# Chapter 21 - Natural Language Processing

## Topics and Chatbots

### 21.1 - Introduction

As with many topics with machine learning, it was the internet that both created the need and provided the necessary data to do natural language processing at scale. Ultimately the goal of natural language processing is for algorithms to understand our written content in the same way that other humans do. This includes parsing the text, understanding the relationships between the words, the resolution of entities and how the relate to one and other. But this is a big and challenging endeavor, so the focus currently still is on the various subtasks that make up this challenge, with some additional useful solutions also being explored.

### 21.2 - Use Cases - Text classification

Text classification was unfortunately one of the earliest needs the internet presented. With email came spam, and with that, we needed spam filters: an automated way to detect and flag spam messages and move them aside away from real content. We will implement a classic example of a spam filter using naive Bayes classifiers. However, instead of spam versus ham, we will try to detect cancer specific content within a larger corpus of medical information

Figure 1 - We call unwanted email spam because of the Monty Python sketch

### 21.3 - Use Cases - Translation

Translation is a difficult task to automate. Dictionary-based word-by-word translation lack proper context and will yield poor results. Google has solved this problem using its access to the vast amount of content on the web it has access too. Rather than translating single words, Google uses a rule-based translation method that utilizes predictive algorithms to guess ways to translate texts in foreign languages. It aims to translate whole phrases rather than single words, and then gather overlapping phrases for translation.

Figure 2 - Google has been on the forefront of automated translation, primarily because of its access to massive amounts of text data

### 21.4 - Use Cases - Topic modeling

Topic modeling is another NLP mainstay and is still being actively researched. How can we and should we group different texts, like webpages, messages, posts, blogs together automatically in a meaningful way? We will run through an example of topic modeling that uses PCA (described in the previous chapter) and various metrics of vectorizing documents based on the words that occur in them.

### 21.5 - Use Cases - Text Summarization

Text summarization is the logical next step after text classification and topic modeling. If we can quickly and accurately summarize texts in a readable and understandable format, humans would be far more efficient in finding and understand the content they are looking for.

### 21.6 – Use Cases - Entity resolution

Entity resolution is an important step towards understanding content. Entities represent such things individuals, places, events, objects. Since such entities can appear in our content in many different ways, which need to be resolved or mapped to single entity.

Figure 3 - Entity resolution aims to map all references to the same object to a canonical form. For example: '44th president, Barack, Obama' all map to the entity of Barack Obama.

### 21.7 – Use Cases - Knowledge graphs

With the entities resolves, we can start building graphs that link them. Using the entities as nodes we find the edge that link them. In proper knowledge graphs, these edges denote relationships. It is however possible to build knowledge graphs that simply look at some measure of proximity of entities (or words). The most intuitive of the proximity measures is cooccurrence. If we consistently see the same two words appear together in a single text, it is safe to say they are most likely related closely, compared to a third word that never appears in those same texts, but does appear in other texts. We will work out an example using the transcripts of our MSK information portal.

Figure 4 - Knowledge graphs link concepts (nodes) by their relationship (edges).

### 21.8 – Use Cases - Text Synthesis

Not always with the best intention in mind, we are already seeing a lot of websites that autogenerate content. When it is used to unburden humans from having to write large amounts of repetitive text, this seems a welcome development. If, however, it is used for generating clickbait, not so much.

### 21.9 – Use Cases - Spelling and grammar

Spelling and grammar checkers now also increasingly rely on probabilistic models computed from internet content, rather than dusty Websters dictionaries. We will run through a simple example of how Hidden Markov Models can be used to find possible errors as well automatically generating content.

### 21.10 – Use Cases - Word2Vector

Word2Vector is a very useful methodology that allows us to map individual words onto a unique vector, where similar words, in terms of their meaning, map onto similar vectors. We will build a vectorization model for a large medical corpus.

### 21.11 - Preprocessing

NLP relies on a set of universally adopted preprocessing steps that will result in better, cleaner, and more interpretable models down the line. This includes standardization, stemming, lemmatization and stop word removal. As an initial standardization step, we make sure that all non-alphanumeric characters are removed. This sometimes excludes the period, comma, colon, and semi-colon as these can signal substructures with the text, like sentences or clauses. A spell checker can be used remove some of the noise introduced by typos or incorrectly spelled words. Converting all words to a the same (e.g., lower) case, and replacing words representing numbers by actual numbers are additional and optional steps that can improve the quality of the NLP models.

### 21.12 - Stemming and Lemmatization

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For example, ‘organization’ and ‘organizations’ can be reduced to ‘organization’ and ‘am’, ‘are’, ‘is’ can be reduced to the verb’s base form ‘be’. The difference between the two is that stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. In contrast, lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.

### 21.13 - Stopword Removal

Many of the words in any text carry little to no information about the content in it. Therefore, in many NLP applications we use a list of stop words to remove them throughout the text. Typically, stop words represent the top-n most frequently used words in a language, such as ‘a’, ‘the’, ‘is’, etc.

Table 1 – 1000 Most frequent words in english. It is easy to see that they carry no meaning about the topic of a text.

### 21.14 - Bag of Words versus Generative and Probabilistic Models

A variety of traditional machine learning NLP algorithms fall under the header of so called 'bag of words' models. In this approach the order of words in phrases, sentences and even whole documents is essentially ignored. By doing so, the problem is greatly simplified, and we can instead just focus on the number of times a word occurs in a document, rather than were. We do still consider the co-occurrence of words as an important source of information, but exactly where they happen to be relative to each other is not deemed to be vital. The bag of words approach is typically seen in conventional text classification and document retrieval algorithms. In contrast, probabilistic and generative models do keep track of word order, to capture the probability of a sequence of words occurring. For example, when observing the phrase 'Mary had a little' we will likely observe the word 'lamb' next. Estimating these probabilities accurately depends in large on the amount of available data. Even in a simple model where we would only consider a dictionary of 1000 words, we need massive amount of text to compute the probability of seeing word 1 followed by word 2, since there are 1000 x 1000 different probabilities to be estimated (word 1 -> word 2, word 1 -> word 3 ..., word 1000 -> word 1000. A single pairing of 2 words is called a bigram, and sequence of words of n words long is called an n-gram. As you can imagine, the longer the n-gram, the exponentially larger data set we need. Estimating realistic probabilities on these n-grams with some reasonable accuracy only became possible recently and was driven largely by tech companies with access to all this data, like Google. Probabilistic models also allow us to artificially generate sentences, text, and documents. Given a seed, like the start of a sentence, an algorithm can use the estimated probabilities to add words or phrases by picking the most likely one. Again, this requires massive amounts of data because the algorithm needs to consider more than just the past few words as not to degenerate quickly into spouting gibberish. However, with a new generation of neural networks like OpenAI's GPT-3 and increasingly large data sets, AI is now capable of generating reasonably human like language and carry on a conversation with a human where it might pass the Turing test on occasion in fooling the human it is indeed human as well.

### 21.15 - Bag of Words - Topic Modeling

Topic modeling aims to finds naturally occurring groupings of documents using the words in them. Historically, topic modeling algorithms use a bag of words approach. It faces two challenges that can be approached in different way, with mixed results depending on the type of content being considered. The first challenge is to rank individual words by their relative importance. Very common words (like stop words) don't carry any meaningful signal on what the document might be about. At the same time, very rare words are not necessarily ideal to base topics on. Assigning too much value on rare words increases the risk of overfitting. What is deemed 'rare' is a function of the data itself and requires some experimentation to get right. The second challenge is how we group documents together based on how similar they are in terms of words the contain, and their ranking. There are several algorithms to do this with, including Principal Component Analysis (which we discussed in the chapter on Dimensionality Reduction), Latent Dirichlet Analysis (or LDA) and non-negative matrix factorization. In this section we will briefly outline one (PCA using TFiDF), leaving its full implementation to the accompanying demo notebooks.

### 21.16 - Bag of Words - Topic Modeling - TFiDF

The first step in building our topic modeling algorithm is to decide how to represent document our documents and the words in them. To allow for any numerical operation we have to move away from the document and words themselves. Traditionally, this is done by representing each document in the corpus as a vector, where each index of the vector represents one unique word. The length of these vectors is equal to the number of different words in our dictionary (n), and we combine them into a matrix representing all the vectors, one for each document (m). We thus want to construct this matrix D of size m (documents by size n (dictionary length), where each entry Dij represents a specific word - document combination. After this step we will need to consider our dictionary. Which words do we include, and which ones do we leave out? As discussed above, the first step of most NLP algorithms is to remove stop words. Different lists of different lengths exist, but they will ultimately all aim to have the most frequent English words (should our corpus be in English of course). Alternatively, this decision could be based on the dictionary created from the corpus itself, should it somehow deviate from typical written English. In addition, we need to decide where to stop including words in our dictionary, should they become too rare. Finding the appropriate cutoff on both sides (lower and upper frequency of words to exclude) does depend on your corpus, so some experimentation is to find a right balance. With the dictionary created, we now turn to how to represent each word-document entry in our matrix. So, how should we value the importance of certain words within documents and across documents? The most straightforward way is to create a binary matrix where Dij represents the presence or absence of a word j in document i. This is not a bad start, but it glosses over the fact that word used multiple times within a single document is likely to carry some weight on what the document is about after we removed stop words we expect to see, regardless of topic. To improve our measure, we can therefore count the number of times a specific word occurs and use that for out matrix D. However, this is also prone to bias, as documents do differ in length, with longer documents producing artificially higher counts. To overcome this, we compute the Term Frequency (TF), which is computed as the frequency of a word in document. For example, in the 'the man bit the dog', the raw count of 'the' is 2, and its term frequency is 2/5 = 0.4, whereas the term frequency of the other three words is 1/5 = 0.2. Some algorithms operate on TF specified matrices directly, whereas other add one final correction. Even if we were careful in removing stop words, some words might still be more common across documents than others. We want to value words higher if they appear often within a document the TF but simultaneously lower this value should they appear in a large subset of all documents. Think of this data science course. Each chapter has a clear topic, by design. But in almost all chapters there are one or multiple mentions of 'data', 'algorithm’, and 'learning'. These are obviously not stop words, but they also don't really narrow down how one chapter differs from another. This second measure is called the inverse document frequency. It computes the ratio (total number of documents / total of documents containing the word) and takes the natural logarithm of this ratio. This operation produces a measure that increases as a word appear in fewer and fewer documents. To combine these metrics, they are usually simply multiplied together for each word/document combination to yield the final metric abbreviated as TFiDF (Term Frequency - inverse Document Frequency).

### 21.17 - Bag of Words - Topic Modeling - LDA

### 21.18 – Probability algorithms – Intro to Bayes’ law

Before we start building our cancer content filters with naive bayes classifiers, lets quickly review some basic probability theory, and specifically Bayes theorem, the relationship between probabilities that our spam filter will use to decide whether something is about cancer, or another medical condition. The probability of an event a is donated by p(a), ‘p of a’. It is a bounded value between 0 and 1, where 0 means the event, a will not happen, 1 means it will happen, or 0.5 where there is 50/50 chance of event a happening. This is the classical interpretation of probability , which was the interpretation used in Bayes’ era. The other interpretation sees probabilities not as describing individual events, but rather as the average over many of those events.

Equation 1 – Bayes Theorem

The difference is subtle, and numbers and computations don’t change, just how we interpret the results. And does not change our discussion, but nonetheless worth pointing out. Think of a specific cancer. When we say that an individual’s risk of developing that cancer is 0.01, the classical interpretation states he/she will have a 1% chance of that individual developing the cancer. A frequentist will offer a different interpretation. They will state that if there were a hundred people just like me, out of those 100, 1 will develop the cancer. The second important quantity is the conditional probability of event a, given another event b, denoted by p(a|b), or in words: ‘p of a given b’. It is the probability that event a will happen after we already know event b has happened. Think about rolling a die. What are your odds of rolling 6? Well, if the die is fair, exactly 1/6. But what if I roll the die without you looking and tell you at least that the die is even. What are the odds if it being six this time? Since now your only options are 2, 4, 6, our odds of it being 6 increase to 1/3. In general, when p(a|b) is not the same as p(a) we say that the events are dependent. If they are the same, the outcome of b had no effect on the probability of a, and therefore independent. Now, substitute that event for the occurrence of a particular word, and event b for the occurrence of a cancer-related text. Before we outline Bayes law, it is important to note that p(word|cancer) does not equal p(cancer|word). The values can obviously be same for whatever reason, but they do not denote the same probability. They are not identical. Imagine we notice a repeated occurrence of the word the ‘leukemia’. It is almost a given that any text that contains the word ‘leukemia’ is about cancer. In other words, p(cancer|metastatic) is close to one. See the word ‘leukemia’? Probably an article on cancer. However, there are many other text that are about cancer, but not specifically about leukemia. In other words, p(leukemia|cancer), or the chance of seeing ‘leukemia’ given that the text is about cancer is actually low. However, these two probabilities are linked! And they are so by way of Bayes’ theorem or law.

p(cancer|word) = p(word|cancer) \* p(cancer) / p(word)

Equation 2 – Bayes’ theorem in action for our corpus

In words: the probability of a text being about cancer given that contains a particular word (left hand side of the equation) is equal to the probability of seeing that particular word when the text is about cancer multiplied by the probability of any text in our corpus being about cancer and divided by the probability of seeing the word in the text, regardless of whether it is about cancer or not (right hand side). In this formula, we refer to p(cancer|word) the posterior and p(cancer) as the prior. The posterior is the change in probability after we apply additional evidence in the form of p(word|cancer) and normalize that using p(word).

### 21.19 – Probability algorithms – Naive Bayes Classifiers

Hopefully by now, you will an intuition for Bayes law, and on how we might use it to build a cancer/no cancer classifier. But let us outline the steps we need to take.

1.Find all the unique words, and their count across all emails. By dividing each unique words count by the total count we obtain p(word) for every unique word in our emails.

2.Count the number of emails that are about cancer, count the number that are not, and divide those count by the total count. That gives us p(cancer) and also p(other medical condition), which is 1 - p(v) because there are no other classes of text.

3.Now, for each unique word, we count how often it appears in a cancer text. Dividing this by the total number of spam emails gives us p(word|spam) or the probability of seeing a word in a cancer text when we know it is about cancer. We do the same for the non-cancer texts, giving us p(word|other medical condition) for the same word. Note that these values do not have to add up to 1. If we only see a word once, and it is in one of the thousands of cancer texts, but none of the 1000 non-cancer text, we have p(word|cancer) = 0.001, and p(word|other medical condition) = 0

4.We now have all the necessary parameters to do prediction, which is done by combining all the words in a single email to accumulate evidence for the email being spam and ham. Some words add not much evidence either way, like ‘the, a, he, there’. Others are very informative, like the before-mentioned ‘leukemia’ for cancer texts, or words like ‘heart attack’ and ‘dementia’ for texts about other medical conditions.

We accumulate evidence for cancer in an unclassified text by multiplying the p(word|cancer) for each word together. We do these same for the p(word|other medical condition) probabilities. Finally, we normalize these two values to add to 1 by dividing both by their sum. The result value will be close to 0.5 if the evidence for each type of topic is similar but leans towards 0 or 1 if one or more words tip the scale sufficiently in one way or another.

### 21.20 – Word2Vec

Words, sentences, and complete texts are somewhat of a special class of stimuli. Most of machine learning will use numerical inputs. When our inputs are not numerical values representing physical quantities directly, they are still seen as numerical. Images are bit arrays of values representing the red, green, and blue intensities of individual pixels, audio can be representing as a vector describing the waveform of the sound. We could transform our textual data into a binary string representing each character. However, this removes the most important aspect of words: their meaning. Leukemia and cancer are closely related words, relative to word ‘airplane’. But the bit string describing the characters in each of these words clearly don’t reflect any of that similarity. Luckily, there is a nifty trick we can use to turn individual words in to a numerical representation by mapping them onto a vector, where similar words (in terms of their meaning) are mapped onto similar vectors (i.e., pointing in similar directions), a method commonly referred to as Word2Vec. There are several ways to approach it, and we will discuss and work through three of them.

### 21.20 – Word2Vec - PCA

Using our MSK cancer information content, we can build our own version. In the first approach, we will use extremely versatile method of PCA we explored in a previous chapter. We will use the corpus we used previously, in which each document represented a webpage. Again, all documents were preprocessed and had stop words removed prior to analysis. To create a vectorization model, we first build a matrix in which each row represents a document, and each column a unique word. After removing the stop words, it is likely the number of unique words is still quite high. This is typical occurrence in texts that are very topical and scientific. It is full of many words we don’t typically use in daily speech. Therefore, let us simplify our efforts slightly by creating a mapping for only the top-1000 most frequent words in our corpus. We build the matrix by simply putting ones in (row, column) if a particular word (column) appears in a particular text (row). The resulting matrix will be binary and quite sparse: most documents only contain a handful of the top 1000 words and each of those words only appears in a handful of documents. There are of course exceptions, like the word ‘cancer’ itself. We then compute the principal components for this matrix and examine the amount of explained variance. In the coding example I use a cut off of 0.8, meaning I am retaining the first n principal components that together explain 80% of the variation in the data.

Each principal component represents an orthogonal linear weighted mixture of the original matrix’s columns, i.e., the words. If words frequently co-occur in the same text, they will show substantial overlap in the binary column vectors that represent their presence or absence in each of the documents in our corpus (i.e., the rows). These patterns are captures by the principal components. By projecting the original binary matrix onto the principal components, we transform the original space into the PCA space. In this new space, the axis are no longer documents, but principal components. And where in the original space a word was a binary column vector representing in which document it occurred, in the new space, it is a continuous column vector describing its weight to each principal component. It is this vector that now captures the meaning of the word numerically, relative to all other words in our analysis.

### 21.20 – Word2Vec – PCA – Limitations and a way forward

However, the approach described above has the significant drawback of the linear assumption embedded in PCA. Any non-linear relationship between words is missed. It also doesn’t allow us to do anything with the ordering of the words within the documents. The proximity of one word (e.g., lymph) to another word (e.g., node) is obviously something to take note of. This led to new methods being explored, using deep learning. Two of them were especially influential: the Continuous Bag of Words model (CBOW) and the Continuous Skip-Gram Model.

### 21.21 – Advanced Word2Vec Methods – CBOW

The CBOW approach is perhaps better known as it is the easiest to grasp intuitively. In the previous method using PCA, we essentially just counted how many times a word appeared across and within documents, or merely took note of it being present or not. In CBOW, we are going to create a slightly more complex numerical representation of our text. Instead of considering

### 21.21 – Advanced Word2Vec Methods - Skip-Gram

### 21.22 – The Geometry of Word2Vec – Google WordNet

Perhaps the most relevant aspect of the Word2Vec approach in relation to this book, is that it turns the relationship between words into a geometry. After having

### 21.23 – Text Synthesis – OpenAI’s GPT-3

### 21.24 – Demos

### 21.25 - References