Bulut Cakmak, Oguzhan Bayol

ORF 307

05/06/2016

Final Project Written Description

**Problem 1**

To solve this problem, we’ve first created three parameter arrays: the first (xY) for the x values retrieved from Y.txt, the second (xN) for the x values retrieved from N.txt, and the last (xTest) for the x values retrieved from the test file YesNo.txt.

In addition, we have created a variable array called alpha to store the alpha values that will be generated by the program and define the coefficients of coordinates in the hyperplane. Another variable is b which gets assigned with the value of the regression intercept calculated by the program. The last variable is e which gets assigned with all error values of the regression line, that is, the distance of data points to the hyperplane separating the two regions.

We use an additional parameter array that defines the weight of the error values, which is called weight. We’ve assigned all indices of this array with the value (1/50) since both xY and xN arrays include 50 different vectors.

The objective function “distance” maximizes the weighted sum of the e values as instructed by the assignment page. There are two separate constraints on the values of e. The e value corresponding to each xY vector is equal to negative b plus the elementwise multiplication of alpha and the vector. The e value corresponding to each xN vector is equal to b minus the elementwise multiplication of alpha and the vector. The last constraint imposes a normal alpha vector, that is, the Pythagorean sum of its coordinates equals one.

After defining the problem, we read in the xY vectors from the Y.txt file and xN vectors from the N.txt file. We also read in the test vectors for xTest from the YesNo.txt file. At this point, we choose “minos” in AMPL to solve the problem. Any solver in AMPL could be used in this code, but for our program, minos is the quickest and the most accurate solver.

Finally, we apply a conditional test on each vector in xTest. For each vector in xTest, we calculate negative b plus the elementwise multiplication of alpha vector and the test vector. If this value is greater than or equal to zero (we don’t have it strictly greater than as a conservative measure to count for possible error in the data such as Y values lying on the hyperplane separating the two regions), then the program assigns Y to that vector. If it is less than zero, then the program assigns N to the vector.

Upon running the code, AMPL suggests using meminc = 0.334 in minos options, but given the limited time we had on this assignment, we decided minos as it is performed well.

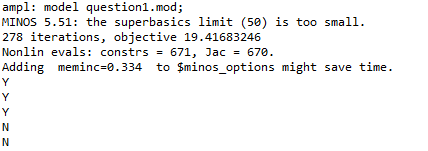


Figure : The output of question1.mod

**Problem 2**

This problem required major modifications to what we’ve done for the first problem because we needed a program that would identify all letters in the alphabet in a given test file.

First, we define a set called alphabet that contains all characters in the alphabet and the whitespace. Our first parameter array is called xData. It’s a two-dimensional array that stores the x vectors of each and every character from their training sets. The parameter arrays xTest3, xTest4, and xTest5 store the vectors of the test files.

The variables alpha, b, and e are defined analogously to those in Problem 1. There are slight differences since we now need to account for all characters in the alphabet. Alpha is now a two dimensional variable that contains 64 alpha values for each one of the 27 characters. b is now a one-dimensional array that contains 27 different values for each character. e is a two dimensional array that contains 27\*50 (one per each vector in the optimization problem) for each one of the 27 characters.

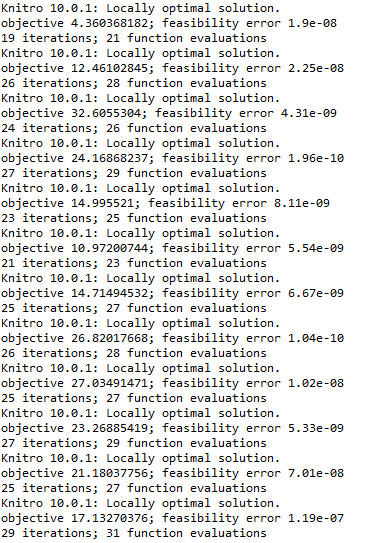
The weight parameter is categorized into two counterparts. weightCurrent is always (1/50) since this is a constant: the number of vectors examined for each character. weightOther also has a constant value of (1/(50\*26)): the number of all other vectors that are being examined against the character at hand.

The objective maximizes the weighted sum of the e values of the vector that’s being examined plus the weighted sum of the e values of the other vectors. (It uses the aforementioned weight parameters to do so.) Two constraints are set up to constrain the e values of the other vectors (those not being examined) to be the b value of each one of them minus the vector values multiplied with their corresponding alpha values. The other constraint ensures that the vectors of the examined character lie above the hyperplane, setting e equal to negative b plus the elementwise multiplication of alpha vector and each vector belonging to the character. The last constraint again requires the alpha to be a normal vector.

After these commands, we use read commands to retrieve data from the training sets. xData is to be filled with the data from these sets. The data from Slogan3.txt, Slogan4.txt, Slogan5.txt are to be read and saved to xTest3, xTest4, and xTest5.

At this point, our program uses “knitro” to solve the problem for each character, filling the parameter arrays with the data from the character and slogan files. Other solvers are not suggested, since knitro is significantly more effective and efficient than its counterparts.

Finally, the program reads in the data stored in xTest3, xTest4, and xTest5. For each vector in each set of data, the program calculates the value of negative b plus the elementwise multiplication of alpha and the vector for each hyperplane. The greatest result is stored in a parameter called “likelihood” and the program assumes the hyperplane yielding this result is the character the vector belongs to. We don’t impose a greater than or equal to zero condition this time to account for the error in the data. A vector in corrupt data could yield negative results for all hyperplanes. Also, a vector could yield positive results for multiple hyperplanes. In order to avoid conflicts arisen from these problems, our program finds the most optimal hyperplane and assigns each vector to that character. And with the help of a counter parameter called “m,” the program solves this optimization 27 times. The value of m also indicates which character we are examining.

The output of the program isn’t 100% correct, but that is because the handwriting could be misleading. For example, looking at the handwriting in G’s of the first slogan, the first two could easily be taken for a Q by a human observer as well. This justifies that the program’s output is wrong in cases such as distinguishing between G and Q (the first slogan), or M and X (the second slogan). Overall, it is easy to read the actual message if the user takes the initiative to understand and correct the output.

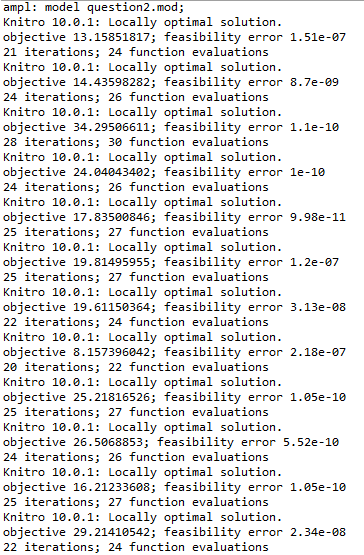
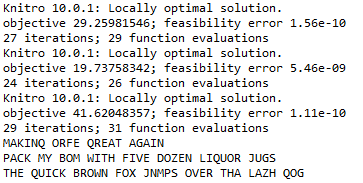


Figure 2: The output of question2.mod

Works Cited

“AMPL Format Strings for Printf and Sprintf.” *Format Strings for Printf and Sprintf*. AMPL, 22 Apr. 1996. Web. 06 May 2016.

Fourer, Robert, David M. Gay, and Brian W. Kernighan. “Parameters and Expressions.” (2003): 109-128. *AMPL Electronic Book*. AMPL. Web. 06 May 2016.

Fourer, Robert, David M. Gay, and Brian W. Kernighan. “Specifying Data.” (2003): 143-168. AMPL Electronic Book. Web. 06 May 2016.

“New in AMPL: Handling Character Strings.” *New in AMPL: Handling Character Strings*. AMPL, 22 Apr. 1996. Web. 06 May 2016.

“Knitro / AMPL Reference.” *Artelys Knitro 10.0 User’s Manual Guide*. Knitro, n.d. Web. 06 May 2016.

“Looping and Testing 1.” *Looping and Testing 1*. AMPL, n.d. Web. 06 May 2016.cx