Natural Language Processing

Michael Rose

In which we see how to make use of the copious knowledge that is expressed in natural language.

22.1 | Language Models

Languages can be vague. Thats why it is more fruitful to define a natural language model as a probability distribution over sentences rather than a definitive set. That is, rather than asking if a string of words is or is not a member of the set defining the language, we instead ask for $P(S = words)$ - what is the probability a random sentence would be words.

22.1.1 | N-gram Character Models

A model of the probability distribution of n-letter sentences is called an n-gram model. An n-gram model is defined as a **Markov chain** of order $n - 1$. For example, for a trigram model (a Markov chain of order 2), we have

$$P(c_i|c_{1:i-1}) = P(c_i|c_{i-2:i-1})$$

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

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

For a trigram character model in a language with 100 characters, $P(C_i|C_{i-2:i-1})$ has a million entries, and can be accurately estimated by counting character sequences in a body of text of 10 million characters or more. We call a body of text a **corpus**, from the Latin word for body.

One task for which these are suited is **language identification**. We can build a trigram character model of each candidate language, $P(c_i|c_{i-2:i-1}, l)$, where the variable l ranges over languages. For each l the model is built by counting trigrams in a corpus of that language (about 100,000 characters of each language are needed). That will give us a model of $P(Text|Language)$, but we want to select the most probable language given the text, so we use Bayes' rule followed by the Markov assumption to get the most probable language:

$$\begin{aligned} l^* &= \arg\max_l P(l|C_{1:N}) \\ &= \arg\max_l P(l)P(C_{1:N}|l) \\ &= \arg\max_l P(l) \prod_{i=1}^{N} P(c_i|C_{i-2:i-1}, l) \end{aligned}$$

The exact number for our priors is not critical because the trigram model usually selects one language that is several orders of magnitude more probable than any other.

Other tasks for character models include spelling correction, genre classification, and named-entity recognition.

22.1.2 | Smoothing n-gram Models

A major complication with n-gram models that the training corpus only provides an estimate of the true probability distribution. If our corpus gives us a 0% chance of a pattern occurring, this could cause problems down the road. We want our models to generalize well to texts that they haven't seen yet. Thus, we adjust our language model so that sequences that have a count of 0 in the training corpus will be assigned a small nonzero probability (and the other counts will be adjusted slightly downward so that the probability still sums to 1). The process of adjusting the probability of low-frequency counts is called **smoothing**.

The simplest type of smoothing is Laplace Smoothing which says that, in the lack of further information, if a random Boolean variable X has been false in all n observations, then the estimate for $P(X = true)$ should be $\frac{1}{(n+2)}$. He assumes that with two more trials, one might be true and one false. Laplace smoothing performs relatively poorly.

A better performing smoothing algorithm is a **backoff model**, in which we start by estimating n-gram counts, but 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 that combines trigram, bigram and unigram models by linear interpolation. If 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 $\lambda_3 + \lambda_2 + \lambda_1 = 1$. The parameters λ_i can be fixed, or they can be trained with an expectation-maximization algorithm.

22.1.3 | Model Evaluation

A way of describing the probability of a sequence is with a measure called **perplexity**, defined as

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

Perplexity can be thought of as the reciprocal of probability, normalized by sequence length. It can also be thought of as the weighted average branching factor of a model.

22.2 | Text Classification

We can represent a message as a set of feature / value pairs and apply a classification algorithm h to a feature vector X. We can make the language modeling and machine learning approaches compatible by thinking of the n-grams as features. The features are the words in the vocabulary, and the values are the number of times each word appears in the message. This makes the feature vector large and sparse. This representation is called the **bag of words** model. We can think of the model as putting the words of the training corpus in a bag and then selecting words one at a time. The notion of order of the words is lost; the model gives the same probability to any permutation of the text.

We can augment our data with other features to complement our bag of words model. Once we have chosen a set of features, we can apply any of the supervised learning techniques. Popular ones for text classification include knn, svms, decision trees, naive Bayes, and logistic regression.

22.2.1 | Classification by Data Compression

We can consider classification as a problem in **data compression**. A lossless compression algorithm takes a sequence of symbols, detects repeated patterns in it, and writes a description of the sequence that is more

compact than the original. Compression algorithms work by building dictionaries of subsequences of the text, and then referring to entries in the dictionary.

In effect, compression algorithms are creating a language model. The LZW algorithm directly models a maximum entropy probability distribution. To classify by compression, we can group together by class and compress each of them individually. Then, given a new message to classify, we append it to each class in turn and compress the result. Whichever class compresses best - adds the fewest additional bytes for the new message - is the predicted class. The idea is that a class specific message will tend to share dictionary entries with other messages of the same class, and thus will compress better when appended to a collection that contains the given class.

22.3 | Information Retrieval

Information Retrieval is the task of finding documents that are relevant to a user's need for information. An information retrieval system can be characterized by

1. A corpus of documents
2. Queries posted in a query language
3. A result set
4. A presentation of the result set

22.3.1 | IR Scoring Functions

A scoring function takes a document and a query and returns a numeric score; the most relevant documents have the highest scores. We will be looking at the **BM25 Scoring Function** which looks like tf-idf.

Three factors affect the weight of a query term:

1. The frequency with which a query term appears in a document (term frequency)
2. The inverse document frequency of the term (idf)
3. The length of the document

The BM25 algorithm takes all of these into account. We assume we have created an index of N documents in the corpus so that we can look up $TF(q_i, d_j)$, the count of the number of times word q_i appears in document d_j. We also assume a table of document frequency counts, $DF(q_i)$, that gives the number of documents that contain the word q_i. Then, given a document d_j and a query consisting of words $q_{1:N}$, we have

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

where $|d_j|$ is the length of the document d_j in words, and L is the average document length in the corpus: $L = \sum_i \frac{|d_i|}{N}$. We have two parameters, k and bm which can be tuned by cross validation.

IDF is the inverse document frequency of word q_i, given by

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

It would be impractical to apply the BM25 scoring function to every document in the corpus. Instead, systems create an index ahead of time that lists, for each vocabulary word, the documents that contain the word. This is called the **hit list** for the word. Then when given a query, we intersect the hit lists of the query words and only score the documents in the intersection.

22.4 | Information Extraction

Information extraction is the process of acquiring knowledge by skimming a text and looking for occurrences of a particular class of object and for relationships among objects.

22.4.1 | Finite-State Automata for Information Extraction

The simplest type of information extraction system is an **attribute based extraction system** that assumes that the entire text refers to a single object and the task is to extract attributes of that object. To extract from these, we define a template for each attribute we would like to extract. The template is defined by a finite state automaton, the simplest example of which is the **regular expression**.

One step up from attribute based extraction systems are **relational extraction systems**, which deal with multiple objects and the relations among them. A relational extraction system can be built as a series of **cascaded finite-state transducers**. The system consists of a series of small, efficient finite-state automata, where each automaton receives text as input, transduces the text into a different format, and passes it along to the next automaton.

A typical relational-based extraction system is FASTUS, which consists of 5 stages:

1. Tokenization
2. Complex word handling
3. Basic group handling
4. Complex phrase handling
5. Structure merging

22.4.2 | Probabilistic Models for Information Extraction

When information extraction must be attempted from noisy or varied input, simple finite-state approaches fare poorly. In these situations it is better to use a probabilistic model, of which the simplest is the Hidden Markov Model. Recall that a HMM models a progression through a sequence of hidden states x_t, with an observation e_t at each step. To apply HMMs to information extraction, we can either build one big HMM for all the attributes or build a separate HMM for each attribute.

HMMs have two big advantages over FSAs for extraction:

1. HMMs are probabilistic, and thus tolerant to noise.
2. HMMs can be trained from data

Once the HMMs have been learned, we can apply them to a text, using the Viterbi algorithm to find the most likely path through the HMM states. One approach is to apply each attribute HMM separately. This is appropriate when the extraction is sparse - when the number of extracted words is small compared to the length of the text. A second approach is to combine all the individual attributes into one big HMM, which would then find a path that wanders through different target attributes. Separate HMMs are better when we expect just one of each attribute in a text and one big HMM is better when the texts are more free-form and dense with attributes.

22.4.3 | Conditional Random Fields for Information Extraction

One issue with HMMs for information extraction is that they create a generative model, which may be more than is needed for the task at hand. We could instead use a **discriminative model**, which models the conditional probability of the hidden attributes given the observations (the text). Given a text $e_{1:N}$, the

conditional model finds the hidden state sequence $X_{1:N}$ that maximizes $P(X_{1:N}|e_{1:N})$. By modeling this directly, we don't need the independence assumptions of the Markov model - we can have an x_t that is dependent on x_1. A framework for this type of model is the **conditional random field**, which models a conditional probability distribution of a set of target variables given a set of observed variables.

One common structure is the **linear chain conditional random field** for representing Markov dependencies among variables in a temporal sequence. Thus HMMs are the temporal version of naive Bayes models, and linear chain CRFs are the temporal version of logistic regression, where the predicted target is an entire state sequence rather than a single binary variable.

Let $e_{1:N}$ be the observations (e.g. words in a document) and $x_{1:N}$ be the sequence of hidden states (e.g. prefix, target, and postfix states). A linear chain conditional random field defines a conditional probability distribution

$$P(X_{1:N}|e_{1:N}) = \alpha e^{\sum_{i=1}^{N} F(X_{i-1}, X_i, e, i)}$$

where α is a normalization factor, and F is a feature function defined as the weighted sum of a collection of k component feature functions:

$$F(X_{i-1}, X_i, e, i) = \sum_k \lambda_k f_k(x_{i-1,x_i,e,i})$$

where the λ_k parameter values are learned with a MAP estimation procedure that maximizes the conditional likelihood of the training data. The feature functions are the key components of a CRF. An example of a feature function is

$$f_1(x_{i-1,x_i,e,i}) = 1 \quad \text{if} \quad x_i = \text{Speaker and } e_i = \text{Andrew}$$
$$0 \text{ otherwise}$$

This feature is binary, but in general a feature function can be any real valued function. The CRF formalism gives us a great deal of flexibility in defining the types of features we would like to include.

22.5 | Summary

- Probabilistic language models based on n-grams recover a surprising amount of information about a language. They can perform well on such diverse tasks as language identification, spelling correction, genre classification, and named entity recognition.
- These language models can have millions of features, so feature selection and preprocessing of the data to reduce noise is important.
- Text classification can be done with naive Bayes n-gram models or with any of the classification algorithms we have previously discussed. Classification can also be seen as a problem in data compression.
- Information retrieval systems use a very simple language model based on bags of words, yet still manage to perform well in terms of recall and precision on very large corpora of text. On web corpora, link analysis algorithms improve performance
- Question answering can be handled by an approach based on information retrieval, for questions that have multiple answers in the corpus. When more answers are available in the corpus, we can use techniques that emphasize precision rather than recall.
- Information extraction systems use a more complex model that includes limited notions of syntax and semantics in the form of templates. They can be built from finite state automata, HMMs, or conditional random fields, and can be learned from examples.
- In building a statistical language system, it is best to devise a model that can make good use of available data, even if the model seems overly simplistic.