Assignment 1

COMP9418 – Advanced Topics in Statistical Machine Learning

Lecturer: Gustavo Batista

Last revision: Monday 24th June, 2024 at 14:12

Instructions

Submission deadline: Sunday, 30th June 2024, at 18:00:00 AEDT.

Late Submission Policy: The penalty is set at 5% per late day for a maximum of 5 days. This is the UNSW standard late penalty. For example, if an assignment receives an on-time mark of 70/100 and is submitted three days late, it will receive a mark reduction of $70/100 * 15\%$. After five days, the assignment will receive a mark reduction of 100%.

Form of Submission: This is an **individual** or group of **two students** assignment. Write the name(s) and zID(s) in this Jupyter notebook. **If submitted in a group, only one member should submit the assignment. Also, create a group on WebCMS by clicking on Groups and Create and include both group members**.

You can reuse any piece of source code developed in the tutorials.

You can submit your solution via WebCMS.

Alternatively, you can submit your solution using give. On a CSE Linux machine, type the following on the command line:

```
$ give cs9418 ass1 solution.zip
```

Recall the guidance regarding plagiarism in the course introduction: this applies to this assignment. If evidence of plagiarism is detected, it will result in penalties ranging from loss of marks to suspension.

The dataset and breast cancer domain description in the Background section are from the assignment developed by Peter Lucas, Institute for Computing and Information Sciences, Radboud Universiteit.

Introduction

In this assignment, you will develop some sub-routines in Python to implement operations on Probabilistic Graphical Models. You will code an efficient independence test, learn parameters from complete data, classify examples, and estimate the performance of the classifiers using accuracy and calibration. Our classifiers will be based on Bayesian networks (Week 2), Naïve Bayes and Tree-augmented Naïve Bayes classifiers (Week 3).

We will apply these classifiers to the diagnosis of breast cancer. We start with some background information about the problem.

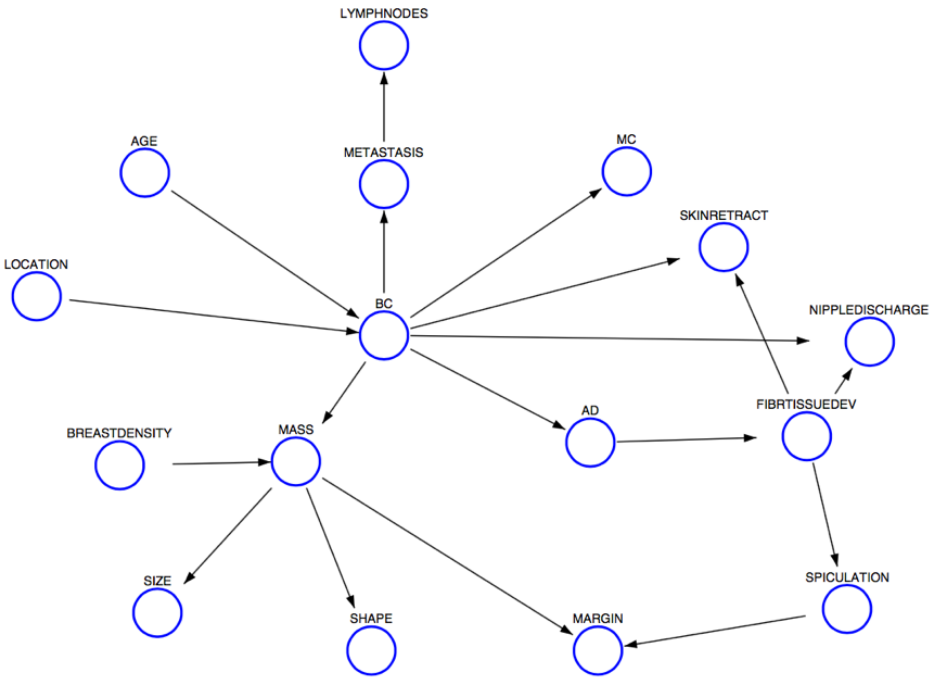
Background

Breast cancer is the most common form of cancer and the second leading cause of cancer death in women. Every 1 out of 9 women will develop breast cancer in their lifetime. Although it is impossible to say what

exactly causes breast cancer, some factors may increase or change the risk for the development of breast cancer. These include age, genetic predisposition, history of breast cancer, breast density and lifestyle factors. Age, for example, is the most significant risk factor for non-hereditary breast cancer: women aged 50 or older have a higher chance of developing breast cancer than younger women. The presence of BRCA1/2 genes leads to an increased risk of developing breast cancer, irrespective of other risk factors. Furthermore, breast characteristics, such as high breast density, are determining factors for breast cancer.

The primary technique used currently for the detection of breast cancer is mammography, an X-ray image of the breast. It is based on the differential absorption of X-rays between the various tissue components of the breast, such as fat, connective tissue, tumour tissue and calcifications. Radiologists can recognise breast cancer on a mammogram by a focal mass, architectural distortion or microcalcifications. Masses are localised findings, generally asymmetrical to the other breast, distinct from the surrounding tissues. Masses on a mammogram are characterised by several features that help distinguish between malignant and benign (non-cancerous) masses, such as size, margin, and shape. For example, a mass with an irregular shape and ill-defined margin is highly suspicious for cancer, whereas a mass with a round shape and well-defined margin is likely to be benign. Architectural distortion is focal disruption of the normal breast tissue pattern, which appears on a mammogram as a distortion in which surrounding breast tissues appear to be "pulled inward" into a focal point, often leading to spiculation (star-like structures). Microcalcifications are tiny bits of calcium that may appear in clusters or patterns (like circles or lines) and are associated with extra cell activity in breast tissue. They can also be benign or malignant. It is also known that most cancers are located in the upper outer quadrant of the breast. Finally, several physical symptoms of breast cancer are nipple discharge, skin retraction, and palpable lump.

Breast cancer develops in stages. The early stage is in situ ("in place"), meaning that cancer remains confined to its original location. When it has invaded the surrounding fatty tissue and possibly has spread to other organs or the lymph, so-called metastasis is referred to as invasive cancer. It is known that early detection of breast cancer can help improve survival rates.



[10 Marks] Task 1 – Efficient d-separation test

In this part of the assignment, you will implement an efficient version of the d-separation algorithm. Let us start with a definition for d-separation:

Definition. Let $\mathbf{X}, \mathbf{Y}$ and $\mathbf{Z}$ be disjoint sets of nodes in a DAG G. We will say that $\mathbf{X}$ and $\mathbf{Y}$ are d-separated by $\mathbf{Z}$, written dsep$(\mathbf{X}, \mathbf{Z}, \mathbf{Y})$, iff every path between a node in $\mathbf{X}$ and a node in $\mathbf{Y}$ is blocked by $\mathbf{Z}$ where a path is blocked by $\mathbf{Z}$ iff there is at least one inactive triple on the path.

This definition of d-separation considers all paths connecting a node in $\mathbf{X}$ with a node in $\mathbf{Y}$. The number of such paths can be exponential. The following algorithm provides a more efficient test implementation that does not require enumerating all paths.

Algorithm. Testing whether $\mathbf{X}$ and $\mathbf{Y}$ are d-separated by $\mathbf{Z}$ in a DAG G is equivalent to testing whether $\mathbf{X}$ and $\mathbf{Y}$ are disconnected (ignoring edge direction) in a new DAG G', which is obtained by pruning DAG G as follows:

1. We delete any leaf node W from DAG G as long as W does not belong to $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}$. This process is repeated until no more nodes can be deleted.
2. We delete all edges outgoing from nodes in $\mathbf{Z}$.

More details on this algorithm are available on page 66 (and surrounding pages) of the textbook (Darwiche).

Implement the efficient version of the d-separation algorithm in a method `BN.d_separation(X, Z, Y)` that returns a boolean: true if $\mathbf{X}$ is d-separated from $\mathbf{Y}$ given $\mathbf{Z}$ and false otherwise.

[5 Marks] Task 2 – Markov blanket

The Markov blanket for a variable X is a set of variables that, when observed, will render every other variable irrelevant to X. If the distribution is induced by DAG G, then a Markov blanket for variable X can be constructed using X's parents, children, and spouses in G. A variable Y is a spouse of X if the two variables have a common child in G.

You will implement a method `BN.Markov_blanket(X)` that returns a Python set with the Markov blanket of X in the Bayes Net BN.

After implementing this function, you can use the d-separation test to assess the correctness of your Markov blanked implementation and vice-versa. You can use the definition of Markov blankets below to understand the connection between these two concepts.

A Markov blanket for a variable $X \in \mathbf{X}$ is the set of variables $\mathbf{B} \subseteq \mathbf{X}$ such that $X \notin \mathbf{B}$ and $X \perp \mathbf{X} \setminus (\mathbf{B} \cup \{X\}) | \mathbf{B}$.

[5 Marks] Task 3 - Learning the outcome space from data

In this and the following tasks, we will implement three classifiers: Bayesian network, Naïve Bayes and a Tree-augmented Naïve Bayes classifier. We will need to create an *outcome space* for all these implementations.

We defined the outcome space as a Python dictionary in the tutorials, and it maps each variable to a tuple with its possible outcomes. In the tutorials, we manually defined the outcome spaces for our examples. However, as we work with more extensive problems, learning the outcome space from data becomes more efficient.

Thus, in this task, you will implement a function `BN.learn_outcome_space(data)` that learns the outcome space from the pandas dataframe `data`. Refer to our tutorials for more details about how we defined the outcome space dictionary.

[5 Marks] Task 4 – Estimate Bayesian network parameters from data

Estimating the parameters of a Bayesian Network is a relatively simple task if we have complete data. The file `bc.csv` has 20,000 complete instances, i.e., without missing values. This task will estimate and store the

conditional probability tables for each graph node. As we will see in more detail in the Naïve Bayes and Bayesian Network learning lectures, the Maximum Likelihood Estimate (MLE) for those probabilities are simply the empirical probabilities (counts) obtained from data.

In this task, you will implement a method, `NB.learn_parameters(data, alpha)`, that learns the Bayesian network `BN` parameters. This method should work the same as the tutorial function, except implement additive smoothing with parameter α.

[5 Marks] Task 5 – Bayesian network classification

The previously described Bayesian network has a variable that plays a central role in the analysis. The variable `BC` (Breast Cancer) can assume the values "No", "Invasive", and "InSitu". Accurately identifying its correct value would lead to an automatic system that could help with early breast cancer diagnosis.

We will now implement two related methods, `predict_proba` and `predict`. We start with `predict_proba`, and the implementation of `predict` will be just a few lines of code with a call to `predict_proba`.

The method `BN.model.predict_proba(class_var, evidence)` **efficiently** computes the probability of an attribute `class_var` with complete data. As we are working with complete data, `evidence` instantiates all variables in the Bayesian network `BN` but `class_var`. This method returns a `DiscreteFactor` object with `class_var` as a single variable and the probabilities associated with each `class_var` value.

The method `BN.model.predict(class_var, evidence)` is a direct consequence of the previous method. It returns the MPE value for the attribute `class_var`.

This task requires an **efficient** implementation that differs from the one done in the tutorials, as it will only involve relevant factors. Ensure you watched Lecture 6 when we discuss the design and implementation of classifiers with complete data.

Pro tip: Our library has an `evidence2` method that sets evidence for a variable `X` and removes `X` from the factor domain. This method makes things easier when we want a resulting factor that does not mention the evidence variables.

[5 Marks] Task 6 - Naïve Bayes classifier structure

Let's work now on the Naïve Bayes classifier. This classifier is a Bayesian network with a pre-defined graph structure. We can learn this pre-defined structure directly from the dataset definition, i.e., the class attribute and the features.

As the Naïve Bayes classifier is a Bayesian network, we can use the existing `BayesNet` class to create a new class `NaiveBayes`. Thus, the Naïve Bayes class will inherit all the methods we implemented for the Bayesian networks.

You will start creating a method, `NB.learn_structure(data, class_var)`, that learns the Naïve Bayes graph structure from pandas data frame `data` using `class_var` as the class variable. This function should store a graph object with the learned graph in the attribute `self.graph`. The graph class was implemented in the first tutorial.

[5 Marks] Task 7 – Naïve Bayes classification

Similarly to Task 5, we will create two methods for implementing a classification procedure, but now the Naïve Bayes model. **To make things different, we will work with log probabilities this time. This is a very popular procedure for the Naïve Bayes classifier when used in text mining, as these applications often work with thousands of variables, such as words.**

The method `NB.predict_log_proba(class_var, evidence)` **efficiently** computes the probability of an attribute `class_var` with complete data. As we are working with complete data, `evidence` instantiates all

variables but `class_var`. This method returns a `DiscreteFactor` object with `class_var` as a single variable and the probabilities associated with each `class_var` value.

The method `NB.predict(class_var, evidence)` uses the method `predict_log_proba` to implement the classification with complete data. This function should return the MPE value for the attribute `class_var` given the `evidence`.

[15 Marks] Task 8 - Tree-augmented Naïve Bayes structure

Let's work now with the Tree-augmented Naïve Bayes classifier (TAN). The TAN classifier is a mid-term between a Naïve Bayes and a Bayesian network, and it allows a richer graph structure learned directly from data using mutual information.

We will start by creating a new method, `TAN.learn_structure(data, class_var)`, that learns the Tree-augmented graph structure from a pandas dataframe `data`. Refer to Lecture 6, slide 24 for the algorithm that describes the steps to learn the graph structure from data. Remind that mutual information (MI) was defined in Lecture 3, slide 29.

Also, the TAN classifier is a Bayesian network. Therefore, we can use the existing BayesNet class to help implement this classifier.

[5 Marks] Task 9 - Accuracy estimation

We have finished implementing the models and will compare their classification performance.

We start with the method `assess` which computes the accuracy of a given model. The model can be a Naïve Bayes, Bayes Net or a TAN. To work with different models, `assess` will call the method `predict` implemented for all three models.

Design a new method, `assess(model, data, class_var)`, that uses the test cases in `data` to assess `model` performance at classifying the variable `class_var`. This function will return the model's accuracy according to the examples in `data`.

Such a function should return the classifier accuracy (a value between 0 and 1) defined as:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP are the true positives, TN are the true negatives, FP are the false positives, and FN are the false negatives.

[5 Marks] Task 10 - Assessment with cross-validation

Implement a function called `cross_validation(model, data, class_var, k)` that returns the average accuracy and standard deviation of the model specified in `model` using `k`-fold cross-validation.

This function should use the previous task's `assess` function to obtain the classifier accuracy for each cross-validation fold. We provide a scaffold in the cell below.

[10 Marks] Task 11 - Helper Function for the Reliability Diagram and the Expected Calibration Error (ECE)

Now, let's assess our classifiers differently. We will use their probabilistic nature to check the accuracy of their probability estimates. In other words, we will measure the **classifier calibration**.

A well-calibrated classifier's predicted probabilities are accurate representations of the actual probabilities of the events. For example, if a calibrated classifier predicts a probability of 0.8 for a positive class, then approximately 80% of those predictions should be positive.

We will start with a helper function `bin_pos_prob(model, data, class_var, pos_label, bins)`. This function divides the examples in `data` into `bins`. Each bin corresponds to a probability range. We use uniform ranges for simplicity.

The examples are split into bins according to the predicted probability of being a positive example given evidence, i.e., $P(\text{class\_var} = \text{pos\_label}|\text{evidence})$.

The function `bin_pos_prob` returns three lists:

1. `inst_num`: the number of instances in bin i.
2. `pos_ratio`: the faction of positive examples in bin i relative to the number of instances in the same bin.
3. `mean_pos_prob`: the mean probability of positive given evidence for the examples in bin i.

[5 Marks] Task 12 - The Expected Calibration Error (ECE)

This task computes a probabilistic classifier's Expected Calibration Error (ECE). The ECE will be useful in providing a numerical score for the calibration quality of a given model.

The ECE measures the average discrepancy between predicted probabilities and the actual outcomes. ECE is computed by partitioning the predictions into bins, calculating the absolute difference between the average predicted probability and the actual fraction of positive instances in each bin, and then averaging these differences, weighted by the number of samples in each bin. A lower ECE indicates better calibration, meaning the predicted probabilities more accurately reflect the true likelihood of the outcomes.

The formula for the Expected Calibration Error (ECE) is:

$$\text{ECE} = \sum_{i=1}^{b} \frac{|B_i|}{n} \cdot |\text{acc}(B_i) - \text{conf}(B_i)|$$

where:

- b is the number of bins.
- $|B_i|$ is the number of samples in bin i.
- n is the total number of samples.
- $\text{acc}(B_i)$ is the observed frequency of the positive class in bin i.
- $\text{conf}(B_i)$ is the average predicted positive probability in bin i.

In other words, for each bin, the ECE calculates the absolute difference between the average predicted probability and the actual fraction of positives, multiplies this difference by the proportion of samples in that bin, and sums these values across all bins to get the final ECE.

[20 Marks] Task 13 – Report

Write a report (**with less than 500 words**) summarising your findings in this assignment. Your report should address the following:

1. Make a summary and discussion of the experimental results. You can analyse your results from different aspects such as accuracy, calibration, runtime, coding complexity and independence assumptions.
2. Discuss the time and memory complexity of the implemented algorithms.

Use Markdown and Latex to write your report in the Jupyter Notebook. If you want, develop some plots using Matplotlib to illustrate your results. Be mindful of the maximum number of words. Please be concise and objective.