Introduction to Natural Language Processing

Druhan Rajiv Shah

Spring - 2023

Contents

Course Structure

- 20% Assignment
- 40% Project
- 10% Midsem
- 10% Quizzes
- 20% Endsem

There used to be Vivas, but not anymore. (too many damn students lmao)

Objectives

- Demonstrating knowledge of the fundamental building blocks of NLP
- Apply NLP techniques for classification, representation and parsing
- Demonstrating the use of Dense vector representaton for NLP
- Explaining concepts behind distributed semantics
- Discussing approaches to global and contextual semantic representation
- Applying the above concepts for fundamental NLP tasks

Syllabus breakdown

1. Stages of NLP
2. Morphology, POS tagging, Chunking
3. Syntax
4. Distributed semantics
5. Contextual distributed semantics

Admin stuff For deadlines and all that, there will be a sort of Course Almanac shared later, based entirely on the Institute Almanac. Attendance will be managed by the Institute.

1 Introduction

Potential motivations

- Fascination with existing models like ChatGPT, Lambda
- Translation systems
- Strengthen theoretical base (?)
- Interesting applications
- To create something (AI, perhaps?)

Unknown option: smooth-scroll

Language Language has three main modalities (text, speech, and sign), and each of them overlaps with each other (a lot going on tbh).

This course will mainly cover the text side of things.

What's going on? Processing natural language has been the final target of Computer Science since the very beginning (ever since Ada). But Computer Science is just doing math but quickly. Math can pretty much be attributed entirely to Panini, who worked primarily on Logic. Converting Language into a Logical form, providing structure is what he's mainly known for.

Enter Linguistics, stage left.

Someone much later tried to create a much more formal structure of languages. That is now known as either Formal Language and Automata Theory, or Theory of Computation (à la Chomsky).

Theory of Computation Four levels of languages:

1. Regular Language
2. Formal/Context-free Language
3. Recursive Language
4. Recursively Enumerable Language

This leads to much more fundamental problems like the Halting problems or Satisfiability problems.

iNLP time This course is not about modern, State-of-the-Art models, but about the theoretical base that those models are built on.

This is going to be a very fast-paced course.

The focus of this course is not coding, but there will be coding-based assignments and all.

History lesson Modern computer science is considered to begin around Alan Turing's time. So pretty much most of Computer Science is based on code-breaking. Turing's code-breaker was gifted by the British Intelligence Services to the CIA and a company was born out of it. Code-breaking was pretty much all that the CIA did back then. The commpany that it collaborated with was AT&T (Bell Labs back then), which did most of the research on code-breaking which led to the development of NLP. The code-breaking work led to the development of Information Theory, which led to Coding Theory. Some of the lead singers in this concert were Claude Shannon and Huffman.

Monitoring communication is essentially what led to the development of Computer Science itself, so it makes sense that Natural Language is a major focus even now.

Language shenanigans We mainly deal with language in a sequential manner (as opposed to something like Arrival). So time(order of some sort) has to be considered regardless of language. For years, most NLP models handled text sequentially, until the advent of Transformers, which process text as a group, sort of parallelly. The sequential treatment made sense because speech is dealt with that way.

Beyond the grammar of language, there is the layer of meaning, which is a lot harder to deal with (see Dense vector representation and distributed semantics as opposed to Formal logic representation).

The '97 paper on distributed semantics marked the beginning of the modern age of Language processing, which is primarily meaning-based.

2 'Nuff Philosophy

If we disregard the socio-political factors for the development of this field, there is a certain fascination for the sheer complexity of natural language as we speak it, which we take for granted only because we use it all the time.

So what makes a language a language? Prairie dogs have developed modes of communication to convey messages of whether the thing approaching the den is a threat or not. In fact, these messages are extremely detailed. Crows and parrots are well known to be capable of developing intelligent communication.

There are things that language allows us to do in communication, which when we think about it, are extremely fascinating.

Essentially, at its core, NLP is an attempt to understand how the human brain works in dealing with language. NLP is all about computationally understanding language in order to use it for something else.

Processing of Natural Language

- Speech and text processing
- Identification of language phenomena (grammar, writing systems, etc.)
- Understanding language content (semantics, pragmatics, sentiment, etc.)

Systems Consider the OS definition of a system. A system is when multiple parts or components interact with each other to perform functions.

By this logic, language is a very complex system, consisting of several components. The typological structures that we are familiar with, can be claimed to be there for a reason. These structures incentivize us to structure our thoughts in a way that makes it efficient to convey the information necessary for communication (an organically developed protocol, so to speak). These are sort of analogous to evolution, where there are some things that just make sense, which is what ends up becoming the norm, due to selection. This is also what most likely happened with language; the documentation of the structures that we study is only a post-development analysis.

Language as we know it is not bound by any physical laws, the only mechanism that restricts language is the mode of communication, be it sound or speech. In the case of sound, the human vocal tract is only able to produce so many sounds.

Oops, more philosophy One can even say, the most chaotic system possible is human society; Asimov's works almost all verify and build on this idea. Of course, one can apply the physical definition of chaos to this system, and calculate entropy and all that, but that's another rabbit hole entirely.

Okay, back to NLP Simple keyword matching is the most basic method of dealing with things. Matching documents that contain certain keywords is something literally any layperson can do, though.

2.1 Generations of NLP

Expert systems Back in the good ol' days, there were all hand-crafted knowledge-engineering systems were all the rage (Expert systems so to speak, see MDL notes). There was significant linguistic knowledge, even computational knowledge, but all systems were knowledge-based. So pretty much rule-based systems as in CL2.

Machine Learning The next generation decided that this built up too much a priori cost and was too labour intensive. Machine Learning started with the idea of learning rules. Thus, we can use ML systems to learn those rules to set up the rule-based systems on their own. Step 4: Profit.

This led to an increase in the use of disjunctive logic as opposed to conjunctive logic.

An ML system allows us to abstract things out and can be applied in far more fields than rule-based systems, but can have possibilities of the model learning rules that are not actually valid. This requires the manual introduction of a bias in the model. The easiest way to implement such disjunctive logics with bias, is probabilities. Note that conjunctive probabilities can simply be comverted to disjunctive using complements (ggez). Sometimes these disjunctive conversions are made in order to make computation easier or even possible.

Again, pay close attention to MDL for this.

Rise and fall of Neural Networks If all this probabilistic trickery ca be simplified into a function based on some parameters, then all one needs to do is to devise that function mapping a vector of attributes onto a class space. Perceptron-based learning handles linear function approximations of pretty much everything. Marvin Minsky pointed out that this is infeasible for several reasons. This ended up killing pretty much all work in the Neural Network field.

Most of language technology, however, was developed in this era. It would be a great disservice to ourselves.

Unsupervised Semantics The most recently developed field of learning systems, are especially applicable in language systems because they are entirely based on semantics. The attempts to represent ideas using mathematical vectors was inefficient for reasons that will be covered later.

2010 saw the development of Word2Vec technologies (as opposed to word embeddings), which blew this field up suddenly.

2.1.1 Learning styles

Types of ML models There's either Supervised, Unsupervised, or Self-Supervised learning. Supervised is based on manual approval or disapproval of classifications. This could be done by pre-annotated data as well.

Note that pretty much every problem can be boiled down to some sort of Classification problem.

Self-supervision? You mean slacking off? Not quite a misnomer. It's all about converting unsupervised data into supervision, sort of. Considering one plain sentence with no expert supervision, then we can't get a lot from it on its own. But, by replacing a word with a class and we can obtain some pseudo-classification problem-like situation. This can be done only if the model 'understands' what the surrounding words mean. Here, the meaning of 'understand' kinda gets messed up, but don't worry about it.

Now, given some more data, the model can actually compute a distribution for the possible replacements for the removed word (This is called a Mask prediction model).

2.2 Structure of language

- Words
- Syntax
- Semantics
- Discourse
- **Bonus:** applications exploiting each of the above

Despite dictionary propaganda, words themselves don't have meaning, they obtain their meaning from context. Similarly, each of these fields relies deeply on each other.

A Challenger approaches Ambiguity can be context-induced or structural (despite dictionary propaganda), the latter of which can be context-interpreted. PP-attachment ambiguity is an example of structural ambiguity. Some such ambiguities can't even easily be resolved by humans either! Headlines are even worse (Foot Heads Arms Body :P).

Language allows for a lot of creativity, which is what computers pretty much hate. After all, isn't creativity just a resolution of entropy, which would lead to this being a chaotic system?

Internet shenanigans The advent of the internet boom, especially considering the incrased reach off late, has posed several challenges in language processing. One such challenge is language evolution which is caused due to this increased reach. This also leads to another challenge which is scale. There is simply way too much data on the internet to reasonably handle as of now.

2.3 Finally, Natural Language Processing

This course We will be progressing from Morphology and Syntax all the way upto touching a bit of Discourse analysis. In this course, we will be focusing on text-based NLP, ignoring speech and sign.

3 Morphology and Syntax

Tokenization Recall CL2. Consider the sentence "This is my brother's cat." Considering "brother's" as its own token is dangerous because the factor of 'brother' is completely lost in it. However, not all apostrophe-ending words are to be considered similar.

Punctuated with some Writing Space-based tokenization is infeasible because there are several languages (Traditional Mandarin, Modern Thai) that don't even use spaces. (Ni-ho-n-go-ga-wa-ka-ri-ma-su-ka?) Some languages use spaces between adpositions, while some don't (Hindi vs Gujarati lmao). The existence of Sandhi or rendaku (compounding, even) alone is sufficient example for how spacing is not absolutely necessary. Language on its own does not use punctuation. There are extra-syntactic features, but there is no strict punctuation as is. In fact, a person can use any pause length or frequency as they wish.

Old Sanksrit poetry uses only two punctuation marks because of the oral tradition that led to the advent of Sanskrit as a language. In fact, writing as a system are a very recent phenomenon as compared to language as a whole. One only needs punctuation to indicate pauses. Essentially, punctuation is just a printer's device.

Post Scriptum note: Punctuation can be treated as an annotation over the text for the reader to recreate the speech more accurately. Their job is to convey some aspects of extra-linguistic information.

Terminology

Type is a word which might be present in its root form in the Dictionary

The set of all types in a language is its vocabulary

A high type-token ratio indicates that a language has a rich, variegated morphology. This could even imply agglutinativity. Marathi has 97 freaking morphological forms for verbs lmao.

Is tokenization difficult? When it comes to tokenization, types are no longer of any concern to us. One can use rule-based tokenization, but that's too messy. One can let the machine itself learn how tokens are split, which has its own challenges. Hyphenation, abbreviations, numbers, dates and punctuation can pose problems on their own.

Regular expressions are rampant here :P

3.1 Language Models

Models are a set of parameters that numerically describe reality. So it's essentially just a bunch of numbers.

Now, models are not necessarily perfect. Rather, they aim to describe reality as opposed to strictly being perfect. The process of modelling is constantly refining those models.

This automatically destroys the 'is AI sentient' argument :P

Good enough models can attempt at omniscience, but it's just a farce.

3.1.1 N-grams

What can we do with raw data? From raw data, can we learn a set of parameters that performs a certain task? Claude Shannon proposed that given a sentence, can we how likely it is to belong to a certain language? Now, considering sentence length, we don't use long sentences very often. This would mean that the longer the sentence, the less likely it is to belong to a language (Enter Zipf). Simply considering probabilities of groups of words or sentences is sorta obviously messy.

Let's not Mark-this-ov Here, we make a 'Markov assumption': A word depends on the 'prior local context', which is the previous few words. The length of the 'prior local context window', say n, would make this an n-gram model (3-length window would make this a trigram and so on).

From a linguistic perspective, we already know that there is a structure. However, computationally, we only see strings of words. And in this context, the words can only depend on their surroundings.

By looking at bad enough examples, we can see that given long enough sequences, the data can get very unreliable. Small counts are usually very unreliable, because given large enough n, we can easily get absurd probabilities.

However, there is another gain from this. Were language truly random, probabilities would give absolute answers regardless of n, because true randomness could lead to any possible sentence, equally frequently. This is an argument against the chaotic nature of language.

Also, computationally, given large enough data, the number of bins in the n-gram models grows exceptionally large. We also need to include combinations that we haven't seen in the data before, but are possible in the language in order to model it effectively.

Mar-kons

- The sparseness of words and word clusters can lead to very spread out distributions, which can lead to messy and inaccurate predictions.
- Dealing with unknown words in the prediction output can lead to straight up zero probabilities, which is messed up, yo.

3.1.2 Evaluation and Data Sparsity

Rather perplexing, innit? Perplexity and Entropy are usually the measures used to evaluate how good a model is. Specifically, it measures how well your model fits a corpus, once it's built.

Both of these are based on Information Theory, which deals with chaotic systems and measuring the degree of chaos in a system. Entropy is a measure (heuristically) of the extent of chaos in a system, while perplexity is the measure of how much confusion a model can have when trying to determine the outcome of a system. (See how vague this is?)

Specifically, Entropy is the lower bound on the number of bits needed to encode information (say, about the n-gram likelihood). Perplexity is the average number of choices that can be made, weighted by their probabilities.

$$\text{Entropy } H(X) = \mathbb{E}[-\log_2 \mathbb{P}(X)] = -\sum_{x=0}^{n} p(x) \log_2 p(x)$$

$$\text{Perplexity } PP(X) = 2^{H(W)}$$

Perplexity being a measure of the number of choices to be made, can be used as a metric for determining how well a model approximates something. For example, when matching a model to someone's language style, the fewer the choices required to determine the next possible word in a sentence, the closer the match, so a lower perplexity would indicate a better match.

3.1.3 Zipf it!

When the population of several physical, natural and social systems (like words in a language, or species in a ecosystem) is plotted with their frequencies, the resulting plot is a reciprocal-like curve (Zipf distribution). Thus, when plotted on a log-log graph, the resulting curve is a straight line.

Another way of putting this, is that when ranked (1-indexed), the product of the frequency of a word and its rank is approximately a constant.

Note: Do not confuse this with Dirichlet's Theorem which considers the residue classes of primes to be evenly distributed.

This makes sense for languages, because naturally, function words would be way more frequent than, say, proper nouns. Anaphora and Specifiers make up some of the most frequent words in English at least.

At this point, bringing up the issue of deciding whether to include stopwords in models is necessary. Sometimes, when the functions played by stopwords are not necessary for our model, it makes sense to exclude them, but when considering n-gram models, it makes sense that stopwords be included since they would affect the sentence formed due to their large probabilities.

3.1.4 Back to N-grams

Now, when considering sparse distributions or while encountering new words to predict, then the model can either mess up or fail entirely. This needs to be countered.

1. We can reduce the value of n, possibly down to 1, but that cannot overcome the issue of new words.
2. We can define probabilities as $\frac{f_w+1}{\sum_w (f_w+1)}$ (Again, very ad-hoc)
3. We can group words into categories, each with their own probabilities, and then uniformly distribute the lexicon of that category.

4. We can assign a function of n-gram probabilities of the context and all possible follow-ups to the probability of the context with a new word. This covers the issue of new words, but doesn't deal with unseen contexts instead of new words.

5. We can use idea 2, but with $\lambda > 0$ instead of 1.

Aight buddy, back it up now Idea 1 combined with Idea 4, implemented with considering a certain function of the counts without explicitly considering the empirical probability definition, can actually work out. Such a model is called a Back-off model. This is because every time a new word is obtained in the context of a word, we back off down to a smaller n value, and for new words to be predicted, we follow Idea 4.

In a way, we are also using the Idea 5 in the sense that we aren't using the conventional definition of empirical probabiliy which uses purely word counts.

Damn that's smooth There are always some sentences, however, that a human would find very natural, that a model would not even consider. This is because the model only considers immediate surroundings and that too linearly. That is not how a human would read it (Recall ITL1 and CL1).

Granted, back-off models are computationally intensive, because we would need to compute multiple large distributions, one for each n, instead of simply one large distribution. Now, this can be simplified using tricky computational methods. (To be updated) Potentially, interpolation can be used to simplify this.

Such improvements in models, including interpolation, are called smoothing techniques. See:

- Laplace smoothing
- Lidstone's Law
- Held-out estimation
- Good-Turing estimation

3.1.5 Smoothing techniques

We shall define some notation here:

- We define V to be the vocabulary size
- w_{i-n+1}^{i} is the sequence of n tokens $w_{i-n+1}, w_{i-n+2}, \ldots, w_i$
- $c(w_{i-n+1}^{i})$ is the frequency of that particular n-gram.
- When we use $P(w_{i-n+1}^{i})$ we mean $P(w_i|w_{i-n+1}^{i-1})$. (Haha abuse of notation go /ʙ:/)
- $N_{1+}(w_{i-n+1}^{i-1}\bullet) = \left|\{w_i : c(w_{i-n+1}^{i}) > 0\}\right|$ is the number of unique words that follow the occurrence of the given $(n-1)$-gram.
- Similarly, $N_{1+}(\bullet w_{i-n+2}^{i}) = \left|\{w_{i-n+1} : c(w_{i-n+1}^{i}) > 0\}\right|$ is the number of unique 'histories' preceding the given $(n-1)$-gram

Add one Laplace to the mix Idea 2 and 5 as mentioned earlier seem quite reasonable. In fact, they are well-known methods, the first (Idea 2) being called Add-one smoothing, or Laplace smoothing, where the frequency of everything is increased by one, so the unknown terms have a nonzero probability. Thus, we would have

$$P_l(w_{i-n+1}^{i}) = \frac{c(w_{i-n+1}^{i}) + 1}{c(w_{i-n+1}^{i-1}) + V}$$

Put a Lid on it! The problem with this is that the proportion of the probabilities assigned to new words is simply too much. The solution? Add something less than one instead. This idea (Idea 5) is called Lidstone smoothing where instead of one, there is some other small value added, say λ. Thus, we have

$$P_l(w_{i-n+1}^i) = \frac{c(w_{i-n+1}^i) + \lambda}{c(w_{i-n+1}^{i-1}) + V \cdot \lambda}$$

With $\mu = \frac{c(w_{i-n+1}^{i-1})}{c(w_{i-n+1}^{i-1})+V\cdot\lambda}$, we can also describe it as

$$P_l(w_{i-n+1}^i) = \mu \frac{c(w_{i-n+1}^i)}{c(w_{i-n+1}^{i-1})} + (1-\mu)\frac{1}{V}$$

This is a surprise tool which will help us later.

Incidentally, $\lambda = 0$ gives us the MLE distribution and $\lambda = \frac{1}{2}$ gives us the expected values of the likelihoods: what is called the Expected Likelihood Estimator (based on the Jeffrey-Perks Law)

Holding out on probabilities Held-out estimation is called so because it deals with splitting the given data into two parts, one of which pretends to approximate real-life data to test the accuracy of our training data. This is done to ensure the model is not too dependent on the training data.

It also works under the assumption that n-grams with similar frequencies behave similarly.

$$P_{ho}(w_{i-n+1}^i) = \frac{T_r}{NN_r}$$

Where N_r is the number of n-grams with frequency r in the training data and .

$$T_r = \sum_{c_{tr}(w_{i-n+1}^i)=r} c_{ho}\left(w_{i-n+1}^i\right)$$

Now, we also define the reestimated frequency $r^* = \frac{T_r}{N_r}$ Thus, the probability is calculated keeping both training and held-out data in mind.

Delete this! But which of these works better? What happens if the two of them are switched?

This can work if we consider both and simply average them out. One way is $\frac{r_{01}^* + r_{10}^*}{2}$. But this is a little too weird. Jelinek and Mercer (1985) recommended

$$P_{Del}(w_{i-n+1}^i) = \frac{T_r^{01} + T_r^{10}}{N(N_r^{01} + N_r^{10})}$$

This is called deleted estimation.

Turing's kinda good no? It is natural for data for some frequency bins to be rather empty all of a sudden. Now, what if we consider words that appear very rarely and treat them as we would treat other equally rare words? Isn't a word that appears only once pretty much the same as one that doesn't appear at all?

More specifically, since words with similar frequency behave similarly, we can use this principle to 'smooth out' similar frequencies, so to speak (Similar to Deleted estimation).

Once there is more structure to the data, this estimation becomes less and less valid. However the claim is that this becomes valid for larger and more natural data, so to speak.

In this estimation technique, we calculate the adjusted frequency as

$$r^* = (r+1)\frac{E(N_{r+1})}{E(N_r)}$$

And then we consider the adjusted probability as $\frac{r^*}{N}$

Assignment 0.5

Read the paper Simple Good-Turing Algorithm and prepare an around-1-page report.

Yeah that's right, back it up Back-off estimation is essentially changing the value of n to deal with unseen words as mentioned earlier (See paragraph **Aight buddy, back it up now**). It can also possibly be increasing the value of n to deal with shenanigans.

Katz (2019) We can redefine the Katz back-off probabilities as

$$P_k(w_{i-n+1}^i) = \begin{cases} \alpha \cdot \frac{c(w_{i-n+1}^i)}{c(c_{i-n+1}^i)} & \text{Counts are greater than some } k \\ (1-\alpha) \cdot P_k\left(w_{i-n+2}^i\right) & \text{Otherwise} \end{cases}$$

Linear Interpolation This ends up working really well provided it reserves a probability mass for unseen items. It can also be generalised and the paradigm applied in other smoothing contexts. Note that here, each of the λ_i need to be learned separately.

$$P_{li}(w_{i-n+1}^i) = \lambda_1 P(w_n) + \sum_{k=2}^{n-1} \lambda_k P(w_{i-k+1}^i)$$

This is Witten weally Bell Witten-Bell smoothing manages to capture the robustness of interpolation while also handling the intuitiveness of back-off models. It also considers the situations where for a large number of possibilities, the history is either very 'productive' or the new word itself is prone to many possible histories.

Additionally, Witten-Bell smoothing is an instance of the recursive interpolation method as visible in the definition given below.

$$P_{wb}(w_{i-n+1}^i) = \lambda_{w_{i-n+1}^{i-1}} P_{ml}(w_{i-n+1}^i) + (1 - \lambda_{w_{i-n+1}^{i-1}}) P_{wb}(w_{i-n+2}^i)$$

The n-gram interpolation weight λ is defined as:

$$\lambda_{w_{i-n+1}^i} = \frac{c(w_{i-n+1}^{i-1})}{c(w_{i-n+1}^{i-1}) + N_{1+}(w_{i-n+1}^{i-1}\bullet)}$$

Watch me Kneser-Ney This smoothing procedure derives from the idea of Absolute Discounting where we intentionally reduce the probabilities of the appreared units in order to account for unseen units. The idea of absolute discounting is to discount a fixed value from all the frequencies and use those to account for unseen tokens. That is,

$$P_{ad}(w_{i-n+1}^i) = \frac{\max(c(w_{i-n+1}^i) - d, 0)}{\sum_{w_i} c(w_{i-n+1}^i)} + \lambda P_{ad}(w_{i-n+2}^i)$$

Now, Church and Gale (1991) felt that $d = 0.75$ is a good enough value, and it seems to be so, but Ney et al. (1994) set $d = \frac{n_1}{n_1+2n_2}$ as the discount value where n_i is the number of tokens with frequency i in the training data.

Now, Kneser-Ney discounting works on a similar principle, except that we consider the interpolated probability to be some 'continuation' probability, that is 'how likely is this unigram to appear as a new continuation?'

Thus, we define the continuation probability to be

$$P_{cont}(w_i) = \frac{N_{1+}(\bullet w_i)}{\sum_{w_i} N_{1+}(\bullet w_i)}$$

Observe how for an empty string the probability becomes a constant value $\frac{1}{V}$. This is a surprise tool which will help us later.

Thus, we would have the final recursive definition of the Kneser-Ney probability for bigrams as

$$P_{kn}(w_{i-1}^i) = \frac{\max(c(w_{i-1}^i) - d, 0)}{\sum_{w_i} c(w_{i-1}^i)} + \lambda_{w_{i-1}} P_{cont}(w_i)$$

We can generalize this to higher orders as

$$P_{kn}(w_{i-n+1}^i) = \frac{\max(c_{kn}(w_{i-n+1}^i) - d, 0)}{\sum_{w_i} c_{kn}(w_{i-n+1}^i)} + \lambda_{w_{i-n+1}^{i-1}} P_{kn}(w_{i-n+2}^i)$$

Note that we have used a different count function here. We define the new Kneser-Ney count function for an n-gram as simply $c(w_{i-n+1}^{i})$ for the highest order andthen $N_{1+}(\bullet w_{i-n+1}^{i})$ for lower orders. Additionally, we have the normalizing constant

$$\lambda_{w_{i-n+1}^{i-1}} = \frac{d \cdot N_{1+}(w_{i-n+1}^{i-1}\bullet)}{\sum_{w_i} c_{kn}(w_{i-n+1}^{i})}$$

Now, the value of the discounting term matters greatly here. Chen and Goodman (1998) gave a modification to this smoothing technique that performs really well, which is to use three different discount terms for n-grams with count 1, 2, and above 2.

3.2 POS Tagging and Chunking

In all that we have done so far, we have not considered the underlying structure of language. In a lot of examples above, we treated all words in a vocabulary as equivalent even though some words would never occur in certain contexts. However, this is not necessarily the case. So basically, words can be categorized, sometimes deeper than simply POS.

Some words are more equal than others Here, the number of arguments taken by verbs also comes into the picture since it affects which verbs can take the place of a blank depending on the number of arguments (Recall frames from CL2). Also based on X-bar theory, the choice of complement and adjunct also plays a role in determining the frame itself and hence the choice of verb or argument. Semantic rules like the existence of an ACTOR and a PATIENT in a sentence are also major players. Even within arguments, some arguments can only accept some kinds of noun phrases.

I completely agree with this statement The relations between parts of a sentence can be either grammatical or semantic. Semantic relations have different types and names. Grammatical relations (restrictions too perhaps?) are called agreements. These, along with meaning-based relations (semantic) lead to how complex and hard to generate automatically a sentence can possibly be. Whenever we see a well-formed sentence, it agrees with all these rules.

Thus, language or any text has significantly more depth than any model can deal with. There is always some selective restriction being placed by the surrounding words and grammar on a word or a part in a sentence. The scaffolding for a section exists, but we cannot be sure it is strictly grammatical.

Tag, you're it! Tagging a sentence's words with their parts of speech is a possible annotation of a text to convey a part of the structural information embedded in the sentence. This is not the only possible one, it only conveys part of the information, but it is one for sure. It also ends up being a sort of disambiguation.

Digression to Austin, TX Language can be boiled down to simply a series of reports of actions. Specifically, we can say each sentence is at its core, simply a verb (Enter copulae). Independent of Speech Acts, every sentence conveys a pure verb and its arguments, action behaviour of state change notwithstanding.

Disambiguating the disambiguation In a lot of contexts, words can be tagged with multiple possible POSes and determining which one applies in which context is a challenge. (Okay, who am I kidding? It's literally the only challenge here.) For example a token like "that" can behave like either a demonstrative (_J), a pronoun (_N) or a complementizer (_F), depending on the context it appears in.

One way of 'solving' this is to simply assign the most common tag (Why would you do this?).

3.2.1 Formalizing this problem

The entire task in POS tagging is essentially a sequence labeling problem. That is, for a given sequence $W = (w_1, \ldots, w_n)$ we need to produce a tag label sequence $T = (t_1, \ldots, t_n)$ such that they 'belong', *i.e.* we maximise $P(T|W)$.

At first glance, it looks like we can just learn $P(T|W)$ from the data.

Now, it's quite easy to see why that just doesn't work.

What does work then? A standard probabilistic model doesn't work for two reasons:

- The limited horizon of the previous $(n-1)$ tokens which is not enough for a reasonable deduction.
- The time invariance, basically $P(X_t = i|X_{t-1} = j) = P(X_1 = i|X_0 = j)$

Instead, since we are considering states over time and the transitions between them, we can instead consider a Markov Model. While this doesn't quite obviously deal with the first problem, it does provide a very convenient framework to deal with the second.

The formal definition of a Markov chain is the tuple (Q, A, π), where $Q = \{q_1, \ldots, q_n\}$ is the set of states, $A = [a_{ij}]_{n\times n}$ is the transition probability matrix and $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ is the initial probability distribution.

Y'all better hide The purpose of a Markov Model in general is to study state transitions. Now, we can use a specific type of model called a Hidden Markov Model, which deals with 'hidden' states, which in our case are the POS tags because they aren't technically observed, but are inferred implicitly and in fact do affect the result of the tagging prcess. We can define a Hidden Markov Model as a tuple (Q, A, B, V, π), which generates an output sequence of T observations $O = \langle o_1, \ldots, o_T \rangle$. The tuple consists of:

$Q = \{q_1, \ldots, q_n\}$	The set of states
$A = [a_{ij}]_{n\times n}$	The transition probability matrix
$B = \langle b_1(o_t), \ldots, b_n(o_t) \rangle$	The emission probabilities of an observation o_t emitting from a state i
$V = \{v_1, \ldots, v_V\}$	The output alphabet
$\pi = \langle \pi_1, \ldots, \pi_n \rangle$	The initial probability distribution

Assume spherical cow Along with the standard Markov assumption (the transition from one state to another only depends on the current state and not the history), there is another assumption made in an HMM, that is the assumption of Output Independence:

> The probability of an output observation is only dependent on the state that produces it and none other.

That is,

$$P\left(o_i|q_1^T, o_1^T\right) = P(o_i|q_i)$$

This is in a way similar to the Markov assumption, but formally it needed to be distinguished since it deals with observations and not states. For heuristic purposes, they can be considered the same.

Ay, pal you got a problem? An HMM can essentially be described by three problems:

1. Given an HMM $\lambda = (A, B, \pi)$ and an observation sequence, determine the likelihood $P(O|\lambda)$.
2. Given an HMM $\lambda = (A, B, \pi)$ and an observation sequence, discover the most optimal hidden sequence Q.
3. Given an observation sequence Q and the set of states Q, determine the HMM parameters $\lambda = (A, B, \pi)$.

Complexity time! Now, when we naïvely approach the first problem as described above, we see that for a given state set Q, observation sequence O and parameters λ, we have (with $|Q| = N, |O| = T$):

$$\begin{aligned} P(O|\lambda) &= \sum_Q P(O|Q, \lambda) \cdot P(Q|\lambda) \\ &= \sum_Q \left(P(Q|\lambda) \cdot \prod_{i=1}^{T} P\left(o_i|q_i, \lambda\right) \right) \\ &= \sum_Q \pi_{q_1} \cdot \prod_{i=1}^{T-1} a_{q_i q_{i+1}} \prod_{i=1}^{T} b_{q_i}(o_i) \end{aligned}$$

Now, computing this as it is will take $\mathcal{O}(T \cdot N^T)$ time which is way too slow.

Two ways to solve a problem... To elucidate with a quote from an editorial on CodeForces:

> Given a problem, there are two ways to solve it: a smart way, and dynamic programming.

The best optimization of the time complexity for this case happens to be through dynamic programming techniques called the forward and backward procedures. The forward procedure essentially does compute the observation probabilities by summing over every hidden state path possible, but does so by implicitly folding each one into a single forward trellis.

We define $\alpha_t(j)$ as the probability of being in state j at time t, *i.e.* after seeing the first t observations. Formally:

$$\alpha_t(j) = P(o_1, \ldots, o_t, q_t = j|\lambda)$$

Now, for a given state j at time t, it is easy to see that $\alpha_t(j)$ is computed as

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t)$$

With the initialization as $\alpha_1(i) = \pi_i b_i(o_1)$ Now it isn't hard to see that the recursion terminates as

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

This procedure, when implemented in the standard dynamic programming style, gives a time complexity of $\mathcal{O}(N^2T)$, which is *much* faster than the exponential time we had using the naïve method.

This can be computed in a reverse way too, called the backward procedure, which shall be dealt with in problem 3. Naturally, the backward procedure works in a similar fashion and gives the exact same time complexity.

DP again? What is this, competitive programming? Tackling the second problem, that of decoding the hidden state sequence, we use another dynamic programming algorithm, called Viterbi's algorithm.

To further hammer home the reasoning for why it is necessary, let's consider the naïve method for this. We can run the forward algorithm and compute the probability of the observation sequence given a hidden state sequence. Then we could choose the hidden state sequence with the maximum observation likelihood. It is easy to see that this would lead to an exponential time complexity because there are exponentially many hidden state sequences.

Now, in Viterbi's algorithm, we define the value of each cell of the trellis (similarly to the forward procedure) as $v_t(j)$ representing the probability that the HMM is in the state j at time t after passing through the most probable state sequence $q_1, \ldots, q_{t-1}$. Formally,

$$v_t(j) = \max_{q_1, \ldots, q_{t-1}} P(q_1, \ldots, q_{t-1}, o_1, \ldots, o_t, q_t = j|\lambda)$$

Now, this can't be DP without a recursive redefinition, which in this case is simply

$$v_t(j) = \max_{i-1}^{N} v_{t-1}(i) a_{ij} b_j(o_t)$$

With the initialization of $v_1(j) = \pi_j b_j(o_1)$. The algorithm differs from the forward algorithm in that we need to maintain at every step a back pointer that keeps track of what the optimal path taken in that transition is. This is because the purpose of this algorithm is not only to obtain the maximum probability, but also to determine the path that got us this maximum probability. This would lead to another recursive definition for the backpointers using the argmax instead of the max function, which can also be easily formalized and implemented.

Models need training too The entire point of the third problem is obtaining a method to train such an HMM when created. That is, we need to adjust or reestimate λ to better fit the data, *i.e.* to improve on $P(O|\lambda)$. This uses an algorithm called the Baum-Welch algorithm, also called the Forward-Backward algorithm.

It's called so because it uses elements and principles from both the forward and backward procedures as defined in paragraph **Two ways to solve a problem...**. Specifically, it uses the definitions and computations of $\alpha_t(j)$ from the forward procedure as well as those of $\beta_t(j)$ from the backward procedure (definition of β is similar to α but backward).

Now, we define the probability ξ that a transition occurred from state i to j between times $t, t+1$. That is,

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O, \lambda) = \frac{P(q_t = i, q_{t+1} = j, O|\lambda)}{P(O|\lambda)}$$

The second equality follows from a trivial application of Bayes' Theorem.

Now, we define an auxiliary function (a surprise tool which will help us later), which Jurafsky and Martin creatively (and appropriately) call 'not-quite-ξ', but which we, for ease of typesetting, will call ζ.

$$\begin{aligned}\zeta_t(i,j) &= P(q_t = i, q_{t+1} = j, O|\lambda) \\ &= \alpha_t(i) \cdot a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_{t+1}(j)\end{aligned}$$

Now, it is easy to see that

$$\xi_t(i,j) = \frac{\zeta_t(i,j)}{\sum_{i=1}^{N} \alpha_t(i) \cdot \beta_t(i)}$$

Using the interpretation of ξ that we used to define it as so, and looking at the semantics as we sum it up over different parameters, we get that:

- Summing it up over t gives us the overall probability of a transition from state i to j. When interpreting this as the expected value of a Bernoulli random variable, we can treat it as the expected number of transitions from i to j.
- Summing it over i gives us the expected number of transitions into j at time $t+1$. Thus, summing that over t gives us the expected number of transitions into j.
- Summing it over j gives us the expected number of transitions from i at time t. Thus, summing that over t gives us the expected number of transitions from i.

Now, using these interpretations, we can obtain the predicted value of the transition probabilities as:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)}$$