CSE 494 Project A

Garrett Wolf

Introduction

The main purpose of this project was to give us a chance to experiment with various ranking algorithms. First, we were provided an interface with which we could perform Boolean queries on the underlying index. After running some of the sample queries using the Boolean Model search interface, we were asked to implement the Vector Space Model to rank the documents. This write-up will provide a background on the Vector Space Model, an explanation of the ranking algorithm that was implemented, and a brief evaluation of the algorithm's performance.

Vector Space Model Background

In this project, the Vector Space Model ranking algorithm was implemented using the cosine theta normalized similarity metric where term weights in the documents and queries are represented using a tf-idf weighing scheme. In this scheme, the following formulas are used in calculating the document term weights.

$$tf(i,j)=\frac{freq(i,j)}{\max(freq(i,j))}$$ = the normalized term frequency of term t_i in document d_j

$$idf(i)=\log\left(\frac{N}{n_i}\right)$$ = the normalized inverse document frequency of term t_i where N is the total number of documents and n_i is the number of documents containing the term t_i

$$wij=tf(i,j)*idf(i)$$ = weight of term t_i in document d_j

Queries entered by the user are also represented as a vector of term weights. The formula for the query term weights uses the *idf* formula from above from above but has a slightly different *tf* formula. For each term in the query, a weight is assigned using the following formulas and is assigned a zero weight otherwise.

$$tf(i,q)=0.5+0.5*\frac{freq(i,q)}{\max(freq(i,q))}$$ = the normalized term frequency of term t_i in query q

$$wiq=tf(i,q)*idf(i)$$ = weight of term t_i in query q

Using these formulas, we can define the cosine similarity formula as the sum of the document/query term weights divided by the product document and query vector norms.

$$Sim(q, d_j) = \frac{\sum wiq * wij}{|d_j| * |q|}$$ = the similarity between document d_j and query q

Vector Space Model Algorithm

Figure 1 provides definitions for variables and data structures used in this implementation of the Vector Space Model algorithm. These definitions will be used throughout the discussion of the algorithm shown in Figure 2.

Figure 1: Vector Space Model Algorithm Definitions

```
Let N = number of documents in the index
Let n_i = the document frequency for term T_i
Let idf_i = inverse document frequency of term T_i in the index
Let tf_ij = term frequency of term T_i in document D_j
Let tf_iq = term frequency of term T_i in query Q
Let wiq = weight of term T_i in the query Q
Let qNorm = the norm of the query vector Q
Let qTerms = set of terms and their weights in query Q
Let wij = weight of term T_i in the document D_j
Let sim[j] = similarity score for document D_j
Let maxFreq[j] = maximum term frequency of document D_j
Let maxTfq = maximum term frequency of query Q
Let docTotal[j] = total sum of squares for document D_j
Let norm[j] = normalization factor for document D_j
AddToMaxHeap(j, sim[j]) = adds document number and similarity score for
                          document D_j to a sorted max heap
```

Figure 2 shows an outline of the algorithm used in the implementation of the Vector Space Model. The algorithm consists of three main steps: computing the document norms, computing the query weights/norm, and computing the document/query similarity.

To save valuable time and improve performance from the user perspective, the normalization factors for each document are precomputed when the system first starts. To compute these normalization factors, the system iterates through each term in the index (*line 01*). The first step is to calculate the inverse document frequency (*line 02*) according to the formulas provided in the background section. Next, the system iterates through each document that contains the term (*line 03*). Remember that when calculating a document's normalized term frequency, the maximum term frequency over all terms in the document must be used. For this reason, the system tracks the maximum term frequency for each document in an array (*lines 04 & 05*). Part of the normalization factor is going to include the norm of the document where the norm is the square root of the sum of the document term weights squared. The sum of the document term weight squares are tracked for each document (*line 06*). After iterating through each term in the index, another loop begins which iterates through each document in the index (*line 07*). This iteration computes the normalization factor for each document (*line 08*). The factor is computed as one divided by the square root

of the sum of squares divided by the maximum term frequency. Here the sum of squares is divided by the maximum term frequency because when calculating the term weights, the maximum term frequency was not known and therefore the *tf* value was not normalized. All of this is equal to one over the norm of the document. This algorithm goes an extra step by dividing the entire value by the maximum term frequency to eliminate the need to store the maximum term frequency for later use in the document term weight calculation of the similarity function's quotient. Not storing the maximum term frequency for each document helps to save resources and can come in quite useful if the number of documents in the index becomes quite large. An example is provided below.

$$Sim(q,d_j)=\frac{\sum wiq*wij}{|d_j|*|q|}=\frac{\sum wiq*\left(\frac{freq(i,j)}{\max(freq(i,j))}*idf(i)\right)}{|d_j|*|q|}=\left(\frac{1}{\max(freq(i,j))}\right)*\frac{\sum wiq*\left(freq(i,j)*idf(i)\right)}{|d_j|*|q|}$$

The next main step in the algorithm is computing the query term weights and norm of the query vector. The query is represented as a vector with the terms and their corresponding weights. First, the system iterates through each term in the query (*line 09*). If the term is already in the vector, the system gets its associated frequency; otherwise it sets the frequency to zero (*lines 10 & 11*). The term frequency is then incremented by one (*line 12*) and if the incremented weight is greater than the query's maximum term frequency, it becomes the new maximum term frequency (*lines 13 & 14*). The vector is then updated with the frequency (*line 15*).

The final step of the algorithm computes the document/query similarity. For each term in the query (*line 16*) the query term weight is calculated using the formulas provided in the background section (*line 17*). Similar to the calculation of the document norms, the sum of the query term weight squares are tracked (*line 18*). For each document containing the term (*line 19*), the document term weight is calculated (*line 20*). The document and query term weights are then multiplied and added to the similarity score for the document (*line 21*). After completing the loop, the query normalization factor is computed as one divided by the square root of the sum of query term weight squares (*line 22*). Finally, for each document (*line 23*), the similarity score is normalized by multiplying it by the document and query normalization factors (*line 24*). The normalized similarity score and document number are then added to a sorted max heap (*line 25*) where they can then be displayed to the user in vector space model ranked order.

Figure 2: Vector Space Model Algorithm

```
    // precompute document norms
01  For each term Ti in the index
02        Set idfi = log(N/ni)
03        For each document Dj containing term Ti
04              If tfij > maxFreq[j]
05                    Set maxFreq [j] = tfij
06              Set docTotal[j] = docTotal[j] + (tfij * idfi)2
```

```
07  For each document Dj in the index
08        Set norm[j] = 1 / (docTotal[j] / maxFreq[j])^.5 / maxFreq[j]


    // compute query term frequencies
09  For each term Ti in the query Q
10        If qTerms contains Ti
11              Set tfiq = frequency of Ti in qTerms
12        Set tfiq = tfiq + 1
13        If tfiq > maxTfq
14              Set maxTfq = tfiq
15        Set add/update Ti and tfiq to qTerms


    // compute document/query similarity
16  For each term Ti in the query Q
17        Set wiq = (.5 + .5 * [freq(i,Q)/max(freq(i,Q)]) * log(N/ni)
18        Set qNorm = qNorm + wiq^2
19        For each document Dj containing Ti
20              Set wij = freq(i,j) * log(N/ni)
21              Set sim[j] = sim[j] + wiq * wij
22  Set qNorm = 1 / qNorm^.5
23  For each document Dj in the index
24        Set sim[j] = sim[j] * norm[j] * qNorm
25        AddToMaxHeap(j, sim[j])
```

As mentioned earlier, precomputing the normalization factors for each document results in a user perceived performance increase. The loop at line 01 requires at most *t+1* operations to loop through each term and set the idf. There are at most *dt+3* more operations performed during the inner for loop starting at line 03. At line 07 the norm is calculated for each document requiring at most *d+1* operations. Therefore, the running time for the segment that precomputes the document norms is O(*tdt*) where *t* is the number of terms in the index and *dt* is the maximum document frequency of the terms.

$$(t+1)(dt+3)+(d+1) = tdt + dt + 3t + d + 4 = \mathrm{O}(tdt)$$

The actual running time of the algorithm that executes after the user enters the query can be calculated as follows. At most *qt+6* operations for the loop starting at line 09. Starting at line 16, at most *tq+2* operations are needed. The loop starting on line 19 requires at most *qdt+2* operations. A single operation is needed for line 22 and *d+2* more are needed for the loop starting on line 23. Therefore, the running time of the algorithm that executes after the user enters their query is O(*tqqdt*) where *tq* is the number of terms in the query and *qdt* is the maximum document frequency of a term in the query.

$$(qt+6)+(tq+2)(qdt+2)+1+(d+2) = tqqdt + 2tq + 2qdt + qt + d + 13 = \mathrm{O}(tqqdt)$$

Conclusion

The Vector Space Model provides better results than the Boolean model because it allows documents that only partially match the user's query to be returned. In addition, by computing a similarity score, the documents can be ranked according to their closeness to the user's query. By normalizing the term frequency and inverse term frequency when computing the weight of a term and by dividing by the norm of the document and query vectors, you wind up with a much more accurate similarity score.