Natural Language Processing Principles and Applications (Elective -II) [AML 5233]

Marks: 50 Duration: 90 mins.

II Sessional Exam (Additional Test) – Answer Scheme

1. (a) In a corpus of 10000 documents you randomly pick a document, say D, which has a total of 250 words and the word 'data' occurs 20 times. Also, the word 'data' occurs in 2500 (out of 10000) documents. What will be the tf-idf entry for the term 'data' in a bag of words vector representation for D. (5 Marks)

Solution: Using normalized term frequency tf('data', D) = 20/250 .

Idf is idf = ln(10000/2500).

So tf idf =2/25 × ln 4.

(b) You have the following three documents - D1, D2, D3:

D1: Natural language processing is becoming important since soon we will begin talking to our computers.

D2: If computers understand natural language they will become much simpler to use.

D3: Speech recognition is the first step to build computers like us.

Answer the following with respect to the above set of 3 documents. (5 Marks)

What are the number of bigrams and trigrams in D1?

Solution: D1 after normalization looks as follows:

Natural Language process become important soon begin talk computer.

Bi-grams and trigrams are extracted by sliding windows of size 2 and 3 respectively over the sentence. So, if n is the length of the sentence then No. of bigrams=(n − 1) where n ≥ 2 and No. of trigrams=(n − 2) when n ≥ 3. So, we get 8 bigrams and 7 trigrams.

2. Consider that our document collection S has the following documents: D1, ..., D5:

document	words
D1:	Data Base System Concepts
D2:	Introduction to Algorithms
D3:	Computational Geometry: Algorithms and Applications
D4:	Data Structures and Algorithm Analysis on Massive Data Sets
D5:	Computer Organization

Our dictionary DICT consists of 8 words: {w1 = data, w2 = system, w3 = algorithm, w4 = computer, w5 = geometry, w6 = structure, w7 = analysis, w8 = organization}. We consider that, by stemming, "computer" and "computational" are regarded as the same word, and so are "algorithms" and "algorithm".

Problem 1. Let tf (w, D) denote the term frequency of term w in a document D. Give the value of tf (wi, Dj) for all 1 ≤ i ≤ 8 and 1 ≤ j ≤ 5.

Sol: tf (wi, Dj) for all 1 ≤ i ≤ 8 and 1 ≤ j ≤ 5. is

	D1	D2	D3	D4	D5
w1	1	0	0	2	0
w2	1	0	0	0	0
w3	0	1	1	1	0
w4	0	0	1	0	1
w5	0	0	1	0	0
w6	0	0	0	1	0
w7	0	0	0	1	0
w8	0	0	0	0	1

Problem 2. Let idf (w) denote the inverse document frequency of term w. Give the value of idf (wi) for all 1 ≤ i ≤ 8.

Sol:

w_1	1.32
w_2	2.32
w_3	0.74
w_4	1.32
w_5	2.32
w_6	2.32
w_7	2.32
w_8	2.32

For example, $idf(w_1) = \log_2(|S|/2) = \log_2(5/2) = 1.32$. In particular, the 2 in the denominator is because w_1 appears in two documents D_1 and D_4.

Identify the sentence that gets a higher probability with 4-gram model. (Use the corpus as in fig.) (6 Marks)

Given Corpus

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

a) <s> Do Henry like college </s>
b) <s> Do I like Henry </s>

Comment if there is a need for Laplace smoothing required here. (4 Marks)

Word	***Frequency***
<s>	*7*
</s>	*7*
I	*6*
am	*2*
henry	*5*
like	*5*
college	*3*
do	*4*

a) <s> Do Henry like college </s>
= P(like|<s> Do Henry) x P(college|Do Henry like) x P(</s> Henry like college)
= 1/1 x 1/1 x 1/1
= 1
b) <s> Do I like Henry </s>
= P(like|<s> Do I) x P(Henry | Do I like) x P(</s> I like Henry)
= 2/2 x 1/2 x 1/1
= 0.5

The sentence with higher probability is <s> Do Henry like College </s>

Laplace smoothing prevents the model from assigning zero probabilities to features not present in the training data, ensuring that the model can make predictions for previously unseen words. Since we don't have any zero probabilities here, we don't need to use Laplace Smoothing.

Illustrate the concept of cosine for computing word similarity in vector semantics with appropriate example.

To measure similarity between two target words v and w, we need a metric that takes two vectors (of the same dimensionality, either both with words as dimensions, hence of length jVj, or both with documents as dimensions, of length jDj) and gives a measure of their similarity. By far the most common similarity metric is the cosine of the angle between the vectors.

The cosine—like most measures for vector similarity used in NLP—is based on the dot product operator from linear algebra, also called the inner product:

$$\text{dot product}(\mathbf{v},\mathbf{w}) = \mathbf{v}\cdot\mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

The dot product acts as a similarity metric because it will tend to be high just when the two vectors have large values in the same dimensions. Alternatively, vectors that have zeros in different dimensions—orthogonal vectors—will have a dot product of 0, representing their strong dissimilarity.

This raw dot product, however, has a problem as a similarity metric: it favors long vectors. The vector length is defined as

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

The dot product is higher if a vector is longer, with higher values in each dimension. More frequent words have longer vectors, since they tend to co-occur with more words and have higher co-occurrence values with each of them. The raw dot product thus will be higher for frequent words. But this is a problem; we'd like a similarity metric that tells us how similar two words are regardless of their frequency.

We modify the dot product to normalize for the vector length by dividing the dot product by the lengths of each of the two vectors. This normalized dot product turns out to be the same as the cosine of the angle between the two vectors, following from the definition of the dot product between two vectors a and b:

$$\begin{aligned} \mathbf{a}\cdot\mathbf{b} &= |\mathbf{a}||\mathbf{b}|\cos\theta \\ \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} &= \cos\theta \end{aligned}$$

The cosine similarity metric between two vectors v and w thus can be computed as:

$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

For some applications we pre-normalize each vector, by dividing it by its length, unit vector creating a unit vector of length 1. Thus, we could compute a unit vector from a by dividing it by jaj. For unit vectors, the dot product is the same as the cosine.

The cosine value ranges from 1 for vectors pointing in the same direction, through 0 for orthogonal vectors, to -1 for vectors pointing in opposite directions. But since raw frequency values are non-negative, the cosine for these vectors ranges from 0–1.

Let's see how the cosine computes which of the words cherry or digital is closer in meaning to information, just using raw counts from the following shortened table:

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\cos(\text{cherry},\text{information}) = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = .018$$

$$\cos(\text{digital},\text{information}) = \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = .996$$

The model decides that information is way closer to digital than it is to cherry, a result that seems sensible.

What is perplexity in NLP? Discuss about the perplexity metric in language model and show how to calculate perplexity for a bigram model?

We don't use raw probability as our metric for evaluating language models, but a variant called perplexity. The perplexity (sometimes called PPL for short) of a language model on a test set is the inverse probability of the test set, normalized by the number of words.

For a test set W = $w_1 w_2 : : : w_N$,:

$$\begin{aligned}\text{perplexity}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}\end{aligned}$$

We can use the chain rule to expand the probability of W:

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

The perplexity of W computed with a bigram language model is still a geometric mean, but now of the bigram probabilities:

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

Note that because of the inverse in Equation the higher the conditional probability of the word sequence, the lower the perplexity. Thus, minimizing perplexity is equivalent to maximizing the test set probability according to the language model.

What we generally use for word sequence in equation is the entire sequence of words in some test set. Since this sequence will cross many sentence boundaries, we need to include the begin- and end-sentence markers <s> and </s> in the probability computation. We also need to include the end-of-sentence marker </s> (but not the beginning-of-sentence marker <s>) in the total count of word tokens N.

perplexity is a function of both the text and the language model: given a text W, different language models will have different perplexities. Because of this, perplexity can be used to compare different n-gram models. Let's look at an example, in which we trained unigram, bigram, and trigram grammars on 38 million words (including start-of-sentence tokens) from the Wall Street

Journal, using a 19,979 word vocabulary. We then computed the perplexity of each of these models on a test set of 1.5 million words, using relevant equations for bigrams, and the corresponding equation for trigrams. The table below shows the perplexity of a 1.5 million word WSJ test set according to each of these grammars.

	Unigram	Bigram	Trigram
Perplexity	962	170	109

As we see above, the more information the n-gram gives us about the word sequence, the higher the probability the n-gram will assign to the string. A trigram model is less surprised than a unigram model because it has a better idea of what words might come next, and so it assigns them a higher probability. And the higher the probability, the lower the perplexity

Perplexity for Bigram <S> I like college </S>

=P(I| <S>) × P(like | I) × P(college| like) × P(</S> | college)

=3/7 × 3/6 × 3/5 ×3/3 = 9/70=**0.13**

PP(w)=(1/0.13)$^{1/4}$=1.67