UNIVERSITY OF LEEDS

Cheat, NLTK, SpaCy: Text Analytics in Python

Professor Eric Atwell: Hello, this is Eric Atwell, and, in this lecture, I want to talk about CHEAT and NLTK and other platforms for text analytics in Python. This is going to be an introduction to some Python tools, just an introduction. I'm not going into much detail. This too, specifically for computational linguistics or text analytics.

And I want to look at one particular example, the CHEAT approach to a text analytic challenge, the Morpho Challenge. I'd like you to have a look at the website for NLTK and just go to nltk.org, and also to read a couple of short papers from conference proceedings on these topics-- one on the NLTK, Natural Language Toolkit by Steven Bird and another on the Combinatory Hybrid Elementary Analysis of Text or CHEAT system by myself, Eric Atwell.

You could say a lot of computational linguistics or text analytics can be done by linguists and others, computer scientists even, without doing any programming. If you use Sketch Engine, you can collect a corpus or use an existing corpus, analyse it in various sorts of ways, get results out, analyse the data, and come up with conclusions. You can do text analytics experiments without any programming.

And if it's not a functionality within Sketch Engine, then Weka is another toolkit, where you can load in data, run various filters to change its format in different ways, you can choose from a very wide range of classifiers, clustering algorithms, association algorithms, and so on. There's visualization tools, data analysis tools, and so on, all within Weka by graphical user interface pull down menus.

And that works fine. If you want to try out some existing machine learning algorithms or other existing algorithms, but if you're a researcher or if you're doing a research project where you want to develop and test some new machine learning algorithm, it won't be in changing your Weka. By definition, It's new. So therefore, for those special cases, you do need to write some code.

For example, in Python, you don't just have to write from scratch. You might look at some existing tools that are already there, like NLTK. I'm going to look in a minute at the CHEAT program. If you read the CHEAT paper there is a very simple Python program written from scratch, which does the analysis.

But really, you shouldn't have to code from scratch using basic Python. For example, if you look at the Google Code Archive, or NLTK, or many other toolkits, there are quite a lot of existing bits of code which you can reuse, and a smart program can be just a few lines calling on some library functions.

This may sound a bit strange for a computer scientist, but you don't have to program. In fact, you shouldn't program unless you actually have to. The first thing you should do when you have any experiment, or any task is to look at the task and see what's the easiest way of doing it. And it may be there is already some tool like Sketch Engine which will solve your problem, therefore you don't need

to code. So, don't code unless you really have to. If for example you're trying to design a search project proposal, you can think about what sorts of methods you're going to use, write up a work plan, and without doing any coding at all.

Let's look at Natural Language Toolkit. It is only one example. If you google text analytics Python or computational linguistics Python tools, this will probably be one of the top ones you find. You may find reviews of different tools and NLTKs has been going on for a long time. And so, it's there. NLTK is the leading platform for building Python programs to work with human language data.

Well, they would say it's a leading platform. This is a quote from the website, It provides an easy to use interface to over 50 corpora and lexical resources, such as WordNet. WordNet is a resource which has words grouped together into what they call synsets, groups of words which have similar meaning. And they have relationships between them as well. And you can call this WordNet network or ontology directly within NLTK and others as well.

There's also a suite of text processing libraries for text classification, text tokenization, stemming-- that's working out that stemming is made of "stem" and "ing," or tagging-- like tagging is a verb in this case, as opposed to an adjective, parsing-- working out the grammatical structure of sentences, semantic reasoning-- working out the semantic structure and making inferences from text. There are also wrappers for industry strength NLP libraries, so you can access tools outside of NLTK from within NLTK wrappers.

And there's also very importantly, an active discussion forum. If you sign up to the NLTK discussion forum, then if you have a question, then you can pose it and get answers from other people, or you can just listen in to see the other discussion, get good ideas for ways of doing things. It also has a hands-on guide, introducing programming fundamentals alongside topics in computational linguistics. There's an online guide with lots of API documentation so how to connect to NLTK from other tools or how to connect NLTK to your other programs.

And there's a textbook. The authors Steven Bird and others have developed this over many years and written quite an extensive textbook on introduction to natural language processing using Python. I did consider using this textbook for this course, but it is a bit specialized. It is specifically about Python programming and there are other courses, other modules, in the program where you learn more about programming. I thought for this module, we'll think more about the theory and underlying methods and assume that you can do programming, so I don't want to spend too much time on programming.

The NLTK also has lots of tools not just for text analysis, but for testing your classifiers, displaying results, things like that. For example, if you want to use the parseer to work out the grammatical structure of a sentence it will do that and it will come up with internal data representation-- the internal data structure that if you want to display it on screen, then you want to draw a tree structure. And here we have three lines or two and a half lines of text of Python code from NLTK.corpus import treebank.

Treebank is a module containing lots of parseed corpora, where each sentence has a parse tree attached to it. And then you create a variable t, and you give it the value treebank parsee sentences from wsj_001.mrg. And WSJ-- if you know anything about computational linguistics-- that's the Wall Street Journal, one of the very first large scale corpus or corpora of American English from the Wall

Street Journal, the big newspaper in New York. And they've taken those texts at every sentence, added a parsee tree manually, so that we have a treebank or collection of sentences with trees attached.

And this is just extracting the 0 for the first sentence in there with its parse tree. And t.draw, draw is just a way of drawing the parse tree. And we see here is a picture, there's actual sentences, Peter Vinkin, 61 years old, will join the board as a non-executive director November 29th, full stop.

That's the sentence. And we see that the sentence structure is that first of all, there's a noun phrase subject, Peter Vinkin 61-year-old. And then there's a verb phrase, will the board as a non-executive director November 29th. And finally, there's a final punctuation, full stop, and so on. I won't go into more details but you can read this tree as a human, whereas the internal data structure won't be so readable. It will be usable within Python.

NLTK is not the only toolkit available. There's plenty more out there. SpaCy is another one that's widely recommended. I'm not going to go into more details but you can click on this link. If you just google SpaCy or go to spacy.io you'll find out more. SpaCy is particularly popular because it has connections to some of the latest research systems in natural language processing.

NLTK, because it's been developed over many years, it has lots of resources in it but it's more designed for educational purposes and has less direct connection to industrial interfaces, I guess they could say. If there's anything missing in NLTK, there's even a wanted list, so you can specify, I would like NLTK to have x, and then the development team will consider adding it to NLTK.

Apart from general toolkits like NLTK and SpaCy, there are also toolkits for specific sorts of NLP, natural language processing tasks. For example, Gensim has very good techniques for topic modelling and for modelling semantics with sentences and words as vectors for things like, extracting the topic of a document. You could also, if you just google NLP toolkits or Python NLP toolkits, then you'll find Google will turn up numbers of surveys for example, The Top 10 Python NLP Libraries of 2020. That's the first of these links and another one is, Python Libraries for Natural Language Processing, just as a survey of them.

And as well as just listing them, these two surveys also tell you what's a specialism of these things. For example, Gensim is in both of these lists and it's specifically good for topic modelling. And NLTK is in both of these lists and it's particularly good for learning and university education, it's saying. It's widely used in universities for teaching natural language processing.

All these toolkits are very useful if you're going to be developing some new variant of an algorithm or if you want to try a range of different algorithms for some new tasks. So as I said before, if you're trying to build some experiment using known techniques, you might consider simply using Weka or Sketch Engine or some graphical user interface where you don't have to do any programming. You really only want to do programming if you're doing something novel which hasn't been done before, therefore isn't available in one of these tools.

Talking about novel AI research, I have already recommended that you read conference proceedings papers from latest AI and computational linguistics conferences. Very often, these conferences have papers by researchers on the latest research they've just done. For example, the Banko and Brill

paper on scaling to very, very large corpora was a novel piece of research by Microsoft Research Labs and they thought they might as well publish it.

Conferences often particularly in AI and machine learning and in text analytics or computation linguistics, they often have as well as these sort of one-off individual researcher's conference papers or groups of researchers conference papers, they have what they call a shared task or maybe several shared tasks. And the idea is that some organizer has provided an annotated data set and some sort of evaluation and training data sets and a particular task, something that you have to do.

You have to learn something and then anybody can participate. And each of the participants tries to solve the task, comes up with a score using the standard evaluation data set, and then they write a paper about their approach to solving the task and what their score was. And then at the shared task workshop, people can compare their methods and their results and there's usually some winner who gets a handshake or something like that.

SemEval is a good example in text analytics. The magic evaluation tasks for SemEval conference every year has research papers by research groups and also maybe 5 to 10 different shared tasks or competitions. One that I was particularly involved in, not in SemEval but separate from SemEval, was one called Morpho Challenge. And there's the website for Morpho Challenge or if you just google Morpho Challenge you'll find it.

And this was about unsupervised machine learning segmentation of words into morphemes. What that means is a segmentation is-- for example, the word unsupervised can be chopped up or segmented into "un," "supervise," and "duh." Or possibly, it's "un," "super," "vise" and "duh."

And the task is to come up with an algorithm for taking text, not just English text though, in difficult languages like Finnish and Turkish-- Finnish and Turkish are notorious or famous for having quite long words and each word is made up of many morphemes. In English, most words are just one or two or maybe three morphemes. In Finnish or Turkish, they have longer words with several morphemes in them.

The task has to be unsupervised. That means you just get given a list of words in Finnish, let's say, and your algorithm has to somehow work out just from a list of words and from repeated patterns in the words what are likely to be the morphemes. So for example in English, if there's a lot of words starting with "un," that's a clue that "un" is a morpheme. It doesn't tell you what "un" means, but at least it gives you a clue that "un" is a morpheme and whatever is left is also morpheme, so unsupervised suggests that "un" a morpheme and "supervise" is a morpheme. There's lots of words ending in d or e d, which suggests that "un" is a morpheme "supervis" or "vise" is a morpheme, and "ed" or "duh" is a morpheme.

OK, so that's unsupervised machine learning for segmentation of words into morphemes. And because we were developing new machine learning algorithms, we couldn't just use a toolkit, so we had to code in a programming language like Python. And if you read the paper, you'll see there is an actual example of my first Python program which does this.

It didn't work perfectly, so then I had to get my collaborator Andy Roberts from Pearson. Pearson is a company you might have heard of because they recruited you in the first place, maybe. Anyway,

Pearson allowed Andy to work with us on developing a rather more sophisticated version, essentially the same algorithm but with some extra nuts and bolts in it.

The very first Morpho Challenge was back in 2005 and I thought this was a great idea. They had this task, had some data sets, that is lists of words in Finnish and Turkish and also in English, just for people who can't cope with difficult tasks like Finnish and Turkish. And the task was simply come up with a unsupervised machine learning algorithm, which will be able to segment words into their morphemes.

We were given a training set and a separate test set, and we had to produce our results on the test set and send them off to Helsinki University in Finland, who organized the conference. And I thought this would be great. This would be quite challenging for MSc students to do, so let's do it as a piece of coursework for MSc students.

Because it was quite hard, I said pairs of students can work together to come up with a solution. Some of the lead students took this very seriously and actually entered the real contest. But even those who didn't, I was able to get their results because they had to submit to me to mark their coursework and get a grade from me, their code and also their results.

And that meant I was able to use their results in an ensemble. CHEAT is actually an ensemble classifier which combines the results from several different Leeds student entries into a combined system. And as we saw in for example, Banko and Brill found that an ensemble of several classifiers is generally better than one individual classifier. And the same thing happens with unsupervised learning. An ensemble is generally better than all of them.

So now to end this lecture, I'm going to give you a treat. I'm going to show you a video from 2005 because we presented the paper at a conference and the conference was actually-- this was a novel, new idea at the time. Why don't we video record the lecture, the talk, and then put the video recording on the internet so our people can see it too?

And this was because back in 2005, a company called YouTube started and they invited people to upload their own short videos of people doing fun things like dancing and smiling, or the very first one that got famous was somebody went to the zoo, and they took some video of elephants, and they said elephants are really cool because they've got really long trunks. And that was it. That was amazing.

It's a bit like TikTok now. My granddaughter is into TikTok. TikTok lets you record something, and you have a maximum of, I think it's two minutes, for your video. Back then YouTube allowed 10 minutes. And if you click on this link, you'll see not the very first YouTube video, but rather a news story on, I think it's CNS news, some American news channel, talking about the start up company YouTube. That's interesting.

Of course, academics at the time knew that we tended to give lectures that are longer than 10 minutes, so a group of academics set up another website called videolectures.net. And here we have, if you click on this, my lecture-- It's a bit longer than 10 minutes-- on the CHEAT ensemble. This is the presentation given at the conference.

The presentation was actually based on a PowerPoint slide where I had pre-recorded my voice on each slide and then for the lecture, I just played each of them one by one. That's what I'm going to do now. If you want, you can click on this and watch the video-on-video lectures.

Unfortunately, the technology they used at the time may not work on some of our current browsers, so you may have difficulty actually watching the video there. What I've done is I'm now going to include the video, because it was recorded in 2005 on very early technology, sort of Black and white movie, not almost, not quite. It's quite a crackly sound.

I say black and white, It's not black and white. It's green background. You may wonder why the backgrounds for these slides are different from the backgrounds on other slides. This is the standard University of Leeds PowerPoint format from 2005. Back in 2005, we had to have this University of Leeds logo here, and we had to have a green band here, and a green background, and use black text on green background. This whole lecture has been in the style of a 2005 lecture.

And to finish off now, I'm going to go into the talk I gave on the CHEAT system. Enjoy and thank you very much for listening. And I'll hand over to Eric Atwell from 2005.

[AUDIO PLAYBACK]

This talk is entitled Combinatory Hybrid Elementary Analysis of Text, The CHEAT Approach to Morpho Challenge 2005, A Carefully Crafted Acronym by Eric Atwell and Andrew Roberts, with the help of Eric Atwell's Computational and Modelling MSc Class, Karim Ahmad, Adolfo Allendes Osorio, Louis Bonner, Sa Chaudhuri, Min Dang, David Howard, Simon Hughes, Iftikhar Hussain, Li Ki Ching, Nicholas Molaison, Edward Manley, Kalid Rehman, Ross Williamson, and Hong Tung Xiao.

Our guiding principle is to get others to do the work, as inspired by Homer, Homer Simpson that is. We all know that plagiarism is bad but in software engineering reuse is good. However, we can't just copy results from another entrant. But, we may get away with some smart copying.

We can copy results from many systems, then use these to vote on analysis of each word. But how can we get results from the other contestants? Well, we decided to set Morpho Challenge as an MSc coursework exercise. Students must submit their results to the lecturer for assessment before the contest.

But you might say, is this really unsupervised learning? Well according to the Morpho Challenge website, the program cannot be given a training file containing example answers. Well, our program is given several candidate answer files but it doesn't know which one, if any of them is correct. So, it is unsupervised learning by these definitions.

Moreover, it is inspired by the recent film Super-Size Me. We call this triple layer super-sized unsupervised learning. Firstly, unsupervised learning by the students, then unsupervised learning by the student programs, thirdly unsupervised learning by our cheat.py program.

The first stage isn't really machine learning but unsupervised learning by students. Eric Atwell gave background lectures on machine learning and on the principles of morphological analysis. So, the students were not given example answers, in this case unsupervised morphology learning algorithms.

Students had to work this out for themselves, so student learning was effectively, unsupervised learning.

The second layer was unsupervised learning by the student programs. Pairs of students developed Morpho Challenge entries. For example, Sa Chaudhuri and Min Dang and Khalid Rehman and Iftikhar Hussain both developed systems that are published in the proceedings. But as far as we were concerned the student programs were just black boxes. We just needed the results, the output of the programs.

The third layer was unsupervised learning via Python program cheat.py. It read outputs of the other systems line by line and then it selected by majority vote analysis. If it's a tie, it selects the result from the best system, the one with the highest F measure. And then, the program just outputs this result, which is our result now.

The CHEAT program was simple enough to be printed in full in the proceedings paper and it worked in theory. But unfortunately, some student programs read in the input and reordered the word list before putting it out again, so outputs were not aligned like we like. Luckily, I managed to get Andrew Roberts to develop a more robust cheat2.py and this program really did work.

[END PLAYBACK]

And here we have some examples of the outputs. Notice that black box is the combined results and the ones before it are this student in the wants for Finnish and the same thing for Turkish. The black box is the actual combined result, and the student ones are the other ones.

Notice that at all one isn't the best by any means, but the conference organizers took this t approach and here we have the last two columns. They combined C orders, they combine all the results and gave a system which was quite good but not brilliant. But if you combine just the top five, then C top five is better than any of the existing systems. This is the example for Turkish.

[AUDIO PLAYBACK]

In conclusion, we find that machine learning and student learning can learn from each other. It turns out that our CHEAT program is actually a committee of unsupervised learners. One of our reviewers pointed this out. For example, see Bank-owned Brills 2001 paper.

But we didn't actually learn this from literature until after we'd done the program, so maybe this is a forth layer in supersized unsupervised learning. But CHEAT is also a novel idea in student learning. The idea is to get students to implement the learning programs. And so, the students learn about machine learning as well as about the domain, in this case morphology.

All in all, the Morpho challenge contest inspired our students to produce outstanding coursework, so it was worth it for them. We'd like to thank the Morpho challenge organizers for a truly inspiring contest and thanks to the audience for sitting through our presentation in absentia. Bye.

[END PLAYBACK]