<Page Rank and Linear Classifier> Software Design Document

CS2300 Section 4 Fall 2021

<Brodie Crawford>

Project Description

In this project, I will be working with weighted page links for different webpages. These will be given to me in an input file of n x n links, where n is the number of webpages. I will also work with labels and feature sets in two different input files to predict the labels of the feature sets that are not classified. From the page links in part A, the power algorithm can be used to find the dominant eigenvector by way of iteration until the eigenvalues are close enough to each other. This means that the current eigenvalue subtracted from the previous eigenvalue will be smaller than the tolerance, which is preset and a very small number. Once the dominant eigenvector is found with all of its components, I will use the components to order their indeces from greatest to least component value. Part B will involve using the first file to ‘train’ a program to classify features based on a set of features that are already classified. Using another iterative algorithm called the perceptron algorithm, I will take the feature sets and their corresponding labels, and then calculate error based on which label the computer gives the feature set compared to the actual label of the feature set. Labels can either be 1 or 0. After coming up with a set of weights from the algorithm, I can then use the weights to come up with labels for the unclassified feature sets in the second file.

Approach

In order to create a page rank algorithm and a linear classifier, I will need three input files and two output files. For part A, I will read the input matrix into a 2D array of double values and then check if it is stochastic (all the columns add up to 1), and that the values in the matrix are all non-negative. Then, I will read the matrix into an algorithm that will keep track of the eigenvalues from the last two iterations, the current eigenvector, the tolerance desired for convergence, and the scale value to multiply the current eigenvector by each time. This power algorithm will cover many iterations, each of which will multiply the matrix by the current eigenvector to create a new eigenvector, save the value of the previous eigenvector, find the largest value in the current eigenvector to scale it, and then scale the eigenvector for the next iteration.

For the linear classifier in part B, I will read a few feature sets into a 2D array and a list of labels corresponding to them into a regular array containing the correct labels for each feature set, with 1’s as the element at the 0th index of each feature set for bias. Then, I will use the labels and feature sets to create a set of weights using an iterative method. The algorithm will have a tolerance as before, an error value, a variable to hold the dot product of the weights and the current feature set, and a weights vector stored in an array. During each iteration, I will take the dot product variable, compare it to 0, determine the guess for the program’s labeling system, take the error from it, and use it to calculate the updated set of weights. Then, I will use the final set of weights to calculate the labels of the feature sets in the second file, for which the labels are unknown. Finally, I will print the labels of the unclassified feature sets on the first line of the output file, along with the weights on the second line.

Detailed Design

Programming Language Used: Java

Editor: Eclipse

**Programming Language Description:**

Java makes it easy to read and write to files with large amounts of data, because no memory allocation is needed like for any of the C languages, and it has predefined structues like Scanner, BufferedWriter, ArrayList, and HashMap that are easy to use.

**bcrawf\_partA modules:**

public static boolean isStochasticAndNonNegative: Makes sure all the columns add up to 1 and that the matrix values are all non-negative

public static String powerAlgorithm: Does the main work of the program, iterating through and calculating the dominant eigenvector of the matrix, as well as writing output to the file

public static double[] multiplyMatrixByVector: Multiplies a matrix by a vector and returns a vector

public static double largestValueInVector: returns the largest value in the given vector

public static double[] findPageRanks: returns the indeces of the components of the eigenvector from greatest to least

**bcrawf\_partB modules:**

public static double[] perceptronAlgorithm: go through the feature sets using an iterative algorithm and return the weights calculated

public static double[] classify: classifies the test features based on what the algorithm learned using the weights

public static double dotProduct: returns the dot product of the two vectors given

**Module Flowchart:**

Diagram

Description automatically generated

**Key Data Structures:**

Int – stored counters

Double – stored dot product sums, tolerance values, and more

Double[] – stored everything that was a vector, including weights, labels and individual feature sets

Double[][] – stored matrices, including sets of features

HashMap – helped me attach indeces to eigenvector components for part A

DecimalFormat – made all my output look nice

String – helped me print out my output (BufferedWriter doesn’t have a way to write a double directly), including error messages

**Sample Files:**

Input for Part A:

0.400 0.300 0.200 0.100

0.300 0.200 0.100 0.400

0.200 0.100 0.400 0.300

0.100 0.400 0.300 0.200

This input will add up to 1 and is non-negative, and it will easily show me what the power algorithm is multiplying and scaling by when I run it because the values are varied enough, but they are also values with a low number of decimal points.

Input 1 for Part B:

1 25.00 25.00

1 -50.00 50.00

1 -25.00 50.00

0 25.00 -50.00

0 -25.00 -50.00

The input above will train the algorithm to label positive dot product results as 1, and negative dot product values as 0, except that it’s way easier to multiply the values together than for the training input file used for grading.

Input 2 for part B:

2.00 2.00

-4.00 4.00

-2.00 4.00

2.00 -4.00

-2.00 -4.00

This input will have the same label order as the last input, except with different numbers.