UMSL
Mathematics & Computer Science

22. Natural Language Processing

22.1 Formal Languages vs Natural Languages

- Formal languages (Java, Python, etc.) can be characterized by a set of rules (called a grammar).

- Natural languages (English, Spanish, etc.) cannot be characterized as definitive set of sentences.
  - "Thanks for dinner. I’ve never seen potatoes cooked like that before."
- Natural languages are many times ambiguous.
- Natural languages are constantly changing.

- Hence, instead of deriving rules, we build approximate language models.

N-grams are Not the Recent Trend!

- "Word embeddings," an alternative to one-hot word vectors, are the trend.
- But, N-grams are a powerful, unavoidable feature engineering tools when using lightweight, shallow text processing models such as logistic regression and random forests.

22.1.1 N-gram Character Models

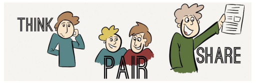
- A simple **language model** for **written text** is a probability distribution of sequences of characters.
    - $P(c_{1:N})$ is the probability of a sequence of N characters, c_1 through c_N.
    - e.g. P("the") = 0.027 and P("zgq") = 0.00000002
- **n-gram** is a sequence of written symbols of length n.
    - 1-gram is "unigram"
    - 2-gram is "bigram"
    - 3-gram is "trigram"
- A model of probability distribution of n-letter sequences is **n-gram model** (n-gram model of characters).

22.1.1 N-gram Character Models

- An n-gram model is defined as a Markov chain of order n-1
    - For example, 3-gram (or trigram) model is defined a Markov chain of order 2
- This means - in a trigram model, the probability of a character c_i depends on two previous characters
    - $P(c_i \mid c_{1:i-1}) = P(c_i \mid c_{i-2:i-1})$
- We can define the probability of a sequence of characters $P(c_{1:N})$ under the trigram model by first factoring with chain rule, and then using the Markov assumption.

$$P(c_{1:N}) = \prod_{i=1}^{N} P(c_i \mid c_{1:i-1}) = \prod_{i=1}^{N} P(c_i \mid c_{i-2:i-1})$$

- For a trigram character model in a language with 100 characters $P(c_i \mid c_{i-2:i-1})$ has a million entries (100x100x100).
- How can we estimate this probability distribution? How much data do we need?



22.1.1 N-gram Models for Language Identification

!

Problem: Given a text, determine what natural language it is written in.

- First build a trigram character model of each candidate language L
  - $P(c_i \mid c_{i-2:i-1}, L)$
- For each L, the model is built by counting the trigrams in a **corpus of that language** (about 1 to 10 million characters from each language)
  - This gives us P(Text | Language), e.g. P("Hello World"| English)
- We know P(Text | Language), how can we get the most probable language?

$$\ell^* = \operatorname*{argmax}_{\ell} P(\ell \mid c_{1:N})$$

$$= \operatorname*{argmax}_{\ell} P(\ell) P(c_{1:N} \mid \ell)$$

Bayes' Rule: P(A | B) = P(A) * P (B | A) / P(B)

$$= \operatorname*{argmax}_{\ell} P(\ell) \prod_{i=1}^{N} P(c_i \mid c_{i-2:i-1}, \ell)$$

Markov chain of order 2

22.1.1 N-gram Models for Language Identification

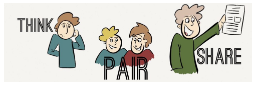
The trigram models for each language can be learned from language corpus, but how to calculate P(L)?

- We may already have some estimates of P(L)
  - For example, if we are selecting a random Web page, we know English is the most likely language and the probability of Macedonian will be less than 1%
- The exact number we select for these priors is not critical because the trigram model usually selects on language that is several order magnitude more probable than any other.
- We could also do P(L) = 1 for all languages when context is unknown.

- N-grams can be used to identify languages with > 99% accuracy.

22.1.2 Smoothing n-gram Models

- The training corpus provides only an estimate of the true probability distribution.
- For common character sequences such as “the” or “␣th” English corpus will give a good estimate: about 1.5% of all trigrams
- However, “␣ht” or “xgz” are very uncommon
    - It is likely that the sequence will have a zero count when the training corpus is standard English
- Should we have P(“␣ht”) = 0?
    - If we do so, the P(English | “The program issues an http request”) will be 0, which is wrong.
    - This is generalization error - We want our language models to generalize well to texts they haven’t seen yet
    - Setting P(“␣ht”) = 0 means that our model thinks that “␣ht” is impossible in English
- Our language model needs to be adjusted!
- What adjustments can we make?



22.1.2 Linear Interpolation Smoothing

- Smoothing is the process of adjusting the probability of low-frequency counts.
- The Backoff model approach
  - We start by estimating n-gram counts
  - For any particular sequence that has a low (or zero) count, we back off to (n-1)-grams.
- Linear Interpolation Smoothing is a backoff model
  - It combines trigram, bigram, and unigram models using linear interpolation
  - Probability estimate is given by

$$\widehat{P}(c_i|c_{i-2:i-1}) = \lambda_3 P(c_i|c_{i-2:i-1}) + \lambda_2 P(c_i|c_{i-1}) + \lambda_1 P(c_i)$$

$$\lambda_3 + \lambda_2 + \lambda_1 = 1$$

  - The parameters λ_i can be fixed or they can be trained.
  - Example: Probability estimate of P(“zgx”) = 0.2 * P(“zgx”) + 0.3 * P(“gx”) + 0.5 * P(“x”)

22.1.3 Model Evaluation

- So many possible models - unigram, bigram, trigram, interpolated smoothing with different values of λ - which model should we choose?
- We can evaluate the model with cross-validation:
  - Step 1: Randomly split the corpus into a training corpus and a validation corpus
  - Step 2: Determine the parameters of the model from the training data
  - Step 3: Evaluate the model on the validation corpus, and obtain average accuracy
    - i.e. Calculate how correct all the predicted probabilities are.
  - Step 4: Repeat step 1 to 3 a few times, say 10 times.
  - Step 5: Average the 10 accuracies - this will be the accuracy of the model

22.1.3 Model Evaluation

- Measuring accuracy can be inconvenient
  - Say that the model predicts the probability of P(“abc”), P(“abd”), P(“abe”), etc.
  - The sum of all these probabilities must be 1 (it is a multi-class classification problem)
  - For a trigram model, if there are 1 million trigrams, the probability of each trigram (on average) is a small number 1/one million
  - If we have a 4-gram model, these probability values are very small numbers, and floating-point underflow can become as issue
- Alternately, we can describe the probability of a sequence using **perplexity**
  - $\text{Perplexity}(c_{1:N}) = P(c_{1:N})^{-1/N}$
  - Perplexity can be thought of as the reciprocal of probability, normalized by sequence length

22.1.3 Perplexity Example

- Suppose there are 100 characters in a language L, and our unigram model says they are all equally likely, i.e. P("A") = 1/100, P("B") = 1/100, etc.
- The perplexity of of the model will be 100 for a sequence of any length
    - For a random sequence of length 1
        - Perplexity = $P(\text{"A"})^{-1/1} = 0.01^{-1} = 100$
    - For a random sequence of length 2
        - Perplexity = $P(\text{"AB"})^{-½} = (0.01 * 0.01)^{-½} = (0.01 * 0.01)^{-½} = 0.0001^{-0.5} = 100$
- If some characters were more likely than others, than the model will reflect it, with a perplexity less than 100.

$$\text{Perplexity}(c_{1:N}) = P(c_{1:N})^{-1/N}$$

Perplexity Example Problem

Suppose there are 3 characters in a language L, and we have built a unigram model. The probabilities for the 3 characters are given by the models are P(“A”) = 0.25, P(“B”) = 0.50, and P(“C”) = 0.25. What will be the perplexity of for the sequences “AAA” and “ABA”? How will the perplexity change for the two sequences if the probabilities were equal for the 3 characters?

- Perplexity(“AAA”) = P(“AAA”)$^{-⅓}$ = $(0.25 * 0.25 * 0.25)^{-⅓}$ = 3.94
- Perplexity(“ABA”) = P(“ABA”)$^{-⅓}$ = $(0.25 * 0.50 * 0.25)^{-⅓}$ = 3.13 (less perplex = high probability)

If the probabilities were equal:

- Perplexity(“AAA”) = P(“AAA”)$^{-⅓}$ = $(0.33 * 0.33 * 0.33)^{-⅓}$ = 2.99
- Perplexity(“ABA”) = P(“ABA”)$^{-⅓}$ = $(0.33 * 0.33 * 0.33)^{-⅓}$ = 2.99

22.1.4 N-gram Word Models

- The vocabulary is much larger
  - There are only about 100 characters (for n-gram character models) but there can be tens of thousands of symbols
  - In English a sequence of letters is surrounded by spaces is a word, but in some languages, like Chinese, words are not separated by spaces
- How to deal with out of vocabulary words?
  - Add a new word the vocabulary: <UNK> (unknown word)
  - Estimate the n-gram counts for <UNK> using the following process:
    - Scan the training corpus and the first time any individual word appears that is previously unknown, replace it with <UNK>
    - All subsequent appearances of the word remain unchanged
    - Then compute n-gram counts for the corpus as usual, treating <UNK> just like any other word
    - When an unknown word appears in a test set, look up its probability under <UNK>
  - This means, words such as “xfhdy” have only single time influence (it is counted only once) even if they appear multiple times

22.1.4 Unigram, Bigram, and Trigram Word Models

- Authors build a unigram, bigram, and trigram models over the words in the book and then randomly sampled sequences of words from the models
    - Unigram: logical are as are confusion a may right tries agent goal the was . . .
    - Bigram: systems are very similar computational approach would be represented . . .
    - Trigram: planning and scheduling are integrated the success of naive bayes model is . . .
    - Which sampling appears more close to AI language?
- Perplexity
    - Unigram model = 891
    - Bigram model = 142
    - Trigram model = 91
- If so, why should we not build 4-gram or 5-gram word models?

22.2 Spam Detection (Classify email as spam or 'ham')

```
Spam: Wholesale FashionWatches -57% today. Designer watches for cheap ...
Spam: You can buy ViagraFr$1.85 All Medications at unbeatable prices! ...
Spam: WE CAN TREAT ANYTHING YOU SUFFER FROM JUST TRUST US ...
Spam: Sta.rt earn*ing the salary yo,u d-eserve by o'btaining the prope,r crede'ntials!


Ham: The practical significance of hypertree width in identifying more ...
Ham: Abstract: We will motivate the problem of social identity clustering: ...
Ham: Good to see you my friend. Hey Peter, It was good to hear from you. ...
Ham: PDS implies convexity of the resulting optimization problem (Kernel Ridge …
```

Language Modeling approach

- Define one n-gram model for P(Message | Spam) by training on spam folder and one model for P(Message | Ham) by training on the inbox (basically, train two models)
- Classify a new message using Bayes' rule (P(c) is estimated by simply counting total number of spam and ham messages):

$$\underset{c \in \{spam, ham\}}{\operatorname{argmax}} P(c \mid message) = \underset{c \in \{spam, ham\}}{\operatorname{argmax}} P(message \mid c) P(c)$$

22.2 Spam Detection (Classify email as spam or ‘ham’)

Machine Learning Approach

- Represent the message as a set of feature/value pairs and apply classification algorithm h to the feature vector X
- N-grams can be input as features
  - For example, for a unigram model, each word is a feature
  - The feature value is the count of the word in the email
  - If there are 100,000 words in the language model, then the feature vector is 100,000 long
  - For a short email message almost all features will have a count zero

	Word 1	**...**	**Word N**	**Spam**
Email 1	2	...	8	0
Email 2	5	...	5	1

- With unigram models, the notion of the order of words is lost

22.2 Spam Detection (Classify email as spam or ‘ham’)

- With bigrams and trigrams the number of features is squared or cubed.
- It can be expensive to run algorithms on a very large feature vector, so often a process of **feature selection** is used
- With feature selection, often large number of features can be reduced to as little as a few hundred features.
- Currently, we can detect spams with 98% to 99% accuracy.

22.3.1 Information Retrieval (IR) Scoring Functions

- **IR** is the task of finding documents that are relevant to a user’s need for information. For example, search engines on the World Wide Web.
- A **scoring function** takes a document and a query and returns a numeric score; the most relevant documents have the highest scores
- If we have score for each documents (for an input query), then we can rank the documents by the score and return the results.

22.3.1 BM25 Scoring Function

- The score is a linear weighted combination of scores for each of the words that make up the query
    - If we search “farming in Kansas”, the total score is the sum of score for each of the three words.
- Three factors affect the weight of the query term, i.e. the BM25 function takes the following into account
    - (1) Term Frequency (TF) - the frequency with which the query term appears in a document
        - For example, the documents that mention “farming” will appear more frequently
    - (2) Inverse Document Frequency Term (IDF)
        - The word “in” appears in almost every document, so it has high document frequency, and thus low inverse document frequency, and so it is not as important to the query as “farming” or “Kansas”
    - (3) Length of the document
        - A million word document will mention all query words, but may not actually be about the query; instead, a short document that mentions all the words is a much better candidate

22.3.1 BM25 Scoring Function

$$BM25(d_j, q_{1:N}) = \sum_{i=1}^{N} \left\{ IDF(q_i) \cdot \frac{TF(q_i, d_j) \cdot (k+1)}{TF(q_i, d_j) + k \cdot \left(1 - b + b \cdot \frac{|d_j|}{L}\right)} \right\}$$

$$L = \sum_i |d_i| / N$$

i.e. calculate BM25 for each term/word in the query and sum them up
[for example BM25(“president lincoln”) = BM25(“president”) + BM25(“lincoln”)]

$$IDF(q_i) = \log_e \frac{N - DF(q_i) + 0.5}{DF(q_i) + 0.5}$$

- The equation above gives the BM25 score for a query consisting of the words $q_{1:N}$ given a document d_j
- $TF(q_i, d_j)$ is the count of the number of times word q_i appears in document d_j
- $|d_j|$ is the length of the document d_j in words
- L is the average document length in the corpus (see above)
- k and b can be tuned by cross validation, typical values are k = 2 and b = 0.75
- $IDF(q_i)$ is the inverse document frequency of word q_i
- $DF(q_i)$ gives the number of documents that contain the word q_i
- N is the number of terms in query and N is the total number of documents

Problem:
A search query “president lincoln” is being scored against a document D. The terms “president” and “lincoln” both appear just once in D. The length of this document D is 90% of the average length of all documents in the corpus. There are 40,000 documents that contain the term “president” and there are 300 documents that contain the term “lincoln”. Assume that IDF for all queries is 1, and k = 1.2 and b = 0.75.

What is the BM25 score for the query against the document D?

Homework 1

A search query “Word1 Word2” is being scored against the following documents (see table). The document corpus (as shown in the table) only contains five documents. The number of times the words “Word1” and “Word2” appear in each of the documents is given in the table. The length of each document is also given (number of pages). Assume k = 1.2 and b = 0.75.

Write a python program to calculate the BM25 score for the query against all the documents and rank the documents by their BM25 score. Your program should read the table from a file, i.e. do not hard code these values into your program.

You may hardcode the values of k and b, but compute IDF, DF, TF, N, L, etc. using the data you read from the text file, at runtime.

The scores you will obtain will range from -2.3 to -4.8.

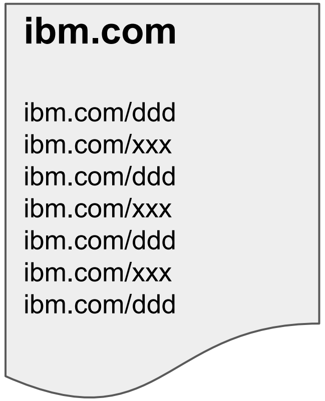
Note: You may get negative scores because of N being close to DF - you can read more here.

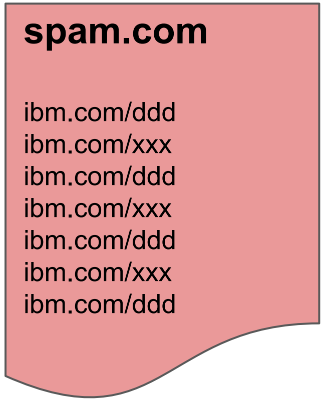
DocumentID	DocumentL	FrequencyOfWord1	FrequencyOfWord2
1	50	0	78
2	80	56	6
3	20	45	89
4	200	89	23
5	10	76	0

22.3.4 The Problem of Tyranny of of TF Scores

Does Term Frequency (e.g. BM25 scoring) always work?

- If the query is “IBM”, how do we make sure that IBM’s home page, ibm.com, is the first result, even if another page mentions the term “IBM” more frequently?
- One possible solution: Check for in-links
  - Say that each in-link is a vote for the quality of the linked-to page
  - The more in-links in a page, the more likely that we will score it higher
  - ibm.com will have many in-links (links to the page), so it should be ranked higher (this assumes that our algorithm can rank pages considering the number of links from other pages)
  - But, if we counted in-links, then it would be possible for a Web spammer to create a network of pages and have them all point to a page of her choosing, increasing the score of that page





22.3.4 The PageRank Algorithm

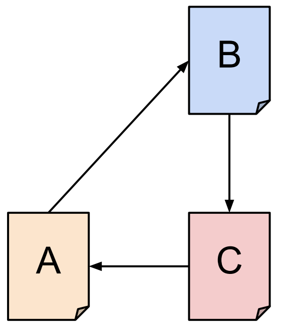
- To solve the tyranny of TF scores problem, the PageRank algorithm is designed weight links from high-quality sites more heavily.
- A high-quality site is the one that is linked-to by other high-quality site (yes, a recursive definition)
- The PageRank for a page p is defined as:

$$PR(p) = \frac{1-d}{N} + d\sum_i \frac{PR(in_i)}{C(in_i)}$$

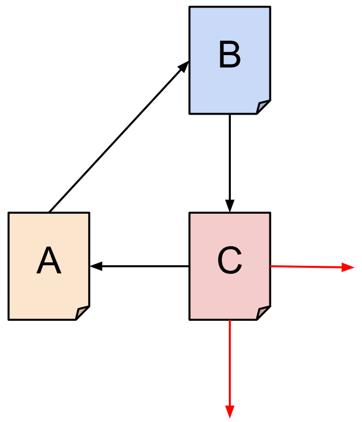
  - PR(p) is the PageRank of page p
  - N is the total number of pages in the corpus
  - in_i are the pages that link in to p
  - $C(in_i)$ is the count of the total number of out-links on page in_i
  - d is a damping factor constant
- PageRank can be computed by an iterative procedure: start will all pages having PR(p) = 1, and iterate the algorithm, updating ranks until they converge

PageRank Example

Given the number of web pages N = 3, and the damping parameter d = 0.7. Calculate the PageRank of the pages A, B, and C. Links between the pages are shown in the graph below.



(a)



(b)

$$PR(p) = \frac{1-d}{N} + d \sum_i \frac{PR(in_i)}{C(in_i)}$$

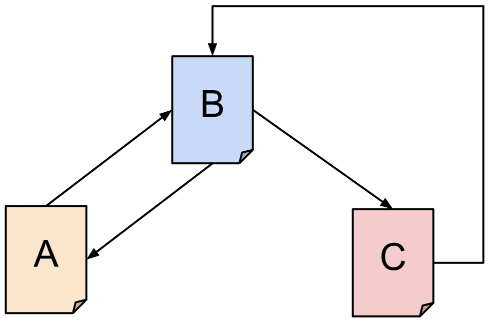
- PR(p) is the PageRank of page p
- N is the total number of pages in the corpus
- in_i are the pages that link in to p
- $C(in_i)$ is the count of the total number of out-links on page in_i
- d is a damping factor constant

Homework 2

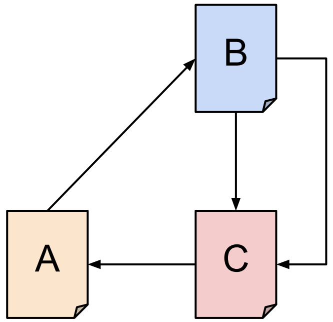
Given the number of web pages N = 3, and the damping parameter d = 0.7. For the two networks shown below, calculate the PageRank of the pages A, B, and C. Links between the pages are shown in the graph itself. Submit (a) the numerical values of PR(A), PR(B), and PR(C) for each of the two networks, and (b) the steps you followed to obtain these values OR the Python code you wrote to calculate the values.

Hints:
Strategy: For each of the two problems, derive a set of (system of) linear equations and iteratively solve them with using a Python program by initializing PR(A) = PR(B) = PR(C) = 1.

Alternate Strategy: For each of the two problems, derive a set of (system of) linear equations and use some online/offline tool to solve the system of linear equations.



(a)



Summary

- Probabilistic language models based on n-grams recover a surprising amount of information about a language
- Language models can have millions of features, so feature selection and preprocessing of data to reduce noise is important
- Text classification can be done with Bayes n-gram models or with any classification algorithms (machine learning)
- BM25 scoring function and PageRank use slightly different techniques to rank documents/pages