CS760 Spring 2019 Homework 2

Due Mar 7 at 11:59pm

General instructions:

- Homeworks are to be done individually.
- For any written problems:
  - We encourage you to typeset your homework in LaTeX. Scanned handwritten submissions will be accepted, but will lose points if they're illegible.
  - Your name and email must be written somewhere near the top of your submission (e.g., in the space provided below).
  - **Show all your work**, including derivations.
- For any programming problems:
  - All programming for CS760, Spring 2019 will be done in Python 3.6. See the `housekeeping/python-setup.pdf` document on Canvas for more information.
  - Follow all directions precisely as given in the problem statement.
- You will typically submit a zipped directory to Canvas, containing all of your work. The assignment may give additional instructions regarding submission format.

Name: YOUR NAME

Email: YOUR EMAIL

Goals of this assignment. For this assignment, you will do the following:

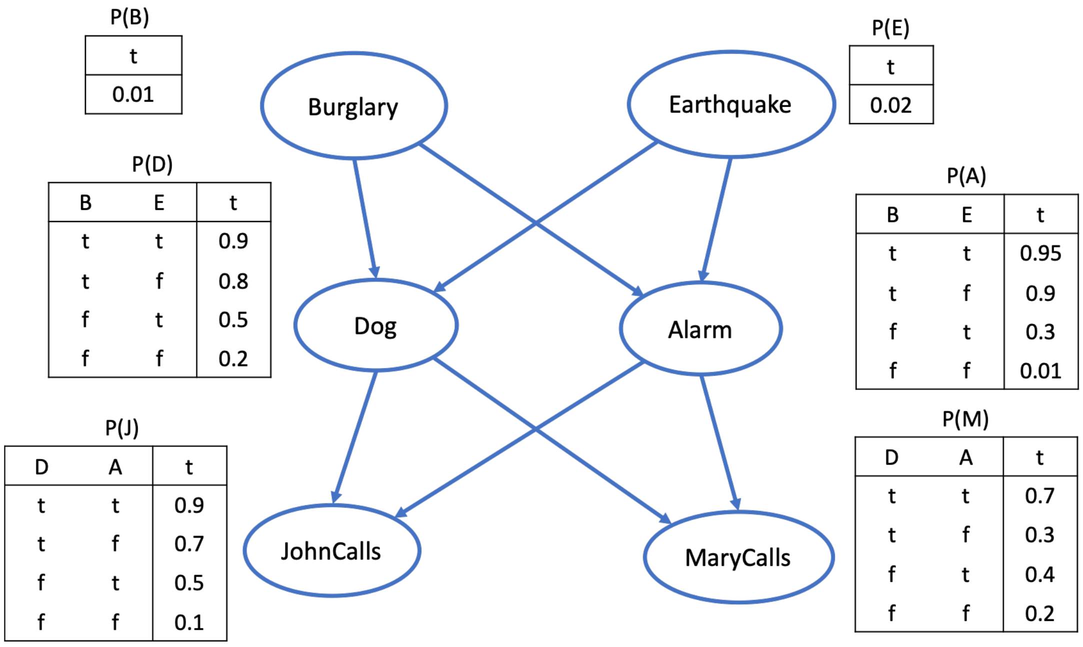
- conduct Bayesian network inference by enumeration;
- apply EM algorithm to deal with missing data;
- implement Naive Bayes and TAN;
- evaluate your methods through Precision/Recall curves, and understand their differences with ROC curves;
- compare your models using a paired t test to see whether one model is significantly better than the other.

Written Problems

NOTE: For the following written problems, put your answer in `hw2.pdf`. You are required to provide detailed solutions including the intermediate results for each step. Otherwise, you will not get full credit. You can also add figures or tables whenever necessary. If your solutions are handwritten, make sure they are legible.

1. (8 pts) Suppose you have a Bayesian network with 6 binary random variables shown as follows, where t and f stand for *true* and *false* respectively.

   Compute the probability: $P(d|b, \neg a, j, m)$.



P(B)

t
0.01

P(E)

t
0.02

P(D)

B	E	t
t	t	0.9
t	f	0.8
f	t	0.5
f	f	0.2

P(A)

B	E	t
t	t	0.95
t	f	0.9
f	t	0.3
f	f	0.01

P(J)

D	A	t
t	t	0.9
t	f	0.7
f	t	0.5
f	f	0.1

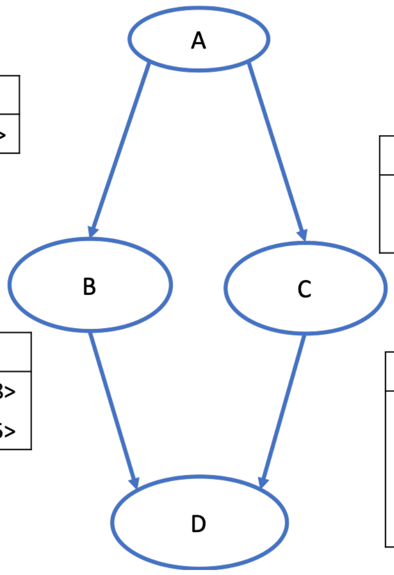
P(M)

D	A	t
t	t	0.7
t	f	0.3
f	t	0.4
f	f	0.2

2. Given the following Bayesian network and sample counts in each table, where sample counts `<n`$_{\texttt{true}}$`, n`$_{\texttt{false}}$`>`, there are `n`$_{\texttt{true}}$ samples with true labels and `n`$_{\texttt{false}}$ samples with false labels for this attribute. For example, `<138, 132>` in table `C|A` says given the condition of $A = true$, there are 138 instances are true and 132 are false with regard to attribute C.

   You need to answer the following two questions.

A
<270, 230>



A	C|A
t	<138, 132>
f	<137, 93>

A	B|A
t	<122, 148>
f	<115, 115>

B	C	D|B,C
t	t	<66, 67>
t	f	<66, 38>
f	t	<64, 78>
f	f	<62, 59>

   (a) (2 pts) Construct the conditional probability tables (CPTs) based on the above sample count tables, using maximum likelihood estimation. You need to both show the true probability `P`$_{\texttt{true}}$ and false probability P$_{\text{false}}$ for each case, and organize them in the format of `<P`$_{\texttt{true}}$`, P`$_{\texttt{false}}$`>`. For example, for the case `Y|X`$_1$`,X`$_2$, your answer will look like `<P(Y|X`$_1$`,X`$_2$`), P(¬Y|X`$_1$`,X`$_2$`)>`. Keep **at least 3 digits of precision.** (You may reuse the same structure as the above tables, just plugging in the conditional probabilities in the place of sample counts. For more information, please refer to the lecture notes `BNs-1.pdf`)

   (b) (10 pts) Show the result of one cycle of the EM algorithm to update the CPTs you derived in step (a), using 10 another instances with `A=true`, `B=false`, `C=?`, and `D=true` ('?' means missing value). Keep **at least 2 digits of precision**.

Programming Problems

Part 1

(50 pts) For this part of the homework, you are going to write a program that implements both **Naive Bayes** and **Tree Augmented Network (TAN)**.

- Your program can assume that all datasets will be provided in JSON files, structured like this example:

  ```
  {
      ‘metadata’: {
                      ‘features’: [ [‘feature1’, ‘numeric’],
                                    [‘feature2’, [‘cat’, ‘dog’, ‘fish’],
                                    ...
                                    [‘class’, [‘+’, ‘-’]
                                  ]

                  },

      ‘data’:    [[ 3.14, ‘dog’, ... , ‘+’ ],
                  [ <instance 2> ],
                  ...
                  [ <instance N> ]]
  }
  ```

  That is, the file contains *metadata* and *data*. The metadata tells you the names of the features, and their types.

  Real and integer-valued features are identified by the ‘numeric’ token. Categorical features are identified by a list of the values they may take. (**In this assignment, the datasets we provide to you only have categorical features.**)

  The data is an array of feature vectors. The order of features in the metadata matches their order in the feature vectors.

  JSON files are easy to work with in Python. You will find the `json` package (and specifically the `json.load` function) useful.

- For this assignment, you should assume:
  - In the JSON files that we provide, the class attribute is named `‘class’` and it is the last attribute listed in the feature section.
  - Your code is intended for binary classification problems only.
  - All of the attributes are discrete valued.
  - Your program should be able to handle a variable number of attributes with possibly different numbers of values for each attribute.
  - You use Laplace estimates (pseudocounts of 1) when estimating all probabilities.

- Specifically, for constructing the TAN model, your program should follow the following steps (refer to lecture notes `BNs-2.pdf` for more details):
  - First, compute conditional mutual information $I(X_i, X_j|Y)$ for every pair of X_i and X_j, where X_i and X_j are features and Y is the class.
  - Then, using the conditional mutual information values as weights, apply *Prim's Algorithm* to find a maximum spanning tree (choose maximal weight edges). To initialize this process, you need to choose the first attribute in the input file for V_{new}.

If there are ties in selecting maximum weight edges, use the following preference criteria:

1. Prefer edges emanating from attributes listed earlier in the input file.
2. If there are multiple maximal weight edges emanating from the first such attribute, prefer edges going to attributes listed earlier in the input file.

- To root the maximal weight spanning tree, pick the first attribute in the input file as the root, and assign edge directions in the MST.
- Finally, add a node for the class attribute Y, and assign an edge from Y to each of the features X_i.

- The program should be called `bayes`, and must be callable from a bash terminal, as follows:

  ```
  $ ./bayes <train-set-file> <test-set-file> <n|t>
  ```

  That is,

  - you ought to have an executable script called `bayes`;
  - the 2nd argument is the path to a training set file;
  - the 3rd argument is the path to a test set file;
  - and the 4th argument is a single character (either '`n`' or '`t`') that indicates whether to use Naive Bayes or TAN.

  You *must* have this call signature—otherwise, the autograder will not be able to analyze your implementation correctly.

- Your program should determine the network structure (in the case of TAN) and estimate the model parameters using the given training set, and then classify the instances in the test set. Your program should output the following in order:

  - The structure of the Bayes net by listing one line per attribute in which you list (i) the name of the attribute, (ii) the names of its parents in the Bayes net (for Naive Bayes, this will simply be the '`class`' variable for each attribute) separated by whitespaces. If an attribute has two parents, you should place '`class`' at the end of the line. After finishing this output, an empty line should be printed.

    For example, you program should first output something like this (given n attributes):

    ```
    <attr 1> <attr 1's first parent> <attr 1's second parent (if there is)>
    <attr 2> <attr 2's first parent> <attr 2's second parent (if there is)>
    ...
    <attr n> <attr n's first parent> <attr n's second parent (if there is)>
    <an empty line>
    ```

  - One line for each instance in the test-set (in the same order as this file), including (i) the predicted class, (ii) the actual class, (iii) and the posterior probability of the predicted class, separated by whitespaces. Again, an empty line should be printed after.

    Your output should have the following format (given N total test samples):

    ```
    <test sample 1's predicted class> <test sample 1's actual class> <probability>
    <test sample 2's predicted class> <test sample 2's actual class> <probability>
    ...
    <test sample N's predicted class> <test sample N's actual class> <probability>
    <an empty line>
    ```

  - The number of the test-set instances that were correctly classified, followed by a new line.

    The output format looks like:

    ```
    <a number of instances correctly classified>
    <an empty line>
    ```

- Let's look at an example, assume we have a dataset which has three categorical features: `attr1`, `attr2`, and `attr3`. For the `'class'` attribute, it has `positive` and `negative` labels. Your output may look like this:

  ```
  $ ./bayes train.json test.json n
  attr1 class
  attr2 class
  attr3 class

  positive positive 0.912132640925
  positive negative 0.817605375051
  ...
  positive positive 0.908394221338

  100
  ```

  You can test the correctness of your code using the files `tic-tac-toe_train.json` and `tic-tac-toe_test.json`, also we will provide you with a smaller subset of the dataset, called `tic-tac-toe_sub_train.json` and `tic-tac-toe_sub_test.json`.

  Note that your output requires the posterior probability having **12 digits of precision**. We will release some reference outputs later.

  For more details of implementing NB and TAN, you may look at notes `BNs-2.pdf`.

Part 2

(15 pts) Plot a precision/recall curve for both methods (NB and TAN), and answer the following question:

1. Compare the two curves, and make a comment about which method (NB or TAN) seems to have more predictive power. Explain why you think that (i.e. what features of the precision/recall curve lead you to this conclusion?).

Consider the class label listed first in the feature list of JSON metadata as the "positive" label (and conversely the second listed label as "negative"). You should use the given test set `tic-tac-toe_test.json` **only** to generate your points for this curve. Include the PR curve plots in `hw2.pdf`, along with your answers for the above question.

NOTE:

1. You do not need to generate plots or answer questions on `tic-tac-toe_sub*.json`.
2. You may not use any built-in library functions to generate your points for the precision/recall curve, i.e. you must do this manually and turn in your Python source code in a file named `pr_plot.py`. You may use the plotting library `matplotlib` for Python to generate your plots.

Part 3

(15 pts) For this part, you will compare your classifiers (NB and TAN) and use a two-tailed paired t-test to see if one of the systems is more accurate than the other.

Using the given data set named `tic-tac-toe.json`, use 10-fold cross validation to obtain 10 accuracy measures for each method. You'll notice that `tic-tac-toe.json` is simply a concatenation of its given train and test files. Use these accuracies to conduct your paired t test and discover whether you accept or reject the alternative hypothesis (that the classifiers truly differ in accuracy). Specifically, calculate the accuracy deltas for each cross validation fold and report the following values/answers:

1. Calculate the sample mean
2. Calculate the t statistic
3. Determine the corresponding p-value for a two-tailed t-test by looking up t in a t-table with $n - 1$ degrees of freedom. Use a threshold of $p = 0.05$ when determining if it is significant or not.

Record your answers (and show your work for partial credit) in your `hw2.pdf` file, and name your source code file as `t_test.py`.

Additional Notes

Submission instructions

Organize your submission in a directory with the following structure:

```
YOUR_NETID/
    # your script
    bayes

    # written answers and plots
    hw2.pdf

    # your source files
    <your various *.py files for part 1>
    pr_plot.py
    t_test.py
```

Zip your directory (`YOUR_NETID.zip`) and submit it to Canvas.

The autograder will unzip your directory, `chmod u+x` your scripts, and run them on several datasets. Their results will be compared against our own reference implementation.

Resources

Executable scripts

We recommend writing your scripts in bash, and having them call your python code. Your script `bayes` might look like this (given your source code named `bayes.py`):

```
#! /bin/bash

python bayes.py $1 $2 $3
```

If this doesn't make sense to you, try reading this tutorial:
`http://matt.might.net/articles/bash-by-example/`

Datasets

We've provided two datasets for you to experiment with:

- `tic-tac-toe*.json`. This is the Tic-Tac-Toe endgame database.

  The task is to classify whether the player "x" has won the game.

  See the UCI repository for more info:
  `https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame`

- `tic-tac-toe_sub*.json`. This is a randomly picked subset of `tic-tac-toe*.json` for your faster debugging.

We will provide reference output for these datasets - you will be able to check your own output against them.

During grading, your code will be tested on these datasets as well as others.