Module 17 Scribe

Bily Yuan, Matt Staton, Tyler Nicholas

August 10, 2016

Vector Space Model Using Distance

We begin with the Vector Space Model using Distance. We use matrices as described below:

- X = raw counts of each word
- x_{ij} = number of occurrences of word j in document i
- rows = documents
- columns = one word (type)
- Z = TF-IDF matrix derived from X
- q = query document with D entries

Once we have these matrices established, we wish to see how similar our query is to the documents in Z. To do this we must establish a distance function to determine which z_i our query is nearest to. We may initially think that distance between vectors would be a good idea. We see from Figure 1 below that due to magnitude of vectors, it will not always be a good metric for distance. Though z_1 and q are very similar, q is actually closer to z_2.

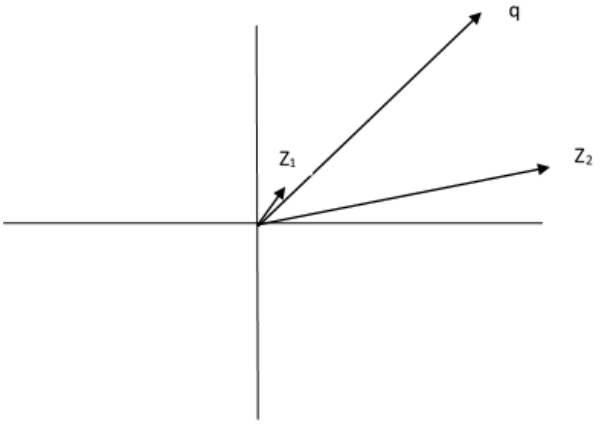

Figure 1:

This leads us to choose a better distance metric which is the cosine of the angle between vectors. The larger the value of the cosine between angles is, the smaller the angle between them will be. Thus the highest values of cosines will give us the document most similar to our query.

In multiple dimensions, if ϕ is an angle between two vectors v_1, v_2, then $\cos(\phi) = \frac{v_1 \cdot v_2}{|v_1| * |v_2|}$

There are two main Problems that can occur using this model:

1) Synonymns/ Polysemy
   An example of this would be with the query: "happy swine". Though this query is similar to a document talking about "excited javelinas" it will not match because the words are synonyms and not exact matches.
   Similary if your query is "Apple Computer", it would be shown to be similar to "Apple Tree" since the word Apple matches. They are however very different contextually giving you a false match.

2) Vectors are huge
   This becomes a problem with storage space and computational speed.

Latent Semantic Indexing - Creating the "V" Matrix

The first part of Latent Semantic Indexing involves creating a matrix whose rows are the types (i.e. "words") and whose columns are the principal components (i.e. the "context").

Let's start with matrix Z, whose rows are the types a and whose columns are the documents. The values of the matrix are the TF-IDF of word i in document j.

- z_i = document i (rows)
- z_{ij} = TF-IDF of word j in document i
- m = number of words
- n = number of documents

$$\mathbf{Z} = \begin{bmatrix} z_{1,1} & z_{1,2} & z_{1,n} \\ z_{2,1} & \cdots & z_{2,n} \\ \vdots & \vdots & \vdots \\ z_{m,1} & z_{m,2} & z_{m,n} \end{bmatrix}$$

As discussed in the previous section, calculating the distances between vectors within matrix Z can be time consuming. If a corpus contains many different types and documents, then the size of matrix X would be enormous. In addition, matrix Z does not handle semantics well: synonyms and polysemy can reduce the accuracy of distance-based models.

To both reduce the size of matrix Z and incorporate semantics into the distance-based model, the vector space could be redefined via *principal component analysis*.

- D is the number of words in the dictionary
- K is the number of principal components
- K will be significantly less than D

The V-Matrix could be represented as:

$$\mathbf{V} = \begin{bmatrix} v_{1,1} & v_{1,2} & v_{1,K} \\ v_{2,1} & \cdots & v_{2,K} \\ \vdots & \vdots & \vdots \\ v_{D,1} & v_{D,2} & v_{D,K} \end{bmatrix}$$

A component is represented by:

$$V_i = (v_{1,1} + v_{i,1} + \ldots + v_{D,1})$$

Each column (i.e. principal component) represents a "context" and the rows are the types. The values are the scores of a word in a component These scores can be used as a proxy for "relevance." For example, if a component is about computers, words such as "laptop" and "CPU" may have scores with high magnitudes.

One significant advantage of the V-Matrix over the Z-Matrix is how the V-Matrix takes context and semantics into consideration. Let's use the word "apple" as an example. Suppose the V-Matrix has many principal components, 3 of which are v_1, v_2 and v_3, and the contexts of these components are cooking, agriculture, and computers, respectively. For a particular corpus, the word "apple" could have relatively high scores for these 3 components. In addition, the word "window" could have a relatively high score in the "computer" component.

While the V-Matrix provides context information, it does not contain information about documents. If we had an article and wanted to find similar articles to it, the V-Matrix alone would not suffice. A matrix that has both document information and context information needs to be used in order to classify documents. This is the S-Matrix, which will be explained in a separate scribe note.

R-Script Review

The nyt_stores.R file has a number of new functions that the class may not have encountered. Some of these functions will be explained below.

which

The "which" function returns the name of the colums that matches "museum."

```
D = ncol(art_stories_DTM_TFIDF)
which(colnames(art_stories_DTM_TFIDF) == 'museum')
```

apply

The "apply" function performs a function specified in the argument across each column. It is similar to using a "for" loop across each column.

```
my.cosine = function(v1, v2) {
  sum(v1*v2) / {sqrt(sum(v1)^2) * sqrt(sum(v2^2))}
}

query.angles = apply(art_stories_DTM_TFIDF, 1, function(x) my_cosine(x, query_vec))
```

In the "query.angles" variable, the function "my.cosine" is applied to each column. The "function(x)" argument is where the desired function to be used is written.