Three data analytics party tricks

Matt Hall, Agile

matt@agilescientific.com

Everything in machine learning today is at the bleeding edge. Its practitioners must, accordingly, learn constantly. Small children and scientists play to learn, and it's especially fun to play with machine learning. I find it fun because the outcomes are unexpectedly good: prediction quality seems disproportionate to the cost of the analysis. Manual interpretation is predictive too, but takes hours of effort, so it's hard to iterate, and impossible to scale to a million times more data. The statistical and neural models of machine learning still require hours of labor, don't let anyone tell you otherwise, but then the game changes because iteration and scale is what computers do best.

In playing with machine learning, maybe we can even figure out some new ways to solve problems in geophysics.

Natural language processing

Our playground will be the world of text mining and natural language processing, or NLP (not to be confused with neuro-linguistic programming; a natural language is a human language, like English or Swahili). Text mining is just the general process of computationally extracting useful information from text. Meanwhile, NLP lives at the intersection of computer science and linguistics, and today is a very hot research topic, thanks largely to social media and the need to parse large amounts of text to extract marketing intelligence, combat spam, or translate it to other languages.

Let's look at machine translation as a proxy for recent trends across the field of machine learning. Translation has been of interest to artificial intelligence researchers since the famous Georgetown–IBM experiment of 1954. Until recently, the prevailing methods revolved around manually constructing models of language using expert knowledge. For example, labeling text with parts of speech and applying complex grammatical rules *in both languages*. Recently, machine learning, and especially 'deep' neural networks (those containing two or more *hidden* layers of neurons between the input and output layers), have successfully applied a fundamentally new approach. Instead of starting with an *a priori* model of either the input or the output language, these networks learn the rules by inference 'on the job', rather like an infant does. The catch is that you have to give them a large — perhaps huge — amount of text (the *corpus*), and therefore train on a large — perhaps colossal — computer. It's not a coincidence that the researchers at the center of this research are at Google and Facebook.

Research scientists in other fields have noticed all this, and not just for text mining. Bioinformatics has benefited from advances in NLP, since protein-coding sequences in DNA and RNA have much in common with words and sentences in a natural language. This has proven to be a more than just an analogy, with significant insights being had into the relationships between protein coding and function. I think it's a short step from there to proposing treating stratigraphy as a natural language, expressing the earth's geological history in a language we don't fully understand. Is geological interpretation a machine translation problem?

Bibliographic database

All the research in this article centers on the bibliographic metadata from *Geophysics*, the SEG's flagship journal. The data was gathered, responsibly and with permission, from seg.org. It builds on the workflow I described in Hall (2010), and on data I acquired for use at the 3rd Geophysics Hackathon in New Orleans, October 2015. The dataset is SEG's property and I cannot share it completely, but I can share a subset of it and provide you with enough data to follow along with my methods, should you wish to.

In brief, the full dataset contains 9607 papers from every issue of *Geophysics* since volume 1. I have attempted to remove non-research papers, such as reviews and discussions, but data cleaning is rarely a precise affair. The data includes abstracts, titles, authors, author institutions, keywords, DOIs, and the DOIs of citing papers. The subset contains partial data from 500 randomly selected papers.

Co-authorship graph

To warm up, we'll start with something familiar. I wrote about my preoccupation with social network analysis in Hall (2010), in which I showed the collaboration network for *Geophysics* volumes 65 to 74. I have since re-run the analysis using the entire history of the journal to date. On some level, the *graph* (the technical term for a network) becomes unwieldy at this scale, because it represents at least a couple of career lengths but the time dimension is flattened, so to speak, making one large *graph* with 9642 *nodes* (authors) and 20 237 *edges* (collaborations, represented by co-authorship). Still, since collaborations often do connect across generations, perhaps it's interesting to revisit this super-network as-is.

After gathering the author relationships into a simple data structure in Python, I expressed the author relationships in Graph Modelling Language (GML), a widely-used plain text format for representing nodes and edges. Most graph analysis software can import GML files; I use the NetworkX Python library (Hagberg et al., 2008). The files, and the script I used to generate them, are at github.com/seg/2017-tle-hall. You can see a visualization of the network in Figure 1.

A key concept in network analysis is *centrality* — the property of being somehow ‘influential’, through the connections in the network. For example, Google uses a type of recursive centrality called PageRank, in which connections to nodes with lots of connections score high, to order pages in search results. There are several other ways to measure centrality, most of which are directly accessible in NetworkX; two of the most intuitive are:

- *Degree centrality* ranks nodes based on *degree* (the number of edges incident on them). The authors with highest degree centrality in the *Geophysics* network are G McMechan, A Green, M Toksoz, A Revil, K Marfurt, and M Landro. These authors have collaborated with many people, e.g. George McMechan has 109 collaborators.
- *Betweenness centrality* ranks nodes based on the number of node-to-node paths that pass through them. The authors with high betweenness are G McMechan, J Harris, B Gurevich, S Singh, S Fomel, and M Toksoz. These authors are ‘connectors’ between other geophysicists.

A popular graph theory party trick is to trace ‘degrees of separation’ between individuals. Movie buffs play Six Degrees of Kevin Bacon, trying to link any actor to him via co-appearances in movies. All mathematicians know their Erdős number, linking them to the prolific Hungarian mathematician Paul Erdős. An Erdős number of 2 means you have co-authored a paper with someone who has collaborated with Erdős himself. According to the American Mathematical Society MathSciNet, Sergey Fomel’s Erdős number is 4. Of course, the property is commutative, so Erdős’s Fomel number is 4. Here are some other geophysicist’s Fomel numbers (at least as far as co-authoring *Geophysics* papers goes):

```
4 Brian Russell
3 Chris Liner
3 Larry Lines
3 Sven Treitel
2 Jose Carcione
2 Vladimir Grechka
2 Jerry Harris
1 Jon Claerbout
```

You can look up other connections in this subgraph at http://collab.geosci.ai.

Some individuals aren’t connected at all — the 774 authors who have only ever published single-author papers are *isolates*. Furthermore, as is typical with large graphs, there are many mutually disconnected *subgraphs*, or *components*. One of these components, again this is typical, is much larger than the others, containing 6648 of the 9642 nodes; the next largest component has only 37. The *diameter* of the largest component is 23, so that’s the longest path between two authors in that subgraph.

The collaboration network bears more extensive study. There are algorithms to help discover cliques and communities in social networks, for example. The collaborations between institutions are also interesting, as are the research trends over time. Study of these networks could change the way we find paper reviewers, or organize workshops, and would make a project for a future hackathon.

A recurrent neural network

I already mentioned that machine learning, and especially deep learning, have completely changed the field of NLP. Among the various deep learning methods, two stand out from a geoscience point of view: convolutional and recurrent neural networks. *Convolutional* neural networks (CNNs or ConvNets) apply spatial convolution and averaging (or *pooling*) steps to learn about *regions*, for example of images. *Recurrent* neural networks (RNNs) apply a particular kind of feedback that favors learning from *sequences* of things. One type of RNN, the long short-term memory (LSTM) network, seems especially promising. To start to learn about RNNs, I trained a character-based LSTM-RNN on the titles and the author names from the *Geophysics* dataset. For each of the two datasets I did the following:

1. Gather the items into a text file, with some obvious separator between items.
2. Train the LSTM-RNN by feeding its text file to the training script, and setting some parameters. I'm using Karpathy's (2015) implementation in Torch, a numerical extension to the Lua programming language that is very popular with deep learning researchers. I used a network with 200 neurons in each of 3 hidden layers. The sequence length (if you like, the memory of the network) was 200 characters. Large neural networks take a long time to train, and most researchers are using graphics processing units (GPUs); this network takes about 20 minutes to train on 2048 GPU cores, or 8 hours on my CPU.
3. Once trained, we can get predictions from the network. Given a sequence of characters, the network provides the a probability for encountering character next. For a party trick, we can start with a random character, then keep sampling from the probability of next characters and feeding the sequence so far back into the network, we can generate random text. Here are some examples:

- S. E. Pollack & C. T. Larrentis, Vertical Pressure From Seismic Reflection Surveys Profiled By Matchade Electrical Fields
- Pewart W. Stewari, Grave Basalted Normal Moveout Estimation In The Tort Vandous Flow Profile
- Sudhar Pajer, Jie Hou & Jon Dirmen, Marine System Surveys Through A Vertically Polarizad Reflector
- D. J. Laniert, Like-Wave Beam

- Djice R. Brywolt, Multidimensional Environments And Curved Induced-Polarization From Macahanaos Theorytack
- Giy Claponio & Rune Mittet, Three-Dimensional Bodies Of Industrial Computation (definitely my favorite)

You can see a web-based implementation of the random text generation at http://rando.geosci.ai. It's just a bit of fun, making random titles and authors, but it's interesting to see how plausible the titles are. This means the model has a decent idea of how to put together a sequence of letters to make a reasonable geophysical idea. To put it another way: given a word, the model can make good guesses about the next word, because it has 'seen' many, many sequences of words before, and its neurons 'understand' the structure of sentences at several scales. To gain more appreciation for the sophistication of these models, I recommend reading Karpathy (2015), especially the visualizations of random code generation.

To connect these games back to geoscience, remember that our 'sequence' does not have to be characters of text. We can train the network on sequences of anything. This leads to the idea that if we train a network on a sequence of rocks (in a wellbore, say) perhaps it will make good guesses about the rocks we haven't seen yet. Perhaps it will even 'understand' something about the sequence that surprises and informs us. To the best of my knowledge, this is an open research question that few geoscientists are working on.

Content-based recommendations

Recommendation systems are central to the recent interest in data science. Several high-profile machine learning contests have focused on the problem, because small improvements in recommendation quality are potentially lucrative for online firms like Amazon and Netflix. Many of these such systems are driven by user interaction: what you have previously viewed or purchased, ratings you have provided, and what people with similar profiles like.

There are some obvious applications of recommendation systems in scientific research, the foremost being as an aid to discovery in the literature. Keyword search can result in very large numbers of hits (and, worse, misses!) since they by definition reduce each document to a small handful of words. We don't have a rich history of user interactions to go on, but we can build a model of how similar documents are, then simply ask for one piece of evidence ("What's your favorite paper?"), and recommend similar documents.

For my next party trick, I'll train a simple system on our dataset of 4833 abstracts, along with titles and DOIs, from *Geophysics*. We're going to infer meaning directly from the abstracts, in a process called latent semantic analysis (LSA). The training steps are illustrated in Figure 2a, and described stepwise here:

1. *Preprocess the data.* There are three steps, tokenizing, stop-word removal, and stemming, all provided by the NLTK package (Natural Language Toolkit; Bird et al. 2009). The document text is turned into a list of individual words, or *tokenized*, using a regular expression (a common text processing tool). Common words with little semantic meaning ('a', 'an', the', etc.) are dropped. Finally, stemming truncates words to their *stems* in order to reduce the size of the vocabulary. For example, 'migrated' and 'migration' are both transformed into 'migrat'. After this step, each document is a string of word fragments with spaces but no punctuation.
2. *Vectorize the data* with the TF–IDF vectorizer in the scikit-learn package (Pedregosa 2011). This is the most complicated step, although it's all contained in a single line of code, so I will break it into pieces:

a. A common approach in NLP is to treat the text as a so-called *bag of words.* But in order to try to recognize that not only words but phrases carry meaning in English, I instead process the text into word *n*-grams: groups of one to (say) three words. So the phrase "reverse time migration" becomes (with stemming) not just "revers", "time" and "migrat" (a very small bag of words), but three 1-grams, two 2-grams, and one 3-gram: "revers", "time", "migrat", "revers time", "time migrat", and "revers time migrat". These *n*-grams become the *terms* in the *vocabulary.* In the dataset of abstracts, the vocabulary contains 755 257 terms.

b. Now we construct a 4833 × 755 257 *term frequency* matrix in which each row is a document, and each column represents one of the terms we discovered. For now, the values in the matrix are the frequencies of each term in the documents. As you might expect, this matrix is *sparse* — the row representing document number 3100, "Reservoir Characterization Using Seismic Waveform And Feedforward Neural Networks", contains only 224 non-zero terms, and this is fairly typical.

c. Finally we apply another transform. To get the so-called *inverse document frequency* from the term frequencies, we scale them by the inverse global frequencies of each term in the dataset. In other words, we give more weight to relatively rare phrases — proposing that they are more distinctive — and less weight to common ones. The result of all this is the TF–IDF matrix.

NB list numbering will continue from before

1. *Dimensionality reduction.* Working with features vectors in 755 257 dimensions is a bit unwieldy, and seems like overkill. We'd like to reduce them to, say, 100 or so. There are lots of ways to do this, but the most common approach for our purposes is truncated singular value decomposition (SVD), applied with a normalization step by scikit-learn. We can visualize the result, albeit in two dimensions, in Figure 2b. After this step, which is a defining feature of LSA, each document in the dataset of N items is represented by M features. These features are artificial, in the sense that

we don't know what they represent in natural language. We store this $N \times M$ matrix because it represents the 100-dimensional vector space of our dataset.

2. *Nearest neighbor tree.* The idea is that if two documents are close to each other in this 100-dimensional space, then they have similar content. We can decide what 'close' means by selecting among various distance metrics, but the most intuitive is just to use the Euclidean distance. Given a document in the space, we'd like to be able to find its neighbors as quickly as possible. An efficient data structure for this kind of query is a k-dimensional tree, or k-d tree, which essentially acts like a look-up table of every point's neighbors and their distances. This step creates and stores the k-d tree.

To get a recommendation, we must nominate or 'like' one or more of the documents in the dataset, and pass their indices to the recommender function. (We could also form a mini-document out of some arbitrary search terms, and then vectorize it in the same way as we did the documents, using their vocabulary, but we'll stick to recommendations based on existing documents.) The recommendation part of the process performs the following steps:

1. 'Like' one or more documents. Liking a large number of dissimilar documents might lead to some strange (or maybe interesting) results.
2. Form a query vector from some central tendency measure of the 'liked' documents' vectors. The most common measure and perhaps intuitive is the centroid (mean). The only potential drawback is that the centroid of two or more documents is unlikely to be a document itself, so the neighborhood search will be around a point that does not represent anything in the database. If this seems like a problem, then we can use the geometric median (an n-dimensional extension of the median), which gives us the most central of the nominated documents.
3. Given the query vector, get the required number of nearest neighbors from the k-d tree. The tree provides both the indices of the documents, and their distances from the query vector. The nominated documents may or may not be in the resulting documents, depending on how similar they were to each other, and on the centrality measure used. In particular, if the 'likes' were very dissimilar, and the centroid was used to calculate the query vector, then the nominated documents probably won't be in the results.
4. Given the indices of the recommended documents, we can recover whatever information we want from the database. Additionally, we can use the distance measures as an indication of how likely the recommendations are to be interesting.

You can see a small implementation of this recommender system, based on a subset of the *Geophysics* dataset, at http://github.com/seg/2017-tle-hall, and a web-based implementation of it at http://georx.geosci.ai.

There are lots of ways to improve on what I've described. The dataset could be much larger, and include other publications and conference proceedings. A visualization of the results could be useful and interesting. One can imagine ways to discover 'chains' of recommendations that might even result in novel ideas. Exploting the author collaboration network could also help discover relevant content. All this would make an excellent theme for a future hackathon.

It's not too much of a stretch to come around to scientific applications. We can imagine our "documents" as well logs, or seismic sections. Once vectorized and dimension-reduced, we could query the similarity tree for "things that are like this reservoir interval" — not unlike a waveform classifier, but capable of searching an entire database very quickly.

Summary

Machine learning is a vast field of powerful methods for inching closer to artificial intelligence. It will change how we do geoscience, once we figure out how to use it. Playing with small problems in familiar data domains is one way to figure things out, and with the silly party tricks in this article I have tried to show you some ways to play, and give you the means to do it. I hope you will explore the software and data at http://github.com/seg/2017-tle-hall.

References

Bird, S, E Klein, and E Loper (2009). Natural Language Processing with Python. O'Reilly Media Inc. http://nltk.org/book.

Hagberg, A, D Schult, and P Swart (2008). Exploring network structure, dynamics, and function using NetworkX. In: Proceedings of the 7th Python in Science Conference (SciPy 2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15.

Hall, M (2010). Collaboration networks. *The Leading Edge*, 29 (11), November 2010. p 1354–1361.

Hutchins, J (2004). The first public demonstration of machine translation: the Georgetown-IBM system, 7th January 1954. AMTA Conference. Online version at http://www.hutchinsweb.me.uk/GU-IBM-2005.pdf.

Karpathy, A (2015). The unreasonable effectiveness of recurrent neural networks. Blog post on http://karpathy.github.io/, May 2015.

Pedregosa, F, et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **12**, pp. 2825-2830.

van der Maaten, L, and G Hinton (2008). Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research* **9**, pp. 2579-2605.

Zeng, Z, et al. (2015). Survey of natural language processing techniques in bioinformatics. *Comput. Math. Methods Med.* doi: 10.1155/2015/674296