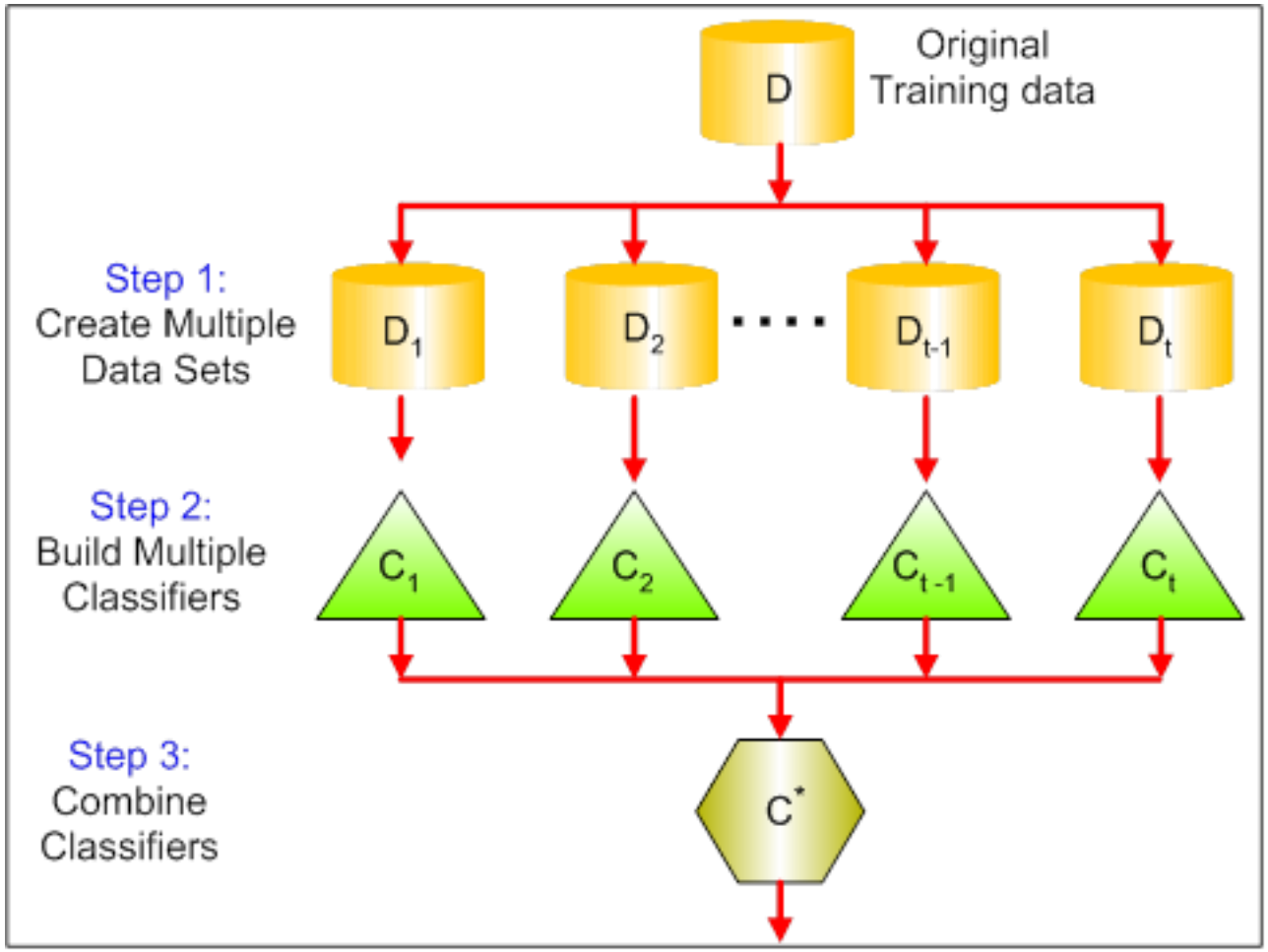
Random Forest

Bagging

Bagging is a technique used to reduce the variance of our predictions by combining the result of multiple classifiers modeled on different sub-samples of the same data set.



How bagging works

1.Create Multiple DataSets:

1. Sampling is done *with replacement* on the original data and new datasets are formed.
2. The new data sets can have a fraction of the columns as well as rows, which are generally hyper-parameters in a bagging model
3. Taking row and column fractions less than 1 helps in making robust models, less prone to overfitting

2.Build Multiple Classifiers:

1. Classifiers are built on each data set.
2. Generally the same classifier is modeled on each data set and predictions are made.

3.Combine Classifiers:

1. The predictions of all the classifiers are combined using a mean, median or mode value depending on the problem at hand.
2. The combined values are generally more robust than a single model.

Random Forest

1.Assume number of cases in the training set is N. Then, sample of these N cases is taken at random but *with replacement*. This sample will be the training set for growing the tree.

2.If there are M input variables, a number mtry<M is specified such that at each node, m variables are selected at random out of the M. The best split on these m is used to split the node. The value of m is held constant while we grow the forest.

3.Each tree is grown to the largest extent possible and there is no pruning.

4.Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression).

Bias variance problem

$$Mean\ Error = Bias^2 + Variance + \sigma^2$$

Bias – is associated with model specification and variable selection. Random forest decreases bias by using subset of the variables for node split.

Variance **–** is a associated with the differences in training data. Random forest solves the problem by using cross-validation.

Participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E). Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes.

```
pml<-read.csv("pml.csv")
table(pml$classe)
```

```
##
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

```
str(pml)
```

```
## 'data.frame':    19622 obs. of  59 variables:
##  $ user_name           : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1: int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232
##  $ raw_timestamp_part_2: int  788290 808298 820366 120339 196328 304277 368296 440390 484323 48
##  $ cvtd_timestamp      : Factor w/ 20 levels "2/12/2011 13:32",..: 15 15 15 15 15 15 15 15 15 1
##  $ new_window          : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ num_window          : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt           : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt          : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ yaw_belt            : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
```

```
##  $ magnet_forearm_x    : int  -17 -18 -18 -16 -17 -9 -18 -9 -16 -22 ...
##  $ magnet_forearm_y    : num  654 661 658 658 655 660 659 660 653 656 ...
##  $ magnet_forearm_z    : num  476 473 469 469 473 478 470 474 476 473 ...
##  $ classe              : Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Make training and testing sets

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
set.seed(1)
index <- createDataPartition(pml$classe, p = 0.8, list = FALSE)
# subset
train <- pml[index, ]
test <- pml[-index, ]
```

randomForest

Use randomForest package

ntree – specifies how many trees do you want to build
mtry (omitted here) – Number of variables randomly sampled as candidates at each split. for classification problem the default value is sqrt(p) where p is number of variables in training set)

.

do.trace –gives verbose output as randomForest is run

```
library(randomForest)
```

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(1)
model_f<-model<-randomForest(classe~., data=train, ntree=50,
                             do.trace=T)
```

we have 5 classes
Output shows the out-of-the-bag error in general and for each class for given number of the trees

```
set.seed(1)
model_f<-model<-randomForest(classe~., data=train, ntree=50,
                             do.trace=T)
```

```
## ntree      OOB      1      2      3      4      5
##     1:   3.30%  1.59%  4.13%  5.30%  3.39%  3.03%
##     2:   4.08%  2.74%  4.91%  5.66%  4.85%  3.08%
##     3:   3.25%  1.92%  4.37%  4.04%  4.12%  2.60%
##     4:   2.75%  1.71%  3.65%  3.73%  2.93%  2.31%
##     5:   2.39%  1.61%  3.29%  3.23%  2.12%  2.09%
##     6:   2.30%  1.60%  3.31%  3.18%  2.25%  1.55%
##     7:   1.98%  1.23%  2.99%  2.99%  1.67%  1.41%

##    47:   0.18%  0.09%  0.13%  0.44%  0.16%  0.14%
##    48:   0.17%  0.07%  0.16%  0.40%  0.16%  0.14%
##    49:   0.15%  0.07%  0.10%  0.37%  0.16%  0.14%
##    50:   0.17%  0.07%  0.13%  0.44%  0.16%  0.14%
```

The error rates

for the last 5 tries

```
model_f$err.rate[45:50,]
```

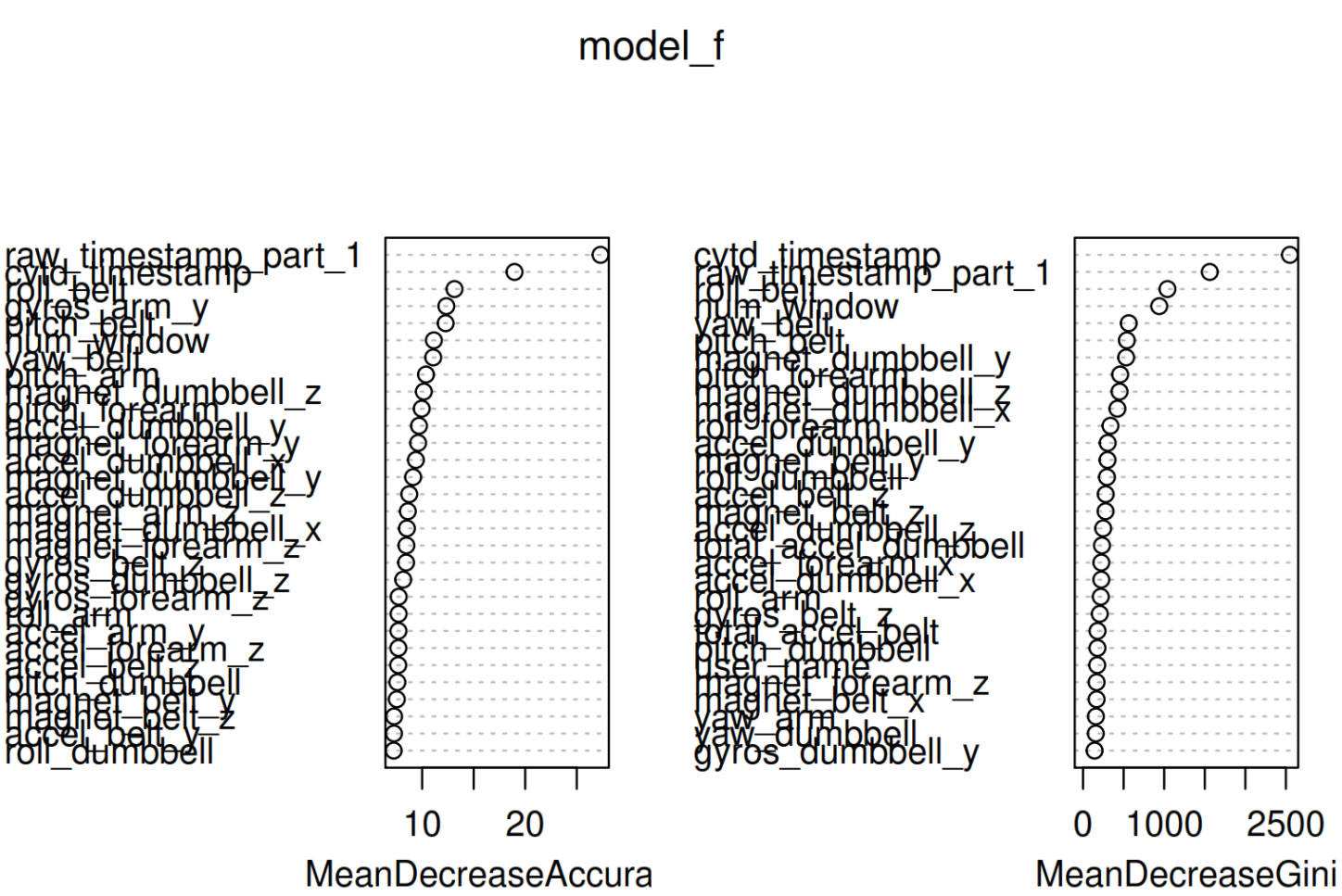
```
##              OOB A           B           C           D           E
## [1,] 0.001274080 0 0.002106927 0.002630041 0.002176617 0.0002772387
## [2,] 0.001325043 0 0.002106927 0.002922268 0.002176617 0.0002772387
## [3,] 0.001376007 0 0.002106927 0.003214494 0.002176617 0.0002772387
## [4,] 0.001325043 0 0.001843561 0.002922268 0.002176617 0.0005544774
## [5,] 0.001325043 0 0.002106927 0.002630041 0.002176617 0.0005544774
## [6,] 0.001325043 0 0.002106927 0.002922268 0.002176617 0.0002772387
```

Add argument importance=T to be able to assess the importance of the predictor variables

```
set.seed(1)
model_f<-model<-randomForest(classe~., data=pml, ntree=50, importance=T)
```

Importance of the variables

```
varImpPlot(model_f)
```



Variable importance

What is the average decrease in accuracy when the given variable doesn't take part into node split (is not among randomly chosen variables) compared to when it takes part into splitting

Making predictions with randomForest

```
pr<-predict(model,newdata=test, type="prob")
pr[1:20]
```

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

```
pr.class<-predict(model, newdata=test)
pr.class[1:20]
```

```
##   1  13  15  17  18  26  30  55  66  77  79  89  92  97  99 107 113 114
##   A   A   A   A   A   A   A   A   A   A   A   A   A   A   A   A   A   A
## 123 132
##   A   A
## Levels: A B C D E
```

Confusion matrix

```
confusionMatrix(pr.class, test$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1116    0    0    0    0
##          B    0  759    0    0    0
##          C    0    0  684    0    0
##          D    0    0    0  643    0
##          E    0    0    0    0  721
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9991, 1)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

Example

- Example: bank data
- The bank is making some offer to its customers. The output “y” indicates if the customer has accepted the offer.
- The goal is to make a classificatory that will predict the probability of the given customer to accept the offer

Bank data

```
bank<-read.csv("bank-1.csv")
str(bank)
```

```
## 'data.frame':    45211 obs. of  17 variables:
##  $ age       : int  58 44 33 47 33 35 28 42 58 43 ...
##  $ job       : Factor w/ 12 levels "admin.","blue-collar",..: 5 10 3 2 12 5 5 3 6 10 ...
##  $ marital   : Factor w/ 3 levels "divorced","married",..: 2 3 2 2 3 2 3 1 2 3 ...
##  $ education: Factor w/ 4 levels "primary","secondary",..: 3 2 2 4 4 3 3 3 1 2 ...
##  $ default   : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 2 1 1 ...
##  $ balance   : int  2143 29 2 1506 1 231 447 2 121 593 ...
##  $ housing   : Factor w/ 2 levels "no","yes": 2 2 2 2 1 2 2 2 2 2 ...
##  $ loan      : Factor w/ 2 levels "no","yes": 1 1 2 1 1 1 2 1 1 1 ...
##  $ contact   : Factor w/ 3 levels "cellular","telephone",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ day       : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ month     : Factor w/ 12 levels "apr","aug","dec",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ duration : int  261 151 76 92 198 139 217 380 50 55 ...
##  $ campaign : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ pdays     : int  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...
##  $ previous : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ poutcome : Factor w/ 4 levels "failure","other",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ y         : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
```

Training a Random Forest model is time consuming so we will take only 25 trees

```
set.seed(1)
ind<-createDataPartition(bank$y, p=0.8, list=F)
Train<-bank[ind,]
Test<-bank[-ind,]
```

```
set.seed(1)
model2<-randomForest(y~., data=Train, ntree=25, do.trace=T)
```

```
## ntree      OOB      1      2
##     1:  13.10%  7.59% 54.67%
##     2:  13.18%  7.58% 54.76%
##     3:  13.14%  7.67% 54.17%
##     4:  12.76%  7.34% 53.61%
##     5:  12.28%  6.98% 52.40%
##     6:  12.06%  6.66% 52.94%
##     7:  11.83%  6.42% 52.88%
##     8:  11.35%  5.95% 52.14%
##     9:  11.27%  5.80% 52.58%
##    10:  11.08%  5.58% 52.70%
##    11:  10.91%  5.36% 52.82%
##    12:  10.87%  5.33% 52.77%
##    13:  10.76%  5.12% 53.39%
##    14:  10.70%  5.03% 53.49%
##    15:  10.47%  4.81% 53.22%
```

Predict the probabilities

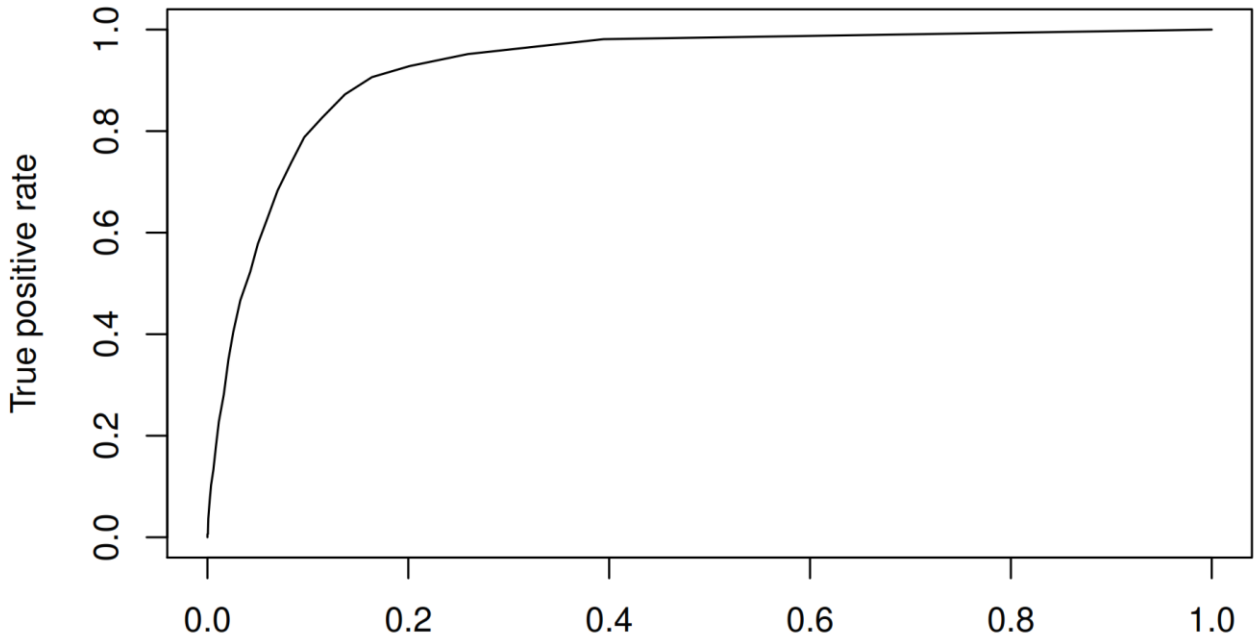
```
pr<-predict(model2, Test, type="prob")
pr[1:25,]
```

```
##       no  yes
## 2   1.00 0.00
## 3   1.00 0.00
## 6   1.00 0.00
## 24  1.00 0.00
## 28  1.00 0.00
## 32  0.96 0.04
## 34  1.00 0.00
## 38  0.24 0.76
## 41  1.00 0.00
## 44  0.40 0.60
```

```
library(ROCR)
```

We take the second column from pr matrix, which is the probability of Yes

```
p_test<-prediction(pr[,2], Test$y)
perf<-performance(p_test, "tpr", "fpr")
plot(perf)
```



Area Under the Curve

```
performance(p_test, "auc")@y.values
```

```
## [[1]]
## [1] 0.9269437
```

Doing with caret

```
library(caret)
```

```
set.seed(1)
trc<-trainControl(method="cv", number=10)
mtry_grid<-expand.grid(mtry=c(4,7,9,10))
```

- We will do grid search for variable mtry
- mtry is the number of variables that are randomly chosen from the data to participate in each split for the decision tree
- The default value is sqrt(M) where M is the total number of variables.

Doing with caret

- We will take small sample from the Train data just to save some time
- inside train function set tuneGrid=mtry_grid, where mtry_grid contains values for mtry that you want to try

```
set.seed(1)
# just for the demonstration process we will take small sample from Train
Train1<-Train[sample(nrow(Train), 5000),]
model3<-train(y~., data=Train1,
          trControl=trc,
          method="rf",
          ntree=25,
          tuneGrid=mtry_grid)
```

Doing with caret

Accuracy measures for different values of mtry

```
model3$results
```

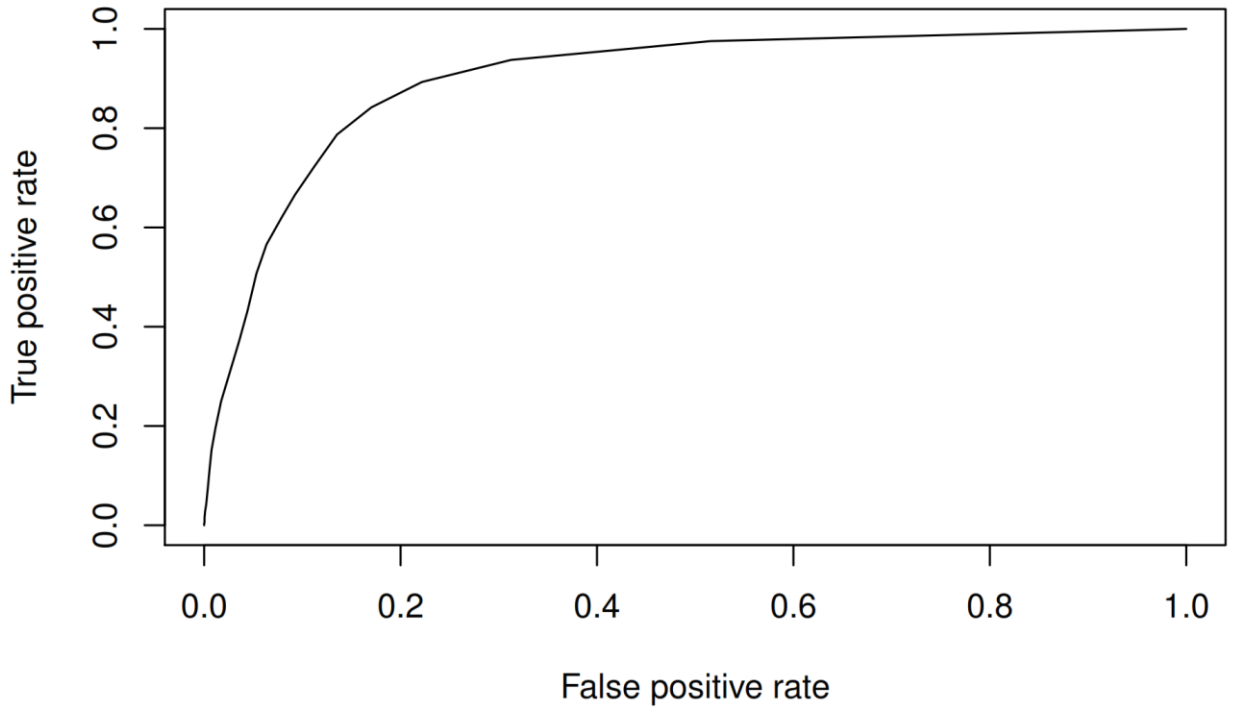
```
##   mtry  Accuracy     Kappa  AccuracySD    KappaSD
## 1    4 0.8949998 0.2655677 0.005357683 0.03993727
## 2    7 0.9009998 0.3795686 0.007322144 0.06390360
## 3    9 0.8951970 0.3606980 0.007710044 0.05652041
## 4   10 0.8982018 0.3879652 0.005577998 0.04050396
```

```
model3$bestTune
```

```
##   mtry
## 2    7
```

Performance

```
pr<-predict(model3, newdata=Test, type="prob")
p_test<-prediction(pr[,2], Test$y)
perf<-performance(p_test, "tpr", "fpr")
plot(perf)
```



Performance

Area Under the Curve

```
performance(p_test, "auc")@y.values
```

```
## [[1]]
## [1] 0.899353
```

Doing with R

- We can tell R to train the models and calculate ROCR measures rather than just accuracy
- Inside train control, specify classProbs=TRUE, and summaryFunction=twoClassSummary

```
set.seed(1)
trc <- trainControl(method = "cv",
                    classProbs = TRUE,
                    summaryFunction = twoClassSummary,
                    number = 5)
```

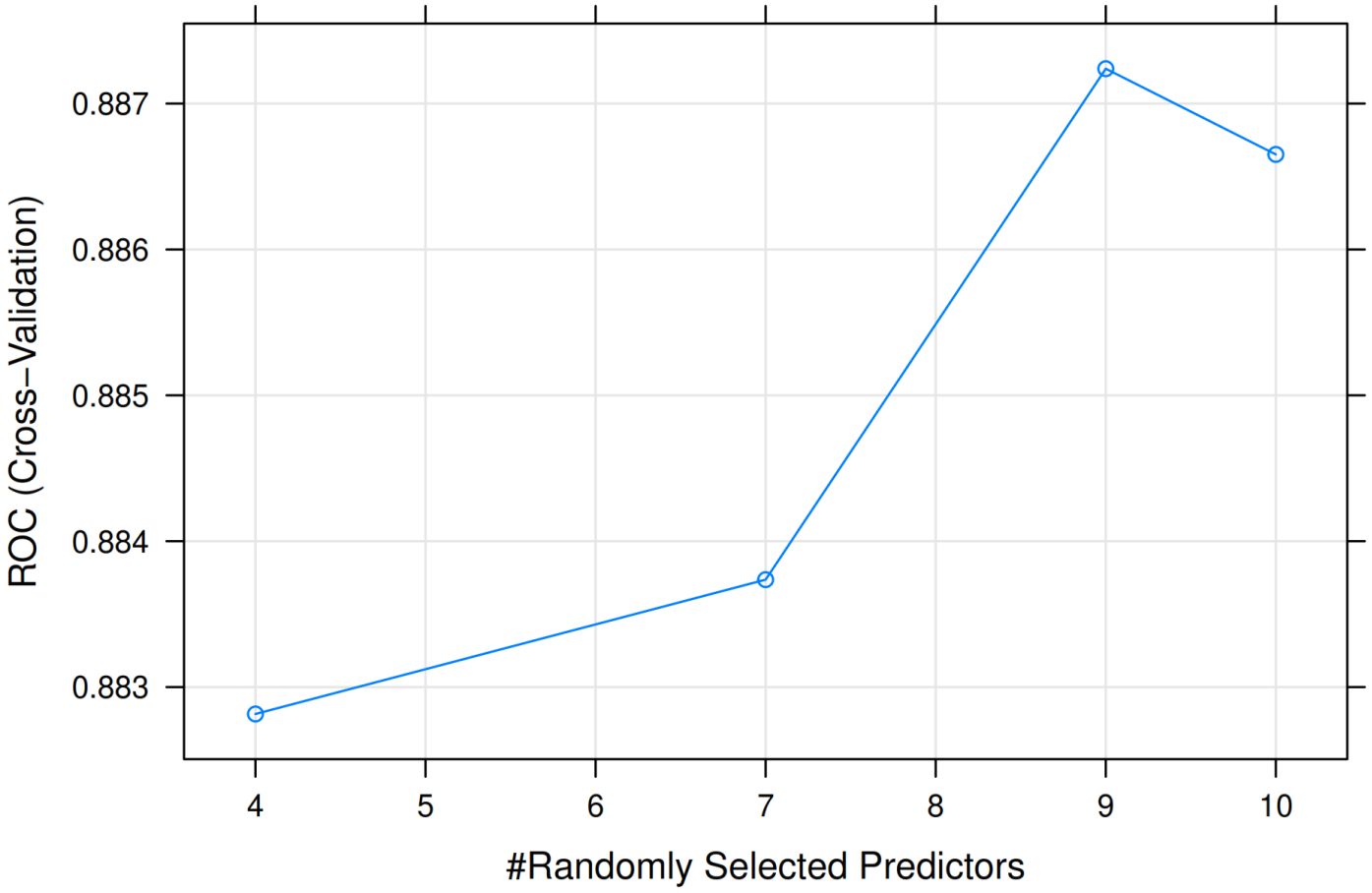
Doing in R

Inside the train function specify metric="ROC"

```
set.seed(1)
mm<-train(y~., data=Train1,
          trControl=trc,
          method="rf",
          ntree=25,
          metric="ROC",
          tuneGrid=mtry_grid)
```

Plot the AUC

```
plot(mm)
```



Importance

```
varImpPlot(mm$finalModel)
```