CS405

Information Retrieval

Earlier we examined various methods to parse English text to generate semantic representations of the text. The resulting representations can then be used for items such as question answering, text summarization, filtering, or information retrieval. Of these, Information Retrieval (IR) is particularly desirable, especially as the WWW grows and more and more documents are available in electronic format. In this lecture we will examine other ways to retrieve information; some of these techniques overlap with artificial intelligence and machine learning.

A specialized form of natural language processing is information retrieval. Typically, shallow text processing is employed to retrieve the information we want. This is concerned with retrieving a textual document in accordance with a particular interest. For example, I might want to find all documents related to "Artificial Intelligence". Or I might want to get some measure of how similar a document is to the category of "Artificial Intelligence." Keyword methods are common and often effective, but anyone who has used search engines is familiar with the types of false hits that it returns (the material here doesn't solve this problem, but can alleviate it somewhat).

The Vector Space Document Model

In the vector space model, we are given a collection of documents and a query. The goal of the query is to retrieve relevant documents out of the collection. In this model, both the documents and the query are represented as vectors. Typically we will assume that the terms in the documents and the query are *independent*; i.e. one term has no influence on another.

In this model, there is a vocabulary of up to T terms. This vocabulary may or may not be controlled at design time. Each term is used to represent a document. This means that a document may be represented by a vector of up to T dimensions.

Associated with each document term is a weight. This weight may be binary (typically whether or not the term exists in the document), or it might be a floating point value between 0 and 1 where 0 indicates that this term is not relevant to the document, and 1 indicates that it is highly relevant to the document. A sample picture of these scenarios is depicted below:

Terms:	**Doc 1:**	**Doc 2:**	**Doc 3:**	**Query**
artificial	1	1	0.3	0.5
intelligence	1	1	0.6	0.9
information	0	0	0	0.01
retrieval	1	0	0.01	0.91
mock	1	1	0.99	0.99
kenrick	0	1	0.85	0.01

Once we have document and query vectors, the process of retrieving relevant documents is to compare two vectors together. Given the following terminology, we can compare vectors in several different ways:

Given vectors $X=(x1,x2,\ldots, x_t)$ and
$Y=(y1, y2,\ldots,y_t)$
Where:
x_i - weight of term i in the document and
y_i - weight of term i in the query

Additionally, for binary weights, let:
$|X|$ = number of 1s in the document and
$|Y|$ = number of 1s in query

The following are some methods for comparing vectors:

	For binary term vectors	For weighted term vectors
Inner product	$\lvert X \cap Y \rvert$	$\sum_{i=1}^{t} x_i y_i$
Dice coefficient	$\frac{2 \lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert}$	$\frac{2 \sum_{i=1}^{t} x_i y_i}{\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2}$

	For binary term vectors	For weighted term vectors
Cosine coefficient	$\frac{\lvert X \cap Y \rvert}{\lvert X \rvert^{1/2} \lvert Y \rvert^{1/2}}$	$\frac{\sum_{i=1}^{t} x_i y_i}{\sqrt{\sum_{i=1}^{t} y_i^2 \sum_{i=1}^{t} x_i^2}}$
Jaccard cofficient	$\frac{\lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert - \lvert X \cap Y \rvert}$	$\frac{\sum_{i=1}^{t} x_i y_i}{\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2 - \sum_{i=1}^{t} x_i y_i}$

There are several other comparison methods. These are only a couple! Each performs slightly differently depending upon the circumstances. The cosine, Dice, and Jaccard coefficients all normalize the relevancy values between 0 and 1.

Let's look at some simple examples of retrieval using these comparison methods. First, lets look at an example of applying the inner product to some binary term vectors:

t=5000

term	1	2	…	12	…	456	…	678	…	5000
Doc-1	0	1		0		1		0		0
Doc-2	1	1		1		0		1		1
...										
Doc-N	0	1		0		1		1		0
Query	1	1		0		0		1		0

Now, applying the inner product of the query to doc1, doc2, and docN gives us 1 for doc1, 2 for doc N, and 3 for doc 2. The ranked list is then document 3, N, and 1 with document 3 being the most relevant.

One more example using the inner product on real values:

Term	1	2	…	12	…456	…	678	…	5000
Doc-1	0	0.3		0	0.5		0		0
Doc-2	0.2	0.6		0.3	0		0.8		0.3
...									
Doc-N	0	0.2		0	0		0.6		0
Query	0.3	0.7		0	0		0.7		0

The relevance values in this case are calculated via:

For Doc-1 : 0.3*0 + 0.7*0.3 + 0.7*0 = 0.21
For Doc-2 : 0.3*0.2 + 0.7*0.6 + 0.7*0.8 = 1.04
For Doc-N : 0.3*0 + 0.7*0.2 + 0.7*0.6 = 0.56

If we use one of the other metrics we will get values normalized between 0 and 1.

Note that none of the metrics specify how terms may be related to one another. In the example of "Kenrick" and "Mock" as terms, the pair together means one thing, while the word "Mock" by itself may mean something else. Obviously, terms are not always independent. Despite this, we typically get fairly good performance by assuming conditional independence (this is the same issue as when we discussed Bayes Theorem with conditional independence).

Now that we can compare document and query vectors, the next question is, how do we come up with the weights to place into the vectors? In the vector space model the weight of a term typically depends on:

A measure of recurrence
A measure of term discrimination
A normalization factor

Weights via Term Frequency - Inverse Document Frequence (tf-idf)

In the information retrieval community, one popular form assigning weights for document indexing is term-frequency indexing coupled with the inverse-document frequency, which is referred to as *tf-idf* and was invented by Gerald Salton.

Tf-idf assumes that the frequency of occurrence of a term or keyword within a document is an indicator of how relevant that term is. However, if a term or keyword appears in many documents, then its predictive power is weak. For example, a common word such as "the" appears many times in one document, but appears in so many documents that it is a useless term that provides almost no information. These two terms may be combined by multiplying the term-frequency (*tf*) by 1/document-frequency (*idf* or inverse document frequency) to obtain a metric of relevancy for each term. By combining terms from a document to form a vector, queries can undergo a similar process and the document vector closest to the query vector is retrieved as the best match.

To express this process mathematically, the weight of a term *t* with respect to document *i* is described by:

$$weight(t_i) = tf(t_i) \times \log\left(\frac{N}{df(t)}\right)$$

Where: t_i = term t in document i
$tf(t_i)$ = Number of occurrences of term t in document i
df(t) = Number of documents containing term t
N = Total number of documents

Since the tf term is a better predictor of a terms value than how common the term is, often the inverse document frequency is scaled to de-emphasize large weights by taking the logarithm.

The example below shows the calculated tf-idf values for three sample documents. Listed below are the term frequencies found in the document:

Doc 1:	Doc 2:	Doc 3:
artificial 2	artificial 3	artificial 0
intelligence 0	intelligence 5	intelligence 5
information 12	information 5	information
retrieval 10	retrieval 0	0
mock 1	mock 3	retrieval 0
kenrick 2	kenrick 1	mock 0
the 35	the 56	kenrick 0
		the 42

From these frequencies we can construct tf-idf values for the terms in document 1:

Global Document Freq:		**Document 1 Freq:**		**TF-IDF Weights**
artificial	2	artificial	2	2*log(3/2)=0.35
intelligence	2	intelligence	0	0*log(3/2)=0
information	2	information	12	12*log(3/2)=2.11
retrieval	1	retrieval	10	10*log(3/1)=4.77
mock	2	mock	1	1*log(3/2)=0.17
kenrick	2	kenrick	2	2*log(3/2)=0.53
the	3	the	35	35*log(3/3)=0

These tf-idf weights would then be stored with each term and used in comparing one document to another.

Note that words are typically checked against a stop list. These words are ignored. With a good stop list, the words with a low idf value will already be thrown out, indicating that the idf term will not be the dominating factor. However, even if there are words missed by the stop list, these words will typically have a large global frequency, making the tf-idf weight small. Conversely, terms that are very specific to a document will have a

small idf and a large tf, making their weight value large. Overall, the process finds terms that are good *discriminators*.

One other item to note is that the terms do not necessarily need to be words from the document. The terms could be extracted features that are in some way related to a document. Consider "collaborative" filtering where a feature might be how many other people liked this document, or consider an intelligent extraction system that extrapolated higher-level features, e.g., noun phrases instead of just word terms.

By computing the tf-idf values for all words in a document (or say the top N words chosen from a dictionary for this domain) we can create a vector of word/tf-idf pairs that represents the document. Similarly, we can do the same thing for a query document.

Now we can compute the similarity of a query vector Q and a document vector V using one of the similarity metrics discussed earlier.

We can apply this same technique to computing the similarity between a document and that of an entire category. This is called a **classification system**. The similarity value is used by creating a category, C, that we have trained with all of the documents belonging to that category. This category can be thought of as a conglomerated vector of term tf-idf values. By computing the similarity of a document to the vector, we can get a measurement of how close a document is to the category. With a proper threshold, we can now categorize documents into a category if the similarity > threshold.

Consider the example below, where doc 1 and doc 2 are known to be in the category, and it has hence been "trained" using the frequencies of the two documents combined together.

Doc 1:
artificial 10
intelligence 20
salton 10
kenrick 2

Doc 2:
artificial 13
intelligence 13
salton 15
retrieval 11

	Category Freq:	**TF-IDF Weights**
artificial	23	0.33
intelligence	33	0.45
retrieval	11	1.00
kenrick	2	0.66
salton	25	0.83

We can now perform vector comparison of a document to the category to see if the document belongs in the category, for automatic classification. For example, consider a category of "Junk Email" and incoming documents are your new emails. Then, we could use this system to filter your junk email. This is a simple form of machine learning, and we will examine more methods in the future. Note that if the system ever performs incorrect classification, we can perform further training to update the frequencies to

hopefully correct the error. Similarly, we can perform "negative" training to decrease the frequency of terms that should perhaps not belong in the category vector.

Since the tf-idf information retrieval scheme is simple, it is one of the most popular methods used today. Additionally, the method provides good results. Depending on the data set, very high precision can be achieved in retrieving relevant articles or categories.

Measurement and Error

It is often useful to distinguish between the different types of errors that are made. To do this, we can construct a confusion matrix. A confusion matrix for a binary classification is shown below:

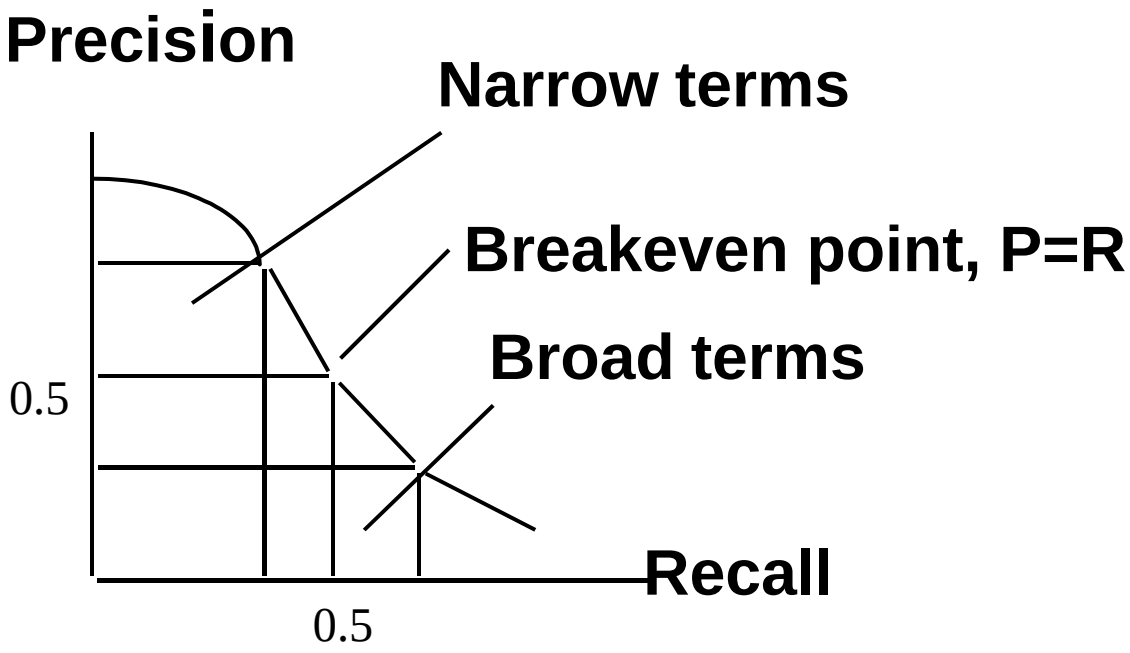
	Class Positive	**Class Negative**
Prediction +	# True Positive	# False Positive
Prediction -	# False Negative	# True Negative

Given this table, we can now compute various other metrics:

Accuracy = (TP + TN) / (TN + TP + FN + FP)
Precision = TP / TP + FP
Recall = TP / TP + FN
Neg Precision = TN / TF + FN
Neg Recall = TN / TN + FP

Precision and recall are common metrics for information retrieval. Precision measures how many items the classifier has been correct on, while Recall measure how many relevant items were classified. Both precision and recall have an inverse relationship. On one extreme, consider if we classified everything as being class positive. This means that there would be no false negatives (misses) but a lot of false positives. In this case, the recall will be a perfect 1, but precision will be low. We can get the opposite effect if we memorize our training cases; precision will be high, but we won't predict any new cases so we will likely miss many classifications. In this example, precision will be high but recall low. This is called overtraining and results on the classifier only being useful for specifically learned (memorized) cases, and not leaning toward more general rules that apply to new unseen cases.

The typical relationship looks something like this:


Precision
Narrow terms
Breakeven point, P=R
Broad terms
0.5
Recall
0.5