ECE 454 Assignment 3 Design Document

Nathan Leung/Sai Wai Maung
20411942/20419051
nhleung/swmaung

Part1 Design

Part 1 involves finding the genes that have the maximum expression value for each sample. The implementation uses only a mapper to complete the task. Each line goes into the mapper, so the sample number and the gene values are passed to the mapper. For each sample, a forloop is used to find the maximum expression value. Then a second forloop is applied to find all the genes with the maximum expression value. The key returned is the sample number and the value returned is an ArrayWritable object containing all the gene values that have the maximum expression value.

Part 2 Design

For part 2, the relation score for each gene is calculated by counting all the sample expression values greater than 0.5 and dividing that number by the total number of samples. In the Mapping stage, the gene and their expression values were mapped from the inputs of the samples. In the Shuffling stage, all outputs from the Mapping stage were shuffled into their own group based on the key of gene number. In the Reducing stage, a gene_counter and a sample_counter were created to count the number of gene that have an expression value greater than 0.5 and the total number of samples. From these two variables, the score of each gene was calculated in the reducing stage by dividing the number of samples with expression values greater than 0.5 by the total number of samples.

Part 3 Design

For part 3, the task is to find the similarity between all the samples, which is the dot product of pair of genes for each sample and then summed up. For this implementation, two mapreduce jobs were implemented. For the first job, the mapper maps the samples to genes. For example, sample_1,0.5,0.6 will be mapped to gene_1,sample_1-0.5 and gene_2,sample_1-0.6. this will allow for easier multiplication in the reducer. In the reducer, all the genes are grouped by the gene number. The value is parsed, and every sample's expression value is multiplied and written out to the context. The output is the pair of samples, followed by the product of their gene expression values. In the second job, the mapper just passes the sample pairs through to the reducer. In the reducer, it collects all the sample pairs and sums up all their expression values. The output key is the sample pair and the value is a FloatWritable object that holds the similarity value.

Before using two mapreduce jobs, we looked at having chainmapper or chainreducer, but found that we needed to reduce twice, once for the multiplication and one for the summation, so we needed two jobs.

Part 4A Design

The implementation is really simple for part 4A. Pig is just used to load the data into the UDF. Pig can easily find the maximum of the sample data, but to format to the way that is needed, where the gene numbers are included requires using a UDF. The UDF called FindMaxGeneVals, is exactly like the mapper in Part 1 implementation. It finds the max, then finds the genes that match the max and returns them.

Part 4B Design

In part 4B, the UDF was used to map the samples to the genes. A UDF that returns Data Bag was created. Inside the UDF, n tuples were added to Data Bag with key gene_i, where i start from 1 to n, and corresponding expression value for that sample, if the value is greater than 0.5, a value of 1 was added, else a value of 0 was added. The results returned from the UDP were group by their key. Finally, the AVG function was used to calculate the relation score of each gene. So all the ones summed up to the total number of samples with expression value greater than 0.5 and averages across the total number of samples for the gene.

Part 4C Design

For part 4C, the data was loaded twice into two objects. These object are converted into key, value pairs, where the key is the sample and the value is a bag of the expression value tuples. These two are crossed with each other to get their dot products. The sample pairs are then filtered by their sample number, which was extracted using STRSPLIT method. If the sample number of the first pair is less than the sample number of the second pair, then it will pass it to the UDF. In the UDF called MatrixMultAndSum, it multiplies all the sample's gene expression values by their respective pair from the other sample. Then it sums it up and outputs the sample pair and the similarity value calculated in the UDF. A difficulty coming up with this implementation was figuring a way to filter out sample pairs that would be redundant. The STRSPIT method above was one of the solutions we considered.