Evaluation form – Progress Reports

V1.06 Feb. 2004

Project ID	2003252	Project Title	An Interactive Recommender System
Student Name	Cavan Yie	Supervisor	R. Zemel
Section #	6	Coordinator	D. Beresford

Presentation	Copy graded: ❒ electronic ❒ paper	❒ comment summary in paper
Coordinators Signature:		Grade /10 (each report worth 2.5%)

Technical Evaluation	**Total Mark**		**Suggested Mark**		**Notes**
Progress	10				• following game plan? (milestones accomplished; milestones missed & why)
Organization	10				• use of human resources? • use of non-human resources? • efficiency of efforts? • preparation & foresight?
Method	15				• are the modules / steps sufficiently tested when 'done'? • do these tests have good structure? • have the interaction specifications of the parts been properly developed?
Decision making	10				• suitable response to difficulties? • appropriate change of course when encounter obstacle or new development? • modification of process as necessary?
Creativity / Complexity / Effort	35	C/C _____ Effort x ____ x 35			Creativity and Complexity rom charts. Scores for creativity, complexity and effort are multiplied by maximum score / 1000 to get the grade.
TOTAL	80				(report #1 worth 7.5 & report #2 worth 12.5% of the student's final grade)
Supervisor	❒ Accept suggested technical mark ❒ Change technical mark (reasoning attached in separate sheet, marks in blank column above)				
Supervisor's Signature:					

Note to students: There is a design award, the Aloha award, that you might wish to apply for. Please check the course website for instructions on what to do to be considered for this award.

Project ID	2003252	Project Title	An Interactive Recommender System		
Student Name	Cavan Yie			Supervisor	R. Zemel
Section #	6	Coordinator/Administrator		D. Beresford	

Averages for this section: Presentation _____

Technical _____

Return this evaluation sheet as described below to

❒ Coordinator

❒ _____

Supervisors:
Please review the progress reports for the students in your group(s). The terms are further explained on the course website using the menu at the top of the website to get to the links on the page at
Student Information | Deliverables/Evaluation | Individual Progress Reports
[The course website is at http://courses.ece.utoronto.ca/ece496y1y/.]
In particular,

- the 'effort' assessment is generally .7 to 1.2, but normally close to 1.0. Follow the link to 'Marking Terms'
- the creativity/complexity mark comes from a graph. Follow the link to 'Mark Matrix'.

You may choose to accept the technical mark on the first page, or to change the marks. If you choose to change the mark, please enter the changed mark in each category on the first page and write a note of justification for the change. The student final grade will be determined after consultation.

Please do not return the reports to the students until you receive notification to do so.
Please return a copy of the first evaluation sheet to the Coordinator with a justification if necessary within a week.

The average marks for the section are shown above. There will be differences between the marks in each section. Marks between sections will be normalized after the final reports are marked (about mid-April). The students are already aware of this.

Supervisors Comments	
Supervisor's Signature:	

The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto

ECE496Y Design Project Course
Individual Progress Report

Title: An Interactive Recommender System

Project I.D.#: 2003252

Prepared by:	Cavan Yie yie@ecf.utoronto.ca
Supervisor:	Richard Zemel
Section #:	6
Section Administrator:	D. Beresford
Date:	February 23, 2004

Table of Contents

1. Executive Summary

In the past few years, e-commerce has brought upon an influx of major businesses going online. One of the key reasons for this is the promoting of online purchasing. A popular tool used for online consumerism is the recommender system. Current recommender systems found on the internet today at sites such as Amazon.com and Netflix.com are not sophisticated enough to generate quality recommendations. Most recommender systems online today simply look at what the customer has purchased in the past, and groups the user with other customers who have purchased the same items. Clustering users in this fashion readily leads to unwanted recommendations. The most important problem that exists in current recommender systems is the so-called "New User Problem". This problem arises when a new user begins using the system and receives unsatisfactory recommendations due to the lack of information the system has about the new user.

The objective of our project is to create an interactive recommender system for music that will accurately predict which songs a particular user will like as well as solving the "New User Problem". The uniqueness of this project lies in its interactive approach.

In our system, the user builds his/her own profile by interactively rating various songs in the database in order for the system to generate better recommendations. Active Collaborative Filtering (ACF) is used to make these recommendations. The overall system consists of three components: Apache/Tomcat web server for online usage, back-end MySQL database, and the Java ACF algorithm logic layer.

The majority of the project is progressing according to schedule. We are currently beginning the research phase of the project where we compare Active Collaborative Filtering to Collaborative Filtering. We plan to measure whether one approach works better than the other, with respect to generating quality recommendations.

2. Introduction

Improving the quality of recommendations is an important issue that needs to be addressed for online businesses to achieve economic growth. There is no question that the recommender systems found online today leave much to be desired. Only 7.4 percent of online shoppers who use recommender systems purchased recommended items. Only 22 percent of customers found the recommendations valuable, and a whopping 42 percent found no interest at all [6]. A website that recommends unwanted items may turn the customer off and is harmful to the business-customer relationship. Developing a system that delivers high quality recommendations is crucial for online businesses to keep loyal customers and to expand their clientele.

The problem with today's recommender systems is the fact that the system does not know enough about the customer when trying to make a prediction on what s/he may like. Simply knowing what the user has purchased in the past does not necessarily tell you what item s/he may want to purchase next. The process of Active Collaborative Filtering begins with "model fitting" which consists of creating a probability model from a given set of user ratings for songs. Based on what the system has learned from this data, it extracts the most important information out of a user (through rating songs) to intelligently generate recommendations. Through this interaction between system and user, the system begins to learn more about the user, and is then able to continually generate high quality predictions with increasing accuracy.

To reiterate, the project was decomposed into three components: Web application, Database, and ACF back-end implementation. The web application and database were developed by Bernie Ma and Andrew Yeung, respectively. I was responsible for the high-level implementation of the ACF algorithm logic layer. In this report, I will describe my progress with respect to the implementation of ACF, the integration of my component with the rest of the system, the research aspect of the project, as well as go into what milestone's still remain. Possible strategies to attack remaining milestones will be discussed.

3. Progress

3.1 Milestone *& Description - Implement Active Collaborative Filtering Logic Layer*

Includes writing code for model fitting as well as ACF algorithm, which includes an API for interfacing with web server. Due Jan 21, 2004.

Responsibility

Cavan Yie

Status at start of reporting period

Probabilistic Rating Model code complete, however only accessed fake test data since we did not have real user ratings in our database at the time. Some errors in the ACF algorithm needed to be corrected. I ran into performance issues – calculations were being performed extremely slowly. It took minutes to generate recommendations.

Status at end of reporting period

Fully functional ACF logic layer which includes model fitting module. Speed performance issues resolved. Completed Jan 4, 2004.

Actions

Code review and tedious code analyzing was performed on both Java modules. Corrected some logic in the ACF module - the generating of the prediction matrix was being done incorrectly. The prediction matrix was being generated for all users in the system, rather than generated for only the user using the system – redundant work. Changed data structures that the user profiles were being stored in because of performance issues.

Decisions

Originally, user profiles (personal ratings) were stored as Java Hashtable's – the key being the song ID, and the value being the song rating. The continuous lookup of items stored in a user profile was slowing the system down significantly. It was found that since profiles were stored in Hashtables and looking up a key in a Hashtable is very expensive (with respect to Java processing) a new data

structure was needed to store user profiles. I chose to store the profiles in integer arrays instead of Hastables. The array index represents the song ID, and the actual ratings are stored in the array. With the new data structures in place, the generating of recommendations and optimal queries was reduced to seconds.

Testing & Verification of Progress

I ensured that all model-fitted data was properly stored in data files for efficient future loading of data (so model fitting is not needed every time you start up the system). I also ensured that all the files contained the correct number of elements based on the number of users and items found in the database. Model fitting is an iterative process which terminates when the data converges. I ensured that the data converged to within 1% before terminating the iterative process. This makes our probability model extremely accurate.

Testing of the ACF algorithm was performed in isolation in batch mode before integrating it with the web server. This was done by writing my own main program which mimics the interaction of the web server with the code. Dummy profiles were created and optimal queries were generated and lists of recommendations were generated based on these profiles. The correctness of the results were verified by manually calculating the expected value of information (EVOI) for each item at every step to make sure the calculations were being performed correctly.

3.2 Milestone & Description - *Integration and testing of web server, ACF logic layer, and database*

In order to maximize the effectiveness of the system, the Web Server must interact quickly with the application. This milestone is responsible for a smooth integration of all three components so they work together correctly and combine to compose a fully functional system. Due Feb 13, 2004.

Responsibility

Bernie Ma

Status at start of reporting period

Not Started.

Status at end of reporting period

Completed Jan 19, 2004.

Actions

I created an application programming interface (API) for Bernie's web server in order for him to easily access my code. Two public methods were written in the ACF logic layer for the web server to use. The first method was getNextOptimal() which takes a user's profile and returns the song ID for the next optimal query item. The second method written for the web server was getTopTen(), which takes a user's profile and returns a linked list of song ID's corresponding to the top ten recommendations for the user.

Testing & Verification of Progress

Worked together with Bernie when integrating my code with his web server. Spent much time debugging some minor problems with respect to calling my code and outputting data properly to the GUI. The rest of the testing was the responsibility of Bernie.

3.3 Milestone & Description - *Integration of ACF logic layer with MySQL database*

Enabling the Java code to successfully retrieve and update data to the database. Ensure that communication between the two components is smooth. Due Feb 13, 2004.

Responsibility

Cavan Yie

Status at start of reporting period

Not started.

Status at end of reporting period

Completed Feb 6, 2004.

Actions

I implemented a JDBC (Java Database Connector) connector in order for the logic layer to access ratings stored in the MySQL database. The model fitting module only required runtime access to the database. I removed dummy code that originally accessed test data from a data file, and added code to contact, authenticate, and read data from the database. Runtime access to the database is now seamless and data retrieval is extremely fast.

Decisions

The JDBC connector required a database driver to be installed on the same system that the Java code was running off. Since Bernie Ma already had a JDBC driver installed for his own work on the web server prior to me starting this milestone, we knew we did not require another installation of a JDBC driver. This made things much easier for the integration of the logic layer with the database.

Testing & Verification of Progress

The code was tested by running it in batch mode, and observing the messages that were output to the screen. Print statements were inserted in to the code. Various messages were used to signal successful access to the database, successful authentication, and successful retrieval of all rows in the ratings table found in the database. If any of these steps failed, an exception was caught (see Appendix C). This isolated many of the difficulties I had when trying to integrate my code with the database.

3.4 Milestone & Description - *Testing, comparison and implementation of alternative recommender algorithms*

Comparison of "active" versus "non-active" approaches to collaborative filtering. Analyze their performances and accuracy to real life usage.

Responsibility

Cavan Yie

Status at start of reporting period

Not started.

Status at end of reporting period

In the process of planning and strategizing the best way of conducting the experiments to compare ACF to CF.

Actions

Collaborative filtering is essentially identical to active collaborative filtering except that the EVOI is not calculated for each item to determine the next "optimal" query. CF does not care about extracting the maximum amount of information out of the user. A new piece of code was written to simulate CF. Rather than calculating the item corresponding to the maximum EVOI, the CF approach simply returns a random item with no regard to EVOI. The point is to see whether ACF generates better recommendations and faster as well compared to CF. This method best simulates the non-active approach to collaborative filtering.

We have strategized a good way to compare the two methods. The experiment starts with gathering a handful of *complete* user profiles where the user's have rated all items in the database. These ratings will be used on the ACF algorithm as well as the CF algorithm and the recommendations for each approach will be compared against those items already in the user's profile to see whether the recommended items were rated favourably or not.

Most of the complete user profiles have been gathered from family and friends.

Decisions

Once we have reached a sufficient number of experimental profiles, we will be begin conducting our experiments. We feel that 10 to 15 profiles should be adequate to make qualitative conclusions about the benefits or non-benefits of ACF.

4. Conclusion

The project is progressing smoothly and according to plan. We still need to compare ACF to CF to see which method performs better, and whether ACF is indeed a better approach when developing recommender systems. The oral presentation and the poster for the design fair are the other outstanding milestone's that are quickly approaching.

References

[1] B. Marlin. *Active Collaborative Filtering with Naive Bayes*. Unpublished. http://www.cs.toronto.edu/~marlin/research/research.shtml. 2002.

[2] Cox, B. 2003. "E-commerce News: E-commerce Industry Soaring." E-commerce-Guide Website. http://ecommerce.internet.com/news/news/article/0,,10375_1585731,00.html. Site accessed 24/9/03.

[3] Craig Boutilier, Richard S. Zemel, and Benjamin Marlin. *Active Collaborative Filtering*. In Proc. of the 19th Conference on Uncertainty in Artificial Intelligence. 2002.

[4] Craig Boutilier and Richard S. Zemel. *Online queries for collaborative filtering*. In AI-Stat, 2003.

[5] J. Ben Schafer, Joseph Konstan, John Riedl. *Recommender Systems in E-Commerce*. In Proc. of the 1st AC conference on Electronic commerce 1999.

[6] L. Guernsey. "Making Intelligence a Bit Less Artificial." The New York Times. http://www.cs.toronto.edu/~marlin/library/nyt_rec.pdf. Site accessed 01/8/2003.

Glossary

ACF - Active Collaborative Filtering:

A type of Collaborative Filtering Algorithm that applies principled methods from decision theory to obtain more informative user data and provide more accurate recommendations and predictions.

CF - Collaborative Filtering:

using a database about user preferences to predict additional topics or products a new user might like.

EVOI - Expected Value of Information:

It is the value associated to a query, given a rating profile that represents the expected improvement in decision quality one obtains after asking the user to rate that particular query item.

Appendix A - *Model-Fitting Java Module*

```
/*
 * Created on Nov 12, 2003
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author cavan
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */


import java.util.*;
import java.io.*;
import java.sql.*;


public class NaiveBayesModelFitting2 {

      public static final int MAX_RATING = 5;
      public static final int NUM_ATTITUDE_TYPES = 20;  // use 20
      //public final int num_items = 5;
      //public final int num_users = 10;
      public static int num_users;
      public static int num_items;

      //public final String WORKING_DIR = "C:\\My Documents\\ECE496-Design
Project\\java\\";
      //public final String WORKING_DIR = "/h/42/yie/DesignProject/";
      public final String WORKING_DIR = "/h/42/mab/";

      /*double[][] gamma = new double[NUM_ATTITUDE_TYPES][num_users];
      double[] theta = new double[NUM_ATTITUDE_TYPES];
      double[][][] beta = new double[MAX_RATING][num_items][NUM_ATTITUDE_TYPES];
      int[][] knownRatings = new int[num_users][num_items];*/
      public static double[][] gamma;
      public static double[] theta;
      public static double[][][] beta;
      public static int[][] knownRatings;


      /**
       *    This method accesses the MySQL DB to access known user ratings.
       *    Stores data in knownRatings matrix.
       */
      private void loadKnownRatings() {
            System.out.println("Loading known ratings from database...");

            int user, item;
            Connection conn = null;

            // initialize all values to zero
            for (int i=0;i<num_users;i++) {
                  for (int j=0;j<num_items;j++) {
```

```
                        knownRatings[i][j] = 0;
                  }
            }

            /* Connect to database */
            try {

               String userName = "root";
               String password = "andrew";
               //String url = "jdbc:mysql://mysql/bin/mysql";
               String url = "jdbc:mysql://localhost:3306/musicratings";
               //Class.forName ("com.mysql.jdbc.Driver").newInstance ();
               Class.forName ("org.gjt.mm.mysql.Driver").newInstance ();
               conn = DriverManager.getConnection (url, userName, password);
               System.out.println ("Database connection established");
            }
            catch (Exception e) {

               System.err.println ("Cannot connect to database server");

            }

            /* Load data */
            try {
                  Statement s = conn.createStatement ();
                  s.executeQuery ("SELECT * FROM ratings");

                  ResultSet rs = s.getResultSet ();
                  int userCount = 0;
                  int rating;

                  while (rs.next ()) {

                        for (int i=1;i<=num_items;i++) {

                              if (rs.getString(i+1).equals("N/A")) {

                                    knownRatings[userCount][i-1] = 0;
                              }
                              else {
                                    knownRatings[userCount][i-1] =
rs.getInt(i+1);
                              }
                              System.out.println("knownRatings
userCount="+userCount+" itemNum="+i+" =" + knownRatings[userCount][i-1]);
                        }

                        ++userCount;
                  }
                  rs.close ();
                  s.close ();
                  System.out.println (userCount + " rows were retrieved and
loaded");
            }
            catch (Exception e) {
                  System.err.println ("Could not load ratings from database");
            }

            /* Close connection to database */
            try {
```

```
            conn.close ();
            System.out.println ("Database connection terminated");
        }
        catch (Exception e) { /* ignore close errors */ }


        /*for (int i=0;i<num_users;i++) {
              for (int j=0;j<num_items;j++) {
                    System.out.println(knownRatings[i][j]);
              }
        }*/
    }


    /* Takes the table of ratings and performs model fitting */
    private void generateBetaTheta() {

        initializeBetaTheta();


        /*for (int i=0;i<NUM_ATTITUDE_TYPES;i++) {
              System.out.println(theta[i]);
        }
        System.out.println("\n");

        for (int i=0;i<num_items;i++) {
              for (int j=0;j<MAX_RATING;j++) {

                    System.out.println(beta[j][i][4]);
              }

        }*/


        /* model fit - use 20 iterations for now */
        for (int i=1;i<=40;i++) {
        System.out.println("iteration: " + i);
              // E-Step
              for (int z=0;z<NUM_ATTITUDE_TYPES;z++) {

                    for (int u=0;u<num_users;u++) {

                          gamma[z][u] = (theta[z] *
gammaNumerator(u,z))/gammaDenominator(u,z);
                          //System.out.println("z="+z+" u="+u+"->
"+gamma[z][u]+"\t");
                    }
              }

              // M-Step
              for (int z=0;z<NUM_ATTITUDE_TYPES;z++) {

                    theta[z] = thetaNumerator(z)/thetaDenominator(z);
                    //System.out.println("z="+z+"-> "+theta[z]+"\t");
              }

              for(int v=0;v<MAX_RATING;v++) {

                    for (int y=0;y<num_items;y++) {

                          for (int z=0;z<NUM_ATTITUDE_TYPES;z++) {
```

```
                                    beta[v][y][z] =
betaNumerator(v,y,z)/betaDenominator(y,z);
                                    //System.out.println("v="+v+" y="+y+"
z="+z+"-> "+beta[v][y][z]+"\t");
                                }
                            }
                        }

                }
        }

        private double gammaNumerator(int u, int z) {

                //double result=theta[z];
                double result=1;

                for (int y=0;y<num_items;y++) {

                        for (int v=0;v<MAX_RATING;v++) {

                                if (knownRatings[u][y] == v) {

                                        result = result * beta[v][y][z];
                                }
                        }
                }
                return result;

        }


        private double gammaDenominator(int u, int z) {

                double result=0;

                for (int aType=0;aType<NUM_ATTITUDE_TYPES;aType++) {

                        result += theta[aType] * gammaNumerator(u,aType);

                }
                return result;
        }


        private double thetaNumerator(int z) {

                double result=0;

                for (int u=0;u<num_users;u++) {

                        result += gamma[z][u];
                }
                return result;
        }

        private double thetaDenominator(int z) {

                double result=0;

                for (int aType=0;aType<NUM_ATTITUDE_TYPES;aType++) {
```

```
            result += thetaNumerator(aType);
      }
      return result;
}

private double betaNumerator(int v, int y, int z) {

      double result=0;

      for (int u=0;u<num_users;u++) {

            if (knownRatings[u][y] == v) {

                  result += gamma[z][u];
            }
      }
      return result + 1;
}

private double betaDenominator(int y, int z) {

      double result = 0;

      for (int rating=0;rating<MAX_RATING;rating++) {

            result += betaNumerator(rating,y,z);
      }
      /*if (result != 0)
            return result;
      else
            return 1;
      */
      return result + MAX_RATING;
}


/* This method initializes beta and theta with random values */
private void initializeBetaTheta() {

      Random generator = new Random();
      double r;

      /* Initialize THETA with random values */
      double total=0;
      for (int i=0;i<theta.length;i++) {
            r = generator.nextDouble();
            theta[i] = r;
            total += r;
      }
      /* normalize THETA so values sum to 1 */
      for (int j=0;j<theta.length;j++) {
            theta[j] = theta[j]/total;
      }

      /* Initialize BETA with random values */
      total=0;
      for (int z=0;z<NUM_ATTITUDE_TYPES;z++) {

            for (int y=0;y<num_items;y++) {
```

```
                        for (int v=0;v<MAX_RATING;v++) {

                              r = generator.nextDouble();
                              beta[v][y][z] = r;
                              total += r;
                        }
                        /* normalize the vector so its sum is 1 */
                        for (int n=0;n<MAX_RATING;n++) {

                              beta[n][y][z] = beta[n][y][z]/total;
                        }
                        total=0;
                  }

            }
            System.out.println("Done initializing beta and theta...");

      }



      /* Saves calculated Theta (attitude type distribution) and
         Beta (rating distribution for items given an attitude type) matrices
         to local disk for future use.
      */
      private void storeThetaAndBeta() throws IOException {

            File thetaFile = new File(WORKING_DIR + "theta.txt");

            if (!thetaFile.exists()) {

                  thetaFile.createNewFile();
            }

            BufferedWriter theta_out = new BufferedWriter(new
FileWriter(thetaFile));

            for (int i=0;i<theta.length;i++) {

                  theta_out.write(""+theta[i]+"\n");
            }

            theta_out.close();

            System.out.println("Created updated theta.txt...");

            File betaFile = new File(WORKING_DIR + "beta.txt");

            if (!betaFile.exists()) {

                  betaFile.createNewFile();
            }

            BufferedWriter beta_out = new BufferedWriter(new
FileWriter(betaFile));

            for (int z=0;z<NUM_ATTITUDE_TYPES;z++) {

                  for (int y=0;y<num_items;y++) {
```

```
                              for (int v=0;v<MAX_RATING;v++) {

                                    beta_out.write(""+beta[v][y][z]+"\n");
                              }
                        }

                  }
                  beta_out.close();
                  System.out.println("Created updated beta.txt...");

            }



            /* This program generates a Naive Bayes' probability model
               based on a set of user ratings on specific items
               (e.g. movies, songs, etc).  Stores data into data files
               on the local disk.
            */
            public static void main(String[] args) {

                  if (args.length != 2) {
                        System.out.println("Usage: java NaiveBayesModelFitting <#
users> <# items>");
                        return;
                  }
                  else {
                        num_users = Integer.parseInt(args[0]);
                        num_items = Integer.parseInt(args[1]);
                  }


                  gamma = new double[NUM_ATTITUDE_TYPES][num_users];
                  theta = new double[NUM_ATTITUDE_TYPES];
                  beta = new double[MAX_RATING][num_items][NUM_ATTITUDE_TYPES];
                  knownRatings = new int[num_users][num_items];


                  NaiveBayesModelFitting2 n = new NaiveBayesModelFitting2();

                  /* Load known ratings */
                  try {
                        n.loadKnownRatings();
                  }
                  catch (Exception e) {
                        return;
                  }


                  /* Model Fitting */
                  n.generateBetaTheta();

                  /* Store beta & theta probability distribution to disk */
                  try {
                        n.storeThetaAndBeta();
                  }
                  catch (IOException e) {}

            }
}
```

Appendix B - *Active Collaborative Filtering Logic Layer Java Module*

```
import java.util.*;
import java.io.*;
import java.lang.Integer;


public class ACF {

      public static final int MAX_RATING = 5;
      public static final int NUM_ATTITUDE_TYPES = 20;  // use 20

      //public static final String WORKING_DIR = "C:\\My Documents\\ECE496-
Design Project\\java\\";
      //public static final String WORKING_DIR = "/h/42/mab/jakarta-tomcat-
5.0.16/webapps/ACF/WEB-INF/classes";
      //public static final String WORKING_DIR = "/h/42/yie/DesignProject/";
      public static final String WORKING_DIR = "/h/42/mab/";

      public static int num_users = 63;  //63
      public static int num_items = 126; //126

      public static double[] theta;
      public static double[][][] beta;

      public static double[][] predictionVector;
      public static double[][] predictionVectorTemp;

      public static int[][] knownRatings;



      /**
       * Generates the prediction matrix from our beta and theta
       * probability distributions.
       */
      public static double[][] generatePredictionVector(int[] profile) {

            double[][] pVector = new double[num_items][MAX_RATING];

            double entry=0;
            double[] theta_prime = new double[NUM_ATTITUDE_TYPES];



            /*for (int i=1;i<127;i++) {
                  if (profile[i] != 0) {
                        System.out.println("item "+i+" is: " +profile[i]);
                  }
            }*/


            /* Calculate theta prime */
            for (int i=0;i<NUM_ATTITUDE_TYPES;i++) {

                  //theta_prime[i] = theta[i] *
(getNumerator(profile,i)/getDenominator(profile));
                  theta_prime[i] = theta[i] * getNumerator(profile,i);

            }
```

```
            /*for (int i=0;i<20;i++){
                  System.out.println("theta_prime["+i+"]="+theta_prime[i]);
            }*/

            for (int y=0;y<num_items;y++) {

                  for (int v=0;v<MAX_RATING;v++) {

                        for (int i=0;i<NUM_ATTITUDE_TYPES;i++) {

                              entry += beta[v][y][i] * theta_prime[i];
                        }

                        pVector[y][v] = entry;
                        //System.out.println("pVector: y="+y+" v="+v+"->
"+pVector[y][v]+"\t");
                        entry=0;
                  }
            }

            return pVector;
      }


      private static double getNumerator(int[] p, int z) {

            double result=1;

            for (int y=0;y<num_items;y++) {

                  for (int v=0;v<MAX_RATING;v++) {

                              if (p[y+1] == v+1) {
                                    result = result * beta[v][y][z];

      //System.out.println("beta["+v+"]["+y+"]["+z+"]="+beta[v][y][z]);

                              }

                  }
            }

            //System.out.println("getNumerator for z="+z+ " is: " + result);
            return result;
      }


      private static double getDenominator(int[] p) {

            double result=0;

            for (int i=0;i<NUM_ATTITUDE_TYPES;i++) {

                  result += theta[i] * getNumerator(p,i);
            }

            if (result != 0) {
                  //System.out.println("getDenom: " + result);
                  return result;
            }
            else {
```

```
                  //System.out.println("getDenom:1");
                  return 1;
            }
      }

      /* Returns the item number of the next optimal
         query.  This method assumes the profile index corresponds
         to the item number.  Index zero does not contain any info.
         BERNIE MUST DECLARE THE SIZE TO BE "num_items + 1"
      */
      public static int nextOptimal(int[] profile) {

            int returnItem = 0;

            double voi_orig;
            double evoi = 0;
            double evoi_max = 0;

            int[] profileCopy = new int[num_items+1];

            /* Find VOI for given profile (i.e. V(pi)) */
            //voi_orig = getVOI(profile);

            predictionVector = generatePredictionVector(profile);

            /*for (int i=1;i<127;i++) {
                  System.out.println("profile: i=" + i + " -> " + profile[i]);
            }*/

            /*for (int i=0;i<num_items;i++) {

                  for (int v=0;v<MAX_RATING;v++) {
                        System.out.println("predictionVector: item="+i+"
rating="+v+" -> "+predictionVector[i][v]);
                  }
            }*/


            /* Find item corresponding to maximum EVOI */
            double aug_voi = 0;
            for (int item=1;item<(num_items+1);item++) {

                  if (profile[item] == 0) {  // if profile rating contains a
zero, it means item was not rated.

                        /* copy contents of profile into auxilary array */
                        for (int i=1;i<(num_items+1);i++) {

                              profileCopy[i] = profile[i];
                        }

                        for (int rating=0;rating<MAX_RATING;rating++) {

                              aug_voi = getAugVOI(profileCopy, item, rating);
                              //System.out.println("aug_voi = " + aug_voi);
                              evoi += predictionVector[item-1][rating] *
aug_voi;
                              //System.out.println("evoi = " + evoi);
                        }
                        //evoi = evoi - voi_orig;
```

```
                  System.out.println("Item#"+item+" EvOI=" + evoi);

                  if (evoi > evoi_max) {
                        evoi_max = evoi;
                        returnItem = item;
                  }

                  evoi=0;
            }
            //System.out.println(profile[item]);

      }
      System.out.println("nextOptimal returned: " + returnItem);
      return returnItem;

}

/* This method finds the VOI for an augmented profile (i.e. V(pi_n+1)) */
private static double getAugVOI(int[] profile, int item, int rating) {

      //Hashtable h_temp = profile;
      int[] profile_aug = new int[num_items+1];
      double voi_augmented = 0;
      double voi_temp;
      double[][] pv = new double[num_items][MAX_RATING];

      profile_aug = profile;

      /* Augment profile */
      profile_aug[item] = rating+1;

      /* Generate new prediction vector for augmented profile */
      pv = generatePredictionVector(profile_aug);

      /* Get VOI for this augmented profile */
      voi_augmented = getVOI(profile_aug,pv);

      return voi_augmented;

}


/* This method returns the value of information contained
   in a rating profile.
 */
private static double getVOI(int[] profile, double[][] pv) {

      double maxValue = 0;
      double voi = 0;
      for (int item=0;item<num_items;item++) {

            if (profile[item+1] == 0) {

                  for (int rating=0;rating<MAX_RATING;rating++) {

                        voi += (rating+1) * pv[item][rating];
                  }

                  if (voi > maxValue) {
                        maxValue = voi;
                  }
```

```
                }
                voi = 0;
            }

            return maxValue;
        }


        /*
         * Returns list of recommendations.  *** NOTE: Using an array instead of
LL could improve performance
         */
         public static LinkedList topTen(int[] profile) {

            LinkedList recList = new LinkedList();
            Integer itemNum = null;
            double[][] pv = new double[num_items][MAX_RATING];

            pv = generatePredictionVector(profile);

            for (int i=0;i<10;i++) {

                  //itemNum = new Integer(getRecommendedItem(profile, pv,
recList));
                  itemNum = new Integer(getRecommendedItem(profile, pv,
recList));
                  recList.add(itemNum);
                  System.out.println("topTen item: " + itemNum);
            }

            return recList;
         }


         /**
          *   Returns item number with highest VOI that has not been
          *   chosen for recommendation.
          */
         private static int getRecommendedItem(int[] profile, double[][] pv,
LinkedList list) {

            int returnItem = 0;
            double maxValue = 0;
            double voi = 0;

             for (int item=0;item<num_items;item++) {

                  if (profile[item+1] == 0) {

                        if (!list.contains(new Integer(item+1))) {  // item
can't be in profile either! fix this.

                              for (int rating=0;rating<MAX_RATING;rating++) {

                                    voi += (rating+1) * pv[item][rating];
                              }

                              if (voi > maxValue) {
                                    maxValue = voi;
                                    returnItem = item+1;
                              }
```

```
                        }
                  }
                  voi = 0;
            }

            return returnItem;

       }



      /* This method loads theta from a data file already stored on disk */
      private static void loadTheta() throws IOException, FileNotFoundException
{


            BufferedReader br = new BufferedReader(new FileReader(WORKING_DIR +
"theta.txt"));
            String thisLine = null;
            int i=0;
            while ((thisLine = br.readLine()) != null) {

                  theta[i] = Double.parseDouble(thisLine);
                  //+
                  //System.out.println("theta: " + theta[i]);
                  i++;
            }
            br.close();
            System.out.println("Theta.txt successfully loaded.");

      }

      /* This method loads beta from a data file already stored on disk */
      private static void loadBeta() throws IOException, FileNotFoundException {


            BufferedReader br = new BufferedReader(new FileReader(WORKING_DIR +
"beta.txt"));
            String thisLine = null;


            for (int z=0;z<NUM_ATTITUDE_TYPES;z++) {

                  for (int y=0;y<num_items;y++) {

                        for (int v=0;v<MAX_RATING;v++) {

                              beta[v][y][z] = Double.parseDouble(br.readLine());
                              //System.out.println("beta: v="+v+" y="+y+"
z="+z+"-> "+beta[v][y][z]);
                        }
                  }
            }
            br.close();
            System.out.println("Beta.txt successfully loaded.");

      }

      public static void init() {

            System.out.println("num_items: " + num_items);
```

```
            System.out.println("num_users: " + num_users);

            predictionVector = new double[num_items][MAX_RATING];
            predictionVectorTemp = new double[num_items][MAX_RATING];

            theta = new double[NUM_ATTITUDE_TYPES];
            beta = new double[MAX_RATING][num_items][NUM_ATTITUDE_TYPES];
            knownRatings = new int[num_users][num_items];

            try {

                  loadTheta();
                  loadBeta();
            }
            catch (FileNotFoundException f) {

                  System.out.println("Ensure that the probability model fitting
has been performed before running this application.");

            }
            catch (IOException e) {

                  System.err.println("Error: " + e);
            }

      }


      /* Main program */
      public static void main(String[] args) {

            if (args.length != 2) {
                  System.out.println("Usage: java ActiveCollaborativeFiltering
<# users> <# items>");
                  return;
            }
            else {
                  num_users = Integer.parseInt(args[0]);
                  num_items = Integer.parseInt(args[1]);
            }

            predictionVector = new double[num_items][MAX_RATING];
            predictionVectorTemp = new double[num_items][MAX_RATING];

            theta = new double[NUM_ATTITUDE_TYPES];
            beta = new double[MAX_RATING][num_items][NUM_ATTITUDE_TYPES];
            knownRatings = new int[num_users][num_items];


            //ActiveCollaborativeFiltering3 acf = new
ActiveCollaborativeFiltering3();
            ACF acf = new ACF();


            // Load stored beta & theta from data files
            try {

                  loadTheta();
                  loadBeta();
            }
            catch (FileNotFoundException f) {
```

```
                  System.out.println("Ensure that the probability model fitting
has been performed before running this application.");
                  return;

            }
            catch (IOException e) {

                  System.err.println("Error: " + e);
                  return;
            }

            int newItem;

            int[] p = new int[num_items+1];
            p[15] = 2;
            p[39] = 3;
            p[45] = 5;
            p[61] = 5;
            p[86] = 5;

            for (int i=1;i<127;i++) {
                  if (p[i] != 0) {
                        System.out.println("item "+i+" is: " +p[i]);
                  }
            }

            // Generate Prediction Matrix
            System.out.println("Generating prediction matrix from main()...");

            predictionVector = acf.generatePredictionVector(p);



            System.out.println("Successfully generated prediction matrix from
main()");

            //newItem = acf.nextOptimal(p);
            //System.out.println("\nNext Optimal is: " + newItem);

            //LinkedList list = null;

            //list = topTen(p);

            /*for (int i=0;i<10;i++) {
                  System.out.println("#"+(i+1)+"-> item
"+((Integer)(list.get(i))).intValue());
            }*/

      }

      /*
      //Bernie's Main
      public static void main(String[] args) {

            init();
            int[] sendRatings = new int[127];

            int Optimal;
            Integer temp1;
            int temp2;
```

```
            LinkedList list;



            for(int i=1 ; i<127 ; i++)
            {
                  sendRatings[i] = 0;
                  //System.out.println(returnRatings[i]);
            }

            LinkedList info = new LinkedList();
            list = topTen(sendRatings);

            Optimal = nextOptimal(sendRatings);

            System.out.println("TopTen:");
            for(int i=0; i<10 ; i++)
            {
                  temp1 = (Integer)list.get(i);
                  temp2 = temp1.intValue();
                  System.out.println(temp2);
            }
            System.out.println("nextOptimal:");

            System.out.println(Optimal);


      }
      */

}
```

Appendix C – Original Milestones Chart

	Description	Assigned To	Start Date	End Date
1.	**Clear definition of project objectives, methodologies and software and hardware components that will be used:** Collection and research of resources and papers that will be used for project. Obtain computer resources from Professor Zemel and obtain access priviledges to Artificial Intelligence labs.	Andrew	09/01/2003	10/17/2003
2.	**Implement Active Collaborative Filtering Methodology in Java.** The back-end of the system will be coded in Java. This includes the calculations of EVOI for queries, model fitting for the probability model, and computing rating predictions. Its main purpose is to return a set of recommendations to a user given a database of user ratings.	Cavan	10/17/2003	11/21/2003
3.	**Database Implementation in MySQL:** MySQL database will be created to store music information. Decisions on the music information to be displayed, types and categories of music to be used(10 genres), and music sample format (mp3) and size (~100 songs) will be made based on research of related sites and considerations such as transfer speed.	Andrew	10/17/2003	11/07/2003
4.	**Webserver / GUI Setup using Jakarta Tomcat, Java Server Pages, Apache Server.** A user friendly and informative interface will be	Bernard	10/17/2003	11/14/2003

	designed and implemented. The GUI will provide features such as information about the music to be rated (mp3 sound clips, artist information etc.).			
5.	**XML structure definition for music, users and ratings:** A Document Type Definition structure will be created for use in the transfer of music information between the web server, application and database modules, and ultimately to the user. This step requires an understanding and coordination with the three main modules to ensure optimal compatibility.	Andrew	10/31/2003	11/17/2003
6.	**Integration of web server and database:** Interface for modular database component will have a focus on easy and fast access with webserver. Testing of speed of data retrieval will be looked at and optimizations and modifications to both components will be considered if necessary.	Andrew	11/14/2003	12/19/2003
7.	**Integration of web server and application.** In order to maximize the effectiveness of the system, the Web Server must interact quickly will the application. The speed of use will be tested, and modifications to both components will be made if necessary. .	Bernard	11/21/2003	12/19/2003
8.	**Integration of application and database.** Enabling the Java code to successfully retrieve and update data to the database. Ensure	Cavan	01/04/2004	01/25/2004

	that communication between the two components is smooth.			
9.	**Testing, comparison and implementation of alternative recommender algorithms.** Comparison of "active" versus "non-active" approaches to collaborative filtering. Analyze their performances and accuracy to real life usage.	Cavan	01/26/2004	02/12/2004
10.	**Oral Presentation** Summarization of progress will be condensed into presentation format. Visuals and aids will be considered and created to provide an effective emphasis and attract audience. Rehearsal of presentation and analysis of strengths and weaknesses. Results of testing will be added in as tests are completed.	Andrew	02/13/2004	02/27/2004
11.	**Design Fair Poster Presentation** The poster will attempt to give an overall view of our design project while keeping in mind the audience will mostly be comprised of 3rd year ECE students. The results of our research will be displayed through the use of charts and graphs. A computer will also be available to provide a hands on demonstration of our system.	Bernard	02/27/2004	03/16/2004
12.	**Completion of group final report** Intregration of the documentation of all components which make up the design project including diagrams, figures, references.	Cavan	03/16/2004	08/04/2004

Appendix D – Modified Milestones Chart

	Description	Assigned To	Start Date	End Date
1.	**Clear definition of project objectives, methodologies and software and hardware components that will be used:** Collection and research of resources and papers that will be used for project. Obtain computer resources from Professor Zemel and obtain access priviledges to Artificial Intelligence labs.	Andrew	09/01/2003	10/17/2003
2.	**Implement Active Collaborative Filtering Methodology in Java.** The back-end of the system will be coded in Java. This includes the calculations of EVOI for queries, model fitting for the probability model, and computing rating predictions. Its main purpose is to return a set of recommendations to a user given a database of user ratings.	Cavan	10/17/2003	01/21/2004
3.	**Database Implementation in MySQL:** MySQL database will be created to store music information. Decisions on the music information to be displayed, types and categories of music to be used(10 genres), and music sample format (mp3) and size (~100 songs) will be made based on research of related sites and considerations such as transfer speed.	Andrew	10/17/2003	11/07/2003
4.	**Webserver / GUI Setup using Jakarta Tomcat, Java Server Pages, Apache Server.**	Bernard	10/17/2003	12/13/2003

	A user friendly and informative interface will be designed and implemented. The GUI will provide features such as information about the music to be rated (mp3 sound clips, artist information etc.).			
5.	**Creation of Website to gather User Ratings using PHP, MySQL and HTML** Website created and tested to gather appromixately 75 user ratings and other information such as age, sex and mood for songs in the database. Information will be gathered and stored in multiple MySQL database tables to be used by the recommender system.	Andrew	11/15/2003	01/17/2004
6.	**Integration and testing of web server and application and database.** In order to maximize the effectiveness of the system, the Web Server must interact quickly with the application. Interface for modular database component will have a focus on easy and fast access with webserver. .	Bernard	12/14/2003	01/24/2004
7.	**General Testing of web server / application / database functionality:** As integration of web server, application and database progresses, general tests of the system will be conducted to determine boundary exceptions, broken links, correctness of algorithm implementation, and ease of communication and speed between modules.	Andrew	1/17/2003	02/13/2003
8.	**Integration and testing of application and database.** Enabling the Java code to	Cavan	01/04/2004	02/13/2004

	successfully retrieve and update data to the database. Ensure that communication between the two components is smooth.			
9.	**Coordination of Research Schedule.** Decision on two methods of testing will be decided, numbers of test subjects to be used, and types of measuring scales to be used to determine accuracy and usefulness of recommendations.	Andrew	01/30/2004	02/13/2004
10.	**Testing, comparison and implementation of alternative recommender algorithms.** Comparison of "active" versus "non-active" approaches to collaborative filtering. Analyze their performances and accuracy to real life usage.	Cavan	02/13/2004	05/03/2004
11.	**Oral Presentation** Summarization of progress will be condensed into presentation format. Visuals and aids will be considered and created to provide an effective emphasis and attract audience. Rehearsal of presentation and analysis of strengths and weaknesses. Results of testing will be added in as tests are completed.	Andrew	02/13/2004	03/26/2004
12.	**Design Fair Poster Presentation** The poster will attempt to give an overall view of our design project while keeping in mind the audience will mostly be comprised of 3rd year ECE students. The results of our research will be displayed through the use of charts and graphs. A computer will also be available to provide a hands	Bernard	02/27/2004	03/16/2004

	on demonstration of our system.			
13.	**Completion of group final report** Intregration of the documentation of all components which make up the design project including diagrams, figures, references.	Cavan	03/16/2004	04/08/2004