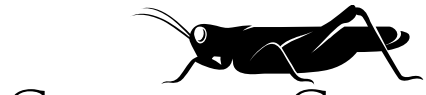

16 April 2019

Text Mining:
How Do Computers Understand Language?

Professor Richard Harvey FBCS

Text is everywhere. Not only is it the intelligent person's communication mechanism of choice, the widespread use of email, twitter and the web means that text is the King of the Internet Jungle — it is highly expressive, very compact and relatively portable. However, drawing automatic inferences from text has proven to be a remarkably tricky problem and unlike previous lectures in this series on audio and image recognition, I feel that there are many mysteries of text that have yet to be revealed to computer scientists. Let's look at some.

Nowadays every modern computer has a keyboard and a method for displaying text[1]. However, scouring back through the history of early computing for this lecture, I was surprised to find that the keyboard was a comparatively late addition to the operator's console of a computer. The earliest picture I can find of a computer qwerty keyboard is of one attached to an operator's console of a UNIVAC 1 computer dating from the 1950s[2]. And, notwithstanding arguments for alternative keyboards, the qwerty keyboard has remained a firm favourite ever since. Modern digital computers handle numbers, usually in chunks of 8-bits. With 8-bits one could represent $2^8 = 256$ possible characters so all that was needed was an agreed standard on what was meant by each number, then computers could handle text. There was relatively early agreement on how best to map American English into the 256 numbers. The map in common use was called ASCII (the American Standard Code for Information Interchange)[3] and "everyone" knew that, for example, decimal 32 (Hexadecimal 20) represented the space character. Since early commercial computers were made by Americans there was an interval of a few decades when learning to program also meant learning how to spell like an American! Nowadays computers are more inclusive, and we use *Unicode* which handles around 140,000 characters (or strokes) from all the major languages but Unicode is more complicated and not yet a completed project[4]. Character encoding is important, as it allows billions of people around the world to access the internet and information technology using symbols with which they are familiar but as far as computers are concerned, symbols are symbols — we can process them, count them, display them, without understanding them at all. Indeed, in the early days of Artificial Intelligence this very argument was used by Roger Searle to argue that computers were incapable of thought. In Searle's "Chinese room" are non-Chinese-speaking-people receiving pieces of paper containing English text and then following a set of complicated rules, they manipulate the English characters into Chinese and eject slips of paper that are translations of the English. Looking back on this argument, it seems naïve and childish, but it is actually a good model of how a computer handles text at a low level – by shuffling symbols around.

---

[1] Or what used to be quaintly called a VDU or Visual Display Unit.

[2] Early computers had no need of integrated keyboards as they were programmed via punched cards or tapes which themselves were created separately.

[3] It led to a joke that British computer scientists found hilarious at the time "Arthur ASCII – what a character!" Unfortunately, the people old enough to know who Arthur Askey was were too old to know about ASCII and those young enough to know what ASCII is were too young to know who Arthur Askey was. It is therefore a joke with the narrowest of demographics: British Computer Scientists in their 50s.

[4] Unicode is not without its controversies. There is considerable dispute as to whether all Japanese characters can be properly represented using strokes for Chinese, Japanese and Korean characters. According to Wikipedia, In 1993, the Japan Electronic Industries Development Association (JEIDA) published a pamphlet titled "未来の文字コード体系に私達は不安をもっています" (We are feeling anxious for the future character encoding system JPNO 20985671). I shall enter this title into any competition for the most abstruse document title in the world.

One of the simpler analyses of text is to count the frequency of occurrence of the symbols. If we are just counting single units then these are referred to as *unigrams*. If we are counting pairs of characters then these are *bigrams*, threes *trigrams* and so on. Counting *n-grams*, can be surprisingly effective. It is well known from early cryptoanalysis that unigram frequency tables can be highly distinctive of language and the domain[5]. Likewise, higher-order n-grams. Despite the antiquity of this idea, it is still in common use for language identification, and for the identification of authors and topics. Indeed, it is rather gratifying that a rather ancient method [1] still has something to offer modern times. In [1] it is noted that, if we order *N*-grams by frequency then the most common are the unigrams (which identify the language) then come the common prefixes and suffixes (which are more indicative of topic, domain of knowledge and authorship).

When considering *N*-grams of characters it is natural to consider extending the idea to consider *N*-grams of words. Of course, this is more challenging. If there are *M* characters in a language then there are *M* unigrams, M^2 bigrams, M^3 trigrams and M^N *N*-grams. Even for modest *M* and *N* this can be a very large number indeed. For 5-grams in the Italian language then we need to store 21^5 or 4 Million numbers. Not impossible, but to accurately estimate the frequency of 4 Million numbers needs around 40 million characters (1600 pages of typed text). This problem is called the "curse of dimensionality" — as *N* increases linearly, the amount of storage and computation increases exponentially. Several traditional cures have been proposed to this problem, but as we shall see, none of them very satisfactory.

To process more than a few characters of text, people have proposed using word *N*-grams. This is fine for small vocabularies (a few thousand words) but can soon become intractable. Some people advocate removing small words that stop good text retrieval working. These, so called, "stop words" are found by a variety of different methods and lists exist ([2] for example). However, there is no general agreement what these words are and it feels rather unsatisfactory to hand-remove something which is patently learnable. That said, "tokenisation" which is the removal of punctuation and the merging of synonyms is usual. Another approach is "stemming". The classic algorithm, devised by Martin Porter [3], is known as the Porter Stemming algorithm and is a staple of university Information Retrieval modules. The idea is than words such as "run", "runner" and "running" all relate the same concept of high-speed human power bipedal motion so let us just return the stem of the word "run" and save a lot of time and effort later. The failures of Porter stemming are well known and Wikipedia lists "universal", "university", and "universe" as being incorrectly stemmed to "universe." Nowadays most people avoid stop-words and stemming although they are still commonplace in old systems[6].

A more modern viewpoint is to view text processing as two stages. In the first, the text is converted into a vector of numbers. In the second stage we attempt to draw inferences from that vector of numbers. The second stage is identical to the problem described in previous lectures which is known as pattern recognition (AI in the parlance of the age). And the current trendy way of doing pattern recognition is to use deep neural networks (see previous lecture). So, the nub of modern text processing is to convert text into a "feature" or a list of numbers, that is representative enough of the text such that the problem can be solved. Since problems vary, it follows that features vary, and the business of designing features (feature engineering) is something of an art.

We have already met one feature — the list of unigrams. In English there are 26 letter frequencies, so unigrams provide a 26-dimensional feature detector. And that feature detector is good enough to solve really easy problems, like differentiating Shakespeare plays from Sherlock Holmes novels. The bigrams list is rather longer (676 elements) but manageable and good enough to differentiate Shakespeare comedies from tragedies [4]. As we move to N-grams the list becomes too long, so in [1] for example, the *N*-grams are sorted by frequency and only the top 300 are considered — this is effective for language identification.

[5] The idea was that, if a cryptographic system merely replaces a letter with another letter, then the letter frequencies will be unaffected, and it is simple to crack the code. Needless to say, no modern cryptographic system, would do this!

[6] Computer scientists rather charmingly refer to old systems as "legacy" systems. It must be one of the few recorded cases where a legacy has the potential to reduce your fortune rather than increase it.

Another buzzword is the "bag of words" model. A unigram word model just counts how many occurrences of words happen[7] but it is debatable how much of the semantics this captures because in the bag of words model the phrase "I killed Robin" is treated identically to "Robin killed I". To overcome this issue there are a variety of approaches which are variously known as "text embedding" or "text vector spaces" of which the most venerable is a technique called Latent Semantic Analysis or LSA.

LSA derives from early work by the American psychologist George Kelly. Kelly was interested in untangling the patient's view of the world from the lens of the analyst — at the time it was commonplace for a psychoanalyst to declare that they were a Jungian and therefore explain the patient's condition in Jungian terms. Kelly thought it was strange that a patients frame of mind might alter with the type of psychotherapist who was asking the questions so, alongside his work on personal construct theory he devised a technique called "repertory grids" which, using a mathematical technique called Singular Value Decomposition was able to factorise the matrix of question responses into separate components. In our field the key matrix is the term-document matrix. If we imagine three documents:

D1 = "I killed Robin"
D2 = "Robin killed I"
D3 = "With my bow and arrow"

Then, if we ignore the stop words of "I", "my," "with" and "and" then we can construe these three documents as a matrix

	arrow	bow	killed	robin
D1	0	0	1	1
D2	0	0	1	1
D3	1	1	0	0

This is a matrix that contains a count of the terms in each document hence *term-document matrix*. Usually the raw count is replaced with a count that is normalised by the number of times that term appears in all documents. This is the TF-IDF (term frequency – inverse document frequency) matrix. The inverse document frequency is related to the amount of information (see Lecture 1) that a term provides. The idea is to approximate the matrix above, let's call it *X*, with a simpler matrix X' that itself is a product of three much simpler matrices $X' = U \Sigma V^T$. It turns out that U encodes the relations between the documents and V the relations between the terms. In our example "killed" and "robin" always appear together so they are the same concept (true enough). And that concept, the killing of Robin, does not occur in Document D3. The space of terms in this example is 2. Concept 1 is the killing of Robin and Concept 2 is the bow and arrow. In LSA we would say Concept 1 = 0.5 "Robin" + 0.5 "killing" and Concept 2 = 0.5 "arrow" + 0.5 "bow". If a new document comes along, we project it into this concept space and our machine learning works using the size of the projection.

LSA is a mathematically attractive idea and therefore popular fodder in undergraduate courses on information retrieval but I cannot honestly claim to have used it for very much. It still suffers from the bag of words restriction and, despite being presented as a panacea for Search Engine Optimisation there is precious little evidence it helps[8]. What is needed was a vector space that took account of linguistic context (words before and after a word). This is exactly the idea behind word2vec and its variants [5].

Word2vec uses deep learning as described in a previous lecture. It uses an architecture with a bottleneck — a window of five words is used as the input and the task is to recreate the input at the output. Six billion words later, the network is trained. Precisely what it is modelling is still an ongoing puzzle (as with many deep neural networks it is often not crystal clear on, they form their inferences) but the model is very impressive, and it maps words to multidimensional vectors. The vectors usually have around 1000 dimensions which is one of the reasons why they are hard to visualise. However, one test of the system is to check that vector differences are meaningful.

[7] As Eric Morecombe protested – "I am playing all the right notes…not necessarily in the right order."
[8] The idea was that augmenting one's webpage with synonyms would help the search engine better map your document into its concept space. Since search engines do not use LSA the reasoning was fanciful.

So if we input the word "France" and get the vector output **v**("France") (here I am using a bold **v** to denote a list of 1000 numbers) then as the capital of France is Paris it would be remarkable if **v**("France") – **v**("Paris") + **v**("Italy") = **v**("Rome"). But this is exactly what happens. Furthermore, words that mean similar things all map to similar vectors. Of course, there are now many incremental improvements to Word2vec but the basic idea persists and is the basis of many impressive text processing systems.

Such systems can also be thought of as machine translation — one set of text is "translated" to another set via this vector space (also known as an "embedding"). Let's imagine we had a very large set of inputs, each input describing the review of a restaurant, and outputs which were the review. The input might be:

5 Public House Las Vegas NV Gastropubs Restaurants
Which is the short-form of Yelp review — a 5* review (a review that has previously been awarded fives stars by readers of Yelp — of a Pub is Las Vegas, Nevada. And the associated output is a review
Excellent food and service. Pricey, but well worth it . I would recommend the bone marrow and sampler platter for appetizers.

So, if we learn the relationship between the eatery and the review then we can generate 5* or 1* reviews at will. Currently this is done by hand – one pays humans to write fake reviews in a process known as "Crowd turfing." But now crowd-turfing can be done automatically [6] and, in tests, humans cannot tell the automatic fake reviews from the human ones (fake or otherwise). One challenge of the system is that the first attempt at computer generated reviews generated texts that were too grammatically correct (real people make frequent errors) so these reviews must be corrupted to look human! Incidentally, machine learning can easily tell computer generated reviews apart from human ones. Furthermore, there are systems that claim to be able to spot fake reviews so, for the time-being, your review of La Gavaroche is safe.

More impressively, a group of researchers collated 8300 images and poems inspired by those images and a collection of 93,000 poems. Again, the task is "translate" the image into a poem. Their methods are quite intricate, and the results do not yet fool the experts but they are good enough to fool some of the people some of the time [7].

That said, machine translation is not perfect — in our earlier lectures we ran an online translation system provided as a free add-on for Microsoft PowerPoint. Although it was considerably better than my efforts at Chinese, it was still quite errorful. One of the standard challenges of machine translation is gender[9]. Human translators resolve these imprecision by parsing sentences over long distances (many words apart). Current embeddings cover a small window so are well suited for poetry but I fear that legal documents, where the definitions are pages away from their use might be a challenge.

The current apex of achievement in text processing is IBM Watson's triumphant win at "Jeopardy". Jeopardy is an interesting quiz since the questions are often highly elliptical. Thus, under a category such as "Presidential poetry". Might be a clue of "Barrack's pack animals". To which the contestant might be expected to answer, "What is Obama's Llamas?" The rules state that the contestant should formulate a question to which the original clue is an answer. It is highly impressive that Watson was able to beat the two best champions of this game and without access to the internet. The TV programme made compelling viewing not only because of Watson's impressive feats but also because of its infrequent errors — as with many neural systems the errors were often outlandish or childish. A nice feature of the Watson system was a probabilistic assessment of its own confidence — if it was confident it buzzed in early.

In summary computer text processing has had a brief period in the doldrums where many of the classic algorithms were over ten years old (this is very unusual in computer science). That time is now over and the new tools of

[9] Professor Andy Stanford-Clark alerted to me this example. Type "She is a Doctor. He is Nurse" into Google Translate. Translate to Turkish and then back to English. You should get "He is Doctor. She is Nurse." Clearly in gendered languages translations can be quite subtle.

vector embedding, and neural translators are opening a whole new world of text processing some of which overlaps into the field of creativity which is the topic of the next lecture.

1. Cavnar, William B. and John M. Trenkle. "N-Gram-Based Text Categorization". Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval (1994)
2. https://code.google.com/archive/p/stop-words/
3. M.F. Porter, 1980, An algorithm for suffix stripping, Program, 14(3) pp 130−137.
4. "Unigrams, bigrams and LSA. Corpus linguistic explorations of genres in Shakespeare's plays", January 2008, in: "New Directions in Literary Studies", Chapter: 5, Publisher: Newcastle, England: Cambridge Scholars Publishing, Editors: W.Van Peer & J.Auracher, pp.108-129
5. Mikolov, Tomas; et al. (2013). "Efficient Estimation of Word Representations in Vector Space". arXiv:1301.3781
6. Juuti M., Sun B., Mori T., Asokan N. (2018) Stay On-Topic: Generating Context-Specific Fake Restaurant Reviews. In: Lopez J., Zhou J., Soriano M. (eds) Computer Security. ESORICS 2018. Lecture Notes in Computer Science, vol 11098. Springer.
7. "Beyond Narrative Description: Generating Poetry from Images by Multi-Adversarial Training", Bei Liu, Jianlong Fu, Makoto P. Kato, Masatoshi Yoshikawa, ACM Multimedia 2018.