Joan Wang

6/3/2014

CMSC 123 Spr 14

**REDD Energy Disaggregation: Final Paper**

**Introduction**

Energy disaggregation is the process of taking a dataset of the aggregate energy consumption of a household, and using a model to determine the separate contributions of each device in a household to the total energy consumption. The problem of energy disaggregation is important in that knowing the energy usage of each separate appliance in a household has been proven to cause consumers to make less wasteful decisions in terms of energy consumption. However, the problem is also nontrivial because installing devices to measure the separate energy outputs of each device in a house is costly and time-consuming, and researchers are thus looking for methods to accurately predict the separate outputs of each device based on the easily available aggregate energy signal of a house.

To examine the methods used in energy disaggregation, I used the REDD dataset, a 3GB, publically available documentation of the energy usage for 6 houses in the Boston area. Collected by researchers at MIT, the dataset is unique in that it is the first publically available big dataset that separates energy usage for the houses by the usage of each appliance, making it possible to train energy disaggregation models. The dataset is split into low-frequency and high-frequency data. For the purposes of this project, I concentrated on the low-frequency dataset, which contains the energy output in millisecond intervals for each of the main appliances of each house.

**Project Design**

The purpose of this project is to compare different methods used in energy disaggregation. The REDD paper that introduced the REDD dataset stated that the researchers had obtained benchmark results using the Factorial Hidden Markov Method (FHMM). For this project, I tried to replicate the FHMM method, in order to compare the accuracy of my results against the REDD paper results. Additionally, I tried to implement a model using the K Nearest Neighbors algorithm to disaggregate the data. The accuracy of each model would be compared to determine which model was able to predict the data with the smallest amount of error.

**FHMM**

The Factorial Hidden Markov Model was briefly described in the REDD paper as their method for testing the dataset. The model can be described as a Hidden Markov Model with multiple chains of hidden states.

For the REDD data, the hidden states were the varying levels of “on” or “off” of the devices in each household, which were unknown (hidden) but would determine the energy output of each device. Since there are multiple devices in a house and thus multiple chains of hidden “on” and “off” states, the FHMM is applicable.

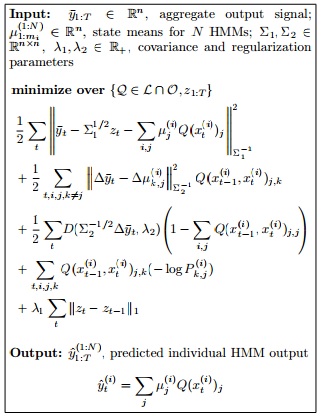
To use the FHMM, I first had to determine the parameters of the FHMM by training the model on five of the houses. After the parameters were set to optimally describe the houses, the FHMM was applied to the sixth house to predict the energy outputs of each of the devices in the sixth house, thus disaggregating its total energy signal.

The parameters of the FHMM are found by first considering the parameters of each device in a house, each of which can be modeled as an HMM. Letting N be the number of possible states per device (we used 4 states) and M being the number of possible observations (450 discrete observation values in our case), the components of an HMM lambda are {A, B, pi}. A is a NxN transition matrix such that a\_i,j describes the probability of transitioning from state i to state j. B is a NxM probability matrix such that b\_i,j is the probability of observing output (the energy signal of the device) j given the state i. pi is the initial probability matrix of size 1xN, such that pi\_1,j is the probability of initially observing state j.

To determine the components of the model lambda = {A, B, pi} for each device, I applied the standard Baum-Welch (EM) algorithm. Before using the algorithm, A, B, and pi are first initialized to random values, such that each row adds up to 1 (row stochastic). The algorithm involves an alpha pass that calculates two matrices, c and alpha, by moving forward in time through the training observations (the observed output of the device), and a beta pass which calculates a matrix beta by moving backward in time through the training observations, and is dependent on the previously calculated matrix c. A gamma pass is used to calculate matrices gamma and digamma using the alpha and beta matrices. The gamma and digamma matrices are used to recalculate the probabilities in A, B, and pi, and then a log probability of the model being a good fit for the training observations is used to determine if the process should be repeated.

After fitting the models to the training observations for each device in a house, the individual HMM models needs to be combined into a FHMM model to describe the entire house. There are several methods to do this through approximating the model, since exact inference under the FHMM model, which has k^n hidden states, is computationally intractable. The method used by the block Gibbs sampling method. Other methods include using sparse coding methods, and the structured mean fields method. For this project, I used the AFAMAP algorithm, developed in 2012 by Kolter and Jaakkola at MIT.

The AFAMAP algorithm takes the aggregate output signal for the test house, the state means from the HMMs calculated using the method above, the covariance matrix of the states, and regularization parameters lambdas (chosen so that the model does not assign all output to a signal zt used in the algorithm). The algorithm is included below[[1]](#footnote-1):



**KNN**

The K Nearest Neighbors algorithm predicts an output by determining the k closest elements in a training dataset to a portion of the testing data, and assigns the testing data the output observed from the training elements that they are closest to.

For this project, I applied the KNN method by considering portions of the aggregate energy output of a testing house, compared to portions of the aggregate energy output of the rest of the training houses. For example, if I used a granularity of 4 milliseconds (granularity is specified by the user), the aggregate energy output of the testing house would be split into 4 millisecond blocks. Each block would then be compared to every 4 millisecond block from the aggregate output data of the training houses. (Since each house had a different number of devices, only the devices that also appeared in the testing house were used to determine the aggregate output data in a training house). To compare the distance between the testing block with the training blocks, I used a Euclidean distance (distance = sum of aggregate energy output in testing block – sum of aggregate energy output in training block). The testing block was then assigned the same disaggregated output values as that of the training house block with the smallest distance from the testing block.

**Implementation**

To implement the FHMM, I first tested, during the prototype phase of the project, a version of the FHMM as written through the python library sklearn. However, each HMM would take about an hour to run, making it unviable for a large dataset since the pre-built functions were difficult to parallelize. Thus, I rewrote the FHMM program so that I could parallelize it using pthreads.

The data was appropriately formatted for the task by writing a python program data\_cleaner.py. This data was then used in the c program fhmm\_pthread.c, which used pthreads to determine HMMs lambda = {A, B, pi} for each device in a house. This program fits the HMMs to each device using the Baum-Welch algorithm described above, and parallelizes the process by having each thread work on developing a HMM model for a separate device in the house, so that all HMMs per house can be calculated at once. Once completed, the lambdas are written to a file “house\_(house\_id)\_hmm\_output.txt”.

The file above, containing the lambdas per device for the given training house, is then used in the python program afamap\_alg.py, which combines these HMMs into an FHMM for the house, and predicts the disaggregated energy information for a given testing house. Afamap\_alg reads in the HMMs for each house, as well as the aggregated output observations for the testing house. The state means for the HMMs are calculated, and the AFAMAP algorithm as described above is applied. The paper describing the AFAMAP program had solved the minimization problem using CPLEX on Matlab, which I am not able to obtain, and so I approximated the formula in the python program. To minimize the formula, I iterated over the formula using different variables for the minimization target vector Q, placing a limit on the number of iterations and a threshold amount of error between the predicted and actual output that was acceptable. To adjust the variables in Q, I used a random number generator for floats between 0 and 1, and either added or subtracted the random variable based on the success of the previous operation[[2]](#footnote-2). The predicted disaggregated output is then written to a file “output/fhmm\_(test\_id).csv”. The predicted output can be compared against the actual disaggregated output for accuracy using the program output\_compare.py.

For the KNN algorithm, I used map reduce in Python with mrjob to parallelize the process. To correctly format the data, I first wrote the program knn\_file\_setup.py. In the python program mrjob\_knn.py, I used mrjob to perform a map reduce task, where the mapper took the blocks of data from the aggregated output of the testing house, and used a Euclidean distance to compare this block with blocks from the aggregated output of training houses. The training block with the smallest distance from the testing block is then selected. The mapper outputs the id of the block (based on the time order in which the block appears in the aggregate data, so the first block is 1, etc.) as the key, with the value being the disaggregated data that corresponds with that of the testing block of the least distance.

Since there are five houses to test the testing data against, each block of testing data is entered into the mapper function five times, with each mapper job pairing the same block to a different disaggregated list of values. The reducer function then groups these five different values for the same block together. It finds the average of the disaggregated values in the output, and sets this as the predicted disaggregated value of the block. This data is then written to a file, which can be compared for accuracy against the actual disaggregated output for the testing data using ouput\_compare.py.

**Results**

The REDD paper reached 50% accuracy in its FHMM model using blocked Gibbs sampling. The AFAMAP paper claimed errors between 10% and 30% for the AFAMAP FHMM method, depending on the number of HMMs and the number of states used. The AFAMAP paper also claims a 10% to 40% error for the structured mean field method of combining HMMs into FHMMS. For my own algorithms, I have as yet been unable to obtain results.

**Conclusion**

Though my algorithms have not been able to obtain results, I can conclude from reviewing the literature and from my own experience coding these algorithms that the more mathematically rigorous methods such as the AFAMAP or the Baum-Welch algorithm are difficult to parallelize, since their time complexity is a result of the number of iterations over a complete data set, which cannot be separated into separate tasks. These algorithms can reach a high degree of accuracy, as seen in the papers referenced above, but the time cost taken to perform the algorithms is most likely large. The KNN method that I employed makes it much easier to parallelize the tasks, but the process is not mathematically rigorous.

For future explorations of this topic, I would like to focus more on the FHMM method, which seems to be the most accepted, mathematically proven method for disaggregation. I would try to build models using the structured mean field, sparse coding, and exact MAP methods, and compare these with the blocked Gibbs and the AFAMAP methods to determine the differences in time cost and accuracy of each.

**References**

“Algorithm 11.7: The Mean-Field Approximation Algorithm.” Structure Variational Approximations, Stanford, accessed 2014. http://pgm.stanford.edu/Algs/page-455.pdf

Ghahramani, Zoubin and Jordan, Michael I. “Factorial Hidden Markov Models.” MIT Artificial Intelligence Laboratory, 1996. <http://dspace.mit.edu/bitstream/handle/1721.1/7188/AIM-1561.pdf?sequence=2>

Jensen, Paul A. and Bard Jonathan F. “Nonlinear Programming Methods.S2 Quadratic Programming.” Operations Research Models and Methods. <http://www.me.utexas.edu/~jensen/ORMM/supplements/methods/nlpmethod/S2_quadratic.pdf>

Kolter, J. Zico and Jaakkola, Tommi. “Approximate Inference in Additive Factorial HMMs with Application to Energy Disaggregation.” Proceedings of the 15h International Conference on Artificial Intelligence and Statistics, 2012. <http://www.cs.cmu.edu/~zkolter>  
/pubs/kolter-aistats12.pdf

Kolter, J. Zico and Johnson, Matthew J. “REDD: A Public Data Set for Energy Disaggregation Research.” Computer Science and Artificial Intelligence Laboratory, MIT, 2011. <http://www.cs.cmu.edu/~zkolter/pubs/kolter-kddsust11.pdf>

Rabiner, Lawrence R. “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.” IEEE, 1989. [http://people.sabanciuniv.edu/berrin/cs512/  
reading/rabiner-tutorial-on-hmm.pdf](http://people.sabanciuniv.edu/berrin/cs512/reading/rabiner-tutorial-on-hmm.pdf)

Stamp, Mark. “A Revealing Introduction to Hidden Markov Models”. San Jose State University, September 28, 2012. <http://www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf>

Wilkinson, Darren. “A Simple Gibbs Sample.” Newcastle University, accessed 2014. http://www.mas.ncl.ac.uk/~ndjw1/teaching/sim/gibbs/gibbs.html

1. Though the AFAMAP algorithm was the most feasible for me, there were still components which I were unsure of how to determine. For example, I was unable to determine zt or D, which depended on “priors”, which I am unfamiliar with. The first term is calculated using the Mahalanobis distance, which was also hard to represent. In my code in afamap\_alg.py, I have tried my best to represent this algorithm, but there are some components that are missing, and the method used to minimize Q is not the quadratic variable method mentioned by the researchers. [↑](#footnote-ref-1)
2. For example, if adding the variable brought the predicted output closer to the target, for the next iteration, the variable would be added again. If adding the variable brought the predicted output further from the target, the next variable would be subtracted. [↑](#footnote-ref-2)