# Automation of Biological Research: 02-450/02-750

# Carnegie Mellon University

## Homework 3

*Version: 1.0; updated 9/29/2015*

**Due: October 13 by 11:59pm**

**Hand-in:** Email your responses to: ABR-instructors@googlegroups.com

### Overview

This homework has 3 questions that require a small amount of programming. It is intended to test your understanding of the online learning algorithms covered in lecture 8.

**Question 1 (Hedge)**

**Background:** One of the earliest stages in drug discovery involves screening thousands to millions of small molecule **compounds** to determine which bind to the **target** molecule (usually a protein). These experiments can be done *in vitro* or *in silico* (i.e., in a computer). *In vitro* screens are the most accurate, naturally, but they are also significantly more expensive than *in silico* screens. Therefore, *in silico* screens are often used to rank compounds by their likelihood of binding, and thus create a priority list for *in vitro testing.*

*In silico* screens are performed by programs that input a description of a compound and the target molecule (e.g., their structures and chemical compositions) and predict the binding affinity between the pair of molecules. The program is run over each (compound, target) pair and the compounds are then ranked according to their predicted affinity to the target.

There are many programs capable of performing *in silico* screens, but experience has shown that their accuracies are very sensitive to the choice of target molecule. In general, no single problem outperforms the other, so we must identify the best program for each target. In this question, you will use online learning to identify the best program amongst a collection.

**Scenario:** Consider the following scenario: You have a library of a million compounds and you have selected a target protein. You don't have enough money to perform *in vitro* binding assays for 1 million compounds, so you are going to perform *in silico* screening to come up with a short list of, say, 1000 compounds for *in vitro* testing. You also have *n* programs that can be used for *in silico* screening, but you don't know which one will work best for your chosen target. You

decide to perform up to 100 *in vitro* tests on 100 randomly selected compounds to generate measurements that will be compared with the predictions made by the *n* programs to identify the single best program for this target.

Naturally, you could go ahead and run all 100 tests, compare the measured values with the predictions, and then select the best program. You suspect, however, that you might be able to identify the best program using far fewer than 100 tests by using a modification to the **Hedge algorithm** for online learning that was discussed in class.

**Files**: The files *squaredErrorLoss.m, getDataForTarget1.m , and getDataForTarget2.m* are provided to you as subroutines for computing two different loss functions. You will have to complete the implementations of the functions in the files *computeRegret.m,* and *hedge.m* and write some small scripts to run experiments and plot results.

*Note*: In order to simplify the code somewhat, the functions *getDataForTarget1* and *getDataForTarget2* will return *n* by 100 matrices that contain *all* the data from 100 trials. Your code, however, will process this file one column at a time, as if you were receiving the data in a sequential fashion.

**Tasks:**

A. (15 points) Complete the implementation in the file *computeRegret.m*. Recall that given a finite set of experts (hypotheses), $H = \{h_1 \ldots h_n\}$, data, $D = \{(x_1, y_1) \ldots (x_t, y_t)\}$, and a loss function, *L(h,x,y)= L(h(x),y)*, the regret associated with a specific expert, $h_i$, is: $R_i(t) = \sum_{j=1}^{t} L\big(h_i(x_j), y_j\big) - \sum_{j=1}^{t} L\big(h^*(x_j), y_j\big)$, where $h^*(t)$ refers to the best hypothesis (the one that minimizes $\sum_{j=1}^{t} L\big(h_i(x_j), y_j\big)$). You should read the comments in the file *computeRegret.m* so that you understand the inputs and outputs. You will use the squared error as your loss function. An implementation of the loss is given in the file *squaredErrorLoss.m*.

B. (20 points) Complete the implementation in the file *hedge.m* to implement a variation on the Hedge algorithm discussed in class. The version of Hedge discussed in class assumed binary labels. The version you will implement will hand real-valued labels. You should read the comments in the file *hedge.m* so that you understand the inputs and outputs. You may find the built-in matlab function *mnrnd* to be helpful. The weights should be updated as follows: $w_i(new) = w_i(old) * eta * \exp(-1 * |h_i(x_j) - y_j|)$. That is, weights will be exponentially down weighted according to both the learning rate and the magnitude of the error.

C. (5 points) Use the provided function *getDataForTarget1.m* to generate data for your experiment. This function has one argument, *n*, the number of experts. It returns an *n* by 100 matrix, called data, and a 1 by 100 vector, called labels. Matrix element (i,j)

contains the predicted binding affinity of the ith expert for the jth compound. Vector element (j) contains the true affinity. The function returns a different set of data each time you call it. Call this function with argument *n* = 10 and pass the results to your implementation of Hedge using a learning rate (eta) of 0.5. Gather the results (regrets, losses, weights). Repeat this 100 times. Plot the average of the regret curves with the average of the loss curves. Be sure to label your plots (use the matlab function *legend*).

D. (5 points) Repeat part (C) but with *n* = 100 (i.e., 100 experts). Plot both regret curves and both loss curves in the same figure (four curves, total). Be sure to label each curve. Compared to the results in part (C), does the average regret curve change much? Why or why not? Does the loss curve change much? Why or why not?
E. (5 points) Repeat part (C) but call *getDataForTarget2.m* to obtain the data. Plot both regret curves and both loss curves in the same figure (four curves, total). Be sure to label each curve. What can you conclude about target 2, in terms of the set of experts?
F. (5 points) Finally, create a version of Hedge where instead of selecting an expert at random, the prediction is made by taking a weighted average of the predictions made by the experts. Is the modified version more or less accurate than the version used in part (C)?
G. (5 points) We started this question planning on running up to 100 *in vitro* tests to identify the best program. Given the results of these experiments, approximately how many tests are needed before we can identify the best program?

**What to hand in:** your code, your plots, and your explanation of the plots.

## Question 2 (Winnow)

**Background:** Some diseases are the result of a mutation to a single locus (e.g., sickle-cell anemia and cystic fibrosis), but the majority of diseases are multi-factorial (i.e., they involve mutations to multiple loci). Finding disease-causing mutations is an active area of research.

**Scenario:** In this question, we will consider a hypothetical disease that can be caused by a mutation to *any* of *k* locations. The value of *k* and the set of disease-associated genomic positions are unknown to us, but it is assumed that *k* is much smaller than the size of the genome. You are given data on patients in a sequential fashion. The data consist of the patient's genome and a label indicating whether the patient has the disease, or not. Your goals are to (a) estimate *k*, the number of disease-causing position and (b) locate the *k* positions on the genome.

**Files**: The file *getDataForQuestion2.m is* provided to you as subroutine. It returns a 500 by 10000 matrix and a 1 by 500 vector. The matrix encodes the genomes for a population of 500

patients. Each genome has 10000 positions (to keep things simple). Matrix element(i,j) will be 1 if position j in the genome of the ith person has a different allele than that of a reference genome, and a 0 otherwise. The vector contains the labels for the 500 patients. The ith vector element will be 1 if the patient has the disease, and 0 otherwise.

*Note*: Once again, in order to simplify the code somewhat, the function *getDataForQuestion2* will return *all* the data for 500 patients. Your code, however, will process this file one row at a time, as if you were receiving the data in a sequential fashion.

You will have to complete the implementations of the functions in the file *winnow.m,* and write a small script to run the experiment.

**Tasks**:

A) (15 points) Implement the Winnow algorithm we discussed in class. You should read the comments in the file *winnow.m* so that you understand the inputs and outputs.
B) (5 points) Examine the weights computed using Winnow. Which loci do you believe are causing the disease?

**What to hand in:** your code and your response to part (B).

## Question 3

**Scenario:** Suppose you are a Synthetic Biologist trying to modify a species of bacteria so that it converts switchgrass into ethanol. You have a promising prototype, but its yield is highly variable. Your next step is to modify the design to maximize the yield. The core of the design is a complex network of 50 genes. The dynamics of the system are very complicated and you haven't been able to develop a useful computational model to help you optimize yield. You decide to pursue a series of knockdown experiments via RNAi.

One way to approach the problem is to perform a separate knockdown experiment for each of the 50 genes. Replicates would be needed, and a power analysis suggests that you should run 10 replicates for each knockdown experiment. Another approach would be to treat this as a multi-armed bandit problem where each gene represents a bandit and the ethanol yield for a given experiment is the reward.

**Files**: The file *RNAiSim.m* *is* provided to you as subroutine. It takes as input a gene id (from 1 to 50) and returns the ethanol yield for the knockdown experiment. The yields are variable and so multiple calls to the function will return different values.

You will have to complete the implementations of the functions in the files *bandit.m* and *replicates.m,* and write a small script to run the experiments.

**Tasks**:

A) (10 points) Implement the upper confidence bound strategy for multi-armed bandits discussed in class by editing the file *bandit.m*. The 'bandits' correspond to the 50 possible RNAi experiments you might run. Your code should return the *total* amount of ethanol produced after 500 pulls of the bandit arm (50*10).
B) (5 points) Implement the systematic strategy where you run 10 replicates for each of the 50 genes in the file *replicates.m*. Again, keep track of the total amount of ethanol produced during the 500 experiments.
C) (5 points) Both approaches should indicate which knockdown increases yield, but the multi-armed bandit strategy should also produce more total ethanol over the 500 experiments. Does it? If yes, what is the percentage increase?

**What to hand in:** your code and your response to part (C).