Probabilistic Reasoning with Naïve Bayes and Bayesian Networks

Zdravko Markov[1], Ingrid Russell
July, 2007

Overview

Bayesian (also called Belief) Networks (BN) are a powerful knowledge representation and reasoning mechanism. BN represent events and causal relationships between them as conditional probabilities involving random variables. Given the values of a subset of these variables (evidence variables) BN can compute the probabilities of another subset of variables (query variables). BN can be created automatically (learnt) by using statistical data (examples). The well-known Machine Learning algorithm, Naïve Bayes is actually a special case of a Bayesian Network.

The project allows students to experiment with and use the Naïve Bayes algorithm and Bayesian Networks to solve practical problems. This includes collecting data from real domains (e.g. web pages), converting these data into proper format so that conditional probabilities can be computed, and using Bayesian Networks and the Naïve Bayes algorithm for computing probabilities and solving classification tasks.

Objectives

The aim of this project is to expose students to two important reasoning and learning algorithms – Naïve Bayes and Bayesian Networks, and to explore their relationship in the context of solving practical classification problems. In particular, the objectives of the project are:

- Learning the basics of Bayesian approach to Machine Learning and the Bayesian Networks approach to Probabilistic Reasoning in AI.
- Gaining experience in using recent software applications in these areas for solving practical problems.
- Better understanding of fundamental concepts of Bayesian Learning and Probabilistic Reasoning and their relationship in the more general context of knowledge representation and reasoning mechanisms in AI.

Project Description

[1] Corresponding author: markovz@ccsu.edu, Department of Computer Science, Central Connecticut State University, 1615 Stanley Street, New Britain, CT 06050.

Similarly to the Web document classification project (http://uhaweb.hartford.edu/compsci/ccli/wdc.htm) this project also has three main steps: Data collection, Data preparation, and Machine Learning. The first two steps of the two projects are basically the same. In fact, documents and data sets in Weka's ARFF format prepared for the former project can also be used for the Bayesian reasoning project. Hereafter we shall describe these steps again, because there are some differences in the software tools used which make the first two steps of the current project more straightforward. The basic difference is in the third step, which includes Bayesian classification with experiments with Bayesian networks.

Data Collection

Generally the data collection step in Machine Learning involves gathering cases, examples, or instances of objects from different types or categories (classes), so that at the ML step a model of these data is created that can be later used to identify groups or similarities in data (in unsupervised learning) or predict the class of new objects (in supervised learning). Although any other objects may be used for the purposes of this project we suggest that web or text documents are collected. The area of text/web document classification has been used in other ML projects and students who have worked on these projects have already gained some experience in this area. Results from other projects may be compared to the results from Bayesian learning which would show the advantages and drawbacks of different algorithms applied to the same data. And last but not least, text documents can be easily collected from the Web along with their class labels (topics or user preferences). This issue was discussed in detail in other two ML projects related to the Web - Web document classification (http://uhaweb.hartford.edu/compsci/ccli/wdc.htm) and Web user profiling (http://uhaweb.hartford.edu/compsci/ccli/wup.htm).

To illustrate the process of data collection we use a small part of the Web, a set of pages describing the departments in the school of Arts and Sciences at CCSU. Each of these pages is a short textual description of the department. Figure 1 shows a directory page that includes links to the department pages. One of the department pages is shown in Figure 2.

As we are going to classify these and also new documents into categories at this point we have to determine a class label for each department page. These labels can be given independently from the document content however as the classification will be based on the document content some semantic mapping between content and class should be established. For this purpose we can roughly split the departments into two groups – sciences (class label A) and humanities (class label B). Table 1 shows these two groups (the department names are abbreviated).

Documents	**Class (A – sciences, B – humanities)**
Anthropology, Biology, Chemistry, Computer, Economics, Geography, Math, Physics,	A

Political, Psychology, Sociology	
Justice, Languages, Music, Philosophy, Communication, English, Art, History, Theatre	B

Table 1. Documents groped by classes

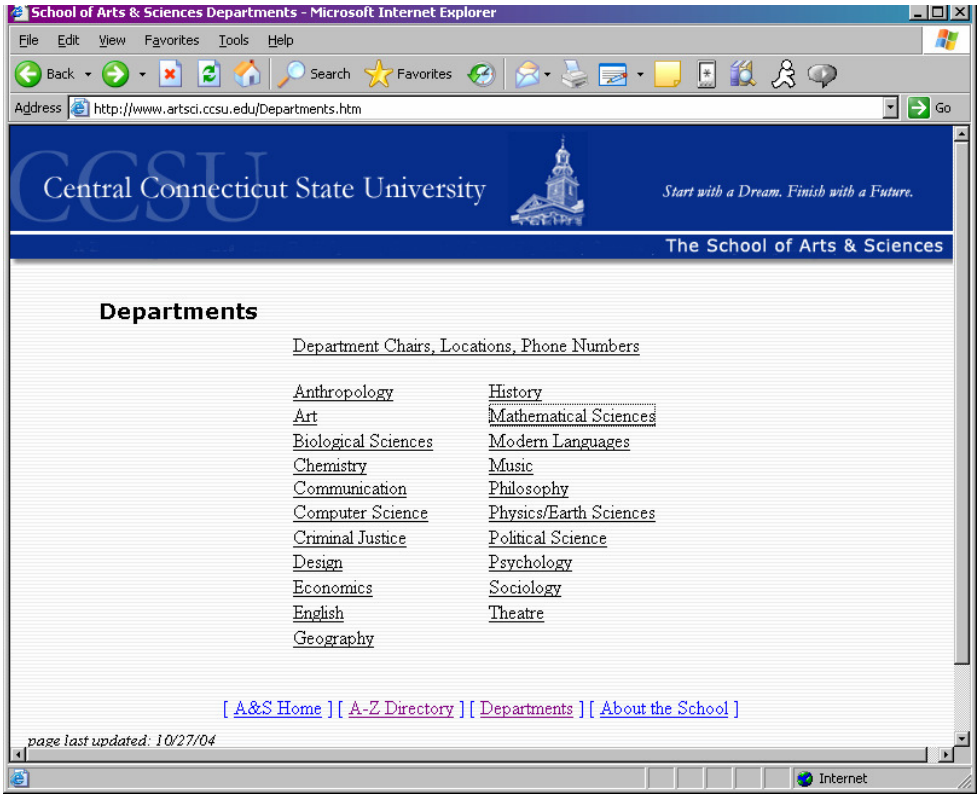


Figure 1. A directory page for a collection of web documents.

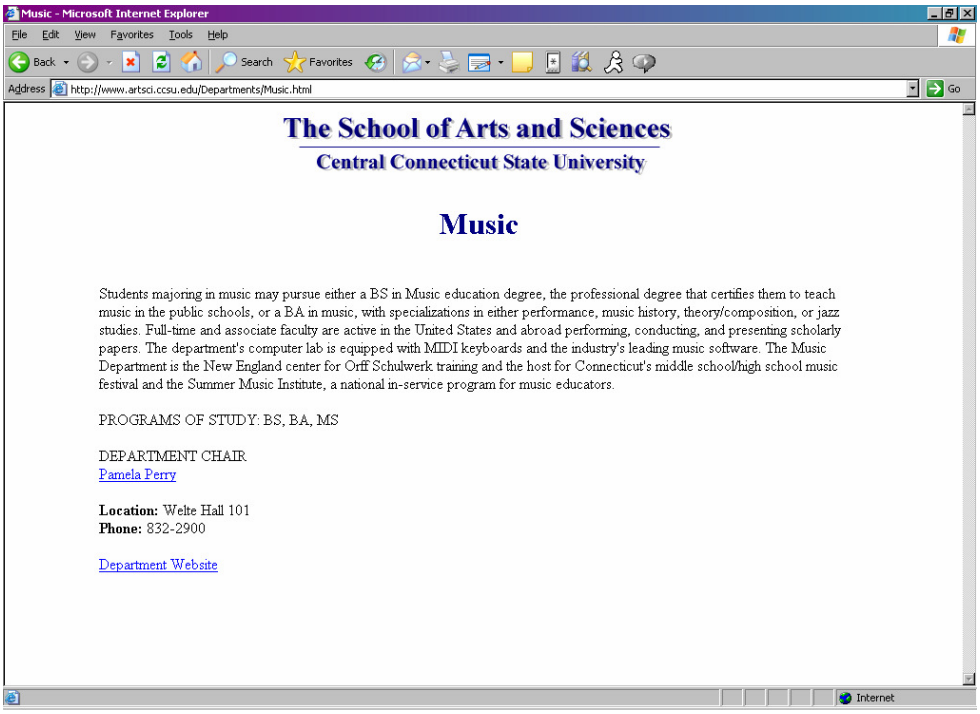


Figure 2. A sample web document

Finally, the web documents are converted into plain text files. The idea is to take off the HTML tags and leave the text only (although the HTML structure can carry some semantics associated with the document class we ignore it to simplify the processing and document representation). The conversion to text can be done is various ways manually or using some tools. For small number of documents we suggest using the Internet Explorer, which allows the currently loaded HTML document to be saved with the "Save As…" option with "Save as type: Text File (*.txt)".

The **deliverable** for the data collection step is a set of plain text files grouped into categories. As an illustration we suggest that students look into the collection of text files describing the A&S departments available from the data repository at http://www.cs.ccsu.edu/~markov/dmwdata.zip, folder "CCSU departments".

Data Preparation

Now we are going to convert the text documents into data sets in Weka's ARFF format to be later used in the Machine Learning phase. With this conversion we will actually illustrate the basic concepts of the **Vector Space document model** that plays an important role in **Information Retrieval** and **Web Search**. We suggest that students read Chapter 1 of book [2] (available for free download from Wiley) before continuing with the steps described below:

1. **Download and install the Weka data mining system (version Weka 3.4)** (http://www.cs.waikato.ac.nz/~ml/weka/). Read the documentation and try some

examples to familiarize yourself with its use (e.g. load and explore the weather data, provided with the installation).

2. **Create a string data file in ARFF format** (see the description of the ARFF format at http://www.cs.waikato.ac.nz/~ml/weka/arff.html). Follow the directions below:

   First create a **concatenation of all text documents** (**text corpus**) obtained from the data collection step and save them in a single text file, where each document is represented on a separate line in plain text format. For example, this can be done by loading all text files in MS Word and then saving the file in plain text format without line breaks. Other editors may be used for this purpose too. Students with programming experience may want to write a program to automate this step.

   Once the file with the text corpus is created enclose each line in it (an individual document content) in quotation marks (“) and add the **document name** in the beginning of the line and the **document class** at the end, all separated by commas. Also add **a file header** in the beginning of the file followed by @data as shown below:

   @relation departments_string

   @attribute document_name string
   @attribute document_content string
   @attribute document_class string

   @data

   Anthropology, " anthropology anthropology anthropology consists …”, A
   …

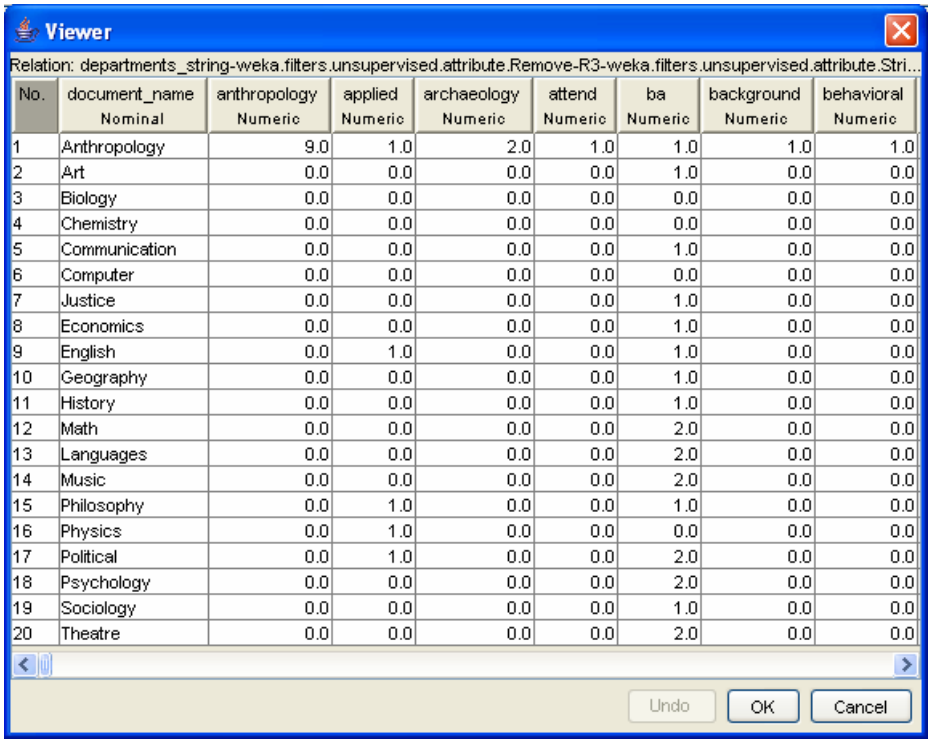
   This representation uses three attributes – document_name, document_content, and document_class, all of type string. Each row in the data section (after @data) represents one of the initial text documents. Note that the number of attributes and the order in which they are listed in the header should correspond to the comma separated items in the data section. An example of such string data file is “Departments-string.arff”, available from the data repository at http://www.cs.ccsu.edu/~markov/dmwdata.zip, folder “Weka data”.

3. **Create Term counts, Boolean, and TFIDF data sets**. Load the string data file in Weka using the “Open file” button in “Preprocess” mode. After successful loading the system shows some statistics about the number of attributes (3) their type (string) and the number of instances (rows in the data section or documents).

   Choose the **StringToNominal** filter and apply it (one at a time) to the first attribute, document_name and then to the last attribute (index 3), document_class. Then choose the **StringToWordVector** filter and apply it with outputWordCounts=true. You may

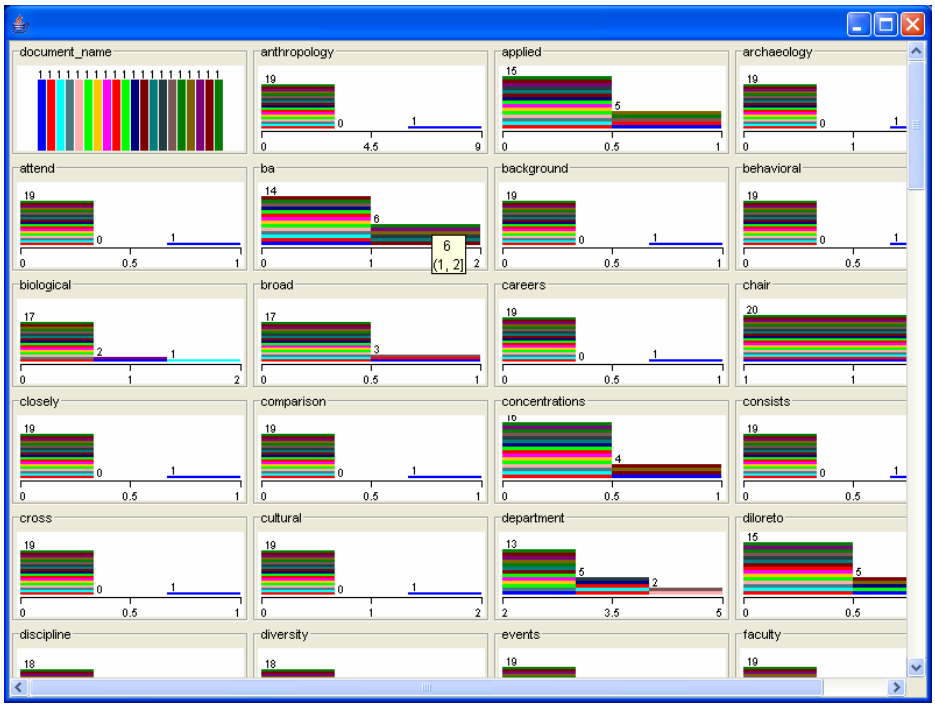
also change the setting of onlyAlphabeticTokens and useStoplist to see how the results change. As Weka moves the class attribute at the second place, move it back last by using the **Copy** filter and the **Remove** button. The result of all these steps is a Weka data set that uses a term count representation for the documents. An example of this data set is the file "Departments-counts.arff", available from the data repository at http://www.cs.ccsu.edu/~markov/dmwdata.zip, folder "Weka data".

Now we have a **document-term matrix** loaded in Weka. Press the "Edit" button to see it in a tabular format, where you can also change its content or copy it to other applications (e.g. MS Excel). Once created in Weka the table can be stored in an ARFF file through the "Save" option. Below is a screenshot of a part of the document-term table for the CCSU A@S departments.



No.	document_name Nominal	anthropology Numeric	applied Numeric	archaeology Numeric	attend Numeric	ba Numeric	background Numeric	behavioral Numeric
1	Anthropology	9.0	1.0	2.0	1.0	1.0	1.0	1.0
2	Art	0.0	0.0	0.0	0.0	1.0	0.0	0.0
3	Biology	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	Chemistry	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	Communication	0.0	0.0	0.0	0.0	1.0	0.0	0.0
6	Computer	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	Justice	0.0	0.0	0.0	0.0	1.0	0.0	0.0
8	Economics	0.0	0.0	0.0	0.0	1.0	0.0	0.0
9	English	0.0	1.0	0.0	0.0	1.0	0.0	0.0
10	Geography	0.0	0.0	0.0	0.0	1.0	0.0	0.0
11	History	0.0	0.0	0.0	0.0	1.0	0.0	0.0
12	Math	0.0	0.0	0.0	0.0	2.0	0.0	0.0
13	Languages	0.0	0.0	0.0	0.0	2.0	0.0	0.0
14	Music	0.0	0.0	0.0	0.0	2.0	0.0	0.0
15	Philosophy	0.0	1.0	0.0	0.0	1.0	0.0	0.0
16	Physics	0.0	1.0	0.0	0.0	0.0	0.0	0.0
17	Political	0.0	1.0	0.0	0.0	2.0	0.0	0.0
18	Psychology	0.0	0.0	0.0	0.0	2.0	0.0	0.0
19	Sociology	0.0	0.0	0.0	0.0	1.0	0.0	0.0
20	Theatre	0.0	0.0	0.0	0.0	2.0	0.0	0.0

Weka can also show some interesting statistics about the attributes. The class distribution over the values of each attribute (including the document name) is shown at the bottom of the Selected attribute area. With Visualize All we can see the class distribution in all attributes. If we change the class to document_name we can see the distribution of terms over documents as bar diagrams. The screenshot below shows some of these diagrams.



Examine the diagrams (the color indicates the document) and find the most specific terms for each document. For example, compare the diagrams of "anthropology" and "chair" and explain the difference. Which one is more representative and for which document?

Similarly we can create the boolean and TFIDF representation of the document collection. Examples of these representations are provided in the files Departments-binary.arff and Departments-TFIDF.arff, available from the data repository at http://www.cs.ccsu.edu/~markov/dmwdata.zip, folder "Weka data". Read the comment in the beginning of each file that explains the steps taken to create it.

To obtain the boolean representation apply the NumericToBinary filter to the term count representation. See what changes in the diagrams. For the TFIDF representation, use the original string representation and apply the StringToWordVector filter with IDFTransform=true. Examine the document-term table and the bar diagrams and explain why some columns (e.g. "chair" and "website") are filled with only zeros.

The **deliverable** for the data preparation step is the three data files – Boolean, Term count, and TFIDF, as well as some statistics (tables and diagrams) and analysis of the attributes and class distributions (see the questions above).

Bayesian Classification and Reasoning

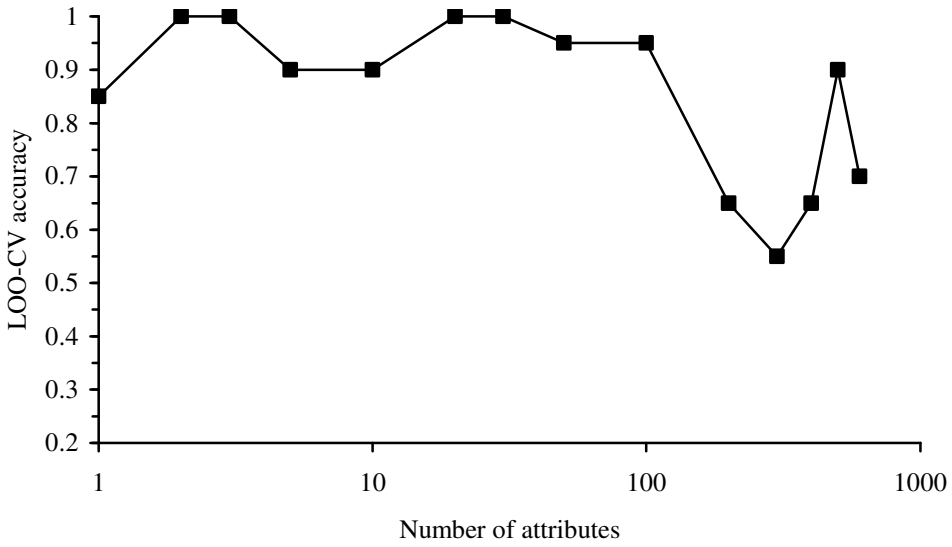
At this step we use Naïve Bayes and Bayesian networks for text document classification and further look into the underlying mechanisms of Bayesian reasoning. However before applying these algorithms we have to reduce the number of attributes in our data sets through feature selection. This will have two positive effects to our experiments. Firstly, it will improve the classification accuracy and secondly, it will simplify and make possible the visual inspection and evaluation of the Bayesian network model, which otherwise would involve thousands of variables.

Feature selection

In the area of text document classification the number of attributes is usually much higher than the number of documents. This poses a problem for many learning algorithms, further aggravated by the presence of too many 0's in the document-term matrix. An obvious solution to this problem is to select a subset of words (terms) that best represent the document collection with respect to the classification task. This process is generally called **feature (attribute) selection** and is aimed at removing the irrelevant to the classification task attributes. There is an interesting experiment that we discuss hereafter. It investigates the relationship between the number of attributes, their relevance and the accuracy of classification.

Attributes may be ranked by relevance using various algorithms. Weka provides a good number of algorithms for this purpose available through the Attribute Selection filter (the same algorithms can be used in Select attribute mode). We are not going into the details of these algorithms, but interested students may read Section "Feature Selection" in Chapter 5 of book [2]. To illustrate this process let's apply InfoGainAttributeEval to our Boolean data set, Departments-binary.arff. Weka now shows the attributes in a different order, starting with document_name, research, science etc. Note that document_name appears to be the most relevant attribute, which is obvious as it distinguishes uniquely every single document. However this attribute cannot be used to classify new documents as they will have different names (or IDs). That is why in most cases when it comes to generating a classification model we remove the ID attribute. Now, after doing so let us look at the document-term table (press on Edit button in Preprocess). There is an obvious difference with the same table showing the attributes in alphabetical order – the number of 0's in the leftmost columns is far less. It's interesting to look at the rightmost columns too – there are columns with all 1's (e.g. chair, phone, website) and such with almost all 0's. These are obviously irrelevant attributes (explain why).

The next step is to run the Naïve Bayes algorithm on the current data (with ordered attributes). We will repeat this with different number of attributes selected from the beginning of the list. Let's try 1, 2, 3, 5, 10, 20, 30, 50, 100, 200, 300, 400, 500, 600 and then record the classification accuracy with 10-fold cross validation (the default test option). Note that we don't count the class attribute, which is actually used in the process of attribute ranking. The results from this experiment are shown in the graph below (the markers on the curves indicate the data points 1, 2, 3, 5 etc.).



With one attribute (research) the algorithm performs relatively well (85% accuracy) and with 2 and 3 it achieves 100% accuracy. Adding more attributes generally decreases the accuracy as they become less relevant. We also see some fluctuations especially at the right end of the curve, which may be a result of inconsistency between the attribute selection method and the attribute relevance with respect to the Naïve Bayes classification method. The effect of adding irrelevant attribute can be explained generally with the fact that an **important assumption** for using the Naïve Bayes algorithm is violated. It is that all attributes should have the **same relevance** with respect to the classification task. The reason for making this assumption can be found in the way Naïve Bayes computes the class likelihoods – they are simply products of the conditional probabilities of all attribute values given the class value.

The results of the above series of experiment show some optimal intervals for the number of attributes needed to achieve maximum accuracy. These are [2, 3] and [20, 30]. Generally these results depend on two factors – the choice of data representation and the attribute selection method used. As a **deliverable** for this step we suggest that students create similar graphs with the other two data sets – Term counts and TFIDF, and also use different attribute selection algorithms to see how these two factors affects the performance of Naïve Bayes and consequently determine the optimal number (and the actual subset) of attributes. The Subset evaluator may also be used to see how Weka will find the optimal subset of attributes with different data sets.

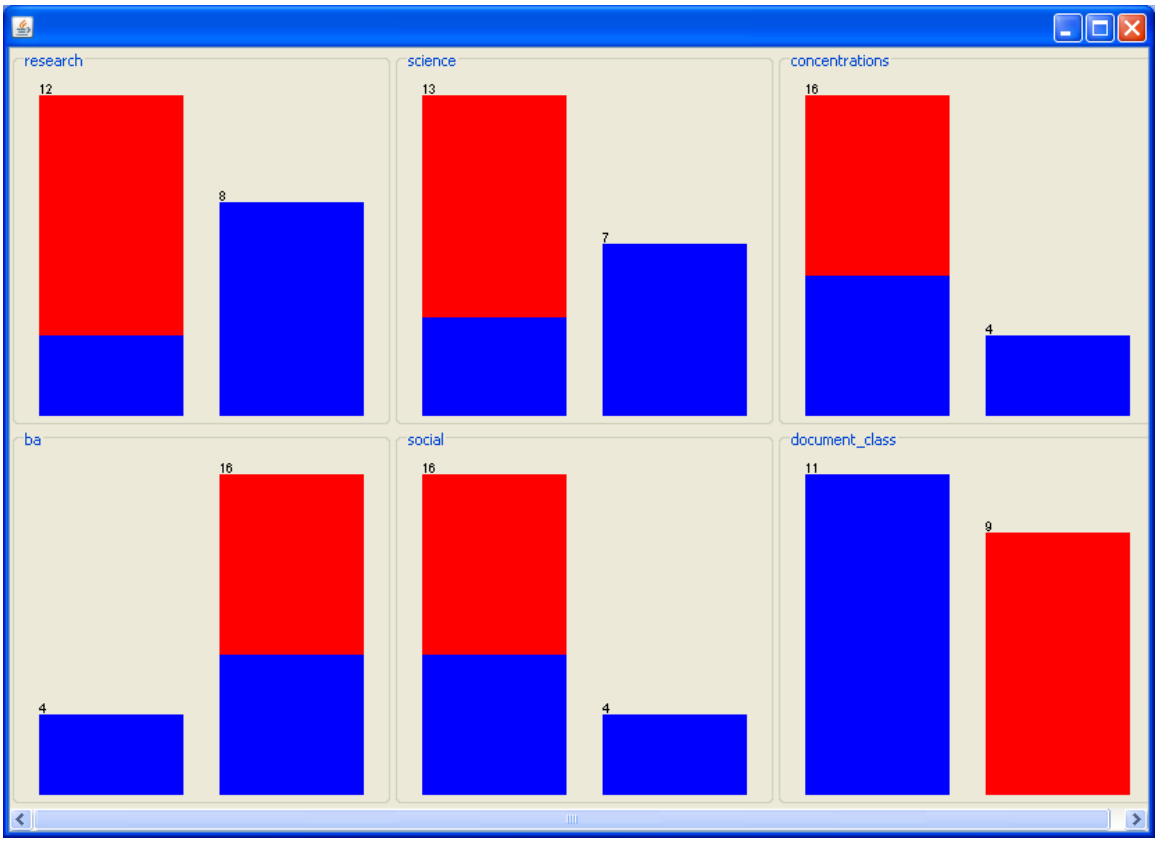
Bayesian learning and classification

At this step we use the data set with 5 attributes with which Naïve Bayes achieved 90% accuracy in the experiments described in the previous section. First we look closely into the classification model built by Naïve Bayes and then investigate how the Bayesian network extends it and thus improves the classification accuracy.

Running Naïve Bayes with 10-fold cross validation produces the following output (some parts of it are skipped for brevity):

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Instances:    20
Attributes:   6
              research
              science
              concentrations
              ba
              social
              document_class
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

Class A: Prior probability = 0.55

research:  Discrete Estimator. Counts =  4 9  (Total = 13)
science:  Discrete Estimator. Counts =  5 8  (Total = 13)
concentrations:  Discrete Estimator. Counts =  8 5  (Total = 13)
ba:  Discrete Estimator. Counts =  5 8  (Total = 13)
social:  Discrete Estimator. Counts =  8 5  (Total = 13)

Class B: Prior probability = 0.45

research:  Discrete Estimator. Counts =  10 1  (Total = 11)
science:  Discrete Estimator. Counts =  10 1  (Total = 11)
concentrations:  Discrete Estimator. Counts =  10 1  (Total = 11)
ba:  Discrete Estimator. Counts =  1 10  (Total = 11)
social:  Discrete Estimator. Counts =  10 1  (Total = 11)

=== Stratified cross-validation ===

Correctly Classified Instances          18               90      %

=== Confusion Matrix ===

 a b   <-- classified as
 9 2 | a = A
 0 9 | b = B
```

Looking at this output two important observations can be made. First, all attributes have only one of its values occurring in class B. This is indicated by the fact that one of the counts is always 1, which means that the actual count is 0 (according to the Laplace estimator used by the algorithm the actual value count is incremented by 1). Second, the confusion matrix indicates that all documents from actual class B are always classified as B, while two documents from class A are wrongly classified as B. Obviously the second observation is a result of the first one (explain why). A look at the class distribution over the attribute values produced by the Visualize All option in Preprocess mode (see below) confirms this conclusion.



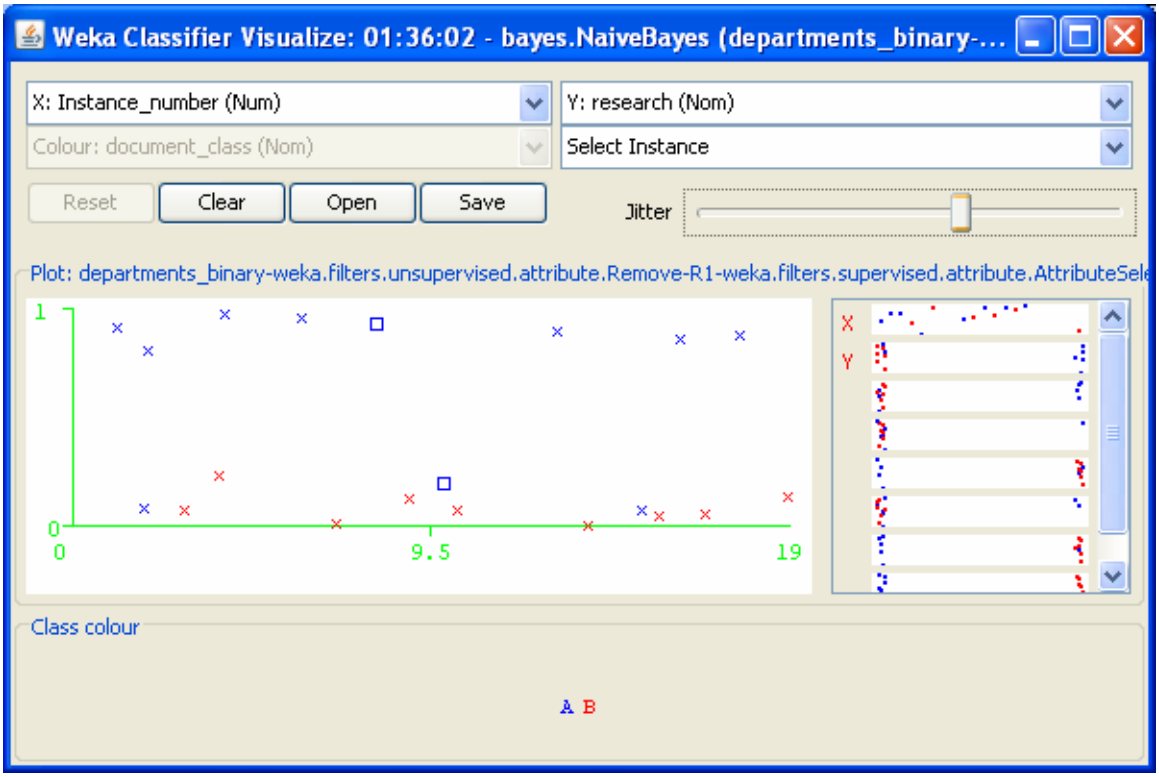
For example, all documents with research=1 (the right bar) fall in class A (blue color), which means that no documents with research=1 fall in class B. Consequently the conditional probability P(research=1|B) = 0. In fact, the Laplace estimator is used to avoid zero probabilities like this, because they make the whole product zero and thus ignore the contribution of other attributes. At the same time however the conditional probability P(research=0|B) will be almost 1 (not exactly 1 because of the Laplace correction). Thus a document with research=0, science=0, concentration=0, ba=1, and social=0 will be clearly classified in class B, because the product P(research=0|B)P(science=0|B)P(concentration=0|B)P(ba=1|B)P(social=0|B) will be much higher that the corresponding product for class A. This can be seen in the Weka output if we look at the probability distribution in the predictions (after checking "Output predictions" in "More options…"):

```
=== Predictions on test data ===

 inst#,     actual, predicted, error, probability distribution
     1        1:A        1:A          *0.981  0.019
     2        1:A        1:A          *0.66   0.34
     1        1:A        1:A          *0.809  0.191
     2        2:B        2:B           0.05  *0.95
     1        1:A        1:A          *0.809  0.191
     2        2:B        2:B           0.05  *0.95
     1        1:A        1:A          *0.985  0.015
     2        2:B        2:B           0.061 *0.939
```

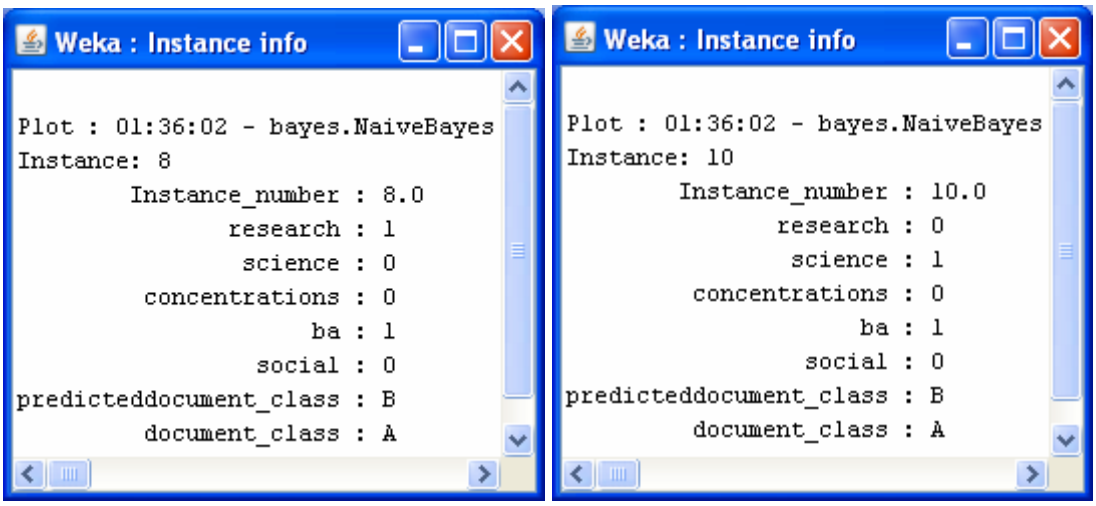
```
1        1:A        2:B      +    0.451 *0.549
2        2:B        2:B           0.044 *0.956
1        1:A        2:B      +    0.35  *0.65
2        2:B        2:B           0.041 *0.959
1        1:A        1:A          *0.997  0.003
2        2:B        2:B           0.069 *0.931
1        1:A        1:A          *0.735  0.265
2        2:B        2:B           0.047 *0.953
1        1:A        1:A          *0.997  0.003
2        2:B        2:B           0.069 *0.931
1        1:A        1:A          *0.956  0.044
2        2:B        2:B           0.056 *0.944
```

For all documents from actual class B, the class distribution decisively predicts class B.

The predictions also show that the two errors (marked with +) happen in actual class A. If we want to know exactly which two documents are wrongly classified we may look at the visualization of the classifier errors (right click in the result list).



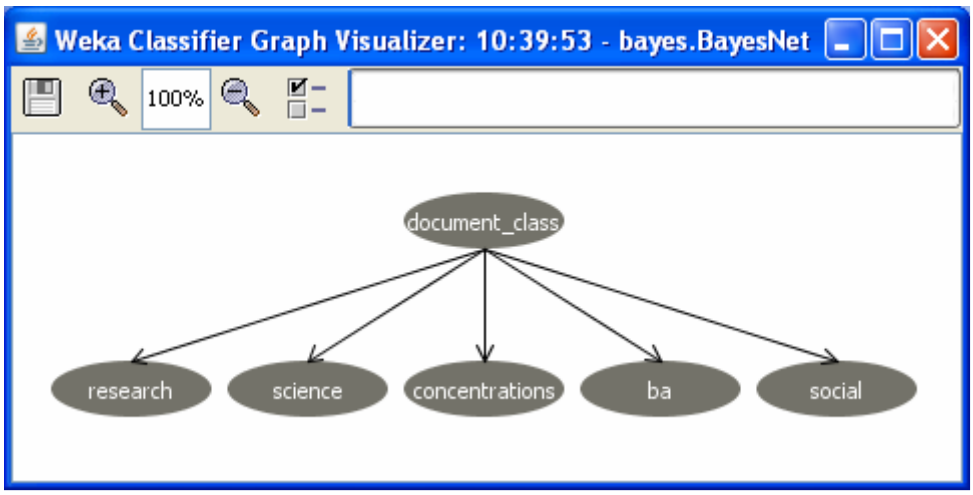
Clicking on the two squares in the plot reveals these two documents:



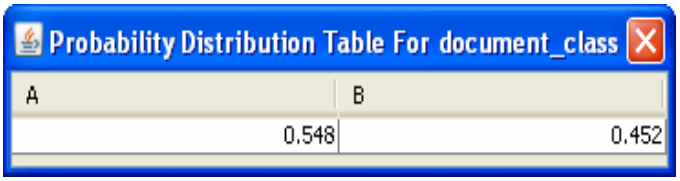
Now, let us apply the Bayesian network classifier to this data set. It's available in group Bayes under the name BayesNet. Running it with the default parameter setting and 10-fold cross validation produces 95% accuracy. Now let us look at the model available in text from the classifier output window.

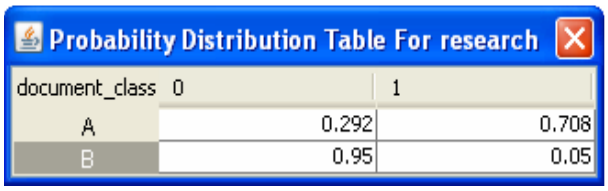
```
Network structure (nodes followed by parents)
research(2): document_class
science(2): document_class
concentrations(2): document_class
ba(2): document_class
social(2): document_class
document_class(2):
```

The graphical representation of this model can be seen too through Visualize graph (right click in the result list).



Note that this structure of the network makes this model equivalent to the Naïve Bayes classifier. That is, we have the class variable at the root with only prior probabilities and conditional probability tables at the leaves as shown below.

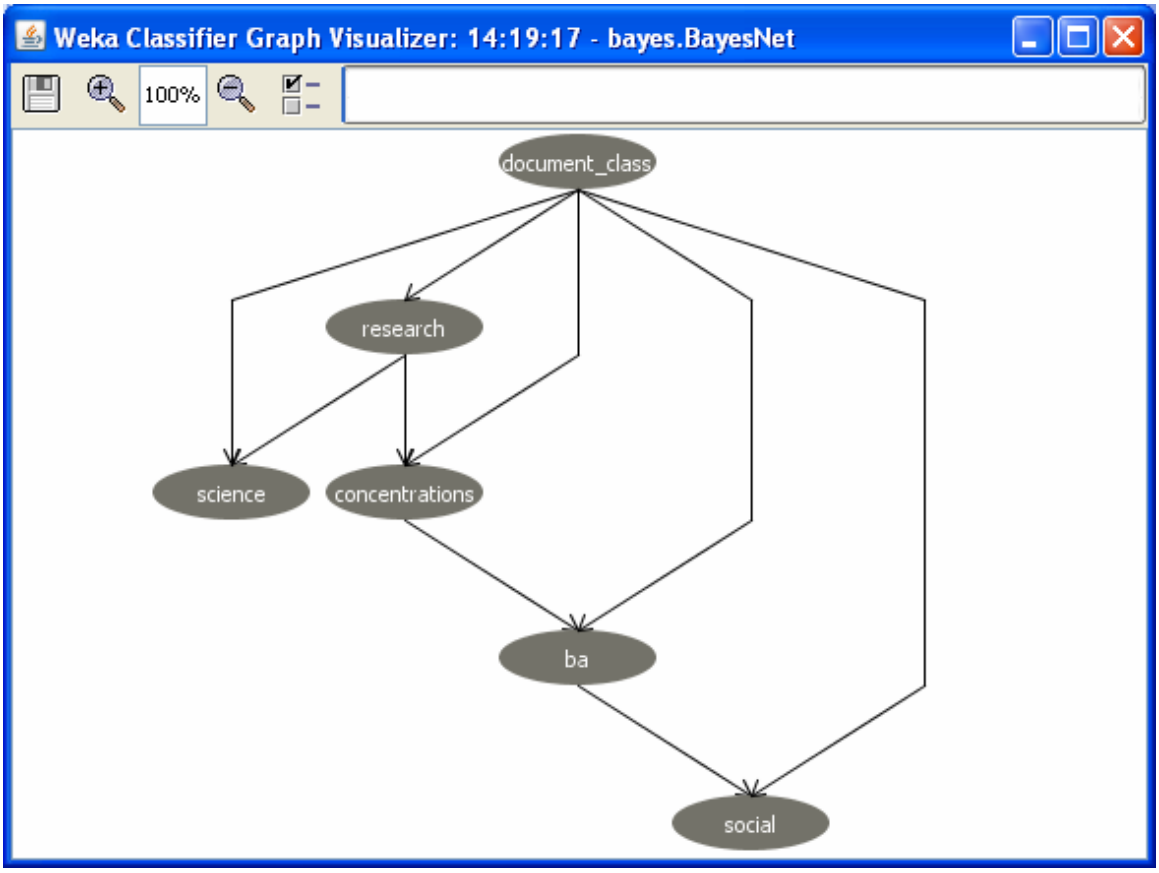




So, why does this model achieve a slightly higher accuracy (95%)? The answer is in the way probabilities are estimated. The default setting (SimpleEstimator -- -A 0.5) defines an initial value for the frequency count as 0.5 (see the explanation provided by the More button). This is different from Laplace used by Naïve Bayes, where the initial count is set to 1. This estimator is also applied to the prior probability calculation of the document class (see the difference from the Naïve Bayes output, where P(A)=0.55 and P(B)=0.45). When we set alpha=1 and run BayesNet again – the accuracy is back 90%. In this case BayesNet works exactly the same as NaïveBayes.

Other estimators may be also used. For example, with the BMAEstimator the accuracy drops to 85% (see the structure of the error shown by the confusion matrix) and with MultiNomialBMAEstimator it is even worse (55%). The later estimator actually assumes a different distribution and thus makes this algorithm equivalent to NaiveBayesMultinomial (NaiveBayes uses Normal or Discrete distribution).

Let us now investigate how changes in the network structure may affect the classification results. With the default setting BayesNet creates a simple network structure equivalent to the one used by Naïve Bayes (all attributes are leaves and the class is their single parent). We may change this by dropping the restriction that nodes should have no more than one parent (maxNrOfParents=1). So, let us use maxNrOfParents=2. This will allow the BayesNet algorithm to create a more general network that may include dependency arcs between non-class variables. This is a conceptual difference between the Naïve Bayes and the Bayesian Network approaches. In fact, this violates the basic assumption used by Naïve Bayes that the attributes are **statistically independent** (this is the reason to call the algorithm "naïve" as this assumption is usually not present in real data). Running BayesNet with the new setting (still with the Laplace estimator, alpha=1) produces a slightly better accuracy of 95%. Let us take a look at the network structure obtained with this setting.



We see now dependencies among the "independent" attributes. As a result some of these attributes have two parents and their conditional probability tables have four rows listing all possible combinations of the values of the parent variables. For example, the table for science is the following:

Probability Distribution Table For science

document_class	research	0	1
A	0	0.2	0.8
A	1	0.5	0.5
B	0	0.909	0.091
B	1	0.5	0.5

The dependency arcs are created by using a Bayesian Network learning algorithm while classification is based on a simple Bayesian network inference, which computes the probability for each class value given the distribution represented by the Bayesian network (all variables except the class one are known). Details about this algorithm as well as a complete documentation for its use through Weka are available from [5]. We recommend that students read this article before or while running the experiments described in this section.

The network structure learnt from data shows that the attributes are in fact dependent. The fact that we got an improvement in accuracy (given the same probability estimation method) means that the Baesian Network inference makes use of this dependency and thus outperforms Naïve Bayes on the classification task. The latter cannot handle dependent attributes and therefore needs the attribute independence assumption.

The last step of using Weka in this experiment is for saving the Bayesian network models on files to that they can be used by another Bayesian Network tool.. For this purpose Weka supports the so called BIF (Bayesian Interchange Format) file format, a XML-based format used to represent Bayesian Networks (see http://www.cs.cmu.edu/afs/cs/user/fgcozman/www/Research/InterchangeFormat/**).**

Once a Bayesian model has been created it can be saved as a XML BIF file by using the Save Graph button in Weka's Graph Visualizer. As a **deliverable** for this step we suggest that students create two files – one with the Naïve Bayes net (the simple structure with the class node as a parent of all attribute nodes, created with maxNrOfParents=1) and the other one with the more general Bayesian net with dependency arcs between independent attributes (created with maxNrOfParents=2). In addition the most general network can be created without using Niave Bayes as an initial network structure (initAsNaivebayes=false) and by allowing more than 2 parent nodes (say, maxNrOfParents=100).

Bayesian Network reasoning

In Weka we can learn Bayesian networks and use them for classification. At the classification step we are given an example (e.g. a document) with unknown class value. That is, we know the values of all variables (observations) except the class and the network inference mechanism computes the distribution at the class node (unobserved variable) thus determining the predicted class value of this example. It's important to note that Bayesian networks also provide other types of inference which may be useful for classification as well as for other types of reasoning. To illustrate this we shall examine our networks created with Weka by using a more general Baysian Network tool. There are many such tools (see for example, a directory page at http://www.cs.ubc.ca/~murphyk/Bayes/bnsoft.html), but for the purposes of this project we recommend using **JavaBayes**. It's a Bayesian Network package written in Java, available from http://www.cs.cmu.edu/~javabayes/. The reason we are using it is that it also supports the XML BIF format that we used to save our networks created with Weka.

At this point students have to install JavaBayes and familiarize themselves with its use by running some simple examples (provided with the package). Then the files created with Weka have to be loaded in JavaBayes. Note that the XML BIF files from Weka have first to be manually edited to fix a small inconsistency in the file formats supported by the two packages. As the value identifier in JavaBayes should begin with a letter we have to replace the 0 and 1 in the Boolean data with, say v0 and v1. This can be easily done if we save the file from Weka in CSV format, replace the values in a text editor (e.g. Notepad)

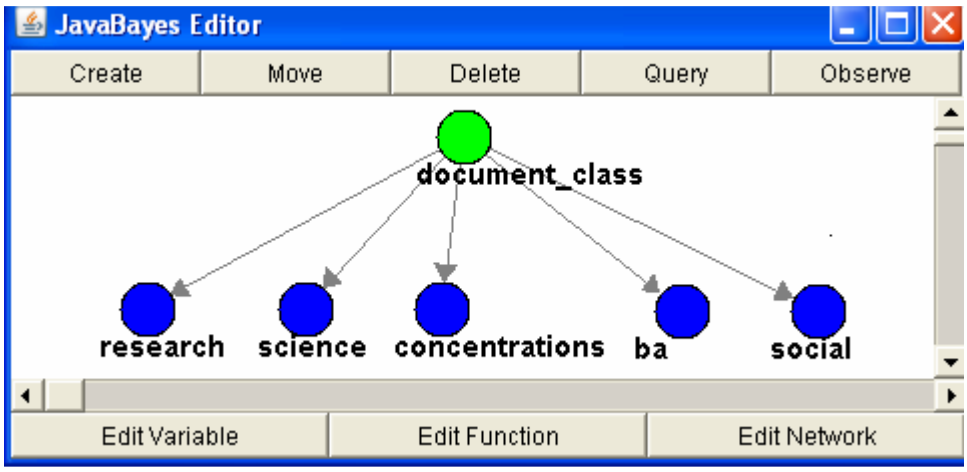
and load it back in Weka. Then run BayesNet and save the graph through the Graph Visualizer in XML BIF format.

After successful loading of the first BIF file created with Weka we see the Bayesian graph in the JavaBayes editor window. By selecting the Observe tab and clicking on the nodes we can set observed values for some variables, while with choosing Query we can see the probability distribution of the values of any variable in the console window. Without entering any observations these probabilities are the estimated ones obtained from the training data. If an observation is set then the probabilities change to 1 for the observed value or 0 for all others.

Several types of experiments can be performed with Bayesian Networks each illustrating a different type of reasoning.

Classification

Let us enter the observations for document number 8 (Economics) which was wrongly classified by Naïve Bayes with 10-fold cross validation. All leave nodes become blue and the console window shows the observations (evidence) as well the probability distribution of the class values.




```
probability (  "research" ) { //1 variable(s) and 2 values
      table
            1.0    // p(v1 | evidence )
            0.0;   // p(v0 | evidence );
}
Posterior distribution:
probability (  "science" ) { //1 variable(s) and 2 values
      table
            0.0    // p(v1 | evidence )
            1.0;   // p(v0 | evidence );
}
Posterior distribution:
probability (  "concentrations" ) { //1 variable(s) and 2 values
      table
            0.0    // p(v1 | evidence )
```

```
            1.0;  // p(v0 | evidence );
}
Posterior distribution:
probability (  "ba" ) { //1 variable(s) and 2 values
      table
            1.0   // p(v1 | evidence )
            0.0;  // p(v0 | evidence );
}
Posterior distribution:
probability (  "social" ) { //1 variable(s) and 2 values
      table
            0.0   // p(v1 | evidence )
            1.0;  // p(v0 | evidence );
}
Posterior distribution:
probability (  "document_class" ) { //1 variable(s) and 2 values
      table
            0.5452999095512584      // p(A | evidence )
            0.4547000904487416;     // p(B | evidence );
```

As P(A | evidence) > P(B | evidence) we can predict class A for this document, which is its actual class too. Note that when it misclassified this document Naïve Bayes did not use the whole training set, but only 9/10 of it or 18 instances. While the above probabilities are computed with the whole training set and thus more accurately reflect the underlying distributions in data.

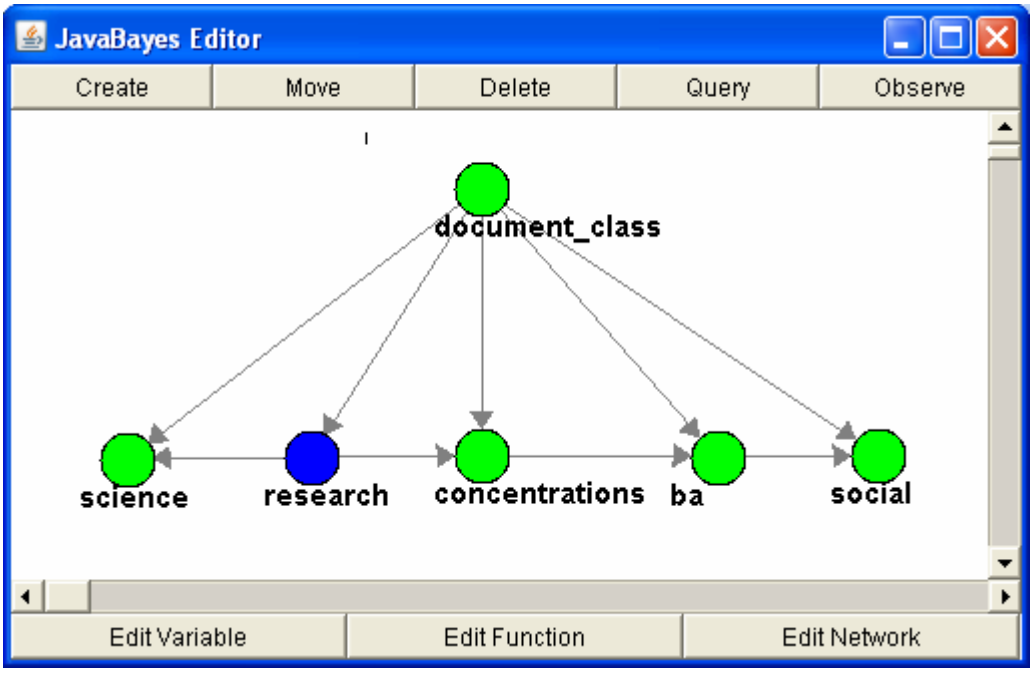
Using partial evidence

Another way of using the Bayesian network is by entering partial evidence. In this way we can investigate how the class probabilities are affected by using different attributes, and how much each attribute value is relevant to the classification of a particular instance. This can be done by setting the observation for one variable at a time and then checking the class probabilities. The following table shows the results of this experiment:

	No evidence	research=v1	science=v0	concentrations=v0	ba=v1	social=v0
P(A|evidence)	0.545	0.901	0.337	0.448	0.448	0.448
P(B|evidence)	0.455	0.099	0.663	0.552	0.552	0.552

It is clear that research is the strongest attribute to support class A, all others actually misclassify instance number 8 (Economics). It's interesting to relate these results to the bar diagrams for the attributes that we see in Weka's preprocess mode. This may suggest the explanation for these results.

Now let us run the same experiments on the network with dependency arcs between independent attributes (load the second BIF file created with Weka). For example, setting the observation for research=v1 produces the same class distribution (0.901/0.099).




```
probability (  "research" ) { //1 variable(s) and 2 values
      table
            1.0    // p(v1 | evidence )
            0.0;   // p(v0 | evidence );
}
Posterior distribution:
probability (  "document_class" ) { //1 variable(s) and 2 values
      table
            0.9013657056145675      // p(A | evidence )
            0.09863429438543249;    // p(B | evidence );
```

However, the other single attribute observations produce different class distributions. It's interesting to find the connection between these changes and the particular structure of the dependency arcs.

Investigating the role of dependency arcs between "independent" variables

Experiments can be done with pairs of variables and then results compared with the Bayesian network with dependency arcs between independent attributes. For example, when research and science are independent (Naïve Bayes) we have

```
p(A | research=v1, science=v0) = 0.795
p(B | research=v1, science=v0) = 0.205
```

While if there is a dependency arc from research to science we have

```
p(A | research=v1, science=v0) = 0.901
p(B | research=v1, science=v0) = 0.099
```

Comparing the two class distributions shows how the dependency between research and science affects the classification. Obviously adding a dependency arc increases the likelihood of class A for this particular evidence.

Relationships between non-class variables

Another type of experiment is to look at the relationships between non-class attributes. Let us investigate the relationship between research and science. Setting research=v1 produces

```
P(science=v1|evidence)= 0.5
P(science=v0|evidence)= 0.5
```

While research=v0 produces

```
P(science=v1|evidence)= 0.296
P(science=v0|evidence)= 0.704
```

In the original context of text documents these numbers can be read in the following way. If a document contains "research" it's likely that it also contains "science". However it does not contain "research" we cannot say anything about "science". This sounds quite feasible for our domain of sciences and humanities.

Further experiments

We suggest the following further experiments:

- Investigate the improvement in accuracy when dependency arcs are added (Bayesian networks) compared to Naïve Bayes.
- Investigate how adding irrelevant attributes changes the classification accuracy of the Bayesian network model.
- Investigate the difference in the effect of adding observations to independent and dependent variables with respect to the class distribution. Does the direction of the dependency make a difference?
- Create more general Bayesian networks by abandoning the restriction to start the search for the network structure from the Naive Bayes network (set initAsNaivebayes=false) and by allowing more than 2 parent nodes (use maxNrOfParents>2). Compare the classification accuracy of these more general networks with that of the simpler ones.
- Repeat all experiments with Naïve Bayes and Bayesian networks using the term count and TFIDF data sets. Note that for the term count representation the multinomial distribution has to be used. Also, when creates the Bayesian network with numeric data (TFIDF) Weka first applies attribute discretization.

The **deliverable** for this phase is the results of all experiments (with the provided data as well as with new data collected and preprocessed by the students) and comment of these results including the answers to the questions and suggested explanations.

Prerequisites and requirements

Students should have basic knowledge of algebra, discrete mathematics and statistics. Another prerequisite is the data structures course. While not necessary, experience with programming in Java would be helpful as the project uses Java-based packages. These packages are open source and students may want to use specific parts of their code to implement stand-alone applications for Bayesian learning or reasoning.

The project is customizable and can accommodate different teaching approaches and different implementations depending on the choice of particular problems to be solved and tools to be used. The data collection step can be implemented manually or by using some software tools. The learning and reasoning steps use implementations of Naïve Bayes and Bayesian Networks algorithms available from free open source software packages. This allows the project to be extended to building stand-alone applications depending on the particular teaching goals and student experience in programming.

The software packages and data sets used in the project are freely available on the Web:

1. Weka 3 – Free open source Machine Learning and Data Mining software in Java available from http://www.cs.waikato.ac.nz/~ml/weka/index.html.
2. JavaBayes – Bayesian Networks in Java, available from http://www.cs.cmu.edu/~javabayes/
3. Data sets for document classification accompanying the book *Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage* ([2]) available from http://www.cs.ccsu.edu/~markov/dmwdata.zip.

It is recommended that before starting the project students read Chapters 13 and 14 of Russell and Norvig's book ([1]), Chapter 1, Chapter 3 (Section "Probability-based clustering") and Chapter 5 (Section "Naïve Bayes Algorithm") of Markov and Larose's book ([2]), or Chapter 6 of Mitchell's book ([3]).

While working on the project students can use Witten and Frank's book [4] and Bouckaert's documentation [5] to get additional information on how to use the algorithms for Naïve Bayes classification and Bayesian Reasoning.

References

1. Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach, Second Edition, Prentice Hall, 2003. Chapters 13, 14.
2. Zdravko Markov and Daniel T. Larose. *Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage*, Wiley, 2007. Chapter 1 is available for free download from Wiley.
3. Tom Mitchell. Machine Learning, McGraw Hill, 1997, Chapter 6.
4. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques* (Second Edition), Morgan Kaufmann, 2005.

5. Remco R. Bouckaert, Bayesian Network Classifiers in Weka, online at http://www.cs.waikato.ac.nz/~remco/weka_bn/index.html, or download a PDF file at http://www.cs.waikato.ac.nz/~remco/weka.bn.pdf