the great big
NLP PRIMER

BY MATTHEW MAYO

kdnuggets.com

1. Natural Language Processing vs. Text Mining

There is no shortage of text data available today. Vast amounts of text are created each and every day, with this data ranging from fully structured to semi-structured to fully unstructured.

What can we do with this text? Well, quite a bit, actually; depending on exactly what your objectives are, there are 2 intricately related yet differentiated umbrellas of tasks which can be exploited in order to leverage the availability of all of this data.

Let's start with some definitions.

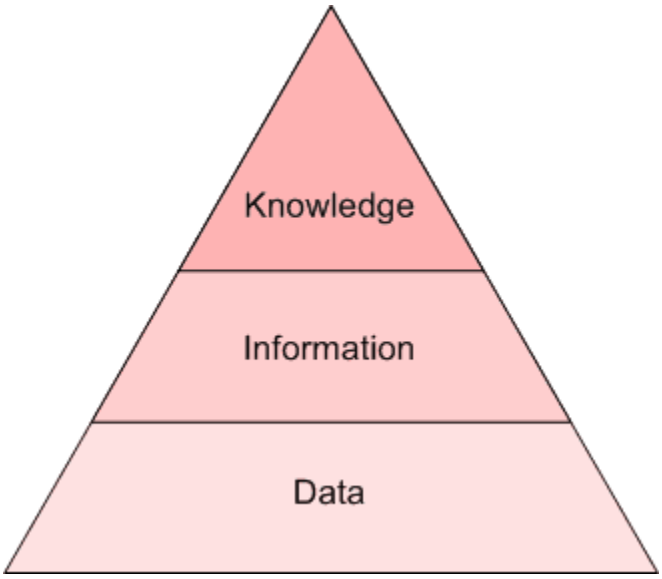
Natural language processing (NLP) concerns itself with the interaction between natural human languages and computing devices. NLP is a major aspect of computational linguistics, and also falls within the realms of computer science and artificial intelligence.

Text mining exists in a similar realm as NLP, in that it is concerned with identifying interesting, non-trivial patterns in textual data.

As you may be able to tell from the above, the exact boundaries of these 2 concepts are not well-defined and agreed-upon, and bleed into one another to varying degrees, depending on the practitioners and researchers with whom one would discuss such matters. I find it easiest to differentiate by degree of insight. If raw text is data, then text mining can extract information, while NLP extracts

knowledge (see the Pyramid of Understanding below). Syntax versus semantics, if you will.

Various other explanations as to the precise relationship between text mining and NLP exist, and you are free to find something that works better for you. We aren't really as concerned with the exact definitions — absolute or relative — as much as we are with the intuitive recognition that the concepts are related with some overlap, yet still distinct.



The Pyramid of Understanding: data, information, knowledge.

The point we will move forward with: although NLP and text mining are not the same thing, they are closely related, deal with the same raw data type, and have some crossover in their uses. For the remainder of our discussion, we will conveniently refer to these 2 concepts as NLP.

Importantly, much of the preprocessing of data for the tasks falling under these 2 umbrellas is identical.

Regardless of the exact NLP task you are looking to perform, maintaining text meaning and avoiding ambiguity is paramount. As such, attempts to avoid

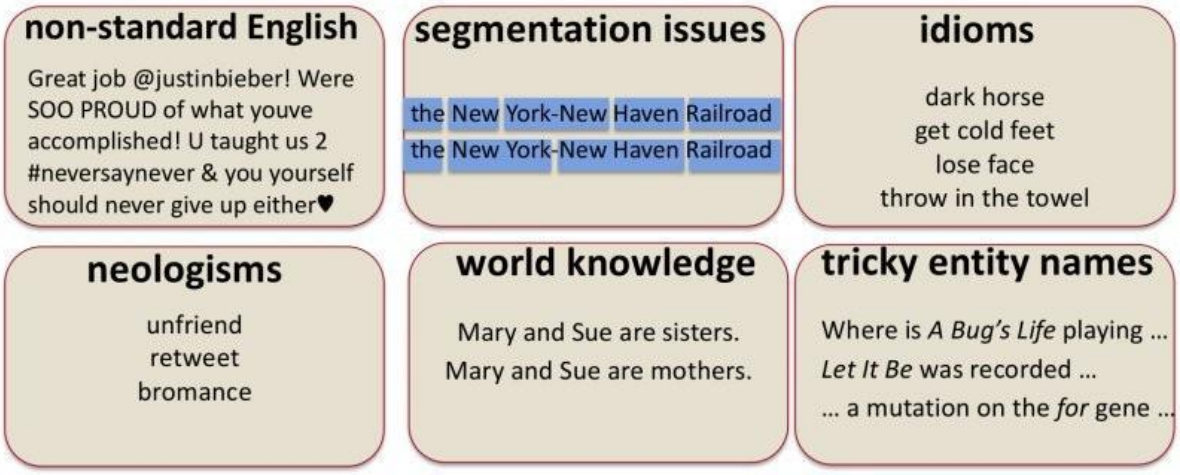
ambiguity is a large part of the text preprocessing process. We want to preserve intended meaning while eliminating noise.

In order to do so, the following is required:

- Knowledge about language
- Knowledge about the world
- A way to combine knowledge sources

If this were easy, we likely wouldn't be talking about it. What are some reasons why processing text is difficult?

2. Why is Processing Text Difficult?



Source: CS124 Stanford (https://web.stanford.edu/class/cs124/)

You may have noticed something curious about the above: "non-standard English." While certainly not the intention of the figure — we are taking it fully out of context here — for a variety of reasons, much of NLP research has historically taken place within the confines of the English language. So we can add to the question of "Why is processing text difficult?" the additional layers of trouble which non-English languages — and especially languages with smaller numbers of speakers, or even endangered languages — must go through in order to be processed and have insights drawn from.

Just being cognizant of the fact that NLP ≠ English when thinking and speaking about NLP is one small way in which this bias can be stemmed.

3. Textual Data Science Task Framework

Can we craft a sufficiently general framework for approaching textual data science tasks? It turns out that processing text is very similar to other non-text processing tasks, and so we can look to the KDD Process for inspiration.

We can say that these are the main steps of a generic text-based task, one which falls under text mining or NLP.

Data Collection or Assembly

- Obtain or build corpus, which could be anything from emails, to the entire set of English Wikipedia articles, to our company's financial reports, to the complete works of Shakespeare, to something different altogether

Data Preprocessing

- Perform the preparation tasks on the raw text corpus in anticipation of text mining or NLP task
- Data preprocessing consists of a number of steps, any number of which may or not apply to a given task, but generally fall under the broad categories of tokenization, normalization, and substitution

Data Exploration & Visualization

- Regardless of what our data is -- text or not -- exploring and visualizing it is an essential step in gaining insight
- Common tasks may include visualizing word counts and distributions, generating word clouds, and performing distance measures

Model Building

- This is where our bread and butter text mining or NLP task takes place (includes training and testing)
- Also includes feature selection & engineering when applicable
- Language models: Finite state machines, Markov models, vector space modeling of word meanings
- Machine learning classifiers: Naive bayes, logistic regression, decision trees, Support Vector Machines, neural networks
- Sequence models: Hidden Markov models, recursive neural networks (RNNs), Long short term memory neural networks (LSTMs)

Model Evaluation

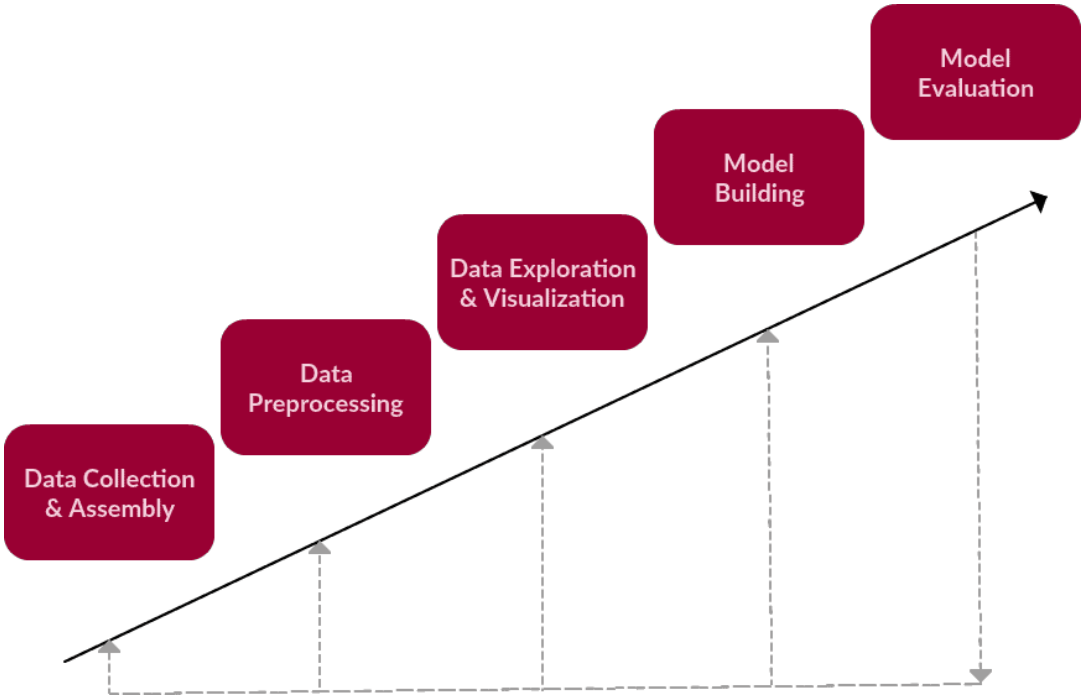
- Did the model perform as expected?
- Metrics will vary dependent on what type of text mining or NLP task
- Even thinking outside of the box with chatbots (an NLP task) or generative models: some form of evaluation is necessary

Clearly, any framework focused on the preprocessing of textual data would have to be synonymous with step number 2. Expanding upon this step, specifically, we had the following to say about what this step would likely entail:

- Perform the preparation tasks on the raw text corpus in anticipation of text mining or NLP task
- Data preprocessing consists of a number of steps, any number of which may or not apply to a given task, but generally fall under the broad categories of tokenization, normalization, and substitution

More generally, we are interested in taking some predetermined body of text and performing upon it some basic analysis and transformations, in order to be left with artefacts which will be much more useful for performing some further, more

meaningful analytic task afterward. This further task would be our core text mining or natural language processing work.



A simple textual data task framework.

So, as mentioned above, it seems as though there are 3 main components of text preprocessing:

- tokenization
- normalization
- substitution

As we lay out a framework for approaching preprocessing, we should keep these high-level concepts in mind.

4. Text Preprocessing Framework

We will introduce this framework conceptually, independent of tools. We will then followup with a practical implementation of these steps next time, in order to see how they would be carried out in the Python ecosystem.

The sequence of these tasks is not necessarily as follows, and there can be some iteration upon them as well.

Noise Removal

Noise removal performs some of the substitution tasks of the framework. Noise removal is a much more task-specific section of the framework than are the following steps.

Keep in mind again that we are not dealing with a linear process, the steps of which must exclusively be applied in a specified order. Noise removal, therefore, can occur before or after the previously-outlined sections, or at some point between).

Corpus (literally Latin for body) refers to a collection of texts. Such collections may be formed of a single language of texts, or can span multiple languages; there are numerous reasons for which multilingual corpora (the plural of corpus) may be useful. Corpora may also consist of themed texts (historical, Biblical, etc.). Corpora are generally solely used for statistical linguistic analysis and hypothesis testing.

How about something more concrete? Let's assume we obtained a corpus from the world wide web, and that it is housed in a raw web format. We can, then, assume that there is a high chance our text could be wrapped in HTML or XML

tags. While this accounting for metadata can take place as part of the text collection or assembly process (step 1 of our textual data task framework), it depends on how the data was acquired and assembled. Sometimes we have control of this data collection and assembly process, and so our corpus may already have been de-noised during collection.

But this is not always the case. If the corpus you happen to be using is noisy, you have to deal with it. Recall that analytics tasks are often talked about as being 80% data preparation!

The good thing is that pattern matching can be your friend here, as can existing software tools built to deal with just such pattern matching tasks.

- remove text file headers, footers
- remove HTML, XML, etc. markup and metadata
- extract valuable data from other formats, such as JSON, or from within databases
- if you fear regular expressions, this could potentially be the part of text preprocessing in which your worst fears are realized

Regular expressions, often abbreviated regexp or regexp, are a tried and true method of concisely describing patterns of text. A regular expression is represented as a special text string itself, and is meant for developing search patterns on selections of text. Regular expressions can be thought of as an expanded set of rules beyond the wildcard characters of ? and *. Though often cited as frustrating to learn, regular expressions are incredibly powerful text searching tools.

As you can imagine, the boundary between noise removal and data collection and assembly is a fuzzy one, and as such some noise removal must absolutely take place before other preprocessing steps. For example, any text required from a JSON structure would obviously need to be removed.

Normalization

Before further processing, text needs to be normalized.

Normalization generally refers to a series of related tasks meant to put all text on a level playing field: converting all text to the same case (upper or lower), removing punctuation, expanding contractions, converting numbers to their word equivalents, and so on. Normalization puts all words on equal footing, and allows processing to proceed uniformly.

Normalizing text can mean performing a number of tasks, but for our framework we will approach normalization in 3 distinct steps: (1) stemming, (2) lemmatization, and (3) everything else.

Stemming is the process of eliminating affixes (suffixes, prefixes, infixes, circumfixes) from a word in order to obtain a word stem.

RUNNING ⟶ RUN

Lemmatization is related to stemming, differing in that lemmatization is able to capture canonical forms based on a word's lemma. For example, stemming the word "better" would fail to return its citation form (another word for lemma); however, lemmatization would result in the following:

BETTER ⟶ GOOD

It should be easy to see why the implementation of a stemmer would be the less difficult feat of the two.

Everything Else

A clever catch-all, right? Stemming and lemmatization are major parts of a text preprocessing endeavor, and as such they need to be treated with the respect they deserve. These aren't simple text manipulation; they rely on detailed and nuanced understanding of grammatical rules and norms.

There are, however, numerous other steps that can be taken to help put all text on equal footing, many of which involve the comparatively simple ideas of substitution or removal. They are, however, no less important to the overall process. These include:

- set all characters to lowercase
- remove numbers (or convert numbers to textual representations)
- remove punctuation (generally part of tokenization, but still worth keeping in mind at this stage, even as confirmation)
- strip white space (also generally part of tokenization)
- remove default stop words (general English stop words)
- remove given (task-specific) stop words
- remove sparse terms (not always necessary or helpful, though!)

Stop words are those words which are filtered out before further processing of text, since these words contribute little to overall meaning, given that they are generally the most common words in a language. For instance, "the," "and," and "a," while all required words in a particular passage, don't generally contribute greatly to one's understanding of content. As a simple example, the following pangram is just as legible if the stop words are removed:

~~The~~ quick brown fox jumps over ~~the~~ lazy dog.

At this point, it should be clear that text preprocessing relies heavily on pre-built dictionaries, databases, and rules. You will be relieved to find that when we

undertake a practical text preprocessing task in the Python ecosystem in our next article that these pre-built support tools are readily available for our use; there is no need to be inventing our own wheels.

Tokenization

Tokenization is, generally, an early step in the NLP process, a step which splits longer strings of text into smaller pieces, or tokens. Larger chunks of text can be tokenized into sentences, sentences can be tokenized into words, etc. Further processing is generally performed after a piece of text has been appropriately tokenized.

Tokenization is also referred to as text segmentation or lexical analysis. Sometimes segmentation is used to refer to the breakdown of a large chunk of text into pieces larger than words (e.g. paragraphs or sentences), while tokenization is reserved for the breakdown process which results exclusively in words.

This may sound like a straightforward process, but it is anything but. How are sentences identified within larger bodies of text? Off the top of your head you probably say "sentence-ending punctuation," and may even, just for a second, think that such a statement is unambiguous.

Sure, this sentence is easily identified with some basic segmentation rules:

> The quick brown fox jumps over the lazy dog.

But what about this one:

> Dr. Ford did not ask Col. Mustard the name of Mr. Smith's dog.

Or this one:

> "What is all the fuss about?" asked Mr. Peters.

And that's just sentences. What about words? Easy, right? Right?

> This full-time student isn't living in on-campus housing, and she's not wanting to visit Hawai'i.

It should be intuitive that there are varying strategies not only for identifying segment boundaries, but also what to do when boundaries are reached. For example, we might employ a segmentation strategy which (correctly) identifies a particular boundary between word tokens as the apostrophe in the word she's (a strategy tokenizing on whitespace alone would not be sufficient to recognize this). But we could then choose between competing strategies such as keeping the punctuation with one part of the word, or discarding it altogether. One of these approaches just seems correct, and does not seem to pose a real problem. But just think of all the other special cases in just the English language we would have to take into account.

Consideration: when we segment text chunks into sentences, should we preserve sentence-ending delimiters? Are we interested in remembering where sentences ended?

5. What NLP Tasks Exist?

We have covered some text (pre)processing steps useful for NLP tasks, but what about the tasks themselves?

There are no hard lines between these task types; however, many are fairly well-defined at this point. A given macro NLP task may include a variety of sub-tasks.

We first outlined the main approaches, since the technologies are often focused on for beginners, but it's good to have a concrete idea of what types of NLP tasks there are. Below are the main categories of NLP tasks.

Text Classification Tasks

- **Representation**: bag of words, n-grams, one-hot encoding (sparse matrix); these methods do not preserve word order
- **Goal**: predict tags, categories, sentiment
- **Application**: filtering spam emails, classifying documents based on dominant content

Bag of words is a particular representation model used to simplify the contents of a selection of text. The bag of words model omits grammar and word order, but is interested in the number of occurrences of words within the text. The ultimate representation of the text selection is that of a bag of words (bag referring to the set theory concept of multisets, which differ from simple sets).

Actual storage mechanisms for the bag of words representation can vary, but the following is a simple example using a dictionary for intuitiveness. Sample text:

> "Well, well, well," said John.
>
> "There, there," said James. "There, there."

The resulting bag of words representation as a dictionary:

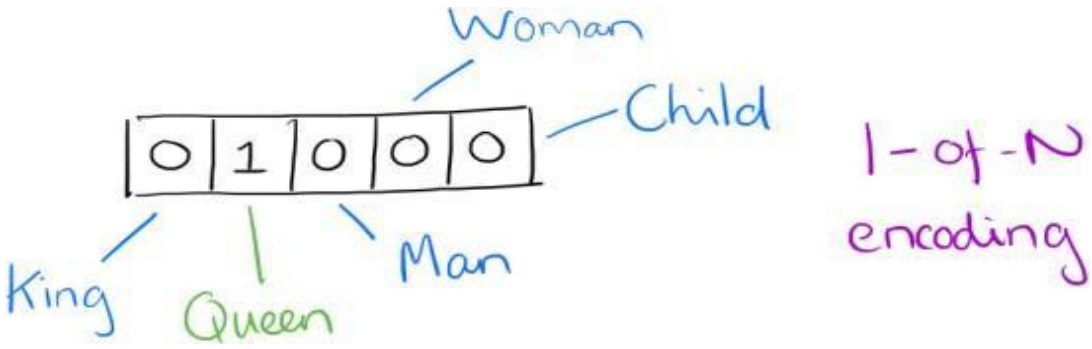
```
{
'well':  3,
'said':  2,
'john':  1,
'there':  4,
'james':  1
}
```

n-grams is another representation model for simplifying text selection contents. As opposed to the orderless representation of bag of words, n-grams modeling is interested in preserving contiguous sequences of N items from the text selection.

An example of a trigram (3-gram) model of the second sentence of the above example ("There, there," said James. "There, there.") appears as a list representation below:

```
[
"there there said", "there said james", "said james there",
"james there there",
]
```

Prior to the wide-spread use of neural networks in NLP — in what we will refer to as "traditional" NLP — vectorization of text often occurred via **one-hot encoding** (note that this persists as a useful encoding practice for a number of exercises, and has not fallen out of fashion due to the use of neural networks). For one-hot encoding, each word, or token, in a text corresponds to a vector element.



Source: Adrian Colyer

We could consider the image above, for example, as a small excerpt of a vector representing the sentence "The queen entered the room." Note that only the element for "queen" has been activated, while those for "king," "man," etc. have not. You can imagine how differently the one-hot vector representation of the sentence "The king was once a man, but is now a child" would appear in the same vector element section pictured above.

Word Sequence Tasks

- **Representation**: sequences (preserves word order)
- **Goal**: language modeling - predict next/previous word(s), text generation
- **Application**: translation, chatbots, sequence tagging (predict POS tags for each word in sequence), named entity recognition

Language modeling is the process of building a statistical language model which is meant to provide an estimate of a natural language. For a sequence of input words, the model would assign a probability to the entire sequence, which contributes to the estimated likelihood of various possible sequences. This can be especially useful for NLP applications which generate or predict text.

Text Meaning Tasks

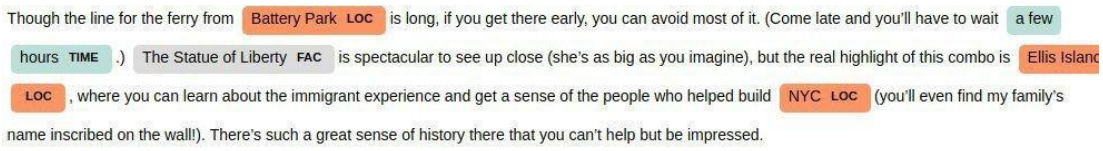
- **Representation**: word vectors, the mapping of words to vectors (n-dimensional numeric vectors) aka embeddings
- **Goal**: how do we represent meaning?
- **Application**: finding similar words (similar vectors), sentence embeddings (as opposed to word embeddings), topic modeling, search, question answering

Dense embedding vectors aka word embeddings result in the representation of core features embedded into an embedding space of size d dimensions. We can compress, if you will, the number of dimensions used to represent 20,000 unique words down to, perhaps, 50 or 100 dimensions. In this approach, each feature no longer has its own dimension, and is instead mapped to a vector.

Source: Adrian Colyer

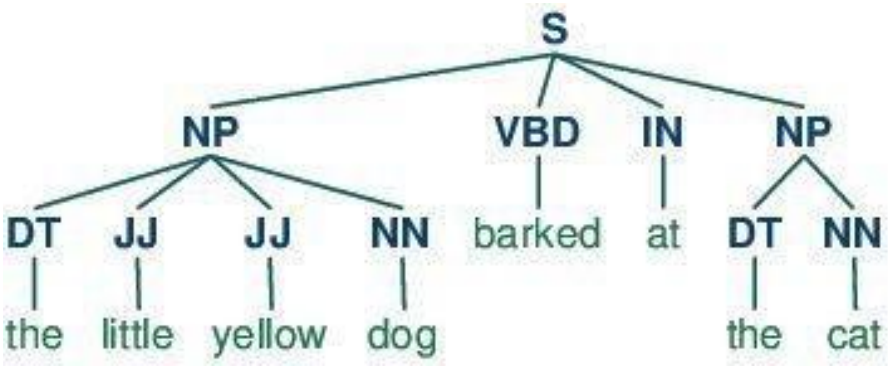
So, what exactly are these features? We leave it to a neural network to determine the important aspects of relationships between words. Though human interpretation of these features would not be precisely possible, the image above provides an insight into what the underlying process may look like, relating to the famous King - Man + Woman = Queen example.

Named entity recognition is the attempt to identify named entities in text data, and categorize them appropriately. Named entities categories include, but are not limited to, person names, locations, organizations, monetary values, dates and time, quantities, etc., and can include custom categories dependent on the application (medical, scientific, business, etc.). You can think of named entities as **proper nouns++**.



Named entity recognition (NER) using spaCy (text excerpt taken from here)

Part-of-speech tagging consists of assigning a category tag to the tokenized parts of a sentence. The most popular POS tagging would be identifying words as nouns, verbs, adjectives, etc.



Sequence to Sequence Tasks

- Many tasks in NLP can be framed as such
- Examples are machine translation, summarization, simplification, Q&A systems
- Such systems are characterized by encoders and decoders, which work in complement to find a hidden representation of text, and to use that hidden representation

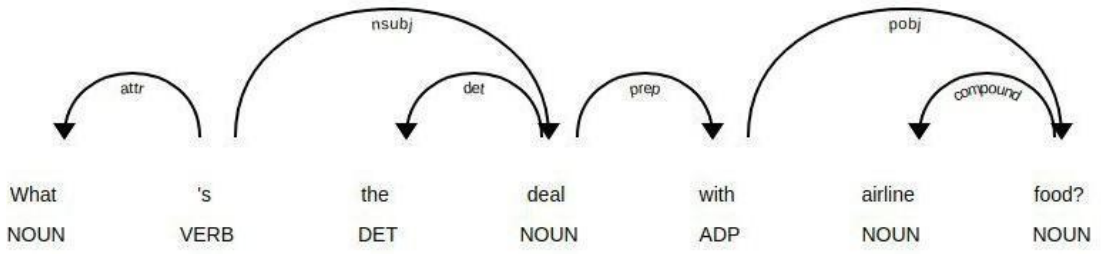
Dialog Systems

- 2 main categories of dialog systems, categorized by their scope of use
- Goal-oriented dialog systems focus on being useful in a particular, restricted domain; more precision, less generalizable
- Conversational dialog systems are concerned with being helpful or entertaining in a much more general context; less precision, more generalization

Whether it be in dialog systems or the practical difference between rule-based and more complex approaches to solving NLP tasks, note the trade-off between precision and generalizability; you generally sacrifice in one area for an increase in the other.

6. Approaches to NLP Tasks

While not cut and dry, there are 3 main groups of approaches to solving NLP tasks.



Dependency parse tree using spaCy

Rule-based

Rule-based approaches are the oldest approaches to NLP. Why are they still used, you might ask? It's because they are tried and true, and have been proven to work well. Rules applied to text can offer a lot of insight: think of what you can learn about arbitrary text by finding what words are nouns, or what verbs end in -ing, or whether a pattern recognizable as Python code can be identified. Regular expressions and context free grammars are textbook examples of rule-based approaches to NLP.

Rule-based approaches:

- tend to focus on pattern-matching or parsing
- can often be thought of as "fill in the blanks" methods

- are low precision, high recall, meaning they can have high performance in specific use cases, but often suffer performance degradation when generalized

"Traditional" Machine Learning

"Traditional" machine learning approaches include probabilistic modeling, likelihood maximization, and linear classifiers. Notably, these are not neural network models (see those below).

Traditional machine learning approaches are characterized by:

- training data - in this case, a corpus with markup
- feature engineering - word type, surrounding words, capitalized, plural, etc.
- training a model on parameters, followed by fitting on test data (typical of machine learning systems in general)
- inference (applying model to test data) characterized by finding most probable words, next word, best category, etc.
- "semantic slot filling"

Neural Networks

This is similar to "traditional" machine learning, but with a few differences:

- feature engineering is generally skipped, as networks will "learn" important features (this is generally one of the claimed big benefits of using neural networks for NLP)
- instead, streams of raw parameters ("words" -- actually vector representations of words) without engineered features, are fed into neural networks
- very large training corpus

Specific neural networks of use in NLP have "historically" included recurrent neural networks (RNNs) and convolutional neural networks (CNNs). Today, the one architecture that rules them all is the transformer.

Why use "traditional" machine learning (or rule-based) approaches?

- still good for sequence labeling (using probabilistic modeling)
- some ideas in neural networks are very similar to earlier methods (word2vec similar in concept to distributional semantic methods)
- use methods from traditional approaches to improve neural network approaches (for example, word alignments and attention mechanisms are similar)

Why deep learning over "traditional" machine learning?

- SOTA in many applications (for example, machine translation)
- a lot of research (majority?) in NLP happening here now

Importantly, both neural network and non-neural network approaches can be useful for contemporary NLP in their own right; they can also can be used or studied in tandem for maximum potential benefit

References

1. From Languages to Information, Stanford
2. Natural Language Processing, National Research University Higher School of Economics (Coursera)
3. Neural Network Methods for Natural Language Processing, Yoav Goldberg
4. Natural Language Processing, Yandex Data School
5. The amazing power of word vectors, Adrian Colyer

About the Author

Matthew Mayo (@mattmayo13) holds a Master's degree in computer science and a graduate diploma in data mining. As Editor-in-Chief of KDnuggets, Matthew aims to make complex data science concepts accessible. His professional interests include natural language processing, machine learning algorithms, and exploring emerging AI. He is driven by a mission to democratize knowledge in the data science community. Matthew has been coding since he was 6 years old.

Cover image courtesy of gpointstudio on Freepik.