LARGE SCALE MUTUAL INFORMATION EXTRACTION USING APACHE HADOOP

1. Introduction

1.1. Aim:
The purpose of the project is to utilize MapReduce framework and Hadoop Cluster to generate word associations and learn how to use MapReduce for statistical text processing tasks. In this report, we describe how we have generated mutual information of all the bigram combinations of words across all the documents present in the dataset.

1.2. Why use map-reduce?
To generate all bi-gram combinations of words as mentioned in the project goal, we need to use a powerful framework like Map-reduce. This is due to the fact that computation of large scale mutual information is very expensive as it involves enumerating all the word pairs and collecting their counts of co-occurrences. The dataset, being large and having a large collection of text documents, increases the computational complexity quadratically. It is thus desirable to use MapReduce framework to solve the problem.

1.3. How is it useful?
Word associations or mutual information of words can be used in information retrieval and other applications to improve retrieval accuracy and to generate relationships. It can also be used in feature selection by maximizing the mutual information between joint distribution and other target variables. In the area of signal processing, mutual information can be used as a measure of similarity between two signals. Image fusion performance measures make use of mutual information in order to measure amount of information that the fused image contains about source images. Though our project focuses on mutual association between words, it can be extended to be applied to such wider applications.

2. Background

2.1. Statistics:
In probability theory and information theory, mutual information or trans-information is a measure of mutual dependence from information content perspective, i.e. the measure of amount of information contained by one about the other. Mutual information is usually used to
measure the correlation of two distributions or variables. In text information management and systems, people use mutual information to measure the correlation of two words.

2.2. Background Study/Explanation:
Suppose we have a collection of \( N \) documents.
For a word \( A \) in the collection, we use \( p(X_A) \), where \( X_A \) belongs to \( \{0, 1\} \), to represent the probability of whether \( A \) occurs \( (X_A=1) \) in one document or not \( (X_A=0) \).
If word \( A \) appears in \( N_A \) documents, then \( p(X_A=1) = N_A/N \) and \( p(X_A=0) = (N - N_A)/N \).
Similarly, we can define the probability \( p(X_B) \) for another word \( B \).
Besides, we also need to define the joint probability of word \( A \) and \( B \) as follows:
1. \( p(X_A=1, X_B=1) \): the probability of word \( A \) and word \( B \) occurring in one document. If there are \( N_{AB} \) documents containing both word \( A \) and \( B \) in the collection, then \( p(X_A=1, X_B=1) = N_{AB} / N \)
2. \( p(X_A=1, X_B=0) \): the probability of word \( A \) occurs in one document but \( B \) does not occur. It can be calculated as \( p(X_A=1, X_B=0) = (N_A - N_{AB}) / N \).

3. Implementation Details

The figure below describes the entire workflow of the project. Each part is described at length below.

![In a NutShell Diagram]

**Figure: Big Picture**
Step 1: Pre-Process Input
The first step is parsing of dataset and generation of Inverted Index across all the documents in the dataset under study. File-1 contains list of unique words with their positions or line number indicating their translation value.

File-1:
Word1
Word2
Word3

File-2:
Doc-ID<SPACE>Word-ID<SPACE>count of that word in that document

File-1 and File-2 are then processed to form inverted index and the format of output for this stage is:
WORD1 <TAB> COUNT_DOCUMENTSN <TAB> DOC1,DOC2,DOC3,DOC4...DOCN
WORD2 <TAB> COUNT_DOCUMENTSN <TAB> DOC1,DOC2,DOC3,DOC4...DOCN

Step 2: Enumerate all Bi-Gram Combinations
From the previous step, we get the values of Na and Nb, which is the number of documents which contains the words a or b.
We use MapReduce to compute pair-wise mutual information for every pair of words in the data set. Instead of doing a sequential scan of O(n^2), we pre-assigned a dedicated number of reducers, R, by invoking the method setNumReduceTasks(R) of the Job class in Apache Hadoop library. Once the number of reducers are available which we pre-set as 196, we adopted the matrix mapping algorithm1 to partition/load balance the whole bi-gram combination space among the 196 reducers available.

Map Logic:
A 14*14 matrix was constructed and for each tuple read from input, a random row and a random column is chosen and traversed. While the 28 different moves during traversal (14 for row and 14 for column), map outputs (key, value), where key is the number between (1,196) and the value is the input line. The Mapper aims to re-distribute and divide the whole space among the different reducers in this fashion.

Reduce Logic:
Each key in the range (1,196) receives a set of values of input tuples which basically is a subset from the original input list. The reducers finds out the combination for this subset of data, and for each such word pair, it generates the mutual information by adding the necessary smoothing functions to avoid log0 error. Each reducer writes its output to a separate file. At the end of the program, all the 196 files (one for each reducer) are merged from HDFS to local file system by using `-getmerge` function of hadoop.

The pseudo code for the approach we adopted is shown below:

![Pseudo code for Matrix mapping.](image)

Step 3: Pruning:
We have implemented the project both with/without pruning and the speed is considerably fast in both cases by using MapReduce framework for computation. There are cases where we might not need all the rare words (count less than 3 times) that occur in most of the documents (frequent in documents). They add lesser value to the documents than words that have a good count and occur more in one document that in the other. In these cases, we need not generate combinations for such words. Pruning option has been provided in the project to enable or disable pruning depending on the application it is used on.

Step 4: Smoothing:
We also need to do smoothing in the formulas for probability estimation in order to avoid the log0 problem (0 occurrences). For joint probability estimation, we assume that each of the four cases (corresponding to four different combinations of values of $X_a$ and $X_b$) gets 0.25 pseudo count, thus in total we introduced $0.25 \times 4 = 1$ pseudo count. We can then compute marginal probability based on the joint probability, i.e. $p(X_a=1) = p(X_a=1, X_b=0) + p(X_a=1, X_b=1)$.

For example, $p(X_a=1, X_b=1) = (N_{AB} + 0.25) / (1 + N)$ and $p(X_a=1) = (N_a + 0.5) / (1 + N)$. 
**Step 5: Normalization**
Most of the mutual information values we got were exponential in value. As an additional step, for the mutual information to be easily interpreted by the users, we had added normalization. In this step, we divide each mutual information value by a common value (since most exponents were \((\text{number} \times 10^{-8})\) we have divided by \(1 \times 10^{-8}\). We have given the normalization factor as an input, but in general we can divide it by the largest of first few values obtained if automation is required.

**Step 6: Removing duplicates**
The project generates bigram permutations of words with same mutual information. To avoid this, as an additional step, we have removed the permutations to give only combinations of words after the output is obtained. This forms the post-processing part of the project.

4. **Results/Evaluation:**

4.1. **Results:**

![Output from different reducers, some of them receiving values and few missing out on input values.](image)
Figure: The merged results; all bigram pairs with mutual information values.

4.2. Observation/Evaluation:

Mutual information is dimensionless and also a symmetric quantity, i.e. \( I(X;Y)=I(Y;X) \). If Step 5 in the implementation details is not performed, then we get bigram word associations like \((X,Y)\) and \((Y,X)\) with the same mutual information value. The mutual information also gives the reduction in uncertainty about one given knowledge of another:

<table>
<thead>
<tr>
<th>Mutual Information</th>
<th>Inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Large reduction in uncertainty</td>
</tr>
<tr>
<td>Low</td>
<td>Small reduction in uncertainty</td>
</tr>
<tr>
<td>Zero</td>
<td>Words are independent</td>
</tr>
</tbody>
</table>

Table 1: Mutual Information values and inference

When data is distributed across machines or deployed in the cloud, the use of framework to process the tasks in a distributed manner is a very useful approach considering big data used in today’s world is almost always distributed.
Complexity Analysis:

Figure 1: A Comparison of Speed

Figure 2: Space Usage
**Technical difficulties or challenges faced:**
Reducer Capacity Limits: Using multiple reducers and partitioning the combination space across the reducers helps in improving the data skew and load balancing. But even with multiple reducers pre-set, certain reducers exceeded the limit of values and consequently, crashed. The solution was to prune data in the map stage itself, so that per-reducer load will be decreased. We filtered out the words based on their Na, Nb values; i.e., if Na and Nb values are below threshold, they are conveniently omitted.

Reducers Availability: When using pre-set number of reducers in the CCT cluster, availability takes a hit which affects the overall performance. Since the cluster is used simultaneously by a lot of people, desired reducers might not be available and hence requires re-thinking the strategy. We had initially allocated 625 reducers, but after CCT gave performance issues, we fixed it at 196 reducers. This would mean more load per node, but solves the purpose albeit slower.

5. Discussion/Conclusion:

5.1 Summary:
MapReduce framework makes it easier to compute mutual information and can be used in many other statistical text processing tasks. It also facilitates the computation of such tasks in a distributed and large-scale manner.

5.2 Future work/extensions:
The project has been implemented in such a way that it can adapt to any dataset that can be parsed to produce the right format of input to the consecutive Hadoop - MapReduce stages. For example, if we take the million songs database, we can find mutual information between words in lyrics of two songs and utilize the results to derive conclusions about similarity, genre, theme, and so on of the songs. The project can be extended to image processing where instead of words a set of points along with their RGB values from images can be used to determine the mutual information between images.

References:
7] Papadimitriou, S. “DisCo: Distributed Co-clustering with Map-Reduce: A Case Study towards Petabyte-Scale End-to-End Mining”