CS671A/CS671: Introduction to Natural Language Processing

Mid-semester exam

Time: 2 hours 21-Feb.-2018
Max marks: 80

---

1. *Answer all 3 questions. The question paper has 2 pages.*
2. *You can refer to your handwritten class notes and class slides. No other paper or electronic documents/gadgets are allowed.*
3. *Keep your answers precise and brief.*
4. **Do all parts of a question together. Do not mixup answers to parts of different questions in the answer script.**

---

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 tfidf entry for the term 'data' in a bag of words vector representation for D.

> **Solution:**
> Using normalized term frequency $tf(\text{`}data', D) = \frac{20}{250}$. Idf is $idf = ln(\frac{10000}{2500})$. So $tfidf = \frac{2}{25} \times ln\, 4$.

(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 after text normalization (stop word removal and lemmatization) has been done on all 3 documents.

i. What is the vocabulary $\mathcal{V}$?

> **Solution:**
> There are multiple answers possible based on which words are treated as stop words and whether some degree of chunking is done - for example one may decide to chunk 'natural language processing' and/or 'first step'. Here we take a rather simple approach. In the context of the given documents we retain words that are likely to provide useful content and discard the rest. We also we do not chunk.
>
> $\mathcal{V}$ =(`become, build, computer, first, important, language, like, natural, processing, recognition, simple, speech, step, talk, understand, us, use`)

ii. What are the number of bigrams and trigrams in **D2**?

> **Solution:**
> **D2** after normalization looks as follows:
>
> `computer understand natural language become simple use`

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 $n \geq 2$ and No. of trigrams=$(n-2)$ when $n \geq 3$. So, we get 6 bigrams and 5 trigrams.

Of course, this answer will differ if your normalized document has more or less words.

iii. What will be the BoW document vector for document **D3** if we are using a *tf* based document vector?

Solution:
Document **D3** after normalization:

`speech recognition first step build computer like us`

Bag-of-words vector with normalized tf:
$\frac{1}{8}(0,1,1,1,0,0,1,0,0,1,0,1,1,0,0,1,0)$
All words occur only once in **D3**. So, if you use plain *tf* then the factor $\frac{1}{8}$ will not be present.

(c) Suppose you have the following two 4-dimensional word vectors for two words w_1 and w_2 respectively:
$\mathbf{w_1} = (0.2, 0.1, 0.3, 0.4)$ and $\mathbf{w_2} = (0.3, 0, 0.2, 0.5)$
What is the cosine similarity between $\mathbf{w_1}$ and $\mathbf{w_2}$? Are the words w_1 and w_2 similar or dissimilar?

Solution:
We can calculate cosine similarity from:

$$cosine(\theta) = \frac{\langle \mathbf{w_1}, \mathbf{w_2} \rangle}{\| \mathbf{w_1} \| \| \mathbf{w_2} \|}$$

The expression is easier to calculate if we scale both vectors by multiplying by 10 - this does not change the cosine. We get,

$$\begin{aligned} cosine(\theta) &= \frac{6+6+20}{\sqrt{4+1+9+16}\sqrt{9+4+25}} = \frac{32}{\sqrt{30 \times 38}} \\ &= \frac{32}{2\sqrt{285}} \\ &\approx \frac{16}{17} \end{aligned}$$

The value is very close to 1 so the words w_1, w_2 are very similar.

(d) In word2vec (skipgram) we compute the predicted probability distribution vector ($\hat{\mathbf{y}}$) on the vocabulary via a softmax over the output vector $\mathbf{z}$ i.e. $\hat{\mathbf{y}} = softmax(\mathbf{z})$. Given that the desired output is $\mathbf{y}$ and the error function $\mathcal{E}$ is the cross entropy function $\mathcal{E} = \sum_i -y_i \, ln(\hat{y}_i)$ derive the gradient for the first step in the backpropagation.

Solution:
Let $\mathbf{h}$ be the output of the hidden layer and W the weight matrix between the hidden and output layers. We can write the output vector $\mathbf{z}$ as: $\mathbf{z} = W^T\mathbf{h} + \mathbf{b}$, $\mathbf{b}$ is the bias. The output vector is transformed by $softmax$ to give the predicted output $\hat{\mathbf{y}}$. So, $\hat{\mathbf{y}}_\mathbf{i} = \frac{e^{\mathbf{z_i}}}{\sum_{j=1}^{|\mathcal{V}|} e^{\mathbf{z_j}}}$, $i = 1..|\mathcal{V}|$.

The error $\mathcal{E}$ is cross entropy error. So, $\mathcal{E} = -\sum_{i=1}^{|\mathcal{V}|} \mathbf{y_i} \, ln(\hat{\mathbf{y}}_\mathbf{i})$. For the first step of backpropa-

gation we will update weights W_{ij} so we want the gradient $\frac{\partial \mathcal{E}}{\partial W_{ij}}$. Now, $\frac{\partial \mathcal{E}}{\partial W_{ij}} = \frac{\partial \mathcal{E}}{\partial \mathbf{z_i}} \frac{\partial \mathbf{z_i}}{\partial W_{ij}}$. Substitute for $\hat{\mathbf{y}}_\mathbf{i}$ in $\mathcal{E}$ and differentiate w.r.t $\mathbf{z_i}$.

$$
\begin{aligned}
\mathcal{E} &= -\sum_{k=1}^{|voc|} \mathbf{y_k}\, ln(\frac{e^{\mathbf{z_k}}}{\sum_{j=1}^{|\mathcal{V}|} e^{\mathbf{z_j}}}) \\
&= -(\sum_{k=1}^{|voc|} \mathbf{y_k}(\mathbf{z_k} - ln(\sum_{j=1}^{|\mathcal{V}|} e^{\mathbf{z_j}}))) \\
\frac{\partial \mathcal{E}}{\partial \mathbf{z_i}} &= -\mathbf{y_i} + \frac{e^{\mathbf{z_i}}}{\sum_{j=1}^{|\mathcal{V}|} e^{\mathbf{z_j}}} \\
&= \hat{\mathbf{y}}_\mathbf{i} - \mathbf{y_i} \\
\frac{\partial \mathbf{z_i}}{\partial W_{ij}} &= \mathbf{h_i} \\
\frac{\partial \mathcal{E}}{\partial W_{ij}} &= (\hat{\mathbf{y}}_\mathbf{i} - \mathbf{y_i})^T \mathbf{h_i}
\end{aligned}
$$

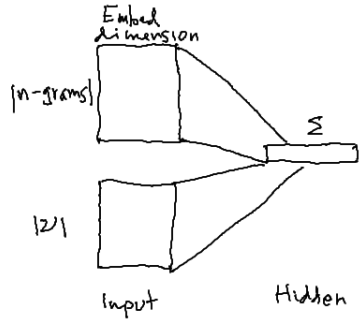
(e) Suppose you want to find vector space embeddings similar to word2vec (e.g. skipgram) for morphologically rich languages[1]. How will you change the standard word2vec skipgram algorithm to 'hopefully' get improved embeddings?

For full credit give as much detail as possible. For example: What is your vocabulary? What is the output? How is it scored to generate error? Etc.

Solution:

There is more than one way to do this. The simplest is to start with pre-trained word vectors and imagine that this word is broken into a sequence of n-grams for small values of n (say 2 to 5) then repeat the skipgram training algorithm but now with central n-gram, context n-gram pairs (similar to central word, context word pair). For a particular value of n the output vector will now have a dimension equal to all possible n-grams (equivalent to $\mathcal{V}$) over the alphabet and the expected n-gram will be a 1-hot vector where only the context n-gram has entry 1 and all others are 0. This process can be repeated for different values of n.

Another variant is to simultaneously feed a word vector and its n-grams to the hidden layer where everything is summed. The rest is similar to skipgram. The sketch below shows how:



A third way is to create a vocabulary of morphemes (will be much smaller than number of n-grams) in the language and then use the above scheme.

[1] In morphologically rich languages words have meaningful sub-structure. So, sub-word elements carry grammatical, semantic or other information. While English is not morphologically rich an example word is: `un-re-mark-able`.

(f) We used the EM (expectation maximization) algorithm for Naive-Bayes to get the parameters of the distribution when the learning set did not have any labels. Can you think of another use for the EM algorithm? Justify.

Solution:

EM is an iterative algorithm to compute hidden/unknown parameters θ when observable variables $\mathbf{x}$ can be thought to be generated by hidden variables $\mathbf{y}$. If $P_\theta(y)$, $P_\theta(x)$ are the probability distribution where θ is the set of parameters.

Many distributions in real life applications can be approximated very accurately by a mixture of Gaussians (i.e. a linear combination of elementary Gaussians).

$$P_\Theta(x) = \sum_{i=1}^{m} \alpha_i P_{\theta_i}(x)$$

where P_{θ_i} are elementary Gaussians and α_i are scalars called mixture coefficients. Parameter vector is: $\Theta = (\theta_1, \ldots, \theta_m, \alpha_1, \ldots, \alpha_m)$.
EM can be used to learn the parameters.

Another similar EM like algorithm is the iterative clustering algorithm where we start with k-centroids for clusters and do:

```
Till convergence
   E-step:  assign points to nearest centroid.
   M-step:  recompute centroids (model parameters) for the least centroid
            distance for points in the cluster.
```

[3,(3,(2,2),3),(3,1),5,5,3=30]

2. When you create a password on a website often there are constraints on the nature of the password. Suppose the constraints on the password are:

   a) Password should be at least 8 characters long but not more than 15 characters.
   b) It must have at least one special character from !%@*.
   c) It must have some letters and some digits - that is at least one letter and one digit.
   d) At least one of the letters should be a capital letter.
   e) It should not have characters not listed above.

   Assume you had a function called `match(regexp,str)` that returns `True` or `False` depending on whether the regular expression `regexp` matches the string `str` or not.

   (a) Write an expression using `match`, egrep/any other regular expression language and boolean operations like `OR, AND, NOT` that will return `True` if the password satisfies the given constraints and `False` otherwise.

   [Hint: egrep has defined character classes: [[:alnum:]] - all alpha-numeric characters, [[:digit:]] - all digits, [[:upper:]] - all upper case characters. Use the character classes to express your answer concisely.]

   Solution:

   The easiest is to write regular expressions for each clause in the constraints and `AND` them. Assume the password string is in variable `passwd`:

   1. `match(([[:alnum:]]|[!@\%*]){8,15}, passwd)` - `True` if the password contains only alphanumeric characters or the given special characters and is between 8 to 15 characters long.

2. `match([[:upper:]], passwd)` - `True` if password contains at least one upper case letter.
3. `match([[:digit:]], passwd)` - `True` if password contains at least one digit.
4. `match([!@\%*], passwd)` - `True` if password contains at least one special character.
5. `NOT match([^A-Z a-z 0-9 !@\%*], passwd) - True` if password does not contain any character other than those mentioned.

An `AND` of the above five boolean expressions will check the constraints for the password.

(b) Can we write just a regular expression without using the boolean function `match` that will match all valid passwords? Briefly, justify your answer.

Solution:
In principle it can be written but it will amount to an alternation of all the possible actual password strings which are finite but very large. This is because it is not possible to enforce constraints like 'at least one each of capital letter or digit or special character should be present in the string' without enumerating all the possibilities.

[20,5=25]

3. (a) In the table below you are given the bag of words in 4 documents and the label of the document (spam or not spam).

Document	**Bag of words**	**Label**
1	{price, weight, loss, vitamin, discount}	spam
2	{vitamin, weight, discount, sad}	spam
3	{loss, sad, sorry}	not spam
4	{weight, foundation, price}	not spam

i. Calculate the parameters of the Naive-Bayes model from the data given (use Laplace - that is add 1 - smoothing for zeroes).

Solution:
The Naive Bayes parameters are: $P(spam)$, $P(non-spam)$ and $P(w_i|c)$ for each $w_i \in \mathcal{V}$ and $c \in$ {spam, non-spam}. These probabilities have to be estimated from the learning set $\mathcal{L}$. We have $|\mathcal{L}| = 4$ where 2 documents each are spam and non-spam. So, $P(spam) = P(non-spam) = \frac{1}{2}$. To calculate class conditional term probabilities for each $w_i \in \mathcal{V}$ conditioned on the class label we use MLE estimates. We must handle the case when a particular term does not occur at all in a particular class leading to a conditional probability of 0 which will further lead to a 0 probability when we try to predict the label for a test document that contains that term.
So, let us assume that we start with a uniform prior ϵ over $\mathcal{V}$ for each class. Then we count occurrences for each term in $\mathcal{V}$ conditioned on the class and add it to the prior. The table below shows this for the given data:

BoW index	discount	foundation	loss	price	sad	sorry	vitamin	weight
Prior	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ
Spam	$2+\epsilon$	ϵ	$1+\epsilon$	$1+\epsilon$	$1+\epsilon$	ϵ	$2+\epsilon$	$2+\epsilon$
Non-spam	ϵ	$1+\epsilon$	$1+\epsilon$	$1+\epsilon$	$1+\epsilon$	$1+\epsilon$	ϵ	$1+\epsilon$

The class conditional probabilties in each case are obtained by dividing each value by the sum of the values in each row for that class - that is:
$9+8\epsilon$ for spam and $6+8\epsilon$ for non-spam. One can choose any value for ϵ. If ϵ is 1 we have Laplace or add 1 smoothing. So, using $\epsilon = 1$ the class conditional probabilities are:

BoW index	discount	foundation	loss	price	sad	sorry	vitamin	weight
Spam	$\frac{3}{17}$	$\frac{1}{17}$	$\frac{2}{17}$	$\frac{2}{17}$	$\frac{2}{17}$	$\frac{1}{17}$	$\frac{3}{17}$	$\frac{3}{17}$
Non-spam	$\frac{1}{14}$	$\frac{2}{14}$	$\frac{2}{14}$	$\frac{2}{14}$	$\frac{2}{14}$	$\frac{2}{14}$	$\frac{1}{14}$	$\frac{2}{14}$

The Naive-Bayes method contains two parts. The first is the probability model for the parameters and the second is the independence assumption. The independence assumptions are common to all Naive-Bayes models and they make the calculations computationally tractable by reducing the number of parameters.

The calculation above used the multi-nomial model for probabilities. This involves actual counts of words. An alternative is the Bernoulli model that only considers presence (1) and absence (0) of the feature (i.e. the word) in the document. For the Bernoulli model the prior probabilities for *spam* and *non-spam* are the same. But the class conditional probabilities have to be calculated by $P(w|c) = \frac{n_w}{N_c}$ where N_c is the number of documents of class c and n_w is the number documents with label c in which w occurs. Again we face the problem of 0 probability if a particular word never occurs in documents of a particular class (e.g. *foundation* in spam) but occurs in a test document. So, we have to adopt a smoothing method. Using equal priors for presence and absence gives $P(w|c) = \frac{n_w+1}{N_c+2}$ (actually assumes a β prior and then an MLE calculation with suitable choice of hyper-parameters leads to this - but can be simply thought of as equiprobability). This gives the following parameter values:

BoW index	discount	foundation	loss	price	sad	sorry	vitamin	weight
Spam	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{2}{4}$	$\frac{2}{4}$	$\frac{2}{4}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{3}{4}$
Non-spam	$\frac{1}{4}$	$\frac{2}{4}$	$\frac{2}{4}$	$\frac{2}{4}$	$\frac{2}{4}$	$\frac{2}{4}$	$\frac{1}{4}$	$\frac{2}{4}$

There is yet another way to calculate smoothed parameters. We can add a dummy document for each term that does not appear in a class. The document is added to all the classes so that the balance of probabilities remain approximately the same. This will ensure that all class conditional parameters are non-zero and will work well when the Bernoulli model is being used.

Note that all the different methods for parameter calculations are actually estimations based on approximations. Often they are not even normalized since ultimately they will be used only to decide a class label. However, in spite of the approximations involved the Naive-Bayes decision is accurate surprisingly often. The actual parameter estimates may be far from the real values but most of the time they are in the right direction and give the correct answer.

ii. What are the probabilities for the labels *spam* and *non spam* for the document with the following bag of words:

{`weight, price, discount, foundation, loss`}?

Solution:

Let d be the test document then:

$P(spam|d) = \frac{1}{2} \times \frac{3}{17} \times \frac{2}{17} \times \frac{3}{17} \times \frac{1}{17} \times \frac{2}{17} = \frac{36}{17^5}$

$P(non\text{-}spam|d) = \frac{1}{2} \times \frac{2}{14} \times \frac{2}{14} \times \frac{1}{14} \times \frac{2}{14} \times \frac{2}{14} = \frac{16}{14^5}$

The BoW vector (lexicographic order) is: $(1, 1, 1, 1, 0, 0, 0, 1)$. Using the Bernoulli estimates for parameters (we have to use presence (1) as well as absence (0)):

$P(spam|d) = \frac{1}{2} \times \frac{3}{4} \times \frac{1}{4} \times \frac{2}{4} \times \frac{2}{4} \times \frac{2}{4} \times \frac{3}{4} \times \frac{1}{4} \times \frac{3}{4} = \frac{216}{2 \times 4^8}$

$P(non\text{-}spam|d) = \frac{1}{2} \times \frac{1}{4} \times \frac{2}{4} \times \frac{2}{4} \times \frac{2}{4} \times \frac{2}{4} \times \frac{2}{4} \times \frac{3}{4} \times \frac{2}{4} = \frac{192}{2 \times 4^8}$

What is the final label for the document?

Solution:
$P(spam|d) \approx \frac{1}{17^4}$
$P(non-spam|d) = \frac{1}{7^4 \times 28} \approx \frac{1}{16.1^4}$

So, the label is non-spam.
Using the Bernoulli parameters it is clear that the label is spam. So, the way in which parameters are modelled can lead to different results. Typically, the multi-nomial is preferred since it accounts for the frequency of a word in a document while the Bernoulli ignores it.

(b) Let $|\mathcal{V}|$ be the size of the vocabulary, n be the size of the learning set, c be the number of labels, ℓ - be the average length of a document. What will be the complexity for calculating the parameters of the Naive-Bayes model and for finding the label for a test document?

Solution:

For training: Compute c apriori class probabilities and $c|\mathcal{V}|$ conditional probabilities for each class (filling in each row in the earlier table). To do this we have to make a pass over all the words in all the documents (i.e. $n\ell$ words). So, complexity is $O(c|\mathcal{V}| + n\ell)$.

For testing: compute the probability for each class which requires $O(c\ell)$ assuming the test document is also of size ℓ. Clearly, the all parameter values are available at testing time and accessible in constant time.

[(17,4),4=25]