**KNN算法在保险业精准营销中的应用**

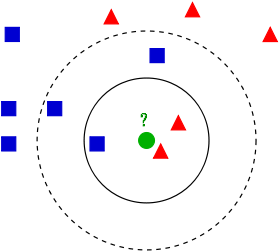
### 一、KNN算法概述

KNN是Machine Learning领域一个简单又实用的算法，与之前讨论过的算法主要存在两点不同：

1. 它是一种非参方法。即不必像线性回归、逻辑回归等算法一样有固定格式的模型，也不需要去拟合参数。
2. 它既可用于分类，又可应用于回归。

KNN的基本思想有点类似“物以类聚，人以群分”，打个通俗的比方就是“如果你要了解一个人，可以从他最亲近的几个朋友去推测他是什么样的人”。

在分类领域，对于一个未知点，选取K个距离（可以是欧氏距离，也可以是其他相似度度量指标）最近的点，然后统计这K个点，在这K个点中频数最多的那一类就作为分类结果。比如下图，若令K=4，则?处应分成红色三角形；若令K=6，则?处应分类蓝色正方形。



在回归（简单起见，这里讨论一元回归）领域，如果只知道某点的预测变量$x$，要回归响应变量$y$，只需要在横坐标轴上（因为不知道纵坐标的值，所以没法计算欧氏距离）选取K个最近的点，然后平均（也可以加权平均）这些点的响应值，作为该点的响应值即可。比如下图中，已知前5个点的横纵坐标值，求$x=6.5$时，$y$为多少？若令K=2，则距6.5最近的2个点是(5.1, 8)和(4, 27)，把这两个点的纵坐标平均值17.5就可以当作回归结果，认为$x=6.5时，y=17.5。$

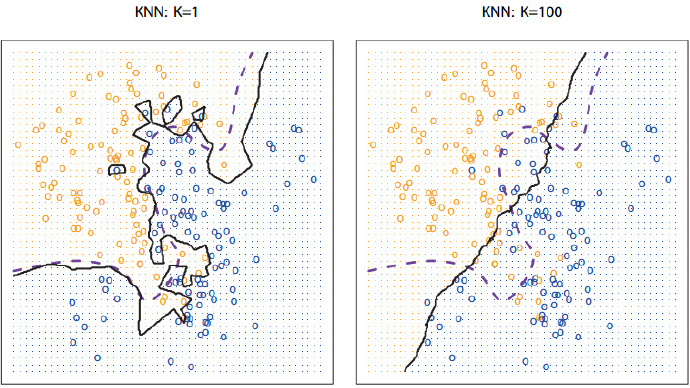
KNN具体的算法步骤可参考延伸阅读文献1。

### 二、KNN性能讨论

KNN的基本思想与计算过程很简单，你只需要考虑两件事：

1. K预设值取多少？
2. 如何定义距离？

其中如何定义距离这个需要结合具体的业务应用背景，本文不细致讨论，距离计算方法可参看延伸阅读文献2。这里只讨论K取值时对算法性能的影响。



在上图中，紫色虚线是贝叶斯决策边界线，也是最理想的分类边界，黑色实线是KNN的分类边界。

可以发现：**K越小，分类边界曲线越光滑，偏差越小，方差越大；K越大，分类边界曲线越平坦，偏差越大，方差越小。**

所以即使简单如KNN，同样要考虑偏差和方差的权衡问题，表现为K的选取。

KNN的优点就是简单直观，无需拟合参数，在样本本身区分度较高的时候效果会很不错；但缺点是当样本量大的时候，找出K个最邻近点的计算代价会很大，会导致算法很慢，此外KNN的可解释性较差。

KNN的一些其他问题的思考可参看延伸阅读文献3。

### 三、实战案例

#### 1、KNN在保险业中挖掘潜在用户的应用

这里应用ISLR包里的Caravan数据集，先大致浏览一下：

1. > library(ISLR)
2. > str(Caravan)
3. 'data.frame': 5822 obs. of 86 variables:
4. $ Purchase: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
5. > table(Caravan$Purchase)/sum(as.numeric(table(Caravan$Purchase)))
6. No Yes
7. 0.94022673 0.05977327

5822行观测，86个变量，其中只有Purchase是分类型变量，其他全是数值型变量。Purchase两个水平，No和Yes分别表示不买或买保险。可见到有约6%的人买了保险。

由于KNN算法要计算距离，这85个数值型变量量纲不同，相同两个点在不同特征变量上的距离差值可能非常大。因此要**归一化**，这是Machine Learning的常识。这里直接用scale()函数将各连续型变量进行正态标准化，即转化为服从均值为0，标准差为1的正态分布。

1. > standardized.X=scale(Caravan[,-86])
2. > mean(standardized.X[,sample(1:85,1)])
3. [1] -2.047306e-18
4. > var(standardized.X[,sample(1:85,1)])
5. [1] 1
6. > mean(standardized.X[,sample(1:85,1)])
7. [1] 1.182732e-17
8. > var(standardized.X[,sample(1:85,1)])
9. [1] 1
10. > mean(standardized.X[,sample(1:85,1)])
11. [1] -3.331466e-17
12. > var(standardized.X[,sample(1:85,1)])
13. [1] 1

可见随机抽取一个标准化后的变量，基本都是均值约为0，标准差为1。

1. > #前1000观测作为测试集，其他当训练集
2. > test <- 1:1000
3. > train.X <- standardized.X[-test,]
4. > test.X <- standardized.X[test,]
5. > train.Y <- Caravan$Purchase[-test]
6. > test.Y <- Caravan$Purchase[test]
7. > knn.pred <- knn(train.X,test.X,train.Y,k=)
8. > mean(test.Y!=knn.pred)
9. [1] 0.117
10. > mean(test.Y!="No")
11. [1] 0.059

当K=1时，KNN总体的分类结果在测试集上的错误率约为12%。由于大部分的人都不买保险（先验概率只有6%），那么如果模型预测不买保险的准确率应当很高，纠结于预测不买保险实际上却买保险的样本没有意义，同样的也不必考虑整体的准确率（Accuracy）。作为保险销售人员，只需要关心在模型预测下会买保险的人中有多少真正会买保险，这是精准营销的精确度（Precision）；因此，在这样的业务背景中，**应该着重分析模型的Precesion，而不是Accuracy。**

1. > table(knn.pred,test.Y)
2. test.Y
3. knn.pred No Yes
4. No 874 50
5. Yes 67 9
6. > 9/(67+9)
7. [1] 0.1184211

可见K=1时，KNN模型的Precision约为12%，是随机猜测概率（6%）的两倍！

下面尝试K取不同的值：

1. > knn.pred <- knn(train.X,test.X,train.Y,k=3)
2. > table(knn.pred,test.Y)[2,2]/rowSums(table(knn.pred,test.Y))[2]
3. Yes
4. 0.2
5. > knn.pred <- knn(train.X,test.X,train.Y,k=5)
6. > table(knn.pred,test.Y)[2,2]/rowSums(table(knn.pred,test.Y))[2]
7. Yes
8. 0.2666667

可以发现当K=3时，Precision=20%；当K=5时，Precision=26.7%。

作为对比，这个案例再用逻辑回归做一次！

1. > glm.fit <- glm(Purchase~.,data=Caravan,family = binomial,subset = -test)
2. Warning message:
3. glm.fit:拟合機率算出来是数值零或一
4. > glm.probs <- predict(glm.fit,Caravan[test,],type = "response")
5. > glm.pred <- ifelse(glm.probs >0.5,"Yes","No")
6. > table(glm.pred,test.Y)
7. test.Y
8. glm.pred No Yes
9. No 934 59
10. Yes 7 0

这个分类效果就差很多，Precision竟然是0！事实上，分类概率阈值为0.5是针对等可能事件，但买不买保险显然不是等可能事件，把阈值降低到0.25再看看：

1. > glm.pred <- ifelse(glm.probs >0.25,"Yes","No")
2. > table(glm.pred,test.Y)
3. test.Y
4. glm.pred No Yes
5. No 919 48
6. Yes 22 11

这下子Precision就达到1/3了，比随机猜测的精确度高出5倍不止！

以上试验都充分表明，**通过机器学习算法进行精准营销的精确度比随机猜测的效果要强好几倍！**

#### 2、KNN回归

在R中，KNN分类函数是knn()，KNN回归函数是knnreg()。

1. > #加载数据集BloodBrain，用到向量logBBB和数据框bbbDescr
2. > library(caret)
3. > data(BloodBrain)
4. > class(logBBB)
5. [1] "numeric"
6. > dim(bbbDescr)
7. [1] 208 134
8. > #取约80%的观测作训练集。
9. > inTrain <- createDataPartition(logBBB, p = .8)[[1]]
10. > trainX <- bbbDescr[inTrain,]
11. > trainY <- logBBB[inTrain]
12. > testX <- bbbDescr[-inTrain,]
13. > testY <- logBBB[-inTrain]
14. > #构建KNN回归模型
15. > fit <- knnreg(trainX, trainY, k = 3)
16. > fit
17. 3-nearest neighbor regression model
18. > #KNN回归模型预测测试集
19. > pred <- predict(fit, testX)
20. > #计算回归模型的MSE
21. > mean((pred-testY)^2)
22. [1] 0.5821147

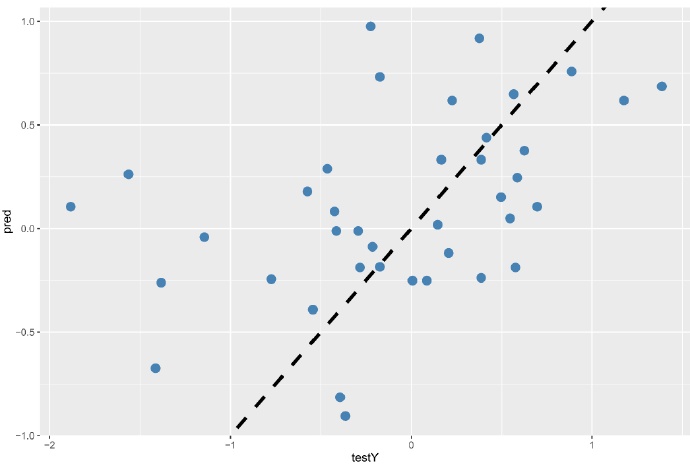
这个KNN回归模型的MSE只有0.58，可见回归效果很不错，偏差很小！下面用可视化图形比较一下结果。

1. > #将训练集、测试集和预测值结果集中比较
2. > df <-data.frame(class=c(rep("trainY",length(trainY)),rep("testY",length(testY)),rep("predY",length(pred))),Yval=c(trainY,testY,pred))
3. > ggplot(data=df,mapping = aes(x=Yval,fill=class))+
4. + geom\_dotplot(alpha=0.8)



这是dotplot，横坐标才是响应变量的值，纵坐标表频率。比较相邻的红色点和绿色点在横轴上的差异，即表明测试集中预测值与实际值的差距。

1. > #比较测试集的预测值和实际值
2. > df2 <- data.frame(testY,pred)
3. > ggplot(data=df2,mapping = aes(x=testY,y=pred))+
4. + geom\_point(color="steelblue",size=3)+
5. + geom\_abline(slope = 1,size=1.5,linetype=2)



这张散点图则直接将测试集中的实际值和预测值进行对比，虚线是$y=x$。点离这条虚线越近，表明预测值和实际值之间的差异就越小。

### 参考文献

Gareth James et al. An Introduction to Statistical Learning.

Wikipedia. [k-nearest neighbors algorithm](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm).

[KNN for Smoothing and Prediction](http://people.revoledu.com/kardi/tutorial/KNN/KNN_TimeSeries.htm).

整理自<http://shujuren.org/article/168.html>

原作者：依然很拉风