Naive Bayesian Network for Stroke Prediction

Maneesh John

1 Introduction

In many real-world scenarios, it is necessary to predict certain quantities or outcomes using limited information. For example, in the medical field, such a problem might be predicting whether a patient will have a stroke based on information about their health and history. However, it is often impossible to make predictions with certainty. An algorithm which seeks to assess a patient's risk of stroke must look for common patterns among other patients, then use those patterns to estimate the likelihood that the new patient will have a stroke.

A simple approach to this problem is a naive Bayesian classifer. This family of classification algorithms uses Bayesian networks to represent probabilistic relationships between features and labels. In order to test the effectiveness of this approach on a real-world problem, I created and trained a naive Bayesian network on a stroke prediction dataset. I then used that trained model to infer the probability that certain patients will have a stroke.

2 Background

Classifiers are algorithms or models designed to take a given observation and identify which of a set of categories it belongs to. Classifiers decide which *label*, or category, best fits an observation by analyzing its *features*, which are individual measurable properties of the observation. In order to learn patterns in the features, many classifiers such as Bayesian networks require training data.

2.1 Stroke Prediction Data

The data used for this project was obtained from "Stroke Prediction Dataset" by user "fedesoriano" from Kaggle, a company which provides public datasets from various sources [1]. The original source is confidential. The dataset contains 5110 observations and 12 different features. Some features such as "age" are numeric, while others such as "gender" are categorical.

For simplicity, the features considered are all binary or very close to binary: gender, whether the patient lives in a rural or urban area, hypertension, heart disease, and whether the patient has ever been married. Each of these can take one of two values. For example, each patient can have

"hypertension" equal to 0 (meaning the patient does not have hypertension) or 1 (meaning the patient has hypertension).

Of the five features considered, "gender" is not quite binary, as there is one patient in the gender category "Other." This patient was not included in the training data because of difficulties implementing random variables capable of more than two values. Only binary features are used to train the Bayesian network, as those are the easiest to work with.

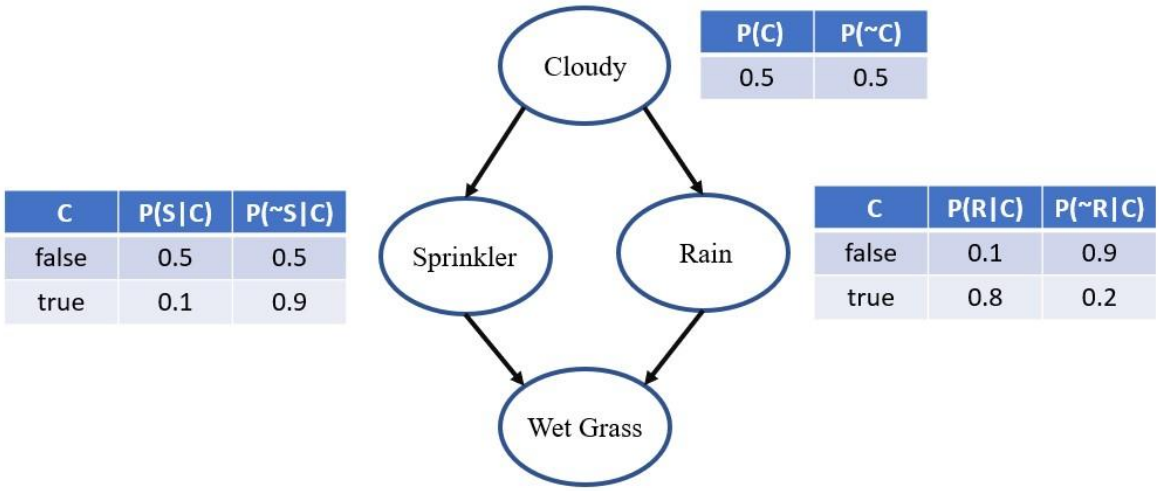
Each patient in the dataset also has an attribute "stroke" which represents whether or not the patient had a stroke. Its value is 1 for patients who had a stroke and 0 for patients who did not have a stroke. Because we are attempting to predict risk of stroke, this "stroke" attribute is the ground truth that we can compare predictions against.

2.2 Bayesian Networks

In order to make classifications, naive Bayes classifiers use Bayesian networks to represent relationships between features and labels. A Bayesian network is a probabilistic model that represent *random variables* and the conditional dependencies between them. In statistics, a random variable is something that takes a value depending on a random process. In the case of this dataset, each of the features and the label can be considered a random variable. Each random variable also has a probability distribution, which is a set of probabilities that show how likely it is to take any of its possible values. A *conditional probability table* (CPT) shows how a given variable's probability distribution is affected by the values of other variables.

Dependencies between random variables are represented using a directed acyclic graph (DAG). A *directed graph* consists of vertices and edges such that each edge connects two vertices, and each edge has a direction. A directed *acyclic* graph is a special case of a directed graph which has no directed cycles. In other words, a DAG cannot have edges forming closed loops. In a Bayesian network, random variables are used as vertices of a DAG, and the conditional dependencies between these random variables form the directed edges of the graph.

A simple example is shown in the diagram below. This Bayesian network has four binary random variables, each capable of being "true" or "false." The arrows between nodes represent edges in the DAG.



P(C)	P(~C)
0.5	0.5

C	P(S|C)	P(~S|C)
false	0.5	0.5
true	0.1	0.9

C	P(R|C)	P(~R|C)
false	0.1	0.9
true	0.8	0.2

R	S	P(W|R,S)	P(~W|R,S)
false	false	0.10	0.90
	true	0.90	0.10
true	false	0.90	0.10
	true	0.95	0.05

Edges of the DAG pointing toward a random variable indicate that the probability distribution of that variable depends on its parents in the graph. For example, the variable *W* is dependent on both *R* and *S*, which is represented by the arrows from *R* to *W* and from *S* to *W*. The likelihood that the variable *W* will have a certain value depends on the values of each of the variable's parents. Thus, that random variable is said to be *conditionally dependent* on its parents. This dependency is represented using edges in the DAG, as well as in the conditional probability tables stored in each vertex, as illustrated in the example above.

A variable's conditional probability table can be used to find its probability distribution given that the values of its parents are known. For example, suppose we need to find *P(R|C=true)*, the probability that *Rain=true* given that *Cloudy=true.* After finding an entry in the *Rain* table that matches the condition *Cloudy=true*, the probability distribution of *Rain* evaluates to *P(R)* = 0.8 and *P(~R)* = 0.2.

2.3 Naive Bayes

In the case of naive Bayes classifiers, the label is the dependent variable, and each feature is an independent parent of the label. Naive Bayes classifiers are among the simplest kinds of

Bayesian networks, because they rely on a key assumption that all of the feature variables are independent of each other.

For instance, in the stroke prediction dataset, the binary features "hypertension" and "heart_disease" may or may not be independent. Regardless of whether there is a correlation between these two features, a naive Bayes classifier assumes that the two are independent. In other words, it is assumed that having heart disease does not affect a patient's probability of having hypertension, and vice versa. Although the assumptions required for a naive Bayes classifier may oversimplify the true relationships between featusres in the data, such classifiers are efficient to design and train because of their simplicity.

Training a naive Bayesian network requires a training dataset. The goal of training is to compute an estimate of the CPTs that show the relationships between each feature and the label. This can be accomplished easily for each feature variable *F* using the formula

$$P(F = f) = \frac{occurences\ of\ f}{total\ observations}$$

For example, if training on the stroke prediction dataset, *P(stroke=true)* can be computed by counting the number of patients that have the attribute *stroke=true*, and then dividing by the total number of patients. This simplicity is due to the fact that all the features are independent.

The process is more complicated for the label variable *L*. Because the label is dependent on each parent, its CPT will have as many probability distributions rows as there are combinations of values for its parents [2]. In the stroke dataset, the label is binary, but in general, the table would have a column for each possible label value. For each possible value *l* of *L* and each unique combination of feature values *f*, the conditional probability can be computed as

$$P(L = l \mid f) = \frac{occurrences\ of\ l\ and\ f}{occurences\ of\ f}$$

This computation must be repeated for each possible value of *L* and each combination of feature values *f* in order to fill the label variable's conditional probability table.

2.4 Inference by Enumeration

One of the requirements for using a Bayesian network as a classifier is an inference algorithm. An inference algorithm computes the probability distribution for a query variable given some *evidence*, which is a set of other variables whose values are known. With a query variable *Q*, and

evidence e being the set of variables $E_1, E_2, \ldots, E_k$ equal to values $e_1, e_2, \ldots, e_k$, the conditional probability desired is the probability of Q given e, or $P(Q \mid e)$. Any variables in the network besides the query variable and the evidence variables are called *hidden* variables.

The simplest technique to compute this probability distribution is "inference by enumeration." This algorithm visits the network's nodes in topological order, selects entries from conditional probability tables that are consistent with the given evidence, and sums out other unknown variables to find a probability distribution for the query [2]. The algorithm is derived from the formula for conditional probability, which states that

$$P(X \mid Y) = \frac{P(X, Y)}{P(Y)}$$

It is important to note that $P(Y)$ is a constant. The division by $P(Y)$ can therefore be replaced by a normalization constant α. This formula can be extended for multiple variables, which leads to the formula for inference by enumeration [3]. Given query Q and evidence e, the algorithm can be expressed mathematically as a summation, where h represents each possible combination of values of the hidden variables:

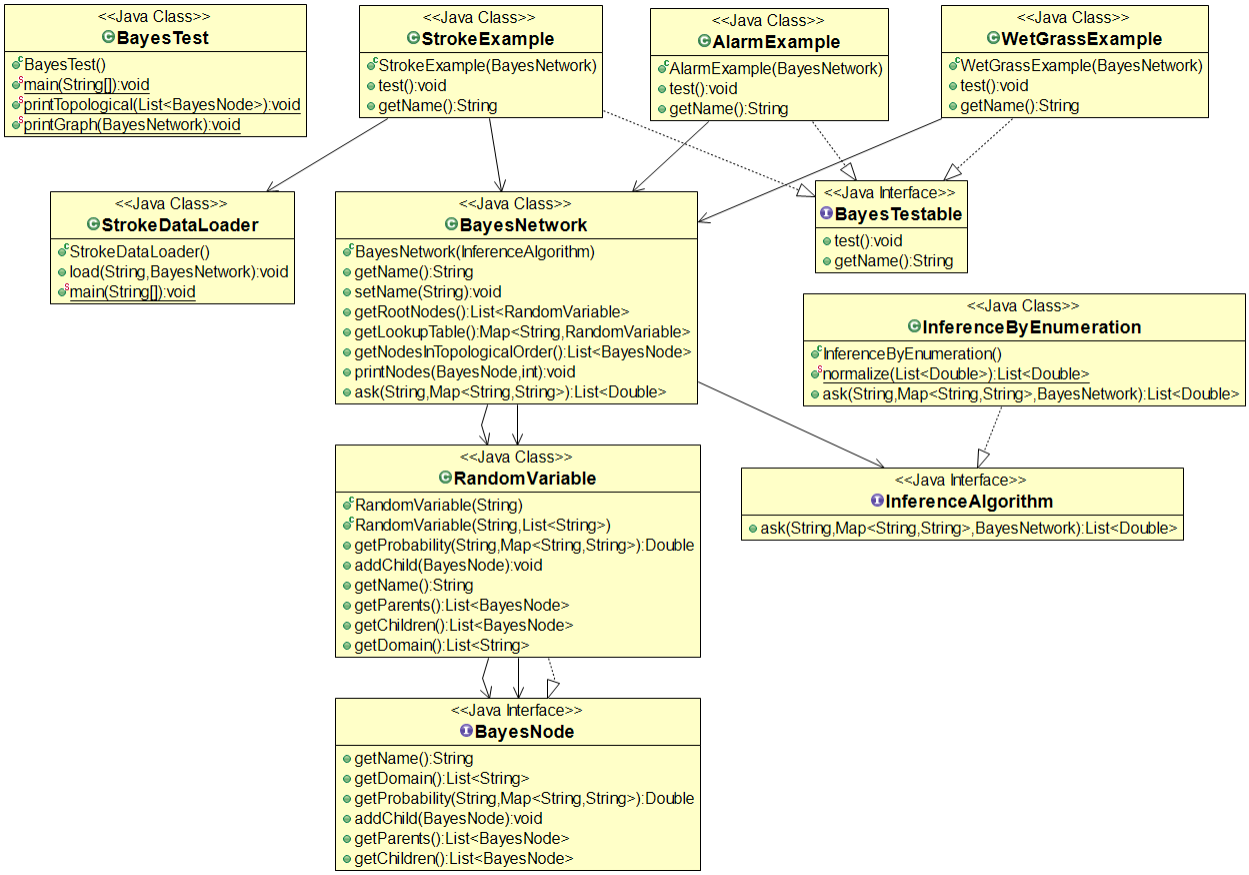
$$P(Q \mid e) = \alpha P(Q, e) = \alpha \sum_{h} P(Q, e, h)$$

An inference algorithm allows a Bayesian network to make predictions. A trained network can use inference by enumeration to compute $P(label \mid evidence)$, the probability distribution of the label given some known features of a patient. For example, such a model could compute the probability that a patient will have a stroke given that the patient is male and has heart disease. The effectiveness of such inferences can be demonstrated by testing the Bayesian network on data where the true labels are known.

3 System Description

A Bayesian network was designed from scratch, written in Java without using any external libraries. Code for the inference algorithm was based on pseudocode from Russell and Norvig's *Artificial Intelligence: A Modern Approach*. The inference algorithm pseudocode is available on GitHub [4]. In order to read the CSV file that contains the stroke prediction data, the code uses

one external library, Apache Commons CSV [5]. All the source code for this project (not including Apache Commons CSV) has been made available online [6].



The diagram above shows the relationships between classes and interfaces in the code. BayesNetwork is the class that represents the network itself, as a graph of interconnected nodes. BayesNode is an interface representing a node of the Bayesian network. This interface is implemented by the class RandomVariable, which represents a binary random variable with a table of conditional probabilities and connections to other BayesNode objects (such as parents and children in the graph). The inference algorithm is implemented in InferenceByEnumeration, which contains a static method that takes evidence, a query, and a BayesNetwork object as input and returns a probability distribution for the query variable.

Building and testing a BayesNetwork is demonstrated with a few simple examples in BayesTest. This class can test three different examples (WetGrassExample, AlarmExample, and StrokeExample) which demonstrate how to build and test BayesNetwork objects. WetGrassExample builds the example network illustrated in the diagram in the Background

section. StrokeExample is the most important one, as it demonstrates the training and testing of a BayesNetwork on the stroke dataset. StrokeExample also uses StrokeDataLoader to load the stroke prediction dataset from a CSV file. StrokeDataLoader manages the training of the BayesNetwork, and StrokeExample manages the testing and displays the results.

4 Demonstration

Ten patients were selected from the stroke prediction dataset. Five were patients who had strokes, and five were patients who did not have strokes. The rest of the stroke dataset was used to train the naive Bayesian network. After training, the ten selected patients were used to test the network's effectiveness. The patients are shown below, sorted by estimated probability of stroke.

Patient	gender	residence type	hypertension	heart disease	ever married	**Estimated probability of stroke**	**Actual value of stroke**
1	Male	Urban	0	1	Yes	**0.1935**	**1**
2	Female	Rural	1	0	Yes	**0.1402**	**1**
3	Male	Rural	0	1	Yes	**0.1321**	**1**
4	Male	Urban	1	0	Yes	**0.1184**	**0**
5	Female	Urban	0	0	Yes	**0.0473**	**1**
6	Female	Rural	0	0	Yes	**0.0442**	**1**
7	Female	Rural	0	0	Yes	**0.0442**	**0**
8	Female	Urban	0	0	No	**0.0189**	**0**
9	Male	Rural	0	0	No	**0.0055**	**0**
10	Male	Rural	0	0	No	**0.0055**	**0**

The Bayesian network generally predicts a higher risk of stroke for the patients who had a stroke than for the patients who did not have a stroke. For the patients who had a stroke, the average of the probabilities predicted by the Bayesian network is about 0.1115, but for the patients who did not have a stroke, the average probability is only 0.0385. While this is certainly not strong statistical evidence, it demonstrates the effectiveness of the Bayesian network.

An important point to note is that although naive Bayes networks are usually used as classifiers, this network does not classify the test patients. It only estimates each patient's risk of stroke as a probability distribution. The task of classification goes one step further, requiring a rule to decide whether a probability is "high" or "low" risk. Although various decision rules have been developed for that problem, a decision rule has not been implemented in this code.

5 Conclusion

I designed a naive Bayes network in Java and implemented an algorithm for inference by enumeration. I then trained and tested the model on a stroke prediction dataset, showing that naive Bayes networks can produce reasonable results. The stroke prediction dataset had many features, but even limited to only five binary features, a naive Bayesian network was somewhat effective at predicting which patients have a higher risk of stroke. This project demonstrates the effectiveness of a naive Bayes classifier on a real-world dataset.

References

[1] Kaggle. 2021. Stroke Prediction Dataset. Retrieved April 26, 2021 from https://www.kaggle.com/fedesoriano/stroke-prediction-dataset.

[2] David Suter and Qince Li. 2018. Bayes' Nets: Inference. Retrieved from https://cs.adelaide.edu.au/~dsuter/Harbin_course/BayesNetsInference.pdf

[3] Mitch Marcus. 2015. Probability, Conditional Probability & Bayes Rule. Retrieved from https://www.seas.upenn.edu/~cis391/Lectures/probability-bayes-2015.pdf

[4] Peter Norvig and Stuart Russell. 2016. AIMA Pseudocode: Enumeration-Ask. Retrieved from https://github.com/aimacode/aima-pseudocode/blob/master/md/Enumeration-Ask.md

[5] Apache Software Foundation. 2020. Apache Commons CSV. Retrieved from http://commons.apache.org/proper/commons-csv/.

[6] Maneesh John. 2021. Github: naivebayesnet. Retrieved from https://github.com/maneeshrj/naivebayesnet.