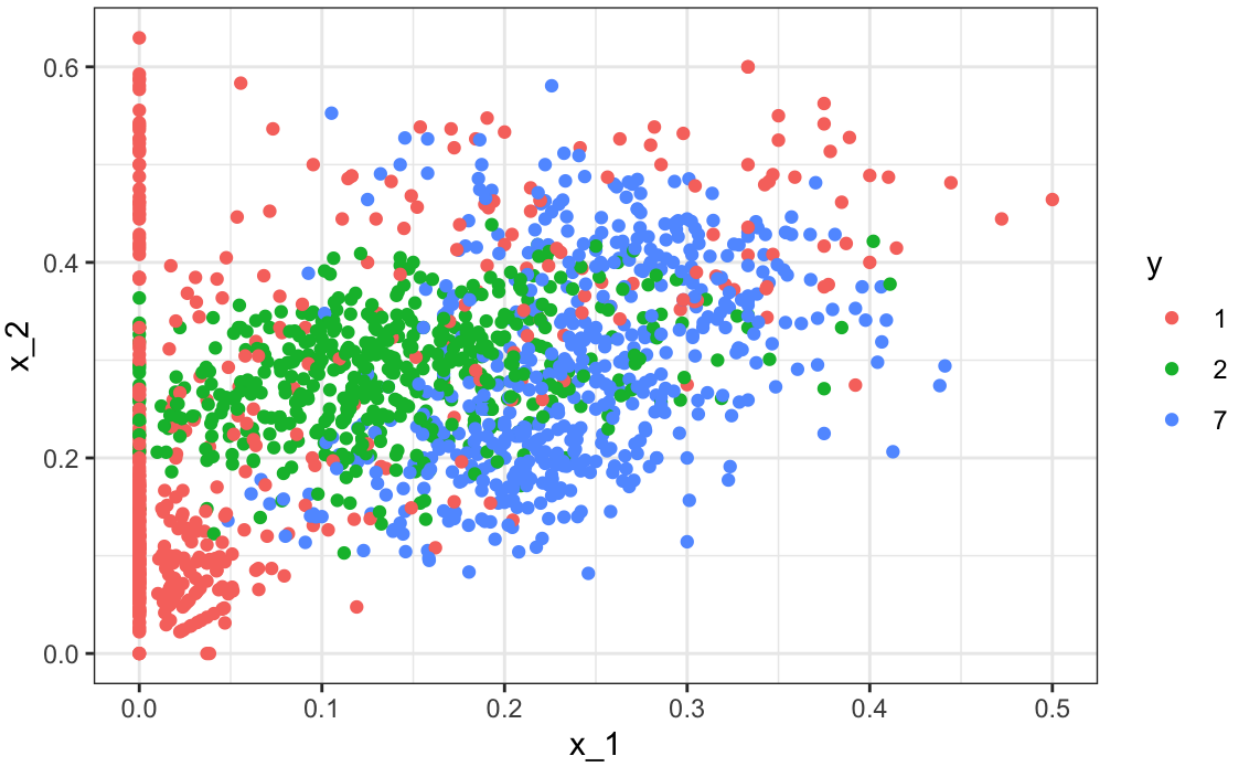
Machine Learning II

Department of Statistics, University of South Carolina

Stat 705: Data Analysis II

Case study: more than two classes

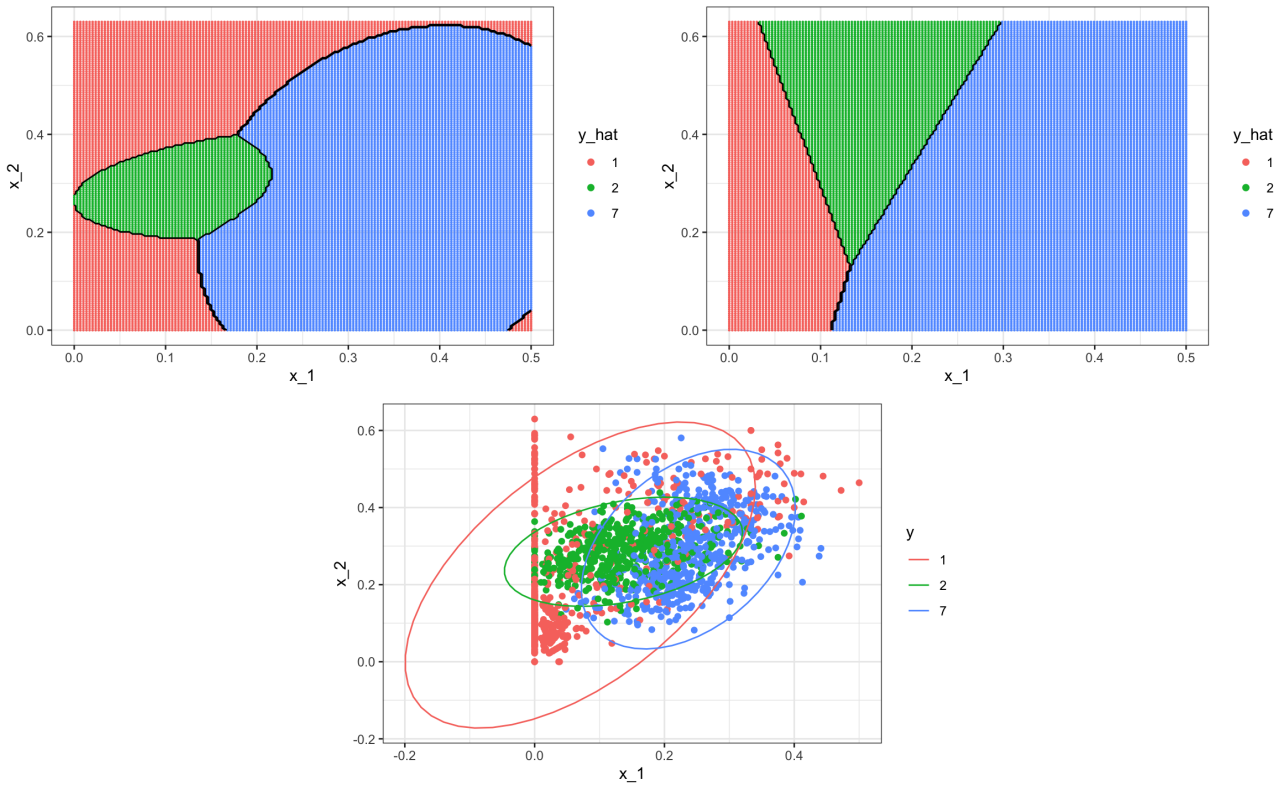


Example

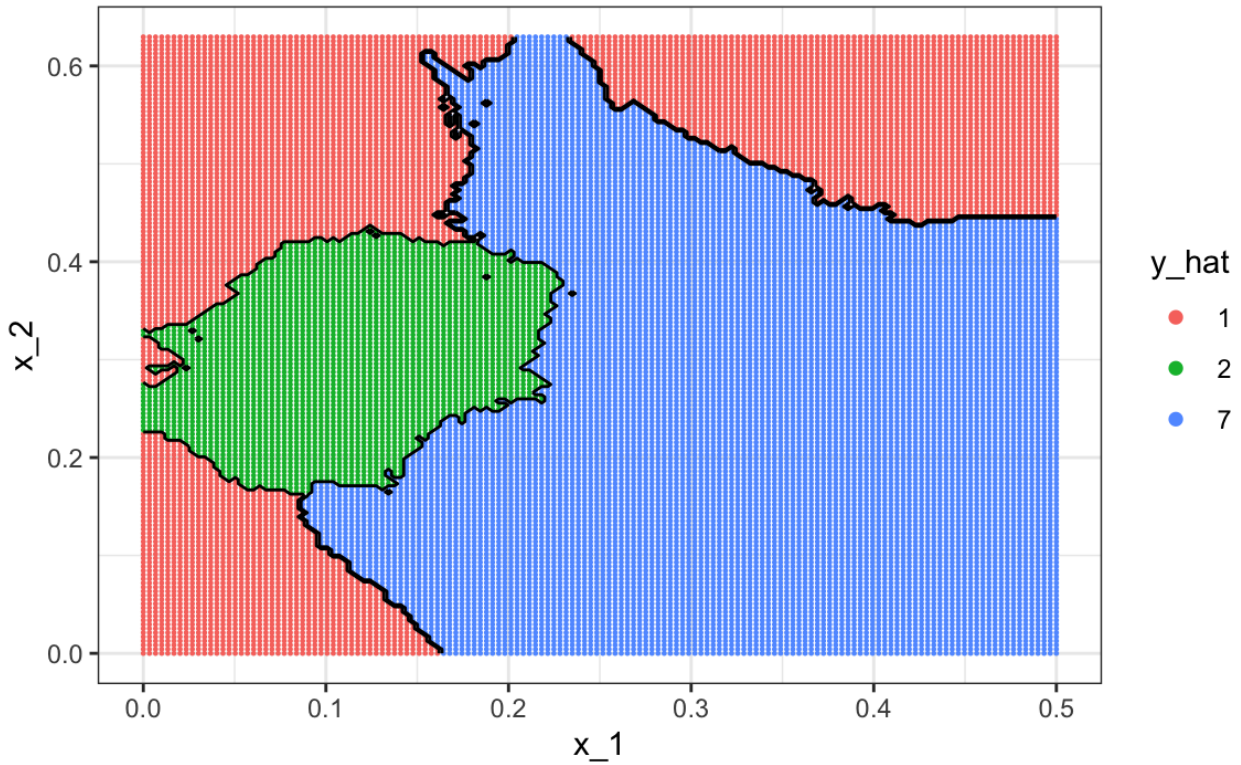
```
> train_qda <- train(y ~ ., method = "qda", data = train_set)
> pred<-predict(train_qda, test_set, type = "prob")
> head(pred)
           1           2           7
1 0.76552737 0.230426894 0.004045739
2 0.20310581 0.725142233 0.071751954
3 0.53959543 0.459086283 0.001318291
4 0.03925568 0.094193777 0.866550547
5 0.96000924 0.009356292 0.030634472
6 0.98652887 0.007239538 0.006231591
> predict(train_qda, test_set) %>% head()
[1] 1 2 1 7 1 1
> confusionMatrix(predict(train_qda, test_set), test_set$y)$table
          Reference
Prediction   1   2   7
         1 111   9  11
         2  10  86  21
         7  21  28 102
```

The accuracy is 0.749.

More than two classes: LDA and QDA

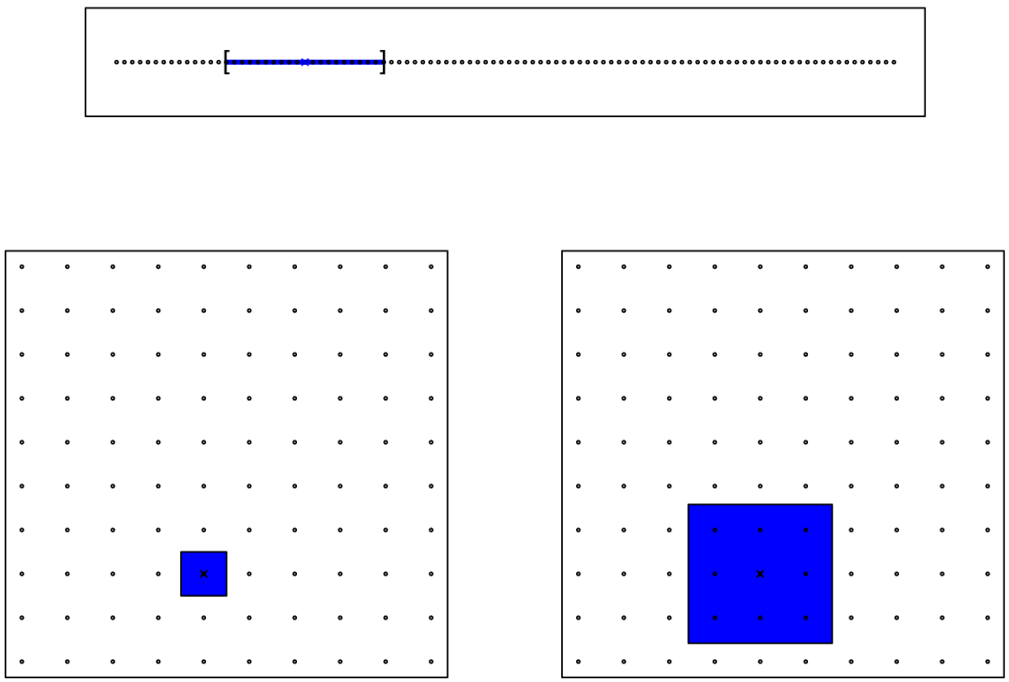


KNN

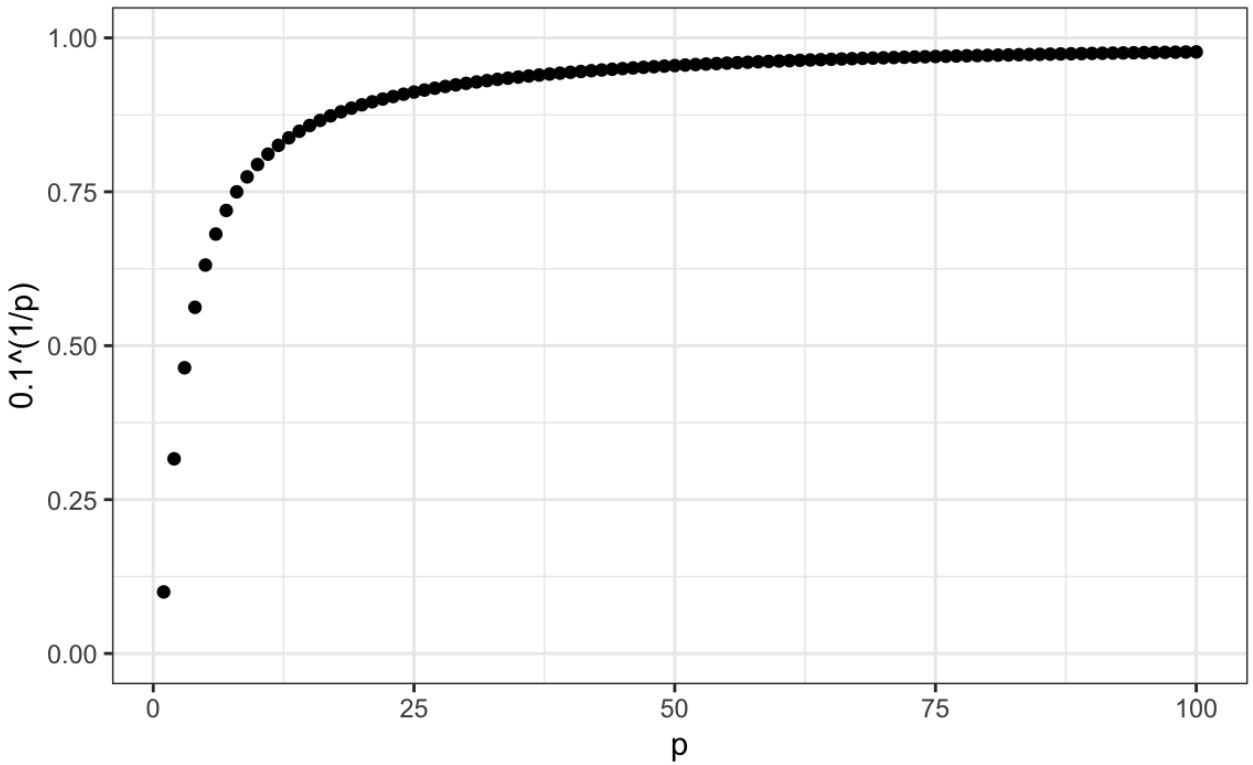


Accuracy=0.749

The Curse of Dimensionality (large p)



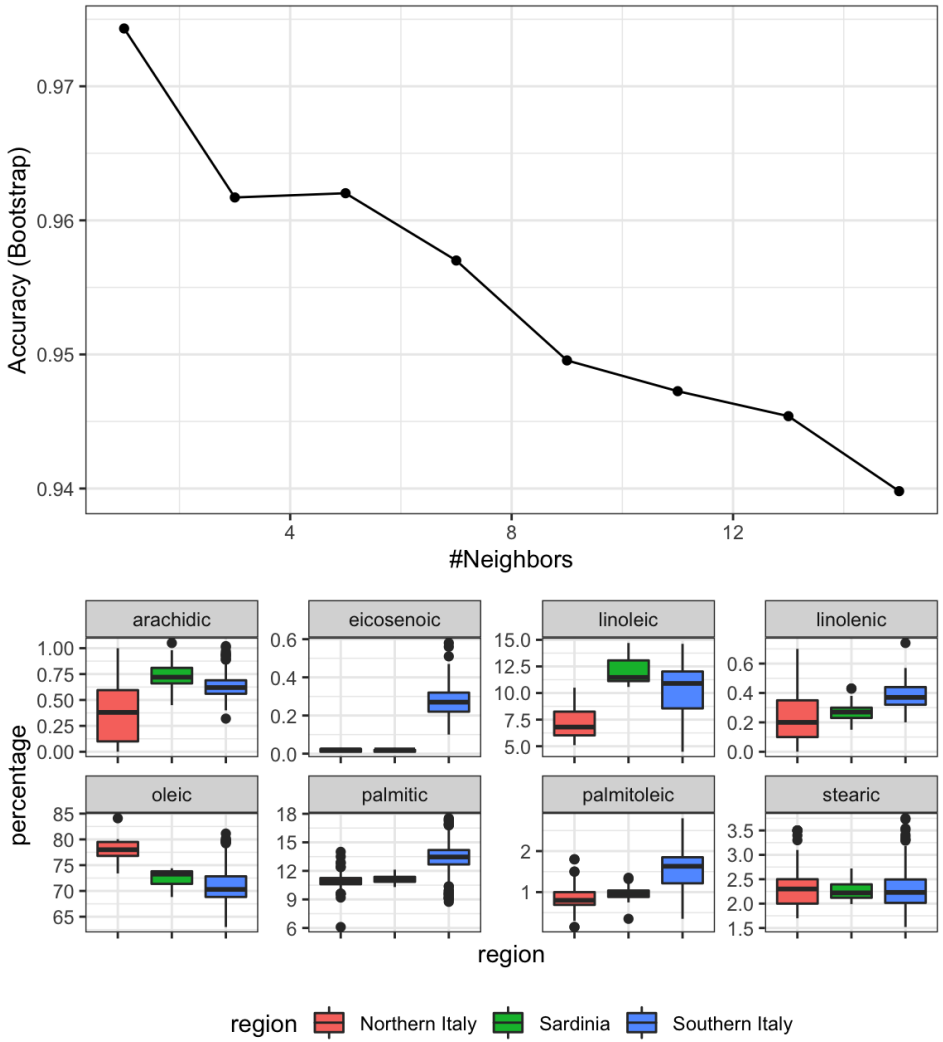
The Curse of Dimensionality



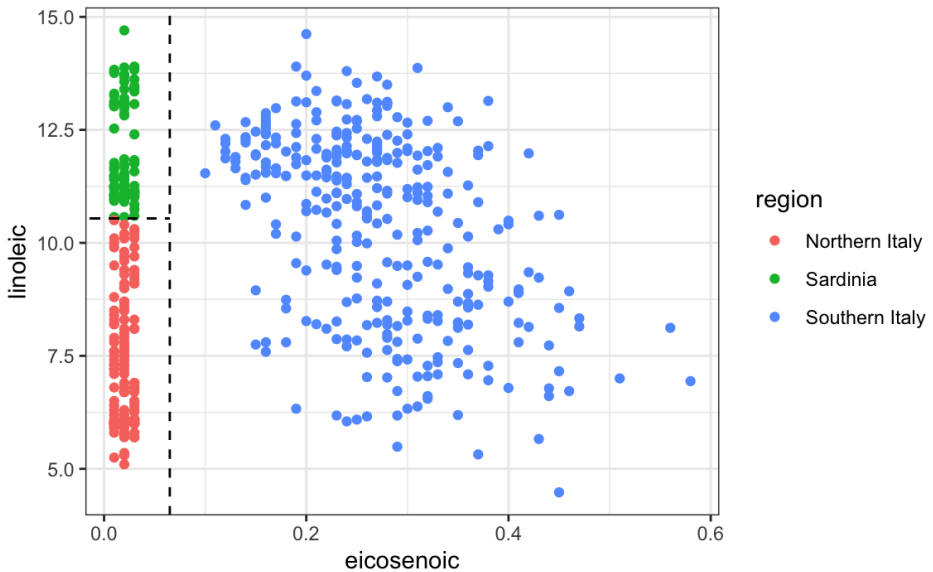
CART Motivation

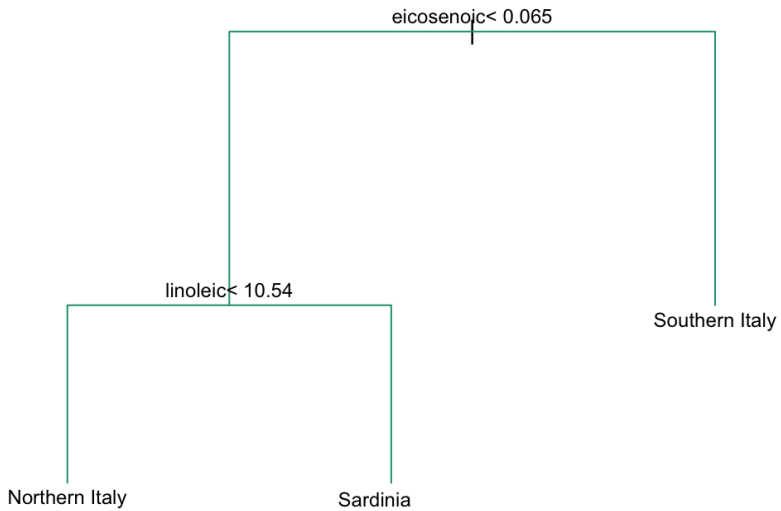
```
> library(tidyverse)
> library(dslabs)
> data("olive")
> names(olive)
 [1] "region"      "area"        "palmitic"
 [4] "palmitoleic" "stearic"     "oleic"
 [7] "linoleic"    "linolenic"   "arachidic"
[10] "eicosenoic"
> olive[1:10,]
           region         area palmitic
1  Southern Italy North-Apulia    10.75
> dim(olive)
[1] 572  10
table(olive$region)
#>
#> Northern Italy       Sardinia Southern Italy
#>            151             98            323
```

KNN


Accuracy (Bootstrap)
0.97
0.96
0.95
0.94
4
8
12
#Neighbors
arachidic
eicosenoic
linoleic
linolenic
oleic
palmitic
palmitoleic
stearic
percentage
region
region
Northern Italy
Sardinia
Southern Italy

Partition Predictor Space





Regression Trees

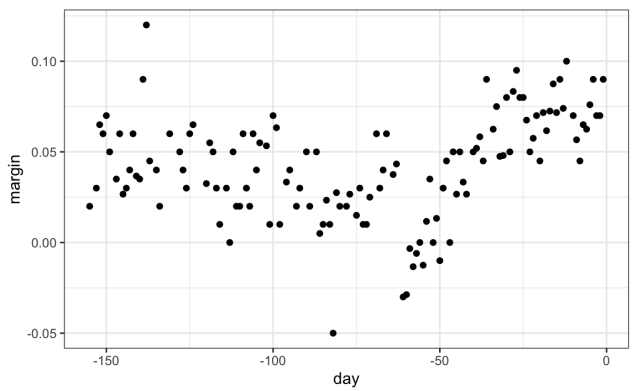
- We divide the predictor space (the set of possible values for X_1, X_2, ...X_p) into J distinct and non-overlapping regions, R_1, R_2, ...R_J.
- For every observation that falls in the region R_j, $\widehat{y_{R_j}} = \sum_{i:x_i \in R_j} y_i / n_{R_j}$.

Regression Trees

$$R_1(j,s) = \{\mathbf{x}|x_j < s\} \text{ and } R_2\{j,s\} = \{\mathbf{x}|x_j \geq s\}$$

Pick j and s that minimizes the residual sum of squares (RSS):

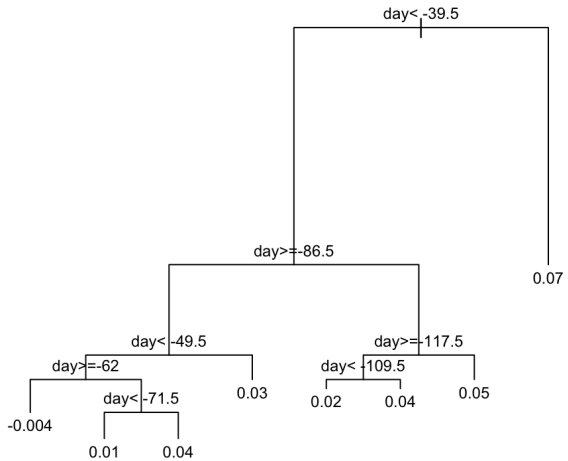
$$\sum_{i:x_i \in R_1(j,s)} (y_i - \widehat{y_{R_1}})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \widehat{y_{R_2}})^2$$

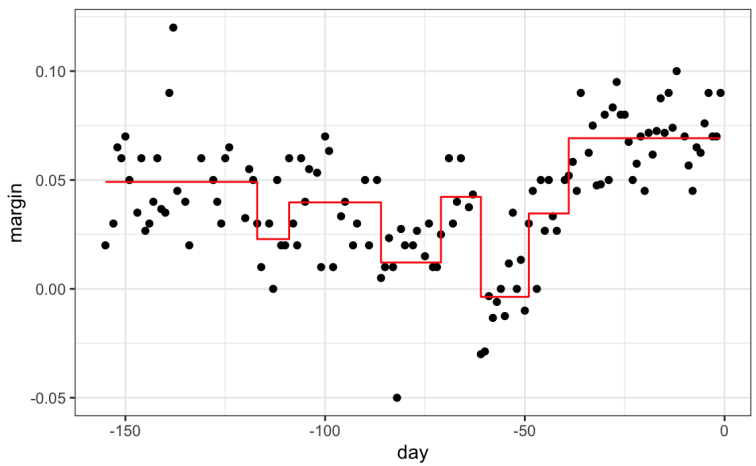


Example

```
> head(polls_2008)
    day margin
  <dbl>  <dbl>
1  -155 0.0200
2  -153 0.0300
3  -152 0.065
4  -151 0.06
5  -150 0.07
6  -149 0.05
> library(rpart)
> fit <- rpart(margin ~ ., data = polls_2008)
> plot(fit, margin = 0.1)
> text(fit, cex = 0.75)
```

Tree

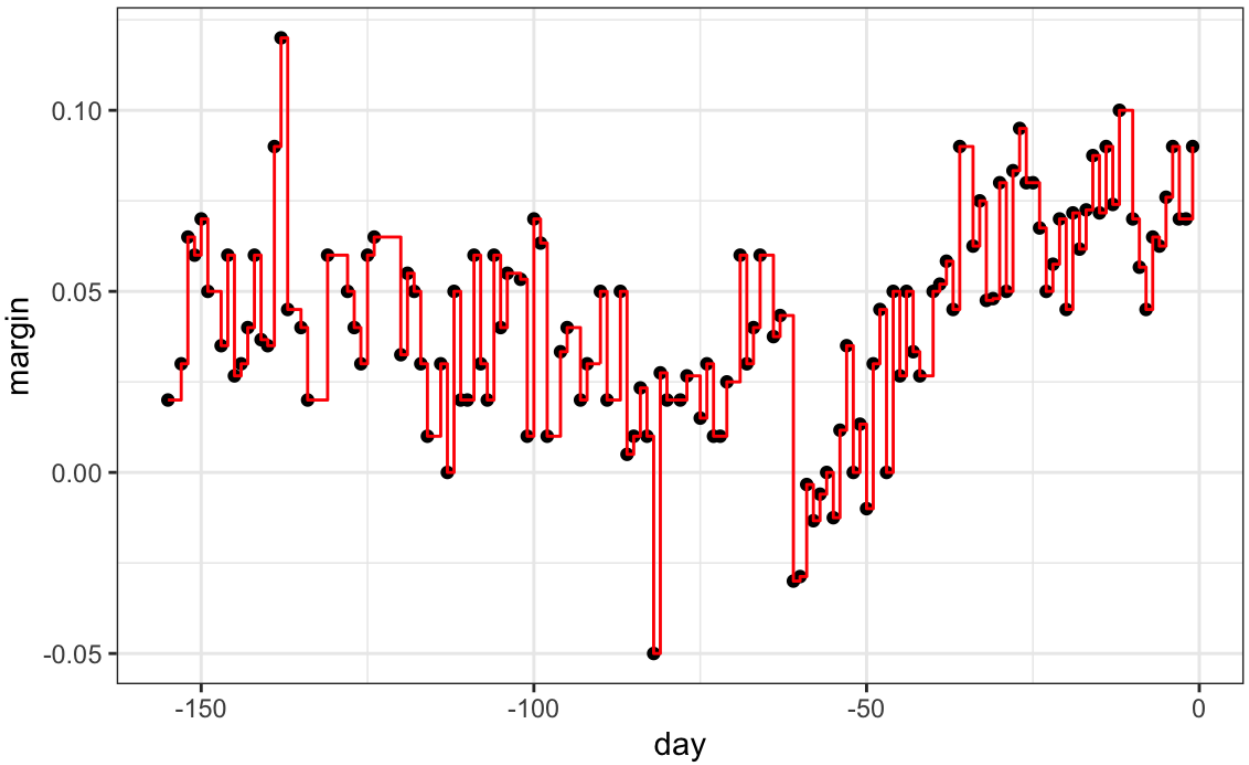

day< -39.5
day>=-86.5
0.07
day< -49.5
day>=-117.5
day>=-62
day< -109.5
0.03
day< -71.5
0.02
0.04
0.05
-0.004
0.01
0.04


0.10
0.05
0.00
-0.05
margin
-150
-100
-50
0
day

Complexity Parameter (CP)

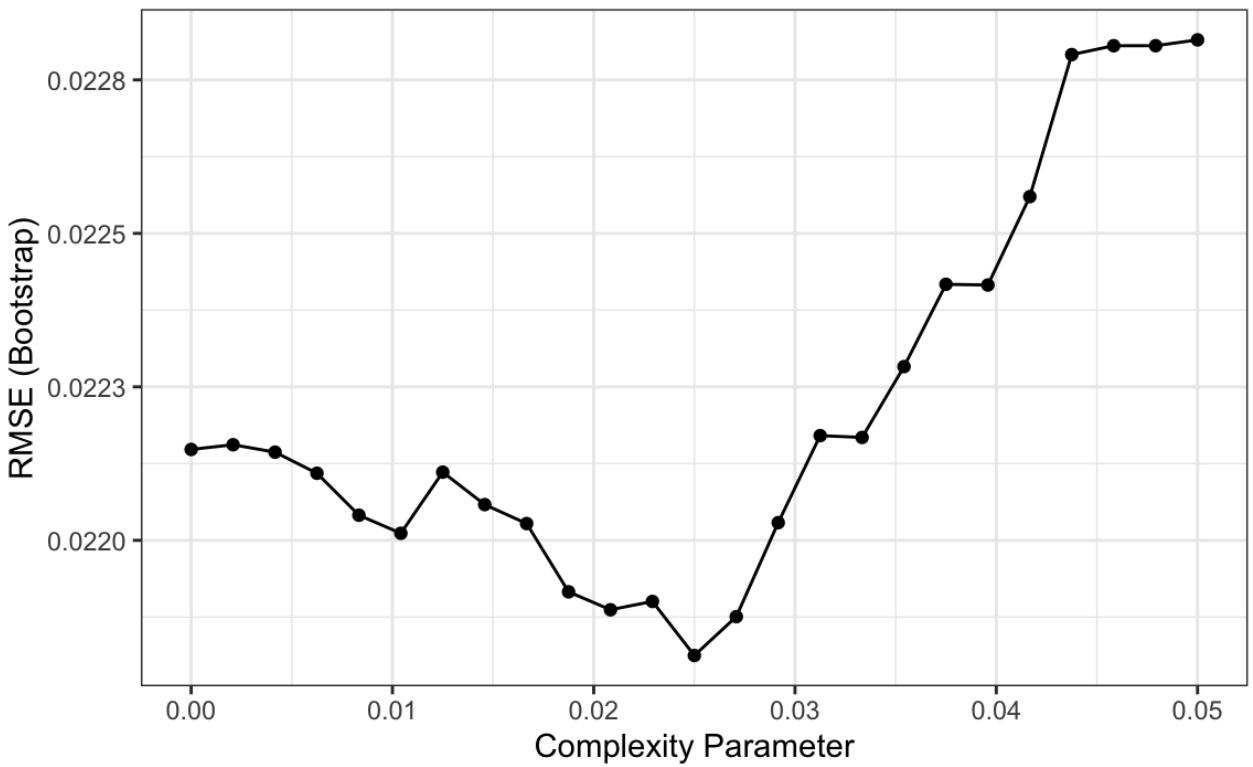
- The RSS must improve by a factor of cp for the new partition to be added.
- Large cp values will therefore force the algorithm to stop earlier (fewer nodes).
- Minimum number of observations in each node (minsplit).

CP=0, minsplit = 2

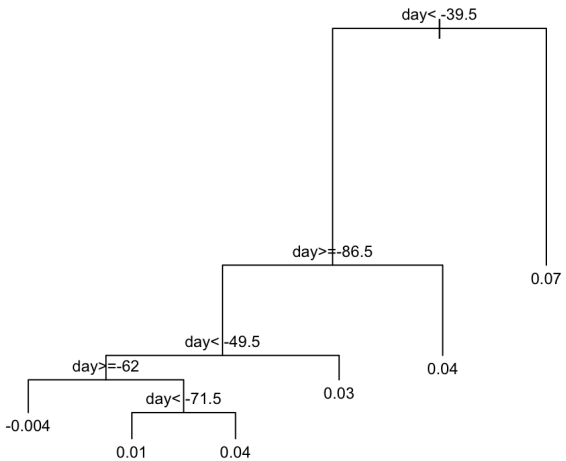


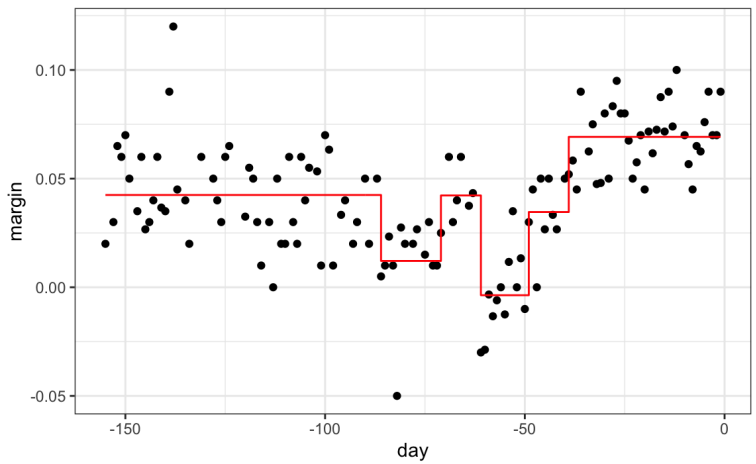
Cross Validation

```
> library(caret)
> train_rpart <- train(margin ~ .,
+                      method = "rpart",
+                      tuneGrid = data.frame(cp = seq(0, 0.05, len = 25)),
+                      data = polls_2008)
> ggplot(train_rpart)
```



Final Model





To change cp value, we can prune the tree as follows.

```
>pruned_fit <- prune(fit, cp = 0.01)
```

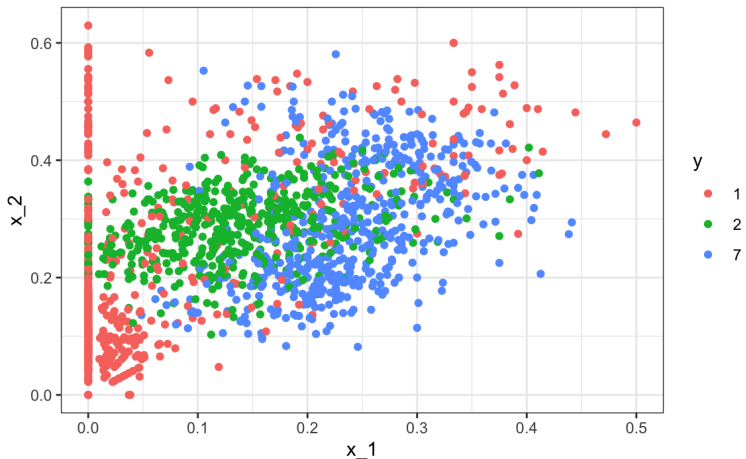
Classification (decision) Trees

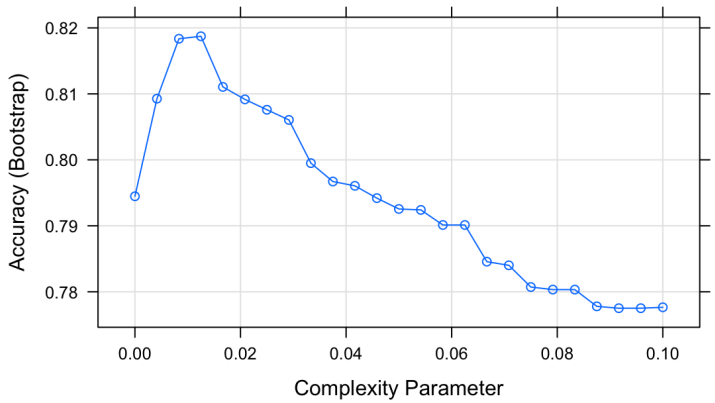
- Classification (decision) trees are used in prediction when the outcome is categorical
- Partition the predictor space
- The majority vote

$$
\begin{aligned}
Gini(j) &= \sum_{k=1}^{K} \widehat{p_{j,k}}(1 - \widehat{p_{j,k}}), \\
entropy(j) &= -\sum_{k=1}^{K} \widehat{p_{j,k}} \log(\widehat{p_{j,k}}), \text{ with } 0 \times \log(0) \text{ defined as } 0
\end{aligned}
$$

where $\widehat{p_{j,k}}$ is the proportion of observations in partition j that are of class k.

Digits example

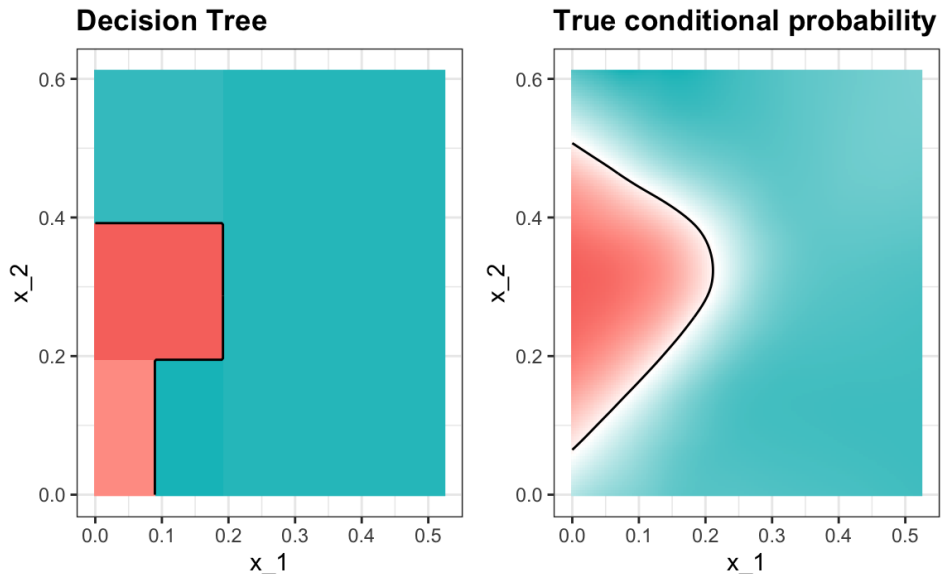





```
> train_rpart <- train(y ~ .,
+                      method = "rpart",
+                      tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 25)),
+                      data = mnist_27$train)
> plot(train_rpart)
```

Regression Tree

```
> y_hat <- predict(train_rpart, mnist_27$test)
> confusionMatrix(y_hat, mnist_27$test$y)$overall["Accuracy"]
Accuracy
    0.82
```



- Each partition creates a discontinuity
- Highly interpretable, easy to visualize (if small enough).
- Model human decision processes
- It is rarely the best performing method (accuracy) since it is not very flexible and is highly unstable to changes in training data.

Random Forest

Reduce instability by averaging multiple decision trees (a forest of trees constructed with randomness)

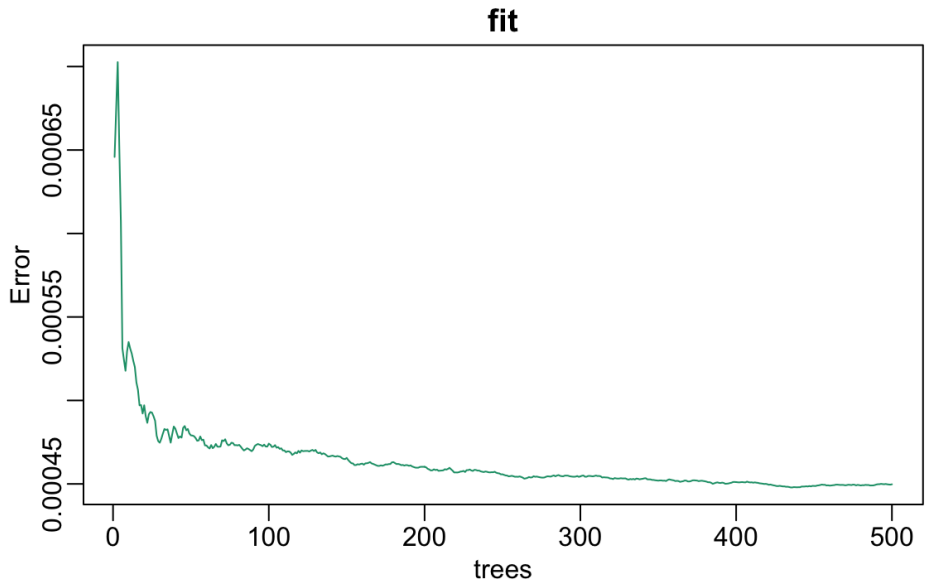
- Build decision trees using the training set. We refer to the fitted models as T_1, T_2, ..., T_B.
- For every observation in the test set, form a prediction $\widehat{y}_j$ using tree T_j.
- For continuous outcomes, form a final prediction with the average $\widehat{y}_j = \frac{1}{B}\sum_{j=1}^{B}\widehat{y}_j$.
- For categorical outcomes, for a final prediction with majority vote.

Random Trees

Let N be the number of observations in the training set. To create T_1, T_2, ..., T_B:

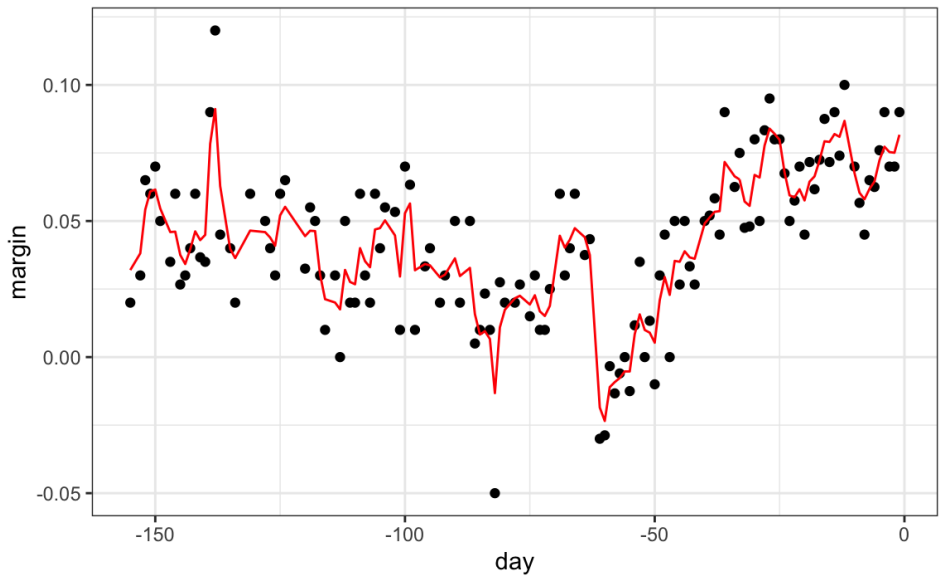
- Create a bootstrap training set by sampling N observations from the training set with replacement
- Randomly selecting features to be included in the building of each tree. A different random subset is selected for each tree. This reduces correlation between trees in the forest, thereby improving prediction accuracy.

Random Forest



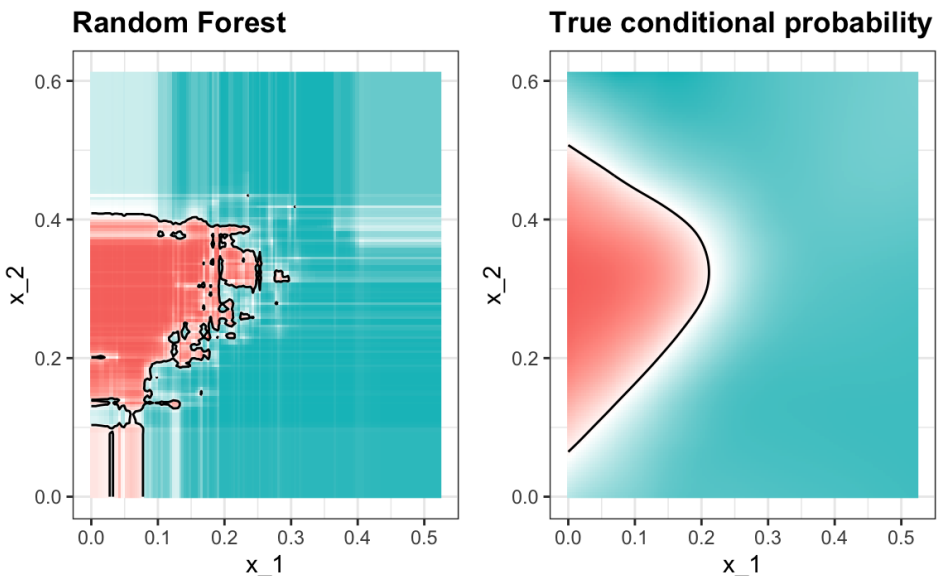
We can see that in this case, the accuracy improves as we add more trees until about 30 trees where accuracy stabilizes.

Random Forest Result



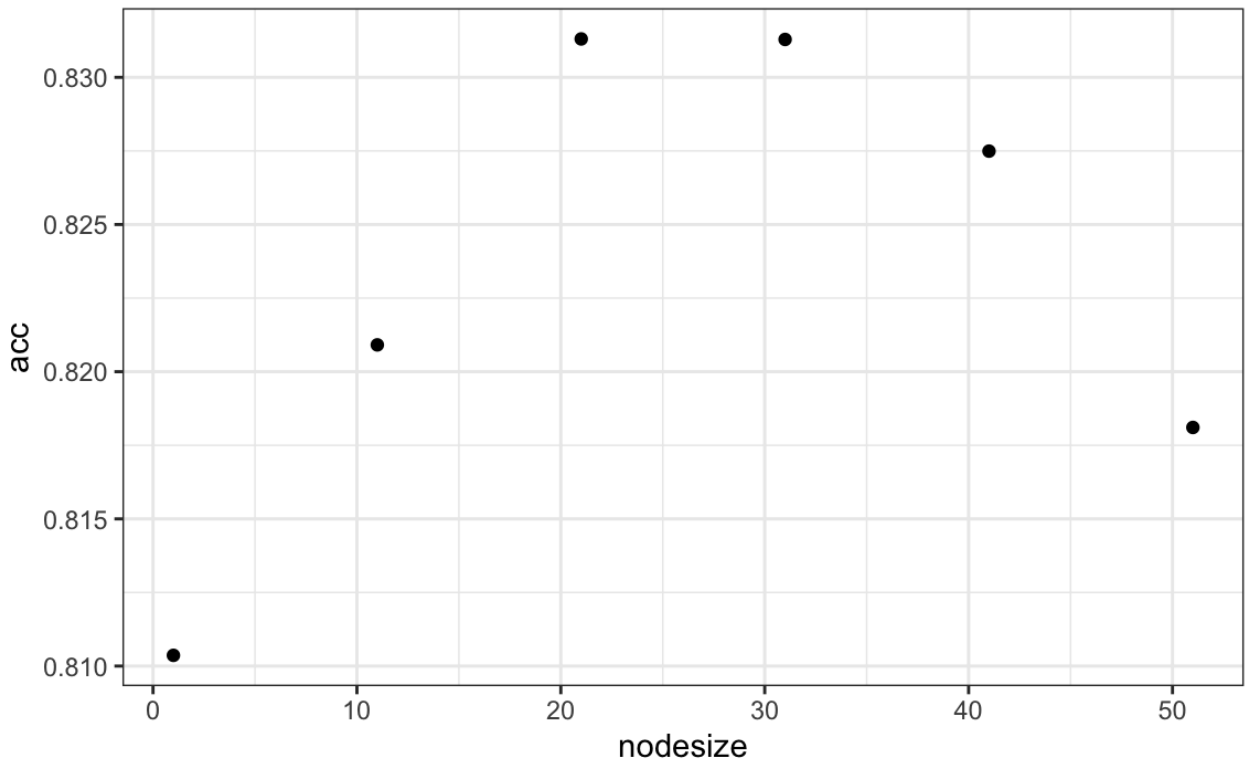
Random Forest

```
library(randomForest)
> train_rf <- randomForest(y ~ ., data=mnist_27$train)
>
> confusionMatrix(predict(train_rf, mnist_27$test),
+                 mnist_27$test$y)$overall["Accuracy"]
Accuracy
   0.785
```



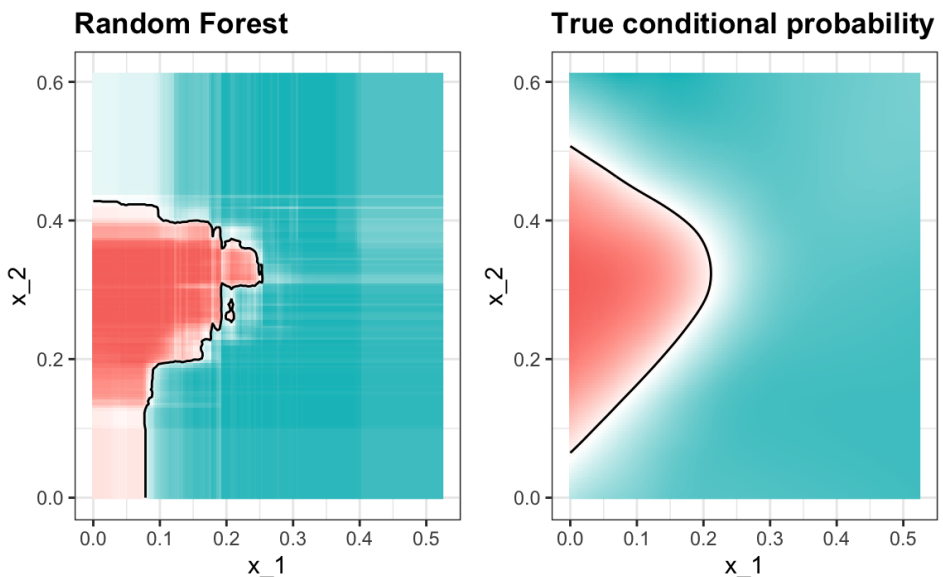
Train Random Forest

- mtry: the minimum number of data points in the nodes of the tree

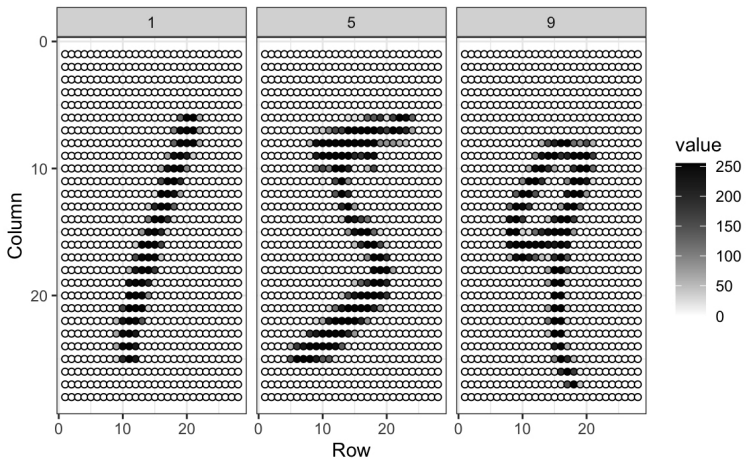


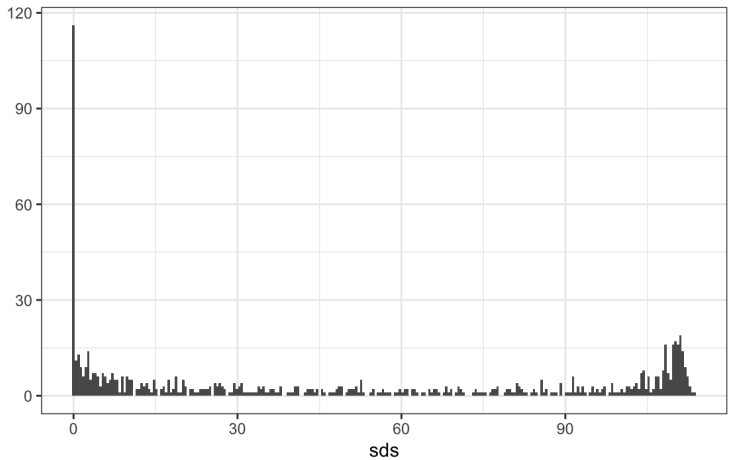
Random Forest

```
> train_rf_2 <- randomForest(y ~ ., data=mnist_27$train,
+                            nodesize = nodesize[which.max(acc)])
>
> confusionMatrix(predict(train_rf_2, mnist_27$test),
+                 mnist_27$test$y)$overall["Accuracy"]
Accuracy
    0.83
```



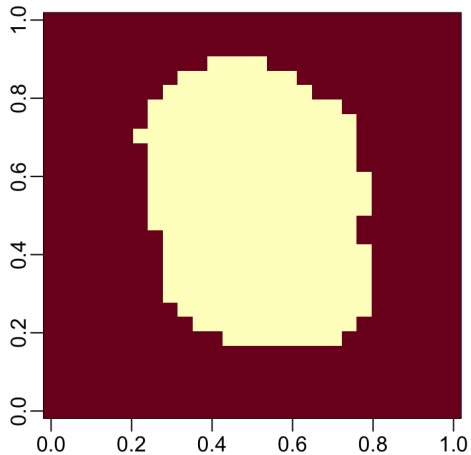
Machine Learning in Practice





Example

```
>library(caret)
>nzv <- nearZeroVar(x)
>image(matrix(1:784 %in% nzv, 28, 28))
```



KNN: training k

```
>control <- trainControl(method = "cv", number = 10, p = .9)
>train_knn <- train(x[ ,col_index], y,
+                    method = "knn",
+                    tuneGrid = data.frame(k = c(3,5,7)),
+                    trControl = control)
>train_knn
>fit_knn <- knn3(x[, col_index], y,  k = 3)
>y_hat_knn <- predict(fit_knn, x_test[, col_index], type="class")
>cm <- confusionMatrix(y_hat_knn, factor(y_test))
>cm$overall["Accuracy"]
#> Accuracy
#    0.953
```

Example

```
cm$byClass[,1:2]
#>          Sensitivity Specificity
#> Class: 0       0.990       0.996
#> Class: 1       1.000       0.993
#> Class: 2       0.965       0.997
#> Class: 3       0.950       0.999
#> Class: 4       0.930       0.997
#> Class: 5       0.921       0.993
#> Class: 6       0.977       0.996
#> Class: 7       0.956       0.989
#> Class: 8       0.887       0.999
#> Class: 9       0.951       0.990
```

- the hardest to detect:

Example

```
cm$byClass[,1:2]
#>          Sensitivity Specificity
#> Class: 0       0.990       0.996
#> Class: 1       1.000       0.993
#> Class: 2       0.965       0.997
#> Class: 3       0.950       0.999
#> Class: 4       0.930       0.997
#> Class: 5       0.921       0.993
#> Class: 6       0.977       0.996
#> Class: 7       0.956       0.989
#> Class: 8       0.887       0.999
#> Class: 9       0.951       0.990
```

- the hardest to detect: 8
- the most commonly incorrectly predicted

Example

```
cm$byClass[,1:2]
#>          Sensitivity Specificity
#> Class: 0       0.990       0.996
#> Class: 1       1.000       0.993
#> Class: 2       0.965       0.997
#> Class: 3       0.950       0.999
#> Class: 4       0.930       0.997
#> Class: 5       0.921       0.993
#> Class: 6       0.977       0.996
#> Class: 7       0.956       0.989
#> Class: 8       0.887       0.999
#> Class: 9       0.951       0.990
```

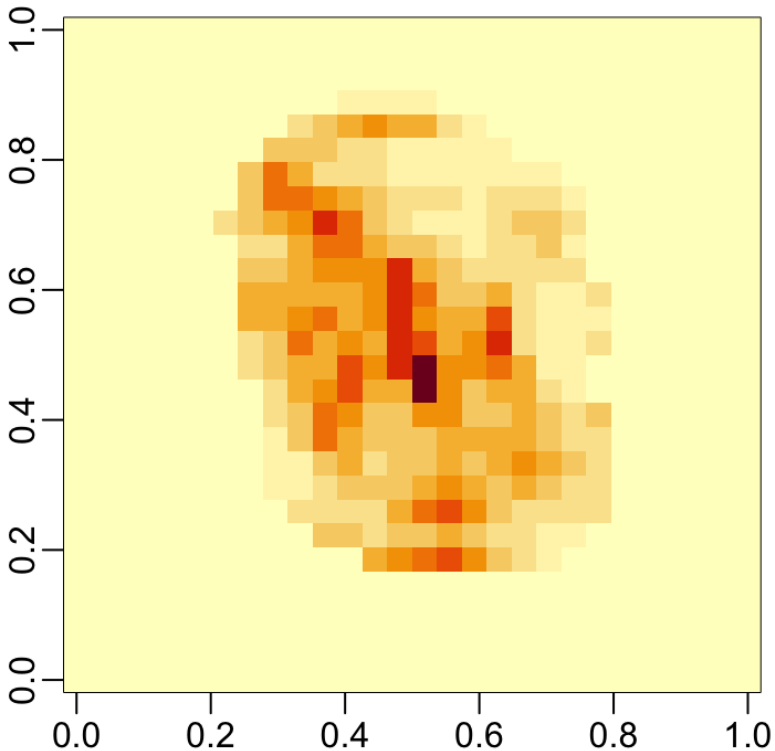
- the hardest to detect: 8
- the most commonly incorrectly predicted 7

Random Forest

```
>library(randomForest)
>control <- trainControl(method="cv", number = 5)
>grid <- data.frame(mtry = c(1, 5, 10, 25, 50, 100))
>train_rf <-  train(x[, col_index], y,
+                    method = "rf",
+                    ntree = 150,
+                    trControl = control,
+                    tuneGrid = grid,
+                    nSamp = 5000)
>fit_rf <- randomForest(x[, col_index], y,
+                        minNode = train_rf$bestTune$mtry)
>y_hat_rf <- predict(fit_rf, x_test[ ,col_index])
>cm <- confusionMatrix(y_hat_rf, y_test)
>cm$overall["Accuracy"]
#> Accuracy
#>    0.956
```

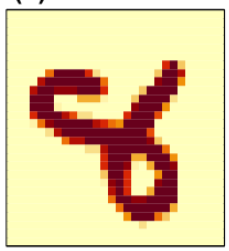
Variable Importance

```
imp <- importance(fit_rf)
```

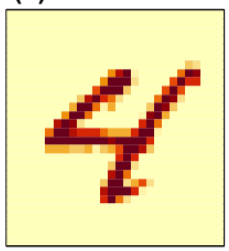


Visual Assessments

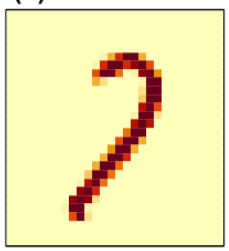
Pr(4)=0.75 but is a 8



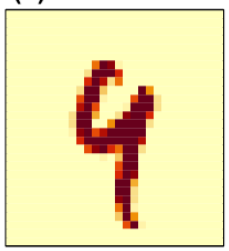
Pr(9)=0.68 but is a 4



Pr(2)=0.64 but is a 7



Pr(4)=0.59 but is a 9



Ensembles

- In machine learning, one can usually greatly improve the final results by combining the results of different algorithms.

Here, we compute new class probabilities by taking the average of random forest and kNN as an example.

```
p_rf <- predict(fit_rf, x_test[,col_index], type = "prob")
p_rf<- p_rf / rowSums(p_rf)
p_knn  <- predict(fit_knn, x_test[,col_index])
p <- (p_rf + p_knn)/2
y_pred <- factor(apply(p, 1, which.max)-1)
confusionMatrix(y_pred, y_test)$overall["Accuracy"]
#> Accuracy
#>    0.961
```