8503507
[10] 0.8338201 0.6878044 0.6053900 0.5508545 0.3962074 0.2
521031 0.1837260 0.1268831
```

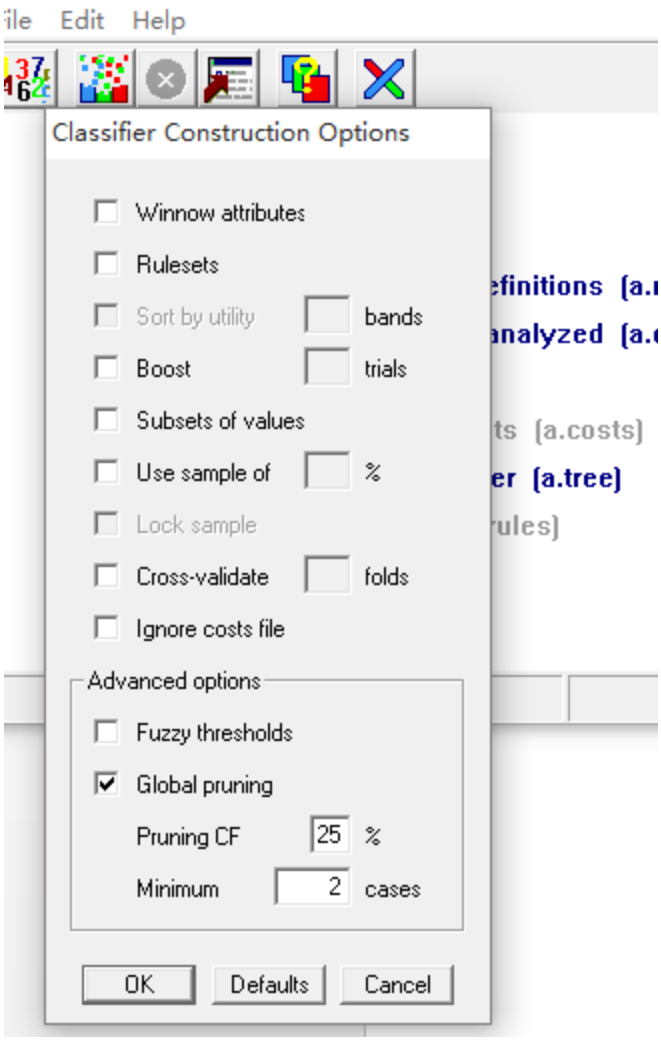
可以看到，确实后面 6 个属性对结果的影响几乎可以忽略，因此决策树这个分类还是合理可以解释的通的。

当然我们看到十折交叉验证用了更多的属性：

```
-----     ----------------
      Size      Errors

   0         13 4774(17.7%)
   1          4 5843(21.6%)
   2          5 6450(23.9%)
   3          8 6770(25.1%)
   4         13 6415(23.8%)
   5         10 6935(25.7%)
   6          4 5465(20.2%)
   7          8 6019(22.3%)
   8          3 4873(18.0%)
   9         11 5136(19.0%)
boost           4828(17.9%)   <<
       (a)    (b)    <-classified as
      ----   ----
     19950   1077    (a): class 0
      3751   2222    (b): class 1
    Attribute usage:

    100.00% EDUCATION
    100.00% PAY_1
    100.00% PAY_2
     98.92% PAY_4
     98.62% PAY_6
     98.60% PAY_3
     98.51% LIMIT_BAL
     94.67% PAY_5
     79.25% PAY_AMT4
     77.07% PAY_AMT1
     43.07% PAY_AMT2
     30.02% MARRIAGE
     20.44% AGE
     17.49% PAY_AMT3
     14.72% PAY_AMT6

Time: 8.5 secs
```

所以难怪十折交叉验证的准确率会更高。

展望：



在之前的 demo.exe 中，我们发现其实 C5.0 决策树还可以搭配 adaboost 来进行训练，我们研究了 adaboost 后，认为如果把这两个进行搭配，那么决策树的准确率大大提高，并且速度可以更快，同时也会减少 Overfitting 的概率。但此次由于时间的关系我们没有继续这个方向，所以将此放在了展望中。

3.BP 神经网络

在论文中，作者认为神经网络是对于此数据集更好的分类方法，我们得出的结果也的确如此。他的分类准确率很高，尤其是在对 1 这个分类的预测上准确率相比其他两个模型提高较多，并且由于神经网络本身的特性，它的鲁棒性与容错性也较强。

但我们在此次处理过程中也发现，虽然其原理比较好理解，但在实际过程中有很多问题需要考虑，比如使用多少层进行训练，隐层节点放置多少，最终阈值究竟取多少都需要通过观察学习过程来继续调整。此次我们采用最通用的参数来进行训练，并且根据最终的准确率来调整阈值，最终确定阈值为 0.4。同时我们发现 BP 训练时间过长，需要几个小时才能训练结束，此次因为我们采用比较保守的 1000 次与 2000 次对比，观察后发现 1000 次已经开始收敛因此准确度可以保证。

未来可以在调整参数上做更多努力，比如确定层数，隐层节点个数等等。在和同学交流后发现，其实该数据集在 100 次迭代后已经可以达到较高的准确度，所以

在实际应用中如果对时间要求较高可以适当以牺牲准确率为代价减少迭代次数。

4.总结：

总的来说，神经网络从结果上来看确实是最佳的算法，与论文作者给出的结论一致。但作者并没有提到时间的问题，如果综合考虑时间与准确率，我们认为 C5.0 才是针对该数据集最佳的算法。当然考虑到可以事先用神经网络进行训练得到权重矩阵，预测时只需要与权重矩阵进行相应的计算即可，那神经网络依旧是较好的选择。

六：评价：

做完整体实验后，我们认为自己的准确率还是较低，的确无法应用到实际场景。在和同学们进行交流后发现所有同学的准确率其实都接近，而大家的数据预处理方法也无非就是降维归一化等等。所以我们小组认为，之所以准确率这么低，关键在于数据预处理的过程。比如对于账单金额与归还金额这两个属性，我们此次在预处理中直接省略了账单金额，并对归还金额在信用额度的基础上做了归一化。但最后进行分析认为，归还金额与账单金额之间的比例也会对最后的结果造成影响。因此我们认为，此次大作业我们在数据预处理这一块上还有改进的空间。

七：感想

做完这次大作业，感触很多很深。之前两个小组成员都没有接触过类似的课程，在上课大致听懂各种算法的原理后，本来以为数据挖掘的任务还是比较简单的，只要对数据预处理一下再放到模型中跑就可以。但实际上手操作才发现竟然有这么多的问题。首先在读懂属性上我们就花费了很长一段时间，之后在如何对数据预处理上也产生了僵局，再到最后的模型选择，参数选择等等。做完整个实验我觉得数据挖掘最大的难度就在于选择太多，我们会面临无数种的选择，然而只有根据这个数据集的特性我们才能做出最正确的选择。但由于我们在这方面都是新手，无法做出准确的判断，因此我们此次在参考论文的基础上对三种算法都进行了实验得出了相应的结论。虽然最终结果不是很好，但在整个过程中我们体会到了整个数据挖掘的流程，对各种分类算法有了更深入的了解，在实践过程中得到了锻炼。

同时，这个大作业也提高了我们解决问题的能力，对一些工具以及 R 语言，python 语言的使用有了更高层次的掌握。

当然很遗憾的是，此次大作业由于时间问题在很多方面都没有考虑完善，如果有机会的话可以继续在这个数据集上进行深入的研究，以便用于实际场景中。

最后要特别感谢老师上课的指导与对问题的解答！