# Data driven parsing for the real world

## Abstract

Data-driven parsers have been used in AI and inductive reasoning in ways beyond the abilities of rule-driven parsers. This paper presents a small bibliography and guidelines for developing multipurpose data-driven parsers in hopes that developers will assist the author in disaster recovery and OLPC development, will push Free Open Software to new heights, and will benefit from commercial opportunities.

Data-driven parsers of the highest types don't require a predetermined set of rules like rule-driven parsers do. This is most like discovery learning that maybe helps children to learn their first language so easily and fluently. The author contends this same type of learning is necessary for natural responses to a wide variety of real world situations. The data drives the progress and in the higher types can change goals just like it real life. Rule driven parsers are primarily useful in closed languages such as computer programs where the parser developer can know all of the rules before starting.

Declarative programming is the basis for some data-driven parsers. One of the beauties of Python is the flexibility to do easily almost anything any other language can do including declarative programming.

# About the author

Johnny Stovall entered the digital world by plugging cables into an IBM card sorter to enable him to sort his 4 language dictionary in 4 different ways. The limitations of the device did not permit data driven programming. Programmers could only make up rules and instruct the device to follow them.

FORTRAN, Cobol, BASIC, and dozens of other computer languages including Java were not really capable of making data driven programming easy. Python makes it as easy as possible but the author spends a large portion of his life doing temporal things such as disaster relief.

Data driven programming in Python will change the mindset of anyone who does it right. That change of mindset can open previously unimagined ways to actually change the world or to accomplish a multitude of lesser goals. The author's fondest wish is that this short talk will be enough to convince you to make a commitment to start learning Python and data-driven programming and continue to keep at it until the computer begins to discover things you could have never known otherwise from the datasets the world and you yourself consider to be most important.

Data-Driven Parsing For The Real World



This paper will follow the last in first out order of solving problems in the same manner as the ancient Greek classics and the Bible. We begin with the problem of people with limited thinking because of a lid. Then we grapple with different ways people want to use computers. This leads to a discussion of the rules for understanding. Next we attack the problem of who or what will be in control. We change direction by tackling the problems of control, then better ways to use computers, then rules that always work, and finally how to remove the lids that hold people back.

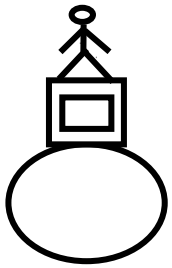
(Graphic1 Trained fleas.) There used to be a lid. Now the lid is off but the fleas will not jump as high as they did when there was a lid.

Many people have an imaginary lid as to how much they think computers can do or how much they want computers to do for them. Even if you are more technically advanced than the rest of us, this introduction will help you deal with other people who had hit a lid in old or inferior software.

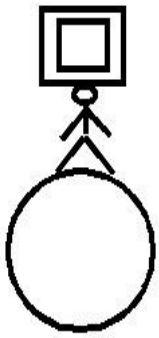
Before we get into the specific details of data driven parsing it is helpful to understand their place in the context of a much larger world. Most people think that human beings have always been and will always be the most intelligent creatures on this earth. I started programming computers in order to assist translators. Several languages including some that I did not understand were translated. The translators said that the translation could not have been possible without my computer assistance. They nicknamed me, “The impossible man.”

Garbage in garbage out was one of the most common phrases and thought patterns of those days. Most people would tell me that you only get out of a computer what you put into it. When I disagreed, most of them thought I was crazy.

(Graphic2 Person on top of computer)Is this the way you want the world to be?

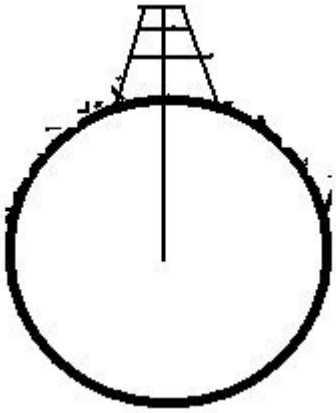


(Graphic3 Will computers rule the world?) Some people fear this will happen.

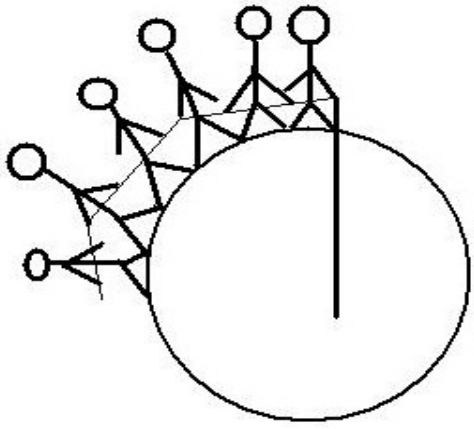


Now it is more than 10 years since the IBM computer Big Blue defeated the world's champion chess player. Most desktop and laptop computers today have much greater power than Deep Blue. They have also repeatedly proven that they can defeat any human chess player when programmed correctly. Opposition to computer programs that learn became less frequent. The real problem is people who unconsciously ignore all of this power. If you think that you are smarter than any computer will ever be then you will never discover the full power of a computer. Because, if you think that a computer is only a tool limited to giving you back a subset of the knowledge that a programmer put into it, then you will not write computer programs that help you learn things you could not have learned without the computer.

(Graphic4 Oil Well rig.) People do not have a problem wanting machines to do things that are beyond the abilities of humans in the physical realm.



(Graphic4B 10Ks people turning the oil drill.) The drill will only move as fast as the person at the end of the bar who must cover a lot of ground to make one revolution.

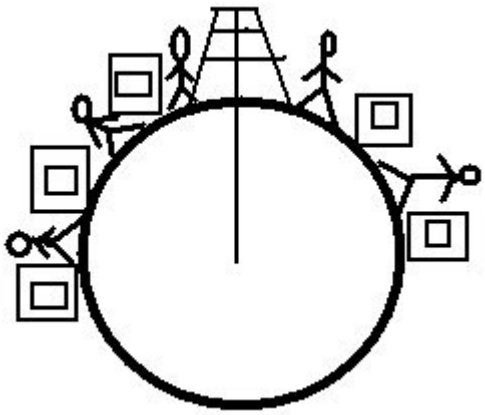


Since 1972, I have been telling everyone who knows me well that my vision of the future is that machines will do mental exercises beyond the power of your mind. But that will be good because after the machines

have done it, we will use the results and the procedures to enable our minds to do more than they could do before the computers showed us greater mental discoveries and how to obtain them. A little computer history will show why I am so confident that this is the future.

In 1975, no person could mathematicaly prove if any arbitrary map could be done in four colors could be painted without running the same color together. When the first mathematical proof of the four color theorem was shown by a team of expensive computers working on it for more than a year; some mathematics professors questioned if it was a valid solution because they could not understand it. But the fact that the first extremely hard mathematical proof of map coloring had been proved on computers caused others to press on to even greater heights with renewed confidence. Only four years later the mathematical proof was shown again with a better algorithm. And since then even more difficult problems have been put forward and solved with the help of computers. Here is my vision of the future. Computers will be working alongside people, neither below or above them because people and computers have complementary capabilities necessary to make a good team.

(Graphic5 People and computers side by side.)



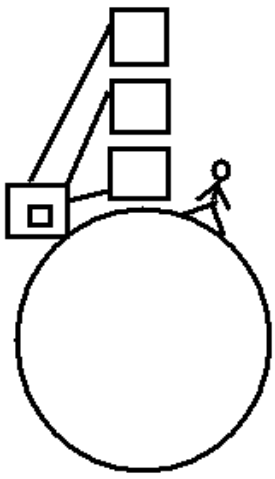
Every instance in every category should only be doing what it does best.

Computer programming is a subset of problem solving. There are three major ways to solve problems: top down, bottom up, and mixed or from any direction. The natural man is filled with pride and think that people are and at the top and must always be at the top. Therefore most people tend to reason top-down or deductively. Your thinking is not bad, if you think that computers are just a bunch of electrical switches incapable of thinking. In addition you might realize that animals can think and solve immediate problems. You may believe that only humans and a Supreme Being can think in terms of the distant future even eternity and other abstract concepts. Even if this is your belief system, I hope you will leave this talk with your mind open to the possibility that enough of those electrical switches turned on in the right way might enable you to learn more about eternity and all other abstract concepts. Because if you do not believe that billions of electrical switches might help you do abstract thinking then it is certain that they never will. Evidently most people either do not believe that they can discover knowledge with these electrical switches or they do not know how to do it.

Most computer programs consist of the computer programmer telling the computer what to do. This is rule-driven programming. The programmer makes up the rules and the computer follows them. This is also top down programming. The programmer makes up the rules instead of finding a way for the data declare its structure and meta-data.

The world has only a few data-driven programs compared to the many rule-driven programs. Many of the few data-driven programs are parsers. I will define a parser as something that looks at information and gives it a structure. This structure often enables new understanding or new ways of behavior. The most common use of a parser is to compile a computer program. In the early '80's I had a friend who had purchased an expensive database engine. He also owned one of the largest commercial databases in the world and obsolete computers that had been accessing that database much faster than humans but thousands of times slower than the database engine and his new computers could have accessed that database. What he needed was a new and simpler language to enable users to easily query that database and a parser to put those user queries into a structure so that a code generator could generate instructions for his multi million setup. In other words he needed a compiler. I took a compiler compiler and instructed it to generate a parser while I instructed an assembly language programmer to write a code generator for the structure I told him that I would produce. This is the way that computer languages are usually built and delivered to programmers. We generally use rule-driven parsers for computer languages because we are sure that we know all of the rules for the tiny invented languages necessary to do rule-driven computer programming. Grammars are nothing more than rules. I will guarantee you that your textbook and teacher for English grammar did not teach you all of the rules of English.

(Graphic6 First example of the importance of discovering rules.) This is a business example of computers discoveries taking the lead to greater profits.



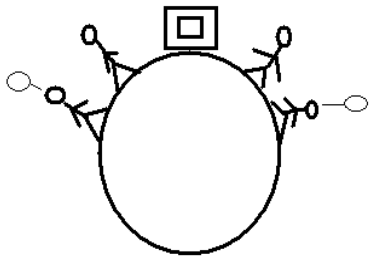
The computer has produced money machines. The inventors and users go to the machines and pick and pick up more money. In reality this is only partially true

for a few companies and some of the world's governments because most of them do not know all of the rules.

Some of the grammars that have been proposed for data-driven parsing are constraint grammars, link grammars, memory based grammars(Kontos et al., 2003) and (Nivre, J., Hall, J. and Nilsson, J. 2004), assumption grammars, semantic grammars, regular expression grammars, hand crafted grammars, context-free grammars, finite state grammars, transformation-based grammars, logic grammars, continuations and hidden accumulator grammars, metamorphosis grammars, extra position grammars, free word grammars, datalog grammars, discontinuous grammars, discourse grammars, experience-based grammars, and approaches that claim to be outside of grammar.

In the real world we do not know all of the rules. Is it enough to just go by the rules we do understand? I will give two examples which I hope will convince you that it is not. Wal-mart, the largest retail store chain in the world, has an elaborate computer network that enables it to be the leader. They were getting data from some locations about some puzzling Friday night sales trends. They saw that on Friday nights these stores sold more baby diapers and more beer than at other times. So they moved baby diapers and beer to the front of these stores beside each other and ran sales on these two items on Friday nights. Those stores became more profitable because sales of these two items increased but also because they were also able to dramatically increase the sales of high cost men's items such as power tools and men's suits. The discovery they made could have been made with a data-driven parser. If you come to my sprint and help write a data-driven parser, I will tell you how you can use the knowledge you will learn in the sprint to get a job discovering the same kind of things for business.

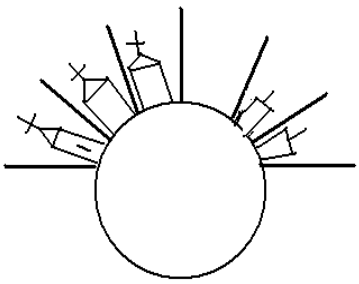
(Graphic6b Another example of the importance of discovering rules.) This shows the consequences of the lack of data-driven discoveries in education.



One of America's very senior State Department Officials came to Indonesia and started his speech like this. "I always remember three things when I come to Indonesia. Indonesia is very dangerous. Indonesia is very important. And Indonesia is very very complex." Much of the rest of the

speech and questions dealt with the complexity. Near the end, I gave an example of how complex things become simple once you understand. Then I said Indonesia is complex because the education system has been poor. If Indonesian education would help students understand how to find the right direction for themselves, then they would simply be doing what is best instead of being blown about by every rumor. So what are we going to do to help Indonesian education? The head of US AID told me that question should have been a home run but he missed it. If you also missed it, I will give you another three/quarters of an example about people who do not know how to find the right direction. This example is very short and terse, so please pay close attention.

(Graphic7 Churches and Mosques with walls on all sides of every one.) Two claims to be the only solutions to bring peace in the world based upon One Perfect Creator And Ruler are hopelessly divided because people interpret His simple Rules in thousands of different and ridiculous ways.



More than half of the world believes that all of the rules, at least all of the spiritual rules, are given in spiritual books. The two most popular collections of letters, rules, and prophecies are the two most misunderstood and misapplied books in the world. The reason is because letters, rules, and prophecies only have meaning in context. A computer can discover true context based upon the whole but people interpret based upon the parts they know and in light of their own context. This results in literally thousands of misinterpretations for each of the hundreds of simple rules.

So at this point I hope you are asking, "How can the computer help us find real world rules?" In a nutshell, I think there is a multi fold answer. First computers can examine all of the data in a variety of ways. In the spiritual example I just gave, most of the world is looking at only part of the data and doing so from their limited perspective. My experience leads me to believe that this is exactly what is happening in every aspect of human life. Second computers can learn by association like little children. Third computers can learn by logic like little children learn by logic. Every normal child from the beginning of time until now learns the languages in which the parents talk to them. They do this without grammars, dictionaries, or any modern technologies simply by making repeated associations and using childish logic.

According to most of the literature data-driven parsing is a means whereby the computer

discovers rules. I like to define it differently. I say the computer is discovering associations just like a little child discovers associations. A little child associates milk or any other object with whatever words the parents use with that object. They associate run or very or any other verb, adverb, adjective, preposition, or whatever with what the parents and other acquaintances associate it. Anyone who has studied psychology knows you can put people or animals on different sides of the same room then ring a bell, turn on a light, or do anything and then administer punishment to one side of the room and reward to the other side. After you do this a few times, both sides of the room will have a very different reaction every time the signal is given even before you administer the punishment or reward. This is the way little children learn and it is the easiest way for us to enable the computer to learn for us. Archaeologists looked at many Egyptian hieroglyphics without understanding anything until the came across the Rosetta Stone. Just a small amount of hieroglyphics with parallel translations to languages they did understand and they could almost immediately understand those same words in other places.

But it was not long before they could understand many words which were not on the Rosetta Stone by using logic and the associated context. Data-driven parsing can do essentially the same thing. It can learn the easy part of data you give it, then learn more and more difficult things from the same kind of data. One way that it can do this is by using constraints to fill in the gaps where an association must be made but the parser has never before seen any portion of the association that must be made. This is more exciting than the kinds of neural networks which are limited to recognizing patterns that they have already been taught.

What data you give it to learn, especially in the beginning, is one of the most important determinants of performance. This parallels what happens in little children. If you teach a little child sign language and a spoken language at the same time they will be fluent in both. They can use sign language as a tool to learn other languages rapidly. Most people are able to think in all of the languages they learned as a little child and continue to use. But for people who only learn one language as a child, the ability to think in other languages or even to learn them is much more difficult. In my opinion, language acquisition is a tool to help the one who learns a language to learn whatever else is available in that language. You could feed retail chain store sales statistics to a data-driven parser and have it alert you to all kinds of associative information such as increased sales of beer and diapers on Friday nights. But if the domain of knowledge is limited to this pattern-matching type of learning would not enable that data-driven parser to use the information just learned as a tool for learning how to change a diaper, explore the solar system, or anything else completely unrelated to sales trends. But if language is learned first and learned correctly, that learning can be used by the data-driven parser as a tool for learning anything that is related to that language. Since most things in this world are related to language, I think it is the best place to start. If you want to start computer assisted learning with language acquisition then you are in luck because you are not likely to find in the literature other uses of data-driven parsing. As far as I know, you have nothing other than my word and the experiences you will gain in writing a data-driven parser to guide you in using a data-driven parser to learn associative information outside the realm of

language acquisition.

I have done my best to find the references to data-driven parsing and annotate them in order to save you time in determining how you can implement a data-driven parser. But before I give those to you, I want to make some observations.

Obviously, you must start with the easiest part of whatever language you want the data-driven parser to learn. That varies from language to language. I will start with the easiest to learn well-known language which could serve as a stepping stone to the acquisition of other languages. Prepositions, inflections, adjectives, and adverbs that have an obvious and almost unvarying meaning are used extensively in classical and koine Greek. In the koine period most educated people in the western and middle eastern world spoke Greek because it was easy. It is possible to learn 18 prepositions, 100 adjectives, 50 adverbs, 250 inflections and 10,000 words. This results in 225,000,000,000 possible combinations. It is impossible for me and you to learn a million words much less 225 thousand times a million words. But when thousands of millions of natural meaning combinations are available in a language that has been the basis for many modern languages, it is tempting to use them. This is why most discoveries in medicine or science are given Greek or Latin names in their patents. Since no person alive speaks classical or koine Greek as their native language, this is something that a data-driven parser could learn in order to make discoveries available to people who want to know but don't have the educational background.

Only 5% of the world speaks English as their first language. But more than half of the world wants to speak English as a second language. The system of going from a word written in English to the correct pronunciation of that word is a horrible mess. A data-driven parser could be used in a number of playful learning ways in order to help these people who want to learn to speak English as a second language.

No child learns everything about their first language in a waterfall manner. Language must be learned a little at a time with unknown chunks all along the way. I think the best way to let a data-driven parser go though the same process is for it to put XML markings on everything the parser thinks it understands. Making multiple passes on the same material with previous XML markings enables the data-driven parser to concentrate on only one type of learning each time the material is parsed. This is much less complex than trying to develop a waterfall type parser that would attempt to learn everything in one pass. The XML tags can be embellished with many other principles learned from artificial intelligence. One such principle is to put a confidence level on everything that is conjectured.

I hope at this point that you are asking, “What is the best way to do this?” It has been proved that

anything that can be computed can in theory be computed using a Turing machine. But with the abundance of computing power and different computer language possibilities available today; it is wise to use the very best that is available. That is why I am giving this talk at a Python conference instead of at a meeting for BASIC users. When I give you my comments on the best publicly available data-driven parsing literature I could find, you will notice that many of the most successful projects that have been done use functional or declarative computer languages. The more abstract, complex, and systematic the problem is to be solved the greater the benefit will be from using this type of language. Simple imperative languages are best for solving simple rule-driven problems. We must be thankful that Python has this ability. I use it daily. But Python also has the functional language capability to solve problems arising from complex self-interpreting data that people usually misinterpret.

Python has some parsers developed with the express purpose of letting users expand them. The PyPy compiler is designed output multiple languages. This parser can already expand the C programming language by adding badly needed security features not available in other C environments. PyPy has already reached version 1.0 with the help of many volunteers sprinting at previous conferences such as this. In fact after all the conference talks are finished, anyone who wants to can join in sprints to provide data-driven educational software, improve PyPy, or any of several sprints that suits your interest. The ZestyPython compiler is still in the early stages of development but it has been designed with the objective to allow users to expand it in any manner they desire. You may download PyPy or ZestyPhython, or many other good free open source Python compilers at http://cheeseshop.python.org.
As always, a Google search will enable you to click and read thousands of articles concerning almost any aspect of either of these compiler-compilers.

The data-driven parser with the longest history of success in the areas where I searched is Maltparser. Versions of this parser require that words be input with tags for different types of attributes. A word is an element with a subset of the six attributes:

1. **id** = Unique id within the sentence.
2. **form** = Word form (string).
3. **postag** = Part-of-speech tag.
4. **head** = Syntactic head (word id).
5. **deprel** = Dependency relation to head.
6. **chunk** = Chunk category.

Maltparser considers a valid input to be a treebank. A treebank is a sequence of sentences. A sentence is a sequence of words.

This is accroding to http://w3.msi.vxu.se/~nivre/research/**MaltParser**.html retrieved several times in 2007. It also says,

MaltParser 0.4 was used in the CoNLL-X Shared Task on multi-lingual dependency parsing in the system that obtained the second best overall score, not significantly worse than the best score, and

that achieved top results for nine languages out of thirteen (with results significantly better than any other system for Japanese, Swedish and Turkish). In this system, MaltParser was combined with pseudo-projective parsing, which requires preprocessing of training data and post-processing of parser output (Nivre and Nilsson 2005). The complete system is described in Nivre et al. (2006). More information is available at:

- CoNLL-X Shared Task: Multi-lingual Dependency Parsing (home page of the shared task with all the official results)
- MaltParser in the CoNLL-X Shared Task (local page with complete information about feature models, options, etc.)
- Pseudo-Projective Parsing (pre- and post-processing tools necessary to reproduce the MaltParser results in the shared task)

Maltparser allows multiple configurations for testing the performance of various types of grammars. Many non-linguists think of grammar as only parts of speech put together in a prescribed order. This author has always personally defined grammar as everything that could be known about a language or anything else that could be expressed to convey meaning. This is the most general case. There are many different ways to derive a subset and declare it as a new type of grammar.

Many of the computer programs that continue to defeat the world's grand master chess players are based upon the artificial intelligence concept of tree pruning which is the same as the parsing or grammar concept of constraints. Using a constraint based grammar or adding one to an already viable grammar is likely to cut processing time. I have already told you that the grammar I want is the one that covers every rule for the domain. The only way to build a grammar that covers every rule is through data-driven discovery methods. So these are the kinds of grammars that we will build in a series of sprints. If you want to pronounce English correctly and help others to do so also, it would be very helpful to know all of the rules for English pronunciation. There is a CMU free and open dictionary of traditional spelling and correct pronunciation for the Spinx CMU speech recognition system to discover the rules for English pronunciation for this domain. There are many other domains which are structured so much like this that you can use the core of the same kind of program to learn them. I also can point you to Free and Open Source XML databases of natural languages with parallel translations. The XML will greatly assist you to develop a data-driven parser whose core looks very similar to the core for going from traditional spelling to pronunciation. You can sprint from anywhere in the world and at any time. All you have to do is send

e-mail to oouc@yahoo.DELETECAPS.com until you get past the spam filter and I reply. Please put

Data Driven? at the beginning of the subject. If there is sufficient interest then write down your questions or eMail them to me and we will have an open space discussion.

If you only want quick and dirty partial knowledge, perhaps any grammar any parser will be beneficial for you. A leader's greatest satisfaction is when they say something which causes someone to discover something which will help them keep on getting what is the best for them. I am going to close

with a story that illustrates how this has worked for someone else and how this talk can do the same for you. When I met him in the early 1980's, Charlie spent much of his time writing BASIC language programming tutorials for computer magazines. Every time I encouraged him to move up to modular programming in Pascal, Charlie reminded me that anything that can be computed could be computed on a Turing Machine. Since he already knew BASIC, he said it was not worth his time to learn another computer language. Charlie eventually enrolled in one of my Pascal classes. Before the course ended he was telling everyone, "Even if you never program in anything except BASIC you should learn Pascal. Learning Pascal will enable you to think in a modular manner and solve problems you could not have otherwise solved." Charlie was a genius but he did not realize that immature BASIC interpreters were a lid holding him back. Many geniuses have always developed rule-driven software which is very valuable but limited to the genius of the developer. Data-driven software can enable you to develop software beyond your own abilities however great those abilities are. Python not Haskell, Java, C++, Pascal or BASIC or any other language is the obvious best choice for doing this. These other languages all have the strength of doing one thing very well. Python has the strength of doing almost all of the things all of them do as good or better than they do it.

You have already seen that there are many types of grammars or systems for discovering rules. I once wrote a computer program to grade students ability to write an essay just like what they had read. I used only a complicated statistical formula which I invented but understood very. Another team of programmers had already developed a highly trade secret program that had analyzed thousands of interviews and open conversations to accurately predict the behavior of jurors, consumers, and many other people by making psychological profiles. I was not allowed to know anything about their program. But we both did a data-driven analysis of every word written by 112 students who were attempting to reproduce the same essay. There was no significant difference between our rankings. Since you have seen Maltparser and others use so many types of grammars, my advice is use whatever you understand the best in order to get started.

The best data to use to get started is data that has been reviewed by many scholars. Find places where you are more than extremely confident that you understand and there is a great amount of repetition and some unique data. The easiest natural language text that I have found is Matthew Chapter 1 verses 1-14. You should be able to pick out the word that means "had a descendant" without any problem. Then you can see how names are rendered differently. You can use this information to quickly learn other things. With a little thought you can develop a Python program that uses the same techniques you used without actually giving the program any answers. If you have done a good job of programming this then your Python program can go to several other places and use the same techniques to learn more. You can use this knowledge to have the computer to help you learn more difficult things. The best way to write this program is to have Python reporting pack to you all of the information that it used to make each decision. As more and more difficult information is learned, the computer will eventually be teaching you almost everything and you will only be giving an okay to approve or else trying new learning techniques or other data to prove that it is right.

http://cheeseshop.python.org

Kontos, J., Malagardi, I., Peros, J. (2003). "The AROMA System for Intelligent Text Mining"

HERMIS International Journal of Computers mathematics and its Applications. Vol. 4. pp.163-173. LEA.

Nivre, J. (2003) An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, Nancy, France, 23-25 April 2003, pp. 149-160.

Nivre, J., Hall, J. and Nilsson, J. (2004) Memory-Based Dependency Parsing. In Ng, H. T. and Riloff, E. (eds.) *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL)*, May 6-7, 2004, Boston, Massachusetts, pp. 49-56.

Nivre, J. and Nilsson, J. (2005) Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 99-106.

Nivre, J., Hall, J., Nilsson, J., Eryigit, G. and Marinov, S. (2006) Labeled Pseudo-Projective Dependency Parsing with Support Vector Machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*.