Breast Cancer Prediction with ML II

CS-715

Howard Nguyen

22 April, 2023

Contents

0.1 Load libraries and dataset

The data was collected from University of Wisconsin Hospitals. Below is a summary of the attributes taken from the UCI Machine Learning repository. - Sample code number id number. - Clump Thickness. - Uniformity of Cell Size. - Uniformity of Cell Shape. - Marginal Adhesion. - Single Epithelial Cell Size. - Bare Nuclei. - Bland Chromatin. - Normal Nucleoli. - Mitoses. - Class. Although the test methodologies differ, the best published results appear to be in the high 90% accuracy such as 96% and 97%. Achieving results in this range would be desirable in this case study.

```
# load packages
library(mlbench)
library(caret)
library(corrplot)
# Load data
data(BreastCancer)
```

0.2 Validation dataset

```
# Split out validation dataset
# create a list of 80% of the rows in the original dataset we can use for training
set.seed(7)
validationIndex <- createDataPartition(BreastCancer$Class, p=0.80, list=FALSE)
# select 20% of the data for validation
validation <- BreastCancer[-validationIndex,]
# use the remaining 80% of data to training and testing the models
dataset <- BreastCancer[validationIndex,]
```

0.3 Explore the dataset

```
dim(dataset)
```

```
[1] 560  11
```

```
head(dataset, n = 10)
```

```
        Id Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
1  1000025            5         1          1             1            2
2  1002945            5         4          4             5            7
3  1015425            3         1          1             1            2
5  1017023            4         1          1             3            2
6  1017122            8        10         10             8            7
7  1018099            1         1          1             1            2
8  1018561            2         1          2             1            2
9  1033078            2         1          1             1            2
10 1033078            4         2          1             1            2
11 1035283            1         1          1             1            1
   Bare.nuclei Bl.cromatin Normal.nucleoli Mitoses     Class
```

```
1            1           3               1       1    benign
2           10           3               2       1    benign
3            2           3               1       1    benign
5            1           3               1       1    benign
6           10           9               7       1 malignant
7           10           3               1       1    benign
8            1           3               1       1    benign
9            1           1               1       5    benign
10           1           2               1       1    benign
11           1           3               1       1    benign
```

```
# Types
sapply(dataset, class)
```

```
$Id
[1] "character"

$Cl.thickness
[1] "ordered" "factor"

$Cell.size
[1] "ordered" "factor"

$Cell.shape
[1] "ordered" "factor"

$Marg.adhesion
[1] "ordered" "factor"

$Epith.c.size
[1] "ordered" "factor"

$Bare.nuclei
[1] "factor"

$Bl.cromatin
[1] "factor"

$Normal.nucleoli
[1] "factor"

$Mitoses
[1] "factor"

$Class
[1] "factor"
```

```
# Remove redundant variable Id
dataset <- dataset[,-1]
# convert input values to numeric
for(i in 1:9) {
dataset[,i] <- as.numeric(as.character(dataset[,i]))
}
```

```
summary(dataset)
```

```
 Cl.thickness      Cell.size        Cell.shape     Marg.adhesion
Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.: 1.000
Median : 4.000   Median : 1.000   Median : 1.000   Median : 1.000
Mean   : 4.438   Mean   : 3.145   Mean   : 3.211   Mean   : 2.788
3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000   3rd Qu.: 4.000
Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000

 Epith.c.size     Bare.nuclei      Bl.cromatin     Normal.nucleoli
Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 2.000   1st Qu.: 1.000
Median : 2.000   Median : 1.000   Median : 3.000   Median : 1.000
Mean   : 3.234   Mean   : 3.484   Mean   : 3.446   Mean   : 2.941
3rd Qu.: 4.000   3rd Qu.: 5.250   3rd Qu.: 5.000   3rd Qu.: 4.000
Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000
                 NA's   :12
    Mitoses            Class
Min.   : 1.000   benign   :367
1st Qu.: 1.000   malignant:193
Median : 1.000
Mean   : 1.587
3rd Qu.: 1.000
Max.   :10.000
```

Interestingly, we can see we have 12 NA values for the Bare.nuclei attribute. This suggests we may need to remove the records (or impute values) with NA values for some analysis and modeling techniques. We can also see that all attributes have integer values in the range [1,10]. This suggests that we may not see much benefit form normalizing attributes for instance based methods like KNN.

```
# class distribution
cbind(freq=table(dataset$Class), percentage=prop.table(table(dataset$Class))*100)
```

```
          freq percentage
benign     367   65.53571
malignant  193   34.46429
```

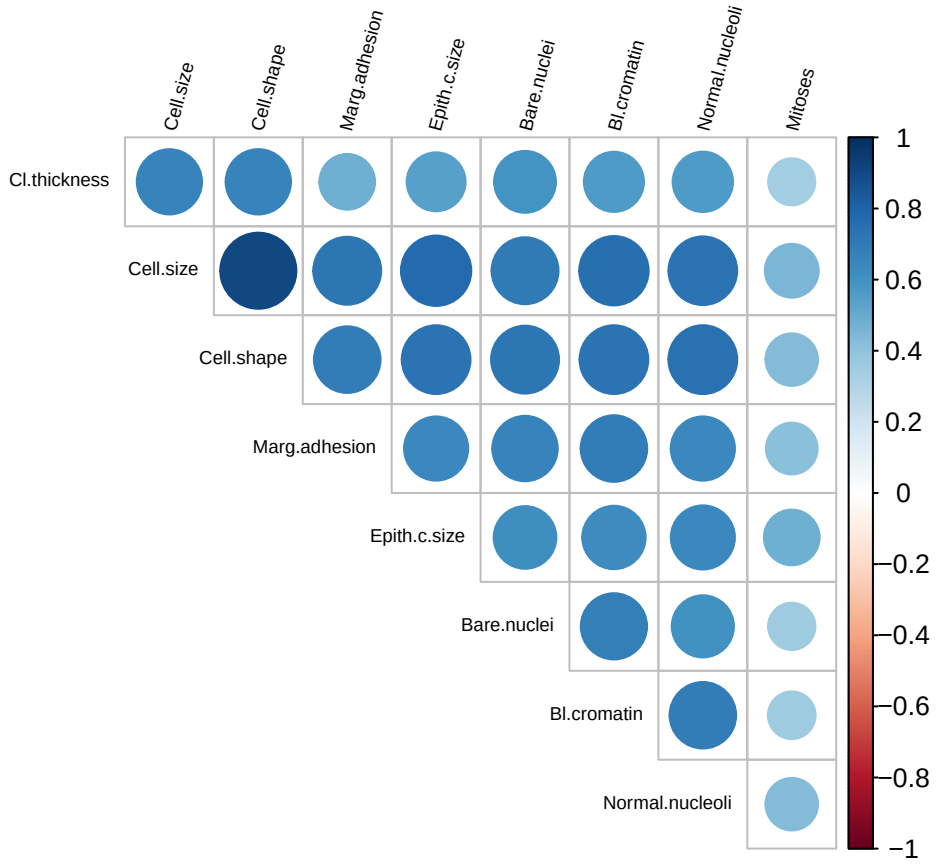
```
# summarize correlations between input variables
complete_cases <- complete.cases(dataset)
cor(dataset[complete_cases,1:9])
```

```
               Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
Cl.thickness       1.0000000 0.6647248  0.6673745     0.4858858    0.5426552
Cell.size          0.6647248 1.0000000  0.9041943     0.7221120    0.7730926
Cell.shape         0.6673745 0.9041943  1.0000000     0.6939359    0.7385534
Marg.adhesion      0.4858858 0.7221120  0.6939359     1.0000000    0.6433283
Epith.c.size       0.5426552 0.7730926  0.7385534     0.6433283    1.0000000
Bare.nuclei        0.5978581 0.7000242  0.7212405     0.6687585    0.6141434
Bl.cromatin        0.5653494 0.7523279  0.7392279     0.6921188    0.6284302
Normal.nucleoli    0.5695270 0.7369191  0.7405848     0.6440775    0.6423265
```

```
Mitoses            0.3474603 0.4534288  0.4318809     0.4192666    0.4850180
                Bare.nuclei Bl.cromatin Normal.nucleoli   Mitoses
Cl.thickness      0.5978581   0.5653494       0.5695270 0.3474603
Cell.size         0.7000242   0.7523279       0.7369191 0.4534288
Cell.shape        0.7212405   0.7392279       0.7405848 0.4318809
Marg.adhesion     0.6687585   0.6921188       0.6440775 0.4192666
Epith.c.size      0.6141434   0.6284302       0.6423265 0.4850180
Bare.nuclei       1.0000000   0.6852433       0.6048941 0.3509201
Bl.cromatin       0.6852433   1.0000000       0.6915619 0.3557886
Normal.nucleoli   0.6048941   0.6915619       1.0000000 0.4316606
Mitoses           0.3509201   0.3557886       0.4316606 1.0000000
```

We can see some modest to high correlation between some of the attributes. For example between cell shape and cell size at 0.90 correlation. Some algorithms may benefit from removing the highly correlated attributes.

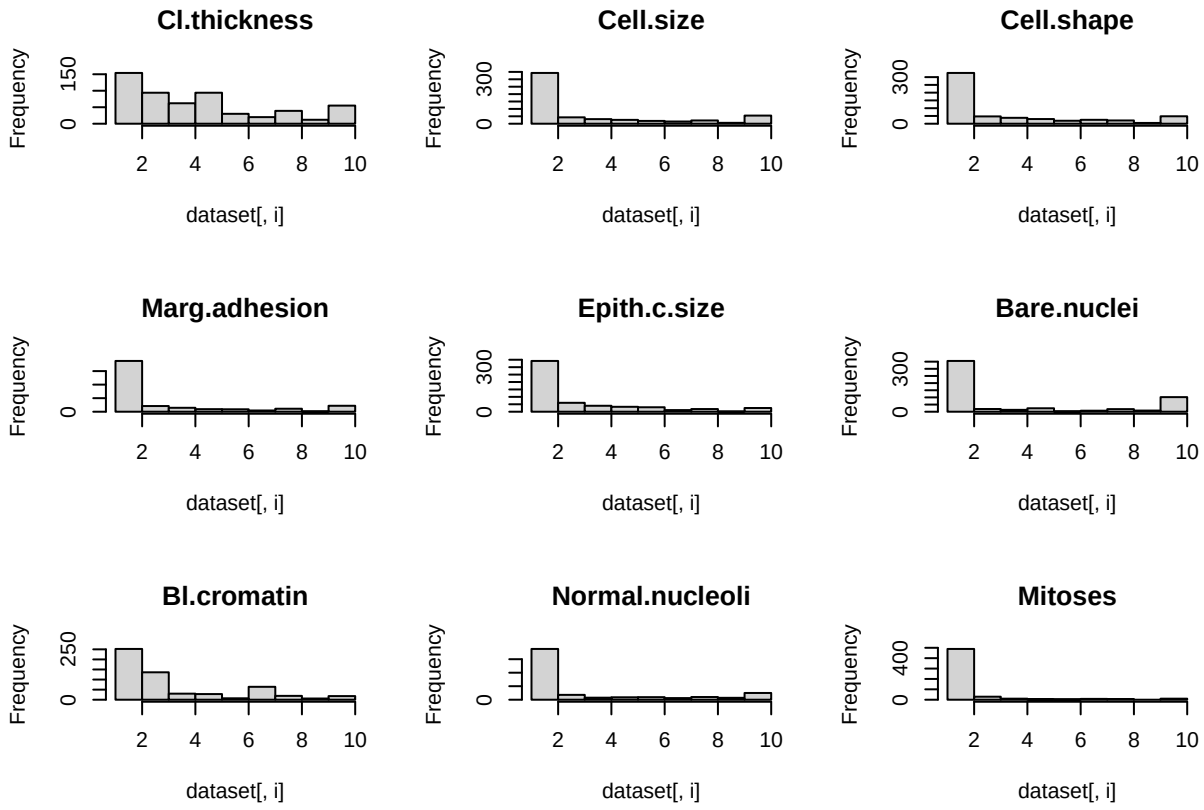
```
# visualized correlations between attributes
dataset_noNA <- na.omit(dataset)
correlationMatrix <- cor(dataset_noNA[,1:9])
corrplot(correlationMatrix,
         order = "original", # order for labels, can be "hclust"
         type = "upper",   # matrix: full, upper, lower
         diag = F,      # remove diagonal
         tl.cex = 0.6,  # font size
         tl.srt = 75, # label angel
         tl.col = "black",
         addrect = 8)
```



0.4 Unimodal Data Visualizations

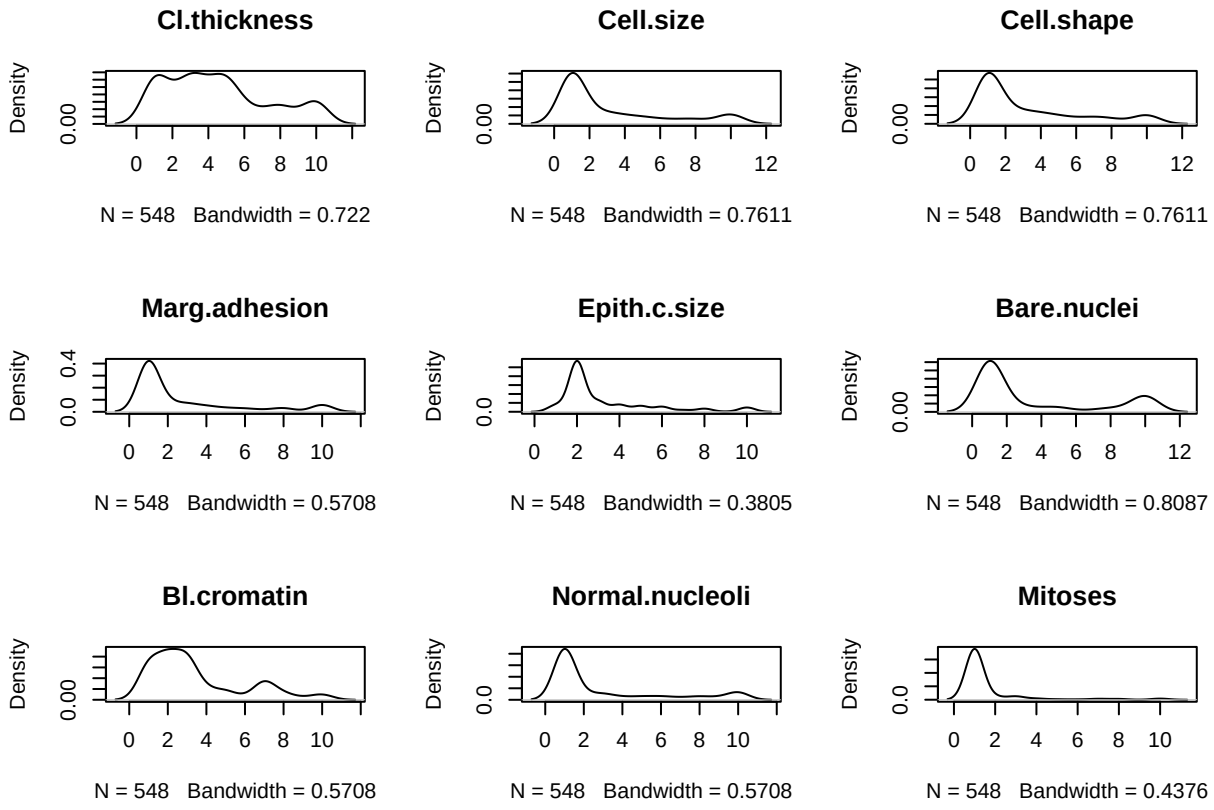
Let's look at the distribution of individual attributes in the dataset. We'll start with histograms of all of the attributes.

```
# histograms each attribute
par(mfrow=c(3,3))
for(i in 1:9) {
hist(dataset[,i], main=names(dataset)[i])
}
```



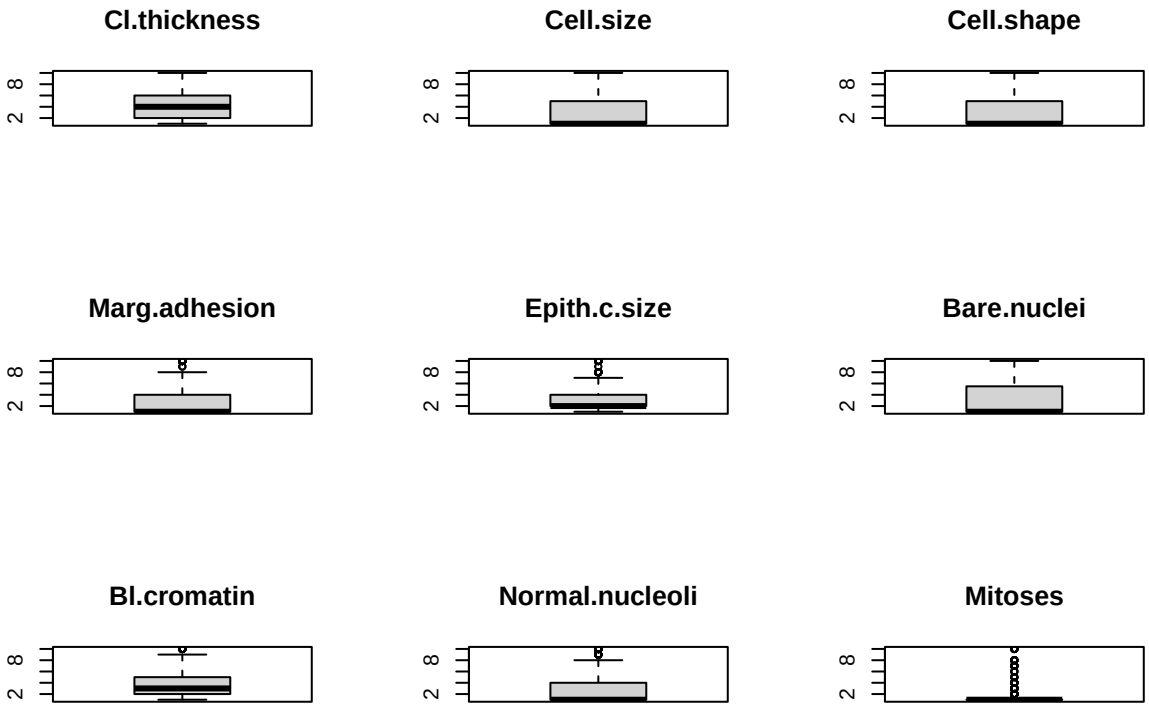
We can see that almost all of the distributions have an exponential or bimodal shape to them. We may benefit from log transforms or other power transforms later on. Let's use density plots to get a more smoothed look at the distributions.

```
# density plot for each attribute
par(mfrow=c(3,3))
complete_cases <- complete.cases(dataset)
for(i in 1:9) {
plot(density(dataset[complete_cases,i]), main=names(dataset)[i])
}
```



These plots add more support to our initial ideas. We can see bimodal distributions (two bumps) and exponential looking distributions.

```
# boxplots for each attribute
par(mfrow=c(3,3))
for(i in 1:9) {
boxplot(dataset[,i], main=names(dataset)[i])
}
```



We see squashed distributions given the exponential shapes we've already observed. Also, because the attributes are scoring of some kind, the scale is limited to [1,10] for all inputs.

0.5 Multimodal Data Visualizations

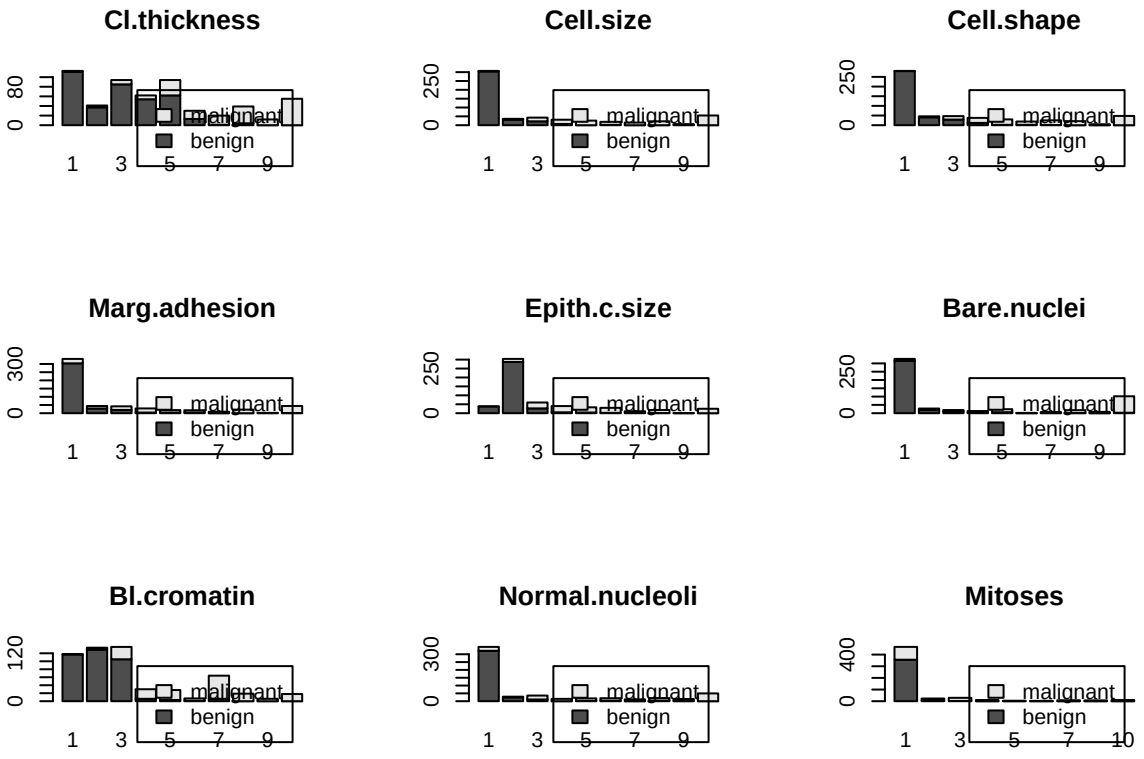
Now, let's take a look at the interactions between the attributes. Let's start with a scatter plot matrix of the attributes colored by the class values. Because the data is discrete (integer values) we need to add some jitters to make the scatter plot useful, otherwise the dots will all be on top of each other.

```
# scatter plot matrix
jittered_x <- sapply(dataset[,1:9], jitter)
pairs(jittered_x, names(dataset[,1:9]), col=dataset$Class)
```

We can see that the black (benign) apart to be clustered around the bottom-right corner (smaller values) and red (malignant) are all over the place.

Because the data is discrete, we can use bar plots to get an idea of the interaction of the distribution of each attribute and how they breakdown by class value.

```
# bar plots of each variable by class
par(mfrow=c(3,3))
for(i in 1:9) {
  barplot(table(dataset$Class,dataset[,i]), main=names(dataset)[i],
          legend.text=unique(dataset$Class))
}
```



This gives us a more nuanced idea of how the benign values clustered at the left (smaller values) of each distribution and malignant all over the place.

0.6 Evaluate Algorithms - Baseline

- Linear Algorithms: Logistic Regression (LG), Linear Discriminate Analysis (LDA) and Regularized Logistic Regression (GLMNET).
- Non-Linear Algorithms: k-Nearest Neighbors (KNN), Classification and Regression Trees (CART), Naive Bayes (NB) and Support Vector Machines with Radial Basis Functions (SVM).

```
library(caret)
set.seed(998)
inTraining <- createDataPartition(dataset$Class, p = .80, list = FALSE)
training <- dataset[ inTraining,]
testing <- dataset[ - inTraining,]
```

```
# 10-fold cross validation with 3 repeats
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
```

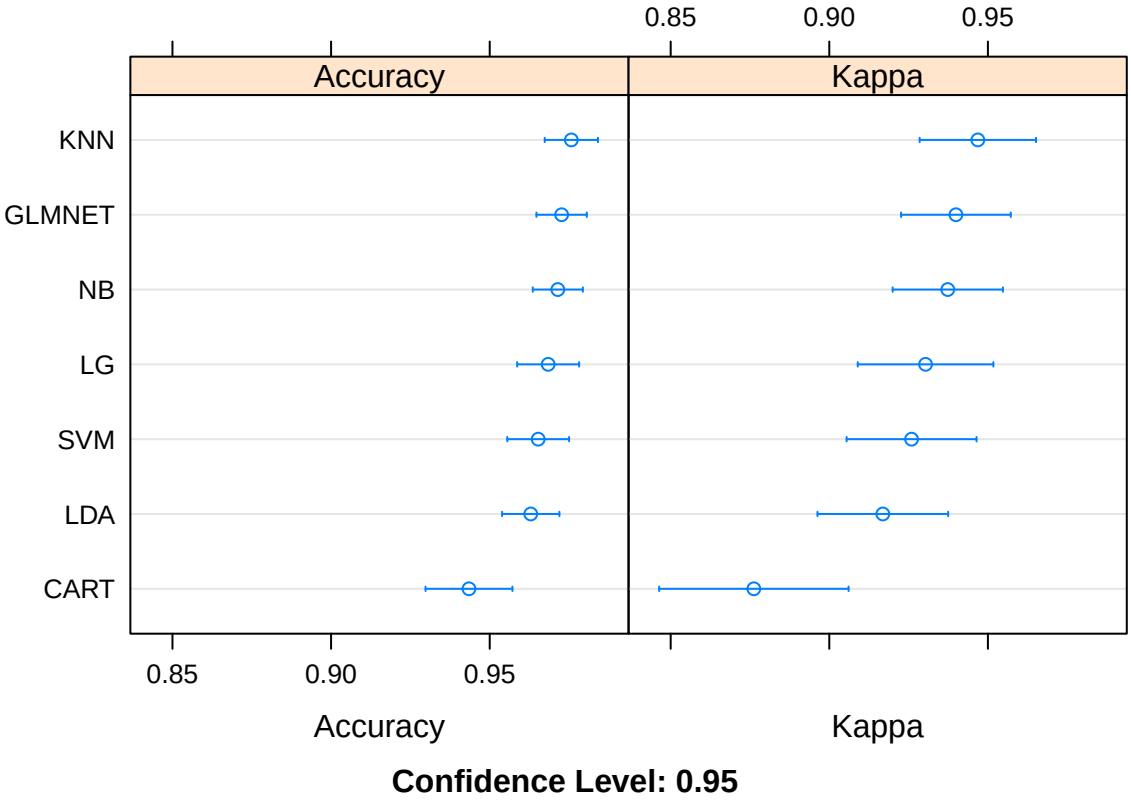
```
# View(trainControl)
# library(nlme)
# str(dataset)
```

```
set.seed(7)
fit.glm <- train(Class ~ ., data = dataset, method = "glm", metric = metric,
                 trControl = trainControl, na.action = na.omit)

# LDA
set.seed(7)
fit.lda <- train(Class~., data=dataset, method="lda", metric=metric,
                 trControl=trainControl, na.action = na.omit)
# GLMNET
set.seed(7)
fit.glmnet <- train(Class~., data=dataset, method="glmnet", metric=metric,
                    trControl=trainControl, na.action = na.omit)
# KNN
set.seed(7)
fit.knn <- train(Class~., data=dataset, method="knn", metric=metric,
                 trControl=trainControl, na.action = na.omit)
# CART
set.seed(7)
fit.cart <- train(Class~., data=dataset, method="rpart", metric=metric,
                  trControl=trainControl, na.action = na.omit)
# Naive Bayes
set.seed(7)
fit.nb <- train(Class~., data=dataset, method="nb", metric=metric,
                trControl=trainControl, na.action = na.omit)
# SVM
set.seed(7)
fit.svm <- train(Class~., data=dataset, method="svmRadial", metric=metric,
                 trControl=trainControl, na.action = na.omit)
# Compare algorithms
results <- resamples(list(LG=fit.glm, LDA=fit.lda, GLMNET=fit.glmnet, KNN=fit.knn,
CART=fit.cart, NB=fit.nb, SVM=fit.svm))
summary(results)
```

```
Call:
summary.resamples(object = results)

Models: LG, LDA, GLMNET, KNN, CART, NB, SVM
Number of resamples: 30

Accuracy
            Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
LG     0.9090909 0.9454545 0.9636364 0.9684259 0.9954545    1    0
LDA    0.9074074 0.9454545 0.9636364 0.9629381 0.9817340    1    0
GLMNET 0.9272727 0.9636364 0.9639610 0.9726692 0.9954545    1    0
KNN    0.9272727 0.9636364 0.9816498 0.9757323 1.0000000    1    0
CART   0.8333333 0.9272727 0.9454545 0.9434636 0.9636364    1    0
NB     0.9272727 0.9631313 0.9814815 0.9714679 0.9818182    1    0
SVM    0.9074074 0.9454545 0.9636364 0.9652834 0.9818182    1    0

Kappa
            Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
LG     0.8062016 0.8799491 0.9205096 0.9303463 0.9900722    1    0
```

```
LDA    0.7856586 0.8799491 0.9181691 0.9168812 0.9591931    1    0
GLMNET 0.8430813 0.9175412 0.9218254 0.9399200 0.9900722    1    0
KNN    0.8430813 0.9198692 0.9592894 0.9468245 1.0000000    1    0
CART   0.6301370 0.8401563 0.8787558 0.8762248 0.9195906    1    0
NB     0.8350825 0.9180536 0.9590968 0.9373496 0.9602888    1    0
SVM    0.8040639 0.8827590 0.9215407 0.9259359 0.9602888    1    0
```

```
dotplot(results)
```



We can see good accuracy across the board. All algorithms have a mean accuracy above 90%, well above the baseline of 65% if we just predicted benign. The problem is learnable. We can see that KNN (97.57%) and Naive Bayes (NB was 97.14% and GLMNET was 97.26%) had the highest accuracy on the problem.

0.7 Evaluate Algorithms: Transform

We know we have some skewed distributions. There are transform methods that we can use to adjust and normalize these distributions. A favorite for positive input attributes (which we have in this case) is the Box-Cox transform. In this section we evaluate the same 7 algorithms as above except this time the data is transformed using a Box-Cox power transform to flatten out the distributions.

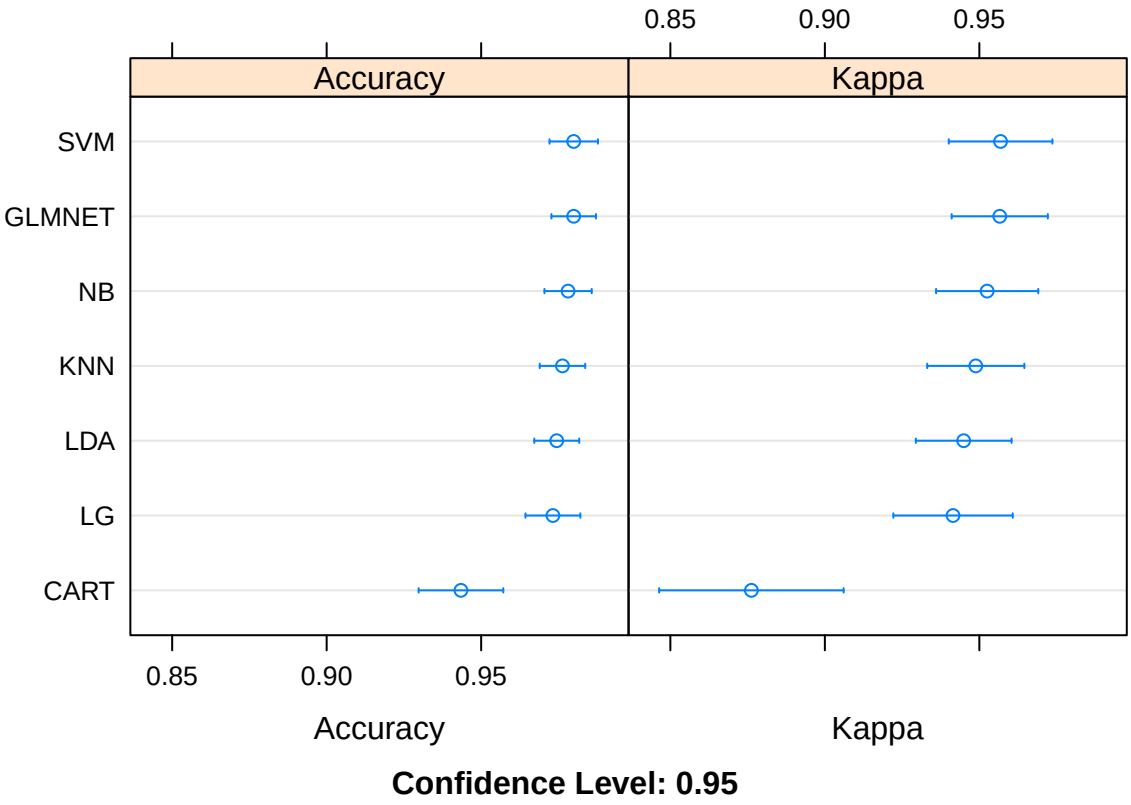
```
# 10-fold cross validation with 3 repeats
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
# LG
set.seed(7)
```

```
fit.glm <- train(Class~., data=dataset, method="glm", metric=metric, preProc=c("BoxCox"),
trControl=trainControl, na.action = na.omit)
# LDA
set.seed(7)
fit.lda <- train(Class~., data=dataset, method="lda", metric=metric, preProc=c("BoxCox"),
trControl=trainControl, na.action = na.omit)
# GLMNET
set.seed(7)
fit.glmnet <- train(Class~., data=dataset, method="glmnet", metric=metric,
preProc=c("BoxCox"), trControl=trainControl, na.action = na.omit)
# KNN
set.seed(7)
fit.knn <- train(Class~., data=dataset, method="knn", metric=metric, preProc=c("BoxCox"),
trControl=trainControl, na.action = na.omit)
# CART
set.seed(7)
fit.cart <- train(Class~., data=dataset, method="rpart", metric=metric,
preProc=c("BoxCox"), trControl=trainControl, na.action = na.omit)
# Naive Bayes
set.seed(7)
fit.nb <- train(Class~., data=dataset, method="nb", metric=metric, preProc=c("BoxCox"),
trControl=trainControl, na.action = na.omit)
# SVM
set.seed(7)
fit.svm <- train(Class~., data=dataset, method="svmRadial", metric=metric,
preProc=c("BoxCox"), trControl=trainControl, na.action = na.omit)
# Compare algorithms
transformResults <- resamples(list(LG=fit.glm, LDA=fit.lda, GLMNET=fit.glmnet, KNN=fit.knn,
CART=fit.cart, NB=fit.nb, SVM=fit.svm))
summary(transformResults)
```

```
Call:
summary.resamples(object = transformResults)

Models: LG, LDA, GLMNET, KNN, CART, NB, SVM
Number of resamples: 30

Accuracy
            Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
LG     0.9090909 0.9629630 0.9816498 0.9732307 0.9955357    1    0
LDA    0.9272727 0.9636364 0.9814815 0.9744649 0.9818182    1    0
GLMNET 0.9444444 0.9636364 0.9818182 0.9799531 1.0000000    1    0
KNN    0.9090909 0.9636364 0.9814815 0.9763167 0.9818182    1    0
CART   0.8333333 0.9272727 0.9454545 0.9434636 0.9636364    1    0
NB     0.9272727 0.9636364 0.9816498 0.9781349 1.0000000    1    0
SVM    0.9272727 0.9636364 0.9818182 0.9799643 1.0000000    1    0

Kappa
            Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
LG     0.8062016 0.9187970 0.9592894 0.9414807 0.9903846    1    0
LDA    0.8467967 0.9209138 0.9589041 0.9449292 0.9609203    1    0
GLMNET 0.8778682 0.9215407 0.9600850 0.9565989 1.0000000    1    0
```

```
KNN    0.8062016 0.9215407 0.9592894 0.9488494 0.9609203    1    0
CART   0.6301370 0.8401563 0.8787558 0.8762248 0.9195906    1    0
NB     0.8430813 0.9215407 0.9595853 0.9525306 1.0000000    1    0
SVM    0.8467967 0.9215407 0.9602888 0.9568705 1.0000000    1    0
```

```r
dotplot(transformResults)
```



We can see that the accuracy of the previous best algorithm KNN was elevated to 97.63%. We have a new ranking, showing SVM with the most accurate mean accuracy at 97.99%.

0.8 Algorithm Tuning

Let's try some tuning of the top algorithms, specically SVM and see if we can lift the accuracy.

0.8.1 Tuning SVM

The SVM implementation has two parameters that we can tune with caret package. The sigma which is a smoothing term, and C which is a cost constraint. You can learn more about these parameters in the help for the ksvm() function ?ksvm. Let's try a range of values for C between 1 and 10 and a few small values for sigma around the default of 0.1.

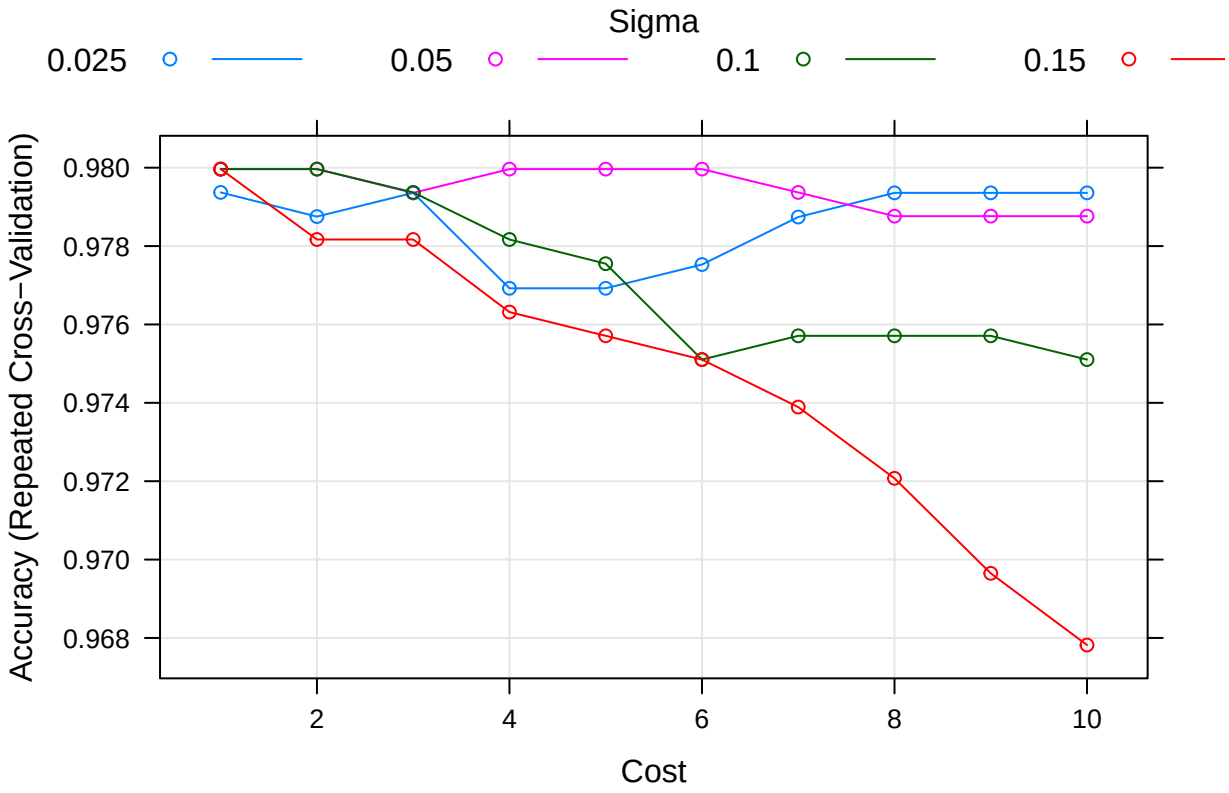
```r
# 10-fold cross validation with 3 repeats
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
```

```
set.seed(7)
grid <- expand.grid(.sigma=c(0.025, 0.05, 0.1, 0.15), .C=seq(1, 10, by=1))
fit.svm <- train(Class~., data=dataset, method="svmRadial", metric=metric, tuneGrid=grid,
preProc=c("BoxCox"), trControl=trainControl, na.action = na.omit)
print(fit.svm)
```

```
Support Vector Machines with Radial Basis Function Kernel

560 samples
  9 predictor
  2 classes: 'benign', 'malignant'

Pre-processing: Box-Cox transformation (9)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 493, 492, 493, 493, 493, 494, ...
Resampling results across tuning parameters:

  sigma  C   Accuracy   Kappa
  0.025   1  0.9793695  0.9555282
  0.025   2  0.9787522  0.9541557
  0.025   3  0.9793583  0.9555115
  0.025   4  0.9769228  0.9500746
  0.025   5  0.9769228  0.9500746
  0.025   6  0.9775289  0.9514312
  0.025   7  0.9787410  0.9541436
  0.025   8  0.9793583  0.9555134
  0.025   9  0.9793583  0.9555134
  0.025  10  0.9793583  0.9555134
  0.050   1  0.9799643  0.9568681
  0.050   2  0.9799643  0.9568692
  0.050   3  0.9793583  0.9555134
  0.050   4  0.9799643  0.9568705
  0.050   5  0.9799643  0.9568705
  0.050   6  0.9799643  0.9568697
  0.050   7  0.9793691  0.9555606
  0.050   8  0.9787630  0.9542036
  0.050   9  0.9787630  0.9542036
  0.050  10  0.9787630  0.9542036
  0.100   1  0.9799643  0.9568705
  0.100   2  0.9799643  0.9568705
  0.100   3  0.9793691  0.9555602
  0.100   4  0.9781678  0.9529764
  0.100   5  0.9775505  0.9516065
  0.100   6  0.9751038  0.9462834
  0.100   7  0.9757099  0.9475738
  0.100   8  0.9757099  0.9475738
  0.100   9  0.9757099  0.9475738
  0.100  10  0.9751038  0.9462168
  0.150   1  0.9799643  0.9568705
  0.150   2  0.9781678  0.9529772
  0.150   3  0.9781678  0.9529764
  0.150   4  0.9763159  0.9489308
  0.150   5  0.9757099  0.9475738
```

```
  0.150   6  0.9751038  0.9462168
  0.150   7  0.9738917  0.9435078
  0.150   8  0.9720731  0.9394384
  0.150   9  0.9696489  0.9339416
  0.150  10  0.9678195  0.9299249

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.15 and C = 1.
```

```r
plot(fit.svm)
```



We can see that we have made very little difference to the results here. The most accurate model had a score of 97.99% (the same as our previously rounded score of 97.99%) using a sigma = 0.50 and C = 1. We could tune further, but I don't expect a payoff.

0.8.2 Tuning KNN

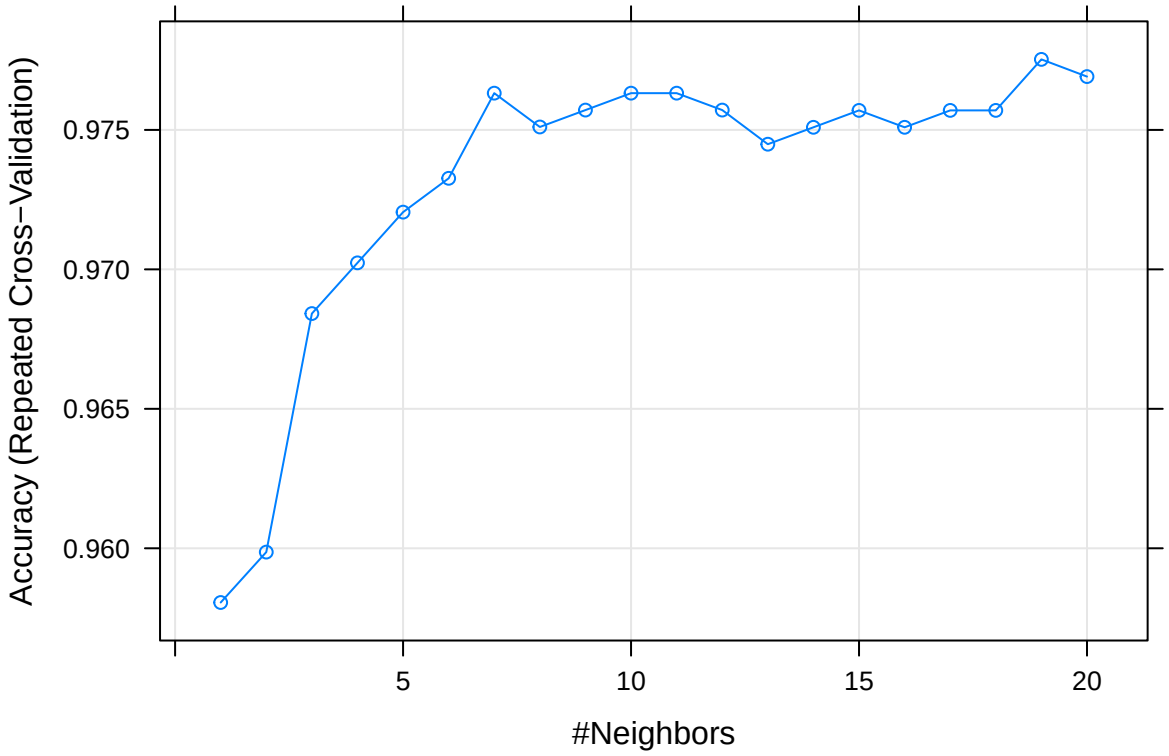
The KNN implementation has one parameter that we can tune with caret: k the number of closest instances to collect in order to make a prediction. Let's try all k values between 1 and 20.

```r
# 10-fold cross validation with 3 repeats
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
set.seed(7)
grid <- expand.grid(.k=seq(1,20,by=1))
```

```
fit.knn <- train(Class~., data=dataset, method="knn", metric=metric, tuneGrid=grid,
preProc=c("BoxCox"), trControl=trainControl, na.action = na.omit)
print(fit.knn)
```

```
k-Nearest Neighbors

560 samples
  9 predictor
  2 classes: 'benign', 'malignant'

Pre-processing: Box-Cox transformation (9)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 493, 492, 493, 493, 493, 494, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.9580560  0.9076258
   2  0.9598633  0.9116015
   3  0.9684151  0.9310511
   4  0.9702337  0.9350732
   5  0.9720519  0.9392293
   6  0.9732640  0.9419714
   7  0.9763167  0.9488494
   8  0.9751046  0.9461366
   9  0.9757107  0.9474282
  10  0.9763167  0.9487852
  11  0.9763167  0.9487523
  12  0.9757107  0.9473607
  13  0.9744873  0.9446992
  14  0.9750934  0.9460908
  15  0.9756995  0.9474150
  16  0.9750934  0.9460234
  17  0.9756995  0.9473800
  18  0.9756995  0.9474471
  19  0.9775289  0.9514969
  20  0.9769116  0.9500624

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 19.
```

```
plot(fit.knn)
```



We can see again that tuning has made little difference, settling on a value of k = 19 with an accuracy of 97.75%. This is higher than the previous 97.57%, but very similar (or perhaps identical!) to the result achieved by the tuned SVM.

0.8.3 Ensemble Methods

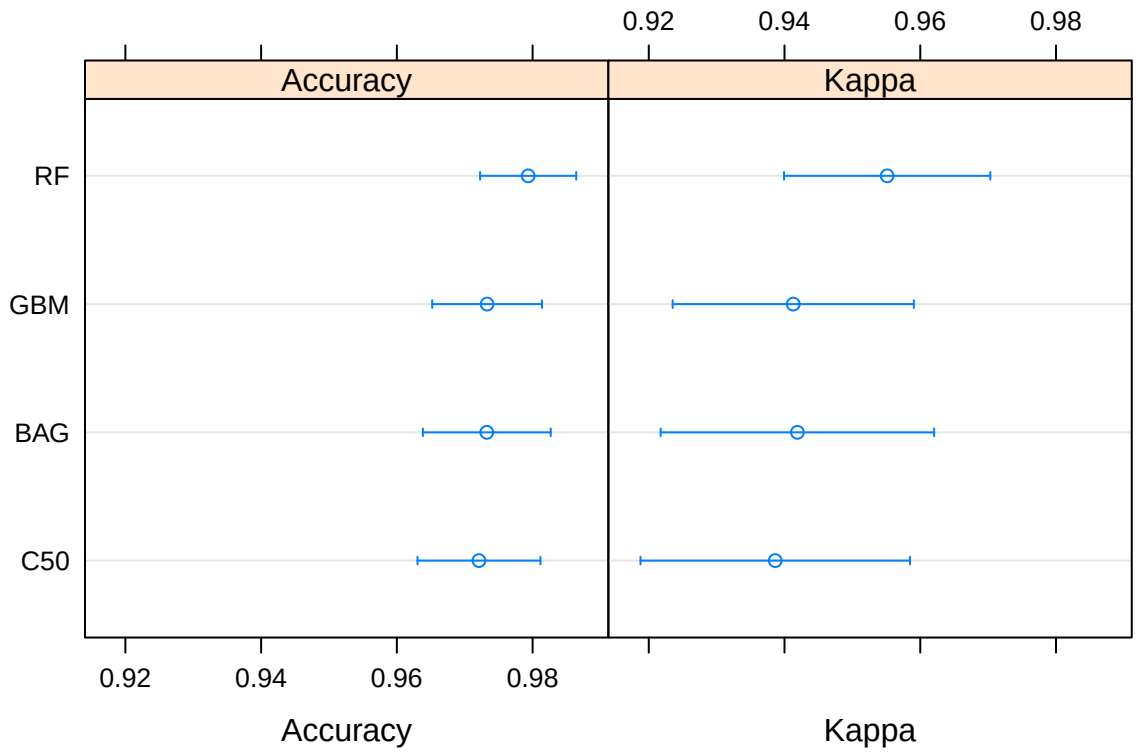
As a final check, lets look at some boosting and bagging ensemble algorithms on the dataset. We expect them to do quite well given the decision trees that underlie these methods. If our guess about hitting the accuracy ceiling is true, we may also see these methods top out around 97.20%. Let's look at 4 ensemble methods: - Bagging: Bagged CART (BAG) and Random Forest (RF). - Boosting: Stochastic Gradient Boosting (GBM) and C5.0 (C50). We will use the same test harness as before including the Box-Cox transform that fattens out the distributions.

```
# 10-fold cross validation with 3 repeats
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
# Bagged CART
set.seed(7)
fit.treebag <- train(Class~., data=dataset, method="treebag", metric=metric,
trControl=trainControl, na.action = na.omit)
# Random Forest
set.seed(7)
fit.rf <- train(Class~., data=dataset, method="rf", metric=metric, preProc=c("BoxCox"),
trControl=trainControl, na.action = na.omit)
# Stochastic Gradient Boosting
set.seed(7)
```

```
fit.gbm <- train(Class~., data=dataset, method="gbm", metric=metric, preProc=c("BoxCox"),
trControl=trainControl, verbose=FALSE, na.action = na.omit)
# C5.0
set.seed(7)
fit.c50 <- train(Class~., data=dataset, method="C5.0", metric=metric, preProc=c("BoxCox"),
trControl=trainControl, na.action = na.omit)
# Compare results
ensembleResults <- resamples(list(BAG=fit.treebag, RF=fit.rf, GBM=fit.gbm, C50=fit.c50))
summary(ensembleResults)
```

```
Call:
summary.resamples(object = ensembleResults)

Models: BAG, RF, GBM, C50
Number of resamples: 30

Accuracy
         Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
BAG 0.9074074 0.9631313 0.9818182 0.9732520 0.9818182    1    0
RF  0.9259259 0.9814815 0.9818182 0.9793466 0.9954545    1    0
GBM 0.9285714 0.9629630 0.9814815 0.9732969 0.9954545    1    0
C50 0.9074074 0.9636364 0.9816498 0.9720960 0.9818182    1    0

Kappa
         Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
BAG 0.8040639 0.9198692 0.9592894 0.9419018 0.9602888    1    0
RF  0.8414097 0.9589041 0.9600850 0.9551226 0.9902827    1    0
GBM 0.8444444 0.9189954 0.9589041 0.9412868 0.9900722    1    0
C50 0.7945205 0.9175412 0.9592894 0.9386319 0.9602888    1    0
```

```
dotplot(ensembleResults)
```



We see that Random Forest was the most accurate with a score of 97.93%. Very similar to our tuned models above. We could spend time tuning the parameters of random forest (e.g. increasing the number of trees) and the other ensemble methods, but I don't expect to see better accuracy scores other than random statistical fluctuations.

0.9 Finalize Model

We now need to finalize the model, which really means choose which model we would like to use. For simplicity I would probably select the KNN method, at the expense of the memory required to store the training dataset. SVM would be a good choice to trade-off space and time complexity. I probably would not select the Random Forest algorithm given the complexity of the model. It seems overkill for this dataset, lots of trees with little benefit in Accuracy. Let's go with the KNN algorithm. This is really simple, as we do not need to store a model. We do need to capture the parameters of the Box-Cox transform though. And we also need to prepare the data by removing the unused Id attribute and converting all of the inputs to numeric format. The implementation of KNN (knn3()) belongs to the caret package and does not support missing values. We will have to remove the rows with missing values from the training dataset as well as the validation dataset. The code below shows the preparation of the pre-processing parameters using the training dataset.

```
# prepare parameters for data transform
set.seed(7)
datasetNoMissing <- dataset[complete.cases(dataset),]
x <- datasetNoMissing[,1:9]
preprocessParams <- preProcess(x, method=c("BoxCox"))
x <- predict(preprocessParams, x)
```

Next we need to prepare the validation dataset for making a prediction. We must: 1. Remove the Id attribute. 2. Remove those rows with missing data. 3. Convert all input attributes to numeric. 4. Apply the Box-Cox transform to the input attributes using parameters prepared on the training dataset.

```
# prepare the validation dataset
set.seed(7)
# remove id column
validation <- validation[,-1]
# remove missing values (not allowed in this implementation of knn)
validation <- validation[complete.cases(validation),]
# convert to numeric
for(i in 1:9) {
validation[,i] <- as.numeric(as.character(validation[,i]))
}
# transform the validation dataset
validationX <- predict(preprocessParams, validation[,1:9])
```

Now we are ready to actually make a prediction in the training dataset.

```
# make predictions
set.seed(7)
predictions <- knn3Train(x, validationX, datasetNoMissing$Class, k=9, prob=FALSE)
confusionMatrix(factor(predictions),
                factor(validation$Class))
```

```
Confusion Matrix and Statistics

           Reference
Prediction  benign malignant
  benign        83         1
  malignant      4        47

               Accuracy : 0.963
                 95% CI : (0.9157, 0.9879)
    No Information Rate : 0.6444
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9203

 Mcnemar's Test P-Value : 0.3711

            Sensitivity : 0.9540
            Specificity : 0.9792
         Pos Pred Value : 0.9881
         Neg Pred Value : 0.9216
             Prevalence : 0.6444
         Detection Rate : 0.6148
   Detection Prevalence : 0.6222
      Balanced Accuracy : 0.9666

       'Positive' Class : benign
```

We can see that the accuracy of the final model on the validation dataset is 96.3%. This is optimistic because there is only 135 rows, but it does show that we have an accurate standalone model that we could

use on other unclassified data. However, this final result shows not better than the SVM model with 97.99% accuracy.