**Searching Correlated Genes by Using Machine Learning Technique**

Chieh-Wei Huang, advised by Dennis Shasha

**Abstract**

 Stochastic gradient descent is a machine learning algorithm for solving optimization problems such as classification or regression. In this thesis, stochastic gradient descent will be applied towards prediction of positively and negatively correlated genes.

 Thanks to Next-generation sequencing technologies, DNA sequencing has become easier and faster to carry out. It will be more and more common to obtain a newly sequenced species that shares a high phylogenetic similarity to other species which already have experimental data. These experiments usually provide genome-wide assay expression measurements. That is, for our research project, we start with inferring expression correlation measurements for genes within a hypothetical newly sequenced species *S*

 To infer the expression correlation of the gene pairs within species s, we will use correlation of gene pairs from other species to train the machine-learning program to predict the existence of edges between gene pairs in species s. Then, test the trained model on a testing set based on experimentally data. In our result, we achieved high recall and high precision for predicting highly correlated pairs on the testing set which would imply that stochastic gradient descent might be an efficient way to solve network inference problems.

**Chapter 1. Introduction**

**Active learning**

[If this is a quote, please note the reference] Active learning is a supervised machine learning method that queries the user actively in order to retrieve the desired output. In some experiments the dataset is extremely large and labeling of all examples is very expensive. Active querying of only the most useful data will lower the number of examples and cost. In our experiment, the algorithm will use only data obtained from active selection (highly correlated gene pairs in later discussion) that exhibit less noise to train our model.

**Gradient descent**

 Gradient descent is a machine learning technique for finding the optimized solution, which minimizes a differentiable cost function C(w). Cost function could be different for different purposes, such as classification or regression. In this thesis, gradient descent is used to solve regression problems in which stochastic gradient descent is used to find out the best weights to predict given data.

 In our experiment, the cost function C(w) in the experiment contains two functions, a loss function L(w) and a regularization term R(w)(Equation 3). The loss function, which computes how well the weights predict given data, is the average squared difference between the predicted value of weights given to each candidate and the value we know to be correct (Equation 4). *The Regularization Term* imposes some constraints on what weights should look like to improve performance on unseen data. Here, our regularization term assumes that the weights should be less than 1 to prevent extreme values on weights (Equation 5). The smaller the cost function is, the more accurate the prediction of data. [Lots of errors here. Formatting here is wrong. wj should be wj  Equation 3 is wrong. Equation 4 is totally wrong as the correction should be sensitive to the total error not on a single entry. I’m afraid the whole thing is completely wrong.

|  |  |
| --- | --- |
| w*i+1*=w*i* - **η\***∆C(w*i*)∆C(w*i*) =C(w*i*)’ in online learning or ∑C(w*i*)’ in batch learningC(w*i*) = L(w) + R(w)L(w*i*) = (1/m) \* (x·w*i* - y)2R(w*i*)=α∑w*i*2\*x is the variables in sample\*y is the score of sample\*w is the weights of the variables | (1)(2)(3)(4)(5) |

 So how does the gradient descent work? Let’s imagine cost function C(w) as a plane with a hole in it and we want to find out the location where it is the deepest. In other words, we want to know the weights (the coordinate of the location) that make the cost function smallest (deepest). First, we will randomly select a location (w0) on the plane to start from. Then, calculate the gradient (or slope), C(w0)’, of this location and move down the opposite direction of gradient for a step whose size is proportional to the size of gradient(**η\***∆C(w*0*)). Then, perform the previous step repeated iteratively until a certain criteria is met (in batch learning) or all the samples are used up (in online learning). In batch learning, the criteria will be when the size of the step is smaller then certain value, which means the gradient at that location is very small and it’s at the minimum so the weights (the coordinates of the location) at this location will give the minimum of the cost function.

 However, in a real situation, the plane could be bumpy with numerous holes (local and global minimum). The algorithm could fall into a smaller hole before arriving at a deeper one. We can solve this problem by starting from different random locations (100 random sets of weights in our experiment) and see which one goes to the deepest hole.

 What if each step (learning speed, **η**) is too small or too big? If the step is too small, online learning might not be able to reach the minimum before using up all the examples. If each step is too big, the algorihm could step across or out of the hole. e To avoid this, we will test the data on different step size (learning speed, **η**) and use the learning speed that gives the smallest cost function.

**Batch learning & Online learning of gradient descent**

There are two types of gradient descent:

1) Batch learning of gradient descent uses a group sample repeatedly to update the weights till it meets certain criteria. Batch learning is best suited to the situation when the sample is small. In each iteration, the gradient is derived from the sum of the error of all samples. Because of this summation in each iteration, the speed will be slow when sample is large.

2) Online gradient descent, also called stochastic gradient descent uses each sample only once to update the weights, which is usually applied to a large dataset. Compared to batch learning, the gradient here is calculated on each sample. Calculations become faster when the data is large but less accurate when data is small. Because of the huge amount of data in our experiment, we will use online learning (stochastic gradient descent) instead of batch learning to deal with our data.

**DNA Microarray**

 [Give a reference to this first sentence and put in quotes if it is a quote] A DNA Microarray (which is also called gene chip, DNA chip, or biochip) is an array of microscopic DNA spots attached to a small chip. Each spot contains only one specific DNA sequence, also known as a probe. Since a single chain of DNA can hybridize (Watson-Crick pair) with a single DNA chain having a complementary sequence, each probe can be designed to detect a specific sequence of DNA, which will correspond to a single RNA. The hybridization of probe and target is usually detected and quantified by targets labeled by fluorophore, silver, or chemiluminescence material. DNA Microarrays allow scientists to quantitatively assess tens of thousands of genes in a sample at the same time, greatly improving the speed and quality of genomic investigation.

 The microarray can be constructed in several ways depending on the number of probes, costs per chip, and the scientific question being asked. Arrays could contain as few as 10 probes or up to 2.1 million micrometer-scale probes. Usually, the probes on microarrays bind to the chip surface by a covalent bond (for instance, epoxy-silane, amino-silane, lysine, polyacrylamide etc.). The chip could be a solid surface (Affymetrix platform) or beads in holes (Illumina platform). In this situation, different DNA probes have to be synthesized in advance and spotted on the chip. This method [which method?] has an advantage that the DNA probes can be longer and the sequence is easy to customize.

 [You should have cites to references all over. This is very unprofessional] Alternatively, the probes can be synthesized directly on the microarray. For example, photolithographic synthesis uses light masks and light-sensitive agents to build a sequence one nucleotide at a time across the entire array. This method doesn’t need to synthesize probes in advance; however, it has a limitation on the length of probes. Also, this method is unable to change probe sequences due to the unmodified nature of a light mask.

 After getting the reads from a DNA microarray, it’s very important to get rid of noise, make different spots or chips comparable in order to analyze the large data output. Statistical tools are commonly used on microarray data; however, in this thesis, we will use a different approach, machine learning technique, to analyze the data and hope to develop a new and effective bioinformatics tool.

Chapter 2. Prediction of Expression-Correlated Gene Pairs.

**Input data**

The input for our algorithm will be two tables,[where is speciestab?] orthotab and edgetab as shown below:

1) Orthotab: target species| target gene | other species | other gene | orthology val

The orthotab gives the gene-to-gene orthology value. The orthology value here is the percentage of identity for top reverse blast hits. The reverse blast is describes as follow: First, take a gene from species A and blast against a database of gene sequence from your organism of interest. Then, the highest-scoring gene is taken and run through Blast again to a gene sequence database of species A. If this returns the gene of species A originally used as the highest scorer, then the two genes are considered putative orthologues. In order to prevent inferences on the same edge, a gene can only be orthologous with one other gene. That is, for each gene, we only use the reverse-blasting result with the highest Blast score.

2) Edgetab: species | geneA | geneB | correlation| p-value [These should be for a single species. If you didn’t do that, your results are useless]

 The edgetab gives a species, a Pearson correlation value of two genes within that species and a p-value. The correlation is based on all conditions to ensure that our tool will be able to predict correlation across many condition. The p-value is the probability of having a correlation this extreme through randomized non-parametric resampling. All the correlation values are experimentally determined rather than putative inferences from other species.

**Plant Model**

 The plant model we use contains three species, Arabidopsis (A), Medicago (M), and Glycine\_max (G)(Soybean). We selected these three species as an initial test case because they have abundant and reliable Affymetrix data and both Medicago and Soybean are legumes, two closely related species.

 This experiment will test the ability of our Network Inference Model to predict the Pearson correlation edges in a “target” species from correlation edges in a “source” species, and the gene-by-gene orthology between them. In this study, we analyzed only those genes in each species that are conserved across all edgetabs and orthotabs.

**Neighborly Network Inference Model**

 In order to train the stochastic gradient descent we have to set up a model first. The model we used is shown in Panel A.

The mean orthologous values in this equation are calculated by taking the average of two orthology values between a source pair and a potential target pair. For example, if the g1 and g2 are the source pair, and g1' and g2’ are the target pair and if g1 and g1’, g2 and g2’ have orthology values in orthotab, then we take the mean of the orthology values, in this case percent identity, between g1 and g1', and between g2 and g2'.

 We define two genes as “highly positively correlated” if their correlation is in the top 5% of all pairs, and “highly negatively correlated” if their correlation is in the bottom 5%, and “in between” otherwise. These three categories (positive, in between, or negative) will be a guide to determine how well our model is able to correctly predict which target pair is in which category.

**Training & Testing Set**

 Panel B shows an example of how we train and test the model. We first train the model according to the correlation data in Arabidopsis (A) and Glycine\_max (G) as well as orthology data between A and G. And then, use the coefficients in the trained model to predict the correlated edges in Medicago. The results of those predictions are then compared to experimentally determined correlation edges in M for validation. In another experiment, we switched the training set and testing set, which use Medicago (M) as source species to infer the correlated edges in target species Glycine\_max (G). We train only on those “highly correlated” gene pairs in order to get rid of the noise “in between” which make 90% of the data and, therefore, might mislead our model.

**Result**

 After we trained the model by using Arabidopsis (A) and Medicago (M) data, we obtained the following values: a1 = 0.0276, a2 = 1.2619, a3 = -0.8109 (The coefficients of model in Panel A). [Panel A also had a4. What’s going on?] Then, we test the coefficients on Arabidopsis and Glycine\_max by calculating recall and precision scores as follows:

|  |
| --- |
| *Positive.recall: number predicted to be +1 that are actually +1 / number that are actually +1**Positive.precision: number predicted to be +1 that are actually +1 / number predicted to be +1**Negative.recall: number predicted to be -1 that are actually -1 / number that are actually -1**Netative.precision: number predicted to be +1 that are actually -1 / number predicted to be -1**\*+1 here means genes that fall into highly positive correlated category**\*-1 here means genes that fall into highly negative correlated category* |



*Table 1.*

 Table 1. Shows that when using Arabidopsis and Medicago as the training set, Arabidopsis and Glycine\_max as the test set, we obtained 18,292 truly positively correlated genes predicted to be highly positive and 3,684 truly negatively correlated genes which are predicted to be highly negatively correlated. This gives us a recall of 0.91, for highly positive correlations, with a precision of 0.96, and for highly negative correlations, we get a recall of 0.62, and precision of 0.89(Table3.)

Chapter 3. Software

**Introduction**

 We made our experiment into a software package. The software will first find out the related gene pairs between target (2 species) and source (1 species). For example, g1 and g2 are source pairs, g1' and g2' are target pairs. If g1 and g1', g2 and g2' have an orthology relationship in the orthotabs, the software will record the index of those genes. After finding out those related pairs, the software will use one target as training set for stochastic gradient descent to train the model, and another target as testing set to examine the reliability of the model. The detail procedure will be described in a later paragraph.

**Environment**

 This software is written in Perl and calls R functions. The program can be run on a HPC server or a single local node that supports Unix, R and Perl. The program was developed and tested on USQ local host of NYU HPC. Since the NYU HPC doesn't have R preloaded, user has to load the R before running this program. In NYU HPC, the commands for loading R are:

module load intel/11.1.046

module load R/intel/2.9.2

**Architecture**



 This program mainly uses NNI.3species.R for conducting the process, manipulating the data, calling modules and printing out the output. NNI.3species.pl reads in the configuration file and runs NNI.species.R in slave mode. findLoop.HPC.R is in charge of sending the jobs to HPC and collecting the results when all jobs are completed. Stochastics\_GD.network.2.R contains all the functions necessary for stochastic gradient descent.

**Processing Procedure**

1) The program filters the correlation of edgetab by the given cutoff. The cutoff is user defined which can be a fixed value or “5per” for only top and bottom 5% of data. 2) Then, it calls R to submit the jobs to HPC to find the related gene pairs. User can also decide to compute this sequentially on a local node. 3) After all jobs are completed, the program will collect all the results and combine them together. 4) The program will choose one target species as a training set and another one as atesting set, then, call an R function to run the stochastic gradient descent on the training set. 5) After obtaining the coefficients, an R function will be used on the testing set to calculate its recall and precision. 6) The program will then exchange the training set and testing set and repeat steps 5) and 6) again. 7) At the end, the program will output the coefficients, recall, and precisions of these two experiments.

**Input data format**

The input data will include two kinds of tabs, edgetab and orthotab, which are both comma separated and contain a header as first line (the software will skip first line when read in the data). Each line in the edgetab is in the following format: species1\_geneA,species1\_geneB,correlation,value. [should not be this way. There should be a separate column for species] The format of orthotab is: species1\_geneA,species2\_geneC,percentage\_of\_identity. The range of all values should be between -1 to 1 or have a similar range. The software package contains three edgetabs (Arabidopsis, Glycine\_max and Medicago) and two orthotabs (Arabidopsis vs. Glycine\_max, Arabidopsis vs. Medicago) that are used in out experiment. The configuration file will read in those files by default.

**Configuration file**

Arguments can be set up in a configuration file called NNI.3species.R.conf. This configuration file is written in R. It contains: 1) working\_directory, which is the directory all results and temporary files are stored. 2) A.filename is the path of source edgetab, which is Arabidopsis in our experiment. The default value for the A.filename is the Arabidopsis edgetab in the package. If the parent directory is not specified, the file will be read from current working directory. 3) M.filename is the path of the edgetab of target1, which is Medicago in the experiment. The default value for the M.filename is the Medicago edgetab in the package. 4) G.filename, the path of the edgetab of target2, which is Glycine\_max in the experiment, uses Glycine\_max edgetab in the package as default. 5) orthotab.AM.path is the path of orthotab of source and target1, which is Arabidopsis and Medicago in our experiment. The default value is the orthotab of Arabidopsis vs. Medicago in the package. 6) orthotab.AM.path is the path of orthotab of source and target2, which is Arabidopsis and Glycine\_max in our experiment. The default value is the orthotab of Arabidopsis vs. Glycine\_max in the package. 7) job\_submit\_file\_path is the directory which contain PBS (Portable Batch System) scripts used to set up and launch jobs on HPC. Its default value is current working directory. 8) sampling\_num is the sampling number of edgetab and orthotab. User can set “NA” when using the whole data. 9) cutoff is a the cutoff of the correlations in edgetab. This should be a positive number or a string,"5per", for the cutoff is top and bottom 5% of data. 10) target.type is used to define which part of target edgetab is used. The program will only use highly correlated genes in target edgetab when target.type=“extreme” or use whole edgetab when target.type=“all”. 11) Source.type is used to define which part of source edgetab is used. It is defined by the same way target.type is. 12) chunknum is the number of jobs our task has separated into. This number depends on how many node user wants to or can use on the cluster (HPC). One can also set this to 1 in order to compute sequentially on a local node. 13) HPC\_ID is the user’s HPC ID, which is used for retrieving the information about how many running jobs left on to cluster so that the program can know when all the jobs is completed.

**Running the software**

 After setting up all the arguments in configuration file, make sure all the files in the list below are present and then run the follow command under Unix:

perl NNI.3species.pl NNI.3species.R.conf .

The result will be saved in to a csv file called Result in the working directory. The execution time depends on the number of jobs running at same time and the size of data.

The list of the files that must present when running the program:

|  |
| --- |
| NNI.3species.R :R code for calling modules and organize the data.NNI.3species.R.conf : Parameters for NNI.3species.RfindLoop.HPC.R :R code for doing the calculation parallelly on HPCfindLoop.R :R code for doing the calculation sequentiallygrn\_submit.sh : Shell file for the configuration of the HPC jobs.stochastic\_GD.network.2.R : R code for Stochastic gradient descentSGD\_network.HPC.launch.R : R code for launching calculation on HPC nodes  |

Chapter 4. Architecture of software for more species with large data

 After these results showing high precision and recall, we want to see if we can apply this method to species that are more diverse (with longer phylogenetic distance). In addition, in this preliminary experiment, we use only the genes in the intersection of orthotabs, so we are developing the software to deal with more species and more data.

**Architecture**



Since the new dataset is large, we will use C++ and Perl to replace most of the R codes in order to increase the speed. Also, the architecture of the software will also be simplified. NNI.pl will be the main script that reads in the files, conducts process, calls the modules and prints out the results.

**Environment**

The input data for the new software will be much bigger so it will be inefficient to use R. Instead, C++ and Perl will be the main language we use in the new software. That is, this software has to be run under the system supporting Unix, R, Perl and be able to compile C++.

**Format of input data**

The format of edgetab and orthotab is same as in preliminary software; however, a speciestab is introduced here because the training set contains several species instead of only one species. The format of speciestab is: species1, species2, similarity.

**Procedure**

 1) The software first reads in files for four species and use one species as target and other three as sources. 2) Then, it calls the function filter.by.cutoff to filter the edgetab by correlation. 3) Next, the program begins to search for related correlated pairs between source and target by submitting jobs to HPC. User could also choose to compute sequentially on a local node. 4) After all jobs on HPC are completed, the program will then collect all the results and combine them. 5) The program will randomly select half of the pairs in the collected data for training and another half for testing and apply stochastic gradient descent to train the model. 6) Use the coefficients from the trained model and the data from the testing set to calculate the precision and recall values. 7) Output the result.

**Reference**

1. Bertsekas, Dimitri (2003). *Convex Analysis and Optimization*. Athena Scientific.
2. Davidon, W. C. (1976). "New least-square algorithms". *Journal of Optimization Theory and Applications* **18** (2): pp. 187–197.
3. *S.-I. Amari. Differential-geometrical methods in statistics. Springer Verlag, Berlin, New York, 1990.*
4. *Stochastic Learning*. Lecture by Léon Bottou for the Machine Learning Summer School 2003 in Tübingen. Also in *Advanced Lectures on Machine Learning* edited by Olivier Bousquet and Ulrike von Luxburg, 2004
5. L ́eon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, Online Learning and Neural Networks. Cambridge University Press, Cam- bridge, UK, 1998.
6. Moreno-Risueno, MA, Busch, W, and Benfey, PN, Omics meet networks-using systems approaches to infer regulatory networks in plants. Curr Opin Plant Biol, 2009.
7. H. J. Kushner and D. S. Clark. Stochastic Approximation for Constrained and Unconstrained Systems. Applied Math. Sci. 26. Springer Verlag, Berlin, New York,1978.
8. A. Benveniste, M. Metivier, and P. Priouret. Adaptive Algorithms and Stochastic Approximations. Springer Verlag, Berlin, New York, 1990.
9. Schena M, Shalon D, Davis RW, Brown PO (1995). "Quantitative monitoring of gene expression patterns with a complementary DNA microarray". *Science* **270** (5235): 467–470.
10. Pollack JR, Perou CM, Alizadeh AA, Eisen MB, Pergamenschikov A, Williams CF, Jeffrey SS, Botstein D, Brown PO (1999). "Genome-wide analysis of DNA copy-number changes using cDNA microarrays". *Nat Genet* **23** (1): 41–46.
11. Bammler T, Beyer RP; Consortium, Members of the Toxicogenomics Research; Kerr, X; Jing, LX; Lapidus, S; Lasarev, DA; Paules, RS; Li, JL et al. (2005). "Standardizing global gene expression analysis between laboratories and across platforms". *Nat Methods* **2** (5): 351–356.
12. Emmert-Streib, F. and Dehmer, M. (2008). *Analysis of Microarray Data A Network-Based Approach*. Wiley-VCH.
13. Sprites, P; Glymour, C; Scheines, R (2000). *Causation, Prediction, and Search: Adaptive Computation and Machine Learning* (2nd ed.). MIT Press.
14. Faith, JJ *et al*. (2007). "Large-Scale Mapping and Validation of Escherichia coli Transcriptional Regulation from a Compendium of Expression Profiles". *PLoS Biology* **5** (1): 54–66.
15. Hayete, B; Gardner, TS; Collins, JJ (2007). "Size matters: network inference tackles the genome scale". *Molecular Systems Biology* **3** (1): 77.