|  |
| --- |
| 辨析分类器和回归模型，分析问题是推断还是预测：  输出变量为数值型，则称之为回归；  输出变量为文字型，则称之为分类；  输入变量与输出变量均为连续变量的预测问题是回归问题；  输出变量为有限个离散变量的预测问题是分类问题；  分类和回归的区别在于输出变量的类型；  推断是根据事实或前提进行推论；  预测是根据已知因素预先估计未来的发展趋势； |

|  |
| --- |
| a=read.csv()读取数据  fix(a)观察数据  rownames(a)=a[,1]剔除第一列  summary()统计汇总信息  pairs()产生一个散点图矩阵  a[,1:10]提取前10列  plot()产生沿边箱线图  e=rep('no',nrow(a))  e[college$top10>50]='yes'  e=as.factor(e)  a=data.frame(a,e)  hist()对定量变量尝试不同的组数制作直方图  par(mfrow=c(2,2))将打印窗口分成四个矩形区域，连续在一个图形窗口上绘制四个图。  探究：模型拟合  plot(a$top10,a$grad)  model=lm(a$top10~a$grad,data=a)  summary(model)  anova(model)  abline(model,col=1,lty=1) |

|  |
| --- |
| 推导岭回归的解析解：&推导单变量lasso解析解公式（即软阈因子） |
| 线性回归的原理都是最小二乘，是一种简单方便的算法，其核心是其中，不等于0，即要可求逆。  当变量之间的相关性较强，具有多重共线性时，或者特征数大于样本数，X不是满秩矩阵，会使得的结果趋近于0，造成拟合参数的数值不稳定性增加，参数间的差距变化很大。  普通最小二乘法带来的局限性，导致很多时候都不能直接使用其进行线性回归拟合，尤其是：  ①数据集的列（特征）数量>数据量（行数），即X不是列满秩。  ②数据集的列（特征）数据之间存在较强的线性相关性，即模型容易出现过拟合。 |
| 假设，  因为  所以  令，得 |
| 岭回归和lasso回归可解决最小二乘法的一些缺点。  岭回归可看作是一种改良后的最小二乘法，通过向损失中添加L2范数有效防止模型出现过拟合，且可以解决非满秩条件下求逆困难的问题，从而提升模型的解释能力。  岭回归两个优点：   1. 防止了线性回归中不可逆的情况；   ②引入了惩罚项防止过拟合。  L损失函数；Y目标变量；X特征矩阵；W系数 |
| 损失函数为：  令，得，E是一个单位矩阵。 |
| 线性回归中，L(w)是经验风险，在经验风险的基础上加上表示模型复杂度的正则项或惩罚项，即结构风险。  线性回归是经验风险最小化，岭回归是结构风险最小化。  与岭回归相似的，lasso回归是通过添加L1正则化来改进普通最小二乘法。  lasso回归不是从根本上解决多重线性问题，而是限制多重共线性带来的影响，lasoo回归可直接用于作特征选择。 |
| 立，为了方便求导，特加入  将上式x,y进行标准化  从而，  其中，为符号函数  当时，  当时，  剩余的，若,取  从而，  简写为,其中,称为软阈算子 |

|  |
| --- |
| 实验分层原则规定，如果模型中包含交叉项，即使主效应的系数p值不显著，也应将其包含在模型中。  如果数据进行了标准化，可直接对系数做比进而比较功效大小。  交叉项系数为正，自变量对因变量的影响随着调节变量的增强而增强；  交互项系数为负，自变量对因变量的影响随着调节变量的增强而减弱；  故而交互项系数衡量的是自变量对因变量的影响，而不是两个自变量之间的交互作用。  交互项系数的大小，与二者之间是否存在交互作用不能构成因果关系。  要验证是否存在交互关系，应当对交互项对应的系数进行假设检验，根据检验结果进行判断。  如果判断结果是拒绝系数为0的原假设，即使交互项系数再小，二者之间也仍然是存在交互作用的。  交互项系数小并不代表二者之间没有交互作用，或可利用交互项系数的显著性来判断二者的交互作用？  某个变量是否对回归方程有影响作用，应该针对该交互项的系数进行回归系数检验，检验显著可认为存在交互作用；  另，即使系数比较小，但交互项对整个方程计算结果有不可忽略的影响时，也不可剔除。  对数据未进行标准化之前，不能只看系数的大小对变量的作用大小进行解释。  对于交互项是否起作用，可以进一步进行模型系数检验。 |

|  |
| --- |
| 多元回归模型  lm\_mutli=lm(sales~price+urban+us,data=carseats)  summary(lm\_multi)  定性变量，  模型系数  回归模型显著性检验  p值接近为0，拒绝原假设，  p值不具有显著性，可去掉该变量，  模型系数的置信区间：  confint(lm\_multi)  模型中的离群点与高杠杆点：  残差图：残差与真实值之间的关系图；  正态检验：Normal QQ用来检测其残差是否是正态分布的，  方差假设检验：Scale-Location用来检查等方差假设的  杠杆图，检查数据分析项目中是否有特别极端的点，  离群点，残差比一般的值更大，  高杠杆点，自变量因子空间中的离群点。  纵轴绝对值大于2的为离群点；横轴大于0.02的为高杠杆点。  R的car包中的influence函数可找出离群点和高杠杆点：  install.packages("car")  library(carData)  library(car)  influencePlot(fit\_new,main="Influence Plot")  纵坐标绝对值大于2被认为离群点，横坐标大于0.02被认为高杠杆点，圆圈大小与影响成比例，圆圈大的点可能是对模型估计造成很强不良影响。 |
| 在一个回归模型中个，异常值点包括离群点，高杠杆点和强影响点，这些点都可能对结果产生较大的负面影响，因此对异常值点的判断及修正对建立正确的回归模型非常重要。  1、离群点，  通常指残差非常大的点，模型预测的y值与真实的y值相差非常大。  通常监测离群点的方法有：  ①用qq图监测，落在置信区间外的点通常被认为是离群点；  ②通常认为标准化残差的绝对值大于2的点可能是离群点，或大于3；  对于离群点，一般会选择删除，删除离群点还有利于提高数据集对于正态分布假设的拟合度。  # 绘制标准化残差图  plot(rstandard(lm2))  2、高杠杆点，  指的是x值比较异常，通常与响应变量值y没有关系。  判断高杠杆点的方法是，计算点的帽子统计量，若该点的帽子统计量大于帽子统计量的均值的2或3倍，通常被认为是高杠杆点，  帽子矩阵的计算：H=X(XTX)^-1XT  帽子统计量就是矩阵H对角线上的元素。  # 绘制帽子统计量图  plot(hatvalues(lm2))  # 计算帽子统计量的均值  mean(hatvalues(lm2))  3、强影响点  高杠杆点，同时是离群点，就是强影响点。  强影响点是指对统计推断有影响的点，一般用cook距离进行判断，若cook距离的值大于4/(n-k-1)，则表明是强影响点。  influenceplot()函数，可以把离群点，高杠杆点，强影响点都整合在一个图上，影响图的横坐标为帽子值，纵坐标为学生化残差，因此纵坐标超过+-2的点被认为是离群点，横坐标用于判断高杠杆点，越往右上角的点，越有可能是强影响点。 |
| 变量系数 Estimate；变量系数标准误 Std. Error；T检验值 t value；T检验p值 Pr(>|t|)；均方根误差 Residual standard error；判定系数 R-squared；调整判定系数 Adjusted R-squared；F检验值 F-statistic；F检验p值 p-value |
| #####判断是否含有高杠杆值点或离群点  #通过帽子 统计量(hat statistic)判断，对于一个给定的数据集， 帽子均值为 p/n，其中  p 是模型估计的参数数目（包含截距项）， n 是样本量，一般来说，若观测点的帽子值大于  帽子均值的 2 或 3 倍，就可以定位的高杠杆值，下面画出了帽子值的分布  hat.plot <- function(x){  p <-length(coefficients(x)) #系数个数，包括截距项  n <- length(fitted(x)) #样本数  plot(hatvalues(x), main = 'Index Plot Of Hat Values') #绘制帽子值分布  abline(h = c(2,3) \* p/n, col = 'red', lty = 2) #查看高杠杆值点  #identify(1:n,hatvalues(x),names(hatvalues(x))) #这句产生交互效果，选中某个  点后，关闭后返回点的名称  }  hat.plot(lm.fit\_1) #这里作图停不下来点击图片右上方 finish |

|  |
| --- |
| 偏差方差分解：  均方误差MSE=估计量的而偏差的平方+估计量的方差  偏差：估计值的均值与真值的差异； |
| 与最小二乘相比，lasso模型灵活性更差。当lasso模型的预测结果的偏差增大的大小小于其方差减小的大小时，lasso模型给出的预测值更为准确。  最小二乘估计时无偏估计，均方误差=估计量的方差；  lasso是有偏估计，偏差大于0，均方误差小于最小二乘；  则lasso偏差增大的量小于估计量方差减少的量。 |
| 与最小二乘相比，岭回归模型灵活性更差。当岭回归模型的预测结果的偏差增大的大小小于其方差减小的大小时，岭回归给出的预测值更准确。  岭回归也是牺牲“无偏性”的代价下，减少估计量方差从而减少均方误差。 |
| 与最小二乘相比，非线性方法的灵活性更好。当非线性方法的预测结果的方差增大的大小小于其偏差减小的大小时，非线性方法给出的预测值更精确。  最小二乘只适用于拟合线性关系，非线性方法可拟合多种线性关系，故非线性灵活性更好。  当自变量与因变量关系明显不满足线性关系时，使用非线性方法可以降低误差，而方差随关系式变复杂而增大，偏差会减小，非线性方法更为准确。 |
| 线性模型用于曲线拟合样本，但分类的决策边界一定时直线的。  区分是否为线性模型，主要看一个乘法式子中自变量x前的系数w，只影响一个x，则为线性模型，或者判断决策边界是否是线性的。  自变量x被两个以上的参数影响，则此模型是非线性的。  最简单判别是否线性模型，只需要判别决策边界是否直线，是否用一条直线划分。 |
| lasso算法最初用于计算最小二乘模型，通过构造一个罚函数得到一个较为精炼的模型，压缩一些系数，同时设定一些系数为零。  保留了子集收缩的优点，是一种处理具有复共线性数据的有偏估计。  偏差大于0，但lasso均方误差小于最小二乘，故其偏差增大量小于估计量方差减小量。  lasso回归的特点是在拟合广义线性模型的同时进行变量筛选和复杂度调整，不论目标因变量是连续的，还是二元或多元离散的，都可用lasso回归建模进行预测。 |
| 岭回归也是回归参数的有偏估计，其结果残差平方和变大，但系数检验变好，正则化的加入，岭回归估计向量的均方误差小于最小二乘，故其偏差增大量小于估计量方差减小量。  岭回归在拟合广义线性模型时同样能在一定意义上解决过拟合问题，原过拟合的大或小的参数，会在经验风险最小化的基础上加入二反噬正则化因子，而被约束到正常，甚至很小的值，但不会为0。 |
| 最小二乘是为了使真实值和估计值之间的差距最小，  线性最小二乘是指f是x的线性函数，对于线性最小二乘是要求min|Ax-y|。  非线性是指f无法表示为x的线性关系。  非线性最小二乘问题是无法构造出一个矩阵线性方程组的，没有办法进一步求解析解，  最小二乘只适用于拟合线性关系，  非线性方法可拟合多种非线性关系。  可通过高斯-牛顿法、裂纹伯格-马夸尔特法等转化为线性方程，同样也可用cholesky分解法、QR分解法、SVD分解法求得。  当自变量与因变量关系明显不满足线性关系时，使用给非线性方法可降低误差，方差可能随关系式变复杂而增大，当误差减小值大于方差增大值时，非线性方法更准确。 |
| 最小二乘将所有的特征作为同样重要的程度来求解，没有做出特征选择，存在过拟合的可能。  Lasso模型在进行系数估计时，需要添加惩罚项，增加了系数估计的条件，因此灵活性更差。  最小二乘模型没有偏差，但是没有考虑方差；  Lasso回归的优势在于综合权衡了偏差和方差，当最小二乘出现较大方差时，以偏差的小幅度增加来换取方差的大幅度下降，从而获得比较好的模型效果。  非线性方法放松了线性模型需要的线性假设，因此建模的方法更加灵活。  相比线性模型，非线性方法可以更灵活地进行拟合，可以有效减小偏差，但可能也因此引起方差的升高。 |
| 非线性拟合模型范围比最小二乘方法范围更广，灵活性更强。预测值与真实值无限接近，从而减少偏差，方差增大。方差增幅小于偏差减幅，误差总和减小。 |
| 偏差，描述的是预测值的期望与真实值之间的差距。偏差越大，越偏离真实数据。  方差，描述的是预测值的变化范围，离散程度，是与期望值的距离。方差越大，数据的分布越分散。 |
| lasso模型可以根据lambda进行参数调节进而自由的选择模型，lasso更易于产生解为0的情况，可以筛选变量。  最小二乘的系数是对应残差平方和最小的情况，相较而言，其系数的确定比较单一固定。  岭回归模型可以根据k进行参数的调节来进而自由选择模型。  非线性回归中，模型比较复杂，估计参数并没有太好的方法。  在求解Q方程的最小值时，可采用迭代的方式：，先选择一个迭代的初始点位，  固定一个参数变量的值，以圆形圆形的方式取很多组另外两个参数变量值，进行计算。  将这一套计算中的最小值处，作为下一个迭代的初始点位，进行新一轮的计算。这么重复直至初始点位为附近的最小值处位置。  这种方法有明显的弊端，估计参数的时候找到的最小值，不见得是全局的最小值，  很可能是局部的最小值点。并且非线性回归分析没有成熟而完整的理论对得到的方程进行假设检验。 |

|  |
| --- |
| 假设n=2，p=2，x11=x12，x21=x22，y1+y2=0，x11+x21=0，x12+x22=0： |
| 岭回归模型的回归系数最优化问题：  岭回归优化一般形式：  最小化：  在这种情况下，。因此，优化表示如下：  最小化：  其中 |
| 证明岭回归模型系数估计满足：：  知以及。  上式变成  对和求导数，为0，得到：  ，  由表达式的对称性知， |
| Lasso模型回归系数最优化问题：  与岭回归类似的：  最小化： |
| 证明lasso模型参数估计结果并不唯一，即c中最优化问题有多个解：  使用lasso约束的替代形式：，所绘制的图形是以原点(0,0)为中心的菱形。  如果考虑平凡化约束：，  由题意可简化为：  最小化：2  有：，平行于lasso菱形边缘。    对原lasso优化问题的解决方案是函数的轮廓与lasso菱形相切。  最后，作为和沿着线，这些轮廓在不同的点接触lasso菱形边缘。  所以，整个边缘就是lasso优化问题的潜在解决方案。  类似的，有  因此，lasso问题没有唯一的解决方案，解决方案的一般形式由两个线段给出： |
| 由于，  则有  和必须是同号，可同时为正号，也可以同时为负号。 |
| c）中最优解表现在图形上为一条直线与一个所有顶点在坐标轴上的菱形的交点，从图形上看，不止有一个交点，因此不止有一个最优解。 |

另，lfx-lzh-wxf-

|  |
| --- |
| 分割训练集和测试集  Size=dim(college)[1]/2  Train=sample(1:dim(college)[1],size)  Test=-train  Train\_data=college[train,]  Test\_data=college[test,] |
| gp<- runif(nrow(college))  college.train<-college[gp<=0.8,]  college.test<-college[gp>0.8,] |
| 最小二乘拟合  Lm\_fit=lm(apps~.,data=train\_data)  Lm\_pred=predict(lm\_fit,test\_data)  Mean((test\_data[,’apps’]-lm\_pred)^2) |
| fit1 <- lm(Apps~Private+Accept+Enroll+Top10perc+Top25perc+F.Undergrad+  P.Undergrad+Outstate+Room.Board+Books+Personal+PhD+Terminal+  S.F.Ratio+perc.alumni+Expend+Grad.Rate,data = college.train)  summary(fit1)  使用测试集对该模型进行检验：  model.step=step(fit1,trace=F)  summary(model.step)  sqrt(sum((predict(model.step,college.test)-college.test$Apps)^2)/nrow(college.test)-11) |
| 岭回归拟合（用交叉验证选择参数lambda）  Train\_mat=model.matrix(apps~.,data=train\_data)  Test\_mat=model.matrix(apps~.,data=test\_data)  Grid=10^seq(4,-2,length=100)  Mod.ridge=cv.glmnet(train\_mat,train\_data[,‘apps’],alpha=0,lambda=grid,thresh=1e-12)  Lambda\_best=mod.ridge$lambda.min  Ridge.pred=predict(mod.ridge,newx=test\_mat,s=lambda\_best)  Mean((test\_data[,‘apps’]-ridge.pred)^2) |
| library(MASS)  ridgelm=lm.ridge(Apps~Private+Accept+Enroll+Top10perc+Top25perc+F.Undergrad+  P.Undergrad+Outstate+Room.Board+Books+Personal+PhD+Terminal+  S.F.Ratio+perc.alumni+Expend+Grad.Rate,data = college.train)  ridgelm$coef  #画出岭迹图  plot(lm.ridge(Apps~Private+Accept+Enroll+Top10perc+Top25perc+F.Undergrad+  P.Undergrad+Outstate+Room.Board+Books+Personal+PhD+Terminal+  S.F.Ratio+perc.alumni+Expend+Grad.Rate,data = college.train,lambda=seq(0,10,1)))  select(lm.ridge(Apps~Private+Accept+Enroll+Top10perc+Top25perc+F.Undergrad+  P.Undergrad+Outstate+Room.Board+Books+Personal+PhD+Terminal+  S.F.Ratio+perc.alumni+Expend+Grad.Rate,data = college.train,lambda=seq(0,10,1)))  根据广义交叉验证法，岭回归参数应该选择1 .  代码：使用测试集验证  x<-as.matrix(college.train)[,2]  y<-as.matrix(college.train)[,-2]  x.test<-as.matrix(college.train)[,2]  y.test<-as.matrix(college.train)[,-2]  cv\_fit <- cv.glmnet(x, y, alpha =0,lambda = lambdas)  cv.glmnet()  plot(cv\_fit)  opt\_lambda <- cv\_fit$lambda.minopt\_lambda  y\_predicted <- predict(v\_fit, s = opt\_lambda, newx = x.test)  sse <- sum((y\_predicted - y.test)^2) |
| Lasso模型拟合（用交叉验证选择参数lambda）  Mod.lasso=cv.glmnet(train\_mat,train\_data[,’apps’],alpha=1,lambda=grid,thresh=1e-12)  Lambda\_best=mod.lasso$lambda.min  Lasso.pred=predict(mod.lasso,newx=test\_mat,s=lambda.best)  Mean((test\_data,[‘apps’]-lasso\_pred)^2) |
| Mod.lasso=glmnet(model.matrix(apps~.data=college),college[,’apps’],alpha=1)  Predict(mod.lasso,s=lambda\_best,type=’coefficients’) |
| library(glmnet)  model.lasso=lars(x,y)  plot(model.lasso)  summary(model.lasso)#给出拟合结果  cv.model.lasso=cv.lars(x2,y,k=10)#十折交叉验证  select=cv.model.lasso$index[which.min.lasso$cv]  coef=coef.lars(model.lasso,mode=”fraction”,s=select)  coef[which(coef!=0)]  n.cp=which.min(model.lasso$cp)  coef1=coef.lars(model.lasso,mode=”step”,s=n.cp)  coef1[which(coef!=0)] |

另：(python)zxh-exl-jyj-zxy；（R）wxf

|  |
| --- |
| Logistic回归模型：  设想有一个二分变量 Y，Y=1 和 Y=0 分别表示一件事情的两种可能性；  Y 的均值用μ表示，μ值就是 Y=1所表示事件发生的比例，也就是支持该决议的概率P；发生比 odds = P/（1-P），其含义是支持该决议的概率对反对该决议的倍数；  logistic 回归的结果一般用发生比 odds 来解释；  Logistic 模型一般形式：  描述了 Y 的对数发生比随 X 的取值变化而变化. |

|  |
| --- |
| 比较LDA和QDA:  线性判别函数LDA（方差齐性），  基于二者具有相同的协差阵假设下，该假设下的贝叶斯决策论边界是线性的。  二次判别函数QDA（方差非齐性），  协差阵不同时，需要在各个类别中对协差阵进行估计，此时的贝叶斯决策边界不是一个超平面，而是一个超二次曲面，特别的，在二维情况下，他是一条抛物线（LDA下为一条直线）  决策边界为非线性时，QDA通常会表现更好；  在训练观察次数较少时，需要减少方差时，LDA表现好；  当训练集较大，差异较小时，QDA表现好；  LDA和QDA在很多数据的分类效果上表现非常好，并不是因为数据是近似正态分布的，也不是因为LDA假定的各类方差相等。更可能的原因是，数据支持各类别边界是线性或是二次函数，LDA和QDA的方法比较稳健；  LDA模型形式简洁，偏差略大，但方差大大降低，是的模型估计准确率总体较高；  QDA的参数个数过多，往往方差较大。 |
| 当贝叶斯决策边界是线性时，  训练集上QDA的效果好：二次判别，需要估计的参数更多，拟合效果更好；  测试集上LDA效果好：LDA线性判别结果更接近真实值（方差小）。  从训练集说，LDA是QDA的一部分，小范围取最优；  从测试集说，QDA可能过拟合，LDA方差更小，更接近真实值。  对着模型复杂度的提高，训练误差逐渐减小，测试误差逐渐增大。  决策边界线性时，QDA没有考虑方差齐性，模型复杂度高，在训练集上效果好；测试集上LDA效果好。 |
| 当贝叶斯决策边界非线性时，QDA在训练集和测试集的效果都比LDA好。  决策边界非线性，LDA没有考虑异方差，模型复杂度低，训练集上QDA效果好，测试集也是QDA效果好。 |
| 当样本量n增大时，相比于LDA的测试预测率，QDA的预测率会更好。  样本量n的提升，自由度更高的模型会产生更好的效果，方差会被大样本所抵消。 |
| 当样本量少时，QDA会过拟合。  训练集上QDA优于LDA，有更大的范围，可找一个更好的边界，  训练集错误率来说，QDA总是更小的，QDA可以表示一个线性决策边界，但在测试集来说，该数据不在模型训练数据中，LDA的方差会更小，更接近真实值，因而对测试错误率来说，LDA会更小。  为降低测试错误率，应该选择LDA。 |
| 选择LDA还是QDA是一个偏差-方差权衡问题。当有p个预测变量时余册协方差矩阵需要p(p+1)/2个参数，QDA需要对每一类分别估计协方差矩阵，需要Kp(p+1)/2个参数。  若有50个观测变量，需要1225个参数。  假设K类的协方差矩阵相同，LDA模型对x来说是线性的，意味着有Kp个线性系数需要估计。  所以LDA模型没有QDA分类器光滑，拥有更低的方差，有改善预测效果的潜力。  同时也要权衡考虑，LDA假设K类有相同的方差，是一个非常糟糕的假设，LDA会产生很大的偏差。  一般而言，训练观测数据量相对较少，LDA是一个比QDA更好的决策，降低模型的方差很有必要。  而训练集非常大，则更倾向于使用QDA，分类器的方差不再是一个核心问题，K类的协方差矩阵相同的假设站不住脚。 |

|  |
| --- |
| 有特征x，一个2类响应变量，类大小为N1、N2，编码为-N/N1、N/N2，K=2： |
| 证明LDA规则分类到类1、2的条件：  a)线性判别函数LDA（方差齐性），  则有：,  相减：  根据LDA：分类给最大的类，所以：  当时，将样本分类给类2；  反之则给类1。 |
| 考虑最小二乘方差准则极小化：  记  则  得  代入，得  取 =，  取 ，=N  又  而  则  其中， |
| 偏导得：  &  前式代入后式：  有  进而得到 |
| 证明β在方向上，最小二乘回归系数等于LDA系数，相差一个因子：  由b知，  有，，所以在方向上， |
| 由于为px1维向量，为1xp维向量，而β为px1维向量，故为一常数：  有 =，所以在方向上。  公式：，可化简为：  当，有，为一常数，  则 ，可得  即，最小二乘回归系数等于LDA系数，相差一个因子 |
| 证明对任意不同类编码均成立：  由b知，  记任意编码为，有  则，=  ==  又知  有，  故，，在方向上 |
| 不同的编码之间的区别只是类别的距离大小有区别，对于二分类问题，只有两个类别，类别之间的距离大小对分类没有影响，因此对任意不同的编码，上述结论均成立。 |
| 如果y>0，则类2，否则类1。证明这与LDA规则不同，除非两个类具有相同的观测个数：  由b知，  预测值：  由，得  只有时，可化简为：  只有此时与LDA的判别规则相同。 |

另：yy

|  |
| --- |
| 对weekly数据进行数值和图像描述统计  data(Weekly)  glimpse(Weekly)  Summary(weekly)  Pairs(weekly)  Cor(weekly[,-9]) |
| 每周交易量随时间增长关系：总体交易量上升趋势，但在 2009 年之后出现下降    每年周增长降低所占比例：只有 2000、2001、2002、2008 这四年全年周 down 概率大 于 up 概率；    随时间变化的周波动：基本呈现随机波动状态，但是在 2000-2003 年附近，2008 年附近 波动较大，从上图也已经得知在这些年份周 down比例超过了up比例： |
| library(ISLR)  library(broom)  library(tidyverse)  library(ggplot2)  library(MASS)  library(class)  library(caret)  library(e1071) |
| 整个数据及建立逻辑回归，将5个滞后时间变量加上volume作为预测变量，cirection作为响应变量。  Attach(weekly)  Glm.fit=glm(direction~.,data=weekly,family=binomial)  Summary(glm.fit)  Tidy(glm.fit)#将模型输出结果转化为数据框 |
| 计算混淆矩阵和整体预测准确率，对混淆矩阵中关于逻辑回归所犯错误类型进行解释:  Glm.probs=predict(glm.fit,type=’response’)  Glm.pred=rep(‘DOWN’,’length(glm.probs))  Glm.pred[glm.probs>0.5]=’Up’  Table(glm.pred,direction) |
| 用caret包中的confusionMatrix计算混淆矩阵：  Attach(weekly)  Pred=factor(ifelse(predict(glm.fit,type=’reponse’)<.5,’Down’,’Up’))  confusionMatrix(pred,direction,positive=’Up’)  prop.table(table(pred)) |
| 根据1990-2008这段时间训练数据拟合逻辑斯蒂回归，计算混淆矩阵和测试集中总体准确率  Train=(year<2009)  Weekly.0910=weekly[!train,]  Glm.fit=glm(direction~lag2,data=weekly,family=binomial,subset=train  Glm.probs=predict(glm.fit,weekly.0910,type=;response’)  Glm.pred=rep(‘Down’,length(glm.probs))  Glm.pred[glm.probs>0.5]=’Up’  Direction.0910=direction[!train]  Table(glm.pred,direction.0910)  Mean(glm.pred==direction.0910) |
| 用LDA：  Lda.fit=lda(direction~lag2,data=weekly,subset=train)  Lda.pred=predict(lda.fit,weekly.0910)  Table(lda.pred$class,direction.0910)  Mean(lda.pred$class==direction.0910) |
| 用QDA：  Qda.fit=qda(direction~lag2,data=weekly,subset=train)  Qda.class=predict(qda.fit,weekly.0910) $class  Table(qda.class,direction.0910)  Mean(qda.class==direction.0910) |
| 用k-1的knn：  Train.x=as.matrix(lag2[train])  Test.x=as.matrix(lag2[!train])  Train.drection=direction[train]  Set.seed(1)  Knn.pred=knn(train.x,test.x,train.direction,k=1)  Table(knn.pred,direction.0910)  Mean(knn.pred==direction.0910) |

另：（python）lkm-yb

|  |
| --- |
| 在平方误差情形下，样本内预测误差和训练误差：  在每个表达式和展开式中增加和减去和，建立的训练误差中的乐观性为： |
| 对展开：  则有:  对同理：  则：  由（预测误差），有；  由，有，同理；  则  由，且与独立，  有  对，（样本与期望之差的和）  由，有  故  由，有，即证。 |

|  |
| --- |
| **Prostate.data数据**  执行最佳子集线性回归分析，计算预测误差的AIC\BIC\5折交叉验证和10折交叉验证，632估计： |
| 当进行回归分析时，通常获取到的自变量并不全是有用的，存在和因变量不相关或者相关性极小的变量。针对这种情况，一般可以认为的根据经验判断筛选对因变量有影响的自变量。  但通常进行回归分析时，我们并不是领域专家，对可能影响因变量的自变量并不了解。于是需要运用算法获得最贴近真实模型的回归模型，比如最优子集回归。  最优子集回归：对p个预测变量的所有可能组合分别使用最小二乘回归进行拟合。  对含一个变量的模型，拟合p个模型；对含两个变量的模型，拟合p(p-1)/2个模型，以此类推，总共拟合2^p个模型。  并按照一定的比较准则从中选择一个最优模型。（如AIC,最小均方误差等） |
| 最佳子集线性回归best subset regression：  #训练集  Train=subset(data,train==TRUE)[,1:9]  Str(train)  #测试集  Test=subset(data,train==FALSE)[,1:9]  Str(test)  #输入数据矩阵化，创建响应变量  X=as.matrix(train[,1:8])  Y=train[,9]  #最优子集线性回归  Library(leaps)  Bestsub=regsubsets(lpsa~.,data=train)  Bestsub.sum=summary(bestsub)  Names(bestsub.sum)  Which.min(bestsub.sum$rss#结果为8，有8个特征的模型具有最小rss  #rss,残差平方和  Plot(bestsub,scale=’adjr2’)#调整r方=1-(1-r2)(n-1)/(n-k-1)，大  Plot(bestsub,scale=’r2’)#r方=(tss-rss)/tss=1-rss/tss，大  Plot(bestsub,scale=’cp’)#cp统计量=rss/n+，小  Plot(bestsub,scale=’bic’)#bic=kln(n)-2ln(L)，小  #模型拟合  Lm.fit=lm(lpsa~.,data=test)  Summary(lm.fit)  #AIC & BIC  AIC(lm.fit)  BIC(lm.fit) |
| Or？  Bestsub.aic=step(lm.fit,trace=F)  Summary(bestsub.aic)  Aic.pred=predict(bestsub.aic,test,se.fit=TRUE)  Aic.pred$residual.scale  N=nrow(train)  Bs.bic=step(lm.fit,k=log(n),trace=0)  Summary(bs.bic)  Bic.pred=predict(bs.bic,test,se.fit=TRUE)  Bic.pred$residual.scale  Or? AIC逐步回归  Library(MASS)  Fit=lm(lpsa~.,data=data)  stepAIC(fit,direction=’backward’)  or？  AIC(fit)  AIC(fit,k=log(length(data[,1])))#BIC |
| K折交叉验证：k-fold-cross-validation，首先将所有数据分割成k个子样本，不重复的选取其中一个子样本作为测试集，其他k-1个样本用来训练。共重复k次，平均k次的结果或者使用其他指标，最终得到一个单一估测。  该方法的优势在于，保证每个子样本都参与训练且都被测试，降低泛化误差。  其中10折交叉验证是最常用的。 |
| #5折交叉验证  K=5  Folds=sample(1:k,nrow(data),replace=T)  cv.error5=0  glm.fit=glm(y~x,data=train)  #delta有两个结果：预测误差的原始交叉验证估计，调整交叉验证估计  #调整旨在补偿不使用留一交叉验证引入的偏差  Cv.error5=cv.glm(train,glm.git,K=5)$delta[1]  cv.error5  #10折交叉验证  Cv.error10=0  Glm.fit=glm(y~x,data=train)  cv.error10=cv.glm(train,glm.fit,K=10)$delta[1]  cv.error10 |
| #5折交叉验证  CV=function(n,z=10,seed=666){  Z=rep(1:z,ceiling(n/z))[1:n]  Set.seed(seed)  Z=sample(Z,n)  Mm=list()  #mm[[i]]是第i个下标集  For(i in 1:z) mm[[i]]=(1:n)[Z==i];return(mm)  }  N=nrow(data);z=5;mm=CV(n,z);D=1  MSE=rep(0,z)#创建一个向量存出结果  For(I in 1:z){ #循环5次  M=mm[[i]];  M=mean((data[m,D]-mean(data[m,D]))^2)  F=lm(lpsa~.,data[-m,])#简单线性回归  MSE[i]=mean((data[m,D]-predict(f,data[m,]))^2)/M#求测试集NMSE  }  Mean(MSE)  10折交叉验证  n=nrow(data);Z=10;mm=CV(n,Z);D=1  MSE=rep(0,Z) #建立一个向量储存结果  for(i in 1:Z){ #循环10次  m=mm[[i]];  M=mean((data[m,D]-meandat(a[m,D]))^2)  f=lm(lpsa~.,data[-m,]) #简单线性回归,[-m]为训练集下标集合  MSE[i]=mean((data[m,D]-predict(f,data[m,]))^2)/M #求测试集NMSE  }  mean(MSE) |
| Library(caret)  #5 fold  Set.seed(123)  Train.control=trainControl(method=’repeatedcv’,number=5,repeats=3)  Model=train(lpsa~.,data=data,method=’lm’,trControl=train.control)  Print(model)  #10 fold  Set.seed(123)  Train.control=trainControl(method=’repeatedcv’,number=10,repeats=3)  Model=train(lpsa~.,data=data,method=’lm’,trControl=train.control)  Print(model) |
| 通过你放交叉验证，可以得到更好的自助法估计，对于每个观测，仅仅跟踪自助法样本中不包含该观测的预测，这个舍一法预测误差的自助法估计由下式定义：  其中，是不包含观测i的自助法样本b的指标集，是样本的个数，在计算时，要么必须选择充分大的B来保证所有的都大于0，或者可以直接删掉的项。  舍一自助法解决了遭受的过拟合问题，但是会有交叉验证中的训练集大小偏差问题（training-set-size），每个自助法样本中不同观测的平均个数大约为0.632\*N，所以它的偏差会大致表现得和两折交叉验证一样，因此学习曲线在样本大小为N/2时会有相当大的斜率，则舍一自助法作为真实误差的估计会有一个向上的偏差upward bias  .632估计是为了减轻偏差而设计的，由下式定义： |
| #自助法  Library(boot)  Betas=function(formula,data,indices){  D=data[indices,]  Fit=lm(formula,data=d)  Return(fit$coef)  }  Results=boot(data=data,statistic=betas,R=1000,formula=lpsa~.)  Results  Results2=boot(data=data,statistic=betas,R=1000,formula=lpsa~lcavol+lweight)  Results2 |
| #boot  Set.seed(123)  Train.control=trainControl(method’boot’,number=100)  Model=train(lpsa~.,data=data,method=’lm’,trControl=train.control)  Print(model) |

另，（R）zxy-lzh-smx；（python）fc-

|  |
| --- |
| 自助法：从训练集中风概率、有放回的重新抽取样本，得到bootstrap数据集Z。  设自助法抽样的样本量是n，则等概率是指每次抽样时，每个样本被抽中的概率是1/n。有放回是指每个样本被抽中之后，并不从初始的训练样本中移除，下一次抽样仍有可能被抽中。  所以在bootstrap样本中，原始训练集中的某个样本点可能多次出现，也可能一次都不出现。 |
| 证明：第一个自助法观测不是原始样本中第j个观测的概率。  （反证）第一个自助法观测是第j个观测的概率是1/n，则不是第j个观测的概率是1-1/n。 |
| 证明第j个观测不在自助法样本里的概率为（1-1/n）^n。  自助法是有放回抽取，每次抽取观测不是第j个观测的概率都是1-1/n，所以自助法样本中不包含第j个观测的概率是（1-1/n）^n。 |
| 第j个观测在自助法样本集中的概率是1-（1-1/n）^n=63.2% |
| 绘制第j个观测在自助法样本的概率结果：  Pr=function(n) return(1-(1-1/n)^n)  X=1:xe+05  Plot(x,pr(x))  Min(y)  Mean(y) |
| 自助法样本量100，j=4，反复产生自助法样本，每次把第4个观测是否包含在自助法样本中记录：  Store=rep(NA,10000)  For(I in 1:10000){  Store[i]=sum(sample(1:100,rep=TRUE)==4)>0  }  Mean(store) |

|  |
| --- |
| 交叉验证法：  可以用测试误差来衡量模型的推广预测能力，但是一般情况下，不能事先得到一个测试观测集。  幸运的是，如今有很多方法可以通过对可获得的训练集数据来估计测试误差。  在拟合过程中，保留训练观测的一个子集，在其余的观测上拟合模型，进而将拟合的模型用于所保留的观测子集上进行预测，从而得到其他预测误差的估计。  cv.glm函数，默认留一交叉验证。Cv.glm$delta给出的是交叉验证的结果，包含两个数据，一个是标准loocv估计，第二个是偏差矫正后的结果。对于loocv，同一个训练集，不管训练多少次，结果都将是一样的。  R中boot包的cv.glm函数可以实现交叉验证。对于任意一个广义线性模型，都可用glm函数拟合对应的模型。设置family=binomial即可拟合logistic模型；使用glm函数时没有制定family参数的值，即默认family=gaussian，与lm函数一样执行线性回归。常使用glm而不是lm函数来处理线性回归问题。 |
| 生成一个模拟数据：n=100，p=2,y=x-2\*x^2+e，e是随机误差项，服从正态分布。  Set.seed(1)  Y=rnorm(100)  X=rnorm(100)  Y=x-2\*x^2+rnorm(100) |
| 作x对y的散点图：  Plot(x,y) |
| 设置一个随机种子数，计算用最小二乘法拟合四个模型产生的loocv误差：  Librart(boot)  Data=data.frame(x,y)  Set.seed(1)  Glm.fit=glm(y~x)  cv.glm(data,glm.fit)$delta  glm.fit(glm(y~poly(x,2))  cv.glm(data,glm.fit)$delta  glm.fit=glm(y~poly(x,3))  cv.glm(data,glm.fit)$delta  glm.fit=glm(y~poly(x,4))  cv.glm(data,glm.fit)$delta |
| 换另一个种子重复，结果一样吗？  一样，因为交叉验证对原数据遍历，固定数据集没有随机性。 |
| 讨论用最小二乘拟合每个模型得到的系数估计的统计意义。结果与交叉验证法得到的结论一致吗？  Lm.fit=lm(y~x)  Coef(lm.fit)  Nls.fit=nls(y~b0+b1\*x+b2\*b^2)  Coef(nls.fit)  Nls.fit=nls(y~b0+b1\*x+b2\*x^2+b3\*x^3)  Coef(nls.fit)  Nls.fit=nls(y~b0+b1\*x+b2\*x^2+b3\*x^3+b4\*x^4)  Coef(nls.fit)  结果与cv相符。 |

|  |
| --- |
| Boston 数据集：  Library(MASS)  Summary(Boston) |
| 给出一个对medv房价中位数的总体均值的估计u：  Sea.seed(1)  Attach(Boston)  Medv.mean=mean(medv) |
| 给出一个对u的标准误差的估计：样本的标准差除以观测数的平方根计算样本均值的标准误差。  Medv.err=sd(medv)/sqrt(length(medv)) |
| 自助法估计u的标准误差：  Boot.fn=function(data,index)return (mean(data[index]))  Library(boot)  Bstrap=boot(medv,boot.fn,1000) |
| 95%置信区间：与t.test（boston$medv）的结果对比  t.test(medv)  c(bstrap$t0-2\*0.4107,bstrap$t0+2\*0.4107) |
| 给出medv总体中位数的估计umed：  Medv.med=median(medv) |
| 估计umed的标准误差，没有一个简单的公式计算，可以用自助法估计中位数的标准误差：  Boot.fn=function(data,index) return(median(data[index]))  Boot(medv,boot.fn,1000) |
| 给出medv的10%分位数的估计，记u0.1：可以使用quantile函数：  Medv.tenth=quantitle(medv,c(0.1)) |
| 用自助法估计u0.1的标准误差：  Boot.fn=function(data,index) return(quantile(data[index],c(0.1)))  Boot(medv,boot.fn,1000)  实际意义：用来表示样本十分位数与总体十分位数的离散程度，标准误越小，二者差距越小，反之越大。标准误用于预测样本数据准确性，标准误越小差距越小，样本数据越能代表总体数据。 |
| Boot(data,statistic,R)，其中data是数据框或矩阵，statistic是作用在data上的一个函数，R是重复次数，bootstrap replicates  在R中进行自助法是利用boot扩展包。  线变下一个求取统计量的自定函数  将自定义函数放入boot()中，得到自助法结果。  用boot.ci()函数求取置信区间  boot.ci(results,conf=0.95,type=c('perc','bca')) 其中 results 为 boot()函数，conf 表示置信水平，type 表示了用何种算法来求区间，perc 即使用百分位方法，bca 表示 adjusted bootstrap percentile，即对偏差进行了调整。 |

|  |
| --- |
| 信息增益比，C4.5算法，生成决策树：  信息增益比，其中  记为年龄、工作、房子、信贷4个特征：  特征熵：  经验熵：  信息增益：  同理：  信息增益比：  同理：  选信息增益大的——选择作为根节点特征；  对是否有房的节点继续分：  经验熵：  信息增益比：  选择作为节点特征。  有房  工作  是  否  是 |

另：（python）wy

|  |
| --- |
| 用平方误差损失准则生成一个二叉回归树：（最小二乘回归树生成算法）——python |
| 最小二乘回归树生成算法：  在训练数据集所在的输入空间中，递归地将每个区域划分为两个子区域并决定每个子区域上的输出值，构建二叉决策树：   1. 选择最优切分变量j与切分点s，求解：   遍历变量j，对固定的切分变量j扫描切分点s，选择使上式达到最小值的对（j,s）   1. 用选定的对（j,s）划分区域并决定相应的输出值：   ,  ,,   1. 继续对两个子区域调用步骤①②，直至满足停止条件 2. 将输入空间划分为M个区域，生成决策树： |
| Import numpy as np  Import matplotlib.pyplot as plt  #节点定义  Class TreeNode(object):  Def \_\_init\_\_(self,tempr,tempc):  #用r存储满足节点划分条件的实例对应的yi  #将输入空间划分为若干单元，对所属每个单元的所有实例xi对应的yi去平均，即为该单元上最优输出值：  Self.r=tempr  Self.c=tempc  Self.left=None  Self.right=None  #CART算法建立回归树：  Def CART(start,end):  If (end-strat>=1):  Result=[]  For s in range(start+1,end+1):  Y1=y[start:s]  Y2=y[s:end+1]  Result.append((y1.std()\*\*2)\*y1.size+(y2.std()\*\*2)\*y2.size)  Index1=result.index(min(result))+start  Root=TreeNode(y[start:end+1],min(result))  Print(‘node:’,y[start:end+1],’s=’,index+1,’最小平方误差:’,min(result))  Root.left=CART(start,index1)  Root.right=CART(index1+1,end)  Else:  Root=None  Return root  Y=np.array([4.50,4.75,4.91,5.34,5.80,7.05,7.90,8.23,8.70,9.00])  Root=CART(0,9)  节点： [4.5 4.75 4.91 5.34 5.8 7.05 7.9 8.23 8.7 9. ] s= 5 最小平方误差： 3.3587199999999986  节点： [4.5 4.75 4.91 5.34 5.8 ] s= 3 最小平方误差： 0.1912  节点： [4.5 4.75 4.91] s= 1 最小平方误差： 0.012800000000000023  节点： [4.75 4.91] s= 2 最小平方误差： 0.0  节点： [5.34 5.8 ] s= 4 最小平方误差： 0.0  节点： [7.05 7.9 8.23 8.7 9. ] s= 7 最小平方误差： 0.6625166666666665  节点： [7.05 7.9 ] s= 6 最小平方误差： 0.0  节点： [8.23 8.7 9. ] s= 8 最小平方误差： 0.04500000000000021  节点： [8.7 9. ] s= 9 最小平方误差： 0.0  [4.5 4.75 4.91 5.34 5.8 7.05 7.9 8.23 8.7 9. ]  [4.5 4.75 4.91 5.34 5.8]  [7.05 7.9 8.23 8.7 9. ]  [4.5 4.75 4.91]  [7.05 7.9]  [5.34 5.8 ]  [8.23 8.7 9. ]  [4.91]  [4.5]  [5.34]  [5.8 ]  [7.9]  [7.05]  [8.23]  [8.7 9. ]  [4.75]  [4.75 4.91]  [8.7]  [9. ] |
|  |

|  |
| --- |
| 证明CART剪枝算法中，当α确定，存在唯一的最小子树使损失函数最小： |
| 反证法：  假设α确定，有两个子树的损失函数都最小：  可能出现如下情况：   1. ： ：   虽然和有相同的最小损失函数，但更小，所以不是最小子树。   1. ： ：   将任一枝剪枝均可得到更小的损失函数值，故和都不是最小子树。 |
| 内部节点是否剪枝只与以该节点为根节点的子树有关。  剪枝过程：  计算子树的损失函数：  其中，，是叶节点个数，K是类别个数。  有剪枝前子树，剪枝后子树，满足，则进行剪枝  反证法：  假设α确定，有两个子树的损失函数都最小：   1. 被剪枝的子树在同一边，可知其中一个子树会由另一个子树剪枝得到，故二者不能同时为最优子树； 2. 被剪枝的子树不在同一边，可知被剪枝的子树都可使得损失函数最小，两颗子树均可继续剪枝，故二者均不是最优子树；   综上，不存在两个最优子树。 |
| **存在性：**  在特征有限、数据集2的情况下，由于能生成的决策树数量是有限的，每一颗子树都对应着某一个损失函数值，故一定存在一个最小的损失函数。  **唯一性：**（上述反证法内容） |

|  |
| --- |
| Carseats数据集：，将sales变量转化为定性响应变量，建立分类树，把响应变量视作定量变量： |
| 划分训练集和测试集：  Train=sample(dim(carseats)[1],dim(carseats)[1]/2)  Carseats.train=carseats[train,]  Carseat.test=carseats[-train,] |
| 对训练集建立回归树，测试错误率是多少？  Traa.carseats=tree(Sales~.,data=carseat.train)  Summary(tree.carseats)  Pred.carseats=predict(tree.carseats,carseats.test)  Mean((carseats.test$sales-pred.carseats)^2) |
| 用交叉验证确定最优的树复杂性，剪枝会减小测试错误率吗？  cv.carseats=cv.tree(tree.carseats,FUN=prune.tree)  par(mfow=c(1,2))  plot(cv.carseats$size,cv.carseats$dev,type=’b’)  plot(cv.carseats$k,cv.carseats$dev,type=’b’)  #得到bestsize=9  Pruned.carseats=prune.tree(tree.carseats,best=9)  Par(mfow=c(1,1))  Plot(pruned.carseats)  Text(pruned.carseats,pretty=0)  Pred.pruned=predict(pruned.carseats,carseats.test)  Mean((carseats.test$sales-pred.pruned)^2)  #剪枝会增大测试误差率 |
| 用装袋法分析数据，得到的测试错误率是多少？用函数importance()确定哪些变量最重要：  Bag.carseats=randomForest(Sales~.,data=carseats.train,mtry=10,ntree=500,importance=T)  Bag.pred=predict(bag.carseats,carseats.test)  Mean((carseat.test$sales-vag.pred)^2)  #装袋法测试错误率降低  因为前者基于当一个给定的变量被排除在模型之外时，所以预测袋外样本的准确性的平均减小值。而后者衡量的是由此变量导致的分裂使结点不纯度减小的总量。  Importance(bag.carseats)#看第二列数据大的 |
| library(tree)  library(ISLR)  attach(Carseats)  **high=ifelse(Sales<=8,"no","yes")**  **Carseats=data.frame(Carseats,high)**  tree.carseats=tree(high~.-Sales,Carseats)  summary(tree.carseats)  plot(tree.carseats)  text(tree.carseats,pretty=0)  #分为训练集和测试集  set.seed(2)  train=sample(1:nrow(Carseats),200)  Carseats.test=Carseats[-train,]  high.test=high[-train]  tree.carseats=tree(high~.-Sales,Carseats,subset=train)  tree.pred=predict(tree.carseats,Carseats.test,type="class")  table(tree.pred,high.test)  (table(tree.pred,high.test)[1,1]+table(tree.pred,high.test)[2,2])/200#测试误差  #剪枝，交叉验证  set.seed(3)  cv.carseats=cv.tree(tree.carseats,FUN=prune.misclass)  names(cv.carseats)  cv.carseats  par(mfrow=c(1,2))  plot(cv.carseats$size,cv.carseats$dev,type="b")  plot(cv.carseats$k,cv.carseats$dev,type="b")  prune.carseats=prune.misclass(tree.carseats,best=9)  plot(prune.carseats)  text(prune.carseats,pretty=0)  #剪枝后预测  tree.pred=predict(prune.carseats,Carseats.test,type="class")  table(tree.pred,high.test)  (table(tree.pred,high.test)[1,1]+table(tree.pred,high.test)[2,2])/200#测试误差  #回归树  library(MASS)  set.seed(1)  train=sample(1:nrow(Boston),nrow(Boston)/2)  tree.boston=tree(medv~.,Boston,subset=train)  summary(tree.boston)  plot(tree.boston)  text(tree.boston,pretty=0)  cv.boston=cv.tree(tree.boston)  plot(cv.boston$size,cv.boston$dev,type="b")  #剪枝  prune.boston=prune.tree(tree.boston,best=5)  plot(prune.boston)  text(prune.boston,pretty=0)  #预测  yhat=predict(tree.boston,newdata=Boston[-train,])  boston.test=Boston[-train,"medv"]  plot(yhat,boston.test)  abline(0,1)  mean((yhat-boston.test)^2) |

|  |
| --- |
| 公司职员考查身体、业务能力、发展潜力，弱分类器为决策树桩。用AdaBoost算法学习一个强分类器：——python |
| From sklearn.ensemble import AdaBoostClassifier  Import numpy as np  from sklearn import metrics  X=np.array([[0,1,3],[0,3,1],[1,2,2],[1,1,3],[1,2,3],[0,1,2],[1,1,2],[1,1,1],[1,3,1],[0,2,1]])  Y=np.array([-1,-1,-1,-1,-1,-1,1,1,-1,-1])  Clf=AdaBoostClassifier()  Clf.fit(x,y)  Y\_pred=clf.predict(x)  score=clf.score(x,y)#预测准确率  report=metrics.classification\_report(y,y\_pred)#包含准确率，召回率等信息  confuse=metrics.confusion\_matrix(y,y\_pred)#混淆矩阵  print(‘raw classes:’,y)  print(‘predict classes:’,y\_pred)  print(‘correct predict prob:{:.2%}’.format(score)) |

另：（python）-fc-长

|  |
| --- |
| 如果每个基预测器方差为，相关系数，证明：Bagging的方差为 |
| Bagging基本思想是将每个基预测器平均来降低方差，若每个基预测器独立，则有；  但bagging抽样是bootstrap有放回抽样，数据集之间有重复值，故不独立。 |
| 对于bagging来说，每个基模型的权重等于，且期望近似相等，则有：  = |
|  |

|  |
| --- |
| **假设第m个分类器误差为εm=0.5-γm, 证明AdaBoost分类器的训练误差上界为** |
| 由  有  由，有 |
| AdaBoost算法:  输入：训练数据集，其中，  输出：最终分类器   1. 初始化训练数据的权值分布 2. 对: 3. 使用具有权值分布Dm的训练数据集学习，得到基本分类器 4. 计算在训练数据集上的分类误差率： 5. 计算的系数， 6. 更新训练数据集的权值分布：;;;   （规范化因子，使得成为一个概率分布。）   1. 构建基本分类器的线性组合：   最终分类器： |
| 将代入，等式成立  对不等式：  由泰勒公式在x=0处  同理：  连乘 |

|  |
| --- |
| Carseats数据集：  实验中，将sales转化为定性响应变量后建立分类树。  现在将响应变量视作定量变量，用回归树预测sales。 |
| 用随机森林分析数据： |
| Rf.carseats=randomForest(sales~.,dara=carseats.train,mtry=5,ntree=500,importance=T)  Rf.pred=predict(rf.carseats,carseats.test)  Mean((carseats.test$sales-re.pred)^2)#测试错误率  Importance(rf.carseats) |
| 每个分裂点处考虑的变量个数m对测试错误率的影响： |
| rf.carseats=randomForest(Sales~.,data=carseats.train,mtry=1,ntree=500,importance=T)  > rf.pred=predict(rf.carseats,carseats.test)  > mean((carseats.test$Sales-rf.pred)^2)  [1] 4.649253  > rf.carseats=randomForest(Sales~.,data=carseats.train,mtry=2,ntree=500,importance=T)  > rf.pred=predict(rf.carseats,carseats.test)  > mean((carseats.test$Sales-rf.pred)^2)  [1] 3.165822  > rf.carseats=randomForest(Sales~.,data=carseats.train,mtry=3,ntree=500,importance=T)  > rf.pred=predict(rf.carseats,carseats.test)  > mean((carseats.test$Sales-rf.pred)^2)  [1] 2.790597  > rf.carseats=randomForest(Sales~.,data=carseats.train,mtry=4,ntree=500,importance=T)  > rf.pred=predict(rf.carseats,carseats.test)  > mean((carseats.test$Sales-rf.pred)^2)  [1] 2.446639  > rf.carseats=randomForest(Sales~.,data=carseats.train,mtry=5,ntree=500,importance=T)  > rf.pred=predict(rf.carseats,carseats.test)  > mean((carseats.test$Sales-rf.pred)^2)  [1] 2.294829  > rf.carseats=randomForest(Sales~.,data=carseats.train,mtry=6,ntree=500,importance=T)  > rf.pred=predict(rf.carseats,carseats.test)  > mean((carseats.test$Sales-rf.pred)^2)  [1] 2.239431  > rf.carseats=randomForest(Sales~.,data=carseats.train,mtry=7,ntree=500,importance=T)  > rf.pred=predict(rf.carseats,carseats.test)  > mean((carseats.test$Sales-rf.pred)^2)  [1] 2.182903  > rf.carseats=randomForest(Sales~.,data=carseats.train,mtry=8,ntree=500,importance=T)  > rf.pred=predict(rf.carseats,carseats.test)  > mean((carseats.test$Sales-rf.pred)^2)  [1] 2.13373 |

|  |
| --- |
| Credit数据集：构建自动化的信用评分模型，1000个贷款信息，每一个贷款有14个自变量和一个变量记录该笔贷款是否违约。 |
| 数据探索和预处理：  Str(credit)  #分类变量因子化  F=c(1,4,5,6,8,9,11,12)  For (I in F) credit[,i]=as.factor(credit[,i])  Str(credit)  #划分1:1的测试集和训练集  I\_test=sample(1:nrow(credit),size=nrow(credit)/2)  Test=credit[i\_test,]  Train=credit[-i\_test,] |
| 使用logistic回归建立个人信用风险评估模型：  Logmodel=glm(V15~.,family=binomial(link=logit),data=train)  Logpred=predict(logmodel,test)  Library(pROC)  Logroc=roc(test$V15,logpred)  Plot(logroc,print.auc=TRUE,main=’logistic ROC’) |
| 使用决策树建立个人信用风险评估模型：  Library(tree)  Treemodel=tree(V15~.,data=train)  Plot(treemodel)  Text(treemodel,pretty=0)  Treepred=predict(treemodel,test)  Treeroc=roc(test$V15,treepred)  Plot(treeroc,print.auc=TRUE,main=’tree ROC’) |
| 使用集成方法提升模型的性能，构建模型预测贷款是否违约：  Library(randomForest)  Rfmodel=randomForest(V15~.,data=train)  Rfpred=predict(rfmodel,test)  Rfroc=roc(test$V15,rfpred)  Plot(rfroc,print.auc=TRUE,main=’randomForest ROC’) |

|  |
| --- |
| 单层感知机是线性模型，所以不能表示复杂的函数，如异或XOR。验证感知机为什么不能表示异或。 |
| 感知机是一种线性分类模型：  输入空间，输出空间，输入表示实例的特征向量，得到输出表示实例的类别，由输入空间到输出空间的映射表示为：  异或问题可根据输出分为两类，显示在二维坐标中如下图：    可以发现，异或问题的结果无法找到一条直线直接将两类结果分开，即感知机无法找到一个线性模型对异或问题进行划分。  所以感知机不能表示异或 |
| 构造如下三层感知机：输入层；隐藏层采用阶跃函数；输出层。  +1  +1  +1  +1  +1  -1  -1.5  -0.5  -0.5  隐藏层阶跃函数：  运算结果：  ①(  上：  下：  输出：  ②(  上：  下：  输出：  ③(  上：  下：  输出：  ④(  上：  下：  输出： |

|  |
| --- |
| 算法实例中，隐藏层采用tansig函数，输出层采用logsig函数，采用随机梯度下降，推导误差反向传播公式： |
| 已知输入数据s，输出数据t:  w  v  a  输入层计算：  隐藏层计算：  输出层计算：  误差计算：  随机梯度下降法： |
| 将原始数据归一化，原始权值0.5，学习率0.5依次计算一轮后的权值：  S\_min=min(s)  S\_max=max(s)  X\_min=-1  X\_max=1  X=[x\_min+(si-s\_min)\*(x\_max-x\_min)/(s\_max-s\_min) for si in s]  T\_min=min(t)  T\_max=max(t)  Y\_min=0  Y\_max=1  Y=[y\_min+(ti-t\_min)\*(y\_max-y\_min)/(t\_max-t\_min) for ti in t]  A=0.5  W=0.5  V=0.5  Elta=0.5  #隐藏层计算  U=[a+w\*xi for xi in x]  H=[(1-math.exp(-ui))/(1+math.exp(-ui)) for ui in u]  #输出层计算  Y\_out=[1/(1+math.exp(-v\*hi)) for hi in h]  #误差返向传播  E=[y[i]-y\_out[i] for I in range(9)]  #总误差  e\_all=sum([ei\*\*2 for ei in e])/2  #批量梯度下降  Ev=sum([e[i]\*y\_out[i]\*(1-y\_out[i]) for I in range(9)])  Ea=-sum([e[i]\*y\_out[i]\*(1-y\_out[i])\*(1-h[i]\*\*2) for I in range(9)])  Ew=-sum([e[i]\*y\_out[i]\*(1-y\_out[i])\*(1-h[i]\*\*2)\*x[i] for I in range(9)])  #调整权值  V=v-elta\*Ev  A=a-elta\*Ea  W=w-elta\*Ew |

|  |
| --- |
| 证明：给定[a,b]上一个一维连续函数，总可构造一个隐层的前馈神经网络以任意精度逼近，只要隐层的节点足够多。 |
| 1、如下形式的样条函数以及它的平移缩放形式组合在一起可以任意精度逼近任意一个函数：  即如图所示：  2、样条函数的形状可由两个threshold函数相减所得到，即可得到如下图所示阶梯：    可简单得知，threshold函数的平移放大以及相互的组合i逼近任意函数。  3、单个神经元的作用函数在权重等于无穷的极限情况下可以变形为threshold函数：  可以是常用的sigmoid函数，代表第个神经元，w是输出层的运算权重。  可以看到当的绝对值趋于无穷的时候，sigmoid函数就变形为以为threshold的threshold函数。  综上，可得神经网络的万有逼近定理。 |

|  |
| --- |
| 数据集pima，768女性信息，每名女性9个变量，以disatolic为因变量，其他为自变量，建立线性回归和随机网络模型，比较预测效果。 |
| Data(pima)  #查看数据  Pima  #线性回归  Fitline=lm(diastolic~.,data=pima)  Summary(fitline)  Predictline=predict(fitline,data=pima)  #均方根误差RMSE  Sqrt(sum((predictline-pima$diastolic)^2)/length(pima$diastolic))  #随机网络模型：BP神经网络  #数据标准化  Nomalize=function(x){  Return((x-min(x))/(max(x)-min(x)))  }  Pima\_norm=as.data.frame(lapply(pima,normalize))  N=dim(pima)[1]  Set.seed(1)  Train\_index=sample(1:n,round(n\*0.7))  Train=pima\_norm[train\_index,]  Test=pima\_norm[-train\_index,]  #隐藏层神经元个数为3  Model=nnet(diastolic~.,data=train,size=3,rang=r,decat=1e-5,maxit=1000)  Summary(model)  #在测试机上进行预测  Pred\_test=predict(model,test[,-3])  Rmse=sqrt(sum((pred\_test-test$diastolic)^2)/length(pred\_test))  Rmse  (max(pima$diastolic)-min(pima$diastolic)\*rmse  #在整个数据集进行预测  Pred\_data=predict(model,pima\_norm[,-3])  Rmse\_data=sqrt(sum((pred\_data-pima\_norm$diastolic)^2)/length(pred\_data))  Rmse\_data\*(max(pima$diastolic)-min(pima$diastolic)) |

|  |
| --- |
| 分别用logistic回归和神经网络拟合垃圾邮件分类数据spam： |
| #查看数据  Df  Dim(df)  Table(df$V58)/dim(df)[1]  #logistic回归  N=dim(df)[1]  Set.seed(1)  Train\_index=sample(1:n,round(n\*0.7))  Train=df[train\_index,]  Test=df[-train\_index,]  Logistic.full=glm(V58~.,family=binomial(link=logit),data=train)  Summary(logistic.full)  #测试集上预测  Pred\_test=predict(logistic.full,test)  Y\_hat=1\*(pred>0.5)  Table(y\_hat,test$V58)  Sum(test$V58==y\_hat)/dim(test)[1]  #BP神经网络  #标准化  Normalize=function(x){  Return((x-min(x))/(max(x)-min(x)))  }  Df\_norm=as.data.frame(lapply(df.normalize))  N=dim(df)[1]  Set.seed(1)  Train\_index=sample(1:n,round(n\*0.7))  Train=df\_norm[train\_index,]  Test=df\_norm[-train\_index,]  R=1/max(abs(train[,-58]))  Set.seed(2)  Model=nnet(as.factor(V58)~.,data=train,size=3,rang=r,decat=1e-5,maxit=1000)  Summary(model)  #在测试集上预测  Pred\_test=predict(model,test[,-58])  Y\_test=1\*(pred\_test>0.5)  Table(test$V58,y\_test)  Sum(test$V58==y\_test)/dim(test)[1] |

|  |
| --- |
| （1）卷积核为2×2的单位矩阵，步长为1： |
| （2）对（1）池化核为2×2，步长为1，最大池化： |
| （3）原数据池化核2×2，步长为2，均值池化： |

|  |
| --- |
| 对于首先Boltzmann机，证明： |
| （1）条件独立性：  易看出，该结果是与有关分式的乘积。  故是条件独立的。  即  同理， |
| （2）激活概率：  隐层，偏置，可见层，偏置，是二值变量  已知，，  引入,  有  则  即  同理 |
| （3）最大似然估计方程： |

|  |
| --- |
| 一个卷积神经网络有6层：输入层+卷积层1+卷积层2+池化层+全连接层+输出层，导出其完整的参数更新公式： |
| 1、前向传播  第层的输出为：  其中，是激活函数，是层的输出，也就是第层的输入，和是第层的权值和偏置。  2、反向传播  损失函数：  其中，  损失函数的第层到输出层的参数集的梯度：  同理，  其中，  若激活函数是sigmoid函数，则有：  根据求导的链式法则，得到递推公式：  根据递推公式可求得每一层的梯度，利用更新公式更新参数： |

|  |
| --- |
| 分别使用普通神经网络（1个隐层）和深度信念网对鸢尾花数据集iris.txt进行学习 |
| #查看数据  Data=pd.read\_table(file,header=None,sep=’’)  #一个隐层的神经网络：  Import pandas as pd  From sklearn.model\_selection import train\_test\_split  #加载数据  Data=pd.reas\_csv(‘iris.txt’,header=None,sep=’’)  Data.columns=[‘sepal length’,’sepal width’,’petal length’,’petal width’,’class’]  Data.head()  #数据向量化  X=data.iloc[:,0:4].values.astype(float)  Y=data.iloc[:,4].values.astype(int)  #划分数据  Train\_x,test\_x,train\_y,test\_t=train\_test\_split(x,y,train\_size=0.8,test\_size=0.2)  #数据标准化  Mean=train\_x.mean(axis=0)  Std=train\_x.std(axis=0)  Train\_x=(train\_x-mean)/std  Test\_x=(test\_x-mean)/std  #标签onehot编码  From keras.utils import np\_utils  Train\_y\_ohe=np\_utils.to\_categorical(train\_y,)  Test\_y\_ohe=np\_utils.to\_categorical(test\_y,)  #定义网络结构  From keras.models import Sequential  Model = Sequential()  From keras.layers import Dense,Dropout  Model.add(Dense(8,activation=’relu’,input\_shape(4,)))  Model.add(Dense(3,activation=’softmax’))  #编译并训练网络  Model.compile(loss=’categorical\_crossentropy’,optimizer=’RMSprop’,metrics=[‘accuracy’])  Model.fit(train\_x,train\_x\_ohe,epochs=10,batch\_size=1,verbose=2)  #模型评估  Loss,accuracy=model.evaluate(test\_x,test\_y\_ohe,verbose=2)  Print(‘loss={},accuracy={}’.format(loss,accuracy))  #模型预测  Classes=model.predict(test\_x,batch\_size=1,verbose=2)  Print(‘test:’,classes) |
| #深度信念网  From keras.models import Sequential  Model=Sequential()  From keras.layers import Dense,Dropout  Model.add(Dense(16,activation=’relu’,imput\_shape=(4,)))  Model.add(Dense(16,activation=’relu’))  Model.add(Dense(3,activation=’softmax’))  #编译并训练网络  Model.compile(loss=’categorical\_crossentropy’,optimizer=’RMSprop’,metrics=[‘accuracy’])  Model.fit(train\_s,train\_y\_ohe,epochs=10,batch\_size=1,verbose=2)  #模型评估  Loss,accuracy=model.evaluate(test\_x,test\_y\_ohe,verbose=2)  Print(‘loss={},accuracy={}’.format(loss,accuracy))  #模型预测  Classes=model.predict(test\_x,batch\_size=1,verbose=2)  Print(‘test:’,classes) |
| A=read.csv(‘iris.txt’,header=False,sep=’’)  Table(a$V5)#查看各个类别的数目  A[,5]=as.factor(a[,5])  Moemalize=function(x){return ((x-min(x))/(max(x)-min(x)))}  A\_norm=a  A\_norm[,1:4]=lapply(a[,1:4],normalize)#前4列标准化  Set.seed(1)  N=dim(a\_norm)[1]  Test\_index=sample(1:n,round(n\*0.3))  Train=a\_norm[-test\_index,]  Test=a\_norm[test\_index,]  N\_train=dim(train)[1]  N\_test=dim(test)[1]  Summary(model)  Pred\_test=predict(model,test[,1:4],type=’class’)  Table(test[,5],pred\_test)#混淆矩阵  Sum(pred\_test==test[,5])/dim(test)[1] |

|  |
| --- |
| 分别用普通神经网络（1个隐层）和卷积神经网络对手写数字数据集upus学习： |
| #一个隐层的神经网络  Import keras  From keras import models  From keras import layers  #定义网络结构  Net=models.Sequential()  Net.add(layers.Dense(256,activation=’relu’,input\_shape=(16\*16,)))  Net.add(layers.Dense(1-,activation=’softmax’))  #编译与训练  Net.compile(optimizer=’RMSprop’,loss=’categorical\_crossentropy’,metrics=[‘accuracy’])  Net.fit(train\_s,train\_y\_ohe,epochs=10,batch\_size=1,verbose=2)  #模型评价  Loss,acc=net.evaluate(test\_x,test\_y\_ohe)  Print(‘acc:’,acc) |
| #卷积神经网络  From keras import layers  From keras import models  Model=models.Sequential()  Mode.add(layers.Conv2D(32,(3,3),activation=’relu’,input\_shape=(16,16,1)))  Model.add(layers.MaxPooling2D((2,2)))  Model.add(layers.Conv2D(63,(3,3),activation=’relu’))  Model.add(MaxPooling2D((2,2))  Model.add(layers.Conv2D(64,(3,3),activation=’relu))  #添加分类器  Model.add(layers.Flattern()))  Model.add(layers.Dense(64,activation=’relu’))  Model.add(layers.Dense(10,activation=’softmax’))  #编译训练网络  Model.compile(optimizer=’rmsprop’,loss=’categorical\_crossentropy’,metrics=[‘accuracy’])  Model.fit(train\_x,train\_y\_ohe,epochs=5,batch\_size=64)  #模型评价  Loss,acc=model.evaluate(test\_x,test\_y)  Print(‘acc:’,acc) |
| Library(R.matlab)  Usps=readMat(‘usps\_all.mat’,encoding=’utf-8)  Data=usps$data  Dim(data  Df(matrix(0,nrow=11000,ncol=257)  For(k in 1:10){  For (j in 1:1100){  Index=1100\*(k-1)+j  For(I in 1:256){  Df[index,i]=data[I,j,k]  }  Df[index,257]=k-1  }  }#将原数据进行转化  Df=as.data.frame(df)  Dim(df)  #归一化  Df[1:256]=df[1,256]/255  Df[,257]=as.factor(df[,257])  #划分数据集  Set.seed(1)  N=dim(df)[1]  Test\_index=sample(1:n,round(n\*0.3))  Ttrain=df[-test\_index,]  Test=df[test\_index,]  N\_train=dim(train)[1]  N\_test=dim(test)[1]  Library(nnet)  R=1/max(abs(train[,1:256]))  Set.seed(2)  Model=nnet(V257~.,data=train,size=3,rang=r,decay=1e-5,maxit=400)  Summary(model)  Pred=predict(model,test[,1:256],type=’class’)  Table(test$V257,pred)  Sum(pred==test$V257)/dim(test)[1] |
| #深度神经网络  Sys.setenv(JAVA\_HOME=’jdk1.8.0\_291/’)  Library(h2o)  H2o.init()  Train.hex=as.h2o(train)  Test.hex=as.h2o(test)  Train.hex[,257]=as.factor(train.hex[,257])  Model=h2o.deeplearning(x=1:256,y=257,train\_frame=train.hex,activation=’tanh’,hindden=rep(160,5),epochs=20)  Pred\_dl=h2o.predict(object=model,newdata=test.hex)  Pred=as.data.frame(pred\_dl)  Table(test$V257,pred$predict)  Sum(pred$predict==test$V257)/dim(test)[1] |

|  |
| --- |
| （1）导出对偶问题的KKT条件：    SVM原始问题可看作是一个不等式约束的凸二次优化问题：  通过拉格朗日对偶性找等价对偶问题：  则优化目标为：  ①  ②消除：  即  对应的KKT条件为： |
| （2）不考虑不等式约束条件下，证明SMO算法，以及和的更新公式    对偶问题：  记，  则  约束条件化简：  无约束：求偏导为0：  令，有，  则  则  由  知，  其中，  令  有  再  则  令  则  则，又，  故 |

|  |
| --- |
| 证明合页损失下的SVM最优估计的概率意义为： |
| 线性SVM原始最优化问题的等价形式：  由，对求导并令导数为0，即  有  从而得到合页损失下的最优估计 |