Knowledge based systems
Analysing the eligibility of a person for higher education using a Bayesian network

Lisa Mischer & Frank Steiler
Interactive and knowledge based systems (T2INF4307)
DHBW Stuttgart
Contact: it12147@lehre.dhbw-stuttgart.de

`http://www.github.com/steilerDev/WhatToStudy`

January 12, 2015

Contents

Chapter 1

Introduction

Within knowledge based systems, logic is a commonly used way to represent connections between data and expressions. Unfortunately logic can not handle uncertainty or imprecise data. Bayesian Networks have been developed to tackle this problem, by representing knowledge as a set of variables and their dependencies within a directed acyclic graph. [Rei14]

A probabilistic network is using conditional probabilities between the nodes of the graph and inferences to calculate the probability of symptoms and/or causes. There are three types of inferences that are occurring in the network and enable the functionality of the graph: diagnostic, causal and inter-causal inference.

A Bayesian Network is defined by a set of edges (nodes) connected through vertices within a graph: $D = (V, E)$. Every node has finite set of mutually exclusive states. On top of that the network is quantifying the dependencies within a separated conditional probability table (CPT) for each node. [Vom05]

Concluding to create and then use a Bayesian Network, a user has to create the correct graph first and then determine all values for the CPT. A correct network can either be created by an expert, or by using data mining techniques to find connections between entities and machine learning to specify the CPT values.

By adding observations for a specific case to the network, it is updating beliefs about other variables. Furthermore the probability of a certain event or state can be predicted by observing other events or states. Therefore it can support decision making and has numerous applications, like the diag-

nosis of diseases, automatic troubleshooting or education testing. [Vom05]

Chapter 2

Preparation of data

As part of our exercise we received a set of data that was supposed to help us derive the structure of the network. To be able to use the stated cases we first had to analyse and harmonise the provided data.

These preparation included the transformation of continuous variables into discrete ones, to enable their usage within the network. The variables we had to adjust were all provided grades, the test results of the online tests and study ability test, as well as the parental income.

Furthermore we chose a general syntax for the naming of values, to have standardised identifiers and consistent ranges. E.g. we defined a not available value as "NA", where the source used different words in different entities, like "keine", "n.a." and many more. We did not need to do this necessarily, but it simplified the work with the network.

A detailed description about the conversation of these values can be found within the documentation of the implementation in chapter 5 on page 9, more specific in section 5.3 on page 14.

These preparations were necessary to train the Bayesian network, as well as to analyse a certain case. Only cases which are prepared in the same way can be evaluated with our result network, but the implementation is able to read a cleaned

Chapter 3

Structure of the network

To find the best structure of the network, we took a look at the provided information, trying to find connections between columns of the data set.

Therefore we plotted the data within a parallel coordinate system, shown in appendix B on page 19. We used this representation to derive connections by reducing the data on a single axis to a smaller range and hoping to observe a similar behaviour on another axis. By revealing such a influence we concluded a direct connection between the entities.

On top of that we thought about logical connections, e.g. the mathematics', German's and physic's grade had to influence the qualification average. Furthermore, we took it for granted that the mathematics grade influences the results of the math online test, as well as the German grade influences the results of the German online test. Moreover the mathematics results influence the physics grade, since a basic understanding of mathematics is essential for physics.

To determine whether or not our Bayesian network is suited to predict the final grade of a specific student, we tested it using the provided data set. This test tried to predict the final grade of a person, using all available information from the data excluding the actual course and the actual final grade. By comparing the predicted and the actual result, we received an error rate for our network. We used this error rate to compare different versions of the network.

The generation of the final network was an iterative process, changing the arcs, CPTs and/or entities within each step. After changing the structure, we determined the new error rate and compared it to the previous one.

By using this process we were able to improve the overall performance of the network. In the end, all our considerations at the beginning proved as wrong, since there is no indirect influence on a variable, only direct ones.

We were able to achieved an error rate of 11%, predicting the course taken, and 4%, predicting the final grade of a student. Appendix A on page 18 is showing our final Bayesian network. The implementation documented in chapter 5 on page 9 is using a slightly different Network, since the optimal one is too big, resulting in a out of memory error of the Java virtual machine. In the adjusted net the gender of a person is no longer considered, to reduce its size. Nevertheless, the error rate only rises slightly to 14%, respectively 6%. This network is then used to recommend for or against studying a specific course.

The tool we used to specify our Bayesian network is called Netica, unfortunately we only had access to a limited version of this program. Concluding, it was necessary to limit our data set to 15 entities. Therefore we had to leave out at least one piece of information and take a slightly more imprecise result into account. We chose to ignore the information about the state, even though it improved the error rate significantly, when taking only 5 of 16 states into account. Unfortunately it is necessary to accept all 16 available states, since all of them could be chosen. This resulted in an entity with 16 different parameter values, which were too much to handle for Netica. Since it was not reasonable to allow only 5 parameter values, we decided to leave this entity out of consideration. Furthermore we left out the income of the parents and the nationality, since these did not provide any gain in information to the network.

Chapter 4

Learning conditional probability tables

After creating a draft of the network layout, containing all plausible node connections, we had to quantify the dependencies between nodes within the conditional probability tables (CPT). Since we were not able to consult an expert about this problem we chose to use machine learning algorithms to generate the tables.

Fortunately the tool we used to specify the network offered a set of machine learning algorithms to generate the CPTs from a data set. These include a 'counting algorithm', an 'expectation-maximization (EM) algorithm' and a 'gradient descent algorithm'. According to the documentation of the tool the 'counting algorithm' is the simplest and fastest. This algorithm is especially well performing when having no uncertainty or missing data. Concluding if the data set has missing values, the user should either choose the 'EM-algorithm' or the 'gradient descent algorithm'. The performance of either is highly depending on the data set and network structure. Therefore the selection of one algorithm over the other needs to be done by directly comparing their performance. [Cor10, p. 47]

After evaluating the performance of each algorithm using our set of data, we chose to use the 'expectation-maximization (EM) algorithm', since it fitted our needs best. 'Briefly, Expectation-Maximization learning repeatedly takes a Bayes net and uses it to find a better one by doing an expectation step followed by a maximization step. In the expectation step, it uses regular Bayes net inference with the existing Bayes net to compute the ex-

7

pected value of all the missing data, and then the maximization step finds the maximum likelihood Bayes net given the now extended data.' [Cor10, p. 48]

After applying the algorithm with several hundert iterations using the provided data set we could create a network that reliably predicted the final grade and therefore could give a recommendation based on all provided values. As mentioned earlier the error rate at predicting the final grade correctly is 4%. Testing the quality of the network based on the correct recommendation of studying the selected course and using the worse network mentioned in chapter 4 on the previous page, achieves an error rate of 2%, where no error is false positive (recommending the course even if the student is probably not going to succeed). This testing is done within the application and its precise documentation can be found in section 5.2.5 on page 13.

Chapter 5

Implementation - WhatToStudy

As part of the project a program had to be implemented that is using a Bayesian network and user specific input to give a recommendation for or against studying a specific course. This program was implemented using Java 8 and the NeticaJ library.

Within this chapter the general functionality and usage of the program is going to be described. For a in-depth documentation of the complete application see the JavaDoc webpage, bundled with this document, or the source code of the application, which is licensed using a GNU General Public License version 2. The complete project and all used assets are hosted on Github (`http://www.github.com/steilerDev/WhatToStudy`).

5.1 General usage

The easiest way to start the application is to use the provided `./run.sh` script. Make sure, that the stated location of the NeticaJ binaries within the script is correct. The NeticaJ binary files can be retrieved from the following URL: `https://www.norsys.com/netica-j.html`.

This project is packed and optimised to run on Macintosh OSX of any version, but since Java is allowing multi platform execution, the start script only needs to be adjusted to your operating system and the correct version of the NeticaJ library needs to be retrieved.

When starting the application without any additional command line ar-

guments you are entering the interactive mode described in section 5.2.1. By adding additional arguments to the program call (or the script execution) you can switch to other modes. A full list of all available modes is given in section 5.2, as well as printed to screen when entering the help mode through passing the `-h` argument (See section 5.2.6 on page 14).

5.2 Operation modes

Within this section every available functionality is listed and its functional principles are described.

5.2.1 Interactive mode

When starting the application without any arguments, the user is entering the interactive mode. This mode is requesting personal information one after another, providing a list of accepted answers. The application is trying to match the user given input with the list of allowed ones until it is accepted.

If the user does not want to give a specific answer, he can always pass an empty String (by hitting the return key without any input). This will most likely lead to a more imprecise prediction than it would be with a complete set of data.

This mode's main task is to collect and parse the user data. After the application gathered the data, it is actually starting to instantiate the evaluation mode, passing the created set of data. The evaluation mode's functionality is described in section 5.2.2.

5.2.2 Evaluation mode

Evaluation mode can be entered by passing the `-e` command line argument together with a case file to the program. The exact specifications of the input file are given in section 5.3 on page 14. In contrast to the specification the file does not need to have a value for the final grade or course. The case file is read by the program and only the first case is evaluated, even if it is stating more.

If the student's course is stated within the file, the program is going to evaluate the recommendation rate using the formula shown in figure 1. This evaluation was chosen based on testing experience using the provided

$$beliefOf(FinalGrade.VeryGood) + \frac{2}{3} \times beliefOf(FinalGrade.Good)$$

Figure 1: The used recommendation-rate formula

$$beliefOf(FinalGrade.VeryGood) + beliefOf(FinalGrade.Good)$$

Figure 2: The conservative recommendation-rate formula

data and the test mode described in section 5.2.5 on page 13. By using this
formula the general error rate of a wrong prediction is at 2% and therefore
double as high as choosing the conservative formula shown in figure 2, but
its non-acceptable error rate is 0% and therefore performs better than the
conservative formula.

A non-acceptable error is a false negative error, saying the program is
recommending the user to study the course, even if the user is, based on the
experience of the network, probably not going to succeed. Concluding an
acceptable error is a false positive one, where the program is discouraging
a student to attend the course, even if he would succeed. Since we really
did not want to give false negative prediction we were trying to reduce the
amount of this kind of error, even if we therefore have to take a slightly
higher overall error rate into account.

If the case within the file is not stating the course the student wants
to attend, the evaluation is checking the likeliness of a recommendation of
every available course by entering the appropriate state within the network
and then returning the course with the highest recommendation rate. That
means that the program is recommending the course, where the student is
most likely going to get a very good or good grade.

Optionally the user can specify a bayesian network, other than the one
packaged with the application to evaluate a case. To do so he can pass the
path to the network file as a third, optional argument. The file path needs
to not contain any white spaces, since those can't be handled by the Java
command line interface. On top of that, the nodes of the user specified file
need to match the ones of the internal network, but only the 'Course' and
'FinalGrade' nodes are mandatory

5.2.3 Draw mode

The draw mode can be entered by passing the `-d` argument to the application during startup. This mode is creating a graphical Java swing frame, which is showing the internally stored Bayesian network. The probabilities of every entity value is displayed as well.

Optionally the user can specify a bayesian network, other than the one packaged with the application to be drawn. To do so he can pass the path to the network file as a second, optional argument. The file path needs to not contain any white spaces, since those can't be handled by the Java command line interface.

5.2.4 Learning mode

A user can enter the learning mode by passing the `-l` argument and the path to a case file (which should not contain any white spaces). The file needs to meet the requirements stated within section 5.3 on page 14. This mode is using the provided data set to learn the values for the conditional probability tables (CPT) of the network.

Before starting the learning process, every stored value within the CPTs is removed, to avoid unsuitable inferences. The used learning algorithm is the 'expectation-maximization (EM) algorithm'. The reason of choosing this algorithm over the other available ones is given in chapter 4 on page 7.

After the learning process is finished, the application is writing a file called 'Learned.dne' into the current working directory, overwriting any equally named file. This file is containing all necessary specification for the network as well as the learned CPTs.

Optionally the user can specify a bayesian network, other than the one packaged with the application, whose CPTs should be learned. To do so he can pass the path to the network file as a third, optional argument. The file path needs to not contain any white spaces, since those can't be handled by the Java command line interface. On top of that, the nodes of the user specified file need to match the ones of the internal network, but only the 'Course' and 'FinalGrade' nodes are mandatory

5.2.5 Testing mode

To enter the testing mode the user needs to pass the `-t` argument together with the whitespace-free path to a case file. The cases in the file need to meet the stated requirements for the input files, described in section 5.3 on the following page.

The test is taking every case, including the information about the course, excluding the final grade, to evaluate the recommendation rate, giving either a recommendation for or against the choice. This recommendation is then compared with the real outcome of the case.

No error occurs, if the program is recommending the student to study the course and he actually achieved a good or very good grade, which would be a true positive prediction. Another correct prediction would be, if the program is discouraging the student to study the selected course and he actually failed or only achieved a satisfying grade, which would be a true negative prediction.

A false positive recommendation is given, if the program discourages the student to study the course, even if he actually achieved a very good or good grade. Concluding the program is giving a false negative prediction if he encourages the student to study the course even if actually failed or only achieved a satisfying grade. A false negative prediction, in contrast to a false positive one, is seen as a non-acceptable error, because we really do not want to encourage a person to waste his time by studying a course he is most likely going to fail. This kind of error should therefore been prevented by all means, even if this means that the overall amount of error is increasing.

At the end of the testing process the program prints out a table containing the result of the test, aggregate into the four categories described in the previous paragraphs. On top of that it provides the error rate (amount of errors over the amount of tests) and the non-acceptable error rate (amount of non-acceptable error over the amount of tests). In an optimal network both rates are as low as possible, but the network should at least be designed in a way to keep the non-acceptable error rate as small as possible.

Optionally the user can specify a bayesian network, other than the one packaged with the application, whose quality should be tested. To do so he can pass the path to the network file as a third, optional argument. The file path needs to not contain any white spaces, since those can't be handled by the Java command line interface. On top of that, the nodes of the user

specified file need to match the ones of the internal network, but only the
'Course' and 'FinalGrade' nodes are mandatory

5.2.6   Help mode

To enter the help mode the user needs to either pass the `-h` command line
argument or any non-valid set of arguments. This mode is printing out a
summary of the general usage of the program, as well as the specification of
an input file. This specification is also shown in section 5.3.

5.2.7   Version mode

The version mode is entered by passing the `-v` argument to the program.
This mode is printing out the version number of the application, as well as a
list of person involved at creating the application, further acknowledgements,
a link to the Github repository of the application and the licensing notice.

5.3   Case file handling

The reading of the case file is a very important part of the application used
by many functionalities. On one hand it has to meet the requirements stated
by our network and on the other hand it needs to be readable by the NeticaJ
library. On top of that it needs to be able to read the file format stated by
the exercise.

First of all the input file needs to have the same format as the file stated
in the exercise, saying it needs to be a CSV file separating the values through
semi-colons. The first line needs to be a header. The order of the columns
is given by the JavaDoc class description of `de.steilerdev.WhatToStudy.`
`Utility.CSVStreamer`. The allowed header values of each columns is de-
scribed in the JavaDoc of the `getValidHeaders()` function of the respective
enumeration.

The values allowed within the columns are also specified within the enu-
meration object of the respective column. They are described within the
JavaDoc of the `clean(String)` function, more precisely within the descrip-
tion of the String input variable.

The file handling is mostly done by the `de.steilerdev.WhatToStudy.`
`Utility.CSVStreamer` class, but the specification of a value is given by

the enumeration classes within the `de.steilerdev.WhatToStudy.Utility.`
`Case` package. Each line within the input files can be represented by an
object of the `de.steilerdev.WhatToStudy.Utility.Case.Case` class. For
more specific information about every functionality can be found in the
JavaDoc documentation of the mentioned classes and enumerations.

Bibliography

[Cor10]   Norsys Software Corp. *Netica-J Reference Manual*. Tech. rep.
           Norsys Software Corp., 2010.

[Rei14]   Prof. Dr. Dirk Reichardt. 'Wissensbasierte Systeme Kapitel 5 -
           Probalistische Netze'. Vorlesungsunterlagen Angewandte Infor-
           matik. Nov. 2014.

[Vom05]   Jiri Vomlel. *Some applications of Bayesian networks*. Presenta-
           tion. 2005.

Appendices

Physics
Satisfying	5.00
NA	23.0
Good	30.0
Very Good	40.0
Failed	2.00

Math
Very Good	44.0
NA	16.0
Satisfying	5.00
Good	34.0
Failed	1.00

German
Good	35.0
NA	16.0
Failed	0 +
Very Good	47.0
Satisfying	2.00

OLT_Math
Good	29.0
Satisfying	24.0
Failed	28.0
Very Good	19.0

OLT_German
Satisfying	46.0
Very Good	9.00
Failed	13.0
Good	32.0

Study_Ability_Test
Good	7.00
NA	81.0
Failed	0 +
Very Good	11.0
Satisfying	1.00

Qualification_Average
Very Good	36.0
Failed	0 +
Good	52.0
Satisfying	12.0

Sex
W	30.0
M	70.0

School_Type
A Gymnasium	61.0
T Gymnasium	13.0
W Gymnasium	8.00
Gesamtschule	2.00
NA	16.0

Qualification
Abitur	84.0
FH	14.0
Techniker	2.00

Course
C Science	19.9
E Engineering	20.3
Engineering	20.0
Economics	20.0
S Work	19.8

Final_Grade
Very Good	25.4
Satisfying	24.7
Failed	24.3
Good	25.7