Ministry of Education and Science of Ukraine
National Technical University of Ukraine
«Igor Sikorsky Kyiv Polytechnic Institute»

**№1.1**

Work with WEKA. NAÏVE BAYES AND DECISION TREE CLASSIFICATION

(link: https://docs.google.com/document/d/19rnX3tvZpiEF89LJEulBQzaCJSvuJUTu/edit)


**The work was done by**
Zvychaynaya Anastasia

**The work was checked by**
Alexander Oriekhov


Kyiv 2022

# Execution of the work:

## 1. SPAM FILTERING

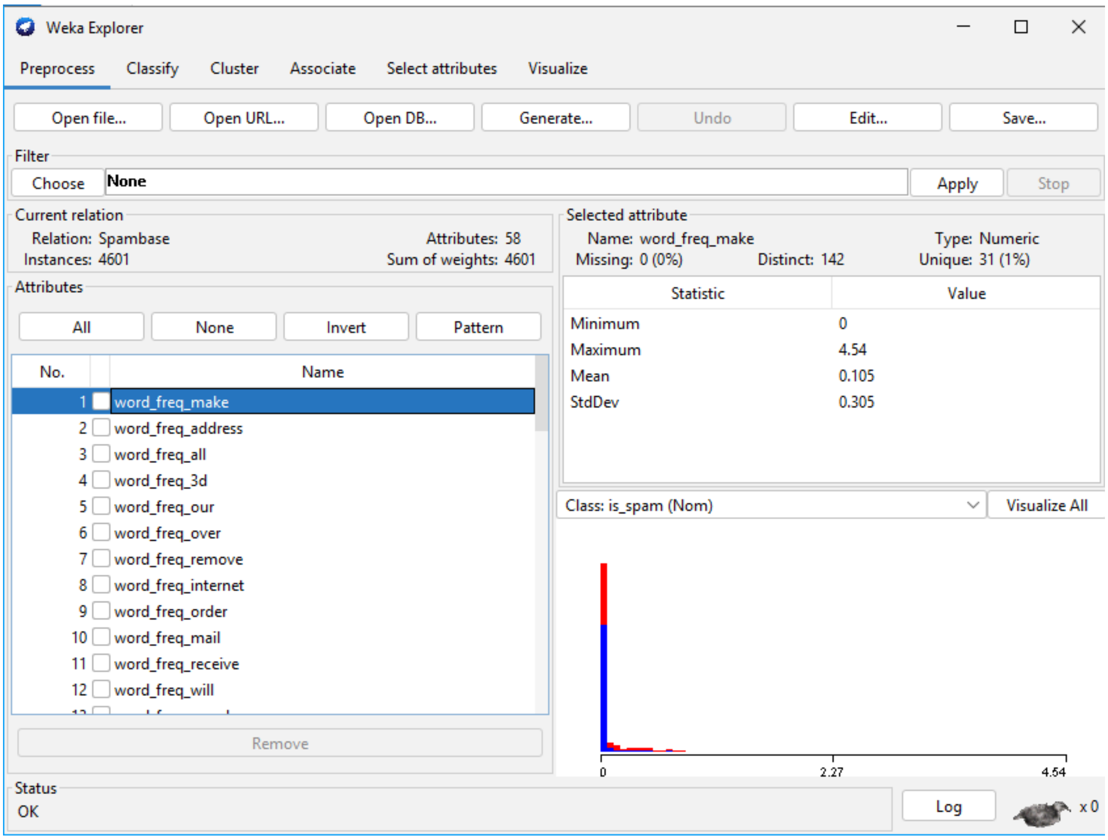
*Some simple preprocessing of the data will be required before it is ready for use. We can do this in Weka:*

1. *Familiarize yourself with the [ARFF format]()*

   ***Done.***

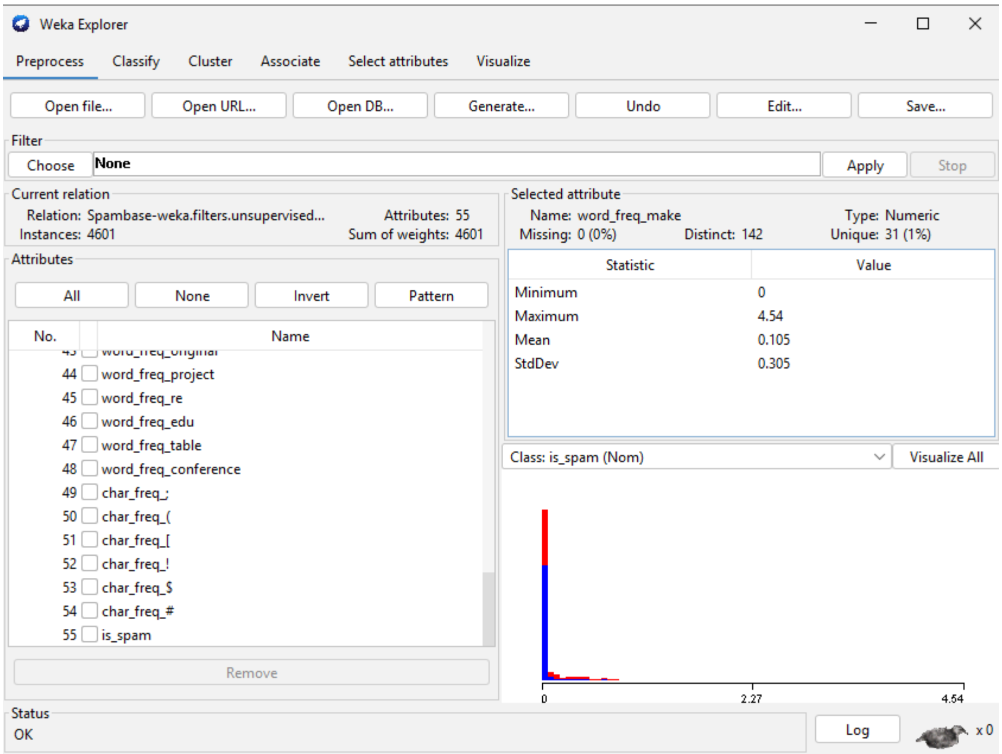
2. *From the Preprocess (default) tab in Weka, hit Open file... and select the `spambase.arff` file that you downloaded above.*

   ***Done:***

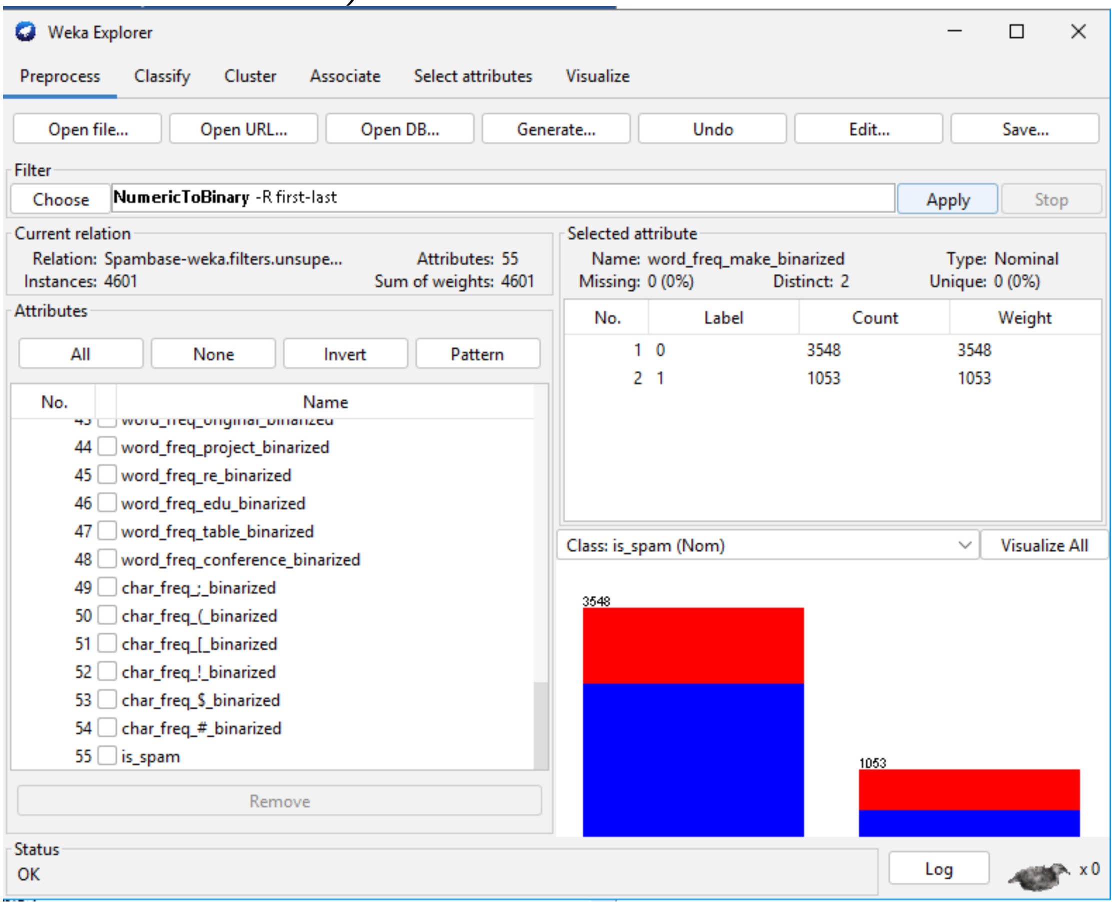



3. *A full list of the attributes in this data set will appear in the "Attributes" frame.*

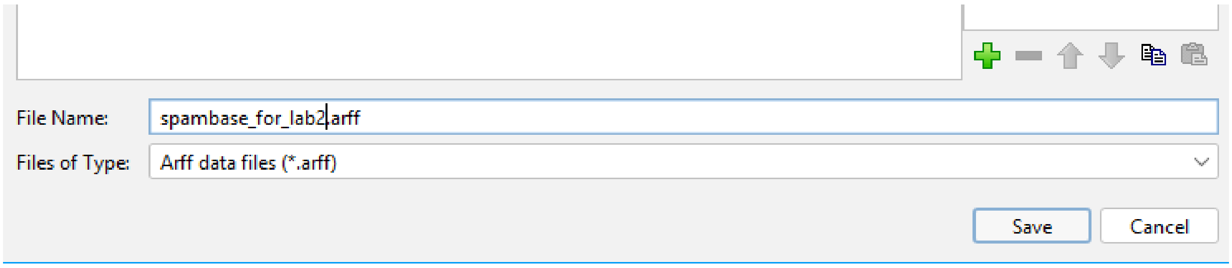
4. *Delete the `capital_run_length_average`, `capital_run_length_longest` and `capital_run_length_total` attributes by checking the box to their left and hitting the Remove button.*




5. *The remaining attributes represent relative frequencies of various important words and characters in emails. We wish to convert these to Boolean values instead: 1 if the word or character is present in the email, 0 if not. To do this, select the Choose button in the Filter frame at the top of the window, and pick filters > unsupervised > attribute > NumericToBinary. Now hit the Apply button. All the numeric frequency attributes are now converted to Booleans. Each e-mail is now represented by a 55 dimensional vector representing whether or not a particular word exists in an e-mail. This is the so called bag of words representation (this is clearly a very crude assumption since it does not take into account the order of the words).*

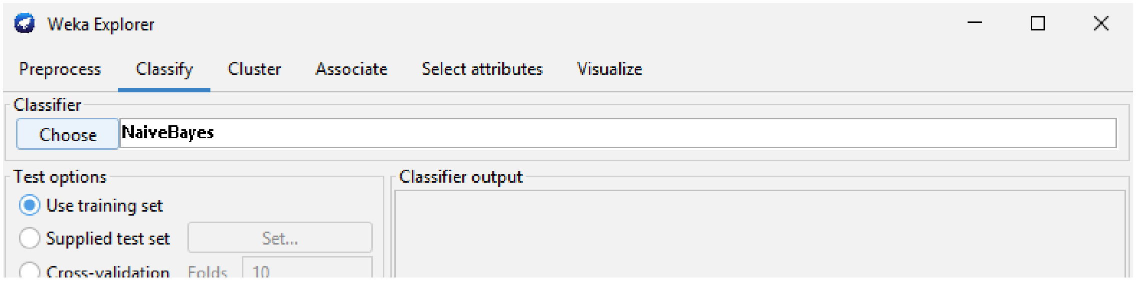


6. *Save this preprocessed data set for future use using the Save... button. You will need this for lab 2.*




*Given the data set we've just loaded, we wish to train a Naïve Bayes classifier to distinguish spam from regular email by fitting a distribution of the number of occurrences of each word for all the spam and non-spam e-mails. Under the Classify tab:*

1. *Select Choose in the Classifier frame at the top and select classifiers > bayes > NaiveBayes.*




2. *Leave the default settings and hit Start to build the classifier. Study the output produced, most importantly the percentages of correctly and incorrectly classified instances. You probably will notice that your classifer does rather well despite making a very strong assumption on the form of the data.*
   - *Can you come up with a reason for the good performance? What would be the main practical problems we would face if we were not to make this assumption for this particular dataset?*

   **Let us talk about Naïve Bayes Assumption.**

   The *Naïve Bayes Assumption*: Assume that all features are independent given the class label Y

$$P(X_1, \ldots, X_n|Y) = \prod_{i=1}^{n} P(X_i|Y)$$

$$\boxed{P(y|x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)}$$

**•When assumption of independence holds, a Naive Bayes classifier performs better compare to other models**

**•It perform well in case of categorical input variables compared to numerical variable(s).**

**•If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero**

**Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.**

- *How long did your classifer take to train and classify? Given this, how scalable do you think the Naïve Bayes classifier is to large datasets? Can you come up with a good reason for this?*

***If percentage split = 20%, then***

```
Classifier output

Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.05 seconds

=== Summary ===

Correctly Classified Instances        3240               88.0196 %
Incorrectly Classified Instances       441               11.9804 %
Kappa statistic                          0.7452
Mean absolute error                      0.1256
Root mean squared error                  0.3257
Relative absolute error                 26.4654 %
Root relative squared error             66.4182 %
Total Number of Instances             3681

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,939    0,209    0,871      0,939    0,904      0,749    0,949     0,964     0
                 0,791    0,061    0,897      0,791    0,841      0,749    0,949     0,933     1
Weighted Avg.    0,880    0,150    0,881      0,880    0,879      0,749    0,949     0,951

=== Confusion Matrix ===

    a    b   <-- classified as
 2078  134 |    a = 0
  307 1162 |    b = 1
```

***If percentage split = 60%, then***

```
Classifier output

Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.03 seconds

=== Summary ===

Correctly Classified Instances        1626               88.3696 %
Incorrectly Classified Instances       214               11.6304 %
Kappa statistic                          0.7515
Mean absolute error                      0.1201
Root mean squared error                  0.3174
Relative absolute error                 25.1825 %
Root relative squared error             65.1829 %
Total Number of Instances             1840

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,926    0,183    0,890      0,926    0,907      0,753    0,951     0,967     0
                 0,817    0,074    0,873      0,817    0,844      0,753    0,951     0,931     1
Weighted Avg.    0,884    0,141    0,883      0,884    0,883      0,753    0,951     0,953

=== Confusion Matrix ===

    a    b   <-- classified as
 1047   84 |    a = 0
  130  579 |    b = 1
```

***If percentage split = 70%, then***

```
Classifier output

Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.02 seconds

=== Summary ===

Correctly Classified Instances        1225               88.7681 %
Incorrectly Classified Instances       155               11.2319 %
Kappa statistic                          0.7621
Mean absolute error                      0.1161
Root mean squared error                  0.311
Relative absolute error                 24.3328 %
Root relative squared error             63.7266 %
Total Number of Instances             1380

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,924    0,169    0,895      0,924    0,909      0,763    0,957     0,972     0
                 0,831    0,076    0,875      0,831    0,853      0,763    0,957     0,938     1
Weighted Avg.    0,888    0,132    0,887      0,888    0,887      0,763    0,957     0,959

=== Confusion Matrix ===

   a   b   <-- classified as
 776  64 |   a = 0
  91 449 |   b = 1
```

***If percentage split = 80%, then***

```
Classifier output

Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances         808               87.8261 %
Incorrectly Classified Instances       112               12.1739 %
Kappa statistic                          0.7445
Mean absolute error                      0.123
Root mean squared error                  0.3226
Relative absolute error                 25.6556 %
Root relative squared error             65.6352 %
Total Number of Instances              920

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,925    0,190    0,877      0,925    0,900      0,746    0,953     0,967     0
                 0,810    0,075    0,881      0,810    0,844      0,746    0,953     0,940     1
Weighted Avg.    0,878    0,143    0,878      0,878    0,877      0,746    0,953     0,956

=== Confusion Matrix ===

   a   b   <-- classified as
 505  41 |   a = 0
  71 303 |   b = 1
```

*If percentage split = 90%, then*

```
Classifier output

Time taken to build model: 0 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances         402               87.3913 %
Incorrectly Classified Instances        58               12.6087 %
Kappa statistic                          0.7357
Mean absolute error                      0.1252
Root mean squared error                  0.3261
Relative absolute error                 26.0343 %
Root relative squared error             66.2359 %
Total Number of Instances              460

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,926    0,201    0,869      0,926    0,896      0,738    0,956     0,969     0
                 0,799    0,074    0,883      0,799    0,839      0,738    0,956     0,942     1
Weighted Avg.    0,874    0,149    0,874      0,874    0,873      0,738    0,956     0,958

=== Confusion Matrix ===

   a   b   <-- classified as
 251  20 |   a = 0
  38 151 |   b = 1
```

**With such data it is difficult to give a clear-cut answer, although my personal opinion is that the more data we have, the more accurate the result is. However, I just have figured out that we do not have a random data, which means we have a spam sequence and then a non-spam sequence, therefore so there is a need to shuffle the data.**

*If percentage split = 20%, then*

```
Classifier output

Time taken to build model: 0.2 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.27 seconds

=== Summary ===

Correctly Classified Instances        2944               79.9783 %
Incorrectly Classified Instances       737               20.0217 %
Kappa statistic                          0.608
Mean absolute error                      0.2001
Root mean squared error                  0.4456
Relative absolute error                 42.128  %
Root relative squared error             90.9349 %
Total Number of Instances             3681

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,705    0,057    0,949      0,705    0,809      0,638    0,940     0,942     0
                 0,943    0,295    0,679      0,943    0,789      0,638    0,941     0,892     1
Weighted Avg.    0,800    0,152    0,841      0,800    0,801      0,638    0,941     0,922

=== Confusion Matrix ===

    a    b   <-- classified as
 1562  653 |    a = 0
   84 1382 |    b = 1
```

***If percentage split = 60%, then***

```
Classifier output

Time taken to build model: 0.08 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.06 seconds

=== Summary ===

Correctly Classified Instances        1478               80.3261 %
Incorrectly Classified Instances       362               19.6739 %
Kappa statistic                          0.6137
Mean absolute error                      0.1969
Root mean squared error                  0.442
Relative absolute error                 41.2403 %
Root relative squared error             90.5437 %
Total Number of Instances             1840

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,714    0,058    0,950      0,714    0,815      0,643    0,933     0,936     0
                 0,942    0,286    0,680      0,942    0,790      0,643    0,941     0,899     1
Weighted Avg.    0,803    0,147    0,844      0,803    0,805      0,643    0,936     0,921

=== Confusion Matrix ===

   a   b   <-- classified as
 799 320 |   a = 0
  42 679 |   b = 1
```

***If percentage split = 70%, then***

```
Classifier output

Time taken to build model: 0.03 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.08 seconds

=== Summary ===

Correctly Classified Instances        1128               81.7391 %
Incorrectly Classified Instances       252               18.2609 %
Kappa statistic                          0.6418
Mean absolute error                      0.1835
Root mean squared error                  0.4265
Relative absolute error                 38.4306 %
Root relative squared error             87.2671 %
Total Number of Instances             1380

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,724    0,039    0,966      0,724    0,828      0,672    0,949     0,951     0
                 0,961    0,276    0,694      0,961    0,806      0,672    0,953     0,917     1
Weighted Avg.    0,817    0,132    0,859      0,817    0,819      0,672    0,950     0,938

=== Confusion Matrix ===

   a   b   <-- classified as
 605 231 |   a = 0
  21 523 |   b = 1
```

***If percentage split = 80%, then***

```
Classifier output

Time taken to build model: 0.03 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.05 seconds

=== Summary ===

Correctly Classified Instances         743               80.7609 %
Incorrectly Classified Instances       177               19.2391 %
Kappa statistic                          0.6228
Mean absolute error                      0.1914
Root mean squared error                  0.4362
Relative absolute error                 40.176  %
Root relative squared error             89.5408 %
Total Number of Instances              920

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,715    0,045    0,962      0,715    0,820      0,655    0,944     0,949     0
                 0,955    0,285    0,679      0,955    0,793      0,655    0,949     0,910     1
Weighted Avg.    0,808    0,138    0,852      0,808    0,810      0,655    0,946     0,934

=== Confusion Matrix ===

   a   b   <-- classified as
 403 161 |   a = 0
  16 340 |   b = 1
```

***If percentage split = 90%, then***

```
Classifier output

Time taken to build model: 0.04 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances         372               80.8696 %
Incorrectly Classified Instances        88               19.1304 %
Kappa statistic                          0.6262
Mean absolute error                      0.188
Root mean squared error                  0.4311
Relative absolute error                 39.1661 %
Root relative squared error             87.733  %
Total Number of Instances              460

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,707    0,043    0,960      0,707    0,814      0,658    0,951     0,950     0
                 0,957    0,293    0,691      0,957    0,803      0,658    0,956     0,934     1
Weighted Avg.    0,809    0,145    0,851      0,809    0,810      0,658    0,953     0,944

=== Confusion Matrix ===

   a   b   <-- classified as
 193  80 |   a = 0
   8 179 |   b = 1
```

3. Examine the classifier models produced by Weka (printed above the performance summary). Find the prior probabilities for each class.

- *How does Naïve Bayes compute the probability of an e-mail belonging to a class (spam/not spam)?*

***On Wikipedia we can find a good explanation:***

**Document classification** [ edit ]

Here is a worked example of naive Bayesian classification to the document classification problem. Consider the problem of classifying documents by their content, for example into spam and non-spam e-mails. Imagine that documents are drawn from a number of classes of documents which can be modeled as sets of words where the (independent) probability that the i-th word of a given document occurs in a document from class *C* can be written as

$$p(w_i \mid C)$$

(For this treatment, things are further simplified by assuming that words are randomly distributed in the document - that is, words are not dependent on the length of the document, position within the document with relation to other words, or other document-context.)

Then the probability that a given document *D* contains all of the words $w_i$, given a class *C*, is

$$p(D \mid C) = \prod_i p(w_i \mid C)$$

The question that has to be answered is: "what is the probability that a given document *D* belongs to a given class *C*?" In other words, what is $p(C \mid D)$ ?

Now by definition

$$p(D \mid C) = \frac{p(D \cap C)}{p(C)}$$

and

$$p(C \mid D) = \frac{p(D \cap C)}{p(D)}$$

Bayes' theorem manipulates these into a statement of probability in terms of likelihood.

$$p(C \mid D) = \frac{p(C)\,p(D \mid C)}{p(D)}$$

Assume for the moment that there are only two mutually exclusive classes, *S* and ¬*S* (e.g. spam and not spam), such that every element (email) is in either one or the other;

$$p(D \mid S) = \prod_i p(w_i \mid S)$$

and

$$p(D \mid \neg S) = \prod_i p(w_i \mid \neg S)$$

Using the Bayesian result above, one can write:

$$p(S \mid D) = \frac{p(S)}{p(D)} \prod_i p(w_i \mid S)$$

$$p(\neg S \mid D) = \frac{p(\neg S)}{p(D)} \prod_i p(w_i \mid \neg S)$$

Dividing one by the other gives:

$$\frac{p(S \mid D)}{p(\neg S \mid D)} = \frac{p(S) \prod_i p(w_i \mid S)}{p(\neg S) \prod_i p(w_i \mid \neg S)}$$

Which can be re-factored as:

$$\frac{p(S \mid D)}{p(\neg S \mid D)} = \frac{p(S)}{p(\neg S)} \prod_i \frac{p(w_i \mid S)}{p(w_i \mid \neg S)}$$

Thus, the probability ratio p(*S* | *D*) / p(¬*S* | *D*) can be expressed in terms of a series of likelihood ratios. The actual probability p(*S* | *D*) can be easily computed from log (p(*S* | *D*) / p(¬*S* | *D*)) based on the observation that p(*S* | *D*) + p(¬*S* | *D*) = 1.

(This technique of "log-likelihood ratios" is a common technique in statistics. In the case of two mutually exclusive alternatives (such as this example), the conversion of a log-likelihood ratio to a probability takes the form of a sigmoid curve: see logit for details.)

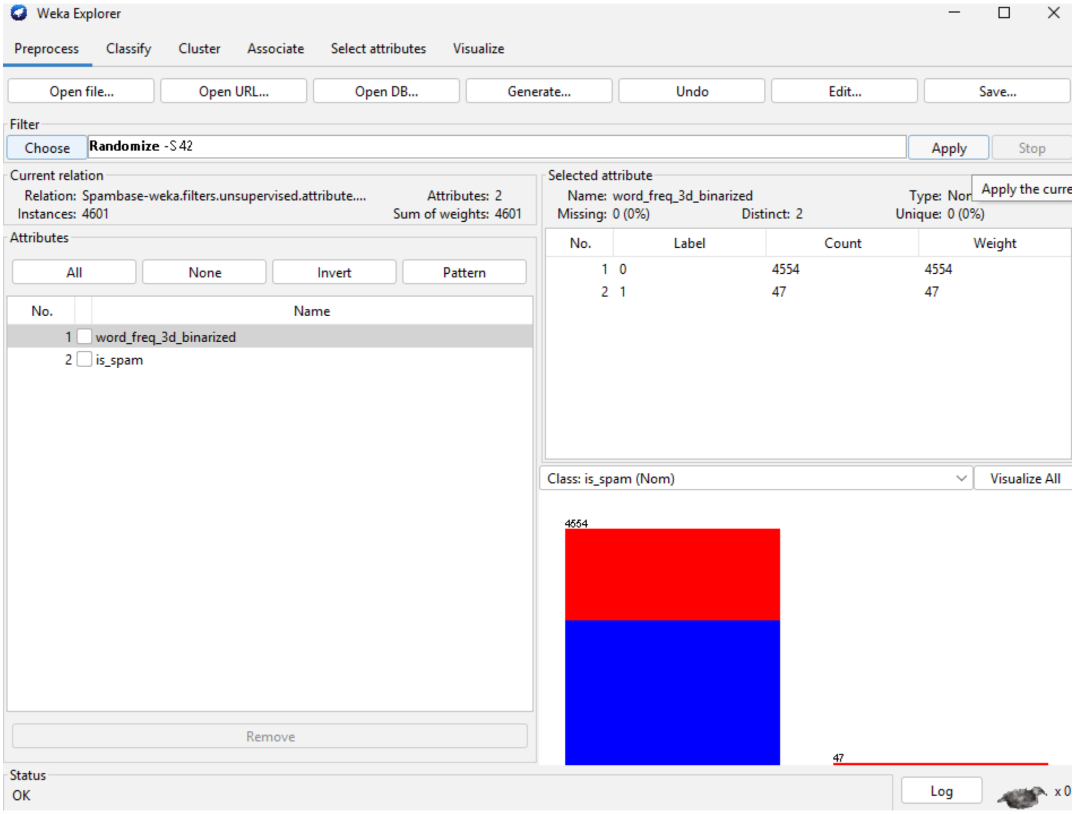
Finally, the document can be classified as follows. It is spam if $p(S \mid D) > p(\neg S \mid D)$ (i. e., $\ln \frac{p(S \mid D)}{p(\neg S \mid D)} > 0$), otherwise it is not spam.

- *Compute the conditional probability of observing the word "3d" given that an e-mail is spam P(3d|spam) and that it is non-spam P(3d|non-spam). To do this, we need to use the counts of the built model that are produced within the Classifier output screen under the Classify tab. The general format of the Weka count output is (Note: this is a toy example. You will need to examine your Weka output to find the true counts for the word "3d".):*

| | Class 0 | Class 1 |
|---|---|---|
| **0** | 1 | 2 |
| **1** | 3 | 4 |
| **total** | 4 | 6 |

Class 1 = Is Spam

***First of all we need to shuffle the data.***




***Unfortunately, I did not find where in WEKA we can do data smoothing (and we need it, because the dataset is VERY unbalanced). So let me make predictions as I can and WEKA gives me.***

```
Classifier output
Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.02 seconds

=== Summary ===

Correctly Classified Instances        1130               61.413  %
Incorrectly Classified Instances       710               38.587  %
Kappa statistic                          0.0214
Mean absolute error                      0.4736
Root mean squared error                  0.4867
Relative absolute error                 99.2023 %
Root relative squared error             99.6888 %
Total Number of Instances             1840

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,996    0,978    0,612      0,996    0,758      0,081    0,509     0,612     0
                 0,022    0,004    0,762      0,022    0,043      0,081    0,509     0,400     1
Weighted Avg.    0,614    0,596    0,671      0,614    0,478      0,081    0,509     0,529

=== Confusion Matrix ===

    a    b   <-- classified as
 1114    5 |    a = 0
  705   16 |    b = 1
```

***So we see that the prediction is just awful. In addition to imbalance we can also notice that classifying by one feature is not a good practice.***

***Absolutely, with a huge number of features the problem of the curse of dimensionality can arise; however, there is no means that we need to make a forecast for one feature.***

*4. For the final part of this section we will now pretend we are spammers wishing to fool a spam checking system based on Naïve Bayes into classifying a spam e-mail as ham (i.e. a valid e-mail). We will now use all of the training data to train our classifier and apply the learnt classifer to a dedicated test set. Load the test set in Weka. Under the Classify tab, select supplied test set > set > open file and set the test file to the supplied spambase_test.arff. This ARFF file contains the binary vector representing one spam e-mail. Run the Naïve Bayes classifer on this test set. Does the classifer classify the spam e-mail correctly?*

**YES, it identified as a spam.**

```
Classifier output
Time taken to build model: 0.01 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0 seconds

=== Summary ===

Correctly Classified Instances           1              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0.0469
Root mean squared error                  0.0469
Relative absolute error                  7.7451 %
Root relative squared error              7.7451 %
Total Number of Instances                1

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 ?        0,000    ?          ?        ?          ?        ?         ?         0
                 1,000    ?        1,000      1,000    1,000      ?        ?         1,000     1
Weighted Avg.    1,000    ?        1,000      1,000    1,000      ?        ?         1,000

=== Confusion Matrix ===

 a b   <-- classified as
 0 0 | a = 0
 0 1 | b = 1
```

*5. Open the test file [spambase_test.arff]() in emacs or another text editor. **Identify good non-spam words and add these to the e-mail.** Important:Leave the class label (last attribute value) in the test data file untouched. During testing, Weka will ignore this attribute and will instead use our previously trained classifier to predict the class label of this e-mail. Re-run the classifer on the modified test set. Has the class label (spam/non-spam) for this e-mail changed?*

**Remember: 1 if the word or character is present in the email, 0 if not.**

0,1,1,0,1,0,0,0,0,0,0,1,0,0,0,1,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1
**Turned into:** 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,0,1,1,0,0,1,0,0,1,0,1,1,1,0,1,1,1,0,0,1,1,0,0,0,1,0,0,1

**Result:**

```
Classifier output
ZeroR predicts class value: 0

Time taken to build model: 0.02 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0.01 seconds

=== Summary ===

Correctly Classified Instances           0                0      %
Incorrectly Classified Instances         1              100      %
Kappa statistic                          0
Mean absolute error                      0.6059
Root mean squared error                  0.6059
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances                1

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 ?        1,000    0,000      ?        ?          ?        ?         ?         0
                 0,000    ?        ?          0,000    ?          ?        ?         1,000     1
Weighted Avg.    0,000    ?        ?          0,000    ?          ?        ?         1,000

=== Confusion Matrix ===

 a b   <-- classified as
 0 0 | a = 0
 1 0 | b = 1
```

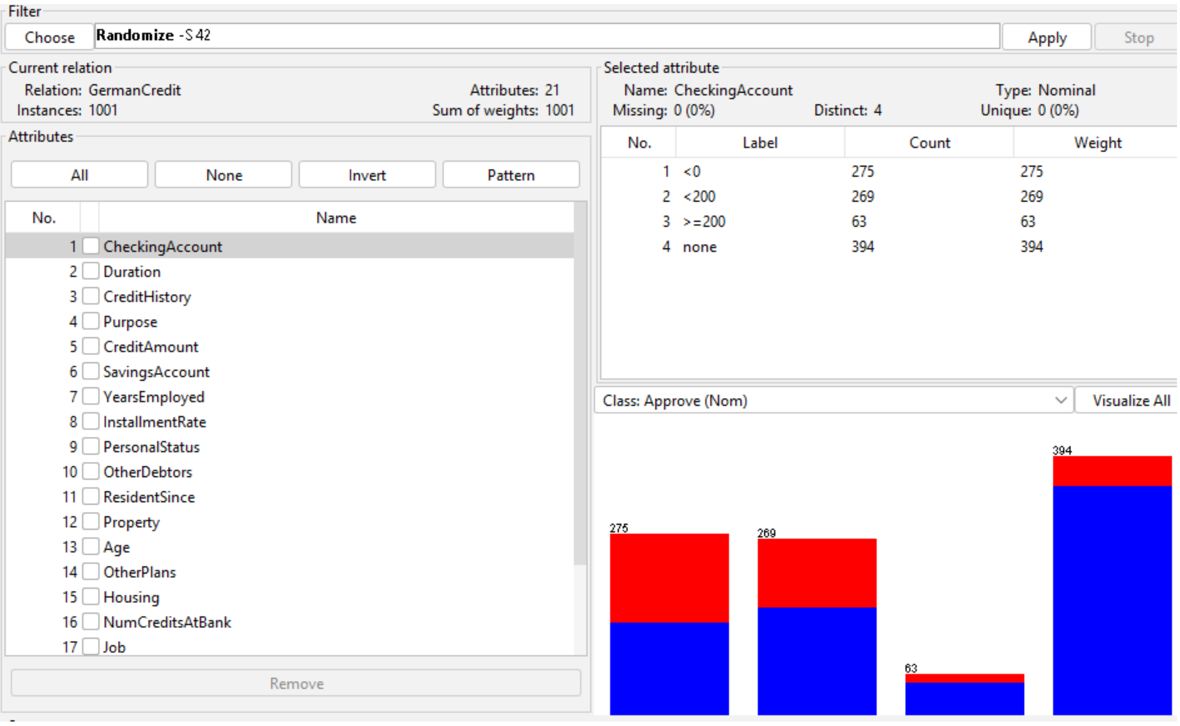
***It is now identified as a non-spam email.***

*You've now managed to switch the predicted class label for that e-mail. Adding more “hammy” words to this e-mail has sufficiently increased the probability that this e-mail is ham so the classifier now outputs "ham" as the e-mail's class label (by changing the word content of the e-mail you have added extra evidence or "votes" towards this e-mail being classified as ham). This is the "stuffing" example given in the lectures and is directly caused by the independence assumption that is made by Naive Bayes. Each word contributes independently of each other to the final score. This is a reason that a lot of spam e-mails include random excerpts from the passages of books so as to effectively add “ hammy” words in the hope that the spam e-mail will bypass the spam filters. For this reason, in practice, many commercial e-mail systems (consider Gmail) likely use a lot more sophisticated spam detection models.*

## 2. DECISION TREES

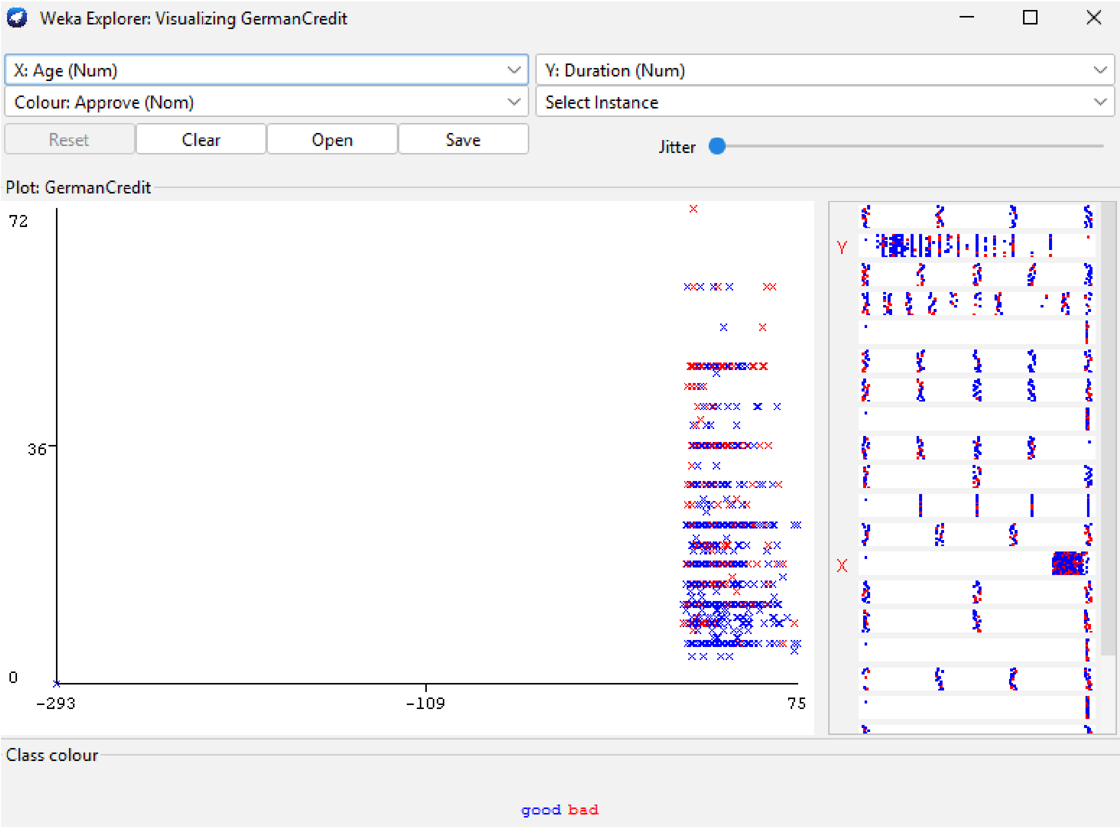
One of the great advantages of decision trees is their *interpretability*. The rules learnt for classification are easy for a person to follow, unlike the opaque "black box" of many other methods, such as neural networks. We demonstrate the utility of this using a German credit data set. You can read a description of this dataset at the UCI site. The task is to predict whether a loan approval is good or bad credit risk based on 20 attributes. We've simplified the data set somewhat, particularly making attribute names and values more meaningful.

1. Download the credit.arff dataset and load it to Weka.

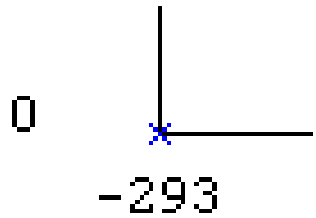



2. When presented with a dataset, it is usually a good idea to visualise it first. Go to the *Visualise* tab. Click on any of the scatter plots to open a new window which shows the scatter plot for two selected attributes. Try visualising a scatter plot of *age* and *duration*. Do you notice anything unusual? You can click on any data point to display all it's values.

**As can be seen from the plot, people more likely take short-term credits then long-term.**




**If we look at the plot with special observation, then in the lower left corner we can see the tick -293 years, which is absurd because the age is always greater than 0.**




3. In the previous point you should have found a data point, which seems to be corrupted, as some of its values are nonsensical. Even a single point like this can significantly affect the performance of a classifier. How do you think it would affect Decision trees? How about Naive Bayes? A good way to check this is to test the performance of each classifier before and after removing this datapoint.

   **Decision Trees are not sensitive to outliers, a different situation with Naive Bayes. However, if we are talking about the -293 point, then this does not seem to be an outlier, but a bug in the database, and it needs to be fixed.**

   **Thus, to argue for deleting point -293, I want to use Decision Trees and Naive Bayes methods with and without that point.**

### *Decision Trees with -293 point:*

```
Classifier output
Time taken to build model: 0.03 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances          66               66      %
Incorrectly Classified Instances        34               34      %
Kappa statistic                          0.1414
Mean absolute error                      0.3934
Root mean squared error                  0.5225
Relative absolute error                 93.6686 %
Root relative squared error            114.0188 %
Total Number of Instances              100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,800    0,667    0,737      0,800    0,767      0,143    0,572     0,745     good
                 0,333    0,200    0,417      0,333    0,370      0,143    0,572     0,339     bad
Weighted Avg.    0,660    0,527    0,641      0,660    0,648      0,143    0,572     0,623

=== Confusion Matrix ===

  a  b   <-- classified as
 56 14 |  a = good
 20 10 |  b = bad
```

### *Decision Trees without -293 point:*

```
Classifier output

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances          76               76      %
Incorrectly Classified Instances        24               24      %
Kappa statistic                          0.2381
Mean absolute error                      0.3155
Root mean squared error                  0.4188
Relative absolute error                 78.2823 %
Root relative squared error             95.9154 %
Total Number of Instances              100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,920    0,720    0,793      0,920    0,852      0,258    0,735     0,862     good
                 0,280    0,080    0,538      0,280    0,368      0,258    0,735     0,503     bad
Weighted Avg.    0,760    0,560    0,729      0,760    0,731      0,258    0,735     0,772

=== Confusion Matrix ===

  a  b   <-- classified as
 69  6 |  a = good
 18  7 |  b = bad
```

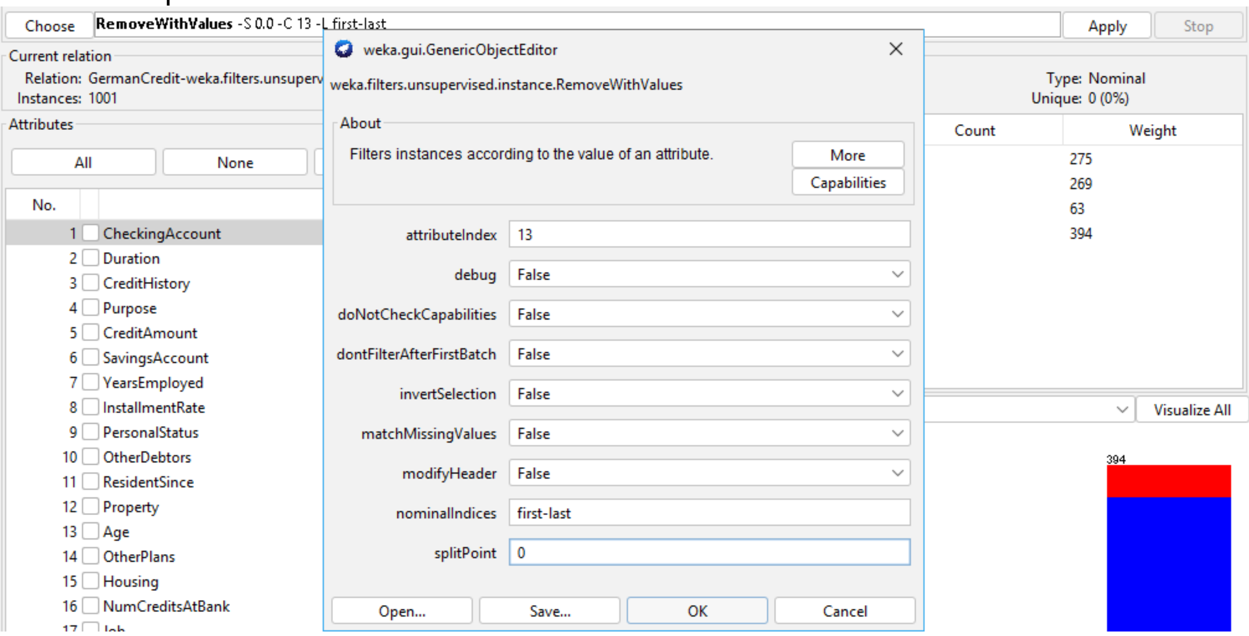
## *Naive Bayes with -293 point:*

```
Classifier output
Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances          37               37      %
Incorrectly Classified Instances        63               63      %
Kappa statistic                          0.0625
Mean absolute error                      0.5774
Root mean squared error                  0.701
Relative absolute error                137.4592 %
Root relative squared error            152.9814 %
Total Number of Instances              100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,100    0,000    1,000      0,100    0,182      0,180    0,794     0,907     good
                 1,000    0,900    0,323      1,000    0,488      0,180    0,794     0,542     bad
Weighted Avg.    0,370    0,270    0,797      0,370    0,274      0,180    0,794     0,798

=== Confusion Matrix ===

  a  b   <-- classified as
  7 63 |  a = good
  0 30 |  b = bad
```

## *Naive Bayes without -293 point:*

```
Classifier output

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances          82               82      %
Incorrectly Classified Instances        18               18      %
Kappa statistic                          0.493
Mean absolute error                      0.2624
Root mean squared error                  0.3811
Relative absolute error                 65.1028 %
Root relative squared error             87.2885 %
Total Number of Instances              100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,907    0,440    0,861      0,907    0,883      0,496    0,805     0,906     good
                 0,560    0,093    0,667      0,560    0,609      0,496    0,805     0,602     bad
Weighted Avg.    0,820    0,353    0,812      0,820    0,815      0,496    0,805     0,830

=== Confusion Matrix ===

  a  b   <-- classified as
 68  7 |  a = good
 11 14 |  b = bad
```

4. To remove this instance from the dataset we will use a filter. We want to remove all instances, where the age of an applicant is lower than 0 years, as this suggests that the instance is corrupted. In the *Preprocess* tab click on *Choose* in the Filter pane. Select *filters > unsupervised > instance > RemoveWithValues*. Click on the text of this filter to change the parameters. Set the attribute index to 13 (Age) and set the split point at 0. Click *Ok* to set the parameters and *Apply* to apply the filter to the data. Visualise the data again to verify that the invalid data point was removed.





5. On the *Classify* tab, select the ***Percentage split test option and change its value to 90%***. This way, we will train the classifiers using 90% of the training data and evaluate their performance on the remaining 10%.

6. First, train a decision tree classifier with default options. Select *classifiers* > *trees* > *J48* and click *Start*. *J48* is the Weka implementation of the *C4.5* algorithm, which uses the normalized information gain criterion to build a decision tree for classification.

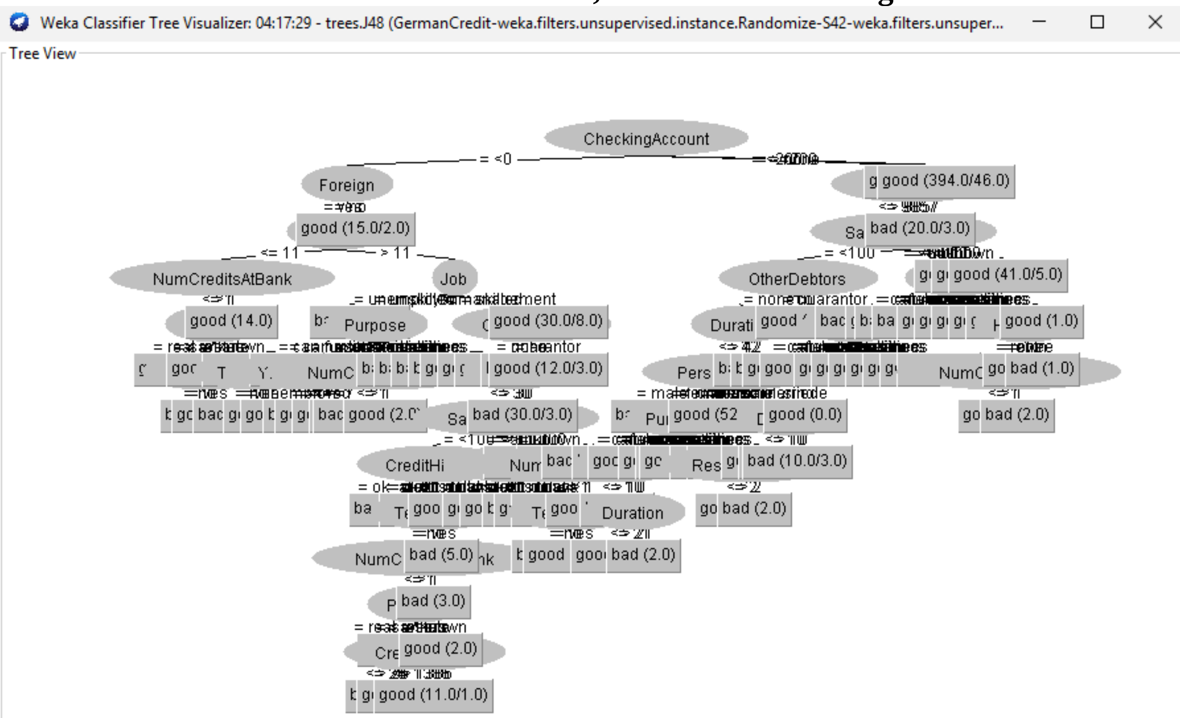
```
Classifier output


Time taken to build model: 0.14 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances          76               76      %
Incorrectly Classified Instances        24               24      %
Kappa statistic                          0.2381
Mean absolute error                      0.3155
Root mean squared error                  0.4188
Relative absolute error                 78.2823 %
Root relative squared error             95.9154 %
Total Number of Instances              100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,920    0,720    0,793      0,920    0,852      0,258    0,735     0,862     good
                 0,280    0,080    0,538      0,280    0,368      0,258    0,735     0,503     bad
Weighted Avg.    0,760    0,560    0,729      0,760    0,731      0,258    0,735     0,772

=== Confusion Matrix ===

  a  b   <-- classified as
 69  6 |  a = good
 18  7 |  b = bad
```

7. After training the classifier, the full decision tree is output for your perusal; you may need to scroll up for this. The tree may also be viewed in graphical form by right-clicking in the *Result list* and selecting *Visualize tree*; unfortunately this format is very cluttered for large trees. Such a tree accentuates one of the strengths of decision tree algorithms: they produce classifiers which are understandable to humans. This can be an important asset in real life applications (people are seldom prepared to do what a computer program tells them if there is no clear explanation).

Classifier output

```
J48 pruned tree
------------------

CheckingAccount = <0
|   Foreign = yes
|   |   Duration <= 11
|   |   |   NumCreditsAtBank <= 1
|   |   |   |   Property = real_estate: good (8.0/1.0)
|   |   |   |   Property = savings
|   |   |   |   |   Telephone = no: bad (2.0)
|   |   |   |   |   Telephone = yes: good (4.0)
|   |   |   |   Property = car: good (2.0/1.0)
|   |   |   |   Property = unknown: bad (3.0)
|   |   |   NumCreditsAtBank > 1: good (14.0)
|   |   Duration > 11
|   |   |   Job = unemployed: bad (5.0/1.0)
|   |   |   Job = unskilled
|   |   |   |   Purpose = car_new
|   |   |   |   |   Telephone = no: bad (10.0/2.0)
|   |   |   |   |   Telephone = yes: good (2.0)
|   |   |   |   Purpose = car_used: bad (1.0)
|   |   |   |   Purpose = furniture
|   |   |   |   |   YearsEmployed = unemployed: good (0.0)
|   |   |   |   |   YearsEmployed = <1: bad (3.0)
|   |   |   |   |   YearsEmployed = <4: good (4.0)
|   |   |   |   |   YearsEmployed = <7: good (1.0)
|   |   |   |   |   YearsEmployed = >=7: good (2.0)
|   |   |   |   Purpose = television
|   |   |   |   |   NumCreditsAtBank <= 1: bad (10.0/3.0)
|   |   |   |   |   NumCreditsAtBank > 1: good (2.0)
```

**I tried to visualize this tree, but it does not look good.**




Observe the output of the classifier and try to answer the following questions:

- How would you assess the performance of the classifier? Is the Percentage of *Correctly Classified Instances* a sufficient measure in this case? Why? *Hint:* check the number of good and bad cases in the test sample, using the confusion matrix. Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. For example let us define an experiment from P positive instances and N negative instances. The four outcomes can be formulated in a 2 by 2 contingency table or confusion matrix, the breakdown of which is given below:

| | | actual value | | |
|---|---|---|---|---|
| | | **p** | **n** | **total** |
| **prediction outcome** | **p'** | True Positive | False Positive | P' |
| | **n'** | False Negative | True Negative | N' |
| **total** | | P | N | |

**I would like to say that the accuracy of classifier is not bad. Nevertheless, we see that there is a high level of Type II error. From a mathematical point of view, this is understandable, because we either have the error of the Type I, or II.**

```
=== Confusion Matrix ===

  a  b   <-- classified as
 69  6 |  a = good
 18  7 |  b = bad
```

- One benefit of a confusion matrix is that it is easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).
- Looking at the decision tree itself, are the rules it applies sensible? Are there any branches which appear absurd? At what depth of the tree? What does this suggest? *Hint:* Check the rules applied after following the paths: (a) *CheckingAccount = <0, Foreign = yes, Duration >11, Job = skilled, OtherDebtors = none, Duration <= 30* and (b) *CheckingAccount = <0, Foreign = yes, Duration >11, Job = unskilled.*

**It is strange that having more than one loan with a skilled worker is seen as a reason for refusing credit when we see the opposite situation with unskilled workers. I believe that a good credit history tells us a lot, therefore when we are talking about people who have skilled jobs, it means that they can buy what they want or they can let themself to take a credit because they are hundred per cent sure that they will repay the loan.**

**If a skilled employee has a phone, it is strange not to give him a loan – I think so.**

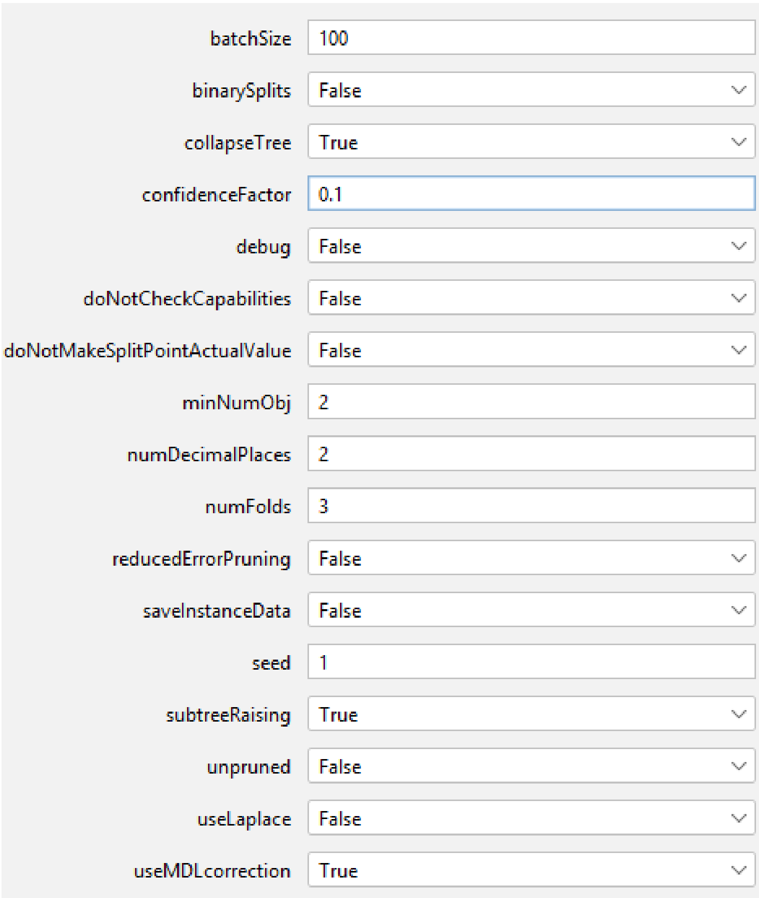
```
Job = skilled
|   OtherDebtors = none
|   |   Duration <= 30
|   |   |   SavingsAccount = <100
|   |   |   |   CreditHistory = ok: bad (8.0/1.0)
|   |   |   |   CreditHistory = ok_at_this_bank: bad (6.0)
|   |   |   |   CreditHistory = ok_til_now
|   |   |   |   |   Telephone = no
|   |   |   |   |   |   NumCreditsAtBank <= 1
|   |   |   |   |   |   |   Property = real_estate
|   |   |   |   |   |   |   |   Age <= 26: bad (5.0)
|   |   |   |   |   |   |   |   Age > 26: good (2.0)
|   |   |   |   |   |   |   Property = savings: bad (7.0/2.0)
|   |   |   |   |   |   |   Property = car
|   |   |   |   |   |   |   |   CreditAmount <= 1386: bad (3.0)
|   |   |   |   |   |   |   |   CreditAmount > 1386: good (11.0/1.0)
|   |   |   |   |   |   |   Property = unknown: good (2.0)
|   |   |   |   |   |   NumCreditsAtBank > 1: bad (3.0)
|   |   |   |   |   Telephone = yes: bad (5.0)
|   |   |   |   CreditHistory = past_delays: bad (4.0)
|   |   |   |   CreditHistory = critical: good (14.0/4.0)
|   |   |   SavingsAccount = <500
|   |   |   |   CreditHistory = ok: good (0.0)
|   |   |   |   CreditHistory = ok_at_this_bank: good (1.0)
|   |   |   |   CreditHistory = ok_til_now: bad (3.0)
|   |   |   |   CreditHistory = past_delays: good (0.0)
|   |   |   |   CreditHistory = critical: good (2.0)
|   |   |   SavingsAccount = <1000: good (4.0/1.0)
|   |   |   SavingsAccount = >=1000: good (4.0)
|   |   |   SavingsAccount = unknown
|   |   |   |   NumCreditsAtBank <= 1
|   |   |   |   |   Telephone = no: bad (9.0/1.0)
|   |   |   |   |   Telephone = yes: good (4.0/1.0)
|   |   |   |   NumCreditsAtBank > 1: good (2.0)
```

```
Job = unskilled
|   Purpose = car_new
|   |   Telephone = no: bad (10.0/2.0)
|   |   Telephone = yes: good (2.0)
|   Purpose = car_used: bad (1.0)
|   Purpose = furniture
|   |   YearsEmployed = unemployed: good (0.0)
|   |   YearsEmployed = <1: bad (3.0)
|   |   YearsEmployed = <4: good (4.0)
|   |   YearsEmployed = <7: good (1.0)
|   |   YearsEmployed = >=7: good (2.0)
|   Purpose = television
|   |   NumCreditsAtBank <= 1: bad (10.0/3.0)
|   |   NumCreditsAtBank > 1: good (2.0)
|   Purpose = appliances: bad (1.0)
|   Purpose = repairs: bad (1.0)
|   Purpose = education: bad (1.0)
|   Purpose = vacation: bad (0.0)
|   Purpose = retraining: good (1.0)
|   Purpose = business: good (3.0)
|   Purpose = others: good (1.0)
```

- How does the decision tree deal with classification in the case where there are zero instances in the training set corresponding to that particular path in the tree (e.g. those leaf nodes that have (0:0))?

  **Most likely wrong in the answer, so we need to minimize the entropy and take into account regularization techniques.**

8. Now, explore the effect of the *confidenceFactor* option. You can find this by clicking on the Classifer name (to the right of the *Choose* button on the Classify tab). On the *Classifier options* window, click on the *More* button to find out what the confidence factor controls. Try the values 0.1, 0.2, 0.3 and 0.5. What is the performance of the classifier at each case?




## What is the confidence factor weka?

The default J48 decision tree in Weka uses **pruning (deleting unnecessary nodes)** based on subtree raising, confidence factor of 0.25, minimal number of objects is set to 2, and nodes can have multiple splits.

*If percentage confidence factor = 0.1, then*

Classifier output

```
=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances          73               73      %
Incorrectly Classified Instances        27               27      %
Kappa statistic                          0.1562
Mean absolute error                      0.3452
Root mean squared error                  0.4173
Relative absolute error                 85.66   %
Root relative squared error             95.5697 %
Total Number of Instances              100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,893    0,760    0,779      0,893    0,832      0,166    0,710     0,860     good
                 0,240    0,107    0,429      0,240    0,308      0,166    0,710     0,418     bad
Weighted Avg.    0,730    0,597    0,691      0,730    0,701      0,166    0,710     0,750

=== Confusion Matrix ===

  a  b   <-- classified as
 67  8 |  a = good
 19  6 |  b = bad
```

*If percentage confidence factor = 0.2, then*

Classifier output

```
=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances          75               75      %
Incorrectly Classified Instances        25               25      %
Kappa statistic                          0.2647
Mean absolute error                      0.3222
Root mean squared error                  0.4087
Relative absolute error                 79.9394 %
Root relative squared error             93.6089 %
Total Number of Instances              100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,880    0,640    0,805      0,880    0,841      0,271    0,739     0,878     good
                 0,360    0,120    0,500      0,360    0,419      0,271    0,739     0,505     bad
Weighted Avg.    0,750    0,510    0,729      0,750    0,735      0,271    0,739     0,785

=== Confusion Matrix ===

  a  b   <-- classified as
 66  9 |  a = good
 16  9 |  b = bad
```

*If percentage confidence factor = 0.3, then*

```
Classifier output

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances          76               76      %
Incorrectly Classified Instances        24               24      %
Kappa statistic                          0.2615
Mean absolute error                      0.3011
Root mean squared error                  0.4131
Relative absolute error                 74.7118 %
Root relative squared error             94.6155 %
Total Number of Instances              100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,907    0,680    0,800      0,907    0,850      0,275    0,750     0,866     good
                 0,320    0,093    0,533      0,320    0,400      0,275    0,750     0,524     bad
Weighted Avg.    0,760    0,533    0,733      0,760    0,738      0,275    0,750     0,781

=== Confusion Matrix ===

  a  b   <-- classified as
 68  7 |  a = good
 17  8 |  b = bad
```

*If percentage confidence factor = 0.5, then*

```
Classifier output

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances          75               75      %
Incorrectly Classified Instances        25               25      %
Kappa statistic                          0.2857
Mean absolute error                      0.2899
Root mean squared error                  0.4487
Relative absolute error                 71.9332 %
Root relative squared error            102.7715 %
Total Number of Instances              100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,867    0,600    0,813      0,867    0,839      0,289    0,671     0,823     good
                 0,400    0,133    0,500      0,400    0,444      0,289    0,671     0,431     bad
Weighted Avg.    0,750    0,483    0,734      0,750    0,740      0,289    0,671     0,725

=== Confusion Matrix ===

  a  b   <-- classified as
 65 10 |  a = good
 15 10 |  b = bad
```

Did you expect this given your observations in the previous questions? Why do you think this happens?

**In my opinion, this was due to a slight overfitting during training, and I believe that 0.25 is the best parameter for the confidence factor.**

9. Suppose that it is worse to classify a customer as good when they are bad, than it is to classify a customer as bad when they are good. Which value would you pick for the confidence factor? Which performance measure would you base your decision on?

**In this case we are talking about a Type II error. As we saw in this lab (see the screenshots above), with a parameter of 0.1 the error is slightly larger.**

*If you have time:* Finally we will create a *random decision forest* and compare the performance of this classifier to that of the decision tree and the decision stump. The random decision forest is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees. Again set the test option *Percentage split* to 90%. Select *classifiers > trees > RandomForest* and hit *Start*. Again, observe the output.

**<u>DEF</u>. A decision stump is a machine learning model consisting of a one-level decision tree. That is, it is a decision tree with one internal node (the root) which is immediately connected to the terminal nodes (its leaves).**

Classifier output

```
=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances          85               85      %
Incorrectly Classified Instances        15               15      %
Kappa statistic                          0.5714
Mean absolute error                      0.2947
Root mean squared error                  0.3498
Relative absolute error                 73.1338 %
Root relative squared error             80.1218 %
Total Number of Instances              100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,933    0,400    0,875      0,933    0,903      0,577    0,897     0,959     good
                 0,600    0,067    0,750      0,600    0,667      0,577    0,897     0,794     bad
Weighted Avg.    0,850    0,317    0,844      0,850    0,844      0,577    0,897     0,918

=== Confusion Matrix ===

  a  b   <-- classified as
 70  5 |  a = good
 10 15 |  b = bad
```

How high can you get the performance of the classifier by changing the number of trees (numTrees) parameter?

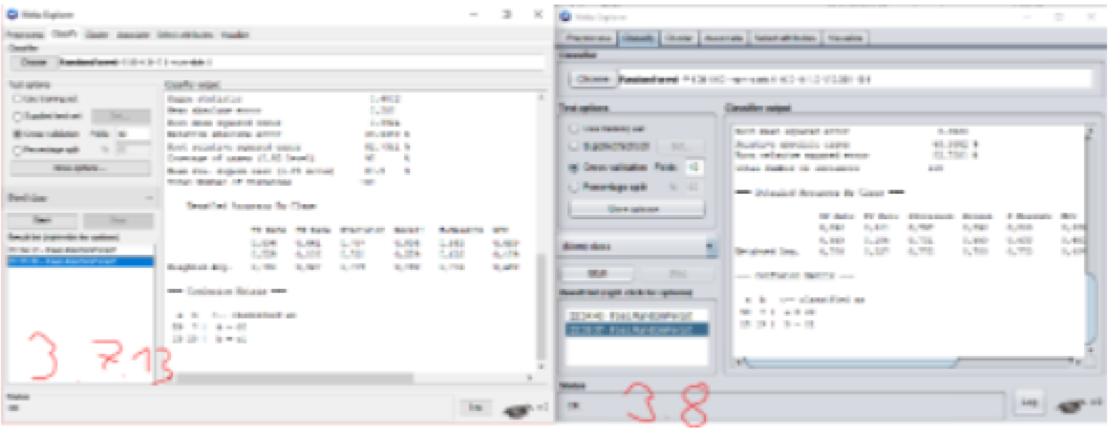
## There is the need to clarify something:

`setNumTrees(int)` `getNumTrees()` and `m_numTrees` pre [weka 3.7.12]

`setNumIterations(int)` in [weka 3.8] it was renamed, apparently in an effort to implement a RandomForest using the `Bagging` classifier with `RandomTree` as underlying implementation.

I'm going by the fact that both use the `-I` tag.

Also I checked, for `-I = 100` and `-I = 10` they have the same output for some random sample data.



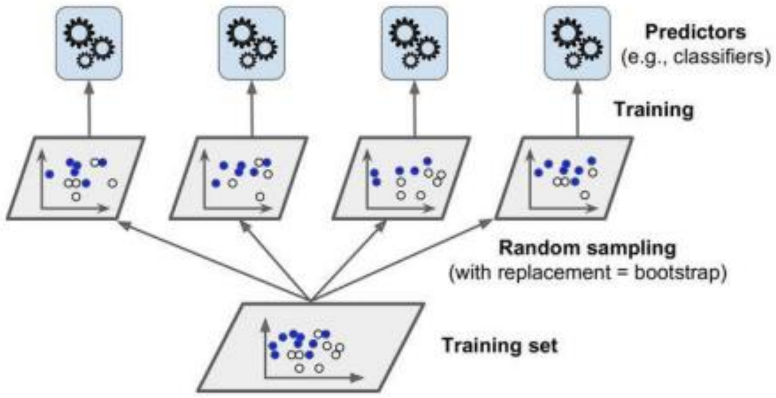

## *The best result was obtained with 150 trees.*

```
Classifier output

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances          86               86      %
Incorrectly Classified Instances        14               14      %
Kappa statistic                          0.5942
Mean absolute error                      0.3018
Root mean squared error                  0.3579
Relative absolute error                 74.8913 %
Root relative squared error             81.9823 %
Total Number of Instances              100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,947    0,400    0,877      0,947    0,910      0,603    0,878     0,953     good
                 0,600    0,053    0,789      0,600    0,682      0,603    0,878     0,721     bad
Weighted Avg.    0,860    0,313    0,855      0,860    0,853      0,603    0,878     0,895

=== Confusion Matrix ===

  a  b   <-- classified as
 71  4 |  a = good
 10 15 |  b = bad
```

| | |
|---|---|
| bagSizePercent | 100 |
| batchSize | 100 |
| breakTiesRandomly | False |
| calcOutOfBag | False |
| computeAttributeImportance | False |
| debug | False |
| doNotCheckCapabilities | False |
| maxDepth | 0 |
| numDecimalPlaces | 2 |
| numExecutionSlots | 1 |
| numFeatures | 0 |
| numIterations | 150 |
| outputOutOfBagComplexityStatistics | False |
| printClassifiers | False |
| seed | 86 |
| storeOutOfBagPredictions | False |

How does the random decision forest compare performance wise to the decision tree and decision stump?

You can use the same training algorithm for every predictor. But to train them on different random subset of the training set.




Random Forest is an ensemble of Decision Trees, trained via the bagging method

**The RF algorithm introduces extra randomness when growing trees: instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features.**