Ensemble learning

Ruxandra Stoean

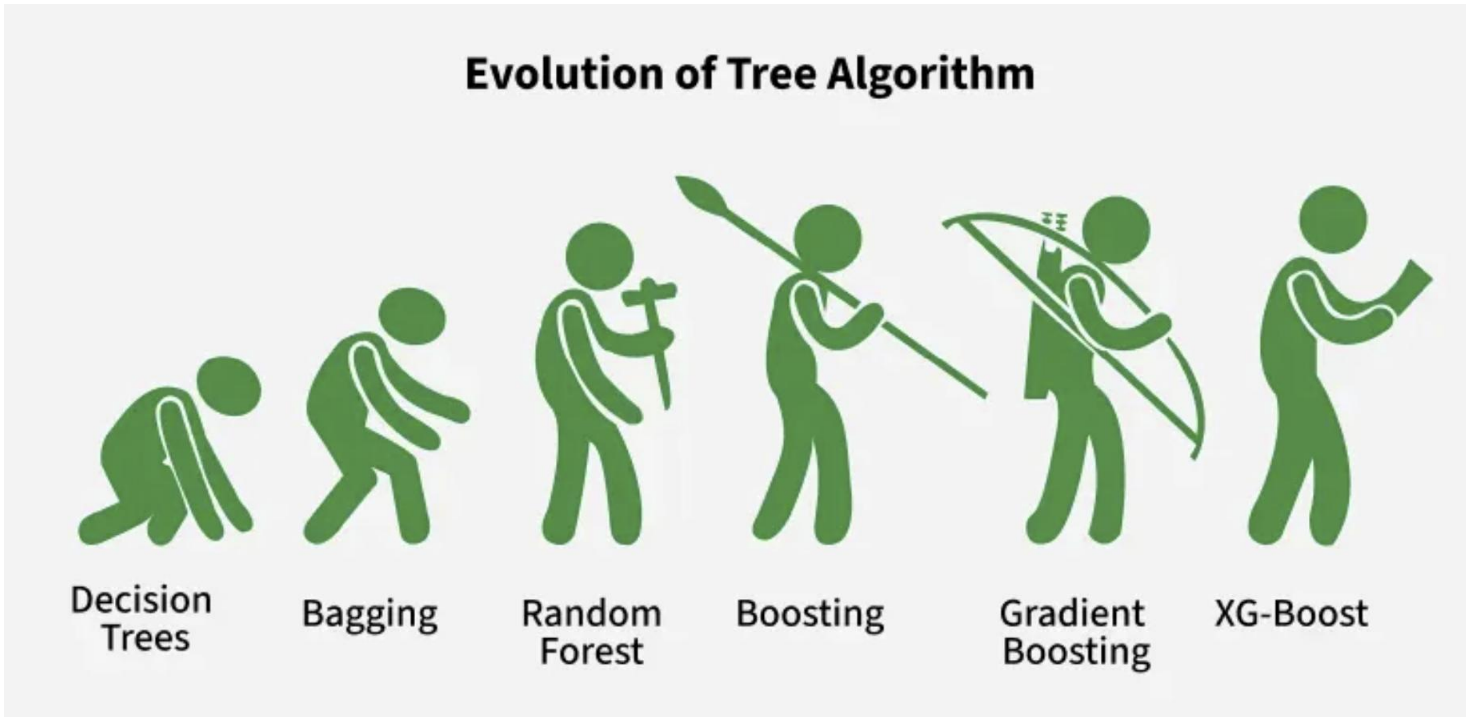
Further bibliography

- Leo Breiman, Bagging Predictors, Machine Learning , vol. 24, issue 2, pp. 123-140, 1996
- Yoav Freund, Rober E. Schapire, Experiments with a new boosting algorithm, 13th International Conference on Machine Learning, 148–156,Morgan Kaufmann, 1996
- Leo Breiman, Random Forests, Machine Learning 45(1), 5-32, 2001.

- Gautam Kunapuli, Ensemble Methods for Machine Learning, Manning, 2023

Ensemble learning

- Several base learners are trained and their predictions are combined.
- The way of application of the learners may take place at several levels.
- Classical approaches:
  - Bagging
  - Boosting
  - Random forests


Evolution of Tree Algorithm
Decision Trees
Bagging
Random Forest
Boosting
Gradient Boosting
XG-Boost

medium.com

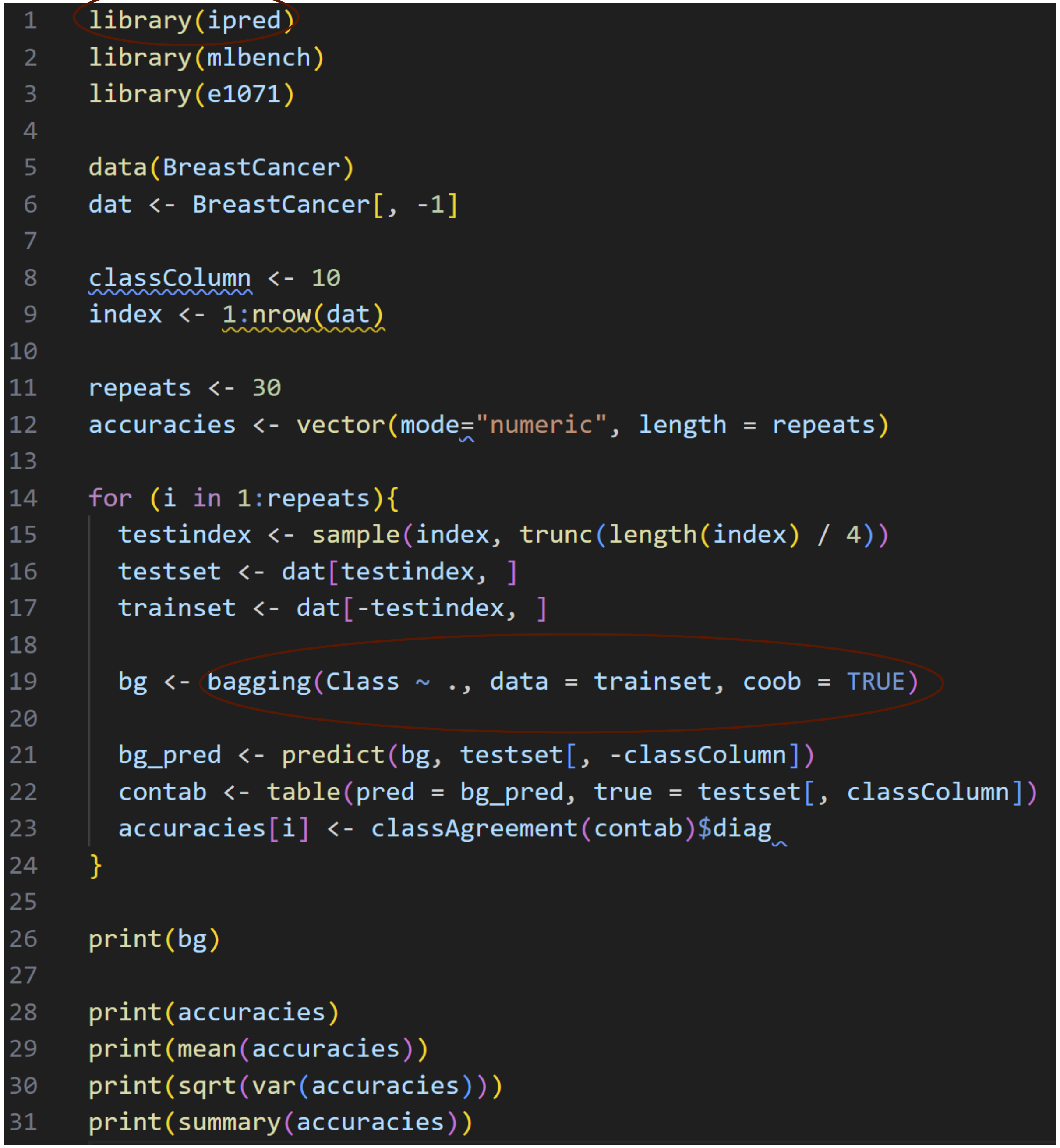
Bagging

- Short for Bootstrap AGGregatING.
- The training data set of *m* samples is split into *b* parts (bags).
    - Every subset has the same number of *m* elements.
    - The data from each subset are sampled with replacement.
- The base learner is trained on each subset.
- The *b* constructed models vote the output for new test data:
    - The mean of the outputs for the *b* models - for regression
    - The class with most appearances in the predictions of the *b* models - for classification
- Examples not selected (out-of-bag) are used to estimate the generalization error.

Package R ipred

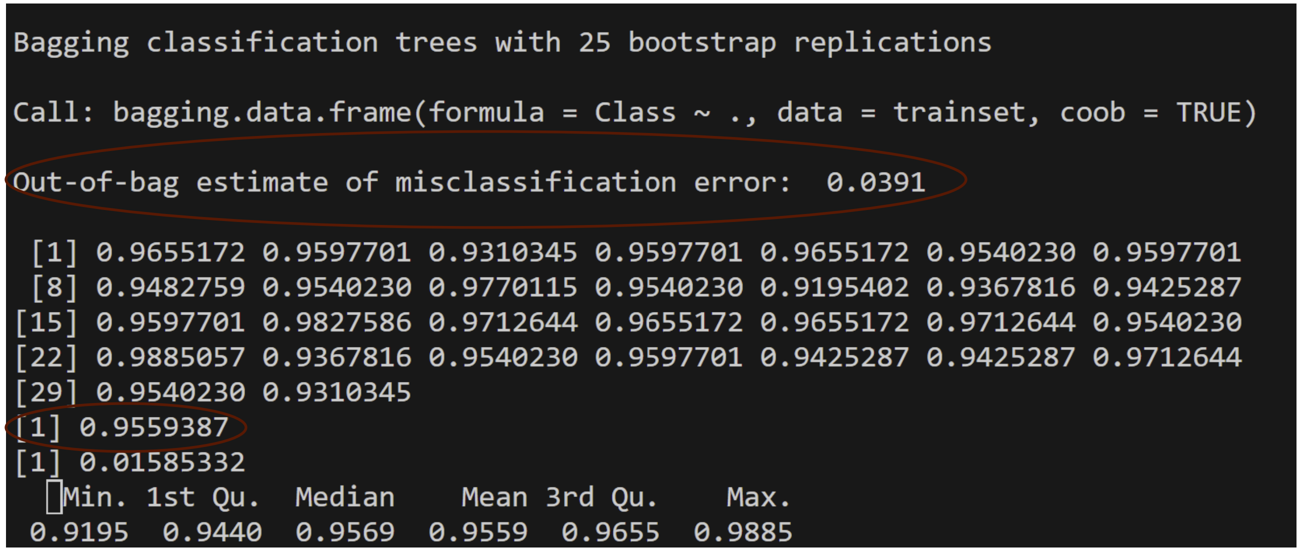
- The base learner is a decision tree (bagged tree).
  - In this sense package rpart is called.
- If the output is seen as factor, classification is considered.
  - If it is continuous, regression is in place.
- Parameter nbagg specifies the desired number of bags – default 25.
- *m* samples are taken in each bag
- Parameter coob=true specifies the scope to compute an estimation of the generalization error (out-of-bag estimation)

Code



```r
library(ipred)
library(mlbench)
library(e1071)

data(BreastCancer)
dat <- BreastCancer[, -1]

classColumn <- 10
index <- 1:nrow(dat)

repeats <- 30
accuracies <- vector(mode="numeric", length = repeats)

for (i in 1:repeats){
  testindex <- sample(index, trunc(length(index) / 4))
  testset <- dat[testindex, ]
  trainset <- dat[-testindex, ]

  bg <- bagging(Class ~ ., data = trainset, coob = TRUE)

  bg_pred <- predict(bg, testset[, -classColumn])
  contab <- table(pred = bg_pred, true = testset[, classColumn])
  accuracies[i] <- classAgreement(contab)$diag
}

print(bg)

print(accuracies)
print(mean(accuracies))
print(sqrt(var(accuracies)))
print(summary(accuracies))
```

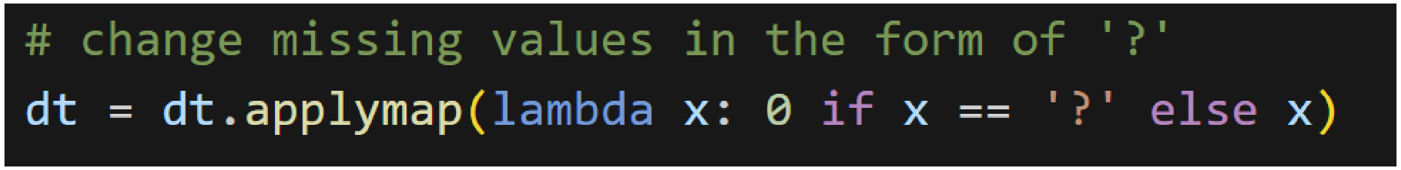
Result



```
Bagging classification trees with 25 bootstrap replications

Call: bagging.data.frame(formula = Class ~ ., data = trainset, coob = TRUE)

Out-of-bag estimate of misclassification error:  0.0391

 [1] 0.9655172 0.9597701 0.9310345 0.9597701 0.9655172 0.9540230 0.9597701
 [8] 0.9482759 0.9540230 0.9770115 0.9540230 0.9195402 0.9367816 0.9425287
[15] 0.9597701 0.9827586 0.9712644 0.9655172 0.9655172 0.9712644 0.9540230
[22] 0.9885057 0.9367816 0.9540230 0.9597701 0.9425287 0.9425287 0.9712644
[29] 0.9540230 0.9310345
[1] 0.9559387
[1] 0.01585332
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.9195  0.9440  0.9569  0.9559  0.9655  0.9885 
```

Python. Data processing WDBC

- For the Wisconsin original data set with 9 attributes and 699 records, we will have to read the .csv file.
- In the collection available from UCI, missing values are denoted by '?'
- Their transformation to 0 will be performed through the use of *applymap* and the anonymous *lambda* function.

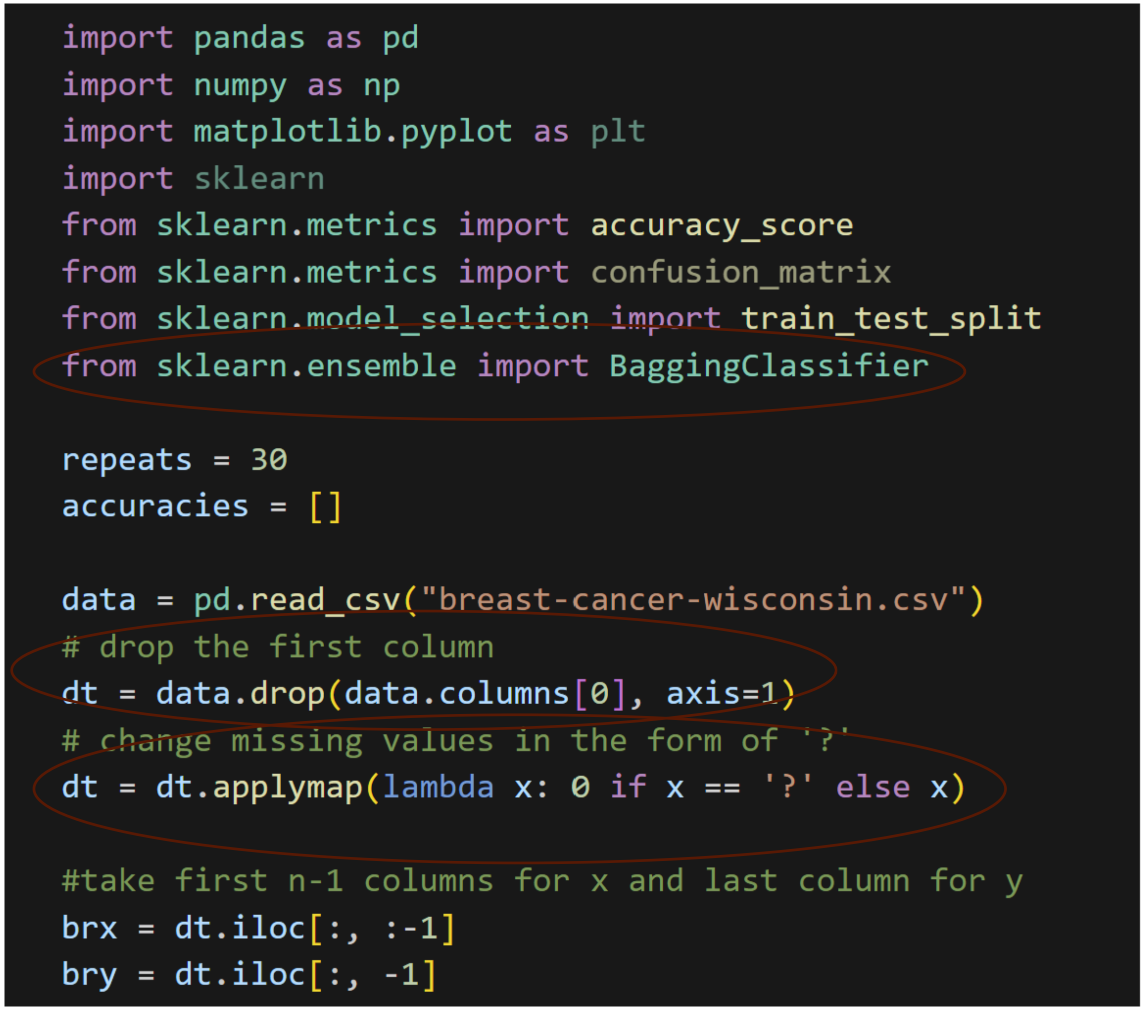


```
# change missing values in the form of '?'
dt = dt.applymap(lambda x: 0 if x == '?' else x)
```

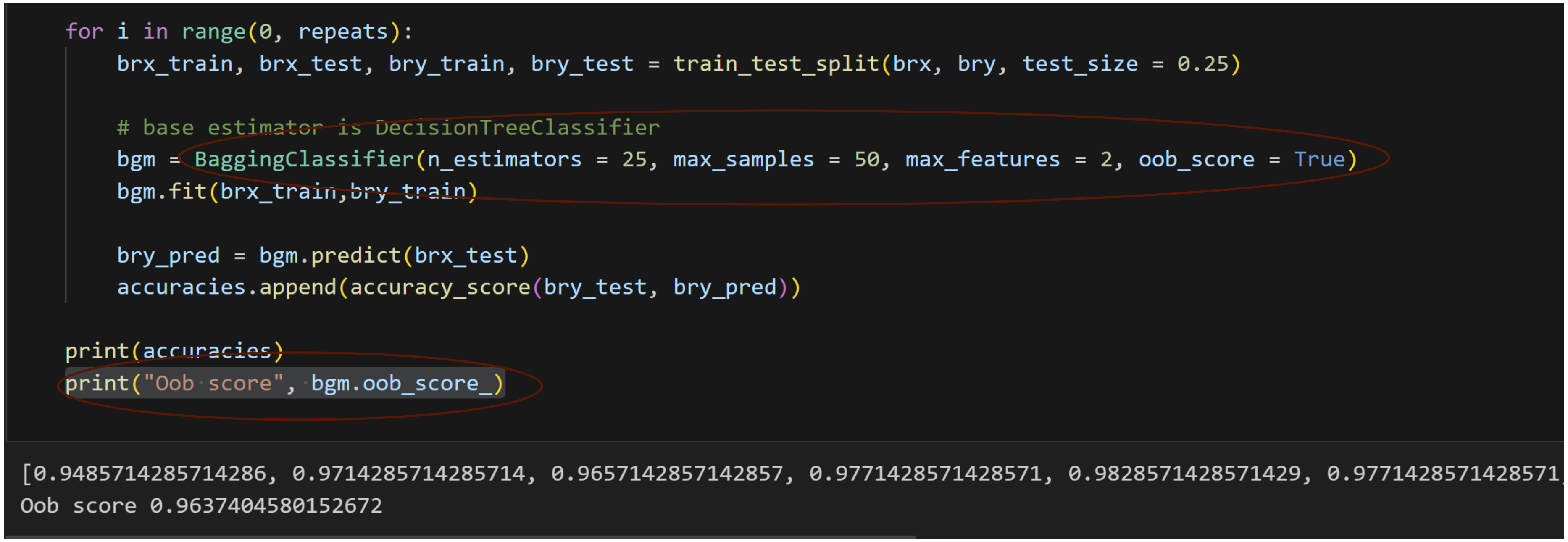
Library BaggingClassifier

- Base estimator is a DecisionTreeClassifier
- n_estimators gives the number of base classifiers in the ensemble
- max_samples denotes the number of records to draw (with replacement) in each bag
- Additionally in this Python library: max_features gives the number of attributes to draw (with replacement) for each classifier
- oob_score specifies whether to use out-of-bag samples to approximate generalization error.

WDBC reading and preprocessing

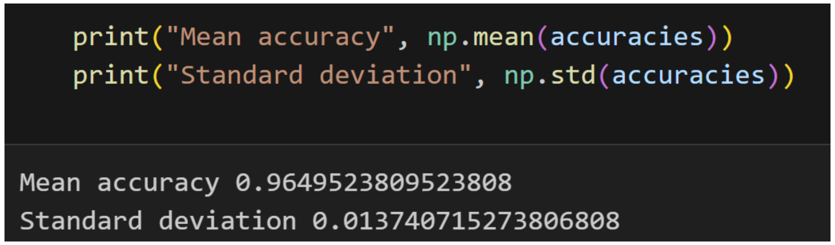

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier

repeats = 30
accuracies = []

data = pd.read_csv("breast-cancer-wisconsin.csv")
# drop the first column
dt = data.drop(data.columns[0], axis=1)
# change missing values in the form of '?'
dt = dt.applymap(lambda x: 0 if x == '?' else x)

#take first n-1 columns for x and last column for y
brx = dt.iloc[:, :-1]
bry = dt.iloc[:, -1]
```

Results



```
for i in range(0, repeats):
    brx_train, brx_test, bry_train, bry_test = train_test_split(brx, bry, test_size = 0.25)

    # base estimator is DecisionTreeClassifier
    bgm = BaggingClassifier(n_estimators = 25, max_samples = 50, max_features = 2, oob_score = True)
    bgm.fit(brx_train,bry_train)

    bry_pred = bgm.predict(brx_test)
    accuracies.append(accuracy_score(bry_test, bry_pred))

print(accuracies)
print("Oob score", bgm.oob_score_)
```

```
[0.9485714285714286, 0.9714285714285714, 0.9657142857142857, 0.9771428571428571, 0.9828571428571429, 0.9771428571428571
Oob score 0.9637404580152672
```



```
print("Mean accuracy", np.mean(accuracies))
print("Standard deviation", np.std(accuracies))
```

```
Mean accuracy 0.9649523809523808
Standard deviation 0.013740715273806808
```

Boosting: Adaboost – ADAptive BOOSTing

- Aims to progressively increase the performance for a learner.
- The base learner is applied repeatedly.
- Every training data point (of the m) has a weight attached – initially all equal to $1/m$.
- At every step, erroneous points get a higher weight.
- The model resulting at every step receives a vote weighted by its performance
  - The weight is given by the measure of its errors on the training set.

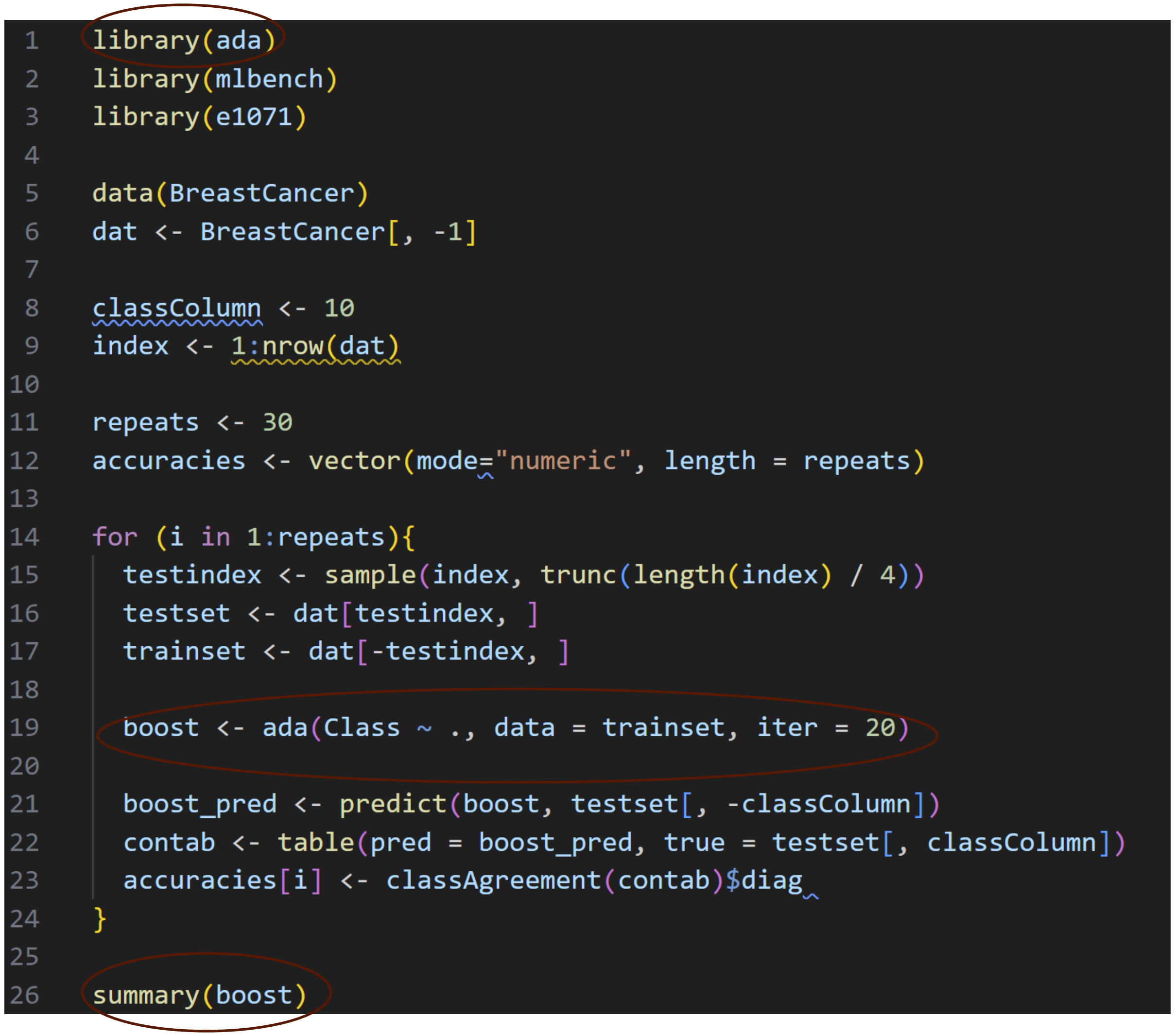
Package R ada

- Implements again a decision tree as base learner (boosted tree) – using the same package rpart.

- Parameter iter specifies the number of desired iterations.

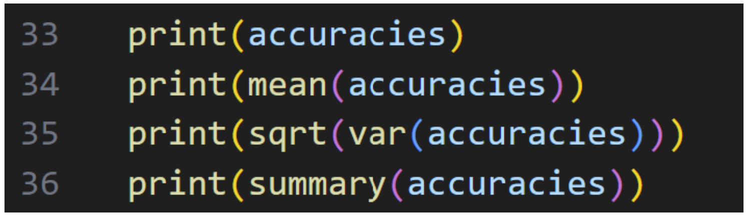
- The function plot offers the posibility of visualization:
  - The error at every iteration of the boosting algorithm
    - The error can be calculated both for the training and the test sets
  - The kappa agreement between the predicted and the real target at each boosting iteration for both sets

Code

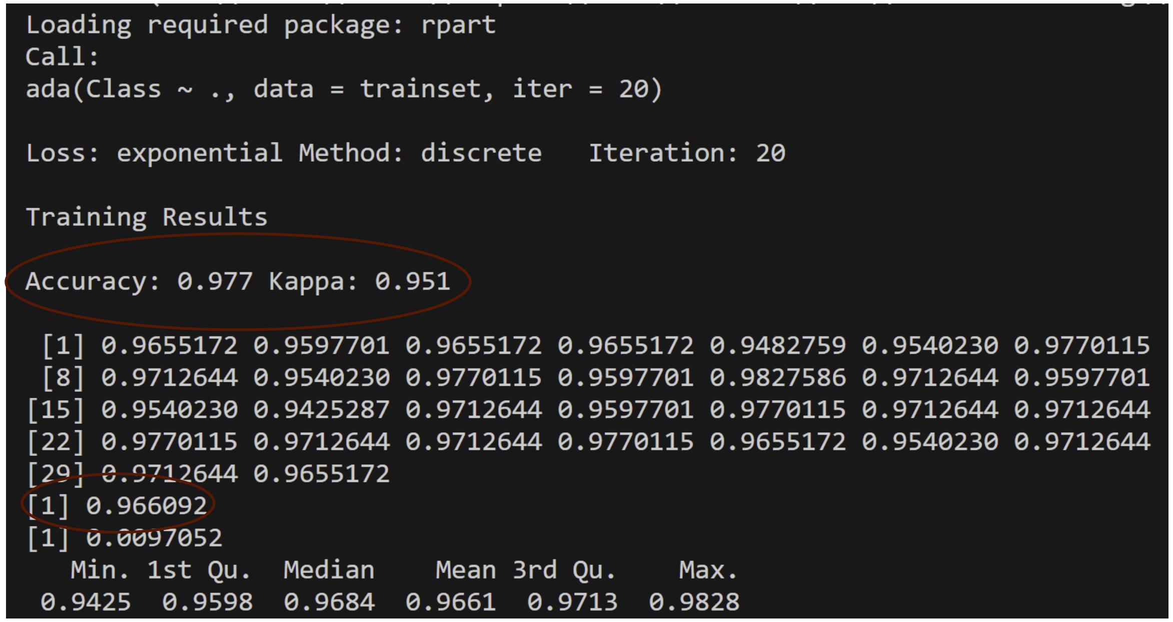


```r
library(ada)
library(mlbench)
library(e1071)

data(BreastCancer)
dat <- BreastCancer[, -1]

classColumn <- 10
index <- 1:nrow(dat)

repeats <- 30
accuracies <- vector(mode="numeric", length = repeats)

for (i in 1:repeats){
  testindex <- sample(index, trunc(length(index) / 4))
  testset <- dat[testindex, ]
  trainset <- dat[-testindex, ]

  boost <- ada(Class ~ ., data = trainset, iter = 20)

  boost_pred <- predict(boost, testset[, -classColumn])
  contab <- table(pred = boost_pred, true = testset[, classColumn])
  accuracies[i] <- classAgreement(contab)$diag
}

summary(boost)
```

Rezultate



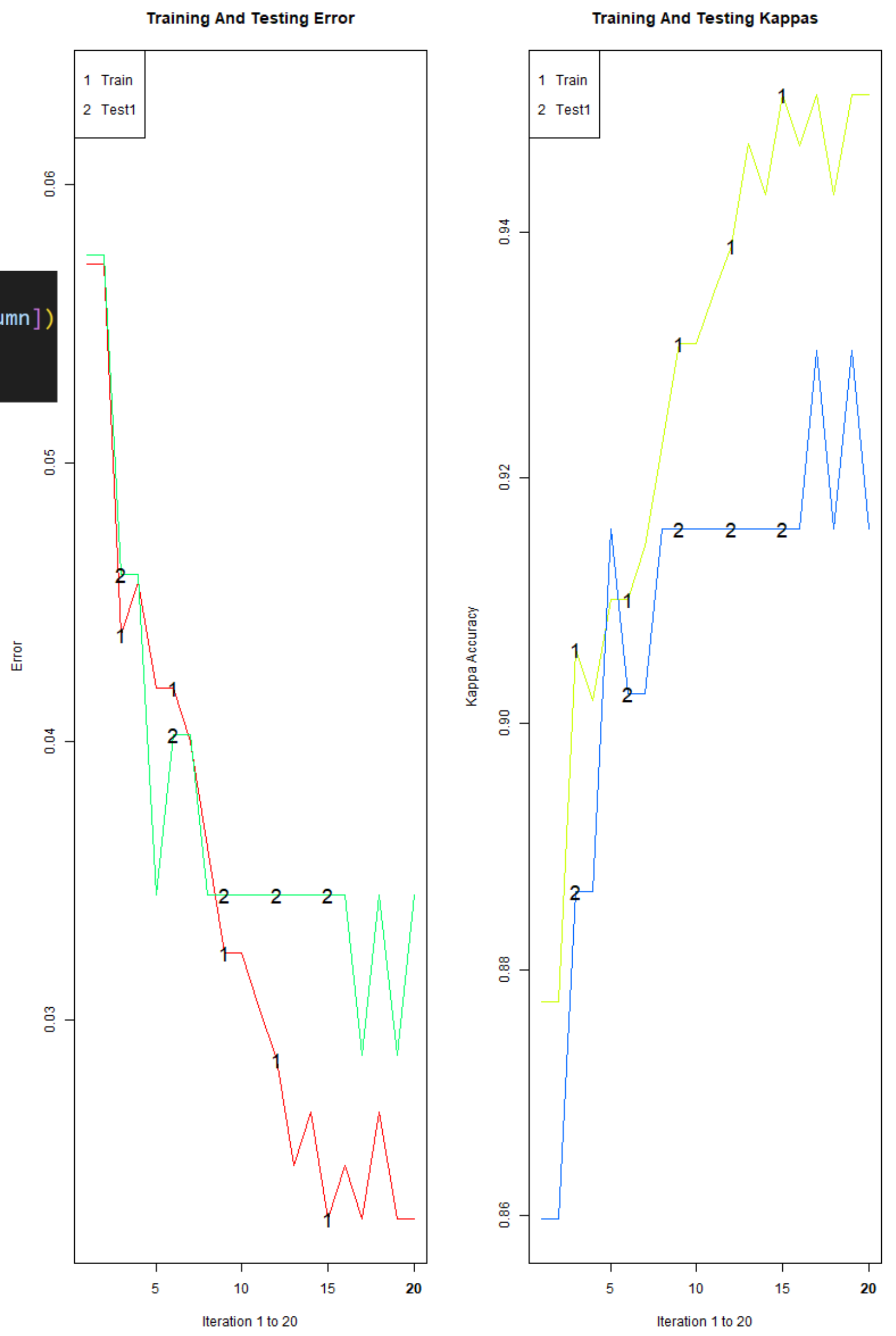
```r
print(accuracies)
print(mean(accuracies))
print(sqrt(var(accuracies)))
print(summary(accuracies))
```



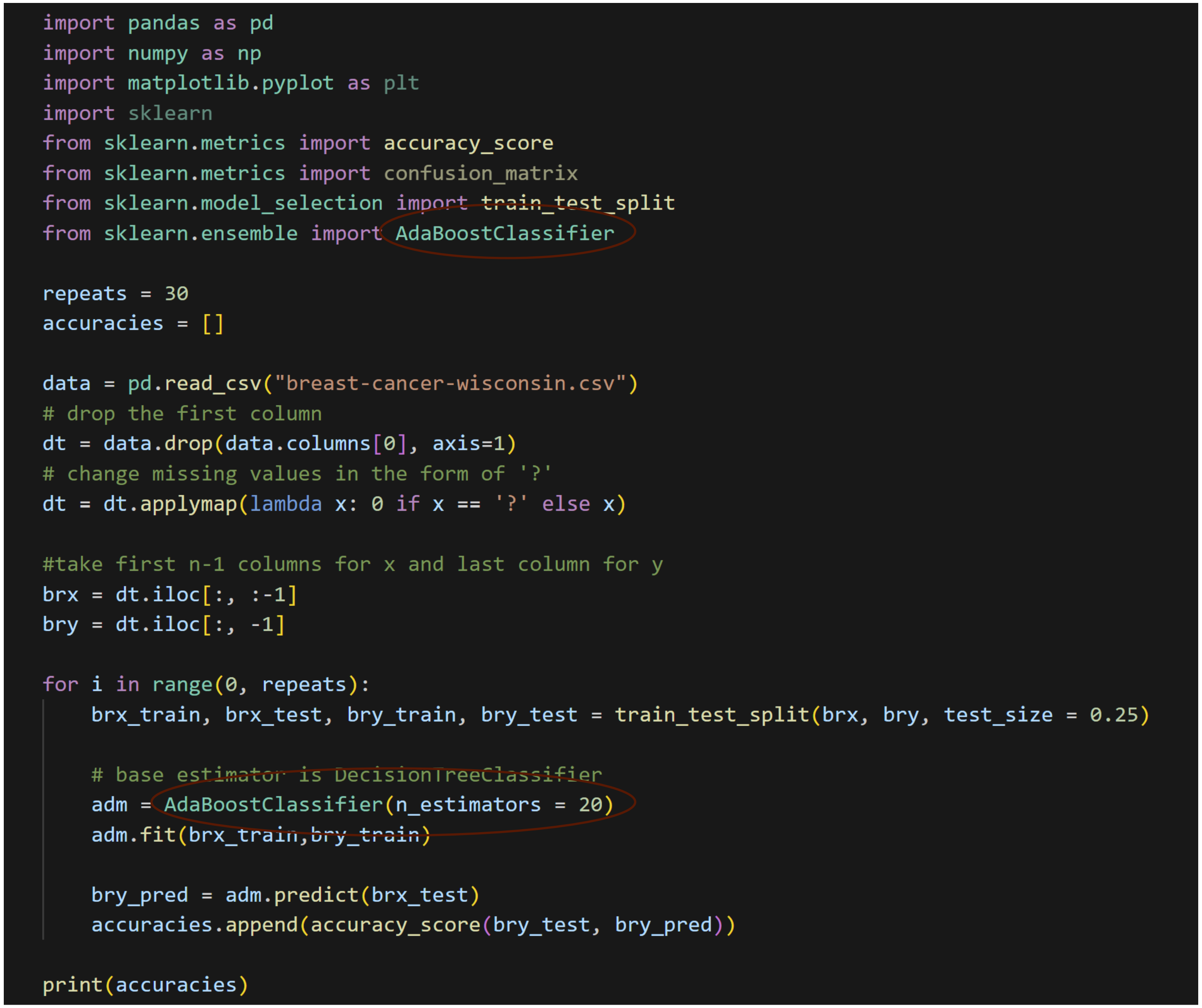
```
Loading required package: rpart
Call:
ada(Class ~ ., data = trainset, iter = 20)

Loss: exponential Method: discrete   Iteration: 20

Training Results

Accuracy: 0.977 Kappa: 0.951

 [1] 0.9655172 0.9597701 0.9655172 0.9655172 0.9482759 0.9540230 0.9770115
 [8] 0.9712644 0.9540230 0.9770115 0.9597701 0.9827586 0.9712644 0.9597701
[15] 0.9540230 0.9425287 0.9712644 0.9597701 0.9770115 0.9712644 0.9712644
[22] 0.9770115 0.9712644 0.9712644 0.9770115 0.9655172 0.9540230 0.9712644
[29] 0.9712644 0.9655172
[1] 0.966092
[1] 0.0097052
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.9425  0.9598  0.9684  0.9661  0.9713  0.9828 
```

Plot

```
##add testing data set
boost <- addtest(boost, testset[, -classColumn], testset[, classColumn])
##plot boost
plot(boost, TRUE, TRUE)
```

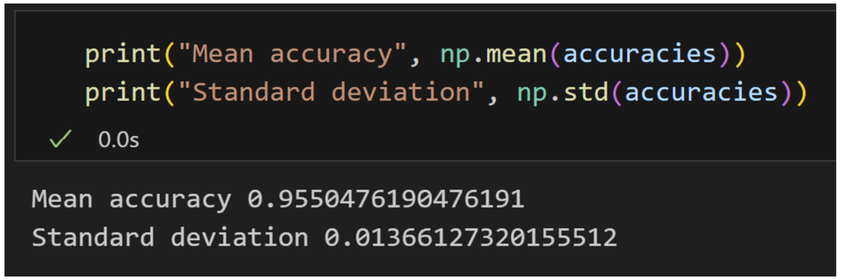


AdaBoostClassifier in Python



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier

repeats = 30
accuracies = []

data = pd.read_csv("breast-cancer-wisconsin.csv")
# drop the first column
dt = data.drop(data.columns[0], axis=1)
# change missing values in the form of '?'
dt = dt.applymap(lambda x: 0 if x == '?' else x)

#take first n-1 columns for x and last column for y
brx = dt.iloc[:, :-1]
bry = dt.iloc[:, -1]

for i in range(0, repeats):
    brx_train, brx_test, bry_train, bry_test = train_test_split(brx, bry, test_size = 0.25)

    # base estimator is DecisionTreeClassifier
    adm = AdaBoostClassifier(n_estimators = 20)
    adm.fit(brx_train,bry_train)

    bry_pred = adm.predict(brx_test)
    accuracies.append(accuracy_score(bry_test, bry_pred))

print(accuracies)
```

Results



```
print("Mean accuracy", np.mean(accuracies))
print("Standard deviation", np.std(accuracies))
```

✓ 0.0s

```
Mean accuracy 0.9550476190476191
Standard deviation 0.01366127320155512
```

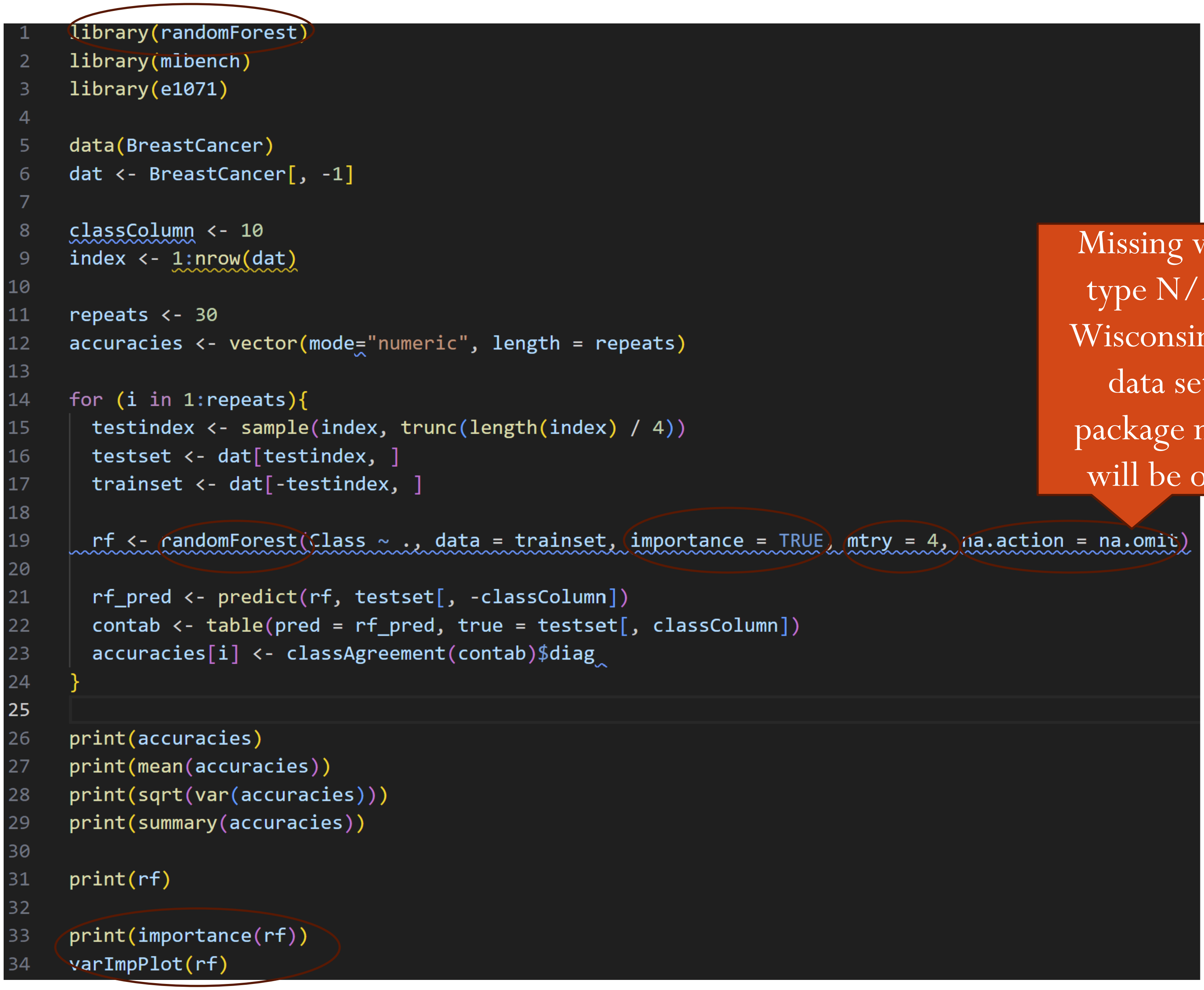
Random forests

- It is an ensemble method that combines several decision trees.
- Every tree is generated by random sampling (with replacement) of:
  - Attributes
  - Records
- For every tree there exists automatically
  - A training (in-bag) set
  - An out-of-bag set to measure the generalization error
- The class of a new example is established by vote for classification and by mean for regression.

Pachetul randomForest

- Parameters:
  - *ntree* – number of trees; implicitly equal to 500
  - *mtry* – number of randomly chosen attributes
  - *Gini* – the default split criterion
- Output:
  - An estimate of the generalization error – out-of-bag (OOB) estimate
  - Confusion matrix
  - Attribute importance

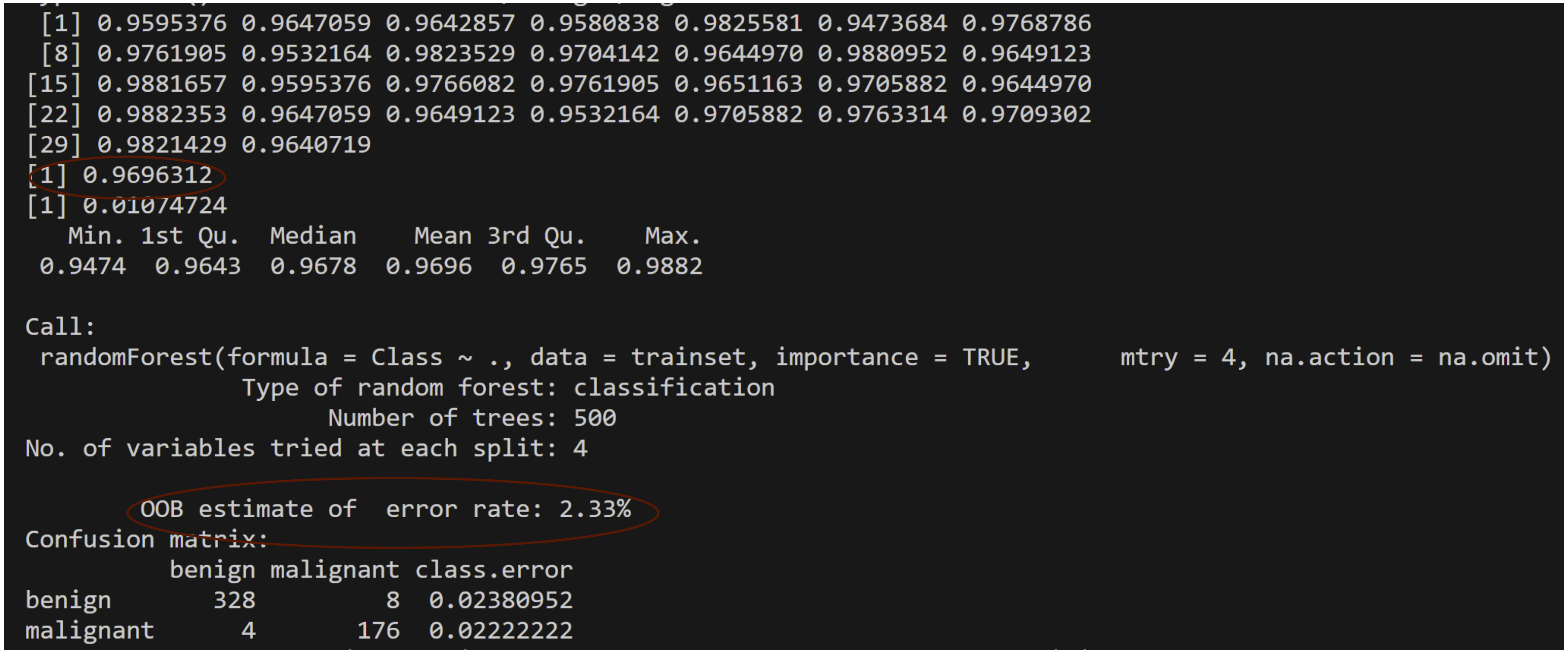
Classification Breast Cancer



```r
library(randomForest)
library(mlbench)
library(e1071)

data(BreastCancer)
dat <- BreastCancer[, -1]

classColumn <- 10
index <- 1:nrow(dat)

repeats <- 30
accuracies <- vector(mode="numeric", length = repeats)

for (i in 1:repeats){
  testindex <- sample(index, trunc(length(index) / 4))
  testset <- dat[testindex, ]
  trainset <- dat[-testindex, ]

  rf <- randomForest(Class ~ ., data = trainset, importance = TRUE, mtry = 4, na.action = na.omit)

  rf_pred <- predict(rf, testset[, -classColumn])
  contab <- table(pred = rf_pred, true = testset[, classColumn])
  accuracies[i] <- classAgreement(contab)$diag
}

print(accuracies)
print(mean(accuracies))
print(sqrt(var(accuracies)))
print(summary(accuracies))

print(rf)

print(importance(rf))
varImpPlot(rf)
```

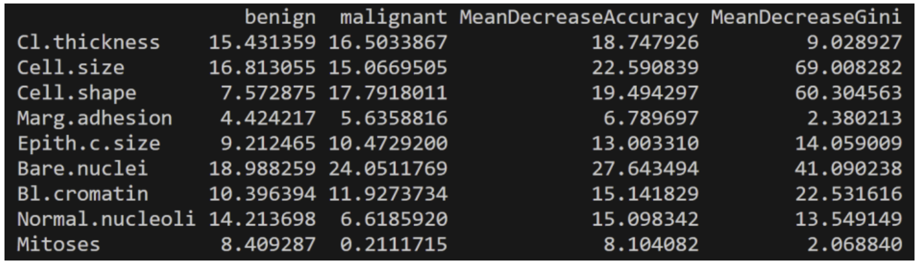
Missing values, as type N/A in the Wisconsin original data set from package mlbench, will be omitted.

Results



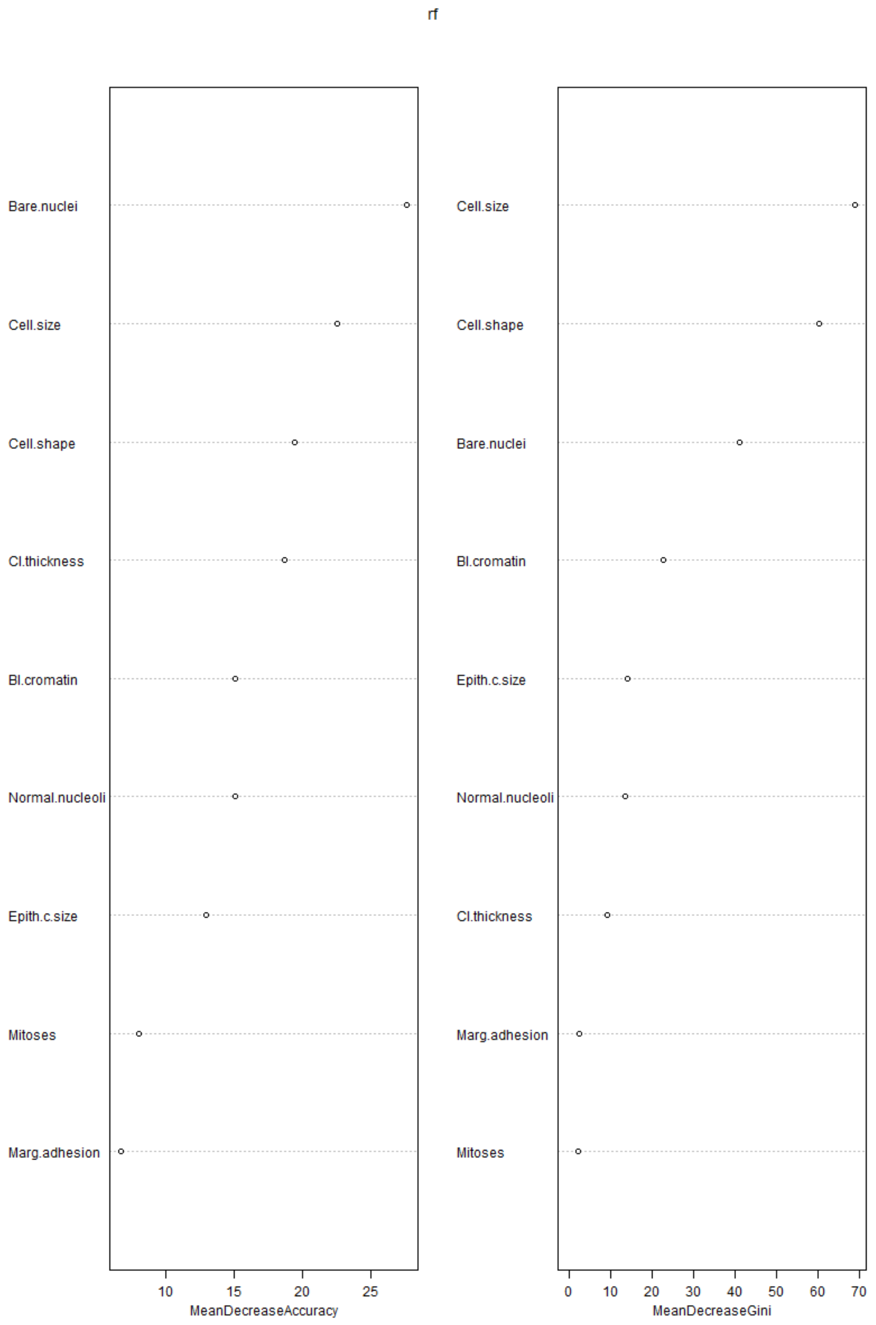
```
 [1] 0.9595376 0.9647059 0.9642857 0.9580838 0.9825581 0.9473684 0.9768786
 [8] 0.9761905 0.9532164 0.9823529 0.9704142 0.9644970 0.9880952 0.9649123
[15] 0.9881657 0.9595376 0.9766082 0.9761905 0.9651163 0.9705882 0.9644970
[22] 0.9882353 0.9647059 0.9649123 0.9532164 0.9705882 0.9763314 0.9709302
[29] 0.9821429 0.9640719
[1] 0.9696312
[1] 0.01074724
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.9474  0.9643  0.9678  0.9696  0.9765  0.9882 

Call:
 randomForest(formula = Class ~ ., data = trainset, importance = TRUE,      mtry = 4, na.action = na.omit) 
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 2.33%
Confusion matrix:
          benign malignant class.error
benign       328         8  0.02380952
malignant      4       176  0.02222222
```

Variable importance



```
                 benign  malignant MeanDecreaseAccuracy MeanDecreaseGini
Cl.thickness     15.431359 16.5033867            18.747926         9.028927
Cell.size        16.813055 15.0669505            22.590839        69.008282
Cell.shape        7.572875 17.7918011            19.494297        60.304563
Marg.adhesion     4.424217  5.6358816             6.789697         2.380213
Epith.c.size      9.212465 10.4729200            13.003310        14.059009
Bare.nuclei      18.988259 24.0511769            27.643494        41.090238
Bl.cromatin      10.396394 11.9273734            15.141829        22.531616
Normal.nucleoli  14.213698  6.6185920            15.098342        13.549149
Mitoses           8.409287  0.2111715             8.104082         2.068840
```

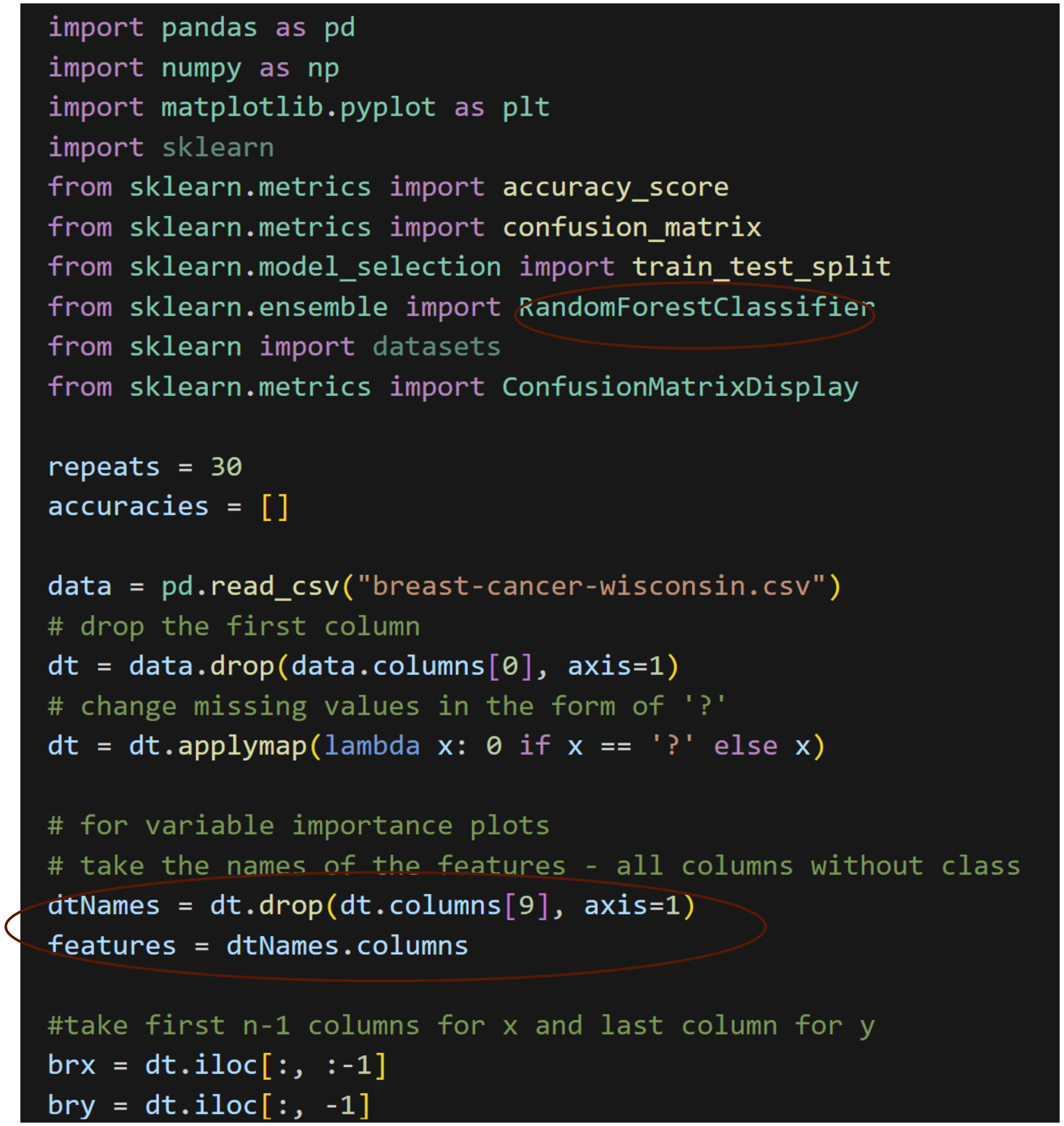
- *Mean decrease in* accuracy on OOB data and shows the amount of loss in accuracy if the variable values are permuted (total and per class)
- *Mean impurity decrease (Gini)* shows the contribution of the variable to the homogeneity of the forest



Library RandomForestClassifier

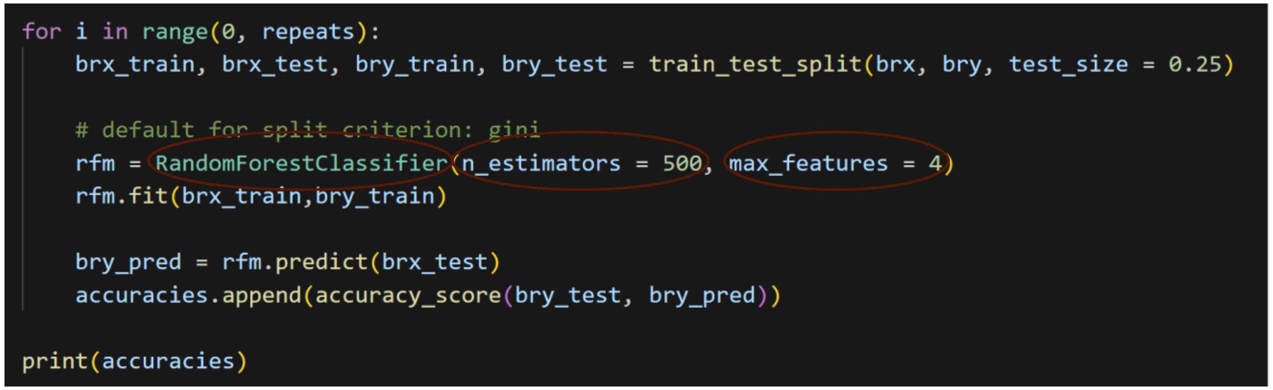
- *n_estimators* to specify number of trees
- *max_features* to indicate number of features randomly chosen as candidates for each split
- *criterion* to give the desired split modality – default Gini

WDBC reading and preprocessing

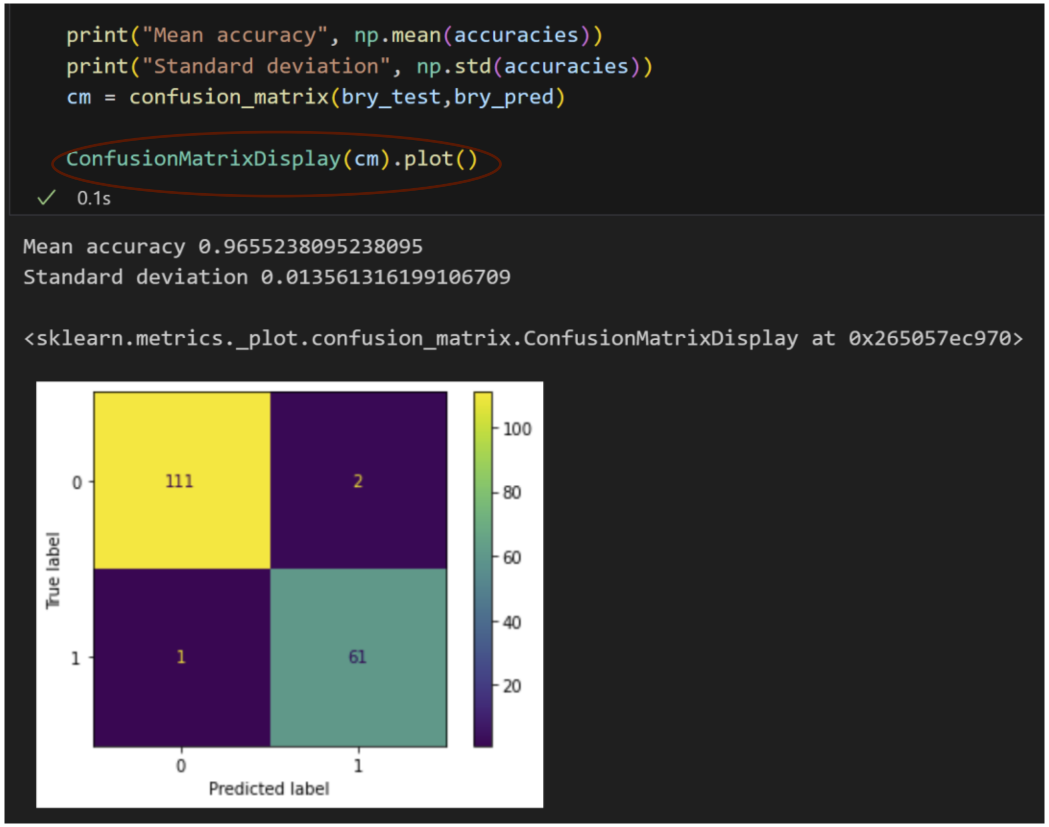


```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn.metrics import ConfusionMatrixDisplay

repeats = 30
accuracies = []

data = pd.read_csv("breast-cancer-wisconsin.csv")
# drop the first column
dt = data.drop(data.columns[0], axis=1)
# change missing values in the form of '?'
dt = dt.applymap(lambda x: 0 if x == '?' else x)

# for variable importance plots
# take the names of the features - all columns without class
dtNames = dt.drop(dt.columns[9], axis=1)
features = dtNames.columns

#take first n-1 columns for x and last column for y
brx = dt.iloc[:, :-1]
bry = dt.iloc[:, -1]
```

RF Classifier in Python



```
for i in range(0, repeats):
    brx_train, brx_test, bry_train, bry_test = train_test_split(brx, bry, test_size = 0.25)

    # default for split criterion: gini
    rfm = RandomForestClassifier(n_estimators = 500, max_features = 4)
    rfm.fit(brx_train,bry_train)

    bry_pred = rfm.predict(brx_test)
    accuracies.append(accuracy_score(bry_test, bry_pred))

print(accuracies)
```



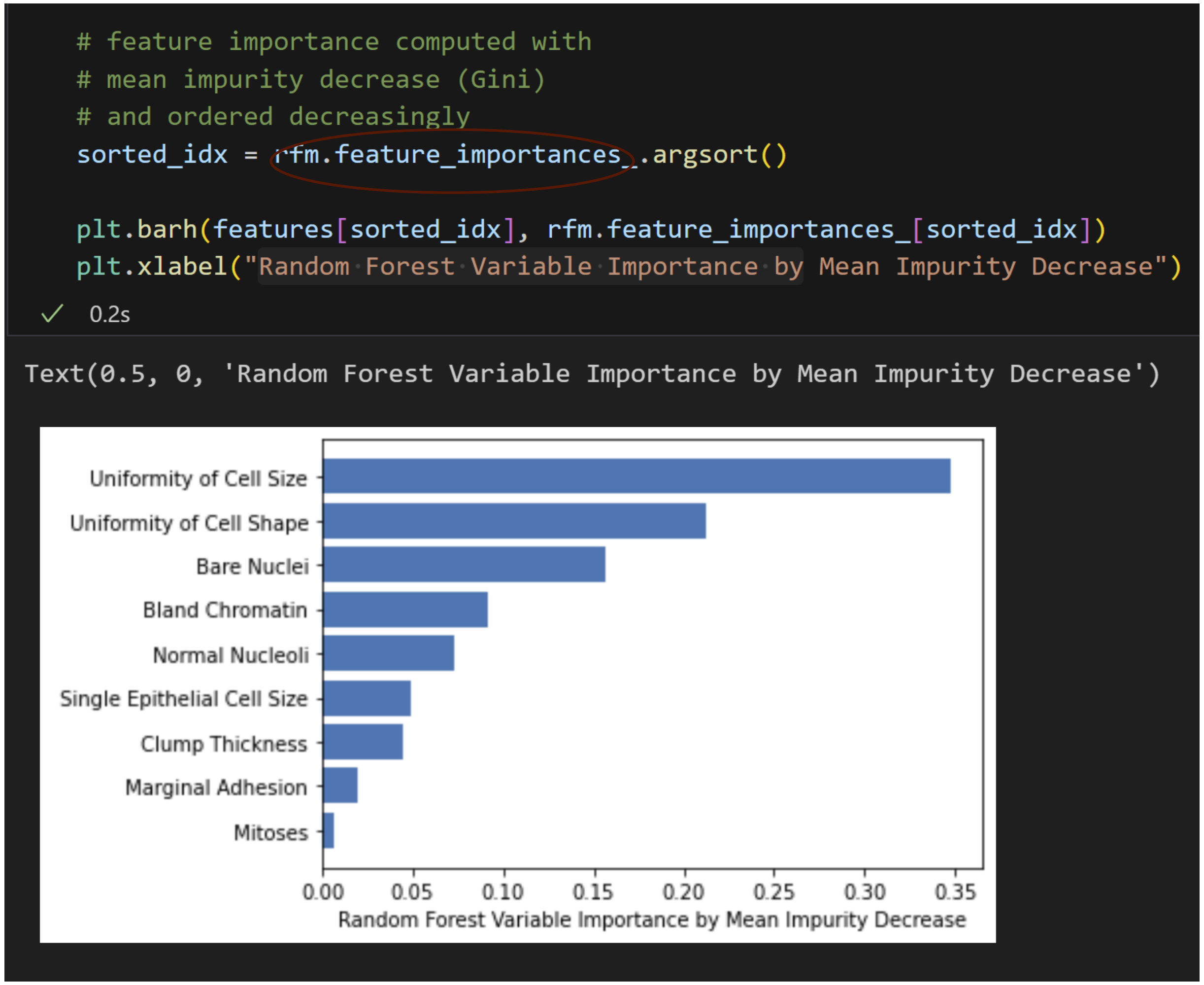
```
print("Mean accuracy", np.mean(accuracies))
print("Standard deviation", np.std(accuracies))
cm = confusion_matrix(bry_test,bry_pred)

ConfusionMatrixDisplay(cm).plot()
✓ 0.1s
```

```
Mean accuracy 0.9655238095238095
Standard deviation 0.013561316199106709

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x265057ec970>
```

Variable importance

- Impurity-based importance
  - based on training set statistics (Gini)
  - biased towards high cardinality (favors numerical variables)
- Permutation importance
  - computed on a held-out test set
  - measures the change in performance when each variable is shuffled randomly
  - shows the generalization ability of the variable for prediction on the test data

Impurity-based importance

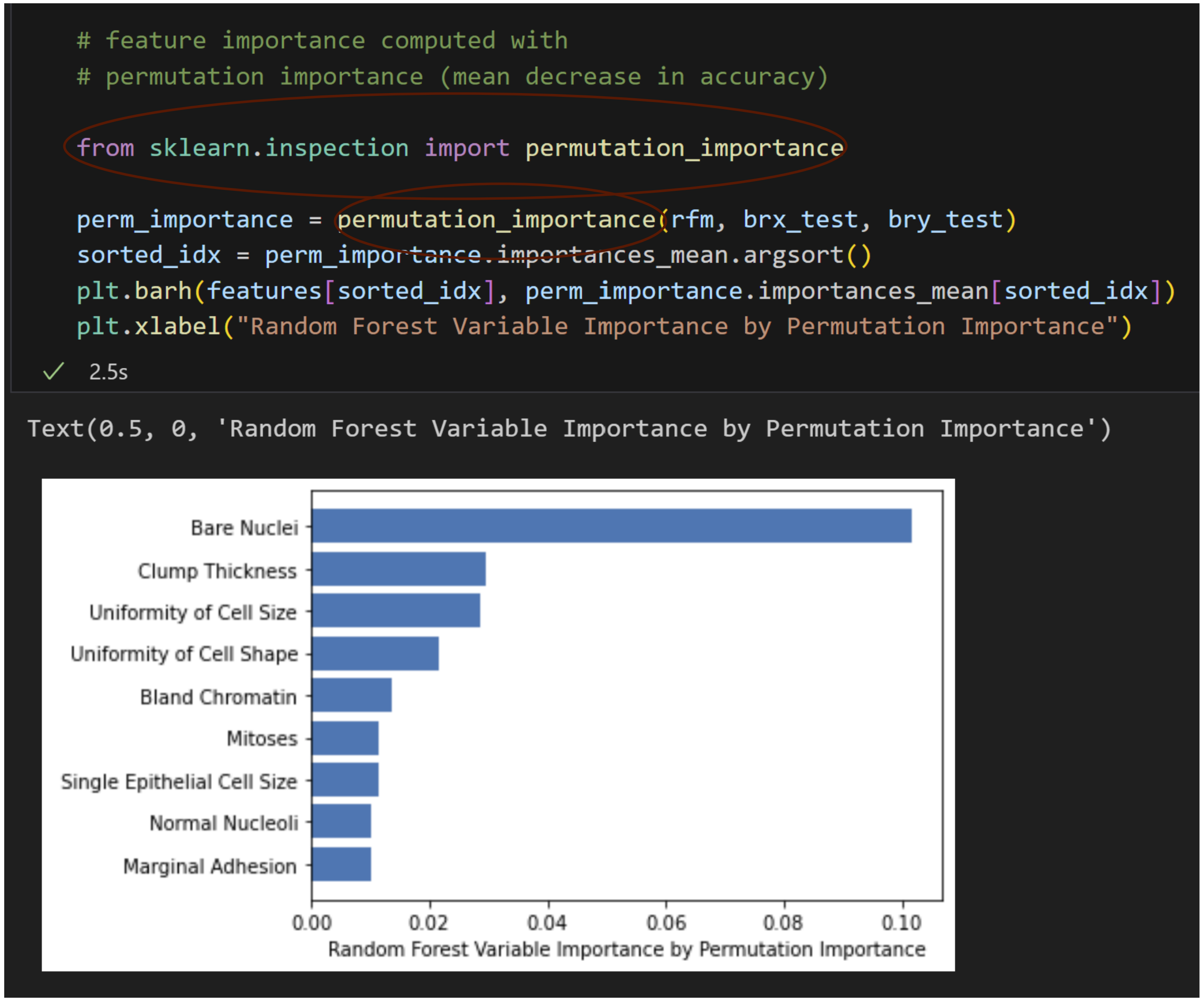


```
# feature importance computed with
# mean impurity decrease (Gini)
# and ordered decreasingly
sorted_idx = rfm.feature_importances_.argsort()

plt.barh(features[sorted_idx], rfm.feature_importances_[sorted_idx])
plt.xlabel("Random Forest Variable Importance by Mean Impurity Decrease")
```

✓ 0.2s

```
Text(0.5, 0, 'Random Forest Variable Importance by Mean Impurity Decrease')
```

Permutation importance

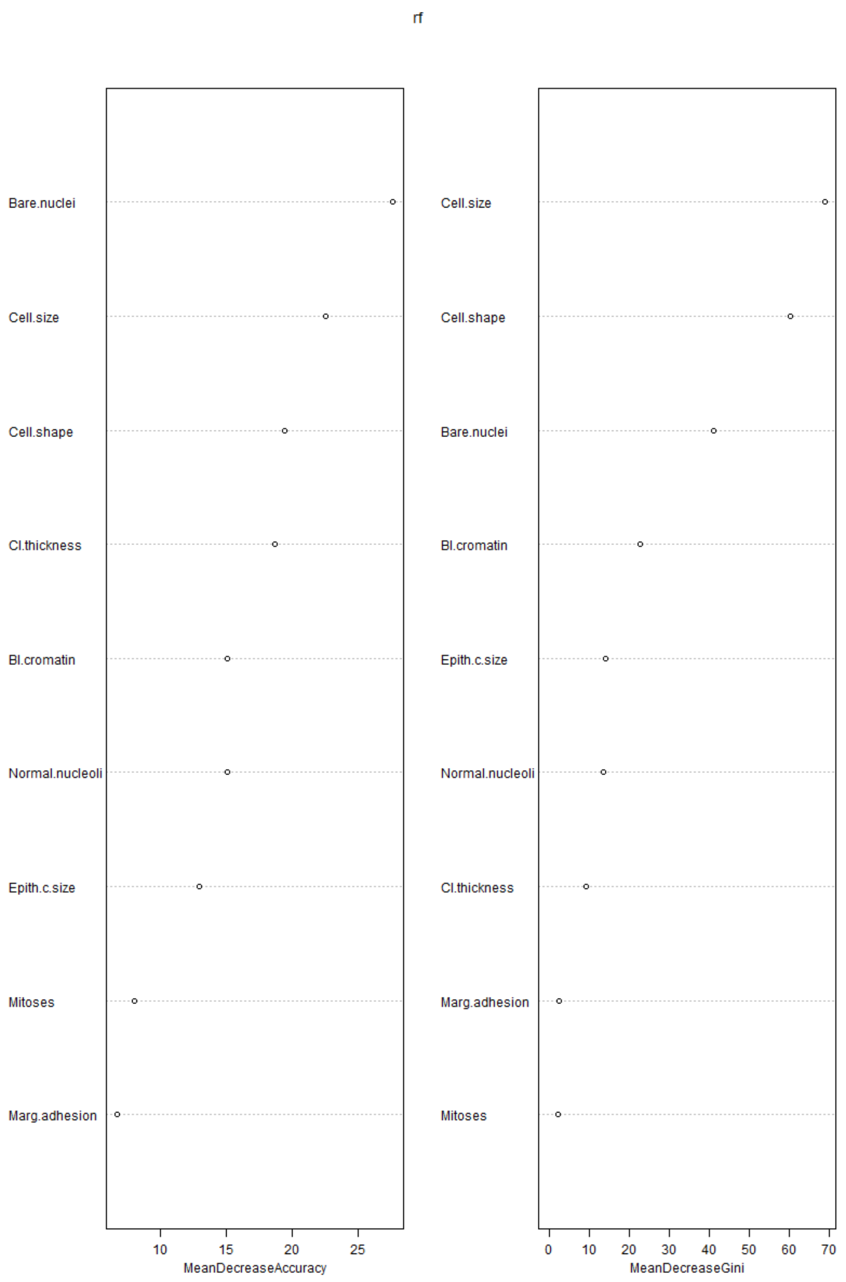


```
# feature importance computed with
# permutation importance (mean decrease in accuracy)

from sklearn.inspection import permutation_importance

perm_importance = permutation_importance(rfm, brx_test, bry_test)
sorted_idx = perm_importance.importances_mean.argsort()
plt.barh(features[sorted_idx], perm_importance.importances_mean[sorted_idx])
plt.xlabel("Random Forest Variable Importance by Permutation Importance")
```

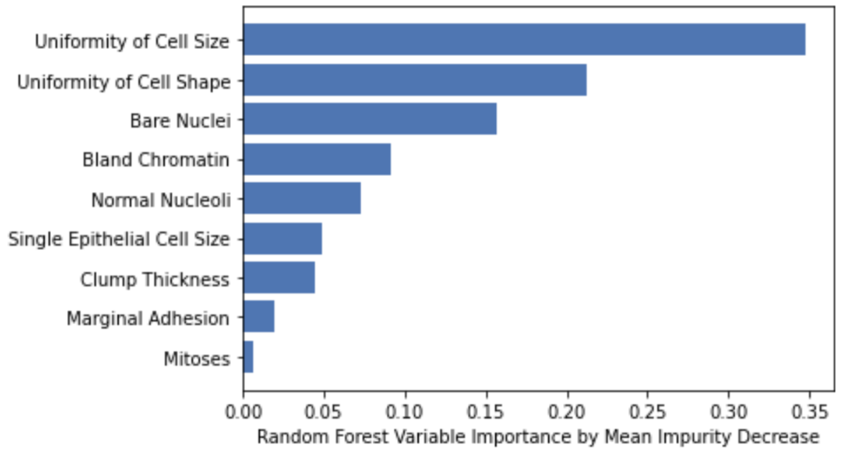
✓ 2.5s

```
Text(0.5, 0, 'Random Forest Variable Importance by Permutation Importance')
```

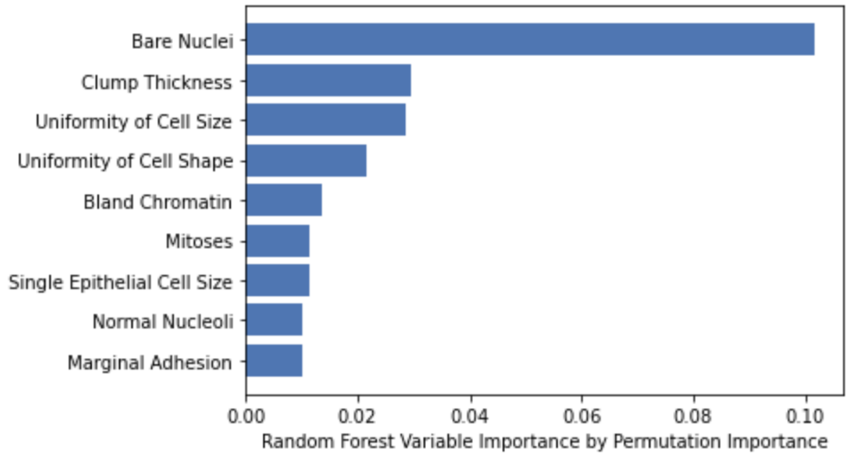
R vs Python



rf
Bare.nuclei
Cell.size
Cell.shape
Cl.thickness
Bl.cromatin
Normal.nucleoli
Epith.c.size
Mitoses
Marg.adhesion
10
15
20
25
MeanDecreaseAccuracy
Cell.size
Cell.shape
Bare.nuclei
Bl.cromatin
Epith.c.size
Normal.nucleoli
Cl.thickness
Marg.adhesion
Mitoses
0
10
20
30
40
50
60
70
MeanDecreaseGini

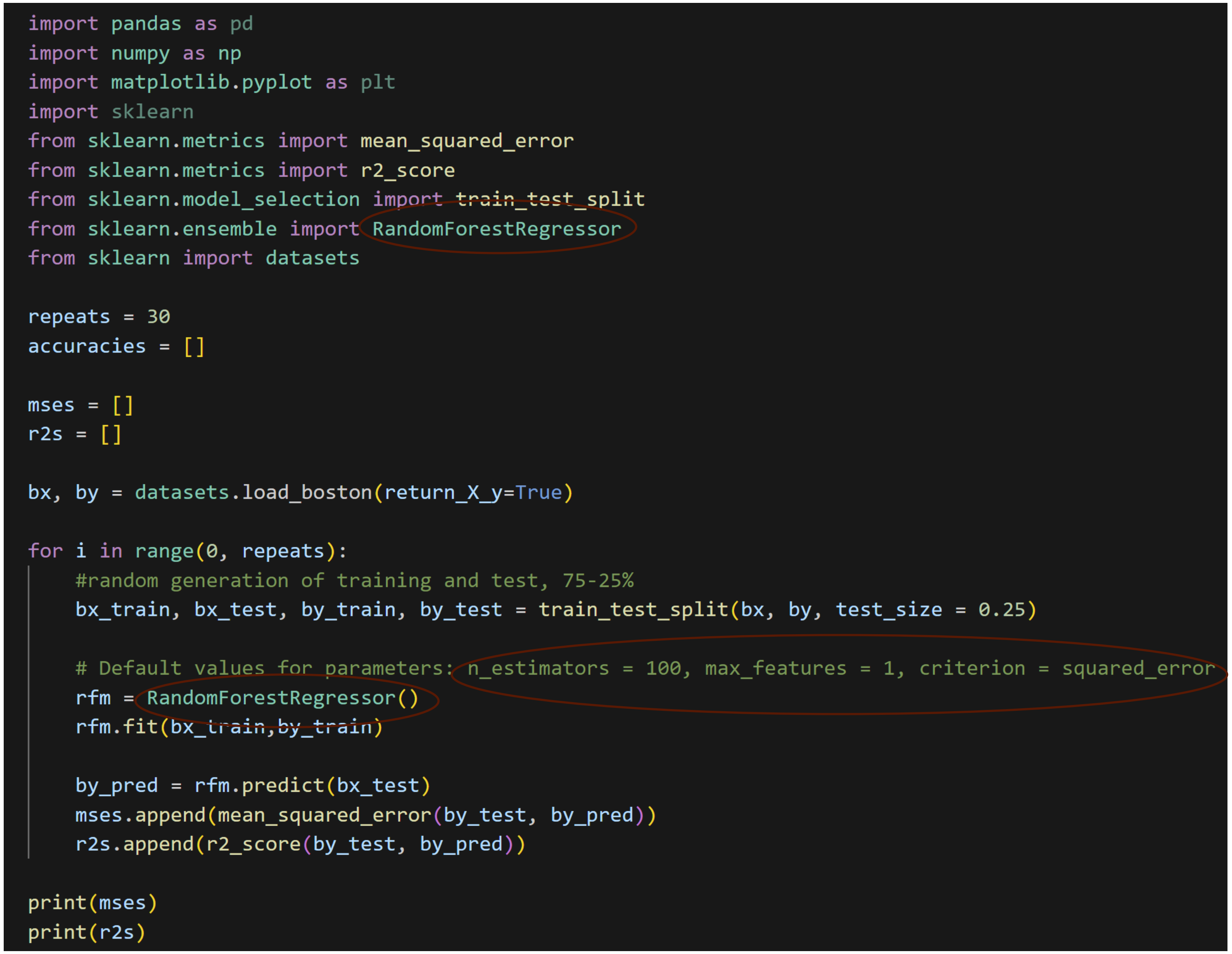


Uniformity of Cell Size
Uniformity of Cell Shape
Bare Nuclei
Bland Chromatin
Normal Nucleoli
Single Epithelial Cell Size
Clump Thickness
Marginal Adhesion
Mitoses
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Random Forest Variable Importance by Mean Impurity Decrease



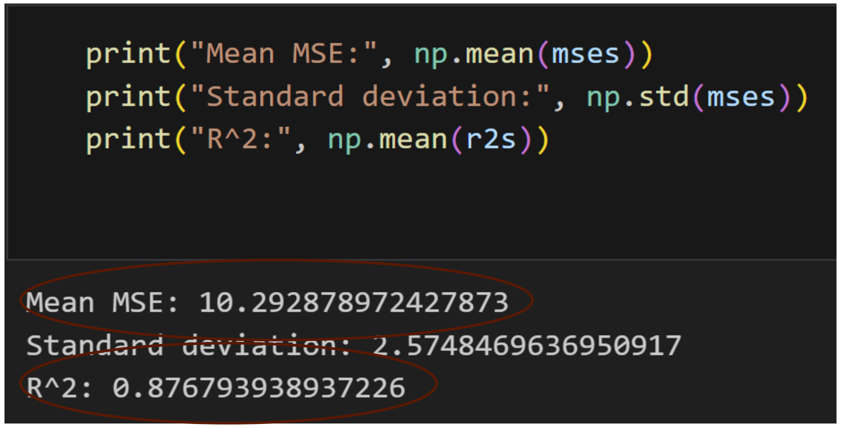
Bare Nuclei
Clump Thickness
Uniformity of Cell Size
Uniformity of Cell Shape
Bland Chromatin
Mitoses
Single Epithelial Cell Size
Normal Nucleoli
Marginal Adhesion
0.00
0.02
0.04
0.06
0.08
0.10
Random Forest Variable Importance by Permutation Importance

RF Regressor in Python



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn import datasets

repeats = 30
accuracies = []

mses = []
r2s = []

bx, by = datasets.load_boston(return_X_y=True)

for i in range(0, repeats):
    #random generation of training and test, 75-25%
    bx_train, bx_test, by_train, by_test = train_test_split(bx, by, test_size = 0.25)

    # Default values for parameters: n_estimators = 100, max_features = 1, criterion = squared_error
    rfm = RandomForestRegressor()
    rfm.fit(bx_train,by_train)

    by_pred = rfm.predict(bx_test)
    mses.append(mean_squared_error(by_test, by_pred))
    r2s.append(r2_score(by_test, by_pred))

print(mses)
print(r2s)
```

Results

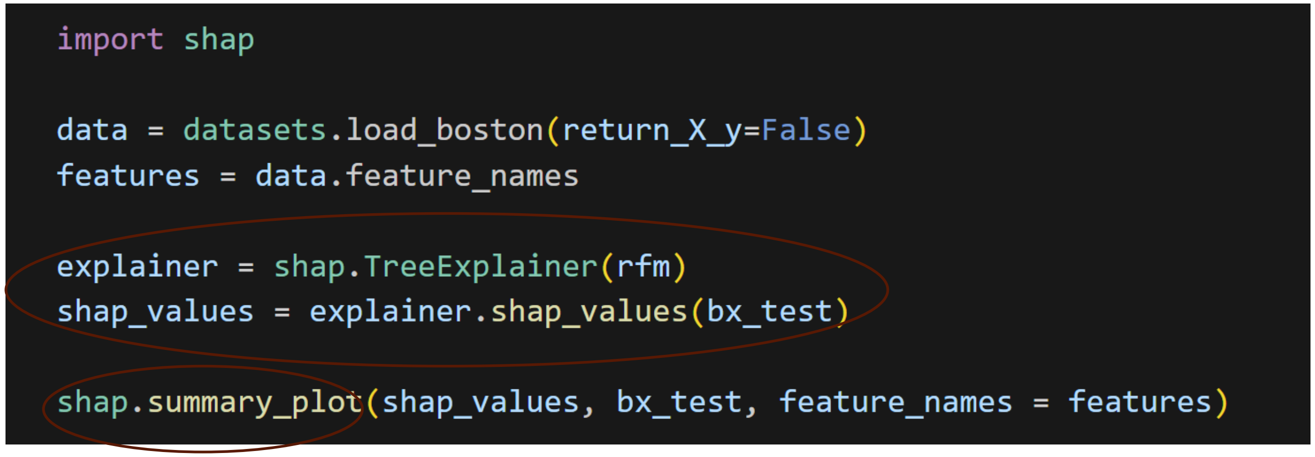


```
print("Mean MSE:", np.mean(mses))
print("Standard deviation:", np.std(mses))
print("R^2:", np.mean(r2s))
```

```
Mean MSE: 10.292878972427873
Standard deviation: 2.5748469636950917
R^2: 0.876793938937226
```

Feature importance - SHAP

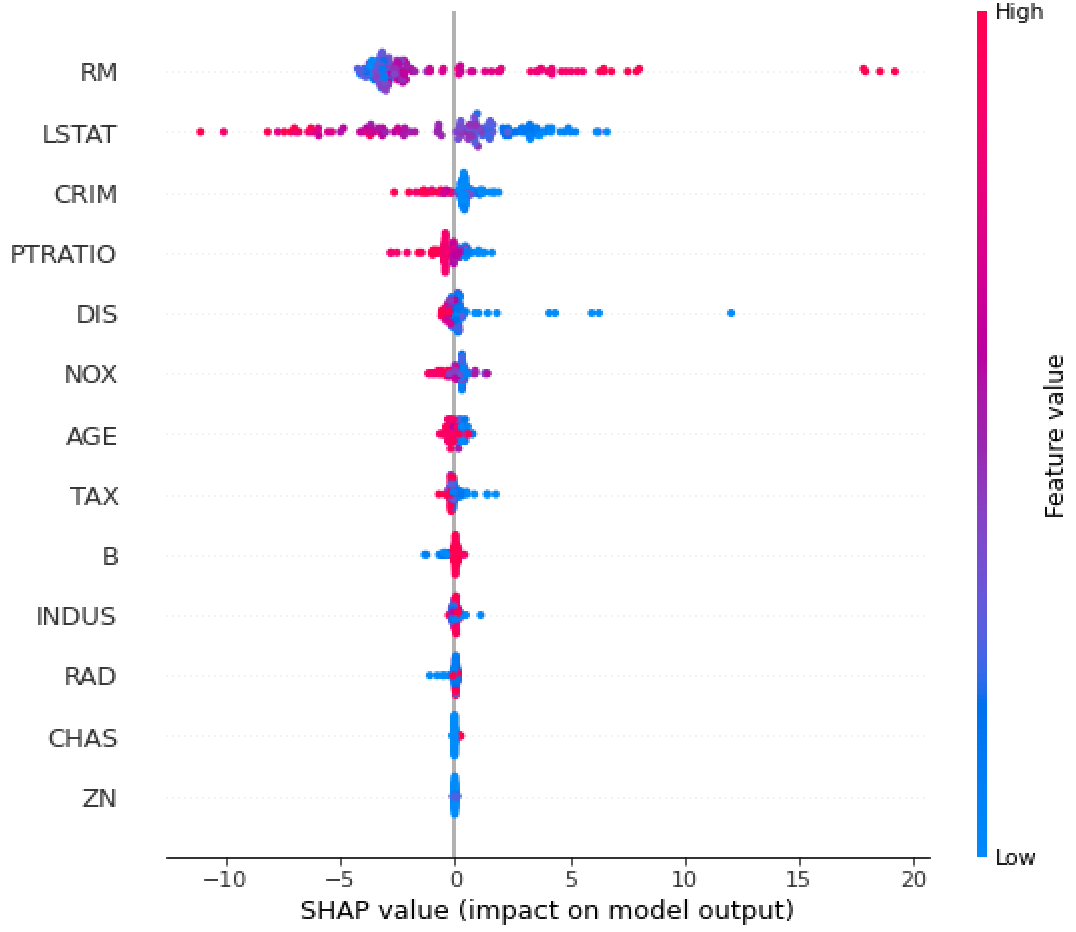
- Model-agnostic variable ranker
- Based on game theory
- Popular for AI explainability
- *pip install shap*



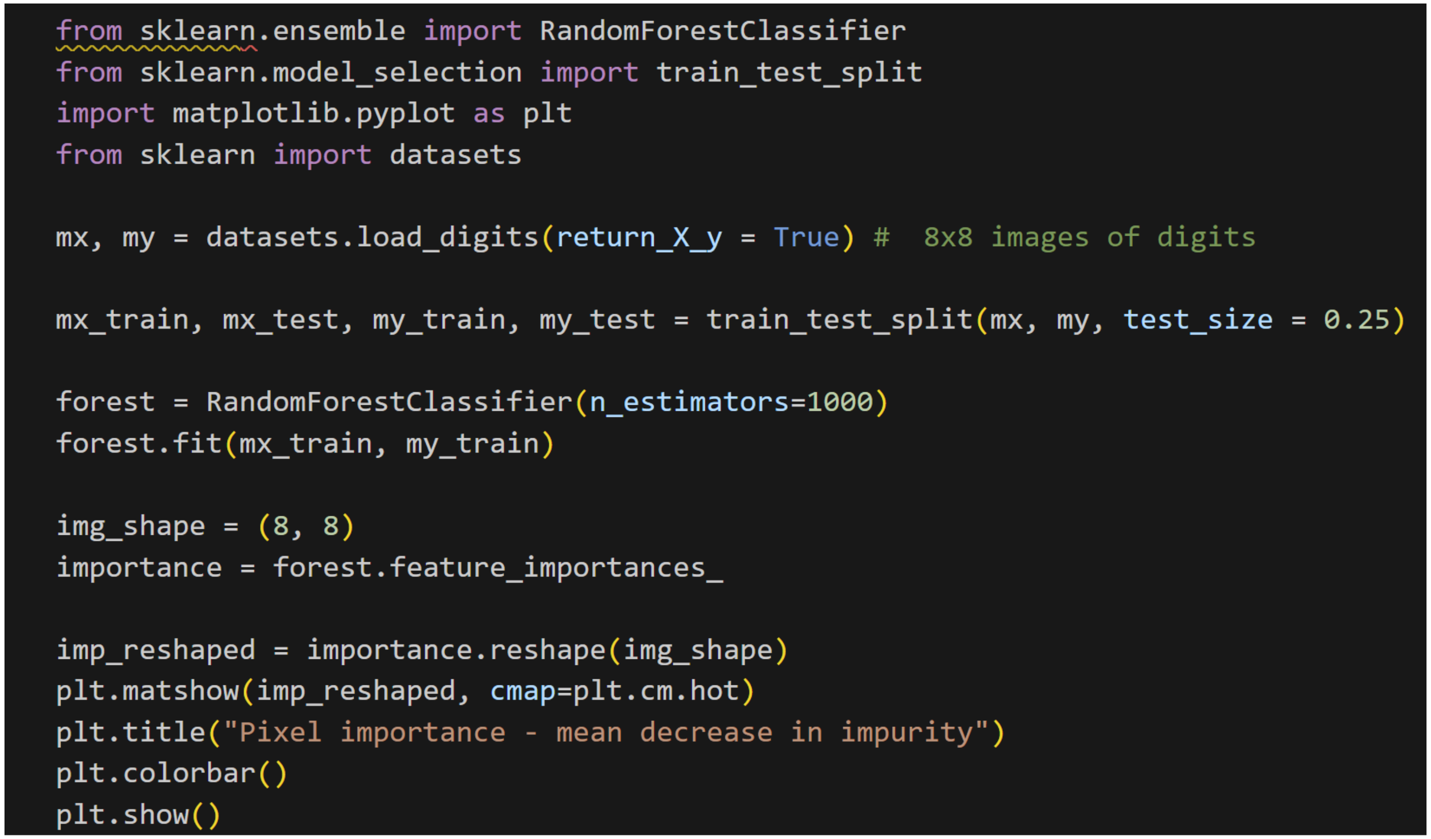
```
import shap

data = datasets.load_boston(return_X_y=False)
features = data.feature_names

explainer = shap.TreeExplainer(rfm)
shap_values = explainer.shap_values(bx_test)

shap.summary_plot(shap_values, bx_test, feature_names = features)
```

Variable importance

- Variable values where resulting SHAP ≥ 0 have a positive impact on the prediction
- Idem for negative
- RM with higher values has a positive impact
- LSTAT with higher values has a negative one

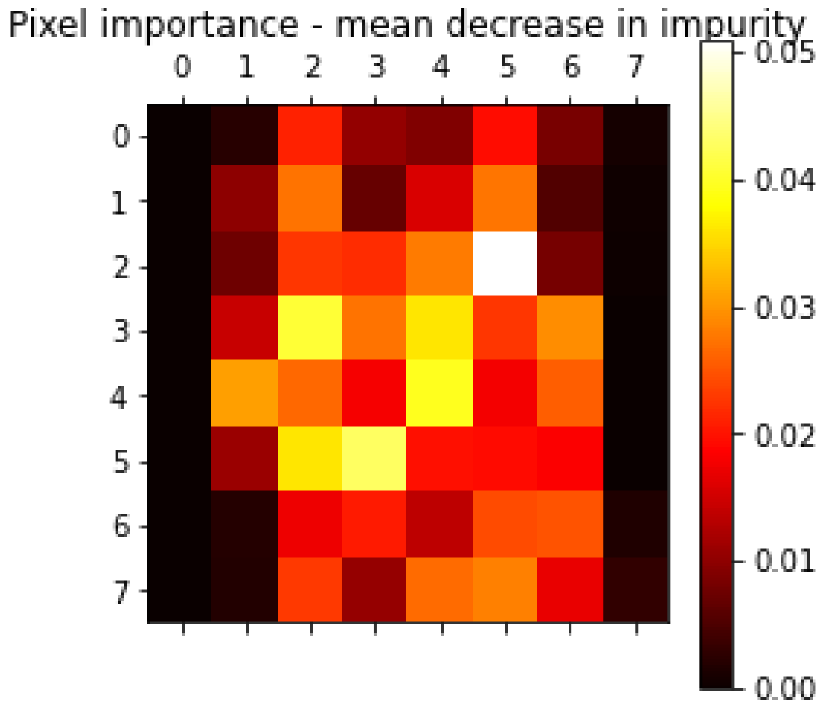


Pixel importance in an image



```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn import datasets

mx, my = datasets.load_digits(return_X_y = True) #  8x8 images of digits

mx_train, mx_test, my_train, my_test = train_test_split(mx, my, test_size = 0.25)

forest = RandomForestClassifier(n_estimators=1000)
forest.fit(mx_train, my_train)

img_shape = (8, 8)
importance = forest.feature_importances_

imp_reshaped = importance.reshape(img_shape)
plt.matshow(imp_reshaped, cmap=plt.cm.hot)
plt.title("Pixel importance - mean decrease in impurity")
plt.colorbar()
plt.show()
```

MNIST pixel importance

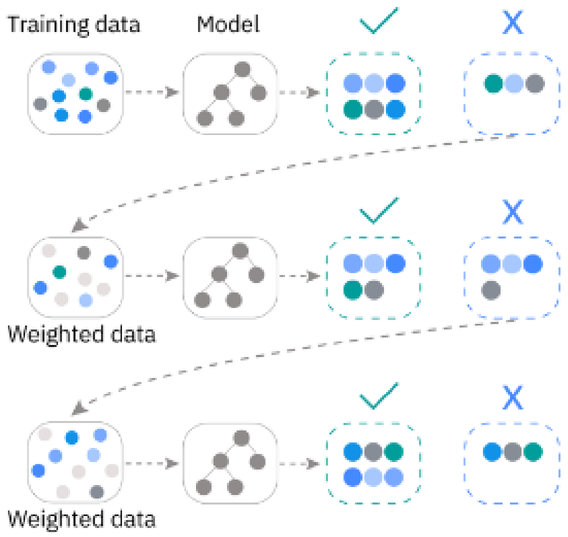


Homework

- Implement in R or Python a decision tree model for
  - either Pima Indians Diabetes
  - or a car price / customer purchasing behaviour.
- Represent the resulting trees through a plot.
- Implement bagging, boosting and random forests in comparison.
- Implement also a neural network as the black-box model.

Extra

- Research XGBoost (gradient boosting), explain its concept of weak learners training on residuals and apply it on your previous data set of choice.



ibm.com

medium.com

I'm a data scientist
Oh yeah? Name 500 models
A Random Forest
That's on me, I set the bar too low.