Natural Language Processing (NLP)

- An agent that wants to do knowledge acquisition needs to understand (at least partially) the ambiguous, messy languages that humans use.
  - Can be viewed from the point of view of specific information-seeking tasks
    - Text classification
    - Information retrieval
    - Information extraction
- Language models are used for the above tasks
  - Models that **predict** the probability distribution of language expressions

Formal languages

- A language can be defined as a set of strings
  - Example: “print(2 + 2)” is a legal program in the language Python, whereas “2)+(2 print” is not.
  - Since there are an infinite number of legal programs, they cannot be enumerated
- Formal languages
  - are specified by a set of rules called a grammar.
  - have rules that define the meaning or semantics of a program
  - Example, the rules say that the “meaning” of “2 + 2” is 4, and the meaning of “1/0” is that an error is signaled

Formal languages vs. natural languages

- Natural languages (ex: English, Spanish), cannot be characterized as a definitive set of sentences.
  - "Not to be invited is sad" is a sentence of English
  - "To be not invited is sad." is vague
- Better to define a natural language model as a probability distribution over sentences rather than a definitive set.
  - Do not ask if a string of words is or is not a member of the set defining the language
  - Ask for P(S = words), i.e., what is the probability that a random sentence would be words?

Formal languages vs. natural languages

- Natural languages are ambiguous.
- “He saw her duck” can mean
  - Either that he saw a waterfowl belonging to her
  - Or that he saw her move to evade something
- Hence, cannot speak of a single meaning for a sentence, but rather of a probability distribution over possible meanings.
- Natural languages are very large, and are constantly changing
  - Language models are, at best, an approximation
  - Start with simplest possible approximations and continue

n-gram model

- A written text is composed of characters—letters, digits, punctuation, and spaces
- Simplest language model is a probability distribution over sequences of characters
  - $P(c_{1:N})$ is the probability of a sequence of N characters, c_1 through c_N.
  - Example: In one web document collection, P("the") = 0.027 and P("zgq") = 0.000000002.
- A sequence of written symbols of length n is called an n-gram
  - 1-gram: unigram, 2-gram: bigram, 3-gram: trigram
- A model of the probability distribution of n-letter sequences is thus called an n-gram model
  - Can have n-gram models over sequences of words, syllables, or other units; not just over characters

n-gram model

- An n-gram model is defined as a Markov chain of order n−1
- In a Markov chain, probability of character c_i depends only on the immediately preceding characters, not on any other characters.
- Example: In a trigram model (Markov chain of order 2)

$$P(ci\,|c_{1:i\_1}) = P(ci|ci_{\_2:i\_1})$$

- Probability of a sequence of characters $P(c_{1:N})$ under the trigram model can be defined by first factoring with the chain rule and then using the Markov assumption

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

- A body of text is called a corpus (plural corpora)

n-gram model: uses

- Language identification
  - Given a text, determine the natural language it is written in
  - Relatively easy task
  - Even with short texts such as “Hello, world” or “Wie geht es dir,” easy to identify the first as English and the second as German.
- One approach to language identification
  - Build a trigram character model of each candidate language, $P(c_i | c_{i-2:i}\text{-}1, l)$,
    - Variable l ranges over languages
  - For each l, build the model by counting trigrams in a corpus of that language to obtain a model of

$$P(Text|Language)$$

n-gram model: uses

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

- Trigram model can be learned from a corpus, but what about the prior probability P (l)?
- May have some estimate of these values
  - Example: Selecting a random Web page
  - English is the most likely language and that the probability of Macedonian will be less than 1%.
- Exact numbers selected for these priors is not critical
  - Trigram model usually selects one language that is several orders of magnitude more probable than any other.

n-gram model: uses

- Character models are also used in spelling correction, genre classification, and named-entity recognition.
- Genre classification
  - Deciding if a text is a news, a legal document, a scientific article, etc.
  - Counts of punctuation and other character n-gram features are very important
- Named-entity recognition
  - Task of finding names of things in a document and deciding what class they belong to.
- Example: “Mr. Sopersteen was prescribed aciphex,”
  - Should recognize that “Mr. Sopersteen” is the name of a person and “aciphex” is the name of a drug
- Character-level models work well as they can associate
  - Character sequence “ex ” with a drug name
  - “steen ” with a person name

and identify words that they have never seen before

n-gram model: smoothing

- A language model should generalize well to texts it has not seen yet.
  - Example: “http” is not usually seen during training
  - Model should not claim that “http” is impossible
- Need to adjust language model so that sequences that have a count of zero in the training corpus will be assigned a small nonzero probability
  - Other counts will be adjusted downward slightly so that the probability still sums to 1
- The process of adjusting the probability of low-frequency counts is called smoothing.

n-gram model: Laplace smoothing

- In the unavailability of information, if a random Boolean variable X has been false in all n observations so far, then the estimate for P (X = true) should be 1/(n+2).
  - i.e., assumes that with two more trials, one might be true and one false.
- Laplace smoothing performs relatively poorly
  - Also called as add-one smoothing

n-gram model: Backoff model

- Start by estimating n-gram counts, but for any particular sequence that has a low (or zero) count, back off to (n − 1)-grams.
- Linear interpolation smoothing is a backoff model that combines trigram, bigram, and unigram models by linear interpolation.
- It defines the probability estimate as

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

where λ3 + λ2 + λ1 = 1

n-gram model: Backoff model

- Parameter values λ_i can be fixed, or can be trained with an expectation–maximization algorithm
  - Also possible to have the values of λ_i depend on counts
  - If high count of trigrams exists, then weigh them relatively more
  - If only a low count of trigrams, put more weight on the bigram and unigram models
- Two approaches to reduce variance in the language model
  - More sophisticated smoothing models
  - Gather larger corpus so that even simple smoothing models work well
- Note that the expression $P(c_i | c_{i-2:i-1})$ needs $P(c_1 | c_{-1:0})$ when $i = 1$, but there are no characters before c_1.
  - Can introduce artificial characters, for example, defining c_0 to be a space character or a special “begin text” character.
  - Or can use lower-order Markov models, in effect defining $c_{-1:0}$ to be the empty sequence and thus $P(c_{1|c-1:0}) = P(c_1)$

n-gram model: Model Evaluation

- How do we know what model to choose?
  - With so many possible n-gram models—unigram, bigram, trigram, interpolated smoothing with different values of λ, etc.
- Evaluate a model with cross-validation
  - Split the corpus into a training corpus and a validation corpus
  - Determine the parameters of the model from the training data
  - Then, evaluate the model on the validation corpus

- Evaluation can be a task-specific metric, such as measuring accuracy on language identification.
- Or can be a task-independent model of language quality
  - Calculate the probability assigned to the validation corpus by the model - the higher the probability the better
  - Inconvenient because probability of a large corpus will be a very small number, and floating-point underflow becomes an issue

n-gram model: Perplexity

- A different way of describing the probability of a sequence is with a measure called perplexity

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

- Perplexity can be thought of
  - as reciprocal of probability, normalized by sequence length
  - as weighted average branching factor of a model
- Suppose there are 100 characters in language, and language model says they are all equally likely.
  - Then for a sequence of any length, the perplexity will be 100
  - If some characters are more likely than others, and the model reflects that, then the model will have a perplexity less than 100

n-gram model: Perplexity

- Number of characters in language = 100
- Language model says all are equally likely
  - p(c) = 1/100
  - Assume a sequence length N
  - $P(c_{1:N}) = \left(\frac{1}{100}\right)^N$
- Perplexity = $\frac{1}{\sqrt[N]{\left(\frac{1}{100}\right)^N}} = 100$

n-gram word model

- Concepts are similar for word and character models
- Main difference is that in word models, vocabulary is larger
  - Vocabulary: Set of symbols that make up the corpus and model
- Only about 100 characters in most languages and they are usually very restrictive
  - Example by treating “A” and “a” as the same symbol, or by treating all punctuation as the same symbol
- Word models have at least tens of thousands of symbols
  - As it is not clear what constitutes a word
  - In English a sequence of letters surrounded by spaces is a word, but this is not true for Chinese
- Chance of a new word that was not seen in the training corpus: Need to be modeled explicitly in language model

Text classification using n-gram

- Given a text of some kind, decide which of a predefined set of classes it belongs to.
  - Examples: Language identification, genre classification, sentiment analysis, and spam detection
- Spam
  - Wholesale Fashion Watches -57% today. Designer watches for cheap ...
  - WE CAN TREAT ANYTHING YOU SUFFER FROM JUST TRUST US ...
  - Sta.rt earn*ing the salary yo,u d-eserve by o’btaining the prope,r crede’ntials!
- Ham
  - Practical significance of hypertree width in identifying more ...
  - Abstract: motivate the problem of social identity clustering: ...
  - Good to see you my friend. Hey Peter, It was good to hear from you. ...

Text classification using n-gram

- Word n-grams such as “for cheap” and “You can buy” seem to be indicators of spam
  - They have a nonzero probability in ham as well
- Character-level features also seem important
  - Spam is more likely to be all uppercase and to have punctuation embedded in words.
  - The word bigram “you deserve” is too indicative of spam, and thus wrote “yo,u d-eserve” instead.
  - A character model should detect this.
- Either create a full character n-gram model of spam and ham or handcraft features such as “number of punctuation marks embedded in words.

Text classification: Language models

- In the language-modeling approach, define one n-gram language model for P(Message | spam) by training on the spam folder, and one model for P(Message | ham) by training on the inbox.
- Then we can classify a new message with an application of Bayes' rule:

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

  where P(c) is estimated just by counting the total number of spam and ham messages

- Works well for spam detection, just as it did for language identification.

Text classification: ML Approach

- Represent the message as a set of feature/value pairs and apply a classification algorithm h to the feature vector X.
  - N-grams as features.
- Example: assume a unigram model (Bag of words model)
  - Features are the words in the vocabulary: “a,” “aardvark,” . . .,
  - Values are number of times each word appears in message
  - Feature vector large and sparse
  - If there are 100,000 words in the language model, then the feature vector has length 100,000
  - but for a short email message almost all the features will have count zero.
  - Order of the words is lost
  - Gives same probability to any permutation of a text

Text classification: ML Approach

- Higher-order n-gram models maintain some local notion of word order
  - Bigrams and trigrams: number of features is squared / cubed
- Can add in other, non-n-gram features
  - Whether a URL or an image is part of the message
  - An ID number for the sender of the message
  - Sender's number of previous spam and ham messages, etc.
- Choosing features is the most important task
  - More important than choice of algorithm for processing
  - As there is a lot of training data, the data can accurately determine if a feature is good or not.
- Necessary to constantly update features, as spam detection is an adversarial task

Text classification: ML Approach

- Once we have chosen a set of features, can apply any of the supervised learning techniques for text categorization
  - k-nearest-neighbors, support vector machines, decision trees, naive Bayes, and logistic regression.

Information Retrieval

- Task of finding documents that are relevant to a user's need for information
  - Example: Search engines
- IR system is defined by 4 characteristics

1. A corpus of documents
   - A paragraph, a page, or a multipage text
2. Queries posed in a query language
   - List of words, phrases, Boolean, non-Boolean operators
   - AI book, "AI book", AI AND book, AI bacics site:www.ai.org
3. A result set
   - Subset of documents found by IR system as relevant to query
4. A presentation of the result set
   - Simple ranked list of document titles, or more visual output

Boolean Keyword model

- Each word in the document collection is treated as a Boolean feature that is
  - True of a document if the word occurs in the document
  - False if word does not occur in document
- Query language
  - Language of Boolean expressions over features
- A document is relevant only if the expression evaluates to true
- Simple to explain and implement
- No guidance on how to order relevant documents for presentation
  - Only yes or no information available
  - Boolean expressions are unfamiliar to users

IR: Factors to be considered

- Term Frequency (TF)
  - Frequency with which a query term appears in a document
  - For the query [farming in Kansas], documents that mention "farming" frequently will have higher scores.
- Inverse document frequency of the term (IDF)
  - The word "in" appears in almost every document
  - Hence, it has a high document frequency and thus a low inverse document frequency
  - Hence, not as important to the query as "farming" or "Kansas."
- Third, the length of the document
  - Very long document may have all query words, but may be unrelated to query
  - Short document that mentions all the words is a much better candidate

IR: System evaluation

- System is given a set of queries and the result sets are scored with respect to human relevance judgments.
- Two measures used in scoring: Recall and precision
- Precision
  - Measures proportion of documents in result set that are actually relevant
- Recall
  - Measures proportion of all relevant documents in collection that are in result set
- Example : IR system (100 docs in corpus) receives a query

	In result set	Not in result set
Relevant	30	20
Not relevant	10	40

IR: System evaluation

- Precision is 30/(30 + 10) = 0.75.
  - The false positive rate is 1 - 0.75 = 0.25
- Recall is 30/(30 + 20) = 0.60.
  - The false negative rate is 1 - 0.60 = 0.40
- In large document collection (www), recall is difficult to compute
  - No easy way to examine every page for relevance
  - Can either estimate recall by sampling or ignore recall
- A summary of both measures is the F1 score, a single number that is the harmonic mean of precision and recall, 2PR/(P + R)
  - Compute F-measure/F-score for above example

Page Rank algorithm

- PageRank was invented to solve the problem of the tyranny of TF scores
  - If 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?
- Idea is that ibm.com has many in-links (links to the page), so it should be ranked higher
  - Each in-link is a vote for the quality of the linked-to page.
  - A web spammer can create a network of pages and have them all point to a page of his choosing, increasing the score of that page
- PageRank algorithm is designed to weight links from high-quality sites more heavily.
  - High quality site: One that is linked to by other high-quality sites.

Page Rank algorithm

- 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.
  - Constant d is a damping factor

Page Rank: Random surfer model

- Consider web surfer who starts at some random page and begins exploring.
  - With probability d (we'll assume d = 0.85) the surfer clicks on one of the links on the page (choosing uniformly among them)
  - With probability 1 − d she gets bored with the page and restarts on a random page anywhere on the Web
- PageRank of page p: Probability that random surfer will be at page p at any point in time
- Can be computed by an iterative procedure
  - Start with all pages having PR(p) = 1
  - Iterate the algorithm, updating ranks until they converge