Programming Assignment 5 Software Design Document

CS2300 Section 3 Spring 2023

Cameron Johnson

Project Description

Part A Description: This program takes values from a file and places them in a matrix, provided that the matrix is a stochastic matrix and contains no negative numbers. The program then applies the power algorithm on this matrix, returning the dominant eigenvector of the matrix. Each eigenvalue in the eigenvector represents a webpage. These webpages are then ranked based on their associated eigenvalue.

Part B Description: This program uses linear binary classification to classify whether or not a user would like a series of unwatched movies based on trained weights from a training file. The program takes a random set of weights and, using features from the training file, takes the dot product between the set of weights and set of features. It then uses this info with the label of the already watched movie to update the weights array for the next sample. After convergence is met, the trained weight array is then dotted with features from an unwatched movie, and will return if the user would like or dislike that movie based on the features it has.

Approach

In part A, the only complex math I needed to use was a dot product, so I made a method for that. I used this same method in Part B, however I modified it to produce a 1 if the dot product was greater than or equal to 0, and a 0 if the dot product was less than 0.

Another piece of math I used in part A was checking if the absolute value of 1 minus the sum of the values in the row/column was less than the specified tolerance. I did this as, for example, a row or column with 3 values of ⅓ sums to 1 in our eyes using fractions, however the computer sees this as 0.99, and wouldn't properly recognize the matrix as a valid stochastic matrix. In my program, I made it so that as long as the row/column of the matrix summed to either 0.9 or 1, it recognized the matrix as valid. I would've used a smaller tolerance, but my program produced errors with any tolerance smaller than 0.1.

The if statement for this tolerance check is as follows:

```
If(Math.abs(1 - rowSum) < tolerance || Math.abs(1 - columnSum) < tolerance) {
    }
```

The rest of the math in part A was done through the power algorithm.

For part B, the weights for the training file were initialized randomly through Math.random(). The only other calculation of note is the dot product mentioned in the first part of my approach.

Detailed Design

Programming Language

I used Java for this assignment. There's no specific reason as to why Java was chosen other than it's my most comfortable language.

Modules

// Description of the modules and their inputs and outputs will be indicated by the double forward
// slash in front
Method names and their parameters will be in bold

Part A

// Method to print to a file
// Inputs: eigenvector, vector of pages ranked
// Output: File with printed outputs
printToFile(double[] eigenvector, int[] rankedPages);

// Method to rank the webpages
// Input: eigenvector
// Output: vector of the ranked pages
rankPages(double[] eigenvector);

// Method to take the values from the file and put them in a matrix
// Input: input file
// Output: returns a matrix with values from file
valuesToMatrix(File inputFile);

// Method to validate the matrix
// Inputs: matrix
// Output: true/false based on validity
isValidMatrix(double[][] matrixToCheck);

// Method to carry out the power algorithm on the matrix
// Inputs: matrix, empty array for eigenvector
// Output: updates eigenvector in main, returns a boolean if algorithm was successful
powerAlgorithm(double[][] matrix, double[] eigenvector);

// Method to calculate the y vector
// Inputs: matrix A, vector r
// Output: returns y vector
calculateY(double[][] A, double[] r);

// Method to find the largest value in the y vector
// Inputs: y vector
// Output: returns the largest y value
largestYValue(double[] y);

// Method to carry out the dot product
// Inputs: two vectors
// Output: the dot product of the two vectors
dotProduct(double[] row, double[] r);

Part B

// Method to validate the training file. The training file is validated based on if the label column only contains
// 1s and 0s, as that was the only way I could see a training file being tested for validation.
// Input: training file
// Output: true/false based on validity
validateTrainingFile(File inputFile);

// Method to find the number of samples in the training file. This counts the number of rows (samples) the file provides.
// Input: training file
// Output: number of rows in the file
findNumSamples(File inputFile);

// Method that counts the number of features per sample in the training file.
// Input: training file
// Output: number of features per sample
countFeatsOfFile(File inputFile);

// Method that counts the number of features provided in each sample of the unclassified file
// Input: file with unclassified features

// Output: number of unclassified features
countTestFeats(File inputFile);

// Method that reads the first column from the file and places the values into a labels array
// Input: training file, labels array
// Output: no return, updates labels array wherever called from
readLabels(File inputFile, double[] labels);

// Method that reads the features provided in each sample of the file, given what sample the algorithm is currently on
// Input: training file, features array, current sample number
// Output: no return, updates features array wherever called from
readFile(File inputFile, double[] features, int sample);
// Method that initializes weights randomly, given a max and a min.
// Input: size of the x vector
// Output: weight vector
initializeWeights(int size);

// Method that carries out the dot product between two vectors. Returns a 1 or 0
// if the dot product is greater than or equal to 0 or less than 0, respectively.
// Input: two vectors
// Output: 1 or 0
dotProduct(double[] vector1, double[] vector2);

// Method that carries out the "machine learning" called the Perceptron.
// Input: training file, features array, labels array
// Output: trained weight vector
perceptron(File inputFile, double[] features, double[] labels);

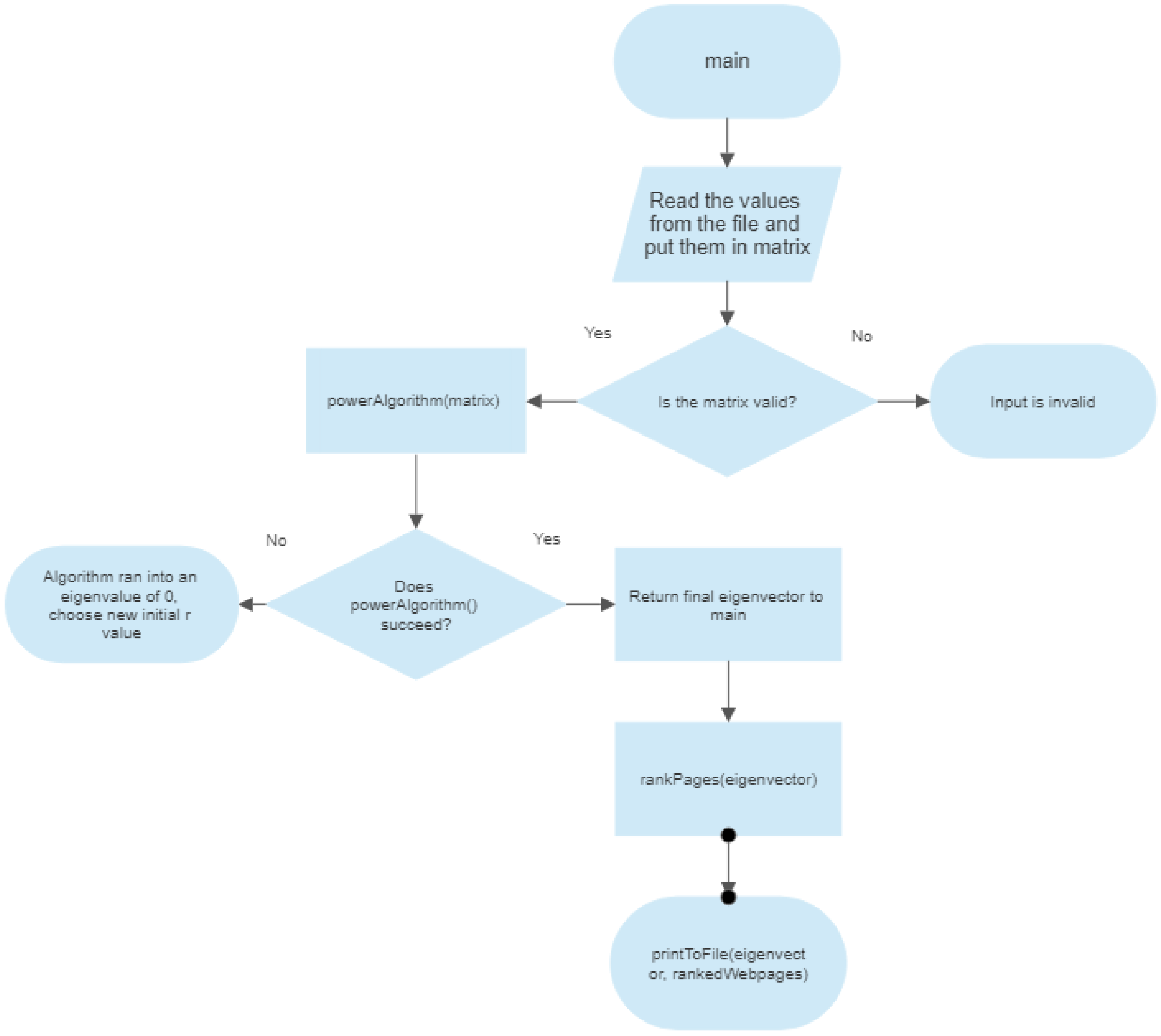
// Method that finds if the Perceptron has reached convergence.
// Input: difference between current weight vector and the one before it
// Output: true/false based on convergence
convergence(double[] difference);

// Method that classifies the features of the unclassified file using the trained weight vector
// Input: file with unclassified features, trained weight vector
// Output: no return, calls printToFile method
classifyFeatures(File inputFile, double[] weightVector);

// Method that prints the labels of the unwatched movies and the trained weight vector
// Input: final labels of the unwatched movies, trained weight vector
// Output: no return, prints both vectors to a file.
printToFile(double[] finalLabels, double[] weightVector);

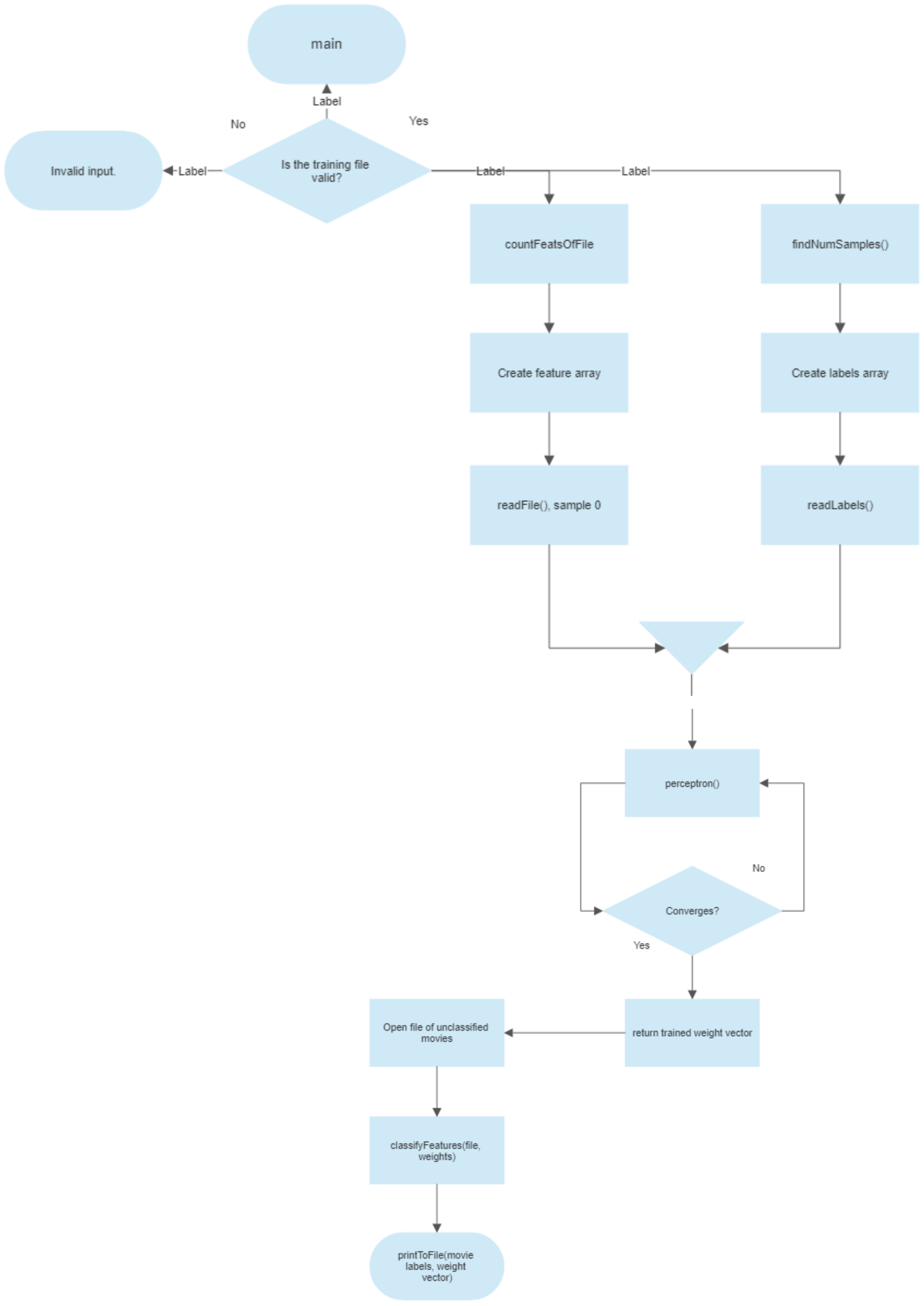
Flowcharts

Part A flowchart:



main
Read the values from the file and put them in matrix
Yes
No
Is the matrix valid?
Input is invalid
powerAlgorithm(matrix)
No
Yes
Algorithm ran into an eigenvalue of 0, choose new initial r value
Does powerAlgorithm() succeed?
Return final eigenvector to main
rankPages(eigenvector)
printToFile(eigenvector, rankedWebpages)

Part B flowchart:



(Had to split this into two pictures because I had to pay for it if I wanted one image. I do apologize for the small print.)

Key Data Structures

No key data structures were used in this assignment.

Test Description

For Part A, I will be using 2 stochastic matrices, 1 non-stochastic matrix, 1 non-square matrix and 1 square matrix with a negative value as tests

- One stochastic matrix will be from the book, the same stochastic matrix used in the example of the power algorithm
  - Input:

    ```
    0 ½ ⅓ ½
    0  0 ⅓ 0
    1 ½ 0  ½
    0 0  ⅓ 0
    ```

  - Output: (JCameron_PartA_bookExample.txt)

    ```
    [0.66, 0.33, 1, 0.33]
    3 1 2 4
    ```

- One self made stochastic matrix (the rows will add up to 1, but not the columns, opposite from the example stochastic matrix above)
  - Input:

    ```
    ½ 0  0  ½
    ¼ ¼ ¼ ¼
    0  ⅓ ⅓ ⅓
    0  0  1  0
    ```

  - Output: (JCameron_PartA_stochasticRows.txt)

    ```
    [1, 1, 0.99, 0.99]
    1 2 4 3
    ```
    (Likely a matter of rounding in the program)

- One self made non-stochastic matrix (square matrix but neither all of the columns or rows sum up to one)
  - Input:

    ```
    ½ 0  0  ½
    ¼ ¼ ⅓ ¼
    0  ¼ ⅓ ⅓
    0  0  1  1
    ```

  - Output: (Console)

    ```
    Input is invalid, please use a valid input.
    ```

- One non-square matrix (the columns/rows sum up to 1, however the matrix is not square.
    - Input: 5x4 stochastic matrix, each row sums to 1

      ```
      ½ 0  0  ½
      ¼ ¼ ¼ ¼
      0  ⅓ ⅓ ⅓
      0  0  1  0
      1  0  0  0
      ```

    - Output: (Console)

      ```
      Input is invalid, please use a valid input.
      ```

- One square matrix with a negative number (each row sums to 1)
    - Input:

      ```
      1 0 0  0
      1 0 -1 1
      0 1 0  0
      0 0 0  1
      ```

    - Output: (Console)

      ```
      Input is invalid, please use a valid input.
      ```

For Part B, I tested the given training file and unclassified file with “exaggerated” values to see if the output was expected. It does look like the output is expected, however I still don’t fully understand the process other than that the training file influences the classification of the movies by training the weighted values.