IMPLEMENT
CONSULTING GROUP_

GETTING STARTED WITH NATURAL LANGUAGE PROCESSING

– handling text as data

Getting started with natural language processing – handling text as data

Data from audio and text contains a huge potential for generating insights arising through traditional channels. Until recently, this potential has been rather inaccessible, but working with natural language processing and analysing text and speech as data, we can now start generating and acting on these new insights.

Introduction

The big data revolution has been raging for a few years. And at roughly the same pace as organisations have begun setting up structured data warehouses for reporting purposes, new and exciting data sources such as text and audio are to a higher extent seen as valuable sources of information. These data sources hold a potential for creating value in everything from customer journeys to public services, and they often contain value and insights beyond what traditional tabular formats do. In other words, audio and text data contain a huge potential for generating – and acting on – insights arising through traditional channels that have not traditionally been as accessible as they are right now.

What is this guide, and who is it for?

In its entity, this guide is for people who can envision some of the potentials that lie embedded in text or audio data but who need more comprehensive insights into potentials, requirements and/or methods. Covering these topics, we know that it is necessary to address complex concepts and their impact, and we acknowledge that certain topics may be (too) difficult for the uninitiated. Therefore, we have decided to split the guide into two parts:

The first part is aimed at those recognising natural language processing (NLP) as a fast-emerging field but who still wonder what sort of text is relevant, which use cases are possible, where they can find text data and how the field has progressed – but who at the same time do not need to understand how or why computers interpret text as they do.

The second part is aimed at the technical side of NLP. It is for readers who are curious about the technical aspects such as how to move from text to 0s and 1s that are readable by a computer and how to approach the complexity that inherently lies in working with text and speech as data.

Why this guide?

At Implement, we are experiencing an increased demand for data science competencies in general, and the

analysis of text as data (often labelled as natural language processing or NLP) is a big driver of that.

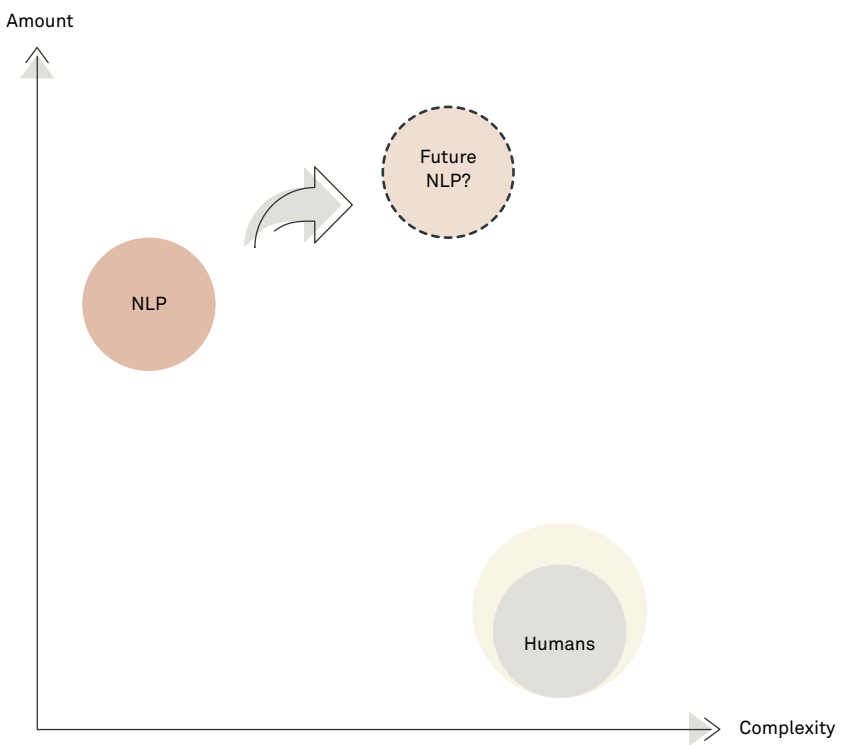
Part one

What is NLP, and how can it be approached?

In this first part of the guide, we will be covering:

- What natural language processing is.
- How broad the field is, and how it has changed over the past years.
- How NLP can be approached from a data source perspective.

First, though, we should agree on when to use NLP – and maybe most importantly, when not to. As with so many other things, we experience a trade-off between approaching a task manually versus deploying advanced analytics. The true value of text analytics lies in the analysis of very large quantities of text, but current NLP solutions cannot grasp the complexities of language to the same degree as a human reader would be able to. Thus, with the use of NLP, we must sacrifice some precision in the analyses, but we are able to conduct our analysis over thousands, or perhaps even millions, of documents. In addition, we see larger and better language models being published at a rapid pace, so perhaps the future of NLP looks like what is depicted in the figure below?



Despite previous warnings, the field has come a long way in recent years. Across our projects, we have already seen natural language processing being used to realise some of the following potentials:

- Comparing very large quantities of text to find similar documents.
- Identifying which products are being negatively mentioned on critics' websites.
- Transcribing and segmenting customer contact based on what the customers need, e.g. help with billing, sales etc.
- Identifying whether a (collection of) document(s) is compliant from a regulatory standpoint, mentioning the required entities.
- Creating a decision support tool that is based on historical case facts, yielding a consolidated overview to increase the quality of future case decisions and processes.

However, NLP is not only becoming more valuable in organisations and professional work. As a matter of fact, you probably encounter NLP several times a day:

- When contacting the website on which you ordered a pair of shoes where a chatbot can look up and return your order status.
- When you search on Google, advanced NLP models enabled by semantic search ensure that the responses you are looking for are found and are at the top of the search results.
- You probably receive less spam in your email inbox because NLP enables advanced spam filters.

- Your doctor's office probably transcribes the doctor's notes and maybe even generates precise medical assessments based on a description of symptoms.

Many organisations have large – and steadily increasing – amounts of natural language data that are ready and waiting. Examples include public sector requirements for documentation, e.g. of meetings with citizens, decisions regarding citizens' rights, hospital logs and much more. In the realm of private organisations, customer contact often occurs via chat(bots), phone calls and/or emails to and from customers.

In a customer experience situation, many organisations do not have a clear idea of their customers' pain points until they ask for explicit feedback from customers in the form of surveys or similar. However, if they use NLP instead, it becomes possible for the organisations to collectively listen to worries, experiences and/or praise, making it possible to be even more in touch with customers and citizens by alleviating problems and, in turn, prevent churn, dissatisfaction or similar at a higher rate.

Perhaps the most obvious point here is that these data sources and applications serve the purpose of automating tasks otherwise to be handled by humans. Instead of spending time and resources on classifying or summarising insights, employees can spend time on other value-adding activities.

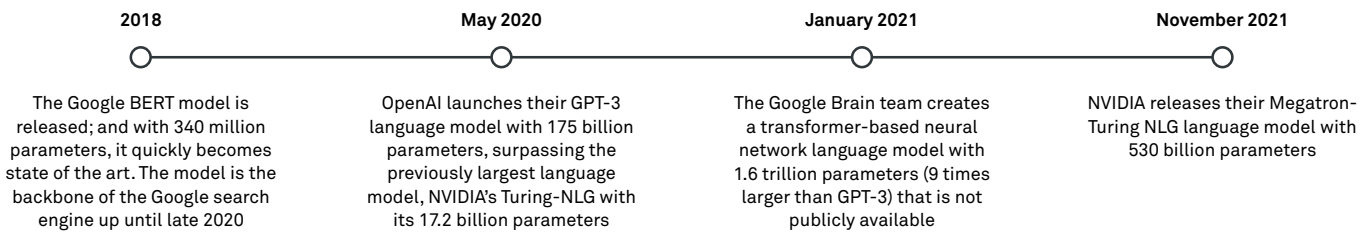
A rapidly developing field sparking organisational change

The field of NLP is changing at an extremely rapid pace, and researchers are developing ever-improving and impressive language models that can solve an increasingly large number of problems.

The timeline below highlights some of the biggest advances made in the field in the last couple of years:

Based on this past development, Geoffrey Hinton, the 2018 recipient of the Turing Award and one of the "godfathers of AI", was prompted to tweet that *"given the performance of the 175 billion parameters in GPT-3, we'd need a model with ~4.4 trillion parameters to know the answer to life, the universe and everything."* To this extent, the Google Brain model from January comes 36% of the way. The models and their underlying architectures have improved over the years; and with the massive increase in the number of parameters, the models become better and better and can be fine-tuned to support a multitude of tasks from answering questions, rewriting text, translating text etc. Any task involving text – and in almost any language – can be solved, as the increase in parameters and the improved architecture help models learn from the text they are trained on.

If we look at surveys such as the Gradient Flow 2021 NLP Survey and the IBM Data Science Community Survey in addition to the works of Gartner



and Chatbots Journal, it is clear that the NLP field is a catalyst for a lot of organisational change, just given these simple facts:

- NLP budgets grew by at least 30% for one in three tech leaders in 2020.
- NLP is in the top 3 of applications that data scientists prefer to work on.
- NLP is one of the most visible and popular forms of applied artificial intelligence. Since language is one of the most fundamental forms of intelligence, its benefits are widespread, well recognised and easy to understand.
- Gartner expects the chatbot market to plateau within two years as a testament to how far the field has come but also as a means of generating even more data – and at that point, it is very likely that, in the future, contact with your bank, insurance company, healthcare provider etc. will all be driven by chatbots, as Gartner predicts that 70% will be interacting with conversational platforms every day.
- The size of the global conversational AI market is expected to grow with a 30.2% CAGR from today until 2024, mainly driven by virtual assistants and chatbots.

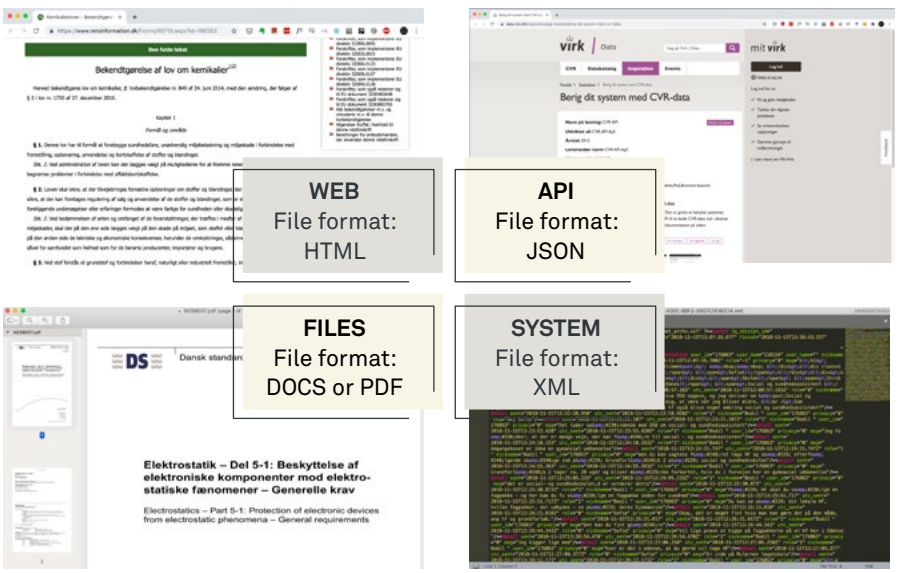
Our introduction to this guide would not be complete without our main source of inspiration: NLP is easy, and NLP is intuitive. Even if it does exist at the intersection of computer science and linguistics ("computational linguistics"), text is easy for our brains to comprehend; and thus, it becomes easy to explain difficult NLP concepts to peers and clients. As far as NLP goes, a lot of the insights we present are, if you think about it, commonsensical. When an adjective is mentioned before a noun, it is likely that the adjective is describing the noun. So, we may rightly ask (e.g. in the context of a customer review analysis): "Which nouns or entities are mentioned using particularly positive or negative adjectives?" We do not need much else before we can start analysing; and before long, we may be able to give an initial answer to what works and what does not work, e.g. across sales processes or supply chains. We hope that this and future guides will enable your organisation to do exactly this quick transition from curious questioning to thorough analyses supported by NLP.

Locating data sources

What is text, and where do we find it?

So far, we have established that the amount of text available to companies is exploding. But where is all this text located?

Text exists in many places, but some of the most common places are illustrated in the figure below:



Often, text exists in a structured, readable format such as large amounts of PDFs or Word documents, but sometimes the text is a bit more hidden and only accessible through system dumps.

It could also be the case that the combination of data sources renders the best result, and your analysis might benefit from adding information available through websites.

It varies whether websites have displayed their content in a structured format through API or if it is necessary to scrape the website.

Scraping is the process of extracting data, usually from websites, by writing codes that can access the contents of the site in the underlying HTML structure. This way, the text available online can be consolidated and parsed into any given programming language ready for further analysis.

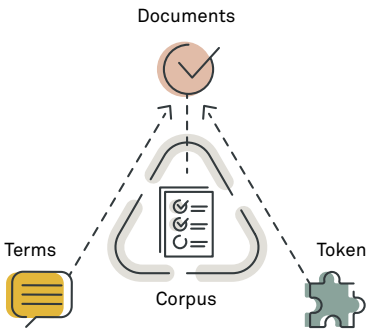
No matter the origin, the first step is to locate text and parse it into the computer. Each data source requires an individual parser to ensure proper load of data, and a lot of the preparation time for a text analysis lies in this initial step.

Part two

Understanding the technical background of NLP

In this second part of the guide, we will dive into how NLP is to be performed, and we will touch upon some of the most relevant considerations. Thus, we now head into the technical side of the guide where we turn to the following topics:

- Pre-processing: From verbal and syntactic to aligned and generalisable text.
- Mathematical representation of text: How text is represented as 0s and 1s.
- Speech as data: How verbal and written communication vary, and what it means.



Establishing common terminology

To begin the journey of working with text as data, you need to establish a common vocabulary of the data set. So, here follows some common NLP terminology that will be useful through the rest of the guide. Our collective text data set is called a *corpus*, which is made up of a collection of *documents*.

A *document* in your corpus does not need to be a *document* in the classical sense. The granularity of your data set depends on the analysis it will be used for. If the analysis to be made provides more value if applied to each section of a given document or to each individual sentence, then the *documents* in your corpus will be a collection of those sections or sentences.

Each document consists of a set of *tokens,* which is just what we call a word or a punctuation when it is being represented as a single unit in computational language. Gathering all unique tokens thus gives us a vocabulary of the data set.

Lastly, words are often referred to as *terms* in NLP and, when parsed to a machine-readable format, could be thought of as features as we know them from conventional machine learning methods.

Pre-processing

Introduction

With the text now being parsed and available, the next step is to clean the data and do some basic pre-processing – all done to streamline the text as much as possible. The following section will describe the pre-processing of text for some of the simpler analysis methods. The approach when working with advanced language models differs and is not in scope for this article.

It is necessary to do pre-processing to ease the task of understanding for the computer. However, it comes with a trade-off. The task of pre-processing almost always removes some of the semantic meaning of the text, as this is an inevitable consequence of preparing the text for computer analysis. In other words, we need to sacrifice some precision if we want to do structured analysis of hundreds or even thousands of documents.

Three basic steps in pre-processing

Let us dive into the basic steps of pre-processing with a working example. Consider the sentence:

"The plane is not an option, as the bus is obviously the more environmental choice!"

The first step would be to tokenise the sentence, making every word or punctuation a unique unit:

{ The, plane, is, not, an, option, as, the, bus, is, obviously, the, more, environmental, choice, ! }

Now we need some alterations to standardise the text to the highest degree. If we were to parse this to the computer, it would assume that "environmental" and "!" carry the same amount of information, as they are both represented as a token in the text. We know that is not the case, and the first step would therefore be to **remove punctuation.** However, all great rules are a product of their exceptions; and thus, punctuation is not always removed. More on that later.

Next, the computer would handle "The" and "the" as two different tokens. We know that there is no meaningful difference between the two. One just happens to be at the beginning of a sentence; and hence, the second step would involve **making everything lower case.** By setting all our text to lower case, we ensure that the significance of a given word is representative of the text and not skewed by sometimes being placed at the beginning of a sentence and, hence, written with an initial capital.

Furthermore, we want to **remove stop words.** The process of removing stop words is an attempt to help the computer focus on the parts of the text that provide the most information. Stop words include "and", "the" and "it", i.e. words that tell us nothing of particular interest to the content of the document we are working on. Their main purpose is to provide semantic meaning, which we have already decided will be our cost of cleaning.

Our sample sentence now looks like this:

{ plane, option, bus, environmental, choice }

We recognise the semantic loss of the sentence, but we also see that the steps above helped us identify the important aspects of the text. We still have the essence showing us that the text is about planes and buses, something with an option and a choice while considering the environment.

The above steps constitute the basic hygiene for cleaning text, but you should apply them intelligently. There may be some types of analysis where the information loss introduced by this cleaning is significant to the accuracy of the analysis. This could be sentiment analysis, in which punctuation might be a clue about an emphasised sentiment, and hence a means of a more accurate analysis. As punctuation may carry valence, the phrases "I don't like it" and "I don't like it!" should perhaps get different sentiment scores, as the latter carries a stronger negative emotion.

Stemming vs lemmatisation

Apart from the three basic steps described above, you can also apply more intelligent pre-processing. Enter stemming and lemmatisation.

As with the other methods, stemming and lemmatisation also try to standardise the text. Their main purpose is to ensure that words like "processing" and "processed" are mapped as the same token in our final vocabulary. Again, we see a loss of semantic information as we, in this case, no longer will be able to tell when the given activity took place. However, we are ensuring that "to process" something is up-weighted in our vocabulary.

How exactly the two tenses of "to process" are standardised is the differentiator between stemming and lemmatisation.

Stemming

Stemming is by far the simpler of the two and is trying to reduce words to the word stem. This is a rule-based approach. We could have designed specific mappings based on defined prefixes or suffixes, and this could be the verb tense suffixes such as "-ed" or "-ing". With this rule in stemming, the words "processing" and "processed" both become "process".

However, in some cases, stemming can return a stem that is not a word in itself, e.g. when the word "trouble" is stemmed to "troubl". This is an example of *overstemming.* When a word is overstemmed, too much of the word is cut off, resulting in a nonsensical word, or different words with different meanings are stemmed to the same word. A classic example of this is the words "universe", "universal" and "university" all being stemmed to "univers". However, the opposite can also happen, in which case the word is said to be *understemmed.* In this case, similar words are stemmed to different stems with another classic example of "alumnus" → "almunu", "alumni" → "alumni" and "alumnae" → "alumna", all of which should return the same stem.

Lemmatisation

So, having established that stemming can get us some of the way in our quest for standardisation of our text, we also recognise that a more intelligent approach could be preferred.
This is where lemmatisation comes into play.

When looking up a word in a dictionary, it is the lemma of that word which is printed. Reducing a word to its lemma cannot be accomplished by applying a set of rules, as the given word's part of speech is necessary to derive the correct lemma. And to this extent, we are in need of a part-of-speech tagger.

A *part-of-speech tagger* is trained on a large corpus of text and is taught a statistical probability of a tag for a given word based on the context in which it appears and the context in which it has appeared in previously seen data. This allows us to label words, e.g. as nouns, verbs or adjectives, which, apart from being a vital component in lemmatisation, can be used in other linguistic analyses.

Using part-of-speech tagging in lemmatisation allows us to reduce "are" and "is" to the lemma "be" and recognise that the noun "universe" and the adjective "universal" should be reduced to different lemmas.

Since lemmatisation is a bit more advanced than stemming, it also requires more computational power and is thus slower. Though with NLP packages in modern programming languages, this is negligible. However, using lemmatisation comes with some requirements for your corpus, as the proper semantic order of words is needed to correctly use part-of-speech tagging to reduce words to their lemma and will thus not work if the text is batched or has a lot of gaps in it.

This concludes the basic steps of pre-processing your text for further analysis. Next, we need to define a way to represent the text so that our machine has a chance to understand it.

Mathematical representation of text

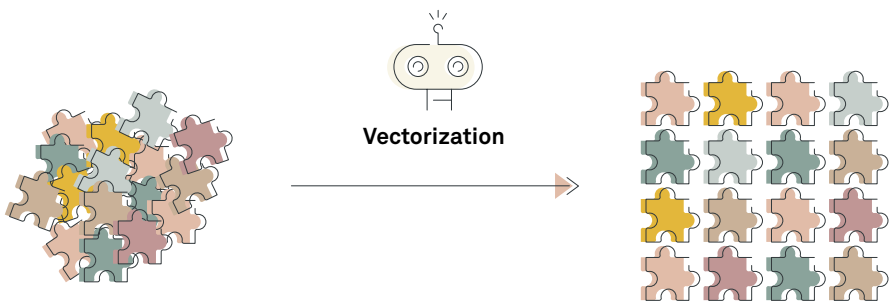
Introduction

With the newly cleaned and pre-processed textual data in hand, you face the ultimate challenge: computers, codes and machine learning models do not understand text in the same way that we humans decode text. For instance, when we read the word "bear", several associations help us decode the meaning of the word. In this case, we are talking about a large, furry and dangerous animal that eats fish, lives in forests and has a fierce roar. To make it even more complex, "bear" might also refer to other situations, such as a burden that is hard to bear, so we may need to decode an entire sentence before we know which "bear" is referred to. This is easy for us humans, as such linguistic associations are the result of millions of years of evolution, whereas our computers and code interpreters start with zero predispositions to understand the meaning of words.

So, the key question we need to ask ourselves is how we in the best possible numeric[1] way can represent textual data so that it is usable in our analyses but to the highest degree possible maintains the semantic meaning.

Vectorization

Discussing the process of preparing text for computer processing, you may encounter the verb "vectorizing". Basically, the verb refers to the process of transforming sentences or documents into machine-readable tables ("vectors"). A good way of thinking about this is to see these vectors as simple tables or as mediums that we utilise to send data into computers with a specific, required structure.



[1] By numeric, we refer to methods that make textual data readable by computers. This demands numerical inputs, e.g. 0s and 1s.
[2] In the following, we work with data on a low level of granularity – sentences (i.e. we have a corpus containing several sentences). It could also have been documents, pages etc.

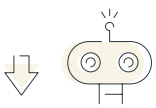
In the following, we will present a few word vectorisation methods. Initially, they rely on the so-called **bag-of-words assumption**: that term order is not relevant. In other words, we can split up sentences by the terms contained in them, throw all of them into a bag and draw them in any order – their presence, not placement, is key. Obviously, neglecting term order means that we lose valuable information, i.e. if we had one sentence describing two different entities in a negative and a positive way, respectively, we could not map adjectives and entities after throwing the words "into the bag" by vectorizing our text. Later, we will loosen this assumption.

Term frequency (TF)

One way of representing text is in a standard count vector. Simply put, we create a table containing the unique terms from across all our textual data sources[2], and then we count how many times each word occurs across the sources – most often yielding a rather sparse vector containing a lot of "empty" columns with 0s.

This is exemplified in the below sentences that are vectorized into one table:

- The man helped the woman cross the street
- The woman drove the car down the street
- A duckling swam across a lake

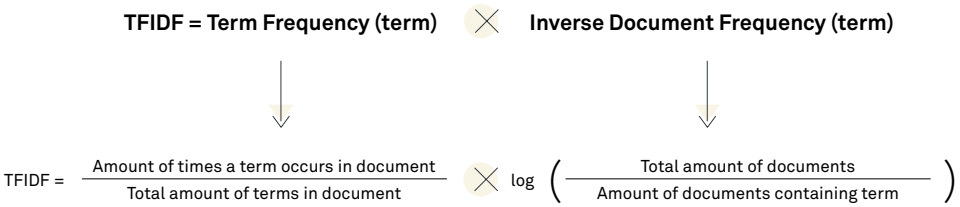


TF vectorizer

Document	The	Man	Helped	Woman	Cross	Street	Drove	Car	Down	A	Duckling	Swam	Across	Lake
#1	3	1	1	1	1	1	0	0	0	0	0	0	0	0
#2	3	0	0	1	0	1	1	1	1	0	0	0	0	0
#3	0	0	0	0	0	0	0	0	0	2	1	1	1	1

A noticeable drawback of term frequency vectorization is that term rareness is not considered. For instance, if we had to use the vectorized text data to assess the topics embedded across our corpus (here, "corpus" refers to the full vector (table) of the three sentences above), perhaps we would like to weigh up words that, although infrequent across the full corpus, are extremely important – rare *and* defining – for a few documents. In this example, "lake" in document #3 is rare because it does not occur in documents #1 or #2, and it is defining, as it makes up 16% of the full document length.

To put it another way, the issue with term frequency vectorization is that the rare words that distinguish documents from one another are weighed equally to all other words contained in the corpus. So, what we need is an approach that lets us weigh our vector in a more relevant way, up-weighing rare and defining words and down-weighing common, general words. Combining term frequency and so-called inverse document frequency (TF-IDF) will help us achieve just that.



Document	The	Man	Helped	Woman	Cross	Street	Drove	Car	Down	A	Duckling	Swam	Across	Lake
#1	0,07	0,06	0,06	0,02	0,06	0,02	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
#2	0,07	0,00	0,00	0,02	0,00	0,02	0,06	0,06	0,06	0,00	0,00	0,00	0,00	0,00
#3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,16	0,08	0,08	0,08	0,08

Term frequency inverse document frequency (TF-IDF)

The TF-IDF vectorizer helps us weigh terms differently so that common terms that appear in many contexts (such as "the", if not removed in the earlier steps of this guide) will be given a lower weight and thus lower importance than relatively more rare words like "lake", even if "lake" appears more infrequently across the entire corpus.

To utilise the vectorizer, we apply the formula below. On the left side, we calculate the term frequency; and on the right side, we calculate the inverse document frequency (the base logarithm of the number of documents divided by the amount that contains the term in question).

And if we calculate the TF-IDF vector for our three example sentences from before, they now look like this:

*As an example, the word "a" is very defining (assuming that we did not remove it when processing stop words) and is thus weighed relatively higher in the third document, as it occurs twice in the document and only exists in that document: (2/6 = 0.33) * log (3/1) = ~0.157. The impact of IDF is also shown in the term "street", as it occurs in two out of three texts in total; and thus, the right side of the equation becomes log (3/2), yielding a lower weight, as the term is not rare when assessing all the texts in our corpus.*

In the above example, we have showcased why the TF-IDF vectorizer solves the issue of weighting up rare (assumed to be relevant) terms in our corpus; but still, with both TF and TF-IDF vectorization, term order is not considered.

Leaving bag-of-words models behind

One of the major drawbacks of utilising bag-of-words models, as touched upon above, is that word context/word order is not considered. Essentially, we can move away from bag-of-words models in two ways – and both ways require us to move away from treating text as single tokens ("unigrams"), thus increasing both precision but also analytical complexity.

N-grams

Essentially, n-gram methods entail converting texts into larger vectors that retain word order by splitting text into distinct sequences of N consecutive words. Consider the below sentence:

> ***The woman drove the car down the street***

With this sentence, we can create several types of n-grams that are showcased below:

Unigram: The, woman, drove, the, car, down, the, street

Bigram: The woman, woman drove, drove the, the car, car down, down the, the street

Trigram: The woman drove, woman drove the, drove the car, the car down, car down the, down the street

And so on for higher n-gram word levels.

The point is that by having both "the woman drove" and "drove the car", we could search for and analyse specific word combinations where e.g. "woman" comes prior to or after the term "drove"; and thus, we can make higher-level inferences than we would be able to if we had thrown several terms, including "woman" and "drove", into the same bag of words. This enables us to find out if someone else drove the woman in question or if the woman in question drove a noun (e.g. a car, a bike etc.), as term order is now considered.

While using n-grams yields the clear benefit of retaining term order, there are still two drawbacks of the method. Firstly, it does not solve the issue of high dimensionality in the feature spare, making it more difficult to process and analyse as the amount of data increases. Secondly, using n-grams can increase the complexity of inferring and making decisions based on data, as several similar n-grams may yield a murky picture of the text at hand. Because of these drawbacks, and because more sophisticated methods were developed and made accessible, n-grams are not often utilised[3].

Enter word embeddings

If we were to credit one "invention" with a long range of NLP advances over the past decade, word embeddings surely would appear on the list. Among other things, the word embedding way of representing text as numerical input for analysis and/or models has enabled the transformer architectures, paving the way for state-of-the-art language models such as BERT, GPT-2, GPT-3 and the Megatron 530B. This has had such a large impact on the NLP scene that whenever you undertake almost any NLP project, odds are that one of the first things you do is to download pre-calculated embeddings (for "standard" NLP problems) or start gathering data for training/or devising custom word embeddings.

The problems solved by word embeddings

In the rundown of TF and TF-IDF vectorization above, we had three sample sentences; and even with these short sentences, there was a lot of "emptiness" (0 values), leaving us with what is called a sparse feature vector. When using either of these approaches, we essentially create a one-hot encoded vector, leaving us with a lot of void space or sparseness. For example, if we had had 10,000 policy documents from a legislative body, we would have an even sparser feature vector. Such sparse feature vectors are not ideal as a text input representation method; but in some specific situations, they can be optimal over word embeddings (i.e. dense vectors)[4].

On a broader note, word embeddings solve most of the problems you would experience using one-hot encoded feature vectors. These solutions are outlined in detail in the text box.

[3] That said, there are some situations, e.g. when working with "small" or less-supported languages, where n-grams can improve the quality of analyses.
[4] If you have a small amount of input features, if you do not expect input features to be correlated in a semantically meaningful way, and you have a large amount of data to train your model on. See e.g. Goldberg (2015). Of course, one-hot encoding of categorical data such as text labels should still be performed, as models cannot deal with non-binary data. https://arxiv.org/pdf/1510.00726.pdf

Issues solved using word embeddings

Semantically similar words such as "bus" and "car" are treated as distinct features in one-hot encoded vectors. Suppose we could perform feature engineering so that we put "1" (or the corresponding TF-IDF value) into our vector if any mode of transport is mentioned in a document. We would forego the luxury of knowing whether it is a car or a bus being mentioned, but we would be able to compare texts much better to assess whether modes of transport are mentioned or not. When performing TF or TF-IDF vectorization, however, the term "bus" is as similar to "car" as it is to "bear" or any other term (i.e. not similar!).

Vocabulary sizes

The mode of representing a sparse feature vector is by having texts represented in one row each and features (terms) in one column each. If t new documents are added containing k new number of terms, the size of the feature vector increases by k^t. So, as more data is sampled to the text corpus, the feature sizes can increase drastically[5]. This means that one would have more model parameters to estimate; and thus, exponentially more data is required to estimate those parameters well enough in the hopes of building a generalisable model.

Computational needs

As the feature vector is most often very sparse, one will often end up with a very large feature space (size of rows x columns). This will take a large load of memory resources and potentially lead to storage concerns. Furthermore, many machine learning models do not work well with high-dimensional, sparse feature vectors.

Generalisation

The aspect of generalisation is the key argument for using dense word representation. If we assume that words like "bus" and "car" are contextually and semantically similar in our NLP problem, we need some way of letting our model know. When a word embedding model is trained, the model learns how to treat singular terms. This means that when the model learns "car" or "bus", these terms are passed through neural network transformation layers that have corresponding biases and weights. So, when the model turns to learning semantically similar words, the training paths of "car" and "bus" may help the model learn what "Ferrari" is, assuming that they share feature embeddings because the term Ferrari then is placed on a similar "path" through the neural network layers instead of the model having to learn its meaning from scratch. This helps models trained using word embeddings to generalise better, as similar feature embeddings help "connect" similar terms.

[5] In Bengio et al. (2001), they describe this as the "curse of dimensionality" and pose an interesting example: If you have 100,000 terms in your vocabulary and a sentence that is 10 terms long, you would need 100,00010 parameters to predict the 11th term.

What does word embeddings look like?

Let us start with a simple-term, frequency-based, one-hot encoded feature vector like the one we saw a few pages back; but for the sake of the example, we will now transpose it:

	Document 1	Document 2	Document 3	Document 4	Document 5	Document 6
Bus	1	0	0	0	...	...
Car	1	1	1	0	...	...
Bear	0	1	0	1	...	...
Panther	0	0	0	1	...	...
Black	1	0	1	1	...	...

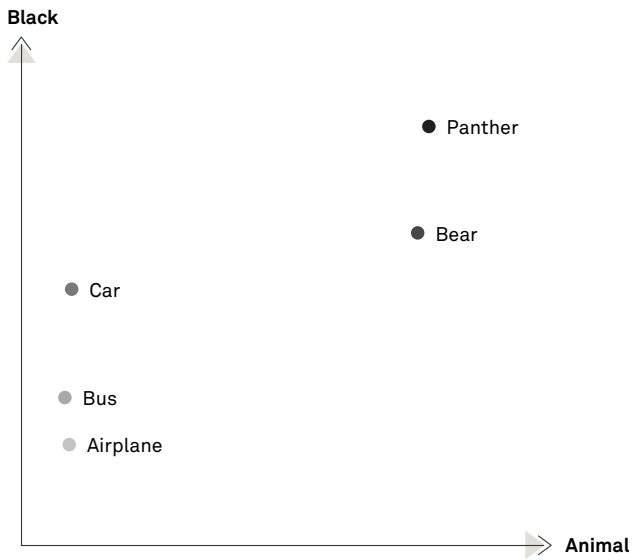
There are probably a few key dimensions we could summarise these terms by. For example, we can utilise that while cars can be dangerous, so can bears. Similarly, panthers and airplanes may both be perceived as fast etc. This intuition is what word embeddings use to realise the many potentials mentioned above:

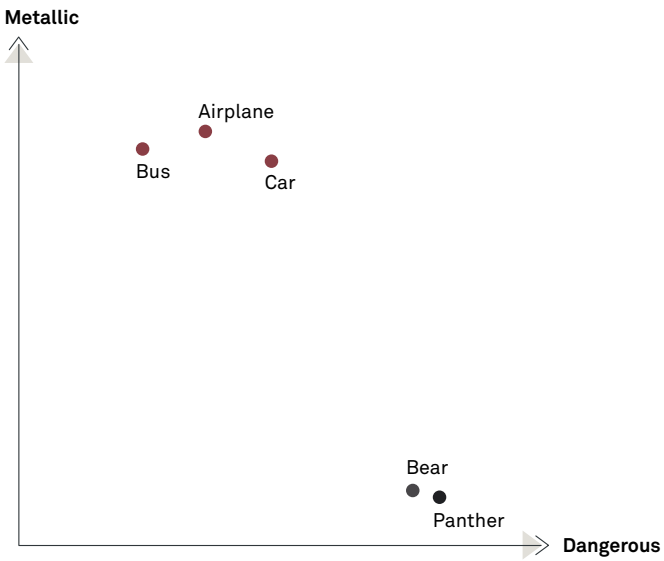
	Metallic	Fast	Dangerous	Animal	Black
Bus	0.82	0.45	0.21	0.01	0.21
Car	0.79	0.60	0.43	0.03	0.44
Bear	0.02	0.40	0.67	0.91	0.56
Panther	0.01	0.88	0.71	0.93	0.78
Airplane	0.85	0.91	0.32	0.02	0.11

This means that we can now describe terms based on their related word embeddings. So, a panther is .88 fast, .93 animal and .78 black.

Now, if your perception at this point is that "this does not make any sense", but you at the same time understand why the feature score of black x panther is .78, then you are following along totally okay. The point we are getting to is that these features can be thought of as a single location in our word embedding higher-dimensional feature space. **This means that we become able to show, and thus pass onto our models, that some words are similar based on their feature similarities and not how the words appear.**

Have a look at the graphs below and see if you can figure out why the words are clustered as they are in this fictive word embedding space (e.g. the "black" x "animal" space and the "metallic" x "dangerous" space).





In the examples on the previous page, we show how the terms map onto a two-dimensional word embedding feature space. In real-world word embedding examples, we would likely have n dimensions, but the advantages become quite clear. Even though the words "panther" and "bear" are not similar at all, the model with our embedded inputs knows that they are not similar to buses, airplanes or cars if we look at the "metallic" x "dangerous" embedding space on the right (meaning that we no longer even need the *animal* embedding). Now, imagine doing this in a 30-, 100- or even 300-dimensional space, mapping all the terms in your corpus onto these feature dimensions. It would be a foolish endeavour trying to visualise it[6], but the terms' placement in the feature space conveys meaning and semantic context and allows for better model generalisation.

This allows us to assign a placement in the word embedding feature space to each term in our documents. For now, the example shows how it is possible to describe the term "bus" as .82 metallic, .45 fast, .21 dangerous, .01 animal and .21 black. This has a huge impact, as word embeddings are much smaller than one-hot encoded vectors, they convey greater similarity and are more generalisable, and we can calculate them with much lower computational cost than traditional vectorization methods.

The above example relies on adjectives, but it is also possible to create word embeddings, e.g. using part-of-speech tags (the embeddings would consist of nouns, pronouns, verbs, adjectives etc.), named entity recognition (the embeddings would consist of organisations, people, places etc.) as well as other formats.

Word embedding algorithms and how to train them

The first word embeddings were created back in 2001[7]; and as 20 years is a lifetime in the NLP field, word embeddings have developed a lot. Nowadays, it is very easy to download pre-trained word embeddings from the internet (such as GloVe, FastText, GigaWord, CoNLL etc.), but it is also possible to train your own on a data set of your choice to fit embeddings to your specific problem. This latter part of assembling a clean and ready-to-use data set is probably the most difficult part of making your own word embeddings.

As far as algorithms go, there are plenty open-source word embedding algorithms out there that are easy to use (see e.g. word2vec, GloVe, ELMo, BERT). Also, the great part is that most of the algorithms work as essentially unsupervised feature extractors. This means that the algorithms calculate features based on terms that are used in the same contexts on their own – following our example on the previous page, perhaps airplanes and panthers are both described as fast. Of course, this depends on the type of word embeddings you are creating.

[6] Although, in fairness, it can be done. Usually, one would then convert the n-dimensional feature space into e.g. a two- or three-dimensional space using dimensionality reduction methods such as t-SNE or PCA.
[7] Bengio et al. (2001) is a great read if you are keen to learn more about word embeddings.


TO BE
CONTINUED

Speech as data

Introduction

In speech analytics and speech as data, audio is the key component. It might be the words you say to your personal virtual assistant in your phone, it might be the complaints your customers are calling your phone hotline with, or it might be recordings of doctors' notes or journalists' interviews.

Speech analytics covers the process of extracting meaning from audio data to find relevant business intelligence insights, most often utilising NLP tools for processing, extraction and subsequent analysis.

In this short section, we will discuss the types of speech data you might find, where and how it may be gathered and why it can be a unique driver of NLP-fuelled business impact.

Types of speech data

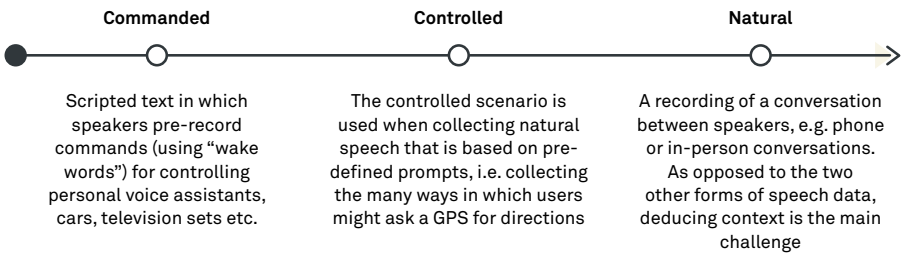
When we approach speech as data and the collection of it, we can do a coarse segmentation of text onto a spectrum ranging from unnatural to natural speech – at the one end, unnatural and command-like scripted text versus natural and non-scripted text in free conversation at the other end.

The first two types of speech data outlined above are mostly related to the training of voice assistants and are set up to maintain control of user inputs. On the other hand, the third type of speech data, natural speech, is where we see the most value existing within speech analytics, as this is the type of speech data used in interactions with customers, patients etc.

Collecting speech as data

Being successful in the domain of speech analytics relies on having access to strong speech recognition technology (speech-to-text algorithms and parsers). These are essentially algorithms that can translate analogue sound input into word vectors by 1) converting sound waves into a digital format (phonemes) and 2) arranging phonemes into coherent sentences using neural networks or Markov models.

Just like with word embedding technology, you could go and train your own speech-to-text algorithm, but plenty of pre-trained services and software already exist, providing a much faster way of assessing the technology and its potential. As an example, you can access speech-to-text algorithms in most cloud environments such as Microsoft Azure, Amazon Web Services and Google suite. You can also reach out to third-party vendors that have spent time and money on developing their own algorithms and often also include business intelligence tools or can help pinpoint relevant use cases. However, if you decide to train your own model, it is not as daunting as it sounds. You will likely use a pre-defined framework such as Facebook's Wav2Vec 2.0 that can create speech-to-text vectors based on audio. What makes the task daunting is that you will need to sample and record a lot of high-quality speech that can be used. Several thousand hours of speech, both labelled and unlabelled.



Community help

Acquiring high-quality speech data takes a long time, and even if you dedicated yourself to recording and validating a lot of your own speech, you would quickly face generalisation issues because phonemes can differ vastly across regions, genders, ages, dialects etc. Because of this, some communities have begun helping each other secure representative and high-quality speech. Anyone can help with this, and if you want to contribute to either speaking or validating a few words, please see https://commonvoice.mozilla.org/da for Danish or https://commonvoice.mozilla.org/en for English.

Processing speech as data

Let us venture past the process of determining which speech to use, how to collect it etc. and assume that we now have raw speech data stored as text in a database.

Speech recognition systems are not (yet) 100% precise; and therefore, there is going to be a certain level of error, often calculated as word error rate (WER). This could mean that some words are missed or written incorrectly, which makes it difficult to keep as high a level of detail as possible when using text as data. For one, you may need to spend additional time on pre-processing the text and sorting out problems manually, since the voice recognition algorithm may consistently mistake the word "again" for "akin"; and so, you will need to spend time on listening and fixing. Secondly, there are certain pre-processing and analysis methods that rely on part-of-speech tags that will likely become misleading if certain words are missing, as previously covered.

So, using pre-trained word embeddings is not great if you are working with incomplete speech data.

Consider, for example, the difference between the two sentences below:

1. It's the bear
2. Not to bear

Let us assume that the voice recognition technology gets the second term wrong in both sentences, i.e. "the" and "to". If this is the case, our word embedding creation has no idea whether this refers to a bear of the kind that is dangerous and animal-like or if it refers to the verb "to bear".

Finally, as previously noted, there are situations in which we may choose to retain punctuation in our text to perform certain analyses. When working with speech as data, we do not have the luxury of using writers' gestures, signalling questions with a question mark or particular (dis) satisfaction with an exclamation mark. Another example is delimiting when sentences end. We do not have full stops, and so we recommend simply delimiting by speakers (e.g. when speaker B interrupts speaker A, a new sentence begins).

This means that we may miss out on meaning conveyed by punctuation, and thus – and this is normal when working with speech as data – we need to be more creative. This omission of punctuation means that we no longer have strong guidelines for identifying questions, valence and sentence structures.

Why speech analytics?

Despite the challenges, speech analytics and speech as data hold a lot of promise. In the realm of customer experience, it is said that only 1 in 26 dissatisfied customers complain – the rest churn. From that angle of customer experience, figuring out what causes dissatisfied customers when they spend time on calling is time and money well spent because it helps enable action before customers churn. You can become more knowledgeable about the customer journey – which touchpoints work/do not work – and generally bring a lot of input on key dimensions for which you have previously relied on manual input or customer surveys.

Speech analytics can also enable real-time fault detection. For example, if 10 customers all call within a short time frame experiencing issues with your webpage, you can get IT started before an in-house person notices the issue.

Taking a broader perspective, we see a lot of organisations that have not yet started the journey towards speech analytics. The important thing to note here is that you can start the journey in steps; and even if we advocate for identifying goals early in the process, it is important to note that for each day you do not stream and store conversations with patients, customers and employees, you miss out on a lot of relevant data.

Also, there may very well be future forms of analysis that have not yet been developed; but when they do, having that data at hand becomes very valuable.

Top five tips for working with speech analytics

1. Identify goals and consider alignment with strategic initiatives.
2. Determine technological wants and needs and pair with potential vendor selection.
3. Build and track KPIs related to identified goals and strategic initiatives.
4. Assess and build competencies across the organisation.
5. Design processes and ensure that solid governance is in place around the solution.

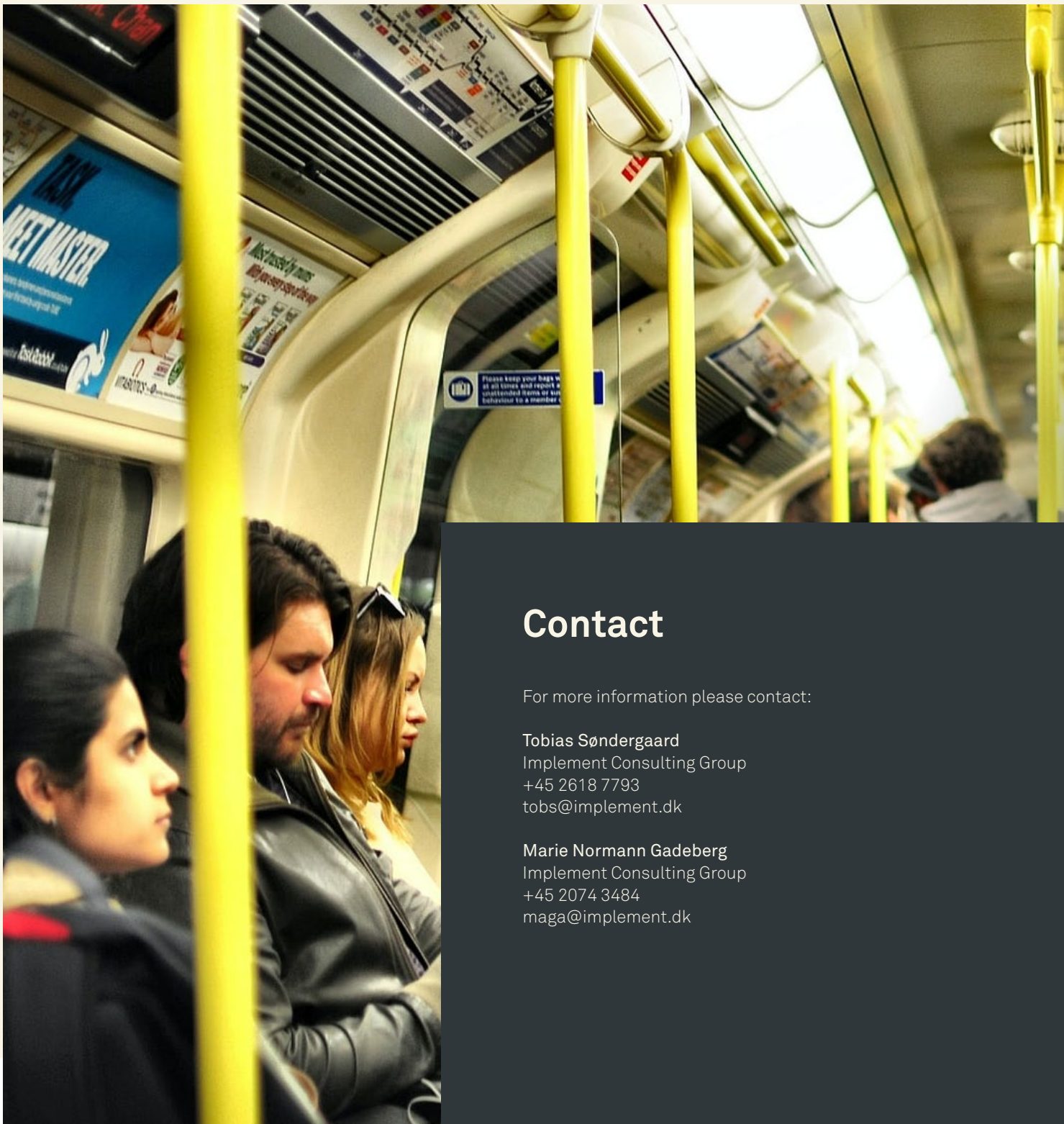
Conclusion

With this guide, we hope to have shown you how rapidly the NLP development is moving but also the value it can bring you to get on board quickly.

This guide has given **an overall introduction to handling text as data** – but there are more steps to follow. Once you have parsed the speech or text, performed initial pre-processing and made a mathematical representation of your text, **you now need to perform a structured analysis.** To do this, you will need to be able to identify which NLP methods and tools provide the most value for the business problem that you are trying to solve.

In our next guide, we will dive deeper into some of the readily available NLP methods that, when combined, can help you solve a wide range of problems and create value in your business or organisation.

If you have any thoughts, questions, wishes or corrections, or maybe you are just curious about getting started on the NLP journey, please do not hesitate to reach out – we would love to hear from you and help where we can.



Contact

For more information please contact:

Tobias Søndergaard
Implement Consulting Group
+45 2618 7793
tobs@implement.dk

Marie Normann Gadeberg
Implement Consulting Group
+45 2074 3484
maga@implement.dk