Machine Learning

Sébastien Plat

Contents

R Caret Package

Useful links: introducing caret, r-project, tutorials, vignette, Model training and tuning, ggplot2 tutorial, caret visualizations, preprocessing with caret, Elements of Statistical Learning, Elements of statistical learning, Modern applied statistics with S, Introduction to statistical learning,

Caret functionality

- Some preprocessing (cleaning)
- preProcess
- Data splitting
- createDataPartition
- createResample
- createTimeSlices
- Training/testing functions
- train
- predict
- Model comparison
- confusionMatrix

Machine learning algorithms in R

- Linear discriminant analysis
- Regression
- Naive Bayes
- Support vector machines
- Classification and regression trees
- Random forests
- Boosting
- etc.

Plotting (Wages example)

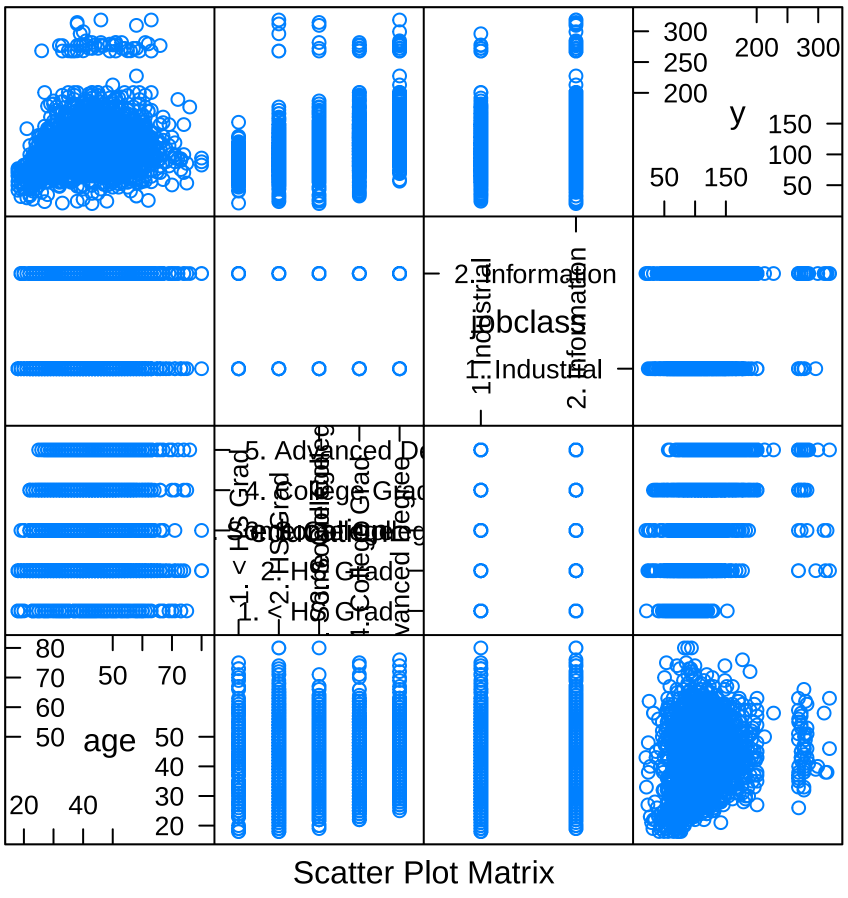
The test set should never be used for exploration: plots should focus on the training set only. Things of interest are:

- Imbalance in outcomes/predictors
- Outliers / Groups of points not explained by a predictor
- Skewed variables

```r
# Get training/test sets (70% of samples in the training set)
inTrain <- createDataPartition(y=Wage$wage, p=0.7, list=FALSE)
training <- Wage[inTrain,]
testing <- Wage[-inTrain,]
```

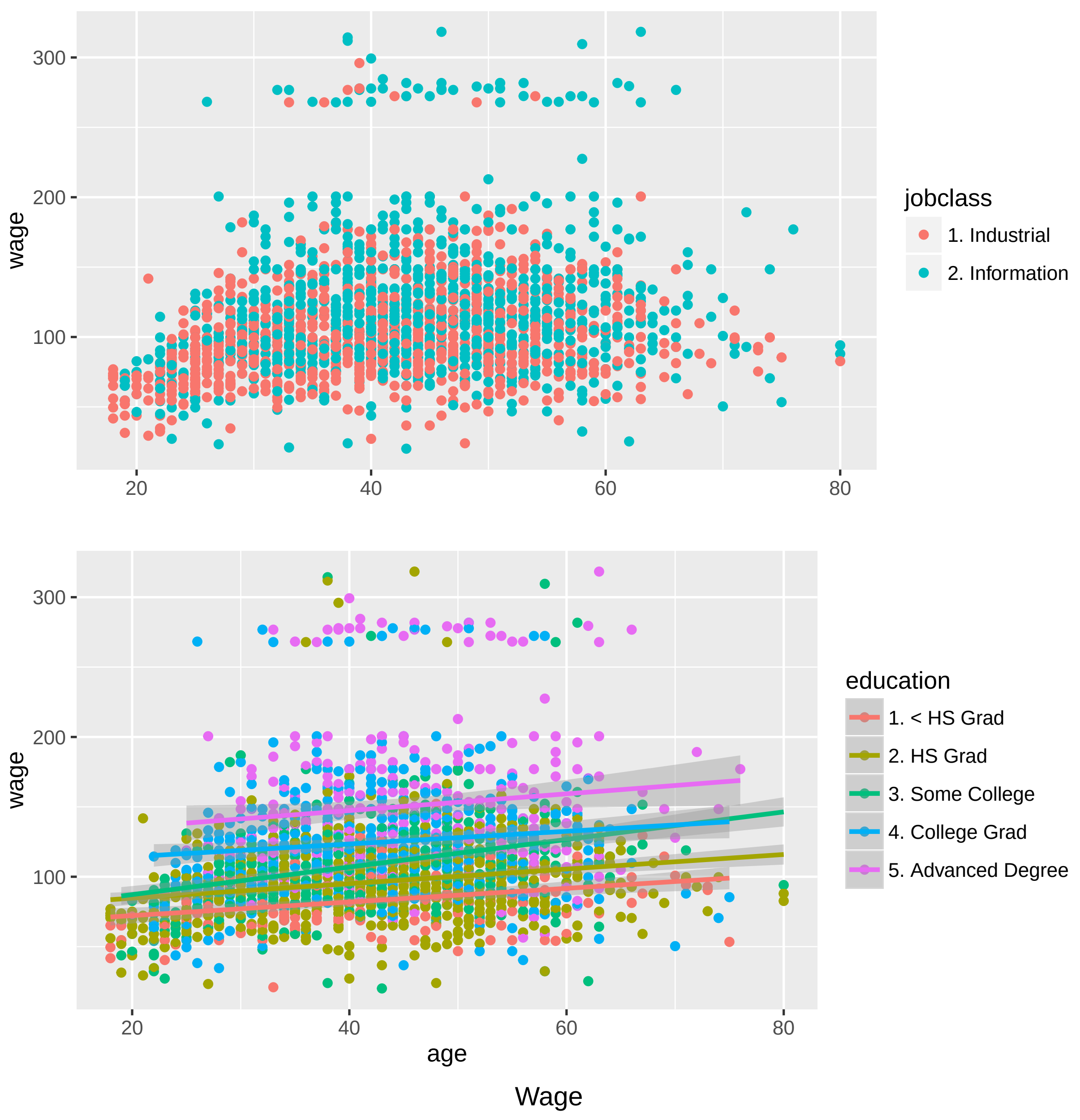
Feature plot

```r
# Feature plot (*caret* package)
featurePlot(x=training[,c("age","education","jobclass")],
            y = training$wage,
            plot="pairs")
```



Cloud of points

```r
# Qplot with color (*ggplot2* package)
qq1 <- qplot(age,wage,colour=jobclass,data=training) + xlab("")

# Add regression smoothers (*ggplot2* package)
qq2 <- qplot(age,wage,colour=education,data=training)
qq2 <- qq2 + geom_point(size=1) + geom_smooth(method='lm',formula=y~x)

grid.arrange(qq1, qq2, nrow=2, bottom="Wage")
```



Boxplot

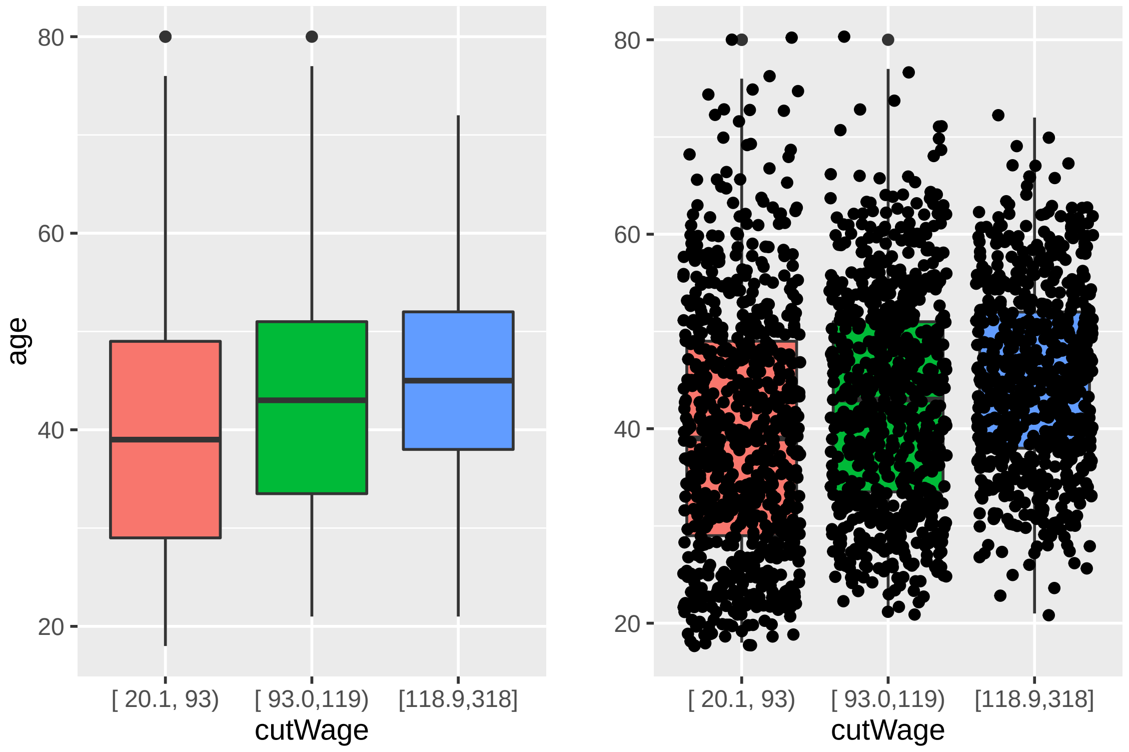
Using the `cut2` function of package Hmisc is very useful to create factors from continuous variables:

```r
# cut2, making factors (*Hmisc* package)
cutWage <- cut2(training$wage,g=3)
t1 <- table(cutWage,training$jobclass)

pander(cbind(Total=table(cutWage), t1, prop.table(t1,1)), split.table=Inf, digits=3,
       caption="Wage groups")
```

Table 1: Wage groups

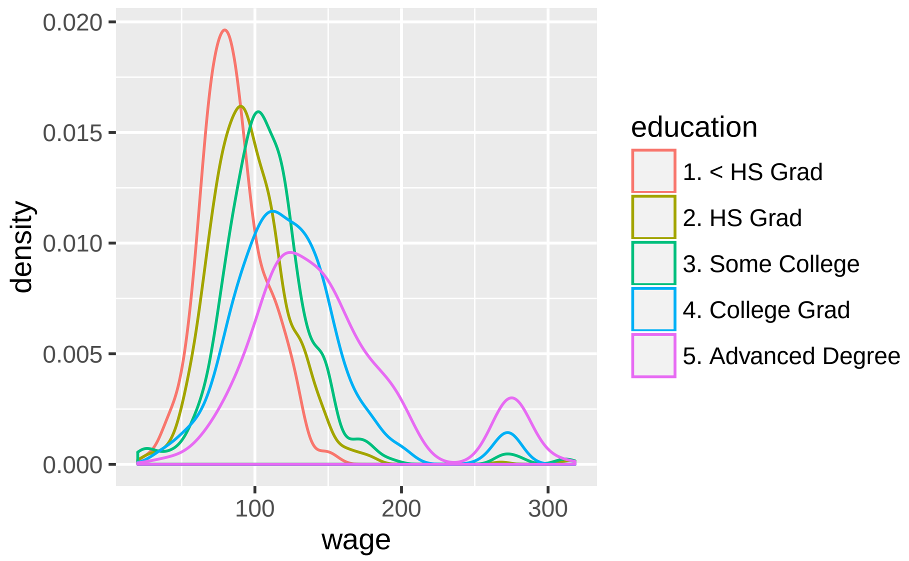
	Total	1. Industrial	2. Information	1. Industrial	2. Information
[20.1, 93)	719	449	270	0.624	0.376
[93.0,119)	711	371	340	0.522	0.478
[118.9,318]	672	279	393	0.415	0.585

```r
# Boxplots with points overlayed
p1 <- qplot(cutWage,age, data=training,fill=cutWage,
      geom=c("boxplot")) + theme(legend.position="none")

p2 <- qplot(cutWage,age, data=training,fill=cutWage,
      geom=c("boxplot","jitter")) + theme(legend.position="none") + ylab("")

grid.arrange(p1,p2,ncol=2)
```



Density Plot

```
# Density plots
qplot(wage,colour=education,data=training,geom="density")
```



Data splitting (Spam example)

Note: the spam dataset has 4601 observations.

Training vs Test set

```
# data splitting - 75% of obsevations in the training set
set.seed(3323)
inTrain <- createDataPartition(y=spam$type, p=0.75, list=FALSE)

training <- spam[inTrain,]
testing <- spam[-inTrain,]
```

Cross Validation - k folds

Cross-validation is used to estimate how the model fit a data set not used to train the model, but without using our test set (only used to test the accuracy of the final model). More information on Wikipedia.

In k-folds CV, the training set is divided in k folds of equal sizes. The model is then applied k times, using k-1 folds for training the remaining one for validation. The k results can then be averaged (or otherwise combined) to produce a single estimation.

In the SPAM example, each training set will have $0.9 \times 4601 \simeq 4140$ observations and each test set $0.1 \times 4601 \simeq 460$.

```
# k-fold - for cross validation purpose
set.seed(32323)
foldsT <- createFolds(y=spam$type, k=10, list=TRUE, returnTrain=TRUE) #10x 4140 obs.
foldsF <- createFolds(y=spam$type, k=10, list=TRUE, returnTrain=FALSE) #10x 460 obs.
```

Cross Validation - Bootstrap resampling

```
# bootstrap resampling - 10 resamples of 4601 observations with replacement
set.seed(32323)
folds <- createResample(y=spam$type, times=10, list=TRUE)
```

Time slices

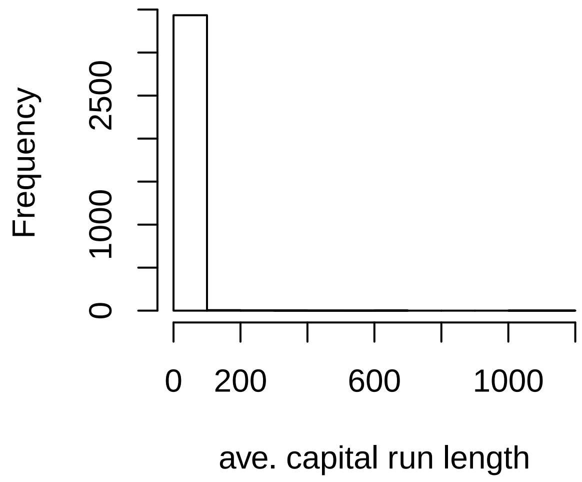
```
# time slices
set.seed(32323)
tme <- 1:1000
folds <- createTimeSlices(y=tme, initialWindow=20, horizon=10)
# returns folds$train & folds$testing
```

Folds	r1	r2	r3
Training	1:20	2:21	3:22
Test	21:30	22:31	23:32

Preprocessing (Spam example)

Note: it is very important to apply the ***proprocessing parameters of the training set*** *to the test set.*

```
hist(training$capitalAve,main="",xlab="ave. capital run length")
```



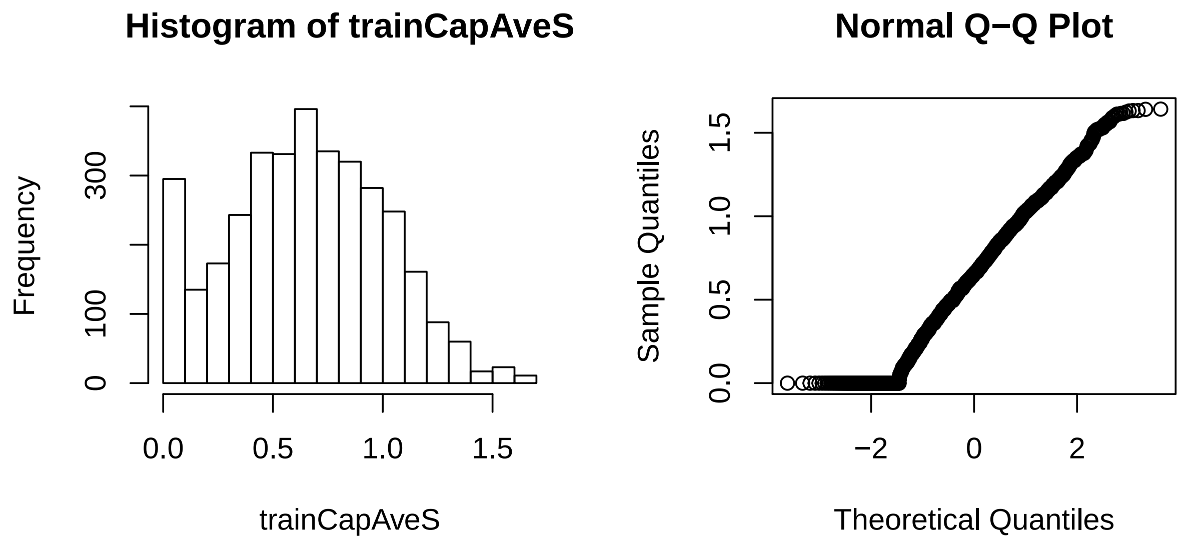
Standardizing

```
# standardizing w/ caret preProcess function (col 58 is our outcome)
set.seed(32323)
preObj <- preProcess(training[,-58],method=c("center","scale"))
trainCapAveS <- predict(preObj,training[,-58])$capitalAve
testCapAveS <- predict(preObj,testing[,-58])$capitalAve
```

set	mean	sd
training	5.48	35.32
trainingStd	0	1
testStd	-0.03	0.48

Standardizing - Box-Cox transforms

```
preObj <- preProcess(training[,-58],method=c("BoxCox"))
trainCapAveS <- predict(preObj,training[,-58])$capitalAve
par(mfrow=c(1,2)); hist(trainCapAveS); qqnorm(trainCapAveS)
```



Standardizing - Imputing missing data

```
set.seed(32323)
# Make some values NA
training$capAve <- training$capitalAve
selectNA <- rbinom(dim(training)[1],size=1,prob=0.05)==1
training$capAve[selectNA] <- NA
```

```
# Impute and standardize
preObj <- preProcess(training[,-58],method="knnImpute")
capAve <- predict(preObj,training[,-58])$capAve

# Standardize true values
capAveTruth <- training$capitalAve
capAveTruth <- (capAveTruth-mean(capAveTruth))/sd(capAveTruth)
```

data	0%	25%	50%	75%	100%
all	-1.6454	-0.0016	-5e-04	1e-04	0.2726
NA	-1.6454	-0.0133	-4e-04	0.0148	0.2726
!NA	-0.8101	-0.0015	-5e-04	0	5e-04

Preprocessing with PCA (Spam example)

Definition

Principal Component Analysis converts observations of possibly correlated variables into values of linearly uncorrelated variables called **principal components** (Wikipedia).

- number of principal components $\leq$ number of original variables
- principal components are ordered in descending variance

Benefits:

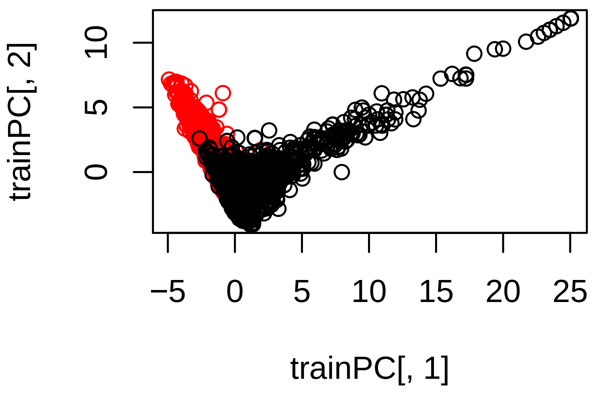
- Reduced number of predictors (data compression)
- Uncorrelated variables that explain much of the variance (statistical gain)
- Reduced noise (due to averaging)

Limitations:

- Most useful for linear-type models
- Can make it harder to interpret predictors
- Outliers can be misleading
  - Transform first (with logs/Box Cox)
  - Plot predictors to identify problems

Spam example

```
# pca pre-processing - only two pca Components (possible use of ..., thresh=desVar)
preProc <- preProcess(log10(training[,-58]+1),method="pca",pcaComp=2)
trainPC <- predict(preProc,log10(training[,-58]+1))

typeColor <- ((training$type=="spam")*1 + 1)
plot(trainPC[,1],trainPC[,2],col=typeColor)
```



Model with PCA

```r
# we train the model on the PCA data
modelFit <- train(training$type ~ .,method="glm",data=trainPC)

# we appy the PCA to the test set
testPC <- predict(preProc,log10(testing[,-58]+1))

# we check the accuracy of the model on the test set
cmFit <- confusionMatrix(testing$type,predict(modelFit,testPC))
```

	nonspam	spam
nonspam	650	47
spam	67	386

Accuracy	AccuracyLower	AccuracyUpper
0.9009	0.8821	0.9175

Model without PCA

	nonspam	spam
nonspam	661	36
spam	48	405

Accuracy	AccuracyLower	AccuracyUpper
0.927	0.9104	0.9413

Covariate creation

Creating covariates from raw data depend heavily on the application. Googling "feature extraction for [data type]" helps finding existing methods for a large variety of topics.

A few examples of covariates:

- **Text files**: frequency of words, frequency of phrases (Google ngrams)or capital letters
- **Images**: Edges, corners, blobs, ridges (computer vision feature detection)
- **Webpages**: Number and type of images, position of elements, colors, videos (A/B Testing)
- **People**: Height, weight, hair color, sex, country of origin.

In some applications (images, voices) automated feature creation is possible/necessary.

It is possible to transform existing covariates into new ones, based on exploratory analysis for example (that should be done *only on the training set*). Caution is advised though, as it could lead to overfitting.

Note: It is more useful for some methods (regression, svms) than for others (classification trees).

Dummy variables

The basic idea is to **convert factor variables to indicator variables**.

```r
dummies <- dummyVars(wage ~ jobclass,data=training)
predict(dummies,newdata=training)
```

Removing zero covariates

Some covariates bring nothing to the model, either because they have one unique value or because their most common value is much more common than the second one (ie. almost unique value). They can be dropped without impacting the results.

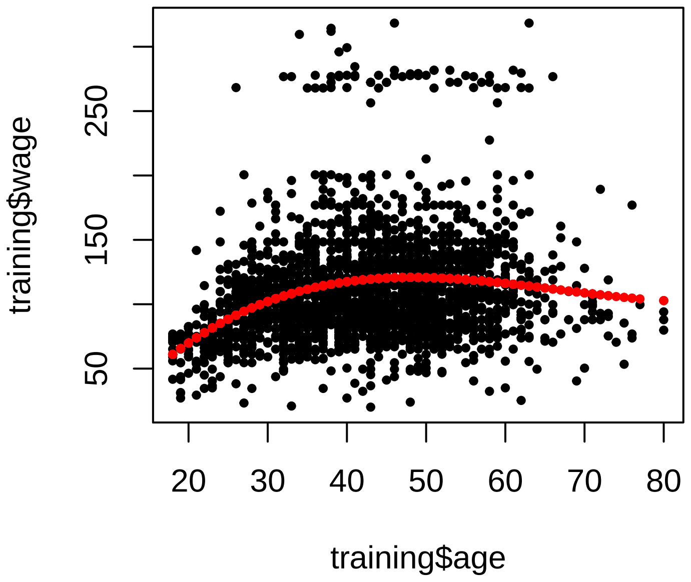
```r
# returns the positions of the zero- or near-zero predictors
nsv <- nearZeroVar(training)
```

Polynomial covariates (B-Spline fit)

Creates new polynomial covariates, which can be used to fit a polynomial linear model to the data.

See also: ns(),poly()

```r
library(splines)

bsBasis <- bs(training$age,df=3)
lm1 <- lm(wage ~ bsBasis,data=training)

plot(training$age,training$wage,pch=19,cex=0.5)
points(training$age,predict(lm1,newdata=training),col="red",pch=19,cex=0.5)
```



```r
predict(bsBasis,age=testing$age)
```

Model Fit (Wages example)

```
[1] 2102   11
```

```
[1] 898  11
```

Train options

```r
args(train.default)
```

metric

Continous outcomes:

- *RMSE* = Root mean squared error
- *RSquared* = R^2 from regression models

Categorical outcomes:

- *Accuracy* = Fraction correct
- *Kappa* = A measure of concordance

trainControl

```r
args(trainControl)
```

resampling method:

- *boot* = bootstrapping
- *boot632* = bootstrapping with adjustment
- *cv* = cross validation
- *repeatedcv* = repeated cross validation
- *LOOCV* = leave one out cross validation

number:

- For boot/cross validation
- Either the number of folds or number of resampling iterations

repeats:

- Number of times to repeate subsampling (repeated k-fold CV only)
- If big this can *slow things down*