Lecture 18: Vector Space Approaches to classifying documents

Information Retrieval

When we start trying to classify large sets of documents, rather than sentences or small sets of words, there are some problems that need to be overcome with vector representations of documents. First, because in many applications the documents being retrieved are fairly large, they need to consider whether the fact that a word appears multiples times in a document is useful information. The second issue is that, while it is clear that some words are more salient than others in distinguishing documents, currently all words are weighted equally in the vector representation and affect the distance measure to the same extent. For instance, consider a set of documents D1,.., D10 and the occurrence of six words shown in table 1.

On the first problem, we need to consider how relevant it is that a word occurs multiple time sin a document, The general consensus in the field is that while the number of times a word occurs in relevant, its relevance decreases as the numbers get larger. Thus the first step taken is usually to reduce this impact. Typical measures include taking the square root of the count, or 1 plus the log of the count. Introducing some notation used in the information retrieval literature, for a word w_i and document d_j, the term frequency $tf_{i,j}$ is the number of times w_i appears in d_j. If $weight_{i,j}$ is the i'th value of the vector representation of document d_j, then we are suggesting that $weight_{i,j} = SQRT(tf_{i,j})$ or $weight_{i,j} = 1+\log(tf_{i,j})$ as values for the vectors. In the following we will use the latter.

The second problem is how informative a word is. We would, for instance, consider the fact that a document contains the word computer as providing more information than the fact that a document contains the word want. An obvious way that we might use to measure informativeness would be the entropy of the probability distribution $P(D \mid W=w_i)$ for each word. A high entropy word would tend to be evenly distributed among documents, whereas a low entropy word would occur in few documents. For instance, say we have 10 documents, and computer occurs once in two of them, and want occurs once in 8 of them. Then the probability distribution over the documents given computer would have two non-zero terms (.5 each). The probability distribution over documents given *want* would have 8 non-zero values (.125 each). The entropy calculations are

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
frog	1		2				1			
snake	3		1		1					
computer		4								1
user	1	1		1		1	1			1
want	4	5		7	1	2		4	3	1
try	1	1	4		1	3	2		1	

Table 1: Word counts (term frequencies) in documents (tf_i)

straightforward

$H(P(D \mid computer)) = 1$
$H(P(D \mid want)) = 3$

We could adjust the weight formula to account for relevant by dividing by the entropy measure of the word. Thus, computer would roughly have three times the influence as the word want. In practice, information retrieval approaches don't compute entropy and simply use what is called the **document frequency**, df_i, which is he number of documents that word w_i appears in. Then the weight is adjusted by some function inversely related to the document frequency, such as $\log(N/df_i)$. This weighting scheme is often called the **inverse document frequency (idf)** weighting. For the example above, the factor for *computer* would be log(10/2) = 2.3 and for *want* it would be log(10/8) = .32. Note there is a factor of about 7 difference using the idf compared to a factor of 3 for the entropy measure. What formula (of these of other variants) produces the best results is an empirical matter. Bringing this all together, we get the **tf.idf** weighting scheme as follows:

$weight_{i,j} = (1+ \log(tf_i))\log(N/df_j)$ if $tf_{i,j} > 0$ and 0 otherwise

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	df_i	$\log(N/df_i)$
frog	1		2				1				3	1.74
snake	2.58		1		1						3	1.74
computer		3								1	2	2.32
user	1	1		1		1	1			1	6	.74
want	3	3.3		3.8	1	2		3	2.6	1	8	.32
try	1	1	3		1	2.6	2		1		7	.51

Table 2: Computing the weights from the $tf_{i,j}$): Showing $1+\log(tf_i)$ and df_i factors

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
frog	1.74		3.48				1.74			
snake	2.58		1		1					
computer		6.96								2.32
user	.74	.74		.74		.74	.74			.74
want	.96	1.06		1.11	.32	.64		.96	.83	.32
try	.51	.51	1.53		.51	1.32	1.02		.51	

Table 3: the tf.idf vectors

These could then be used as before. If we normalize them, we can use the dot-product measure (i.e., the cosine measure) to compare the vector computed from a query to the documents in the database. Table 4 shows how similar each document is to d1 using the cosine measure.

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
d1	1.0	.05	.7	.24	.91	.22	.38	.19	.22	.07

Table 4: The closeness measure between D1 and the other documents

As you can see, document d1 is closest to d5, and then to d3. This might seem counterintuitive because d3 shares two significant words with d1, namely *frog* and *snake*. The reason d5 is preferred is that is has fewer words in its vector, so the one instance of *snake* in d5 accounts for most of the length in its normalized vector (in fact, it is .94). D1 correspondingly, has most of its length in *snake* as well because of the three instances. Thus we get the high cosine score.

Reducing the Size of Vectors

In many applications we would like to reduce the size of the vectors used in the representation. Consider in a general information retrieval task we would need vectors that are as long as there are words in the language! This might be done not only to save on storage and computation time, but also to smooth the representation and better capture generalities. For instance, one document might contain the word *computer* while another contains *computation*. Currently, these would seem unrelated.

The most common initial technique used is to first eliminate a fixed set of words that are clearly not related to semantic content such as the function words *the*, *a, some, of,* and so on. Typically, there might be about 100 of these and they are usually referred to as the **stop list**. While this eliminates a vast number of tokens in the corpora, it doesn't reduce the size of the vectors significantly. Another technique is to use some stemming program to collapse different word forms to a common root. For instance, all the different forms of a verb, say *walk, walks, walked,* and *walking,* would get mapped to the root *walk.* This significantly reduces the size of the vectors, but they still remain unwieldy for most applications.

Next we could try to eliminate the terms that are not informative. We could, for instance, set some threshold on the document frequency (or the entropy of the conditional distribution), and any word that exceeds the threshold would be dropped. But this has a problem. Say we want a vector consisting of only 1000 words. If we took the 1000 words with the lowest entropy measures (as measured over their distribution over documents), we'd end up with a list of words that occurred only in one or two documents! While very distinctive in the training corpus, since they don't occur in many documents, they probably would into be very useful in classifying new documents. To help with this problem, words that occur only once are often discarded as the information they provide is considered unreliable. But whatever limit we set on how many times a word occurs, the words just above that number will tend to be the ones that have the lowest entropy.

Word Clustering

We could also reduce the size of vectors by clustering similar words into groups. For instance, rather than looking at the information in Table 3 as a representation of documents as a vector of words, we could look at it as a representation of words as vectors over the documents. Thus, each word is represented as a ten-element vector. We could then see which vectors are closest using some similarity metric, and merge the two closest words into a single class. We could iterate this procedure until we've reduced the size of our vectors to the desired size. This approach has another advantage besides reducing the size of the vectors. By grouping words into similar classes we are smoothing the data as well. For instance, say we merge *computer* and *CPU* into the same class. Now a document that only contains *computer* will still have some similarity with a document that only contains *CPU*. Thus we are starting to obtain a representation that generalizes over useful semantic categories.

The success of this approach depends on the similarity metric used. One obvious approach is to use the cosine measure, which will tend to cluster vectors that have similar overlaps. Note you might think that the presence of 0's in vectors would allow the other vector to have values for that word without hurting the score. But this is not the case for the vectors are normalized. For instance, consider vectors of length 5 for documents A,B,C,D and E. Say word x only occurs once in A, while y occurs in A and C. Then the normalized vectors are (1,0,0,0,0) and (.71,0,.71,0,0) and so the cosine score is .71. A word that occurs in all documents would be (.45,.45,.45,.45,.45) and thus would have a score of .45. So the cosine score does capture somewhat our intution. One might argue, however, that the difference between the two is not as large as we would like.

There are many other metrics we could use. These are usually expressed in terms of the similarity between probability distributions. To make the transition, observe that we can view a vector as a set of counts that can be used to compute a probability distribution over documents of form $P(D=d_i \mid W=w_i)$. For instance, starting from the raw counts in table 1, we could estimate the distribution P(D|frog) as follows: P(d1 | frog) = 1/4; P(d3 | frog) = 1/2; P(d7 | frog) = 1/4 and P(di frog) =0 for all others. Given these probability distributions, we could now devise some sort of similarity measure to find which two distributions are most similar. There are a number of measures used in practice. The simplest one, called the L1 Norm, is the sum of the absolute values of the probabilities:

$$L1(p, q) = \sum_i | p_i - qi |$$

With this measure, we get the similarities between the words shown in Table 5. From this, we see that *want* and *user* have the most similar distributions, and so would be a good candidate to merge. Once we merge these two, redo the counts, and then recompute the similarity measure (now with the class containing *want* and *user* being one element). We could continue the collapse the most similar classes until we find a vector of the desired size.

	frog	snake	computer	user	want	try
frog	0					
snake	1.1	0				
computer	2	2	0			
user	1.33	1.67	1.33	0		
want	1.7	1.63	1.56	.81	0	
try	.92	1.29	1.85	1.05	1.32	0

Table 5: The L1 Norm comparing the similarity of word distributions

Several other measures have been proposed as well. The **Kullback-Leiber divergence (KL divergence**) uses the following formula:

$$D(P \parallel Q) = \sum_i p_i \log(p_i/q_i)$$

This can be viewed as measuring how much information is lost when we assume distribution q and the real distribution is p. We could also use the cross entropy of p and q to obtain another quite similar measure. The problem with both of these is that they are undefined if there is a qi = 0. In addition, they are suspect as measures of similarity because they are not symmetric, i.e.,

$$D(P \parallel Q) \neq D(Q \parallel P).$$

A more intuitive measure, the **Information Radius (IRad**) is a symmetric version which compares each distribution to their average distribution:

$$IRad(P, Q) = D(P \parallel (P+Q)/2) + D(Q \parallel (P + Q)/2)$$

where (P+Q)/2 is simply the average distribution, i.e., the probability of x is (P(x)+Q(x))/2.

In an evaluation of difference measures in 1997, the IRad measure consistently performed better on a word sense disambiguation task (Dugan, 1997).

Dimensionally Reduction

Yet another way to reduce the size of the vectors is to compute lower dimension vectors that best approximate the higher dimension one. To understand this, let's consider the two dimensional case. Say we have a vector space with three values, (1, .75), (2, 1) and (4, 2.5). In this case, we could fit a line that passes through each point, with the formula .75x -.5. Given this, we could produce an exact representation of these three points by just encoding the first dimension, and using this formula for the second. i.e., we use x to represent the vector (x, .75x-.5). As you can see, x=1, would give the vector (1, .75), x=2 would give (2, 1) and x=3 would give (4, 2.5). Furthermore, we could use distance measures in one dimension to measure closeness (i.e., point 1 is closer to 2 than 2 is to 4,

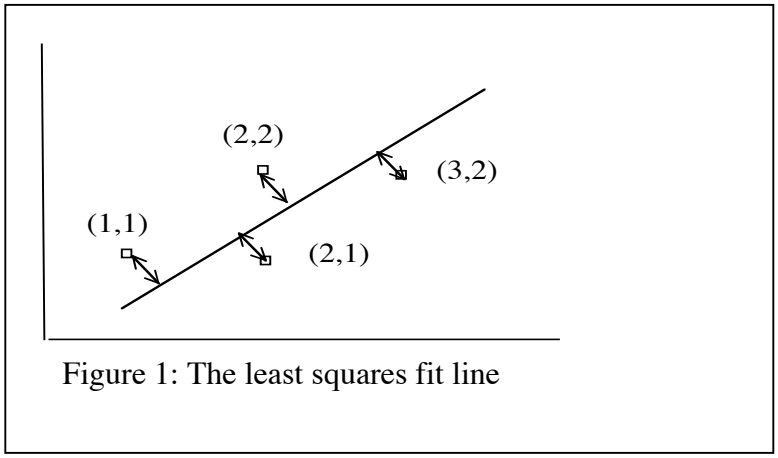

Figure 1: The least squares fit line

just as (1, .75) is closer to (2,1) than (2,1) is to (4, 2.5). Of course, usually we don't have such regularities in data, and must compute approximations rather than exact lower dimension approximations. In that case, we then want to find lower dimensional representations that minimize the distortion (i.e., the error when we convert to the lower dimension representation and then convert back to the higher dimension representation). The classic approach to finding the best approximation for a set of points is linear regression, which finds a line that minimizes the overall distance of the points to the line.

There is a well-known set of formulas for computing the best fit line equation: mx+b. Let μ_x be the mean of the x dimension, and μ_y be the mean of the y dimension, then

$$m = \sum_I (\mu_x - x_i)(\mu_y - y_i)$$
$$b = \mu_y - m*\mu_x$$

Using this formula, we can collapse a two dimensional point (x, y) to the value x (which would map back to the point (x, mx+b). Hence we introduce a distortion error of SQRT(y - mx - b)^2) if we are using Euclidean distance measure.

There is a technique, called Singular Value Decomposition (SVD) which applies similar ideas to higher dimensional spaces, and reduces an n-dimensional vector (where n is the number of words) to a k-dimensional vector much smaller than n. The application of this technique to information retrieval is called **Latent Semantic Indexing (LSI)**, reflecting the intuition that the dimensionally reduction will tend to cluster words of similar distributions in documents, which presumably reflect semantic similarity. The advantage for information retrieval is improved recall. A query involving the term dining might return documents containing restaurant reviews even if the word dining does not appear in it. If there is enough similarity between the documents indexed for dining and the terms in reviews such as restaurant, then these similarities will tend to be reflected in one of the reduced dimensions. Of course, with improved recall, we may have a problem with maintaining good precision.