# From Words to Numbers - Simple NLP Techniques to Unlock the Data in Free Text

LIDA Data Science Intern Sam Relins

One of the most exciting and fastest growing applications of machine learning is NLP - Natural Language Processing. From the virtual assistants in our phones to the spam filters hiding behind all of our email services, NLP is integral to many of the services and applications we use every day. Though the headline-grabbing applications can be as complex as they are fascinating, you don't need to spend months mastering LSTMs and transformer networks to use many of the powerful tools in the NLP toolkit. In this article, we'll explore the basics of free text analysis, and see how these simple techniques were instrumental in my LIDA internship project that aims to predict treatment outcomes in a paediatric community care service.

**The Task**

Bradford Royal Infirmary is home to an innovative 'hospital-at-home 'service for children and young people, called ACE - the Ambulatory Care Experience. ACE is a fantastic service that offers an alternative to hospital referral for patients that require urgent care. They treat and monitor patients in their own homes in a "virtual ward", under the care of a consultant paediatrician. The service has been proven to significantly reduce the cost of urgent care, whilst delivering outstanding outcomes for the patients and their parents.

Key to this approach to urgent care is the triage process - determining which patients can be safely treated at home and which should be referred to hospital. ACE clinicians collect a range of clinical and non-clinical data for each referral, and use this information to make triage decisions. These decisions contribute to the successful discharge of 85% of the patients ACE treat, the other 15% requiring a subsequent referral to hospital.

The ACE team believe the referral data they collect may still contain key insights that indicate a patient's at risk of needing hospital treatment, and propose that data science and machine learning might be the key to their discovery. To test this theory, they have partnered with ourselves at the LIDA Data Science Internship scheme and Bradford Institute for Health Research (BIHR), in a project that aims to model the outcomes of treatment in ACE - using referral data to predict which patients will require hospital treatment.

**Why text analysis?**

Our initial attempts to use the ACE referral data to model the risk of hospitalisation proved challenging. The referral data consist of a standard set of clinical observations that the service uses to indicate suitability for treatment at home, however the only data available for model training consist of **patients that were accepted for treatment**. In other words, our training data consists only of patients that were already decided to be of low risk of hospitalisation, and this decision was largely based on the observations in the training data. This makes the machine learning task extremely challenging - we are asking classification models to identify features in a dataset that trained clinicians have already tried, and failed, to find. Considering this, it is unsurprising that none of our attempts to model outcomes using the standard referral observations were successful.

Fortunately, the basic referral criteria aren't the only data we have at our disposal. Alongside the "structured" observations in the referral data - numeric features and "tick box" categorical features - are free-hand written notes. These text features capture relevant information on patient medical histories and examinations, that aren't captured in the observations that are recorded for every referral. Given that the information captured in these text features is absent from the standard referral criteria, it stands to reason that they aren't considered by default when making referral decisions. As such, they have the potential to indicate hospitalisation risks that haven't already been identified by the ACE team.

All that's needed is a way to statistically analyse free-text.

**How can you analyse text?**

When we think of data, we typically think of "structured" data - forms, excel spreadsheets, company accounts - and with good reason. Structured data is recorded in a manner that is easy to analyse. Given a spreadsheet of info on 100 people with a column named "age", the experienced among us can quickly work out their average age. Given a short 500 word bio for the same 100 people, that task would be considerably more difficult, if not impossible. Unlike structured data, determining specific information from free text is a non-trivial task, and there are no guarantees the desired information will even be present.

Fortunately modern data science has an impressive toolkit of techniques that can help us analyse text statistically. These techniques are generally captured under the umbrella of Natural Language Processing. Though the more complex applications of NLP - voice assistants, chat bots, and so on - tend to be more familiar, applications of NLP need not be so complex.

A simple application of NLP which we are all familiar with is a word cloud. The following is a word cloud generated from the "medical history" notes from the ACE data:


Word clouds separate text into individual (or pairs of) words, and display the most common words visually. Words that appear more often are displayed in larger font, and words appearing less often are smaller. By displaying text this way, we aim to gain an overall impression of a text or collection of texts, which words are "important" or "prominent" and which words are less so.

For example, we can see from the above word cloud, that the patients appear to be exhibiting a lot of cough or asthma symptoms. Words like "wheeze", "inhaler", "asthma" and "cough" are among those that stand out most clearly. We might assume, therefore, that the patients are being treated primarily for chest infections or asthma. This is an accurate interpretation of the notes, given that the ACE dataset actually describes patients that were treated on the "asthma / wheeze" pathway. We have quickly inferred some characteristics about the patients in the dataset using this simple text analysis, and our assumptions turn out to be correct.

Word clouds are a handy illustration of a methodology that underpins much of NLP - dividing text into its component parts and assigning numeric values to the parts. The word cloud quantifies the frequency with which words appear and then increases or decreases their size accordingly. This method is more commonly referred to as a "bag of words".

**Bag of Words**

"Bag of words" is a blunt, but apt, name for the process of splitting text up into words and counting the frequency of each word. It is most easily explained by example - if we take the sentences:

**Sentence 1**: "*the cat sat on the mat"*

**Sentence 2**: *"the dog sat with the ball"*

They can be represented by the counts of the words that appear in them like so:

| | "the" | "cat" | "dog" | "sat" | "on" | | "with" | "mat" | "ball |
|---|---|---|---|---|---|---|---|---|---|
| Sentence 1 | 2 | 1 | 0 | 1 | 1 | | 0 | 1 | 0 |
| Sentence 2 | 2 | 0 | 1 | 1 | 0 | | 1 | 0 | 1 |

This is a really simple, but powerful way to represent text data. The text is no longer a list of characters, but is now structured and represented numerically - it can be analysed and modelled in the same way as any other data found in a database or spreadsheet. Each sentence or note has been converted from a list of characters to a vector of numbers (**sentence 1 = [2,1,0,1,1,0,1,0]**) and so this process is often referred to as vectorisation.

But, before we go off excitedly vectorising all of our notes, it's a good idea to consider a few of the challenges that come with representing text this way.

**Pre-Processing**

Generating word counts is a great way to represent text numerically, but doing so with raw unprocessed text has some serious drawbacks. Thankfully a lot of these drawbacks can be addressed or mitigated by "tinkering" with the text before it is vectorised. This "tinkering" is usually referred to as pre-processing.

The following are some of the challenges of representing text when using the "bag of words" approach, and simple techniques that can help address them:

- **Common words**: By considering words in isolation, a large number of the most commonplace vocabulary becomes meaningless. Take the ten most common words in the English language: seven of them - *"the", "to", "of", "and", "the", "in", "at"* - are important for grammatical structure, but have little meaning when taken out of context. But, as they appear so frequently, these words come to dominate the vectorised counts of texts. To avoid analyses being filled with meaningless words, NLP pipelines will usually include a list of "stopwords" - words that are simply removed from text before it is vectorised.
- **Word variations**: Many words have a common root, but vary with different inflections or conjugations. For example, the words *"reading", "reader", "reads"* all stem from the root word *"read"*. By vectorising these words we treat them all as individuals, separate and independent of one another. The result can be many sparse representations of one concept - loads of individual words that largely represent the same thing - which get lost in among one another when they join the many other terms with multiple inflections. To address this issue, text is often "stemmed" or "lemmatised" in pre-processing. The two techniques differ in methodology, but effectively do something very similar - removing or changing the ends of similar words, reducing them to one common root - so *"reading", "reader", "reads"* all become *"read".*
- **Vocabulary size**: There are an impressive variety of words in written language. Representing a group of texts as a bag of words usually results in a vocabulary with a very large group of uncommon words that appear only once or twice. The larger the vocabulary the longer the vectors

are when the text is vectorised - longer vectors consume more memory and are harder to analyse. Because of this, it is common to set a finite vocabulary size such that the majority of the variation in the text is captured, but the extremely rare words don't eat up all the available computational resources and make analysis more challenging. Effectively, throwing away the least common words so that the analysis doesn't get too complicated.

- **Context**: Considering only words in isolation removes the context of the other text that appears around them. Because of this, “n-grams” are often considered when vectorising text (“n” being the classic placeholder for an integer number). Bi-grams, for example, are pairings of words. The aim is to capture words that often appear in the context of one another - good examples are adjectives followed by nouns, such as *"severe asthma"* or *"good results"*. Note: Because vectorised text often includes n-grams (or even small chunks of words in other NLP applications), we refer to the individual units of text as "tokens" to avoid confusion that might arise from calling them “words".

[*For practical application of the above methods in Python see the fantastic NLTK[1]. In particular* stopwords *from the* corpus *module and the* WordnetLemmatizer *and* SnowballStemmer *classes*]

As an example of these techniques in action, we can walk through the pre-processing of the medical history notes from the ACE data. We begin with individual notes, an example of which looks like this:

*"Recurrent wheezy episodes especially over the last 3 months. Older sister has asthma. No other medical history”*

We then remove a pre-defined set of stopwords:

*"Recurrent wheezy episodes especially last 3 months Older sister asthma medical history***”**

... and then stem the words, which leaves us with:

*"recurr wheez episod especi last 3 month older sister asthma medic histori”*

At this stage we then vectorise the notes, capturing the individual words ["recurr", "wheez", "episod", ...] and bi-grams ["recurr wheez", "wheez episod", ....] from the pre-processed text. Arranging the "tokens" from most frequent to least frequent, we then restrict the vocabulary to 500 words (this vocabulary size isn't a “hard-and-fast” rule - just what worked best in this particular case). The result is our pre-processed, vectorised notes.

So, we now have vectorised notes and have addressed the main problems that crop up when using word / token counts - fantastic! There are still, however, some biases that come with using this counting approach.

**Term Frequency (TF) – Inverse Document Frequency (IDF)**

By representing text numerically, we hope to quantify some element of its meaning. Using the "bag of words" approach, the typical assumption is that the tokens that appear *more frequently* are *more "important" or "prominent"*. This is a useful assumption but, when using raw counts, it comes with the following biases:

- **Sentence length**: In text analysis, it is common to compare texts of different lengths. If we assume that more important tokens appear more frequently, we risk biasing our measure of importance to longer texts. The longer a text is, the more opportunity there is for particular tokens to appear multiple times.

- **Context**: We often analyse text taken from a setting in which certain vocabulary is commonplace. If we take the ACE notes for example, words like "patient" and "history" are very common and may well appear in every note. As such, they don't serve to differentiate notes from one another,

despite appearing frequently. Simply assuming these words are the most important risks biasing analyses to words that are simply common in context, obscuring other words that actually serve to differentiate the notes.

These biases can be accounted for by using Term Frequency-Inverse Document frequencies, or TF-IDFs for each of the tokens in our vectorised notes:

- **Term Frequency**: Exactly as it sounds, the term frequency is simply the *frequency* of a token's appearance within a note. As it is a *proportion* of the total number of tokens in a note, it eliminates the bias in favour of tokens from longer notes.
- **Inverse Document Frequency**: The inverse document frequency is the logarithmically scaled inverse fraction of the documents that contain the word. Put simply, if we have 5 notes and the word "stethoscope" appears once or more in three of them, the inverse document frequency for "stethoscope" would be log (5/3). The more notes a word appears in, the smaller the inverse document frequency becomes.

Note: this is a somewhat simplified explanation of TF-IDF. In practice it usually involves a few extra tricks that help with numerical stability.

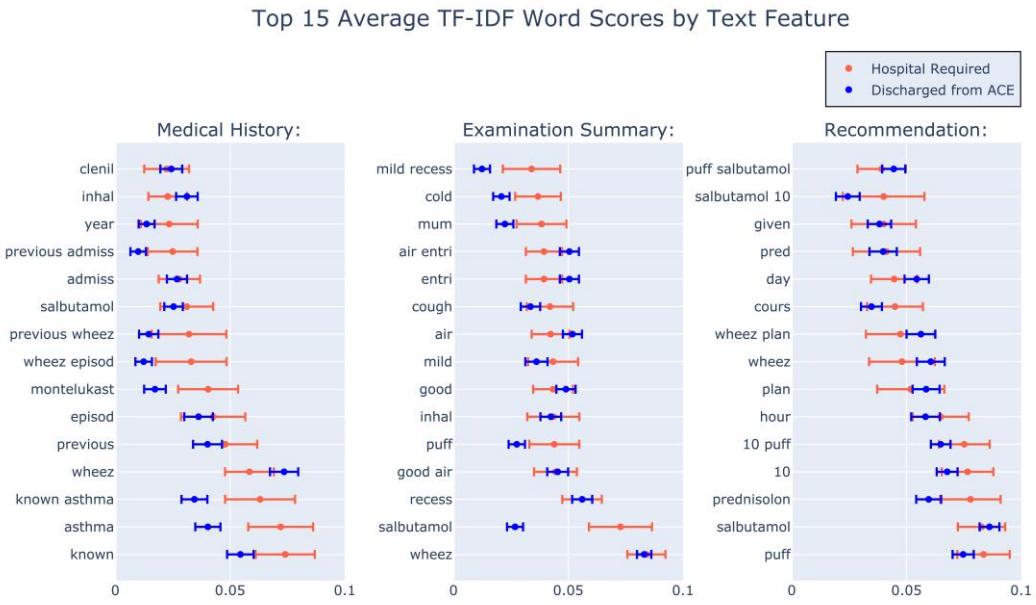
[*For a practical implementation of TF-IDF scoring, see the* TfidfTransformer *tool from the mighty Scikit-Learn[2]*]

With the text pre-processed, vectorised and TF-IDF values calculated, we finally have a structured representation of the ACE referral notes that can be analysed statistically.

**Basic TF-IDF Analysis:**

With TF-IDF representations of the text notes, we can now perform the typical statistical analyses that we would with structured data. An obvious place to start is to visualise the distributions of TF-IDF scores based on the outcomes of treatment - patients successfully discharged from ACE and patients that were later referred to hospital.

The following charts display the 15 words with the highest TF-IDF scores among patients that were referred to hospital, and the corresponding TF-IDF scores for both groups of patients:



Note: The error bars show standard error of the mean. The wider errors for patients referred to hospital signify the smaller proportion of these examples within the dataset - only 15% of examples were referred to hospital.

Analysing the differences in TF-IDF scores helps us to understand the differences and similarities between the notes of patients with different outcomes. For example, the medical histories of patients referred to hospital have more prominent mentions of words relating to "asthma". Similarly the examination summaries of the same patients mention the drug "salbutamol" more prominently. Many of the words also have overlapping scores that they are featured no more or less in the notes of patients that were hospitalised than those that were successfully discharged; for example, the most prevalent words in the recommendations of patients referred to hospital are very similar to those that were successfully treated.

This analysis is definitely informative, but it doesn't tell us which of the tokens are the *best predictors* of, or are *most associated with*, hospitalisation. There are 500 tokens in our TF-IDF data, which is a lot of complexity to sift through to identify words that genuinely correlate with different outcomes. Really, we need a method of linking the individual TF-IDF scores with treatment outcomes, in a way that identifies the most useful predictors:

**Lasso Logistic Regression Analysis**

There is no shortage of data analysis problems that involve identifying predictors from many hundreds (or even thousands) of potential candidates. Great examples are genomic analyses - typically involving profiles of thousands of genes - or financial market analyses - detailing the financials of many hundreds of companies. Lasso logistic regression (sometimes referred to as "ridge regression") is a common technique used to tackle these problems with broad and expansive input data.

Logistic regression is a great way of modelling a classification problem, when we need to understand how the observations in our data contribute to predictions of our outcome variable. Regression models assume that all the input variables, $x_1, x_2, x_3, \ldots, x_n,$ , make a linear contribution to the outcome variable, $y$, like so:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \ldots + \beta_n x_n$$

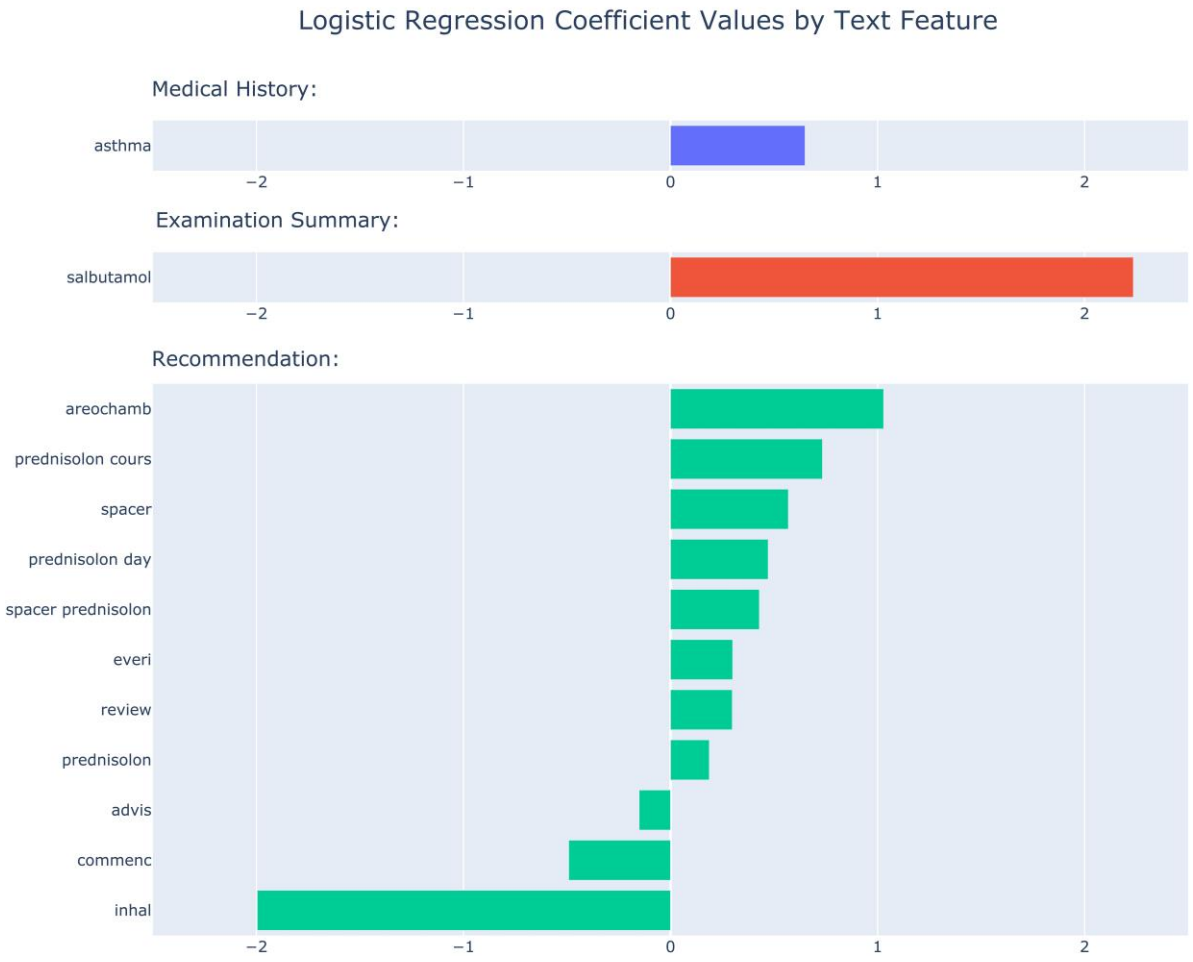
Note: Logistic regression uses some trickery to coerce $y$ to a probability between 0 and 1, but the linear relationship between the inputs and the outcomes holds.

By modelling the problem in this way, we need only look at the coefficients ($\beta_0, \beta_1, \ldots, \beta_n$) to infer the effect each of the observations has on the outcome. A positive coefficient indicates the observation increases the probability of a positive outcome, a negative coefficient indicates the opposite.

With our text data, a standard logistic regression model would produce 500 coefficients for each of the tokens. Trying to interpret 500 different effects would be difficult, indeed it is exactly what we are trying to avoid! That's where the "lasso" part of lasso logistic regression comes in. By adding an extra term in the cost function of the logistic regression algorithm (the bit that determines how well the model fits the data), lasso logistic regression adds a penalty based on the **size** of the coefficients. This penalty "encourages" the model to select as few coefficients as possible. Those coefficients that are selected indicate the most useful predictors among the available observations.

[*for a practical application of lasso/ridge logistic regression see* LogisticRegression *from the* linear_model *library of Scikit-Learn[2]*]

The below graph shows the coefficient values for each of the (nonzero) coefficient values for lasso logistic regression models trained using the TF-IDF scores from each of the three text features in the ACE dataset:




Viewing these results clearly demonstrates the advantages of using lasso logistic regression. Rather than seeing 500 separate tokens for each of the text features, the models have chosen only one token from the medical histories and examination summaries, and 11 for the recommendations.

The positive values of the coefficients for "asthma" and "salbutamol" indicate that their mention in a patient's medical history or examination summary respectively is predictive of hospitalisation. The coefficients from the recommendation model are a little more complex, and perhaps reflect the fact that we didn't see as clear a division between the TF-IDF scores in our initial analyses above.

When we evaluate the predictions of the above models, and compare them with the initial models trained using all the ACE referral data, both the medical history and examination summary models are significantly better predictors of hospitalisation (the recommendation model was no better than a coin flip). It is clear that the referral notes contain a better signal - better predictors of hospitalisation risk - than the structured data in the ACE referral form. These analyses suggest a link between *patient history of asthma* and/or *recent use of salbutamol* and an increased risk of hospitalisation during ACE treatment.

**Final Thoughts**

So there we have it! We have taken medical notes and used them to identify potential predictors of treatment outcomes in the ACE service. None of the techniques we've seen are complex, in theory or in practice - each of them can be applied using a few lines of code in your favoured programming language. The outcomes, however, can be extremely powerful. Indeed, this text analysis was used as a guide for the future of this project, and as the primary evidence to justify the addition of NHS health records to the ACE dataset.

It bears mentioning that, with the simplicity of these methods, come a number of limitations. For example, considering words in isolation ignores the context of words around them. Concepts such as negation are unaccounted for - the phrases "history of asthma" and "no history of asthma" have opposite meanings but are both counted as mentioning "asthma". Given that the purpose of this work was a simple analysis of the content of rough notes, this isn't too much of an issue. However, were we aiming to use the text directly for prediction, or to do more complex analysis of meaning or semantics, these issues would need to be accounted for.

If this brief intro has whetted your appetite for all things NLP, there are plenty of really exciting techniques that take these concepts further:

- Text clustering is a common unsupervised NLP technique that's used to sort texts into groups. It nicely leads on from the pre-processing techniques discussed in this blog.
- Word2Vec is a modelling technique that vectorises words, encoding their meaning within their linguistic context by accounting for the words that surround them.
- Recurrent Neural Networks (RNNs) are the precursor to the more advanced applications of machine learning referenced earlier. RNNs can "read" text without a specified input length and can "remember" the entire history of tokens when interpreting text.

**References**

[1] Steven Bird, Ewan Klein, and Edward Loper. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.", 2009.

Online Documentation: www.nltk.org/

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

Online Documentation: www.scikit-learn.org/