First steps in data science: author-aware sentiment analysis

People often ask me what's the best way of becoming a data scientist (http://yanirseroussi.com/2014/10/23/what-is-data-science/). The way I got there was by first becoming a software engineer and then doing a PhD in what was essentially data science (before it became such a popular term). This post describes my first steps in the field with the goal of helping others who are interested in making the transition from pure software engineering to data science.

While my first steps were in a PhD program (http://yanirseroussi.com/phd-work/), I don't think that going through the formal PhD process is necessary if you wish to become a data scientist. Self-motivated individuals can get very far by making use of the abundance of learning resources available online. In fact, one can make progress much faster than in a PhD, because PhD programs have many overheads.

This post is organised as a list of steps. Despite the sequential numbering, many steps can be done in parallel. These steps roughly recount the work I've done to publish my first paper, which was co-authored by Ingrid Zukerman (http://users.monash.edu/~ingrid/) and Fabian Bohnert (https://sites.google.com/a/bohnert.eu/fabian-bohnert/). Most of the technical details are intentionally omitted. Readers who are interested in learning more are invited to read the original paper (https://dl.dropboxusercontent.com/u/25632965/SeroussiZukermanBohnert2010b.pdf) or chapter 6 in my thesis (http://arrow.monash.edu.au/vital/access/services/Download/monash:89860/THESIS01), which includes more thorough experiments and explanations.

Step one: Find a problem to work on

Even if you know nothing about the machine learning and statistics side of data science, it's important to find a problem to work on. Ideally it'd be something you find personally interesting, as this helps with motivation. You could use a predefined problem such as a Kaggle competition (http://www.kaggle.com/competitions) or one of the UCI datasets (http://archive.ics.uci.edu/ml/datasets.html). Alternatively, you could collect the data yourself to make things a bit more challenging.

In my case, I was interested in natural language processing and user modelling (http://www.csse.monash.edu.au/research/umnl/). My supervisor was given a grant to work on sentiment analysis (https://en.wikipedia.org/wiki/Sentiment_analysis) of opinion polls, which was my first direction of research. This quickly changed to focus on the connection between authors and the way they express their sentiments, with the application of harnessing this connection to improve the accuracy of sentiment analysis algorithms. For the purpose of this research, I collected a dataset of texts by the most prolific IMDb (http://www.imdb.com/) users. The problem was to infer the ratings these users assigned to their own reviews, with the hypothesis that methods that take author identity into account would outperform methods that ignore authorship information.

Step two: Close your knowledge gaps

Whatever problem you choose, you will have some knowledge gaps that require filling. Wikipedia, textbooks, and online courses will be your best guide for foundational areas like machine learning and statistics. Reading academic papers is often required to get a better understanding of recent work on the specific problem you're trying to solve.

Doing a PhD afforded me the luxury of spending about a month just reading papers. Most of the ~200 papers I read were on sentiment analysis, which gave me a good overview of what's been done in the field. However, the best thing I've done was to stop reading and move on to working on the problem. This is also the best advice I can give: there's no better way to learn than getting your hands dirty working on a problem.

Step three: Get your hands dirty

With a well-defined problem and the knowledge gaps more-or-less closed, it is time to come up with a plan and implement it. Due to my background in software engineering and some exposure to early collaborative filtering approaches to recommender systems (https://en.wikipedia.org/wiki/Collaborative_filtering#Memory-based), my plan was very much a part of what Leo Breiman called the algorithmic modelling culture (http://projecteuclid.org/euclid.ss/1009213726). That is, I was more focused on developing algorithms that work than on modelling the process that generated the data. This approach is arguably more in line with the mindset that software engineers tend to have than with the approach of mathematicians and statisticians.

The plan was quite simple:

- Reproduce results that showed that rating inference models trained on enough texts by the *target author* (i.e., the author who wrote the text whose rating we want to predict) outperform models trained on texts by multiple authors

- Use an approach inspired by collaborative filtering to combine multiple single-author models to infer ratings for texts by the target author, where those models are weighted by similarity to the target author
- Experiment with multiple similarity measurements under various constraints on the number of texts available by the training and target authors
- Iterate on these ideas until the results are publishable

The rationale behind this plan was that while different people express their sentiments differently, similar people would express their sentiments similarly (e.g., use of understatements varies by culture). The key motivation was Pang and Lee's finding (http://arxiv.org/pdf/cs/0506075.pdf) that a model trained on a single author is best if we have enough texts by this author.

The way I implemented the plan was vastly different from how I'd do it today. This was 2009, and using Java with the Weka package (http://www.cs.waikato.ac.nz/ml/weka/) for the core modelling seemed like a huge improvement over the C/C++ I was used to. I relied heavily on the university grid to run experiments and wrote a bunch of code to handle experimental logic, including some Perl scripts for post-processing. It ended up being pretty messy, but it worked and I got publishable results. If I were to do the same work today, I'd use Python for everything. IPython Notebook (http://ipython.org/notebook.html) is a great way of keeping track of experimental work, and Python packages like pandas, scikit-learn, gensim, TextBlob, etc. are mature and easy to use for data science applications.

Step four: Publish your results

Having a deadline for publishing results can be stressful, but it has two positive outcomes. First, making your work public allows you to obtain valuable feedback. Second, hard deadlines are great in making you work towards a tangible goal. You can always keep iterating to get infinitesimal improvements, but publication deadlines force you to decide that you've done enough.

In my case, the deadline for the UMAP 2010 conference (http://www.um.org/) and the promise of a free trip to Hawaii served as excellent motivators. But even if you don't have the time or energy to get an academic paper published, you should set yourself a deadline to publish something on a blog or a forum, or even as a report to a mentor who can assess your work. Receiving continuous feedback is a key factor in improvement, so release early and release often (https://en.wikipedia.org/wiki/Release_early%2C_release_often).

Step five: Improve results or move on

Congratulations! You have published the results of your study. What now? You can either keep working on the same problem – try more approaches, add more data, change the constraints, etc. Or you can move on to work on other problems that interest you.

In my case, I had to go back to iterate on the results of the first paper because of things I learned later. I ended up rerunning all the experiments to make things fit together into a more-or-less coherent story for the thesis (writing a thesis is one of the main overheads that comes with doing a PhD). If I had a choice, I wouldn't have done that. I would instead have pursued more sensible enhancements to the work presented in the paper, such as using the author as a feature, employing more robust ensemble methods, and testing different base methods than support vector machines. Nonetheless, I still think that the core idea – that the identity of authors should be taken into account in sentiment analysis – is still relevant and viable today. But I've taken my own advice and moved on.