3 Pre-Processing

`caret` includes several functions to pre-process the predictor data. It assumes that all of the data are numeric (i.e. factors have been converted to dummy variables via `model.matrix`, `dummyVars` or other means).

Note that the later chapter on using `recipes` with `train` shows how that approach can offer a more diverse and customizable interface to pre-processing in the package.

3.1 Creating Dummy Variables

The function `dummyVars` can be used to generate a complete (less than full rank parameterized) set of dummy variables from one or more factors. The function takes a formula and a data set and outputs an object that can be used to create the dummy variables using the predict method.

For example, the `etitanic` data set in the `earth` package includes two factors: `pclass` (passenger class, with levels 1st, 2nd, 3rd) and `sex` (with levels female, male). The base R function `model.matrix` would generate the following variables:

```
library(earth)
data(etitanic)
head(model.matrix(survived ~ ., data = etitanic))
```

```
##   (Intercept) pclass2nd pclass3rd sexmale     age sibsp parch
## 1           1         0         0       0 29.0000     0     0
## 2           1         0         0       1  0.9167     1     2
## 3           1         0         0       0  2.0000     1     2
## 4           1         0         0       1 30.0000     1     2
## 5           1         0         0       0 25.0000     1     2
## 6           1         0         0       1 48.0000     0     0
```

Using `dummyVars` :

```
dummies <- dummyVars(survived ~ ., data = etitanic)
head(predict(dummies, newdata = etitanic))
```

```
##   pclass.1st pclass.2nd pclass.3rd sex.female sex.male     age
## 1          1          0          0          1        0 29.0000
## 2          1          0          0          0        1  0.9167
## 3          1          0          0          1        0  2.0000
## 4          1          0          0          0        1 30.0000
## 5          1          0          0          1        0 25.0000
## 6          1          0          0          0        1 48.0000
```

Note there is no intercept and each factor has a dummy variable for each level, so this parameterization may not be useful for some model functions, such as `lm` .

3.2 Zero- and Near Zero-Variance Predictors

In some situations, the data generating mechanism can create predictors that only have a single unique value (i.e. a “zero-variance predictor”). For many models (excluding tree-based models), this may cause the model to crash or the fit to be unstable.

Similarly, predictors might have only a handful of unique values that occur with very low frequencies. For example, in the drug resistance data, the `nR11` descriptor (number of 11-membered rings) data have a few unique numeric values that are highly unbalanced:

```
data(mdrr)
data.frame(table(mdrrDescr$nR11))
```

```
##   Var1 Freq
## 1    0  501
## 2    1    4
## 3    2   23
```

The concern here that these predictors may become zero-variance predictors when the data are split into cross-validation/bootstrap sub-samples or that a few samples may have an undue influence on the model. These “near-zero-variance” predictors may need to be identified and eliminated prior to modeling.

To identify these types of predictors, the following two metrics can be calculated:

- the frequency of the most prevalent value over the second most frequent value (called the “frequency ratio’’), which would be near one for well-behaved predictors and very large for highly-unbalanced data and
- the “percent of unique values’’ is the number of unique values divided by the total number of samples (times 100) that

approaches zero as the granularity of the data increases

If the frequency ratio is greater than a pre-specified threshold and the unique value percentage is less than a threshold, we might consider a predictor to be near zero-variance.

We would not want to falsely identify data that have low granularity but are evenly distributed, such as data from a discrete uniform distribution. Using both criteria should not falsely detect such predictors.

Looking at the MDRR data, the `nearZeroVar` function can be used to identify near zero-variance variables (the `saveMetrics` argument can be used to show the details and usually defaults to `FALSE`):

```
nzv <- nearZeroVar(mdrrDescr, saveMetrics= TRUE)
nzv[nzv$nzv,][1:10,]
```

```
##        freqRatio percentUnique zeroVar  nzv
## nTB     23.00000     0.3787879   FALSE TRUE
## nBR    131.00000     0.3787879   FALSE TRUE
## nI     527.00000     0.3787879   FALSE TRUE
## nR03   527.00000     0.3787879   FALSE TRUE
## nR08   527.00000     0.3787879   FALSE TRUE
## nR11    21.78261     0.5681818   FALSE TRUE
## nR12    57.66667     0.3787879   FALSE TRUE
## D.Dr03 527.00000     0.3787879   FALSE TRUE
## D.Dr07 123.50000     5.8712121   FALSE TRUE
## D.Dr08 527.00000     0.3787879   FALSE TRUE
```

```
dim(mdrrDescr)
```

```
## [1] 528 342
```

```
nzv <- nearZeroVar(mdrrDescr)
filteredDescr <- mdrrDescr[, -nzv]
dim(filteredDescr)
```

nearZeroVar diagnoses predictors that have one unique value (i.e. are zero variance predictors) or predictors that are have both of the following characteristics: they have very few unique values relative to the number of samples and the ratio of the frequency of the most common value to the frequency of the second most common value is large.

```
## [1] 528 297
```

By default, `nearZeroVar` will return the positions of the variables that are flagged to be problematic.

3.3 Identifying Correlated Predictors

While there are some models that thrive on correlated predictors (such as `pls`), other models may benefit from reducing the level of correlation between the predictors.

Given a correlation matrix, the `findCorrelation` function uses the following algorithm to flag predictors for removal:

为什么取上三角？

```
descrCor <-  cor(filteredDescr)
highCorr <- sum(abs(descrCor[upper.tri(descrCor)]) > .999)
```

For the previous MDRR data, there are 65 descriptors that are almost perfectly correlated (|correlation| > 0.999), such as the total information index of atomic composition (`IAC`) and the total information content index (neighborhood symmetry of 0-order) (`TIC0`) (correlation = 1). The code chunk below shows the effect of removing descriptors with absolute correlations above 0.75.

```
descrCor <- cor(filteredDescr)
summary(descrCor[upper.tri(descrCor)])
```

二者区别在与remove 超过.75相关性的数据？
为什么取上三角？

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.99607 -0.05373  0.25006  0.26078  0.65527  1.00000
```

```
highlyCorDescr <- findCorrelation(descrCor, cutoff = .75)
filteredDescr <- filteredDescr[,-highlyCorDescr]
descrCor2 <- cor(filteredDescr)
summary(descrCor2[upper.tri(descrCor2)])
```

要识别和移除强相关预测因子.
This function searches through a correlation matrix and returns a vector of integers corresponding to columns to remove to reduce pair-wise correlations.

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.70728 -0.05378  0.04418  0.06692  0.18858  0.74458
```

3.4 Linear Dependencies

The function `findLinearCombos` uses the QR decomposition of a matrix to enumerate sets of linear combinations (if they exist). For example, consider the following matrix that is could have been produced by a less-than-full-rank parameterizations of a two-way experimental layout:

```
ltfrDesign <- matrix(0, nrow=6, ncol=6)
ltfrDesign[,1] <- c(1, 1, 1, 1, 1, 1)
ltfrDesign[,2] <- c(1, 1, 1, 0, 0, 0)
ltfrDesign[,3] <- c(0, 0, 0, 1, 1, 1)
ltfrDesign[,4] <- c(1, 0, 0, 1, 0, 0)
ltfrDesign[,5] <- c(0, 1, 0, 0, 1, 0)
ltfrDesign[,6] <- c(0, 0, 1, 0, 0, 1)
```

Note that columns two and three add up to the first column. Similarly, columns four, five and six add up the first column. `findLinearCombos` will return a list that enumerates these dependencies. For each linear combination, it will incrementally remove columns from the matrix and test to see if the dependencies have been resolved.

`findLinearCombos` will also return a vector of column positions can be removed to eliminate the linear dependencies:

```
comboInfo <- findLinearCombos(ltfrDesign)
comboInfo
```

第2、3列加起来就是第1列。类似的，第4、5、6列加起来就是第1列。返回将沿着列向量的位置枚举的相关性列表，它们可以被删除以去除线性相关性。Enumerate and resolve the linear combinations in a numeric matrix

```
## $linearCombos
## $linearCombos[[1]]
## [1] 3 1 2
##
## $linearCombos[[2]]
## [1] 6 1 4 5
##
##
## $remove
## [1] 3 6
```

```
ltfrDesign[, -comboInfo$remove]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    1    0
## [2,]    1    1    0    1
## [3,]    1    1    0    0
## [4,]    1    0    1    0
## [5,]    1    0    0    1
## [6,]    1    0    0    0
```

These types of dependencies can arise when large numbers of binary chemical fingerprints are used to describe the structure of a molecule.

3.5 The `preProcess` Function

The `preProcess` class can be used for many operations on predictors, including centering and scaling. The function `preProcess` estimates the required parameters for each operation and `predict.preProcess` is used to apply them to specific data sets. This function can also be interfaces when calling the `train` function.

Several types of techniques are described in the next few sections and then another example is used to demonstrate how multiple methods can be used. Note that, in all cases, the `preProcess` function estimates whatever it requires from a specific data set (e.g. the training set) and then applies these transformations to *any* data set without recomputing the values

3.6 Centering and Scaling

In the example below, the half of the MDRR data are used to estimate the location and scale of the predictors. The function `preProcess` doesn't actually pre-process the data. `predict.preProcess` is used to pre-process this and other data sets.

设置随机数种子，取相同的随机数，使结果具有重复性

mdrrDescr: the descriptors
mdrrClass: the categorical outcome ("Active" or "Inactive")

```
set.seed(96)
inTrain <- sample(seq(along = mdrrClass), length(mdrrClass)/2)

training <- filteredDescr[inTrain,]
test <- filteredDescr[-inTrain,]
trainMDRR <- mdrrClass[inTrain]
testMDRR <- mdrrClass[-inTrain]

preProcValues <- preProcess(training, method = c("center", "scale

trainTransformed <- predict(preProcValues, training)
testTransformed <- predict(preProcValues, test)
```

seq(along=)求相同长度的索引数。就是取数字用于索引

filteredDescr （528，297），mdrrclass是528项（行），1/2用于训练1/2用于测试，plot时取两列做，group为mdrrclass。

method = "center" subtracts the mean of the predictor's data (again from the data in x) from the predictor values while method = "scale" divides by the standard deviation.

input newdata x, output predicted y -> transformed

The `preProcess` option `"range"` scales the data to the interval between zero and one.

3.7 Imputation

`preProcess` can be used to impute data sets based only on information in the training set. One method of doing this is with K-nearest neighbors. For an arbitrary sample, the K closest neighbors are found in the training set and the value for the predictor is imputed using these values (e.g. using the mean). Using this approach will automatically trigger `preProcess` to center and scale the data, regardless of what is in the `method` argument. Alternatively, bagged trees can also be used to impute. For each predictor in the data, a bagged tree is created using all of the other predictors in the training set. When a new sample has a missing predictor value, the bagged model is used to predict the value. While, in theory, this is a more powerful method of imputing, the computational costs are much higher than the nearest neighbor technique.

In KNN, if some training data are missed, use mean, then proProcess to center and scale the data.
Bagged trees predict the missing training data which is more powerful.

3.8 Transforming Predictors

In some cases, there is a need to use principal component analysis (PCA) to transform the data to a smaller sub–space where the new variable are uncorrelated with one another. The `preProcess` class can apply this transformation by including `"pca"` in the `method` argument. Doing this will also force scaling of the predictors. Note that when PCA is requested, `predict.preProcess` changes the column names to `PC1`, `PC2` and so on.

Similarly, independent component analysis (ICA) can also be used to find new variables that are linear combinations of the original set such that the components are independent (as opposed to uncorrelated in

PCA). The new variables will be labeled as `IC1`, `IC2` and so on.

The “spatial sign” transformation (Serneels et al, 2006) projects the data for a predictor to the unit circle in p dimensions, where p is the number of predictors. Essentially, a vector of data is divided by its norm. The two figures below show two centered and scaled descriptors from the MDRR data before and after the spatial sign transformation. The predictors should be centered and scaled before applying this transformation.

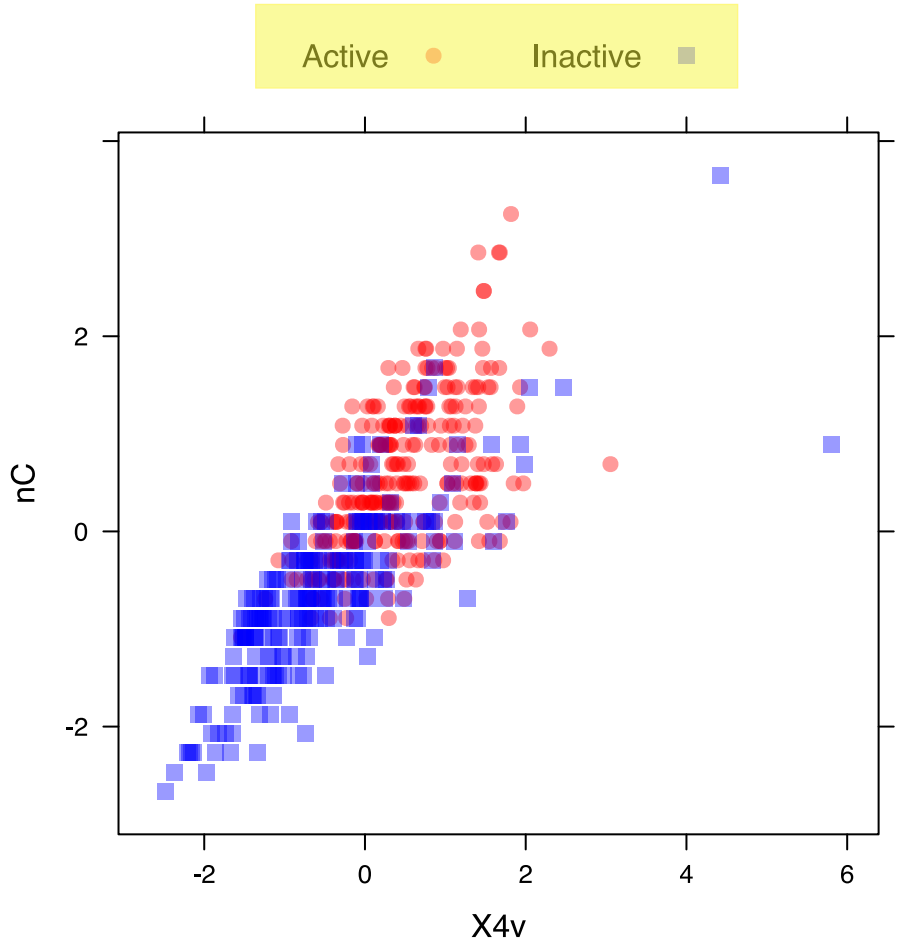
```
library(AppliedPredictiveModeling)
transparentTheme(trans = .4)
```

```
plotSubset <- data.frame(scale(mdrrDescr[, c("nC", "X4v")]))
xyplot(nC ~ X4v,
       data = plotSubset,
       groups = mdrrClass,
       auto.key = list(columns = 2))
```

data.frame() can be used to build a data frame while as.data.frame() can only be used to coerce other object to a data frame.
same results of the two functions.

scale: devided by standard deviation

Only data.frame can be used in xyplot

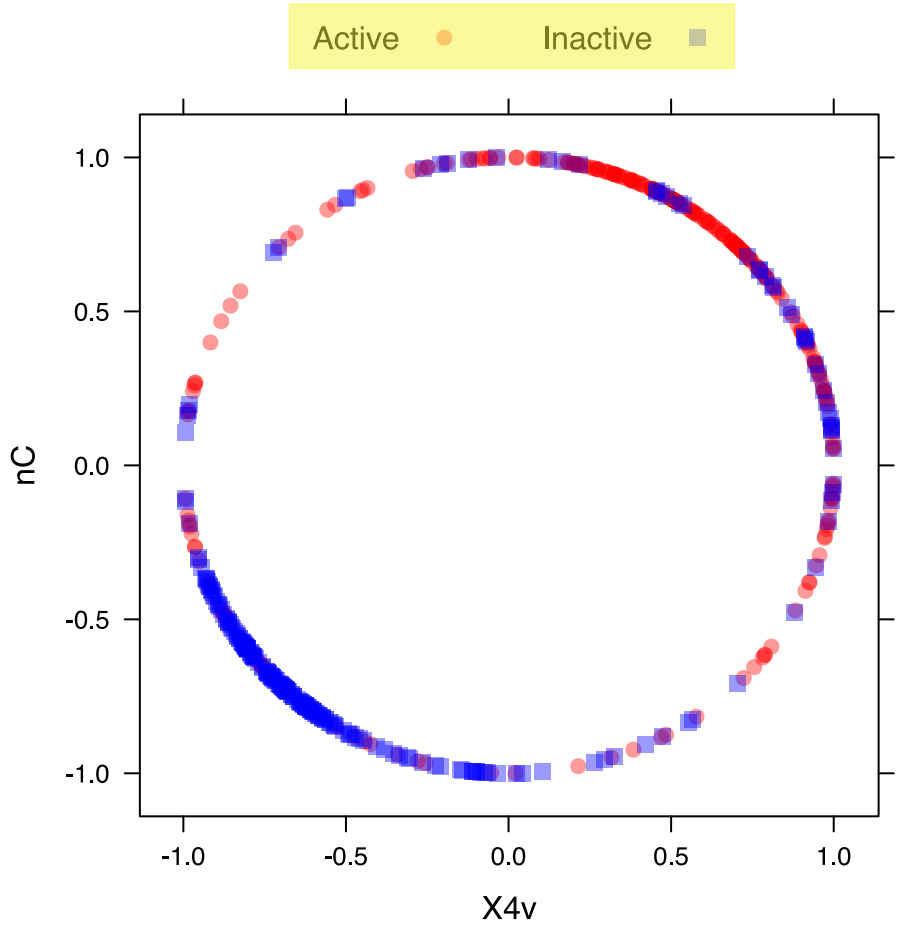


这个图能反映什么吗？
如果NC和X4V与active和inactive真的有相关性（或者说用于分类？）那么这个图是什么样的呢？

After the spatial sign:

```
transformed <- spatialSign(plotSubset)
transformed <- as.data.frame(transformed)
xyplot(nC ~ X4v,
       data = transformed,
       groups = mdrrClass,
       auto.key = list(columns = 2))
```

Compute the spatial sign (a projection of a data vector to a unit length circle). The spatial sign of a vector w is w / norm(w).



同样的问题，这个图有什么用？

Another option, `"BoxCox"` will estimate a Box–Cox transformation on the predictors if the data are greater than zero.

```
preProcValues2 <- preProcess(training, method = "BoxCox")
trainBC <- predict(preProcValues2, training)
testBC <- predict(preProcValues2, test)
preProcValues2
```

BoxCox: 一种神奇的可以在一定程度上减小不可观测的误差和预测变量的相关性的数据变换方式。（自变量只能是正数）

predict (model, newdata)

```
## Created from 264 samples and 31 variables
##
## Pre-processing:
##   - Box-Cox transformation (31)
##   - ignored (0)
##
## Lambda estimates for Box-Cox transformation:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.0000 -0.2000  0.3000  0.4097  1.7000  2.0000
```

用BoxCox估计lamda变换参数

The `NA` values correspond to the predictors that could not be transformed. This transformation requires the data to be greater than zero. Two similar transformations, the Yeo-Johnson and exponential transformation of Manly (1976) can also be used in `preProcess`.

3.9 Putting It All Together

In *Applied Predictive Modeling* there is a case study where the execution times of jobs in a high performance computing environment are being predicted. The data are:

```
library(AppliedPredictiveModeling)
data(schedulingData)
str(schedulingData)
```

```
## 'data.frame':    4331 obs. of  8 variables:
##  $ Protocol   : Factor w/ 14 levels "A","C","D","E",..: 4 4 4 4
##  $ Compounds  : num  997 97 101 93 100 100 105 98 101 95 ...
##  $ InputFields: num  137 103 75 76 82 82 88 95 91 92 ...
##  $ Iterations : num  20 20 10 20 20 20 20 20 20 20 ...
##  $ NumPending : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Hour       : num  14 13.8 13.8 10.1 10.4 ...
##  $ Day        : Factor w/ 7 levels "Mon","Tue","Wed",..: 2 2 4
##  $ Class      : Factor w/ 4 levels "VF","F","M","L": 2 1 1 1 1
```

The data are a mix of categorical and numeric predictors. Suppose we want to use the Yeo-Johnson transformation on the continuous predictors then center and scale them. Let's also suppose that we will be running a tree-based models so we might want to keep the factors as factors (as opposed to creating dummy variables). We run the function on all the columns except the last, which is the outcome.

```
pp_hpc <- preProcess(schedulingData[, -8],
                     method = c("center", "scale", "YeoJohnson"))
pp_hpc
```

YeoJohnson:
与BoxCox相似但自变量可以是0或负数

```
## Created from 4331 samples and 7 variables
##
## Pre-processing:
##   - centered (5)
##   - ignored (2)
##   - scaled (5)
##   - Yeo-Johnson transformation (5)
##
## Lambda estimates for Yeo-Johnson transformation:
## -0.08, -0.03, -1.05, -1.1, 1.44
```

```
transformed <- predict(pp_hpc, newdata = schedulingData[, -8])
head(transformed)
```

去掉第八列

```
##   Protocol  Compounds InputFields Iterations NumPending
## 1        E  1.2289592  -0.6324580 -0.0615593  -0.554123  0.004
## 2        E -0.6065826  -0.8120473 -0.0615593  -0.554123 -0.043
## 3        E -0.5719534  -1.0131504 -2.7894869  -0.554123 -0.034
## 4        E -0.6427737  -1.0047277 -0.0615593  -0.554123 -0.964
## 5        E -0.5804713  -0.9564504 -0.0615593  -0.554123 -0.902
## 6        E -0.5804713  -0.9564504 -0.0615593  -0.554123  0.698
```

The two predictors labeled as “ignored” in the output are the two factor predictors. These are not altered but the numeric predictors are transformed. However, the predictor for the number of pending jobs,

has a very sparse and unbalanced distribution:

```
mean(schedulingData$NumPending == 0)
```

```
## [1] 0.7561764
```

For some other models, this might be an issue (especially if we resample or down-sample the data). We can add a filter to check for zero- or near zero-variance predictors prior to running the pre-processing calculations:

```
pp_no_nzv <- preProcess(schedulingData[, -8],
                        method = c("center", "scale", "YeoJohnson
pp_no_nzv
```

method=c ("center","scale","YeoJohnson","nzv")
nzv 相当于应用nearZeroVar，清除近零方差的自变量

```
## Created from 4331 samples and 7 variables
##
## Pre-processing:
##   - centered (4)
##   - ignored (2)
##   - removed (1)
##   - scaled (4)
##   - Yeo-Johnson transformation (4)
##
## Lambda estimates for Yeo-Johnson transformation:
## -0.08, -0.03, -1.05, 1.44
```

```
predict(pp_no_nzv, newdata = schedulingData[1:6, -8])
```

```
##   Protocol  Compounds InputFields Iterations         Hour Day
## 1        E  1.2289592  -0.6324580 -0.0615593  0.004586516 Tue
## 2        E -0.6065826  -0.8120473 -0.0615593 -0.043733201 Tue
## 3        E -0.5719534  -1.0131504 -2.7894869 -0.034967177 Thu
## 4        E -0.6427737  -1.0047277 -0.0615593 -0.964170752 Fri
## 5        E -0.5804713  -0.9564504 -0.0615593 -0.902085020 Fri
## 6        E -0.5804713  -0.9564504 -0.0615593  0.698108782 Wed
```

Note that one predictor is labeled as “removed” and the processed data lack the sparse predictor.

3.10 Class Distance Calculations

`caret` contains functions to generate new predictors variables based on distances to class centroids (similar to how linear discriminant analysis works). For each level of a factor variable, the class centroid and covariance matrix is calculated. For new samples, the Mahalanobis distance to each of the class centroids is computed and can be used as an additional predictor. This can be helpful for non-linear models when the true decision boundary is actually linear.

In cases where there are more predictors within a class than samples, the `classDist` function has arguments called `pca` and `keep` arguments that allow for principal components analysis within each class to be used to avoid issues with singular covariance matrices.

`predict.classDist` is then used to generate the class distances. By default, the distances are logged, but this can be changed via the `trans` argument to `predict.classDist` .

As an example, we can used the MDRR data.

```
centroids <- classDist(trainBC, trainMDRR)
distances <- predict(centroids, testBC)
distances <- as.data.frame(distances)
head(distances)
```

比较training的预测值（trainBC）与test预测值（testBC）的距离

classDist:
This function computes the class centroids and covariance matrix for a training set for determining Mahalanobis distances of samples to each class centroid.

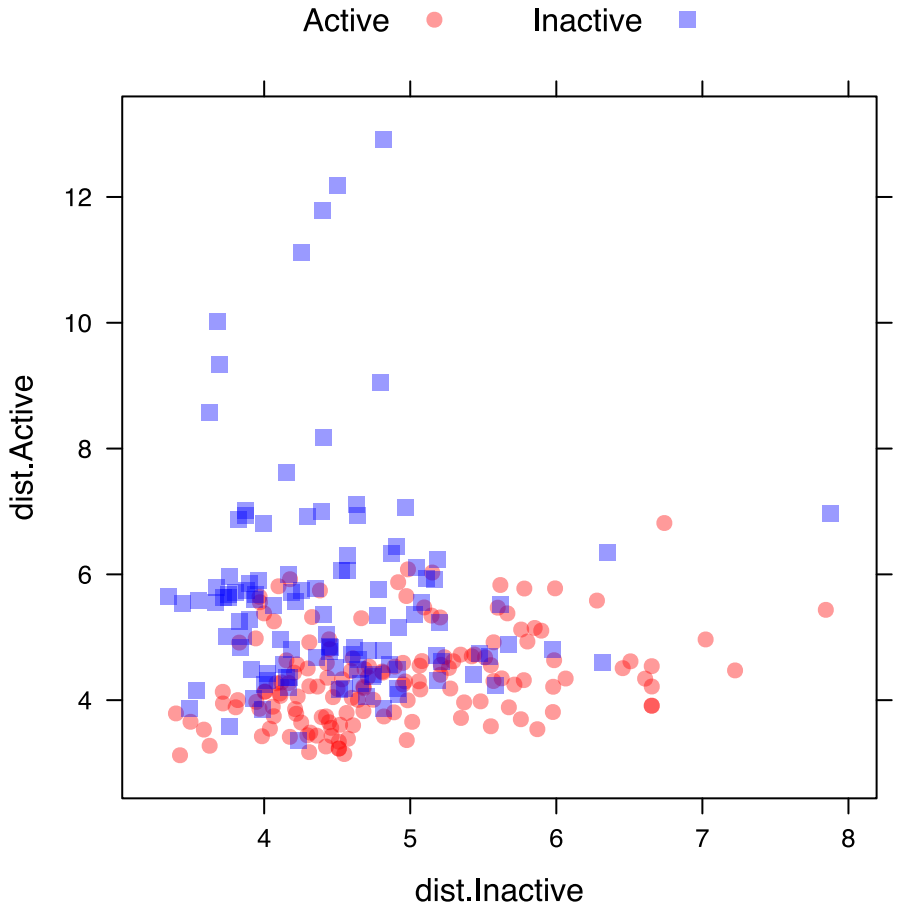
x——a matrix or data frame of predictor variables
y——a numeric or factor vector of class labels

这个heading是classDist加的吗

```
##                dist.Active dist.Inactive
## PROMETHAZINE      5.810607      4.098229
## ACEPROMETAZINE    4.272003      4.169292
## PYRATHIAZINE      4.570192      4.224053
## THIORIDAZINE      4.548315      5.064125
## MESORIDAZINE      4.621708      5.080362
## SULFORIDAZINE     5.344699      5.145311
```

This image shows a scatterplot matrix of the class distances for the held-out samples:

```
xyplot(dist.Active ~ dist.Inactive,
       data = distances,
       groups = testMDRR,
       auto.key = list(columns = 2))
```



这说明训练输出与测试输出距离大多比较近，predictor比较好

Ch3

1.删掉影响预测效果的数据
nearZeroVar（）
[,-nearZeroVar]
findCorrelation (cutoff=%)
findLinearCombos（）

2.dataset
training data
test data
training label
test label
split方法：sample（）
seq(along=row of training data)用作索引，length= 设划分测试集与训练集的比例

3.model<-preProcess(training data，method=有很多("center","scale"))

4.prediction
trainingtransformed<-predict(model,training)
testtransformed<-predict(model,test)

5.求差距
centroids<-classDist(trainingtransformed, label of training)
distances<-predict(centroids,testtransformed)
head(distances)

6.plot
data=distances
Y~x: dist.label1~.dist.label2
group=label of test