# Machine Learning for Language Processing
# Lecture 1: Classification

Stephen Clark

October 3, 2015

**The ML Revolution in NLP** The plot on the slide shows the percentage of papers at the main ACL conference which report research on statistical NLP. Today, in 2015, the figure would be close to 100%. Before 1990, research in NLP was rule-based, where the rules were written by domain experts (for example translators, for machine translation). The limitations of rule-based systems are well-documented: large and complex rule sets which are difficult to modify and maintain; expensive human input needed to create the rule sets; and difficulty in adapting rule sets designed for one language domain, e.g. finance, to another, e.g. biomedical text.

However, there are advantages to rule-based systems, and it is important to maintain some balance in the debate regarding rule-based vs. statistical. Many commercial NLP systems still use rule-based approaches, for the following reasons. One, they are typically *high precision*, meaning that, if a rule, or set of rules, does fire, the rule will probably give the right answer; and two, the rules are easy to inspect manually to determine why a system made a particular decision. The second reason is especially valued in the context of safety-criticial systems: if an automatic air traffic control system instructs two planes to land at the same time on the same runway, explaining that ten layers of a deep neural network made the decsision is going to provide little comfort.

So why did machine learning take over NLP (and other related disciplines such as speech recognition and computer vision)? The main reason is that ML provides a principled means of learning the required knowledge for intelligent behaviour (*assuming we can provide appropriate training data and an appropriate learning objective*). The amount of knowledge required is huge, and varies for each domain, so it is unlikely that we could ever solve this problem through hand-written rules.

**Some History** In the 1950s, the disciplines related to the cognitive sciences, such as psychology and linguistics, were dominated by ideas from empricism, the philosophical doctrine which states that all knowledge is based on experience derived from the senses. Shannon had also recently published his theory of communication, based on information theoretic ideas grounded in probability theory.

In 1957, Chomsky effectively founded the modern discipline of linguistics, with the publication of Syntactic Structures. This book took a more rationalist approach, with the key data source for linguistic theories being the introspective judgements of native speakers. In 1969, Minsky and Papert published Perceptrons, which was a criticism of Rosenblatt's perceptron (a simple form of neural network), arguing for more traditional, knowledge-based approaches to AI. Both books were enormously influential, and rationalist approaches took over from empiricism.

The tide began to turn again in the 1980s. The IBM group, led by Fred Jelinek, had already been successful at applying statistical methods to automatic speech recognition.[1] Neural networks were back on the agenda with the work of the PDP group, although it would take many more years, and the availability of large datasets and powerful machines, before neural networks could become the dominant force they are in 2015. Finally, the IBM group started to apply the same statistical techniques that had been successful for speech recognition to the machine translation problem (before many of that group, including Peter Brown and Bob Mercer, went to Wall Street to found Renaissance Technologies[2]).

For an interesting perspective on machine learning research in NLP, see [1]. This paper takes the position that perhaps empiricism has been *too* successful in NLP, and some combination of empiricism and rationalism is required.

**Statistical NLP not Science?** The rejected COLING 1988 submission referred to on the slide is the original conference paper describing IBM's approach to statistical machine translation, which is now the dominant approach to MT.[3] The rather hostile tone taken in the review was not uncommon at the time, and can be interpreted as reflecting a general misunderstanding of how probabilistic and machine learning approaches are used in science and engineering. For a recent defence of statistical methods in NLP, see [2].

**Text Classification** Text classification is the problem of, given a linguistic unit as input (word, sentence, document), assign a discrete label from a finite set to that input. The classic text classification problem is, given a document as input, assign a label to it which specifies the document's topic. For example, we may wish to classify newspaper reports according to the section of the newspaper they are in, in which case the labels would be *politics, finance, sport, gardening, travel, cookery*, and so on. Another text classification problem is spam detection. Here the inputs are emails, and the output is a single label from the set { *spam, not-spam* }. A final example is sentiment analysis, where the input could be a document, an email, a tweet, or a smaller input such as a sentence or word (in context). The labels in this case would be { *positive, negative* }.

[1] "Every time I fire a linguist ..."
[2] http://cs.jhu.edu/~post/bitext/
[3] http://www.statmt.org/

**Machine Learning Framework** Any machine learning framework relies on the notion of features, which can be thought of as the way that the input is represented mathematically in order to perform the classification. We'll be more precise later about how features are defined mathematically, but for now a useful intuition is to think of features as patterns in the input which we think will be useful for making the classification decision. If the input is a sentence, a feature could be a word in the sentence, or a sequence of words, or a ⟨word, part-of-speech tag⟩ pair, or a more complex pattern such as part of a parse tree. The set of features for a particular input will be represented as a *feature vector*.

**Training and Test Data** Training data is used to train the classifier. We'll mostly be concerned with the *supervised* setting, where each instance of the data also comes with a manually assigned label (for example a set of documents where the correct topic label for each document has been given). *Unsupervised* training is the general problem of finding structure in the input data, often framed as a clustering task, where the cluster labels — or even the number of clusters in the most general case — are not provided in advance.

Finally, there is also *reinforcement learning*, where the training signal is provided in the form of a reward after a number of actions have been taken by the system being trained. One application where reinforcement learning is used in NLP is dialogue modelling. Here, the problem is to learn a dialogue system which knows how to respond to utterances from the user, for example in order to book airline tickets, but the only signal provided during training is whether the required flight was successfully booked, after a number of interactions between the user and system have taken place.

The goal of the training process is to learn a classifier that generalises well to unseen data, i.e. instances not observed in the training set. Hence it is important that the training data and test data — the latter used to evaluate the accuracy of the classifier — are distinct. It is also important that the test data is not used "too often", or more directly to tune the value of a hyperparameter. A separate test set, called a development set, is often held out for this purpose.

**Machine Learning-based Decisions** One way to partition the options available for building a classifier is as follows. First, there are the generative models: *joint* probability distributions over the input and output. Second, there are the discriminative models, *conditional* probability distributions of the output given the input. And third, there is a more general class of discriminative functions, which do not attempt to learn probability distributions, but learn the mapping from inputs to outputs more directly (support vector machines are an example).

We'll encounter all three approaches during the course. All three have advantages and disadvantages, and which one to apply depends on various factors.

**Naive Bayes Classifier** Suppose we are interested in the conditional probability of a class given a document. Rewrite this using Bayes theorem, so that it is proportional to the product of the conditional probability of the document given the class (the *likelihood*) and the marginal probability of the class (the *prior*). The advantage of writing it this way is that there is an easy way to factor the likelihood.

**Bag of Words Model** Suppose we model a document as just a *set* of words, i.e. using a vector of indicator functions, where each component of the vector (a feature) corresponds to a word.[4] Now assume that each word is conditionally independent given the class. Then the conditional probability of the words (the features) given the class is just the product of the probabilities of each word given the class. The advantage of this formulation is that the component probabilities are easy to estimate.

**Probability Estimation** Estimating the prior probability of $P(c_j)$ is easy: just count the number of documents in the training data that have been given label $c_j$ and divide by the total number of documents.

Estimating the likelihood probability of $P(f_i|c_j)$ is equally easy: just count the number of times that word $f_i$ appears in documents assigned class $c_j$ and divide by the number of documents assigned class $c_j$.

These relative frequency estimates can be theoretically justified as *maximum likelihood* estimates; we'll see more on maximum likelihood estimation later in the course.

What if word $f_i$ has not appeared in any documents with class $c_j$? In this case the relative frequency estimate is zero, and the probability for the whole document will be zero (since the zero will propagate through the product). The way to solve this problem is to use a *smoothing* technique, where some of the probability mass is taken from the seen events in the training data and given to unseen events. There is a large literature on how to reassign the mass, since the *sparse data problem* in NLP means that we are often dealing with unseen events at test time. One trivial method is to simply replace each zero count with $\epsilon$, where $\epsilon$ is a small real value greater than zero (being careful to renormalise the relevant distributions so that they sum to 1). We'll see more sophisticated smoothing methods later in the course.

**Sentiment Classification** The paper which began the now huge literature on sentiment analysis was [3]. The models used in the paper are straightforward, as well shall see, but what was noteworthy about the paper was the insight that manually annotated training data could be freely obtained from online movie review sites. For the experiments reported in the paper, they had a dataset consisting of 752 negative reviews and 1,301 positive ones.

**Bag of Words Model** The baseline results show the accuracy that can be achieved by simply counting the number of occurrences of "sentiment bearing" words in a document, where the lists of sentiment bearing words were created

[4]The extension to the multiset, or *bag*, case, where the words are counted rather than simply being present or absent, is straightforward.

manually (in the second case by also inspecting the test data). Machine learning methods can be applied using a similar idea, but this time the positive or negative sentiment of a word is learned automatically from the data.

Two Naive Bayes models were used, one where only the presence or absence of a word is taken into account (the binary model described earlier), and one where the words are also counted. In the latter case the representation of the document as a feature vector is different, since each word *token* in the document is effectively treated as a random variable, with the value being a word from the vocabulary.

**Results** There are two noteworthy aspects of the results. First, the Naive Bayes binary model performs surprisingly well, on a par with the frequency-based model, and almost as good as the more sophisticated models using support vector machines and maximum entropy classifiers. Second, the unigram model which only looks at single words performs surprisingly well, with little value in adding additional feature types such as bigrams or parts-of-speech.

**More Recent Work on Sentiment** Richard Socher's work from Stanford on using (recursive) neural networks for sentiment analysis has gained a lot of interest. However, note that this work also involves a large training set collected using crowd sourcing, so how much is being gained from the fancy neural network is not entirely clear.

### Readings for Today's Lecture

- Chapter 16 of Manning and Schütze's Foundations of Statistical NLP – Text Categorization (perhaps glossing over some of the more technical descriptions of the various models at this stage)

## References

[1] Ken Church. A pendulum swung too far. *Linguistic Issues in Language Technology LiLT*, 2(4), 2007.

[2] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24:8–12, 2009.

[3] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of EMNLP*, pages 79–86, 2002.