Meteorite landings

by Aleksandrs Rudenoks(1405307)

December 18, 2017

Introduction

This document provides information about an experiment on data exploration with use of a publicly available dataset. The selected dataset is taken from Kaggle website(https://www.kaggle.com/nasa/meteorite-landings). It contains information about location, mass, year and composition for over 45000 meteorites that have fallen to Earth from outer space. Originally, the dataset was downloaded from NASA's Data Portal, and is based on The Meteoritical Bulletin Database.

All following tasks are performed using R Studio. The dataset is relevantly small(689KB), therefore, there is no need to use parallel computing techniques(e.g. Hadoop). Such technologies allow processing large datasets across clusters of computers using simple programming models and therefore save time but make no difference when used with small datasets.

Objective

Purpose of this task is to analise given data, provide visual representation of it and use a machine learning algorithm to make predictions.

The main objective of the experement is to visualize the meteorite landings on the map and try to predict, whether the fall of each particular meteorite was seen by humans or it was discovered after the fall.

Data Exploration

Before starting the exploration of the dataset, it is a good idea to have a look at original dataset.

Loading dataset.

```r
curdir = getwd()
setwd(curdir)

# get rid of scientific notation
options(scipen=999)

df <- read.csv('meteorite-landings.csv', header=T,na.strings = c(""))
```

The dataset contains the following variables:

- name: the name of the meteorite
- id: a unique identifier for the meteorite
- nametype: one of:

  valid: a typical meteorite

  relict: a meteorite that has been highly degraded by weather on Earth
- recclass: the class of the meteorite; one of a large number of classes based on physical, chemical, and other characteristics;
- mass: the mass of the meteorite, in grams
- fall: whether the meteorite was seen falling, or was discovered after its impact; one of:

  Fell: the meteorite's fall was observed

  Found: the meteorite's fall was not observed
- year: the year the meteorite fell, or the year it was found
- reclat: the latitude of the meteorite's landing
- reclong: the longitude of the meteorite's landing
- GeoLocation: a parentheses-enclose, comma-separated tuple that combines reclat and reclong

Display variable names.

```
names(df)

##  [1] "name"        "id"          "nametype"    "recclass"    "mass"
##  [6] "fall"        "year"        "reclat"      "reclong"     "GeoLocation"
```

Display first 6 rows.

```
head(df)

##       name  id nametype    recclass   mass fall year    reclat    reclong
## 1   Aachen   1    Valid          L5     21 Fell 1880  50.77500    6.08333
## 2   Aarhus   2    Valid          H6    720 Fell 1951  56.18333   10.23333
## 3     Abee   6    Valid         EH4 107000 Fell 1952  54.21667 -113.00000
## 4 Acapulco  10    Valid Acapulcoite   1914 Fell 1976  16.88333  -99.90000
## 5  Achiras 370    Valid          L6    780 Fell 1902 -33.16667  -64.95000
## 6 Adhi Kot 379    Valid         EH4   4239 Fell 1919  32.10000   71.80000
##               GeoLocation
## 1    (50.775000, 6.083330)
## 2   (56.183330, 10.233330)
## 3 (54.216670, -113.000000)
## 4  (16.883330, -99.900000)
## 5 (-33.166670, -64.950000)
## 6   (32.100000, 71.800000)
```

Display number of rows and columns.

```
cat("Number of rows in the set: ", nrow(df))

## Number of rows in the set:  45716

cat("Number of columns in the set: ", ncol(df))

## Number of columns in the set:  10
```

Display unique values in dataset.

```
unq_values <- sapply(df, function(x)length(unique(x)))
unq_values

##        name          id    nametype    recclass        mass        fall
##       45716       45716           2         466       12577           2
##        year      reclat     reclong GeoLocation
##         269       12739       14641       17101
```

Pre-processing

In order to work with the dataset, some necessary pre-processing steps needs to be applied.
First of all, we make a copy of the dataset, which we are going to work with, and check the information about its structure.

```
df_copy <- df
str(df_copy)

## 'data.frame':	45716 obs. of  10 variables:
##  $ name       : Factor w/ 45716 levels "sterplana 002",..: 68 69 73 77 473 484 496 497 50
##  $ id         : int  1 2 6 10 370 379 390 392 398 417 ...
##  $ nametype   : Factor w/ 2 levels "Relict","Valid": 2 2 2 2 2 2 2 2 2 2 ...
##  $ recclass   : Factor w/ 466 levels "Acapulcoite",..: 333 197 85 1 339 85 360 190 339 24
##  $ mass       : num  21 720 107000 1914 780 ...
##  $ fall       : Factor w/ 2 levels "Fell","Found": 1 1 1 1 1 1 1 1 1 1 ...
##  $ year       : int  1880 1951 1952 1976 1902 1919 1949 1814 1930 1920 ...
##  $ reclat     : num  50.8 56.2 54.2 16.9 -33.2 ...
##  $ reclong    : num  6.08 10.23 -113 -99.9 -64.95 ...
##  $ GeoLocation: Factor w/ 17100 levels "(-1.002780, 37.150280)",..: 16778 16982 16922 910
```

According to Kaggle website, this dataset has either missing or incorrect data. Entries that have latitude and longitude of 0N/0E or have a date that is before 860 CE or after 2016 are incorrect, therefore, should be removed.

```
df_copy <- df_copy[!(df_copy$reclat==0 & df_copy$reclong==0),]
df_copy <- df_copy[(df_copy$year>=860 & df_copy$year<=2016),]
df_copy <- df_copy[(df_copy$reclong<=180 & df_copy$reclong>=-180),]
```

Then we check for missing values.

```
na_values <- sapply(df_copy, function(x)sum(is.na(x)))
na_values

##        name          id    nametype    recclass        mass        fall
##        7462        7462        7462        7462        7569        7462
##        year      reclat     reclong GeoLocation
##        7462        7462        7462        7462
```

And delete rows which contain missing values.

```
df_copy <- df_copy[!is.na(df_copy$reclat),]
df_copy <- df_copy[!is.na(df_copy$reclong),]
df_copy <- df_copy[!is.na(df_copy$mass),]
```

Double check that no missing values left.

```
na_values <- sapply(df_copy, function(x)sum(is.na(x)))
na_values

##        name          id    nametype    recclass        mass        fall
##           0           0           0           0           0           0
##        year      reclat     reclong GeoLocation
##           0           0           0           0
```

Now we can delete columns that will never be used and display result.

```
df_copy$id <- NULL
df_copy$GeoLocation <- NULL
df_copy$mass <- NULL
df_copy$nametype <- NULL
str(df_copy)

## 'data.frame': 31929 obs. of  6 variables:
##  $ name    : Factor w/ 45716 levels "sterplana 002",..: 68 69 73 77 473 484 496 497 502 5
##  $ recclass: Factor w/ 466 levels "Acapulcoite",..: 333 197 85 1 339 85 360 190 339 242 .
##  $ fall    : Factor w/ 2 levels "Fell","Found": 1 1 1 1 1 1 1 1 1 1 ...
##  $ year    : int  1880 1951 1952 1976 1902 1919 1949 1814 1930 1920 ...
##  $ reclat  : num  50.8 56.2 54.2 16.9 -33.2 ...
##  $ reclong : num  6.08 10.23 -113 -99.9 -64.95 ...
```

The last thing to do is to add rounded coordinates and a new column to our data frame.

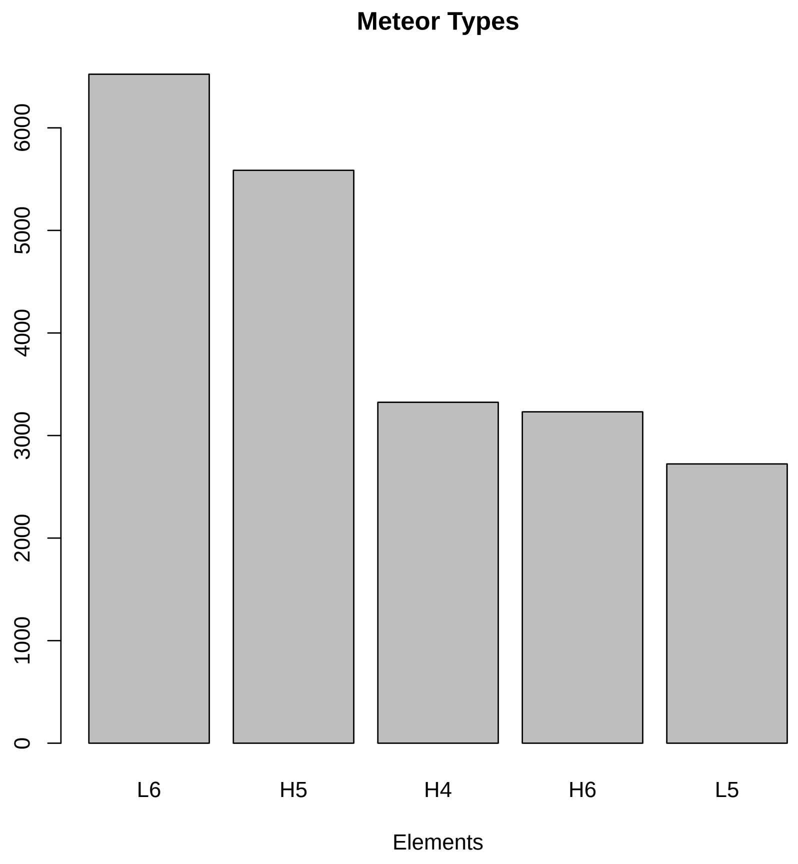
```
# rounded latitude and longtitude
df_copy$roundLat <- round(df_copy$reclat)
df_copy$roundLong <- round(df_copy$reclong)
# fell values displayed as 1("Found") and 0("Fell")
df_copy$newFall <- ifelse(df_copy$fall=="Found",1,0)
df_copy$newFall <- as.factor(df_copy$newFall)
```

We will need this later on. Now the dataset is ready to be worked with.

Visualisation

It is always a good idea to visualise the data in the set. Let's have a look at top 5 matereals that meteors are made of.
Full descriprion of elements can be found at (http://class.meteorites.com.au/).

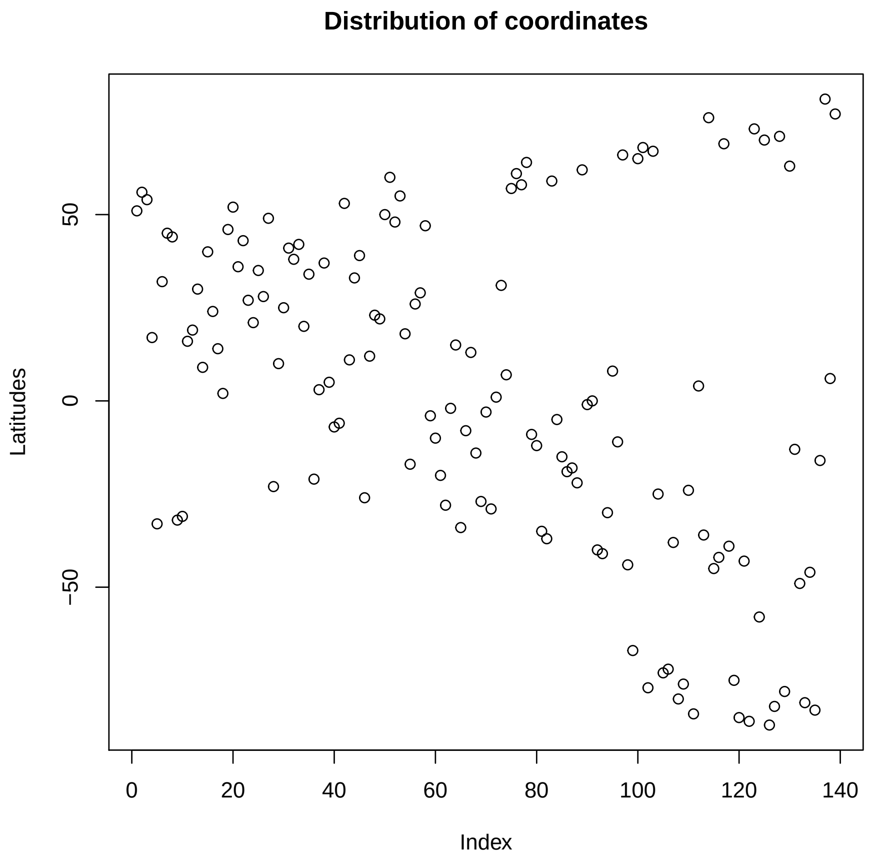
```
meteor_types <- sort(table(df_copy$recclass), decreasing = TRUE)[1:5]

barplot(meteor_types, main="Meteor Types", xlab="Elements")
```



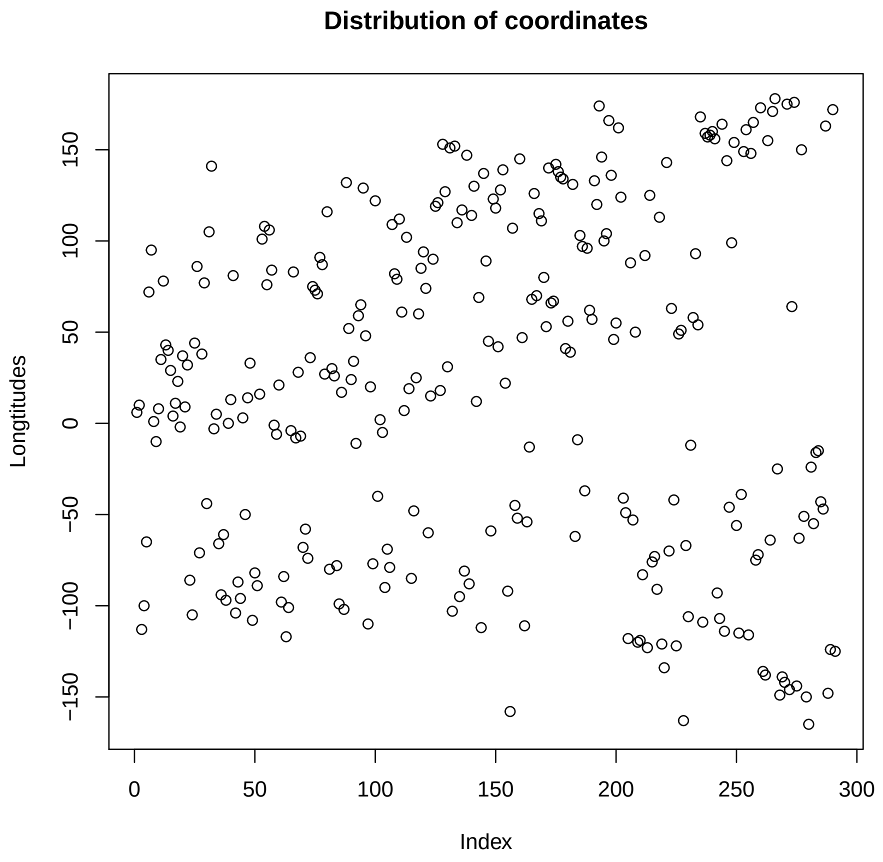
We can also see the distribution of coordinates.
We are going to use rounded coordinates as there are too many unique coordinates in our original dataset and we can't display them all.

```
# original coordinates
coord_values <- sapply(df_copy[5:6], function(x)length(unique(x)))
coord_values

##  reclat reclong
##   12602   14473

# rounded coordinates
unq_coord_values <- sapply(df_copy[7:8], function(x)length(unique(x)))
unq_coord_values

##  roundLat roundLong
##       139       291
```

```
plot.default(unique(df_copy$roundLat), main = "Distribution of coordinates", ylab = "Latitud
```



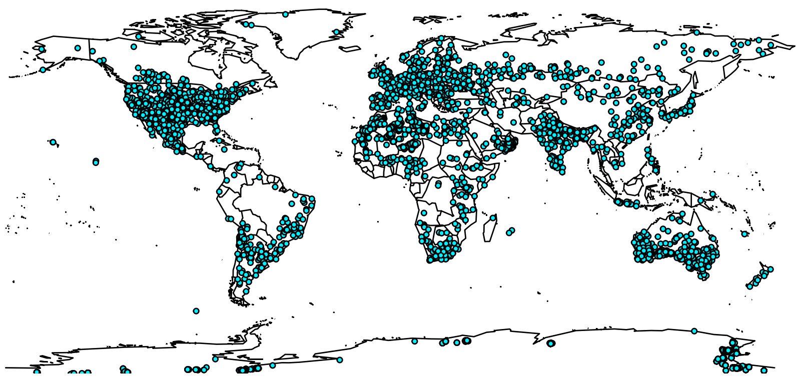
```
plot.default(unique(df_copy$roundLong), main = "Distribution of coordinates", ylab = "Longti
```



Distribution of coordinates
Longtitudes
Index
150
100
50
0
-50
-100
-150
0
50
100
150
200
250
300

It would be handy to display the map of the world and display all coordinates from the dataset. To do that, library map is used and blue circles are ploted on top of it.

```r
# install.packages("maps")
library(maps)
map(database = "world")
# usual coordinates are used for displaying landings
symbols(df_copy$reclong,df_copy$reclat, bg = "#00F0FF",
        circles = rep(1, length(df_copy$reclong)),
        inches = 0.02, add = TRUE)
```



Modelling

Now it is time to devide our working dataset into training and testing subsets.

```
library(caret)

## Loading required package:  lattice
## Loading required package:  ggplot2

inTrain <- createDataPartition(y=df_copy$newFall, p=.6,list=FALSE)
training <- df_copy[inTrain,]
testing <- df_copy[-inTrain,]
```

Lets check if number of rows of divided sets match number of rows of the original set.

```
nrow(training) + nrow(testing) == nrow(df_copy)

## [1] TRUE
```

Check class distribution on both sets.

```
table(training$newFall)

##
##     0     1
##   639 18519

table(testing$newFall)

##
##     0     1
##   425 12346
```

For our testing and training sets we don't need all of the columns. We can look at them and decide which ones will be used.

```
str(training)

## 'data.frame': 19158 obs. of  9 variables:
##  $ name     : Factor w/ 45716 levels "sterplana 002",..: 68 69 73 484 496 497 521 525 527
##  $ recclass : Factor w/ 466 levels "Acapulcoite",..: 333 197 85 85 360 190 242 72 339 18
##  $ fall     : Factor w/ 2 levels "Fell","Found": 1 1 1 1 1 1 1 1 1 1 ...
##  $ year     : int  1880 1951 1952 1919 1949 1814 1920 1974 1949 1838 ...
##  $ reclat   : num  50.8 56.2 54.2 32.1 44.8 ...
##  $ reclong  : num  6.08 10.23 -113 71.8 95.17 ...
## $ roundLat : num  51 56 54 32 45 44 -31 16 30 30 ...
## $ roundLong: num  6 10 -113 72 95 1 -65 -10 35 78 ...
## $ newFall  : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

```r
str(testing)
```

```
## 'data.frame': 12771 obs. of  9 variables:
## $ name     : Factor w/ 45716 levels "sterplana 002",..: 77 473 502 67 533 589 602 605 60
## $ recclass : Factor w/ 466 levels "Acapulcoite",..: 1 339 339 339 113 64 109 386 190 339
## $ fall     : Factor w/ 2 levels "Fell","Found": 1 1 1 1 1 1 1 1 1 1 ...
## $ year     : int  1976 1902 1930 1925 1959 1957 2002 1835 1860 1883 ...
## $ reclat   : num  16.88 -33.17 -31.6 19.08 8.92 ...
## $ reclong  : num  -99.9 -64.95 -65.23 8.38 8.43 ...
## $ roundLat : num  17 -33 -32 19 9 24 46 52 45 45 ...
## $ roundLong: num  -100 -65 -65 8 8 40 6 -2 9 10 ...
## $ newFall  : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

For the test we won't need variables name, recclass, fall, reclat, reclong.

```r
training$name <- NULL
training$recclass <- NULL
training$fall <- NULL
training$reclat <- NULL
training$reclong <- NULL
testing$name <- NULL
testing$recclass <- NULL
testing$fall <- NULL
testing$reclat <- NULL
testing$reclong <- NULL
```

Check remaining columns.

```r
str(training)
```

```
## 'data.frame': 19158 obs. of  4 variables:
## $ year     : int  1880 1951 1952 1919 1949 1814 1920 1974 1949 1838 ...
## $ roundLat : num  51 56 54 32 45 44 -31 16 30 30 ...
## $ roundLong: num  6 10 -113 72 95 1 -65 -10 35 78 ...
## $ newFall  : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

```r
str(testing)
```

```
## 'data.frame': 12771 obs. of  4 variables:
## $ year     : int  1976 1902 1930 1925 1959 1957 2002 1835 1860 1883 ...
## $ roundLat : num  17 -33 -32 19 9 24 46 52 45 45 ...
## $ roundLong: num  -100 -65 -65 8 8 40 6 -2 9 10 ...
## $ newFall  : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

For this data processing we will be using machine learning technique called Random Forrest.

```
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

memory.limit(size = 25000)

## [1] 25000
```

First of all we need to decide on the number of trees we are going to use with our model. For that we run our model 10 times and record the accuracy for each run so that we can decide the best option for us.

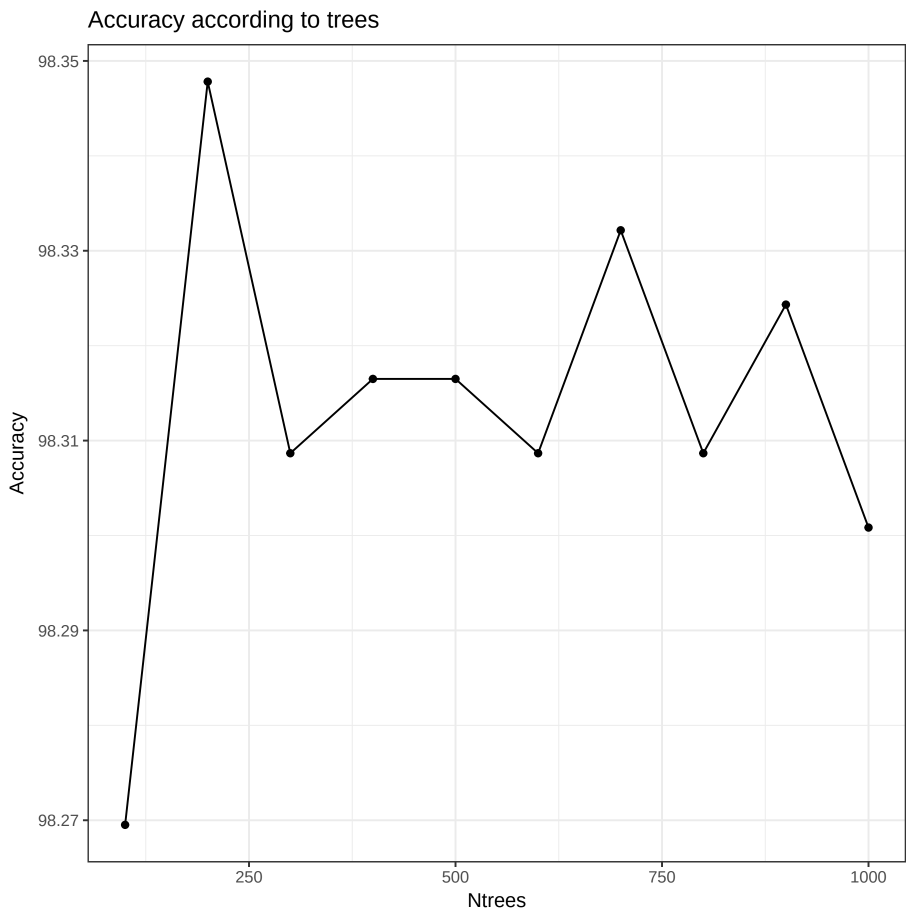
```
# create new dataframe to store accuracy
ac_df <- data.frame(Ntrees = as.numeric(), Accuracy = as.numeric())
# initial number of trees
ntree <- 100
for (i in 1:10){
  test_rf1 <- randomForest(newFall~., data = training, ntrees = ntree )
  testPredict <- predict(test_rf1, newdata = testing)
  # calculate accuracy of each run
  auc <- (sum(testPredict==testing$newFall)/nrow(testing))*100
  # save results in the ac_df data frame
  ac_df <- rbind(ac_df, data.frame(Ntrees=ntree, Accuracy=auc))
  # change number of trees for the next run
  ntree <- ntree + 100
}
```

Display results.

```
ac_df

##    Ntrees Accuracy
## 1     100 98.26952
## 2     200 98.34782
## 3     300 98.30867
## 4     400 98.31650
## 5     500 98.31650
## 6     600 98.30867
## 7     700 98.33216
## 8     800 98.30867
## 9     900 98.32433
## 10   1000 98.30084
```

We can analyze results in more efficient way by plotting them.

```
library(ggplot2)
pl <- ggplot(ac_df, aes(x=Ntrees, y=Accuracy))
pl <- pl + geom_line() + geom_point() + ggtitle("Accuracy according to trees")
pl <- pl + xlim(100,1000)
pl <- pl + theme_bw()
pl
```

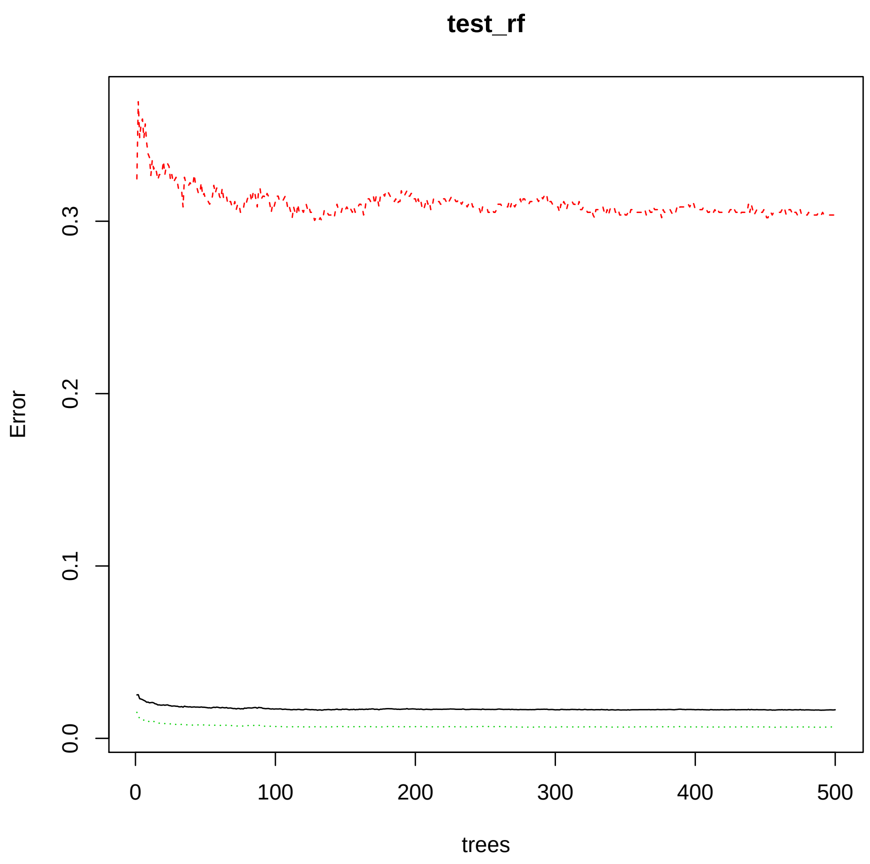


From the output above we can see that accuracy is quite high(98.3 percent) and that number of trees is almost not influencing the accuracy. Therefore, we can use any number of trees for our model. Lets run our model again with 500 trees and display the results.

```
test_rf <- randomForest(newFall~., data = training, ntree = 500)
print(test_rf)

##
## Call:
##  randomForest(formula = newFall ~ ., data = training, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 1.66%
## Confusion matrix:
##     0     1 class.error
## 0 445   194 0.303599374
## 1 124 18395 0.006695826
```

We can have a look at the error against the number of trees. This figure confirms that number of trees is irrelevant to our case.

```
plot(test_rf)
```

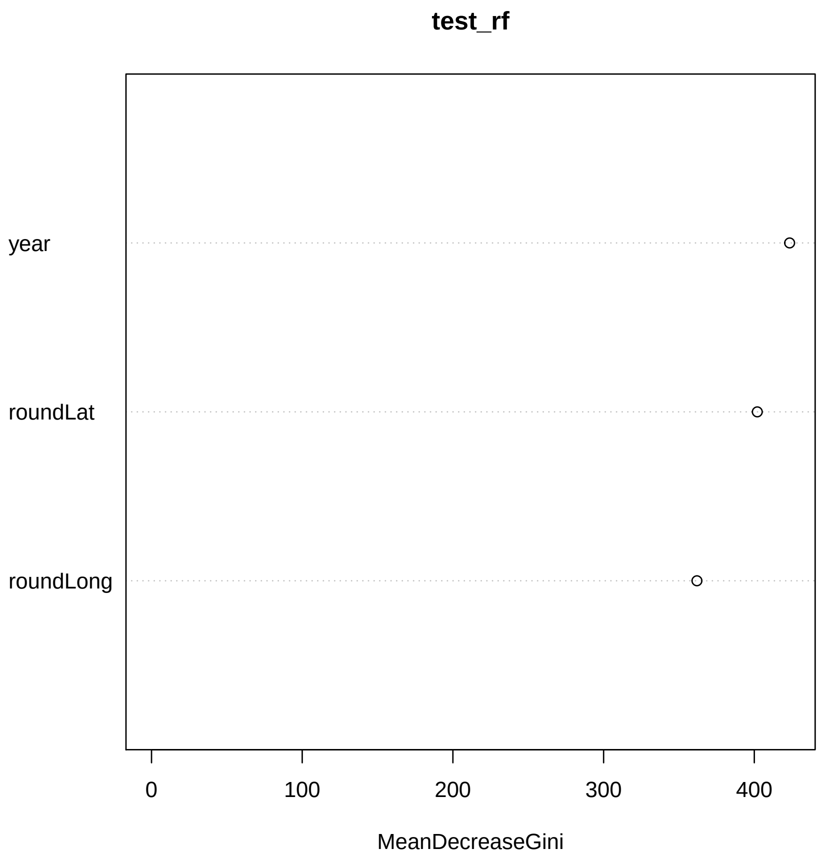


Another good idea is to analize the importance of variables on which decision is based on.

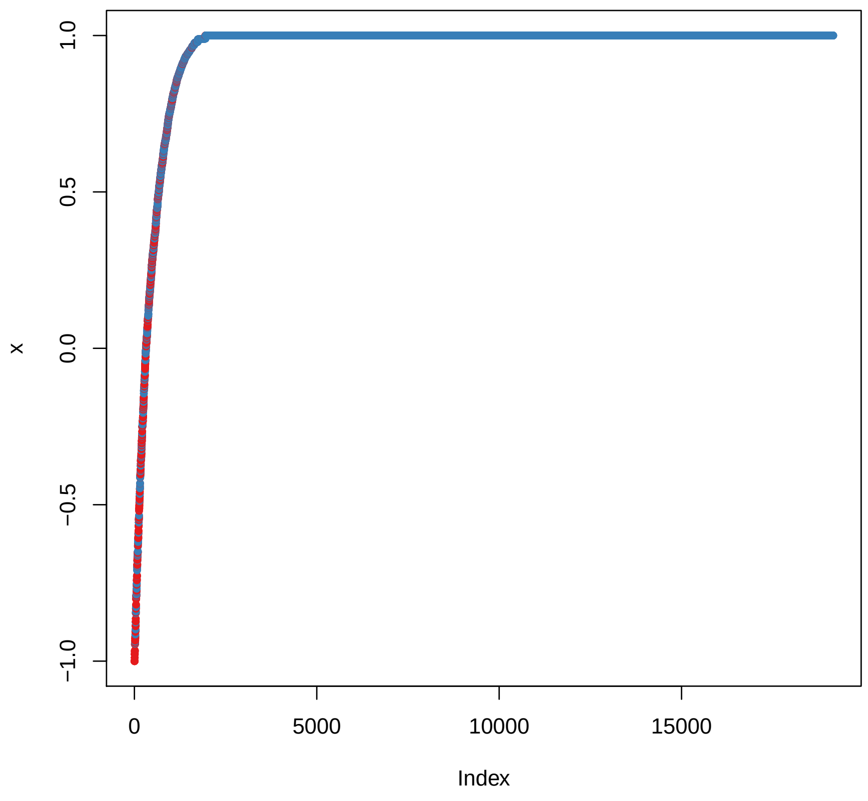
```
importance(test_rf)

##             MeanDecreaseGini
## year                423.4388
## roundLat            401.9847
## roundLong           361.9311
```

From that figure we can see that year is more important feature than coordinates. That can be explained by fact that nowadays people have more precise options to locate the falling meteorite and share that information with others. The other two variables are nearly as important as year. This can be explained that some areas are more populated by humans than others and in those places falling meteorites are more likely to be observed.

```
varImpPlot(test_rf)
```



```
# plot the margin of predictions from randomForest classifier
plot(margin(test_rf, testing$newFall))

## Warning in RColorBrewer::brewer.pal(nlevs, "Set1"):  minimal value
for n is 3, returning requested palette with 3 different levels
```



And, finally, we can display the machine prediction on whether the meteorite fall was seen by human and compare it to testing set.

```
testPredict <- predict(test_rf, newdata = testing)
table(testPredict, testing$newFall)

##
## testPredict     0     1
##           0   286    81
##           1   139 12265
```

We can also add a new column to our testing set and compare the predictions against the real data. That perfectly displays the performance of our model.

```
testing$predRight <- testPredict==testing$newFall
head(testing)

##    year roundLat roundLong newFall predRight
## 4  1976       17      -100       0     FALSE
## 5  1902      -33       -65       0      TRUE
## 9  1930      -32       -65       0      TRUE
## 12 1925       19         8       0      TRUE
## 16 1959        9         8       0      TRUE
## 18 1957       24        40       0     FALSE

table(testing$predRight)

##
## FALSE  TRUE
##   220 12551
```