CS360 Project 3

Part 1: Bayesian Networks

From `http://aispace.org/bayes/` download the AIspace `'Belief and Decision Networks'` Java applet. You might need to add `http://www.aispace.org/` to the `'Exception Site List'` available in the `'Security'` tab of the `'Java Control Panel'` to get the applet to run. Read the documentation available at the download site and try it out. Do this well before the deadline to ensure that it works on your computer.

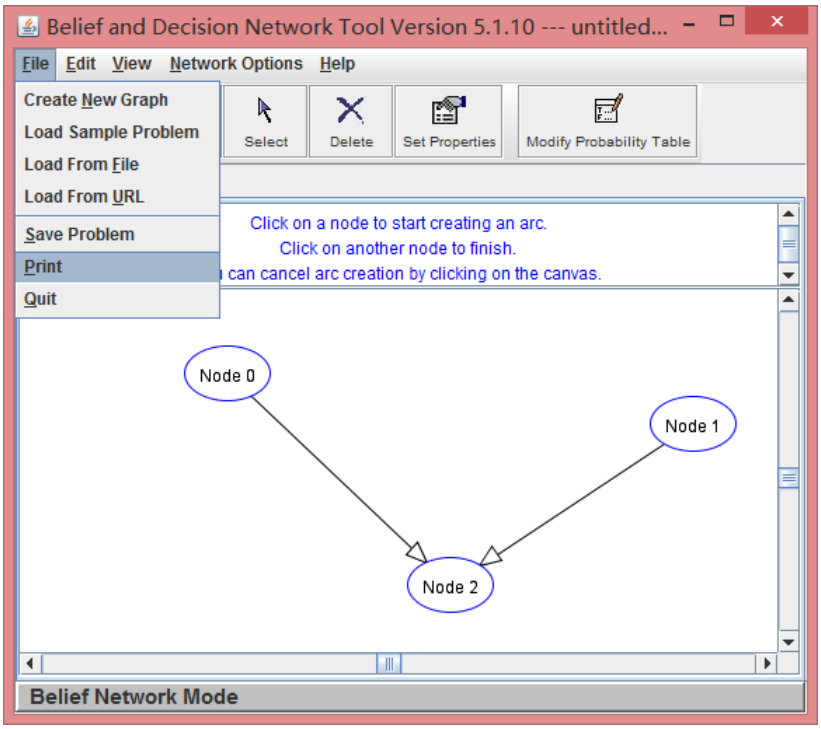

Figure 1: AIspace "Belief and Decision Networks" Java Applet

Problem 1

Using the Java applet, enter the Bayesian network depicted in Figure 14.23 from the textbook. Save your Bayesian network (using `'File'` → `'Save program'`) and include a screenshot of your network in your answer. Use the Java applet to calculate the probability that someone goes to jail given that they broke the law, have been indicted, and face a politically motivated prosecutor.

Problem 2

Suppose we want to add the variable P = PresidentialPardon to the network. Extend your network from the previous question with this new variable and briefly explain any links you add. Save the extended Bayesian Network and include a screenshot of it in your answer.

Problem 3

Using the Java applet, develop a Bayesian network that can be used to predict whether Donald Trump will be elected president. Random variables could include information about who will be the democratic candidate and anything else that you think might be relevant. We expect your Bayesian network to have 10 or more nodes (that is, random variables) and a non-trivial structure. For example, you do not want to have 9 nodes that are all directly connected to the 10th node or 10 nodes that all form a line. Save your Bayesian network and include a screenshot of it in your answer. Create two interesting nontrivial scenarios where the values of some nodes are known. Show the predictions of your Bayesian network for these scenarios and explain why they make sense.

Part 2: Naive Bayesian Learning

In this problem, you will implement a naive Bayesian classifier for classifying newsgroup messages. Given a message, your goal is to classify it as belonging to one of four different newsgroups `comp.graphics`, `sci.space`, `talk.politics.mideast`, or `rec.sport.baseball`.

Data

We provide a dataset which contains newsgroup messages which are labeled as either `comp.graphics`, `sci.space`, `talk.politics.mideast`, or `rec.sport.baseball`.

The directory contains two folders, one for the training set and the other one for the test set. The **training set** is a set of examples used for learning, namely to obtain all probabilities of the naive Bayesian classifier. The **test set** is a set of examples used only to assess the performance of the classifier which is learned from the training set.

The newsgroup messages are divided into separate folders in both the training and the test sets. For your convenience, we have created two files, `training_list` and `test_list` that, respectively, list the locations of all message files in the training and test sets. Each line in these files contains the location of a message file, followed by an integer that specifies the class the message file belongs to (0 = `comp.graphics`, 1 = `sci.space`, 2 = `talk.politics.mideast`, 3 = `rec.sport.baseball`).

Each message file contains a header, followed by the message itself. In this project, we are only interested in the message and disregard the header (which actually lists

the newsgroup that the message belongs to). We identify the beginning of a message by finding the first occurrence of an empty line.

We also provide you with a dictionary of words that can be used to distinguish messages in different newsgroups (`dictionary`). For each word that appears in the dictionary, our classifier will have a Boolean feature that will be true if and only if the word appears in the message that we are trying to classify.

Algorithm Framework

Since the intermediate probabilities can be quite small, we use the summation of log probabilities instead of the multiplication of probabilities in order to avoid underflow errors. In other words, instead of calculating $\prod_i P_i$, you will calculate $\sum_i \log P_i$ since $\prod_i P_i = \exp\{\sum_i \log P_i\}$. This implies that you will store $\log P_i$ instead of P_i for all of the following probabilities.

To learn the naive Bayesian classifier, you should do all the following steps **only for the training set**.

- Read all message files from the training set and identify which words from the dictionary appear in each file. Typically, classifiers use word counts rather than word presences but, for this project, we will be using word presences. As we have mentioned before, we are not concerned with words appearing in the headers of the messages. So, you should first parse the message file until you read an empty line (indicating the end of the header). Then, you should read the rest of the file word by word, strip the word of any punctuation marks and convert all letters that appear in the word to lower case. Note that the resulting string will be empty if the word only contained punctuation marks. You can modify and use the functions `ParseFile` and `ProcessWord` which can be found in the `main.cpp` file in the archive that we provide.

- Determine the prior probabilities for each class $c \in \{$ `comp.graphics`, `sci.space`, `talk.politics.mideast`, `rec.sport.baseball` $\}$:

$$P(C = c) = \frac{\#\{\text{Messages belonging to class } c\}}{\#\{\text{All messages}\}}$$

  Make sure that:

$$P(C = \texttt{comp.graphics}) + P(C = \texttt{sci.space}) + P(C = \texttt{talk.politics.mideast}) + P(C = \texttt{rec.sport.baseball}) = 1$$

- For each word w in the dictionary and for each class $c \in \{$ `comp.graphics`, `sci.space`, `talk.politics.mideast`, `rec.sport.baseball` $\}$, determine $P(w|C = c)$, the probability that a message from class c contains the word w:

$$P(w|C = c) = \frac{\#\{\text{Messages belonging to class } c \text{ that contain } w\}}{\#\{\text{Messages belonging to class } c\}}$$

Note that, if $P(w|C = c) = 0$ (if the word has never appeared in messages belonging to class c in the training set), then, for any message in the test set that contains w, our classifier will predict that the probability that the message belongs to class c is 0 (you can find an example of this in Homework 8, Problem 6, where $\mathrm{P}(\neg L|\neg S, H, W) = 0$, because $\mathrm{P}(H|\neg L) = 0$).

To address this issue, we can pretend that we have observed every outcome one more than we actually did (called Laplace smoothing).

$$P_L(w \equiv true|C = c) = \frac{\#\{\text{Messages belonging to class } c \text{ that contain } w\} + 1}{\#\{\text{Messages belonging to class } c\} + 2}$$

$$P_L(w \equiv false|C = c) = \frac{\#\{\text{Messages belonging to class } c \text{ that don't contain } w\} + 1}{\#\{\text{Messages belonging to class } c\} + 2}$$

Once you have trained your Naive Bayesian Classifier using the training set, you need to evaluate its performance on the **test set**. For a message m and a word w, let w_m denote whether w appears in m (that is, $w_m = true$ if w appears in m and $false$ otherwise). A message is specified by the words that it contains, that is, m is a shorthand for:

$$\bigwedge_{w \in dictionary} (w \equiv w_m)$$

For each message m in the test set, compute the probabilities that it belongs to each class $c \in \{$ `comp.graphics`, `sci.space`, `talk.politics.mideast`, `rec.sport.baseball` $\}$:

$$P(C = c|m) = \frac{P(m|C = c) \cdot P(C = c)}{P(m)} = \frac{(\prod_{w \in \texttt{dictionary}} P_L(w \equiv w_m|C = c)) \cdot P(C = c)}{P(m)}$$

The message m is classified as belonging to class c with the highest $P(C = c|m)$. Notice that the term $P(m)$ can be dropped from the calculations without changing the resulting classification. Also notice that we are using Laplace smoothing, so we are using $P_L(w \equiv w_m|C = c)$ rather than $P(w \equiv w_m|C = c)$. Furthermore, remember that, in order to avoid underflows, instead of multiplying probabilities, we are adding their logarithms. Therefore, we classify message m as belonging to class c that maximizes the following expression:

$$(\sum_{w \in \texttt{dictionary}} \log P_L(w = w_m|C = c)) + \log P(C = c)$$

In case there are ties (which should be very rare), classify the message as belonging to the class with the smallest id (0 = `comp.graphics`, 1 = `sci.space`, 2 = `talk.politics.mideast`, 3 = `rec.sport.baseball`).

Evaluation

Your program should output three different files:

- **network.txt** This file should list the values of all conditional probabilities. For each word w in the dictionary and for each class c, this file should contain a line with the word, the id for the class (0 = `comp.graphics`, 1 = `sci.space`, 2 = `talk.politics.mideast`, 3 = `rec.sport.baseball`), and the value of $P_L(w|C = c)$, the probability that a message from class c contains word w (after performing Laplace smoothing, before taking the logarithm of the probability), separated by tabs.

- **classification.txt** This file should be similar to `test_list`, but, at the end of each line, should also list the prediction that your classifier makes for the message (insert a tab before the prediction and use the id of the class for the prediction).

- **classification-summary.txt** This file should contain a 4×4 matrix of integers. The integer in row r and column c should be the number of messages in the test set whose actual class's id is r and whose the predicted class's id is c.

We will upload sample solutions for the training and test sets that we provide.

Submission

You need to submit your solution of Part 1 as a PDF file named 'Part1.pdf'. It should contain the screenshots of the three different Bayesian networks you have developed for Part 1. You also need to submit three xml files named 'Part1a.xml', 'Part1b.xml' and 'Part1c.xml', one for each Bayesian network (using `'File'` → `'Save program'` generates these xml files).

You need to submit your solution of Part 2 as a zip file that contains your source code and a makefile (our sample code includes a makefile, but you might need to modify it as you add new cpp files. You can check the makefiles in Projects 1 and 2 to give you an idea of how to do so). Make sure that, when we execute the command `make` on `aludra.usc.edu`, your source code compiles and produces an executable named `NewsgroupClassifier`. When we run this executable, it should read the `dictionary`, `training_list`, and `test_list` files (that we will put in the directory that contains your makefile and the executable) and output the three files that we mention in the Evaluation section of Part 2.

You should submit all your files through the blackboard system by Wednesday, Nov. 18, 11:59pm. We have created a discussion board for this project on the blackboard system. Please also feel free to ask the TA in case you have any questions about the project.