1 – Biased history of Artificial Intelligence

(I’m using the term biased because it ignores one of the most important fields of AI: that of robotics)

Artificial intelligence or, at least, the study of human consciousness and the attempt to reproduce it, have long been one the hallmarks of mankind. The earliest attempt to organize the thought process in a systematic way can be found in Aristotle’s “Organon”, where you find a very good attempt at defining what would later be called first order logic. The rediscovering of Aristotle in the later middle ages would be fundamental for the way several religions build their arguments of faith.

But the first serious attempts at building machines that think began in earnest in the middle of the twentieth century, despite the first inklings of programmable mechanic computers like Babbage’s analytical engine.

It started with the new found methods of optimization and top down search that were proposed by Alan Turing in the 40s. Claude Shannon added to this by showing that any logical problem could be solved by an electric circuit.

Other researchers, like Minsky, would try to mimic the basic workings of the systems that we knew were capable of reasoning, like the human brain. His ideas were later implemented when he and Edmonds first built a neural network. Later yet, other researchers managed to build even more complex kinds of neural networks like adaline and the perceptron, which could be trained to react to different stimulus.

Meanwhile, their work bore fruit with software that could generate plans from basic axioms. John MacCarthy, the inventor of lisp, was one of the main developers of this line of approach, creating the Advice Taker, a primitive kind of expert system.

In the late 70s and early 80s, expert systems became a staple of business tools. One famous example is Mycin, a medical diagnostic tool as was also CADUCEUS. There were the Jurisprudential Inquiry and Latent Damage System, both applied to law. Engineering, too, had its slate of expert systems, from VLSI design to factory floor configuration.

Those early attempts at commercialization of AI became an important industry in the mid 80s, reaching gross sales of about $2 bln US dollars.

Unfortunately, it all came crashing down due to the frenzied sales pitch and unrealized expectations. Logic based learning systems couldn’t cope with the volume of ontologies needed and doubts were cast about other methods. Minsky proof that single layer neural networks couldn’t emulate a or gate, for instance. In truth, hardware and computer networks were still not able to cope with the demands of those systems. “2001 – a space odyssey” comes to mind: HAL 9000 just wasn’t going to happen with 2000s hardware, even from a late 80s standpoint.

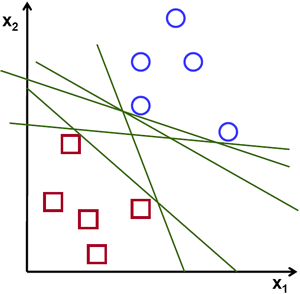
But, starting from the mid 90s, a host of new increasingly more potent hardware and mathematical innovations brought forth a new wave of sophistication. Belief Networks, and in particular, Bayesian Belief Networks, as well as Support Vector Machines and Multi Layer Neural Networks made possible the emergence of more robust AI systems. The field of machine learning, avoiding the use of the term Artificial Intelligence, tainted by the failures of the preceding decade, was again brought to the fore of commercial development. Today, the machine learning is widespread in most fields of human endeavor, be it commercial or scientific. Most of us are, at least, peripherally aware of the convoluted neural networks and support vector machines that identify faces in social media. Or even maybe more aware of the Bayesian networks that correct our misspellings in our text processing and protect us from unwanted mails in our inbox.

2 – Support Vector Machines

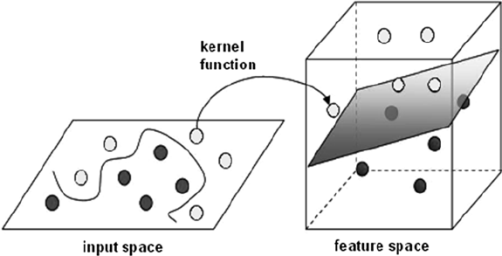
I will now give an example of the ease of implementation and low cost of machine learning , today. But first, let me explain briefly what is a Support Vector Machine.

SVM is a supervised machine learning method that relies on modern developments in numerical analysis: quadratic optimization, kernels and operations on sparse matrices.

The basic idea is to find the vectors that define the hyper-planes that separate two datasets by maximizing the distance between one and the other or minimizing the distance between each hyper-plane and the dataset that it contains.



What if the datasets are intermingled? That’s where the kernels became important. Based on the data available you choose a transformation that widens the space between datasets:



The features that classify a given dataset in the real world are often many but sparse (I will exemplify latter).

One more thing: Vapnick as shown that this trick works in N space. No matter how many dimensions, you can always find a transformation that allows you to separate the different datasets.

3 – A simple example of modern use of Machine Learning (I’m consciously avoiding the use of AI. I’m old enough to remember the crash of the mid 80s)

One of the more time consuming tasks in my line of word, market research, is the classification of open answers in customer questionnaires. Respondents tend to be long winded or terse and each has his own story to tell, sometimes at odds with the question asked. The classification of those answers his subjective and varies with the person classifying it. Long reunions are needed to develop criteria and still there are arguments about a particular classification. To speed up the process of assigning an answer, or parts of one, to a singular category, I developed a method relying on the use of Support Vector Machines. I want to demonstrate that such development was neither cost intensive nor long in development.

First you have to define the “features” of each one of your classes. That means, in this case, simply the absolute frequency of each word in a given class.

The first step in basic text mining (that is the terminology that approximately describes such tasks) is cleaning the text and perform simple classifications. Most of that task can be accomplished in a spreadsheet. Sorting and ordering are common facilities in modern spreadsheet software. A large number of cases can be filtered and pre-classified with those simple tools, mainly by searching keywords or terms. If one of the available answers is everything, nothing or almost, many of the answers can be classified directly with little effort. Even misspellings can be correct quickly, although that requires more time and the use of other plug-ins.

The next step is preprocessing the text. This can be somewhat hard, requiring some knowledge of software building, if your organization doesn’t already have any software to do it. I believe that most of the known statistical software packages have some way or another of doing it. I can only vouch for SAS, SPSS and R, since I’ve used those packages myself. By preprocessing the text I mean the steps of removing whitespaces, punctuation, replacing misspelled words, if that wasn’t done before, and stopwords (words that are used in speech that do not convey additional meaning: “I”, “me”, “my”, “myself” and so on).

Once you classify a sufficient number of examples you can than run your trainer software generating a model (a set of hyper-planes). With that model, given a group of words in a sentence, you can more or less predict the class or classes to which that sentence belongs.

Optionally, although, in practice, almost always, you must run an additional optimization to search for the optimal parameters to your SVM model. That is quite simple if computationally intensive: you lay a grid search of your main parameters – the cost of your choice of hiper-planes (the width of the margin between datasets) and the convolution of your transformation- testing for each point in the grid until you find the best fit.

I’ll specify with the following example:

The respondent is asked his opinion about what changes would satisfy him most with a given service. The classes would be:

A – increase in speed of problem resoution;

B – change in customer manager;

C – manager availability;

D – a greater range of options;

E – less telemarketing;

F – everything.

A sample answer would be:

“I do not want to be bothered with frequent calls!”

After cleaning up the sentence we would have:

“want”, “bothered”, “frequent”, “calls”

Encoding for class A (with want in the 57th position, bothered in the 280th, frequent in the 295th and calls in the 340th:

-1 57:1 280:1 295:1 340:1

Encoding for class E (less telemarketing):

+1 57:1 280:1 295:1 340:1

Once trained and optimized, we would be able to input sentences like:

“I do not care for so many calls.”

And predict a +1 on class E and a -1 on every other class.

4 – Conclusion

For this simple example I used a spreadsheet, python with a common text mining package – nltk – and a free SVM library under current development - libSVM.

I didn’t worry about over-fitting too much because we use a human to correct the misclassified sentences. As the data increases that can become a problem. We had to fall back on statistical methods to prevent this from happening.

Even at this low level of sophistication a full day job with a 4000 observations sample can be reduced to a three hour task, with the added benefit that most observations will be coherently classified by different analysts working concurrently or within a longer time frame. Whatever criteria was used at the time is preserved by the models, which can be saved for further analysis and auditing.