CS 6375 HW 4 Report

Parker Whitehead - pmw180000@utdallas.edu

Link to Colab:
https://colab.research.google.com/drive/14KfPiBM_vgLrprvJWeaIcZqWmSLlT-rl?usp=sharing

Abstract:

Given a Markov Network and a data file of X containing evidence variables (E), query variables (Q), and hidden (H) variables, we are to find the solution to argmax_q P(q | e). The example solution that was given to us utilized a logistic regression model, taking in E and predicting Q. However, in no way does this example utilize the Markov Network that we are given. In an effort to utilize this essential piece of data, I devised an algorithm that calculates the probability of Q given the known variables E via finding the product of relevant Markov Functions in the Markov Network. With this method, I was able to consistently score above 99.9% accuracy on all datasets given.

1. Data Exploration:

Initially, I put effort towards finding an ML model that could outperform the simple logistic regression. However, after using RNNs, SVMs, and various other models, I discovered that the difference was very marginal. Clearly, the crux of the issue regarding a lack of improvement over the simple Logistic Regression was the fact that we weren't including the Markov Networks . However, I could find no clearly defined method for determining missing elements in a Markov Networks , much less an existing library to handle my issue at hand. Thus, I turned to creating my own solution.

After reading over the Markov Networks we were given, I noticed that the variables were heavily intertwined in functions, consistently being a part of 5 functions. As such, I realized that I could use the functions that contained known data to quickly calculate the probability of any given 'q' being true or false.

2. General Overview of Solution:

Although there existed a utility class for handling .data files, there did not exist one for handling .uai files, at least to the extent that I wished. As such, I created a new python library called UAIHelper, which contained a class for storing essential data of a Markov Network. Using this class, I created a method that, when given a set of known data & the indexes of unknown data, was able to predict the probability that the unknown data would be true or false via the product of all functions in which the unknown data participated. Each of the probabilities would then be appended to the E matrix & returned to the user. This simplistic approach proved to be both effective as well as quick, taking no more than 4 minutes for each dataset on google colab's cloud servers.

Once the new matrix was calculated, I would then pass it over to the Logistic Regression model to train.

3. Analysis of Results:

The following are the results from the 4 Datasets using my method of intermediate Markov Probability estimation:

Sample_1_MLC_2022:
    Err: 0.0006200212214935164
    Score: 99.99876110056242
Sample_2_MLC_2022:
    Err: 0.0003103066146650235
    Score: 99.9990293618398
Sample_3_MLC_2022:
    Err: 0.03938934279949535
    Score: 99.9003562796231
Sample_4_MLC_2022:
    Err: 0.00063729238562342
    Score: 99.9983440668887

Consistently, the algorithm performs exceptionally well, achieving a score of 99.9+ on all datasets.

4. Possible Improvements:

Although this method performs well, there are few flaws. First, it fails to address the fact that a given q may have functions with other q variables or other unknown variables. Currently these are ignored, which can create unnecessary bias towards known variables & their functions. To counter this, I could possibly use my predicted data in a cross-validation fashion to help expand my ‘known’ variables for future calculations to collect greater accuracy.