Machine learning for natural language processing

Machine Learning for Natural Language Processing

Martin Emms
mtemms@cs.tcd.ie

Saturnino Luz
luzs@cs.tcd.ie

August 15, 2007

Preface

Over the past decade Machine Learning techniques has become an essential tool for Natural Language Processing. This introductory course will cover the basics of Machine Learning and present a selection of widely used algorithms, illustrating them with practical applications to Natural Language Processing. The course will start with a survey of the main concepts in Machine Learning, in terms of the main decisions one needs to make when designing a Machine Learning application, including: type of training (supervised, unsupervised, active learning etc), data representation, choice and representation of target function, choice of learning algorithm. This will be followed by case studies designed to illustrate practical applications of these concepts. Unsupervised learning techniques (clustering) will be described and illustrated through applications to tasks such as thesaurus induction, document class inference, and term extraction for text classification. Supervised learning techniques covering symbolic (e.g. decision trees) and non-symbolic approaches (e.g. probabilistic classifiers, instance-based classifiers, support vector machines) will be presented and case studies on text classification and word sense disambiguation analysed in some detail. Finally, we will address the issue of learning target functions which assign structures to linear sequences covering applications of Hidden Markov Models to part-of-speech tagging and Probabilistic Grammars to parsing of natural language.

Contents

List of Figures

Algorithms

List of Tables

Lecture 1

Introduction

Below we give a brief introduction to machine learning in general and its applications to natural language processing. We also outline the contents of this reader.

1.1 Machine Learning: a brief overview

Machine learning has been studied from a variety of perspectives, sometimes under different names. It is also not uncommon for a machine learning method or technique to have been studied under different names in the fields of artificial intelligence (AI) and statistics, for instance. Although machine learning techniques such as neural nets have been around since the 50's, the term "machine learning", as it is used today, originated within the AI community in the late 70's to designate a number of techniques designed to automate the process of knowledge acquisition. Theoretical developments in computational learning theory and the resurgence of connectionism in the 80's helped consolidate the field, which incorporated elements from information theory, statistics and probabilistic inference, as well as inspiration from a number of disciplines. More recently, as the concept of "agent" being used as a foundation for much AI research, machine learning has been formulated in terms of perception, tasks and performance. Mitchell (1997), for instance, defines *learning* as follows:

> [An agent] is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with E.
>
> (Mitchell, 1997, p. 2)

As Mitchell himself admits, this definition is very general and would encompass processes that very few people would call "learning". An information retrieval system, for instance, whose performance is measured by its ability to find documents matching a given query, can be said to improve as it indexes more documents and therefore would be considered to “learn”, according to the definition above. From a more abstract perspective, there is also a general model-fitting angle on lots machine learning ie. within a space of possible models, the machine finds one which is a good fit for training experiences E, and which can be used to then carry tasks T. While the above definition has connotations of incrementally learning over a long sequence of individual training experience, with performance steadily increasing over this sequence, from the model-fitting angle, all training experiences are considered en-masse. The larger the amount of training data, usually the better the selected model will be. Finally, an even more noncommittal perspective would be to use the term “statistical natural language processing” to denote the sorts of techniques described in this reader. We find, however, that despite its shortcomings, Mitchell’s definition is useful in that it sets out the main elements for the design of machine learning systems in general (and for natural language applications), and suggests an aspect of automation of the learning process, which is central to the AI approach to machine learning.

From an AI perspective, the need for automating the process of knowledge acquisition by agents follows clearly from the dynamic, complex, noisy and often only partially observable nature of most environments, as well as the agent’s own resource limitations. As early symbolic approaches to these problems proved unfruitful, research turned towards ways of handling inference and action choice under uncertainty, and improving the inferential process by automatic means. A typical example of this trend is the way early work on medical expert systems started with attempts to encode medical knowledge as sets of rules and evolved to focusing on identifying the main variables of the inference process and the causal relationships among those variables, at which point statistical information and probabilistic inference could be used to complement human expert knowledge.

Another example from this shift from purely knowledge-based systems to machine learning, this time in the area of natural language processing, is the evolution of text categorisation systems in the last 20 years or so. One of the earliest text categorisation systems was the CONSTRUE system (Hayes and Weinstein, 1990), developed for classification of Reuters news articles into predefined categories[1]. CONSTRUE employed a set of production rules. If a document matched the antecedent of a rule, expressed as a disjunction of conjunctive clauses whose literals indicate whether a term occurs or not in the document, then that document would be filled under the category specified in the consequent. These rules were hand-coded by experts and the developers reported very good results (up to 90% break-even precision-recall scores) for a non-standard subset of the Reuters corpus. In spite of that, hand-crafted text categorisation have all but disappeared, replaced by

[1] More on the Reuters corpus in chapter 2.

approaches based on machine learning. The difference between the purely knowledge-based and machine learning approaches is that the former seeks to build a system for text categorisation while the latter focuses on creating systems which can themselves generate text categorisation systems. For applications such as text categorisation where large volumes of unstructured data need to be processed and those data are constantly changing (as is the case of newsfeed texts), a manual approach to rule building is impractical and a good case can be made for machine learning approaches.

1.1.1 Machine Learning and Natural Language Processing

There are many applications in which machine learning techniques have been used, including

- speech recognition
- document categorisation
- document segmentation
- part-of-speech tagging, word-sense disambiguation
- named entity recognition (selecting and classifying multi-word sequences as instances of semantic categories)
- parsing
- machine translation

1.2 Using Machine Learning

Main decisions in designing a machine learning system:

- How will the system access an use data; That is: how will the system be exposed to its training experience?
- What exactly should be learned? What is the target function?
- How shall it be represented? How do we define a function to approximate the target function?
- What specific algorithm should be used to learn this approximation?

1.2.1 Data

How will the system access its training experience? It can do it *directly*, if the learning agent is situated in the world and has control over the data it samples and the actions it takes, as in reinforcement learning (Sutton and Barto, 1998). Or it can do it indirectly, via an existing record of past experiences (e.g. a corpus), as in most applications we will describe in this reader.

Another distinction concerns how the data source encodes the function to be approximated. From that perspective, we have *supervised learning*, where the target function is completely specified by the training data (learning experience), *unsupervised learning*, where the system will try to uncover patterns in data which contain no explicit description of the target concept, and techniques situated in between, such as semi-supervised learning (which use the two kinds of data in the learning process) and active learning (in which the learner gets to choose which examples should be labelled at each learning iteration).

1.2.2 Target function

The format of the target function determines a mapping from data to the concept to be learnt. In supervised learning settings, the target function is assumed to be specified through annotation of training data or some form of feedback. Target function definitions that arise in such settings include, for instance: a corpus of words annotated for word senses of specified by a function of the form $f : W \times S \to \{0, 1\}$ where $W \times S$ are word-sense pairs, a database of medical data specifying a mapping from lists of symptoms to diseases, binary user feedback in spam filtering, assessment of outcomes of actions by a situated situated agent, etc. In unsupervised settings explict specification of a target function might not be needed, though the operation of the learning algorithm can usually be charaterised in terms of model-fitting, as discussed above.

1.2.3 Representing hypotheses and data

The goal of the learning algorithm is to induce an approximation $\hat{f}$ of a target function f. The inductive task can be conceptualised as a search for a hypothesis (or model) among a large space of hypotheses which fits the data and sample of the target function available to the learner. The choice of representation for this approximation often constrains the search space (the number of hypotheses).

In addition, the data used in the induction process needs to be represented uniformly. Examples of data representation schemes include representation of text as “bags of words”, Boolean vectors, etc. A hypothesis (the approximation to be learnt) could, for instance, map conjunctions of

Boolean literals to categories, thus exhibiting an *inductive bias* towards the sorts of concepts representable by this choice of representation.

1.2.4 Choosing the learning algorithm

The choice of learning algorithm is conditioned to the choice of representation. Since the target function is not completely accessible to the learner, the algorithm needs to operate under the *inductive learning assumption* that:

> an approximation that performs well over a sufficiently large set
> of instances will perform well on unseen data

How large is a "sufficiently large" set of instances and how "well" the approximation of the target function needs to perform over the set of training instances are questions studied by computational learning theory.

1.3 An example

Mitchell (1997, ch. 2) gives an interesting example illustrating the design questions described above: the design of a system that learns to play draughts (checkers). A correct specification for a checkers playing evaluation would be:

- $V : Board \rightarrow \mathbb{R}$
- if b is a final board state that is won, then $V(b) = 100$
- if b is a final board state that is lost, then $V(b) = -100$
- if b is a final board state that is drawn, then $V(b) = 0$
- if b is a not a final state in the game, then $V(b) = V(b')$, where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game.

This gives correct values, but is not operational.
How about approximating it via a polynomial function of board features?

$$\begin{aligned} \hat{f}(b) &= w_0 + w_1 \times bp(b) + w_2 \times rp(b) + w_3 \times bk(b) + \\ & \quad w_4 \times rk(b) + w_5 \times bt(b) + w_6 \times rt(b) \end{aligned} \tag{1.1}$$

where

- $bp(b)$: number of black pieces on board b
- $rp(b)$: number of red pieces on b

Algorithm 1.1: Least Means Square

```
LMS($c$: learning rate)
 for each training instance $< b, f_{train}(b) >$
   do
     compute $error(b)$ for current approximation
     (using current weigths):
           $error(b) = f_{train}(b) - \hat{f}(b)$
     for each board feature $t_i \in \{bp(b), rp(b), \ldots\}$,
       do
         update weight $w_i$:
           $w_i \leftarrow w_i + c \times t_i \times error(b)$
       done
   done
```

- $bk(b)$: number of black kings on b
- $rk(b)$: number of red kings on b
- $bt(b)$: number of red pieces threatened by black (i.e., which can be taken on black's next turn)
- $rt(b)$: number of black pieces threatened by red

The problem now is how to estimate training values for the function above. A simple rule for estimating training values is:

- $f_{train}(b) \leftarrow \hat{f}(Successor(b))$

Given this rule, one could use the *least means square* (LMS) algorithm (1.1) to learn the weights that best fit the training instances. Note that c is should be a small value, say 0.1, which is used to moderate the rate of learning (i.e. to prevent the algorithm from overstepping the minimum). The algorithm implements gradient descent search and will converge (if certain conditions are met), though possibly very slowly.

1.4 This reader

Many instances of machine-learning in NLP can be seen as instances of the supervised learning of a classifier, whereby from a corpus of classified training instances, a classifier is learnt which assigns categories to unclassified fresh instances. This is the general topic of Chapter 2. A number of techniques are described, including Naïve Bayes, decision trees, nearest-neighbour classifiers and support vector machines. Some of these techniques are further illustrated by considering their use in the tasks of document classification (Naïve Bayes, decision trees) and word sense disambiguation (decision trees, nearest-neighbour).

Unsupervised category learning is the topic of Chapter 3, whereby some kind of categorisation structure is assigned to a corpus of unclassified instances. Hierarchical and partitional clustering are discussed, along with applications to document and keyword clustering, dimensionality reduction and word sense disambiguation.

While Chapters 2 and 3 are concerned with the categorisation of instances, Chapters 4 and 5 are concerned with mapping an input sequence to some kind of complex object: the space from which target complex objects are drawn is infinite and the size of the target objects will generally be a function of the input sequence. Part-of-speech tagging, and natural language parsing are instances of this.

Chapter 4 considers the case where the target complex object is again a sequence. Hidden Markov Models are presented as a technique deployable in this scenario of sequence-to-sequence mappings. The Expectation-Maximisation approach to the unsupervised learning of HMMs is discussed, as well as the supervised learning of HMMs, with particular emphasis given to their application to part-of-speech tagging.

Chapter 5 considers natural language parsing, where the mapping to be learned is from a sequence to a syntax tree. PCFGs are discussed, along with Expectation-Maximisation approaches to their unsupervised learning. Models more complex than PCFGs are considered and their supervised learning.

Lecture 2

Supervised learning and applications

Supervised learning is possibly the type of machine learning method most widely used in natural language processing applications. In supervised learning, the inductive process whereby an approximation $\hat{f}$ is built is helped by the fact that the learner has access to values of the target function f for a certain number of instances. These instances are referred to as the *training set* and their values are usually defined through manual annotation. The feasibility of obtaining a large enough number of instances representative of the target function determines whether supervised learning is an appropriate choice of machine learning method for an application. In this lecture we survey some supervised machine techniques and illustrate them by detailing their use in the categorisation of text.

2.1 Overview of concepts and methods

Supervised learning methods are usually employed in learning of classification tasks. Given an unseen data instance, the learnt function will attempt to classify it into one or more target categories. The different ways in which learning and classification can be implemented determine a number of distinctions within the area of supervised learning as well as the nature of the various techniques employed. In order to characterise these distinctions more precisely, we introduce some notation and state some simplifying assumptions. We start by assuming that all data instances that take part in the learning and classification processes are drawn from a set $\mathcal{D} = \{d_1, \ldots, d_{|\mathcal{D}|}\}$. These instances are represented as *feature vectors* $\vec{d_i} = \langle t_1, \ldots, t_n \rangle$ whose values $t_1, \ldots, t_n \in \mathcal{T}$ will vary depending on the data representation scheme chosen. Instances will be classified with respect to categories (or classes) in a category set $\mathcal{C}$. Therefore, in its most general form, the target function will have signature $f : \mathcal{D} \rightarrow 2^{\mathcal{C}}$. In most cases, where classes are defined manually by human annotators, this function is an idealisation which abstracts away issues of consistency and inter-annotator agreement. That is the case of the applications presented in this lecture. The general formulation given above describes a *multi-label classification* task. In practice, however, classification tasks are often implemented as collections of *single-label classifiers*, each specialised in binary classification with respect to a single class. Each of these classifiers can thus be represented as a binary-valued function $\hat{f}_c : \mathcal{D} \times \mathcal{C} \rightarrow \{0, 1\}$ for a target category $c \in \mathcal{C}$, where

$$\hat{f}(d, c) = \begin{cases} 1 & \text{if } d \text{ belongs to class } c \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Assuming that the categories in $\mathcal{C}$ are all stochastically independent of each other (an assumption often found to be incorrect, but harmless in practice), enables us to express a multi-label classification task as a composition of single-label classifiers by making $\hat{f}(d) = \{c \mid \hat{f}_c(d) = 1\}$.

The classification functions defined above can be regarded as implementing *hard classification* decisions. In many cases, however, the primary output of a classifier is a numerical score, usually expressing the probability that an instance belongs to the target class or, equivalently, the ranking of various classes with respect to an instance. In such cases, the classification function is better defined as $\hat{f} : \mathcal{D} \times \mathcal{C} \rightarrow [0, 1]$, where instance-class pairs are mapped to a real number in the $[0, 1]$ interval. Although ranking classification functions in this form can be useful for indicating the degree of confidence the classifier has in a particular classification decision, the output of ranking classifiers is usually mapped to hard classification functions through *thresholding*: a value is set above which an instance is assumed to belong to a target category. Defining threshold levels can be something of an art, and the strategy employed tends to vary from application to application and also across domains. Situations in which a high cost is associated with a mistaken classification decision, as in, for instance, when a legitimate e-mail message is classified as spam, may justify the use of higher threshold levels. However, as we will see below, a number of techniques exist which can be employed in order to determine reasonable threshold levels for most situations automatically.

Inducing a classification function $\hat{f}$ through supervised learning involves a *train-and-test* strategy. First, the annotated data set $\mathcal{D}$ is split into a *training set*, D_t, and a *test set*, D_e. Part of the training data is sometimes used for parameter tuning. These data can be identified as a *validation set*, D_v. The train-and-test strategy for classifier building can be summarised as follows:

1. an initial classifier $\hat{f}$ is induced from the data in the training set D_t

2. the parameters of the classifier are tuned by repeated tests against the validation set D_v
3. the effectiveness of the classifier is assessed by running it on the test set D_e and comparing classification performance of $\hat{f}$ to the target function f

It is important that D_t and D_e are disjoint sets. That is, no data used in training should be used for testing. Otherwise, the classifier's performance will appear to be better than it actually is. A related problem, which one can also address by iterating through the validation set, is that of *overfitting*. Overfitting refers to the inability of a classifier to generalise to unseen data. If overfitting occurs the classifier will perform very well on the data on which it was trained but poorly under operational conditions. Keeping training and test data separate help identify and tackle overfitting in the classifier development phase. Availability of enough training data is crucial in order to avoid overfitting. However, in many areas, natural language processing in particular, data sparsity is a serious problem. A technique frequently used in machine learning to mitigate the effects of data sparsity is *k-fold cross validation* (Mitchell, 1997). In this approach, k different classifiers $\hat{f}_1, ..., \hat{f}_k$ are built by partitioning the corpus into k disjoint sets $\mathcal{D}_1, ..., \mathcal{D}_k$. The train-and-test procedure is then applied iteratively on pairs $\langle D_t^i, D_e^i \rangle$ of training and test partitions, where $D_t^i = \mathcal{D} \setminus \mathcal{D}_i$ and $D_e^i = \mathcal{D}_i$ for $1 \leq i \leq k$, and the overall effectiveness result is computed by testing the effectiveness for each classifier $\hat{f}_1, ..., \hat{f}_k$ and averaging the results.

Another factor which has a bearing on data sparsity issues is the choice of data representation. The more detailed the representation, the more data is needed to guarantee effective learning of the target function. Too much detail and one risks overfitting, too little detail and features which could prove useful indicators of category membership might be lost. The question of how the data should be represented is therefore one of finding the right amount of detail to include given the amount of data available. As we will see, this question involves no only deciding on how linguistic features should be translated into uniform feature vectors, but also which features should be considered in the first place. The latter is known as the *dimensionality reduction* problem.

2.1.1 Learning and Implementing classification functions

Further distinctions can be established in terms of how the classification function is learnt and implemented which essentially define the major groups of machine learning techniques. With respect to how $\hat{f}$ is implemented, two main types of methods have been proposed: symbolic and numeric. Each method has its particular advantages and choosing between them is often guided by application and domain dependent considerations.

Numeric methods implement classification indirectly. The classification function $\hat{f}$ outputs a numerical score based on which the hard classification decision is finally made. The paradigmatic representatives of this group of machine learning methods are probabilistic classifiers. The output of a probabilistic classifier is an estimation of the conditional probability $P(c|\vec{d}) = \hat{f}(d, c)$ that an instance represented as $\vec{d}$ should be classified as c. However, since each element of vector $\vec{d}$ is regarded as a random variable T_i $(1 \leq i \leq |\mathcal{T}|)$, generating the approximation $\hat{f}(d, c)$ would typically involve estimating conditional probabilities for all possible representations of instances in the feature space $\mathcal{T}$, i.e. $P(c|T_i, \ldots, T_n)$. This is prohibitively costly, in terms of the amount of data and computational resources needed, for most practical applications. For the discrete case and n possible nominal attribute values the modelling space is $O(n^{\mathcal{T}})$. Assumptions about the nature of the conditional probabilities to be estimated help alleviate such complexity issues. A commonly used strategy in probabilistic reasoning is to encode conditional independence assumptions explicitly in the system (Pearl, 2000). An extreme version of this strategy that is commonly employed in classification is to assume that all features in the feature vector of a training instance are independent of each other given the target category:

$$P(\vec{d}|c) = \prod_{k=1}^{|\mathcal{T}|} P(t_k|c) \tag{2.2}$$

Classifiers implementing this assumption are know as Naïve Bayes (NB) classifiers. Since, according to Bayes' rule, the classification probability that interests us can be expressed in terms of the conditional probability given by (2.2) as

$$P(c|\vec{d_j}) = \frac{P(c)P(\vec{d_j}|c)}{P(\vec{d_j})} \qquad \text{(Bayes' rule)} \tag{2.3}$$

an NB learner can build an approximation of the target classification function by estimating prior probabilities of the target categories and conditional probabilities for each feature given the various possible category values. Assuming the category data to be described by a Boolean random variable C, a straightforward criterion for hard classification would be to choose the value of C that maximises the conditional probability. This (i.e. that the c that maximises (2.3) is the class to which $\vec{d_j}$ belongs) is also known as the *maximum a posteriori hypothesis.* If we regard all classes as equiprobable, the choice of the class that maximises $P(\vec{d_j}|c)$ as the class of $\vec{d_j}$ is known as *maximum likelihood hypothesis.*

Different variants of NB classifiers exist which can deal with data discrete- and real-valued data representation schemes. For discrete data representation schemes, two related but essentially different approaches have been used (McCallum and Nigam, 1998): *multi-variate Bernoulli models*, in which features are modelled as Boolean random variables, and *multinomial models* where the variables represent count data. In both cases, the conditional probabilities for the discrete attributes $P(T_i = t|c)$ are modelled as the probability that attribute T_i will have value t when the class is c. For numeric data representation schemes, each attribute is represented by a continuous probability distribution, usually modelled as a normal distribution, though methods that employ non-parametric kernel density estimation has also been proposed (John and Langley, 1995). If a normal distribution is assumed, the conditional probability estimates will be

$$P(T_i = t|c) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{2.4}$$

where μ and σ^2 are the mean and variance of the values taken by the attribute in the data[1]

Other numeric classification methods include various kinds of linear classifiers, such as logistic regression, the perceptron classifier and support-vector machines[2]. These techniques have been widely used in natural language applications, and found to be particularly useful in text classification tasks (Mitchell, 1997; Sebastiani, 2002).

Symbolic methods usually implement hard classification directly, making no use of thresholding techniques. Data instances are usually represented as vectors of discrete (often binary) values and classification is performed through testing of selected subsets of the feature set $\mathcal{T}$. Decision trees (Quinlan, 1986), decision lists and decision rules (Apté et al., 1994) are the best known representatives of symbolic classification methods. Classification in these methods is performed through binary tests on highly discriminative features. In decision trees the sequence of tests is encoded as a tree structure where the leaf nodes are labelled with the target categories, inner nodes represent features to be tested, and edges are labelled with the outcomes of these tests. Given a decision tree, the categorisation of an instance is done by following a labelled path along the tree according to the values in the vector representation of the instance to be classified until a leaf node is reached.

Figure 2.1, generated by applying the decision tree learning algorithm ID3[3] to the sample data set used by Mitchell (1997), illustrates a deci-

[1] For a continuous random variables T_i, the probability density function is actually defined over intervals of values as $p(t \leq T_i \leq t + \delta) = \int_t^{t+\delta} f(t)dt$. Thus, strictly speaking $P(T_i = t|c) = \int_t^t f(t)dt = 0$. The conditional set out in equation (2.4) in fact stands for the probability that the value of T_i lies in the small interval between t and $t + \delta$ as δ tends to 0.

[2] NB can also be regarded as a linear classifier if the features in the data representation vectors are modelled as discrete-valued random variables.

[3] We used the WEKA ("Waikato Environment for Knowledge Analysis") software to generate this and other examples in this reader. More information on WEKA and its free software (GPL) distribution can be found at `http://www.cs.waikato.ac.nz/ml/weka`.

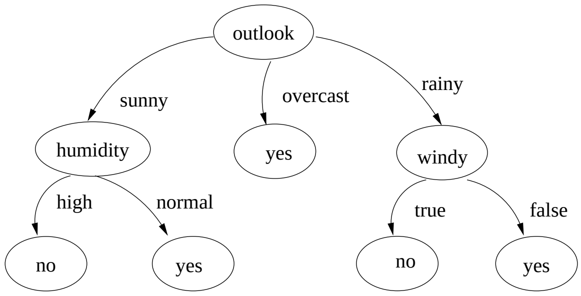


Figure 2.1: Sample decision tree: "tennis weather" task

sion tree for deciding whether to play tennis or not, based on the record of weather conditions and decisions shown in Table 2.1. One of the main advantages of symbolic classifiers in general is that their classification decisions can be easily interpreted by humans. Decision trees, for example, can be interpreted as a set of production rules (e.g. $outlook = outcast \rightarrow play = yes$, $outlook = sunny \wedge humidity = normal \rightarrow play = yes$, ...) and, in fact, the C4.5 algorithm for decision tree learning employs a conversion to rule sets in order to prune over-specific trees (Quinlan, 1993). Other methods induce rules directly from the training data.

Table 2.1: The "play tennis" data set

	outlook	temperature	humidity	windy	play
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

The ID3 decision tree learning algorithm employs a "divide and conquer" strategy. It begins by considering all instances in the the training set D_t. If all instances in D_t belong to the same class, then no further learning is possible (the class to be assigned will be, obviously, the class to which all instances belong). The most likely situation, however, is that the in-

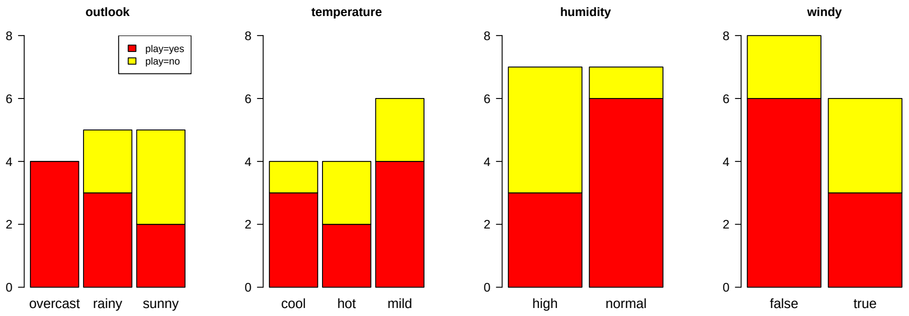


Figure 2.2: Distribution of attributes for the "tennis weather" task

stances in D_t are in different classes. In this case, the learner will attempt to select a feature t_k whose values partition D_t into subsets with as uniform a distribution of target categories as possible. Each subset will form the root of a new branch. This process recurs on each subtree until each leaf contains only training instances of the same class. Figure 2.2 shows the distribution of feature values for the tennis data set (Table 2.1). Notice how feature *outlook* partitions the training set so that one of the partitions (i.e. $\{d \mid t^d_{outlook} = overcast\}$) contains a single classification decision (i.e. $play = yes$). At the other extreme, we have feature *temperature*, whose values are more uniformly distributed with respect to the target decision, and therefore has little discriminative power.

Decision trees have been used in natural language procssing for parsing (Haruno et al., 1999; Magerman, 1995), text categorisation (Lewis and Ringuette, 1994), word-sense disambiguation, tagging and many other applications.

Another type of supervised machine learning technique frequently used in natural language applications is instance-based learning (Aha et al., 1991). The majority of instance-based learners are "lazy learners", in the sense that they simply store the instances in the training set, and classify new instances by retrieving similar instances from this stored instance- (or case-) base and analysing the classes of the retrieved instances. By deferring learning until a new instance is presented for classification, instance-based classifiers can approximate global complex target functions through simpler local approximations. Approaches to instance-based learning vary in the complexity of their data representation schemes and the metric used in retrieving similar instances at classification time (Mitchell, 1997). These range from simple k-nearest neighbour (kNN) classifiers, in which instances are represented as points in the Euclidean space, to case-based reasoning, where instances are described in terms of complex attributes which demand more sophisticated mechanisms for case retrieval. However, even a simple instance-based meth-

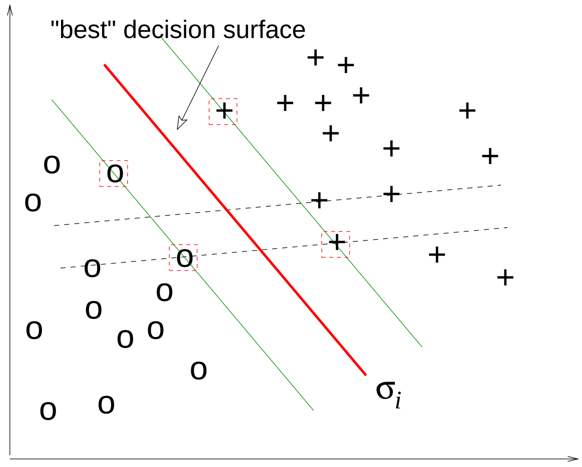


Figure 2.3: Support vectors

ods like kNN has been found to be useful in natural language applications such as text categorisation, where kNN ranks among the best performing techniques (Yang and Chute, 1994; Sebastiani, 2002), probability estimation for statistical parsing (Hogan, 2007), tagging (Daelemans et al., 1996), information extraction (Zavrel and Daelemans, 2003), and speech synthesis (Daelemans and van den Bosch, 1996).

A method which has gained much popularity in the natural language processing community in recent years is Support Vector Machines (SVM). Unlike methods such as kNN, SVMs scale well to high dimensional spaces while remaining robust to overfitting (Joachims, 1998; Sebastiani, 2002). SVMs can be explained in geometrical terms as consisting of decision surfaces, or planes $\sigma_1, \ldots, \sigma_n$ in a $|\mathcal{T}|$-dimensional space which separates positive from negative training examples. Given $\sigma_1, \sigma_2, \ldots$, the SVM will find a σ_i which separates positive from negative examples by the widest possible margin. A 2-d plane on which positive and negative instances are linearly separable is shown in Figure 2.3. The instances defining the best decision surfaces form part of the support vectors. SVMs are also effective in cases where positive and negative training instances are not linearly separable. In such cases, the original feature space is projected, through application of a (non-linear) kernel function to the original data, onto a higher dimensional space in which a maximum-margin hyperplane will separate the training data. Joachims (1998) presents an application of SVM to text categorisation, a task for which the method's ability to handle large feature sets seems to be particularly useful. SVMs have also been used in natural language processing tasks with more complex data modelling requirements, such as tagging and parsing (Collins and Duffy, 2002).

Finally, there are *meta-classification* methods which combine different types of supervised classifiers $\hat{f}_1, \hat{f}_2, \ldots$ into classifier ensembles. The results of the various independent classifiers form the basis for the final classification decision. Ensembles are characterised according to the kinds of classifiers $\hat{f}_i$ they employ and the way they combine multiple classifier outputs. Ideally, these classifiers should be as diverse as possible, in terms of how they represent data, and how they model their approximations of the target function, along the lines explained above. Strategies for combining outputs also vary. Simple strategies such as majority voting and weighted linear combination have been used for committees of (hard) binary classifiers and soft classifiers, respectively. More complex strategies such as dynamic classifier selection and adaptive classifier combination have also been proposed which take performance of individual classifiers on the validation set into account in weighting the final decision. A popular way of creating classifier ensembles is the boosting method (Schapire, 1990). In this method, classifiers are trained sequentially, rather than independently. Thus, the training of classifier $\hat{f}_i$ takes into account the performance of its predecessors $\hat{f}_1, \ldots, \hat{f}_{i-1}$. The committee is used as a way of guiding resampling at each iteration so that the most informative instances are chosen. Polikar (2006) presents a comprehensive tutorial introduction to classifier ensembles for machine learning in general. Ensembles have been applied to a variety of natural language processing applications, including parsing (Collins and Koo, 2005) word-sense disambiguation (Pedersen, 2000; Escudero et al., 2000) and text categorisation (Sebastiani, 2002).

In the following sections we illustrate some representatives of the machine learning techniques described above through detailed descriptions of how various algorithms can be used in a supervised text categorisation task.

2.2 Text categorisation

Text categorisation is a task which consists of assigning pre-defined symbolic labels (categories) to natural language texts. It is a part of a number of distinct tasks which involve some form of "categorisation". These include automatic identification of a set of categories for a given corpus. The latter involve discovery of "natural groupings" of texts and is usually handled by unsupervised (clustering) methods, which will be described in lecture 3. The task we will analyse in this section is essentially a classification task which can be viewed from two distinct but logically (though not always operationally) equivalent perspectives, with respect to whether sets of labels get assigned to instances or the other way around. The former corresponds to what has been termed (Sebastiani, 2002) *category-pivoted categorisation*: given a category c, the classifier must find all text documents d that should be filed under c. The latter can be described as *document-pivoted categorisation*: given a text document d, the classifier must find all categories c

under which d should be filed. The choice between these perspectives is mostly pragmatic. Document-pivoted categorisation is appropriate when the complete set of documents is not available for batch processing, as in email filtering for interactive applications and user interfaces. Category-pivoted categorisation is suitable for applications in which the category set (labels) change after a large, fairly stable corpus has been categorised, as in classification of documents under taxonomies or web hierarchies.

As with many classification tasks, text categorisation was initially approached from a knowledge engineering perspective. A landmark development in this area was the CONSTRUE system (Hayes and Weinstein, 1990) which employed a large set of human-generated rules to classify news articles produced by Reuters. The system was tested in a subset of an early version of the Reuters corpus, Reuters-22173[4] and found to produce very good results. The problem with this approach, however, is that as new data gets added and the nature of the content and categories change, the knowledge base needs to be updated. Manual production of rules is a time-consuming and expensive process which in this and similar cases can easily lead to a so-called *knowledge acquisition bottleneck*. Machine learning techniques were seen in early rule-based AI systems a way of getting around the knowledge acquisition bottleneck. Text categorisation was no exception, and the last decade saw a great number of text categorisation research projects approached through machine learning (Sebastiani, 2002). Virtually all current research on text categorisation focus on the issue of how functions that map texts to categories can be induced from large volumes of annotated data[5].

The conceptualisation of the text categorisation task in terms of the general ideas introduced in section 2.1 should be fairly obvious at this point. Ignoring (for the time being) data representation issues, and assuming a document-pivoted categorisation perspective, we can regard textual "documents" as our data instances d_i. Accordingly, a corpus of documents will be represented as a set $\mathcal{D}$ within which we will distinguish subsets for training (D_t), testing (D_e) and validation (D_v). Document labels correspond to the category set $\mathcal{C}$. In the Reuters corpus, for instance, some of the elements of this set would be "acq" (labelling documents about "acquisitions"), "housing", etc. REUTERS-21578 contains a total of 445 such categories, 33% of which classify at least 20 news stories (documents). A sample Reuters document is shown in Figure 2.4.

[4] An improved version of that collection, REUTERS-21578 (Lewis, 1997), became one of the standard benchmarks in the area of text categorisation. However, this collection has now been largely superseded by the Reuters Corpus Volume I, or RCV1 (Lewis et al., 2004). The original REUTERS-21578 corpus is encoded in SGML. An XML-encoded version of REUTERS-21578 is available at `http://ronaldo.cs.tcd.ie/esslli07/data/`.

[5] The SpamAssassin e-mail spam filtering system can be regarded as a knowledge-based text categorisation system currently in operation. Its latest version, however, includes machine learning components, such as an NB classification module and a neural net for scoring, in addition to a set of rules.

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" ID="96">
 <DATE>26-FEB-1987 16:32:37.30</DATE>
 <TOPICS><D>acq</D></TOPICS>
 <PLACES><D>usa</D></PLACES>
 <PEOPLE></PEOPLE>
 <ORGS></ORGS>
 <EXCHANGES></EXCHANGES>
 <COMPANIES></COMPANIES>
 <TEXT>
  <TITLE>INVESTMENT FIRMS CUT CYCLOPS &lt;CYL> STAKE</TITLE>
  <DATELINE> WASHINGTON, Feb 26 - </DATELINE>
  <BODY>A group of affiliated New York investment firms said they
   lowered their stake in Cyclops Corp to 260,500 shares, or 6.4 pct
   of the total outstanding common stock, from 370,500 shares, or
   9.2 pct. In a filing with the Securities and Exchange Commission,
   the group, led by Mutual Shares Corp, said it sold 110,000 Cyclops
   common shares on Feb 17 and 19 for 10.0 mln dlrs.  Reuter
  </BODY>
 </TEXT>
</REUTERS>
```

Figure 2.4: Sample REUTERS-21578 text

Text categorisation tasks vary with respect to number of categories assigned to a single document. Multi-label classification applies to cases such as Reuters news categorisation (note, for instance, that the text in Figure 2.4 is labelled as both *acq* and *usa*), controlled vocabulary keyword assignment, document classification into web hierarchies etc. Email spam filtering, on the other hand, is a typical example of binary text categorisation. For simplicity we will recast multi-label categorisation as individual binary classification tasks. For the Reuters case, for instance, the learner will learn separate approximations (e.g. $\hat{f}_{acq}, \hat{f}_{usa}, \dots$) for each category so that a hard categorisation decision will be expressed as a mapping $\hat{f}_c : \mathcal{D} \rightarrow \{0, 1\}$ where c is the target category.

2.2.1 Building a text classifier

The target function $\hat{f}_c$ to be learnt for text categorisation with respect to category c, is described by the labels assigned (by human annotators) to the documents in the corpus. The process of building a text categorisation system consists of a number of phases. Initially, the data need to be encoded uniformly so that they can be interpreted by the learning algorithm and categorisation function. This might typically involve preprocessing the document meta-data (in the Reuters corpus, for instance, that would mean extracting category information and document content from the SGML or XML encoding as well as all the mark-up that might be of interest) tokenising the textual content, removing stop-words, stemming, recording frequency tables and building inverted indices. The next step is to induce the classifier, usually through a number of training and validation iterations (i.e. k-fold cross validation, as described above) in order to tune the system's parameters. Finally, testing and evaluation are performed, again, using cross validation in order to improve the reliability of results.

2.2.2 Representing text

As seen above, text needs to be converted to a uniform vector representation that can be used by the induction and the categorisation algorithms. Performing that sort of conversion involves deciding on the nature and size of the feature set $\mathcal{T}$. There is usually a trade-off between these two factors. With respect to the nature of the elements of $\mathcal{T}$, one could consider various alternatives, such as: individual words (or *types*, to use corpus linguistics terminology), word sequences (e.g. word digrams, trigrams etc) or more complex "semantic" representations (Sebastiani, 2002). Representing word sequences can have certain advantages, specially if some knowledge of what constitutes meaningful units is judiciously applied. Take the sequence "White House", for instance. In the absence of positional information, this sequence is likely to be more informative of the content of certain kinds of texts as a single feature than its individual component types. Admitting sequences as features, on the other hand, greatly increases the size of the feature set, since every combination of features needs to be accounted for. This usually means that the classifier induction process would require a prohibitively large training set.

In practice, most systems adopt representations based on single words, additionally reducing the size of $\mathcal{T}$ by performing stemming and stop-word removal on the set of types identified by the tokeniser. It should also be pointed out that tokenisation itself presents decisions which might affect the performance of the text categorisation system. Factors worth considering include whether to distinguish between upper and lower case characters (as the "White House" example above might suggest), how to treat hyphenated words, and what other less conventional sequences of symbols to accept[6].

For the remaining of this chapter we will assume that documents will be represented as vectors containing values assigned to single single units, say, word stems. Each value is mapped to a particular feature, defined by the value's position in the vector. These values are usually a function of the number of occurrences of the feature in the text to be represented. This paragraph could be represented (for example) as the vector $\vec{d} = \langle 3, 2, 4, \dots \rangle$, where the ordering of the features is $\langle for, remain, represent, \dots \rangle$ and their values are counts of the number of occurrences of each word form in the

[6] In the domain of email spam filtering, Graham (2002) suggests that the "word" #FF0000 (the HTML code for the colour red) is one of the most discriminative features for certain types of messages.

text.

2.2.3 Dimensionality issues

The size of the vectors that represent the data, i.e. $|\mathcal{T}|$, is referred to as their *dimensionality*. If these values correspond to real numbers, the representation encodes vectors on a $|\mathcal{T}|$-dimensional space. In text categorisation, it is usually the case that even after stop-word filtering and stemming one is still left with a very large feature set. This situation is undesirable for two reasons. The first is the high computational cost of inducing a classifier for high-dimensional data. The second and most important reason is the negative effect high dimensionality usually has on the effectiveness of the induced classifier. Very large feature sets tend to contain many contingent features along with the features on which the categorisation is actually based. Most machine learning methods perform poorly in the presence of large numbers of contingent features and one risks overfitting if such features are allowed to remain in the representation. *Dimensionality reduction* techniques are techniques aimed at reducing the size of the feature set, either by (i) selecting from the original set $\mathcal{T}$ a much smaller subset $\mathcal{T}_s$ consisting of the most relevant features of $\mathcal{T}$, or (ii) creating a new set of features $\mathcal{T}_x$, which is also much smaller than $\mathcal{T}$ but whose elements are essentially different from those in the original set. The first of these types of techniques is called *feature selection* (or *term selection*, in the case of text categorisation) and the second is called *feature extraction* (Sebastiani, 2002). Feature extraction is usually done through unsupervised learning and will be described in detail in lecture 3. Feature selection employs supervised methods and is described below.

Dimensionality reduction can be done for each category separately, or *locally*, or for the whole set of categories, or *globally*. In local dimensionality reduction different term sets are produced for different categories. Intuitively, a word such as "merger" would seem to be a good feature for a Boolean classifier for REUTERS-21578 category `acq` (acquisitions) but perhaps not so useful for a classifier targeting, say, category `vatican`[7].

In global dimensionality reduction, the same feature set is used for all classifiers. A simple but effective way of performing global dimensionality reduction is to rank all features t_k in $\mathcal{T}$ according to the number of training documents in which they occur ($\#_{Tr}(t_k)$) and pick the top n features. An empirical study by (Yang and Pedersen, 1997) has shown that, in spite of its simplicity, this strategy can reduce dimensionality up to a factor of 10 with no loss in categorisation effectiveness. This is due to the fact that most words occur in a corpus with very low frequency (Zipf's law) and therefore have little discriminative power across the range of categories.

[7] In fact, despite being listed in Lewis' distribution, the category `vatican` doesn't actually appear in any of the REUTERS-21578 documents.

This general strategy of ranking features according to a numerical scores is also employed in more sophisticated local approaches. A score $r_i(t_k)$ is assigned to each feature t_k with respect to category c_i which reflects the relevance or usefulness of t_j as an indicator of c_i. When needed, local scores can be combined in order to produce a single (global) ranking. The combined global score can be taken to be a sum of local scores over the set of all categories: $r_{sum}(t_k) = \sum_{i=1}^{|\mathcal{C}|} r_i(t_k)$, an average of local scores weighted by category generality: $r_{wavg}(t_k) = \sum_{i=1}^{|\mathcal{C}|} P(c_i)r_i(t_k)$, or the maximum local value $r_{max}(t_k) = max_{i=1}^{|\mathcal{C}|} r_i(t_k)$.

Table 2.2: Functions commonly used in feature selection for text categorisation (Sebastiani, 2002).

Name	Definition
Document frequency	$\#(t, c) = P(t\|c)$
DIA factor	$z(t, c) = P(c\|t)$
Expected mutual information	$I(t, c) = \sum_{c' \in \{c, \bar{c}\}} \sum_{t' \in \{t, \bar{t}\}} P(t', c') \log \frac{P(t', c')}{P(t')P(c')}$
Mutual information	$MI(t, c) = P(t, c) \log \frac{P(t, c)}{P(t)P(c)}$
Chi-square	$\chi^2(t, c) = \frac{\|D_t\|[P(t, c)P(\bar{t}, \bar{c}) - P(t, \bar{c})P(\bar{t}, c)]^2}{P(t)P(\bar{t})P(c)P(\bar{c})}$
NLG coefficient	$NLG(t, c) = \frac{\sqrt{\|D_t\|}[P(t, c)P(\bar{t}, \bar{c}) - P(t, \bar{c})P(\bar{t}, c)]}{\sqrt{P(t)P(\bar{t})P(c)P(\bar{c})}}$
Odds ratio	$OR(t, c) = \frac{P(t\|c)[1 - P(t\|\bar{c})]}{[1 - P(t\|c)]P(t\|\bar{c})}$
GSS coefficient	$GSS(t, c) = P(t, c)P(\bar{t}, \bar{c}) - P(t, \bar{c})P(\bar{t}, c)$

Before proceeding to specifying some of the functions proposed as ranking functions in the machine learning literature, we will make another assumption. We will assume a multi-variate Bernoulli model as the base representation for documents, so that a feature will have value 0 or 1 depending on whether it occurs in a document. In other words, documents will be represented as Boolean vectors. The events in this probability model will be represented by as sets of documents drawn from the sample space $\Omega = 2^{\mathcal{D}}$. The probability of a term $P(t)$ will thus be determined by the subset $\mathcal{D}_t$ of $\mathcal{D}$ in which t occurs at least once, so that the probability can be estimated by maximum likelihood as $\frac{\mathcal{D}_t}{\mathcal{D}}$. Similarly, the probability of a category c will

Algorithm 2.1: Feature selection by Expected Mutual Information

```
fsI($\mathcal{T}, c, a$): $\mathcal{T}_s$
    var: $C_l, T_t, T_l$ : list
    for each $d \in \mathcal{D}$ do
        if $f(d, c) = true$ do $append(d, C_l)$
        for each $t \in d$ do
            $put(\langle t, d\rangle, T_t)$
            $put(\langle t, 0\rangle, T_l)$
        done
    done
    $P(c) = \frac{|C_l|}{|\mathcal{D}|}$
    for each $t$ in $T_t$ do
        $D_{tc} \leftarrow \{d|d \in C_l \wedge \langle t, d\rangle \in T_t\}$
        $D_t \leftarrow \{d|\langle t, d\rangle \in T_t\}$
        $P(t) \leftarrow \frac{|D_t}{|\mathcal{D}|}$
        $P(t|c) \leftarrow \frac{|D_{tc}|+1}{|C_l|+|T_l|}$
        $P(t, c) \leftarrow P(t|c)P(c)$, $P(\bar{t}, c) \leftarrow (1 - P(t|c))P(c)$
        $P(t, \bar{c}) \leftarrow P(t) - P(t, c)$, $P(\bar{t}, \bar{c}) \leftarrow (1 - P(t))P(\bar{t}, c)$
        $remove(\langle t, 0\rangle, T_l)$
        $i \leftarrow P(t, c) \log \frac{P(t,c)}{P(t)P(c)} + P(\bar{t}, c) \log \frac{P(\bar{t},c)}{P(\bar{t})P(c)} +$
            $P(\bar{t}, \bar{c}) \log \frac{P(\bar{t},\bar{c})}{P(\bar{t})P(\bar{c})} + P(t, \bar{c}) \log \frac{P(t,\bar{c})}{P(t)P(\bar{c})}$
        $add(\langle t, i\rangle, T_l)$
    done
    sort $T_l$ by expected mutual information scores
    return first $\frac{|T_l|}{a}$ elements of $T_l$
```

be estimated with respect to the size of the subset of documents filled under it. We use $P(t, c)$ to denote the probability of picking a document which is of category c and contains term t. and $P(\bar{t}, c)$ for the probability of picking a document which is of category c and does not contain term t. And similarly for conditional probabilities: $P(t|c)$ for the probability that t occurs in a document of category c etc. That is, we use $P(t, c)$ as an abbreviation for $P(T = 1, C = 1)$, $P(\bar{t}, c)$ as an abbreviation for $P(T = 0|C = 1)$ etc, where T and C are Boolean random variables for events (sets) generated by "choices" of terms and categories.

Sebastiani (2002) gives the comprehensive summary, shown in Table 2.2, of feature ranking functions commonly used in text categorisation. Yang and Pedersen (1997) and Forman (2003) present detailed comparisons of the various techniques in terms of their impact on classification and the scale of dimensionality reduction they can achieve without hurting classification performance.

Let's take a closer look at one of these functions, expected mutual information, and see how it can be implemented[8]. The basic intuition behind this function is that it expresses the amount of information about category c one gains by learning that term t occurs in a document. Assuming the probability model described above, calculating expected mutual information simply involves counting elements in the sets defined by the values of the random variables for term (T) and category (C). A method for performing local feature selection on a feature set $\mathcal{T}$ with respect to a category c and *aggressiveness* parameter a (i.e. the ratio between original and reduced feature set) is given in Algorithm 2.1. Yang and Pedersen (1997) report that reduction of up to a factor (aggressiveness) of 100 can be achieved for combined ranking based on expected mutual information, I_{sum}.

Algorithm 2.1 is implemented as part of the text categorisation module of a (free software) suite of natural language libraries[9]. The probabilities in this algorithm are generated by straightforward counting of set intersection elements (maximum likelihood estimation), except for the conditional $P(t|c)$ on which all joint probability values is based. There, we assign a small amount of extra probability mass to each conditional in order to avoid zero values in cases where terms do not co-occur with categories. The rationale for this will be explained below, in section 2.2.4.

Running it on a 3,000 document subset of REUTERS-21578 for target category `acq` produces the ranking shown in Figure 2.5. Except perhaps for `usair`, these to-ranking terms seem to agree with one's intuition about general terms that can identify news stories about company acquisitions.

The kind of dimensionality reduction technique based on ranking functions described above is known as dimensionality reduction by *term filtering*. An alternative to it is to use a machine learning algorithm to select the features through cross validation. In this approach, commonly called feature selection by *term wrapping*, the features that maximise classification accuracy on the validation set are chosen. This approach, despite being usually effective, tends to be significantly costlier (in computational terms) than its filtering counterpart.

[8] There's a certain amount of terminological confusion surrounding the functions we call expected mutual information and mutual information in Table 2.2. Manning and Schütze (1999b), for instance, use the term "pointwise mutual information" to refer to what we simply call mutual information, and call "mutual information" what we call expected mutual information. Others, use the term "information gain" to refer to the latter (Sebastiani, 2002). In information theory, the expected mutual information of random variables X and Y is usually written as $I(X;Y)$ and the sum is over all values these variables can take.

[9] Available at `http://modnlp.berlios.de/`. If you would like to experiment with different measures download the modnlp-tc module and the XML version of REUTERS-21578 from `http://ronaldo.cs.tcd.ie/esslli07/data/` and follow the instructions in `modnlp-tc/README`

```
stake = 0.046524273483876236
merger = 0.03372606977021521
acquisition = 0.027773960515907872
vs = 0.025951563627849852
shares = 0.021425776013160716
buy = 0.019486040643726568
acquire = 0.01772887289440033
qtr = 0.017520597989855537
cts = 0.016385470717669232
usair = 0.016380931873857446
shareholders = 0.014513788146891683
buys = 0.014410168362922963
```

Figure 2.5: Top-ranked words for REUTERS-21578 category `acq` according to expected mutual information

2.2.4 Classifier induction

In this section we illustrate numeric and symbolic text categorisation techniques by describing two of their best known instances: Naïve Bayes and Decision-tree document classification.

Naïve Bayes text categorisation

As mentioned above, there are various ways of modelling the text categorisation task in a Bayesian framework. The simplest model is possibly the multi-variate Bernoulli model described in section 2.2.3, so we will use it as our starting point.

We have seen in section 2.1 that the classification function can be modelled as a conditional probability, namely $P(c|\vec{d})$, which we can simplify through Bayes' rule and make possible to estimate in practice through the (conditional) independence assumption shown in equation (2.2). The great naïvety of the Naïve Bayes (NB) classifier can be clearly seen in the text categorisation task. It is clear that words in a document are not statistically independent (let alone position-independent) of each other. Yet, despite failing to capture this evident characteristic of the domain it models, NB classification works reasonably well, some would say surprisingly well, in practice.

In terms of our chosen document representation (as Boolean-valued vectors), the random variables for which probabilities must be estimated correspond to words $T_1, \ldots, T_n$ drawn from our vocabulary, the feature set $\mathcal{T}$. Let's derive a (soft) classification function $\hat{f} : \mathcal{D} \times \mathcal{C} \to \mathbb{R}$ for this type of representation (Robertson and Jones, 1988; McCallum and Nigam, 1998; Sebastiani, 2002). The starting points are equation (2.3) and the independence assumption specified by equation (2.2). The main source of complexity is in estimating $\prod_{i=1}^{|\mathcal{T}|} P(t_i|c)$. However, given that the value of each feature t_i

in this model is either 0 or 1, we have that each factor can be rewritten as follows

$$\begin{aligned}
P(t_i|c) &= P(t_i|c)^{t_i} \times [1 - P(t_i|c)]^{1-t_i} \\
&= P(t_i|c)^{t_i} \times [1 - P(t_i|c)]^{1} \times [1 - P(t_i|c)]^{-t_i} \\
&= \left[\frac{P(t_i|c)}{[1 - P(t_i|c)]}\right]^{t_i} \times [1 - P(t_i|c)] \qquad (2.5)
\end{aligned}$$

Note that, for the multi-variate Bernoulli we adopted here, $P(t_i|c)$ stands for $P(T_i = 1|C = 1)$, and $P(\bar{t}_i|c)$ for $P(T_i = 0|C = 1)$ etc. Outside probability formulae, t_i can take value 0 or 1.

Now, replacing (2.2) and (2.5) into (2.3) and taking logs, we get

$$\begin{aligned}
P(c|\overrightarrow{d}) &= \frac{P(c)\prod_{i=1}^{|\mathcal{T}|}\left[\frac{P(t_i|c)}{[1-P(t_i|c)]}\right]^{t_i}[1 - P(t_i|c)]}{P(\overrightarrow{d})} \\
\log P(c|\overrightarrow{d}) &= \log P(c) + \sum_{i=1}^{|\mathcal{T}|} t_i \log \frac{P(t_i|c)}{[1 - P(t_i|c)]} + \qquad (2.6) \\
&\quad \sum_{i=1}^{|\mathcal{T}|} \log[1 - P(t_i|c)] - \log P(\overrightarrow{d})
\end{aligned}$$

Since category is a Boolean variable, we have, analogously to (2.6):

$$\begin{aligned}
\log[1 - P(c|\overrightarrow{d})] &= P(\bar{c}|\overrightarrow{d}) \\
&= \log P(\bar{c}) + \sum_{i=1}^{|\mathcal{T}|} t_i \log \frac{P(t_i|\bar{c})}{[1 - P(t_i|\bar{c})]} + \qquad (2.7) \\
&\quad \sum_{i=1}^{|\mathcal{T}|} \log[1 - P(t_i|\bar{c})] - \log P(\overrightarrow{d})
\end{aligned}$$

Finally, subtracting (2.7) from (2.6) we obtain a monotonically increasing function of $P(c|\overrightarrow{d})$ which can be used for training and classification:

$$\begin{aligned}
\log \frac{P(c|\overrightarrow{d})}{[1 - P(c|\overrightarrow{d})]} &= \log \frac{P(c)}{[1 - P(c)]} + \sum_{i=1}^{|\mathcal{T}|} t_i \log \frac{P(t_i|c)[1 - P(t_i|\bar{c})]}{P(t_i|\bar{c})[1 - P(t_i|c)]} + \\
&\quad \sum_{i=1}^{|\mathcal{T}|} \log \frac{1 - P(t_i|c)}{1 - P(t_i|\bar{c})} + 0 \qquad (2.8)
\end{aligned}$$

Since the first and third addends are constant for all documents, the categorisation function can be written simply as

$$\hat{f}(d, c) = \sum_{i=1}^{|\mathcal{T}|} t_i \log \frac{P(t_i|c)[1 - P(t_i|\bar{c})]}{P(t_i|\bar{c})[1 - P(t_i|c)]} \qquad (2.9)$$

As mentioned in section 2.1, in order to implement a hard classifier from a ranking classification function such as (2.9) one needs to choose a threshold value over which the classification decision is positive. Such values can be determined analytically or experimentally. Analytically derived thresholds are often preferred for text categorisation situations where it is desirable to factor in the cost of misclassification, as in spam filtering, for instance. Misclassification cost in those cases is encoded a *loss measure*. When this measure is known, a threshold value can be found which maximises classifier effectiveness (Lewis, 1995). The experimental approach, however, is far more common. Yang (2001) surveys a number of experimental thresholding strategies and identifies three main approaches: *SCut*, in which one varies a value τ on the validation set (D_v) and chooses the value that maximises effectiveness, *proportional thresholding*, in which one chooses a value τ such that it makes generality of the target category, $P(c)$ on the validation set as close as possible to the category's generality on the training set, and *RCut* or *fixed thresholding*, in which one stipulates that a fixed number of categories are to be assigned to each document.

An alternative implementation of NB is given by Mitchell (1997) and further explored by McCallum and Nigam (1998). The main idea in this approach is to make words appear as *values* rather than *names* of attributes, so that the number of times a word occurs is somehow taken into account in parameter estimation. According to this strategy, a representation for this paragraph, for example, would consist of attribute-value pairs as follows:

$$\vec{d} = \langle a_1 = \text{"an"},\ a_2 = \text{"alternative"},\ a_3 = \text{"implementation"}, \ldots \rangle$$

As before, one can make a conditional independence assumption, obtaining $P(\overrightarrow{d}|c) = \prod_{i=1}^{length(\overrightarrow{d})} P(a_i = t_k|c)$ where $P(a_i = t_k|c)$ is the probability that word in position i is t_k, given category c. A problem remains, however. That is, since each attribute's value would range over the entire vocabulary, many values would be missing for a typical document. In order to alleviate this problem, one more assumption is made, namely, that the order of attributes is irrelevant: $P(a_i = t_k|c) = P(a_m = t_k|c), \forall i, m$. This new representation scheme in effect amounts to reintroducing into the model word-count information which is discarded in the multi-variate Bernoulli approach described above.

From the point of view of estimating the individual probabilities, difficulties might arise if none of the training instances with target category c have an attribute a_i with values t_k. In that case, the estimated value $\hat{P}(a_i = t_k|v_j) = 0$ would cause the estimate for the entire document to be zero. A typical solution for this problem is to adopt a *maximum a posteriori* estimate whereby a certain number of "virtual" training instances is assumed to exist which avoid zero values:

$$\hat{P}(a_i = t_k|c) = \frac{n_c + mp}{n + m} \tag{2.10}$$

Algorithm 2.2: NB Probability estimation

```
1   NB_Learn($D_t$,$\mathcal{C}$)
2     /* collect all  tokens that occur in $D_t$ */
3     $\mathcal{T} \leftarrow$ all distinct words and other tokens in $D_t$
4     /* calculate  $P(c_j)$ and $P(t_k|c_j)$ */
5     for each target value $c_j$ in $\mathcal{C}$ do
6        $D_t^j \leftarrow$ subset of $D_t$ for which target value is $c_j$
7        $P(c_j) \leftarrow \frac{|D_t^j|}{|D_t|}$
8        $Text_j \leftarrow$ concatenation of all texts in $D_t^j$
9        $n \leftarrow$ total number of tokens in $Text_j$
10       for each word $t_k$ in $\mathcal{C}$ do
11          $n_k \leftarrow$ number of times word $t_k$ occurs in $Text_j$
12          $P(t_k|c_j) \leftarrow \frac{n_k+1}{n+|\mathcal{T}|}$
13       done
14    done
```

where n is the number of training examples for which $C = c$, n_c number of examples for which $C = c$ and $a_i = t_k$, p is the prior estimate for $\hat{P}(a_i = t_k|c)$, and m is the weight given to the prior (the number of virtual examples). Algorithm 2.2 is due to Mitchell (1997) and uses a uniform prior and a number of virtual examples equal to the number of possible values an attribute a_i can take. This technique is known as Laplace smoothing. Note that although this data representation scheme does make use of word counts, the counting (as shown in lines 8 and 11 of algorithm 2.2) is done at the level of categories rather than documents.

Once the probabilities have been estimated, calculating the values of the categorisation function for a new document is a question of tokenising the document into t tokens and using the stored probabilities to calculate the value $P(c)\prod_{k=1}^{n} P(t_k|c)$ for the target category. As with the multi-variate Bernoulli model, thresholding techniques can be used to convert this value into a hard classification decision. In the case of single-label categorisation, as in the 20 newsgroups classification task reported in (Mitchell, 1997), one can implement the categorisation function $\hat{f} : \mathcal{D} \rightarrow \mathcal{C}$ directly by choosing the category that maximises the probability:

$$\hat{f}(d) = \arg\max_{c \in \mathcal{C}} P(c) \prod_{i=1}^{n} P(a_i|c) \tag{2.11}$$

Mitchell (1997) reports accuracy of 89% for the NB implementation described above on a categorisation task involving categorising Usenet news posts into 20 different newsgroups, using a corpus of 1000 training instances per newsgroup. Although NB does not figure among the top performing text categorisation methods, its performance is still quite respectable, given the fact that the independence assumption on which the method relies is so obviously false. Equation (2.11) gives us an idea of why the method performs well: we are not really interested in estimating accurate posteriors $\hat{P}(c|\overrightarrow{d})$ but in ranking the likely target categories in the right order. In other words, it suffices that $\arg\max_{c\in\mathcal{C}} \hat{P}(c)\prod_i \hat{P}(a_i|c) = \arg\max_{c\in\mathcal{C}} P(c)P(a_1\ldots,a_n|c)$ for the categorisation decision to be accurate. The same is true of the multivariate Bernoulli variant of NB presented above. Domingos and Pazzani (1996) analyse NB methods in detail, pointing out that NB yields posteriors which tend to be (unrealistically) close to 1 or 0.

Finally, one can still use NB methods if one chooses to represent the data as real-valued vectors. Representing documents in this way is common in the area of information retrieval, so this choice of representation would seem natural for text categorisation as well. Features could be assigned, for instance, TFxIDF (term-frequency, inverse document frequency) scores which better quantify the weight a word (term) is likely to have on the categorisation decision:

$$tfidf(t_k, d_j) = \#(t_k, d_j) log\frac{|Tr|}{\#_{Tr}(t_k)} \tag{2.12}$$

where $\#(t, d)$ is the number of occurrences of token t in document d and $\#_{\mathcal{D}}(t_k)$ is number of documents in the corpus $\mathcal{D}$ in which t occurs. Cosine normalisation is often used to constrain the values in $\overrightarrow{d_j}$ within the interval $[0, 1]$ (Manning and Schütze, 1999b):

$$t_{kj} = \frac{tfidf(t_k, d_j)}{\sqrt{\sum_{i=1}^{|\mathcal{T}||} tfidf(t_i, d_j)^2}} \tag{2.13}$$

Once the vector representation is in place, one can assume that the values are normally distributed and employ equation (2.4) to estimate the probabilities.

Unlike the first two NB methods described in this section, this method employs a data representation scheme which models word counts at the document level. However, this extra level of representational accuracy does not seem to yield better classification results. Of the variants described in this section, the multinomial model is generally regarded as the best performing NB classifier for text categorisation tasks (Yang and Liu, 1999).

Symbolic text categorisation: decision trees

Decision trees for text categorisation can either employ a Boolean vector representation, along the lines of the multi-variate Bernoulli model discussed above, or start off with a real-valued representation and discretise the attributes. We will describe the Boolean vector case, as it is the most straightforward and the principles are essentially the same for multi-valued discretised representations.

Algorithm 2.3: Decision tree learning

```
1  DTreeLearn($D_t$): $2^{\mathcal{D}}$, $\mathcal{T}$: $2^{\mathcal{T}}$, default: $\mathcal{C}$): tree
2  if isEmpty($D_t$) then
3     return default
4  else if $\prod_{d_i \in D_t} f(d_i, c_j) = 1$ then /* all $d_i$ are of class $c_j$ */
5     return $c_j$
6  else if isEmpty($\mathcal{T}$) then
7     return MajorityCateg($D_t$)
8  else
9     $t_{best} \leftarrow$ ChooseFeature($\mathcal{T}, D_t$)
10    $tree \leftarrow$ new dtree with $root = t_{best}$
11    for each $v_k \in t_{best}$ do
12       $D_t^k \leftarrow \{d_l \in D_t \mid t_{best}$ has value $v_k$ in $d_l\}$
13       $sbt \leftarrow$ DTreeLearn($D_t^k$, $\mathcal{T} \setminus \{t_{best}\}$, MajorityCateg($D_t$))
14      add a branch to $tree$ with label $v_k$ and subtree $sbt$
15    done
16 return tree
```

The decision tree learning method outlined in section 1.1 is detailed in Algorithm 2.3. It's crucial step is the choice of the feature to be used in partitioning the training set (line 9). There are different ways of implementing this choice, but they are all based on measuring the reduction in *entropy* that would result from partitioning the training set D_t according to the values of a feature. Entropy is an information theoretic function that quantifies the amount of uncertainty in a probability distribution. In general, the entropy of a discrete random variable X is given by $H(X) = H(p) = -\sum_{x \in X} p(x) \log p(x)$, where $p(x)$ gives the probability mass function for random variable X. Logarithms are base 2, and $0 \log(0) \overset{def}{=} 0$.

In our case, since we are dealing with binary classification, the entropy of (the probability distribution of categories of) a set of documents $\mathcal{D}$ is given by

$$H(\mathcal{D}) = -\frac{|\mathcal{D}_c|}{|\mathcal{D}|} \log \frac{|\mathcal{D}_c|}{|\mathcal{D}|} - \frac{|\mathcal{D}_{\bar{c}}|}{|\mathcal{D}|} \log \frac{|\mathcal{D}_{\bar{c}}|}{|\mathcal{D}|} \tag{2.14}$$

where $|\mathcal{D}_c|$ ($|\mathcal{D}_{\bar{c}}|$) is the number of positive (negative) instances filed under category c in $\mathcal{D}$.

Back to `ChooseFeature(.)`, the most commonly used criterion for choosing a feature is the one adopted in ID3 and its descendants (Quinlan, 1986), namely, to choose the feature that maximises *information gain*. For binary classification and Boolean-valued features the information gain caused by feature T on set $\mathcal{D}$ is:

$$IG(T, \mathcal{D}) = H(\mathcal{D}) - [\frac{\mathcal{D}_{tc} + \mathcal{D}_{t\bar{c}}}{\mathcal{D}} H(\mathcal{D}_t) + \frac{\mathcal{D}_{\bar{t}c} + \mathcal{D}_{\bar{t}\bar{c}}}{\mathcal{D}} H(\mathcal{D}_{\bar{t}})] \tag{2.15}$$

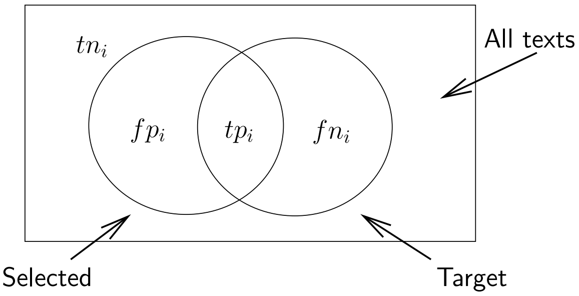


Figure 2.6: Possible classification outcomes

where $\mathcal{D}_t$ and $\mathcal{D}_{\bar{t}}$ are the subsets of $\mathcal{D}$ containing instances for which T has value 1 and 0, respectively.

Although decision trees are naturally suited for hard classification, one can also assign probabilities to a leaf node (i.e. the probability that a new document d belonging to that node should be filled under category c). One can estimate such probabilities for a document filed under node n as shown in equation (2.16), where $P(c|d_n)$ is the probability that a document d_n which ended up in node n belongs to category c, and $|D_{cn}|$ is the number of documents in node n which have been assigned category c. The estimator uses add-one smoothing.

$$P(c|d_n) = \frac{|D_{cn}| + 1}{|D_{cn}| + |D_{\bar{c}n}| + 1 + 1} \tag{2.16}$$

2.2.5 Evaluation

Evaluation of classifiers in general and text categorisation systems in particular is almost always performed experimentally. Experimental evaluation aims at measuring classifier *effectiveness* in terms of its ability to make correct classification decisions for the largest possible number of documents. Two measures are commonly used: *precision* and *recall*. These measures are defined in terms of relations between a *target set* (with respect to category c), defined as $T_c = \{d|f(d,c) = 1\}$ and a *selected set*, defined as $S_c = \{d|\hat{f}(d,c) = 1\}$. These relations are shown in Figure 2.6 (Manning and Schütze, 1999b).

The intersection is occupied by *true positives* (tp_i), i.e. instances the system classified as belonging to the target class c_i which did, in fact, belong to it. Instances mistakenly classified as not belonging to the target class form the set of *false negatives* (fn_i). Instances mistakenly classified as belonging to the target class form the set of *false positives* (fn_i). Instances which are neither targeted nor selected form the set of *true negatives* (tn_i).

Local precision, i.e. precision with respect to category c_i, is defined as the proportion of correctly classified instances in the selected set for that

Table 2.3: Precision and recall scores for sample news articles

	Sport		Politics		World	
Judgment: **E**xpert and **S**ystem	**E**	**S**	**E**	**S**	**E**	**S**
"Brazil beat Venezuela"	true	false	false	false	false	true
"US defeated Afghanistan"	false	true	true	true	true	false
"Elections in Wicklow"	false	false	true	true	false	false
"Elections in Peru"	false	false	false	true	true	true
Precision (local):	$\pi = 0$		$\pi = 0.67$		$\pi = 0.5$	
Recall (local):	$\rho = 0$		$\rho = 1$		$\rho = 0.5$	
$\pi^{\mu} =$			$\frac{0+2+1}{0+2+1+1+1+1} = .5$			
$\pi^{M} =$			$\frac{0+.67+.5}{3} = .39$			

category:

$$\pi_i = \frac{tp_i}{tp_i + fp_i} \tag{2.17}$$

Local recall for classification under category c_i, is defined as the proportion of instances identified by the classifier in the targeted set:

$$\rho_i = \frac{tp_i}{tp_i + fn_i} \tag{2.18}$$

Global scores, i.e. precision and recall across all categories in $\mathcal{C}$ through macro- or micro-averaging. Equations (2.19) and (2.20) show how to calculate micro- and macro-averages for precision. The formulae for recall are analogous.

$$\pi^{\mu} = \frac{\sum_{i=1}^{|\mathcal{C}|} tp_i}{\sum_{i=1}^{|\mathcal{C}|}(tp_i + fp_i)} \tag{2.19}$$

$$\pi^{M} = \frac{\sum_{i=1}^{|\mathcal{C}|} \pi_i}{|\mathcal{C}|} \tag{2.20}$$

An example of how these measures are calculated is shown in Table 2.3. The table shows titles of (made up) news articles and their target ("expert") and classification ("system") functions, along with the precision and recall values per category and globally.

Neither precision nor recall, in isolation, tells the whole story about the effectiveness of a classifier. Classifiers can be tuned to maximise one at the expense of the other, an in some applications it is desirable to do so. The prime example is again spam filtering, where the loss due to a false positive is much greater than that associated with a false negative, and therefore precision usually takes precedence over recall. For purposes of comparison on standard text categorisation benchmarks, measures that combine π and ρ have been proposed. The F measure (van Rijsbergen, 1979) is widely used.

The F score combines precision and recall values and includes a parameter β ($0 \leq \beta \leq \infty$) which is meant to quantify the relative importance of these values. F_β is calculated as follows:

$$F_\beta = \frac{(\beta^2 + 1)\pi\rho}{\beta^2\pi + \rho} \tag{2.21}$$

A β value of 1 assigns equal importance to precision and recall.

Another strategy commonly used to assess the trade-off between precision and recall in ranking classifiers is to plot π as a function of ρ by varying the classification thresholds for the classification function from 1 to 0. With the threshold set to 1, only those documents that the classifier is "totally sure" belong to the category will be selected, so π will tend to 1, and ρ to 0. As we decrease the value of the threshold, precision will decrease, but ρ will increase monotonically. A measure associated with this strategy is the *breakeven point*, defined as the value at which the plot intersects the $\rho = \pi$ line. A related strategy is to plot a *receiver operating characteristic*, or *ROC* to investigate how different levels of fallout (the proportion of non-targeted items that were mistakenly selected) influence recall or sensitivity.

Measures such as accuracy and error are less frequently used, since they are less sensitive to variations in the number of correct decisions, due to the fact that true negatives tends to dominate the distribution of classification outcomes, as illustrated in (Manning and Schütze, 1999b, ch. 8).

Yang and Liu (1999) describe a comprehensive methodology for comparing performance of text classifiers. Sebastiani (2002) surveys the performance of various machine learning methods and systems (reported in terms of breakeven scores) on various benchmark corpora.

2.3 Other classification tasks in NLP

In addition to text categorisation, supervised learning has been applied to a variety of natural language processing tasks. These include re-ranking the output of probabilistic parsers (Collins and Koo, 2005; Shen and Joshi, 2005), authorship identification, language identification, keyword extraction (Witten et al., 1999; Kelleher and Luz, 2005), word-sense disambiguation (Pedersen, 2000; Escudero et al., 2000) and many other application areas. In terms of availability of training data, which tends to be an issue in natural language applications, variants of supervised learning have been proposed which can learn with unlabelled data in addition to small amounts of labelled data. These are semi-supervised learning, which includes algorithms such as the expectation maximisation (EM) and the Baum-Welch algorithms which will be examined in later lectures, and active-learning, which iteratively requests labels for certain instances selected according to how "useful" they may be to the learning process.

```
RIV y be? Then he ran down along the bank , toward a narrow , muddy path .
FIN  four bundles of small notes the bank cashier got it into his head
RIV ross the bridge and on the other bank you only hear the stream , the
RIV beneath the house , where a steep bank of earth is compacted between
FIN op but is really the branch of a bank . As I set foot inside , despite
FIN raffic police also belong to the bank . More foolhardy than entering
FIN require a number .  If you open a bank account , the teller identifies
RIV  circular movement , skirting the bank of the River Jordan , then turn
```

Figure 2.7: Concordances for the word "bank"

As a final illustration of supervised learning, let's consider the task of word-sense disambiguation by decision trees. The disambiguation task can be modelled much the same way as the text categorisation task. Given an occurrence of an ambiguous word, in a context of surrounding words, the classifier must map the word to a single sense. Consider, for instance, the concordances for occurrences of the word "bank" (Figure 2.7), some of which refer to a financial institution (`FIN`) and some of which refer to the land alongside a river (`RIV`). In order to describe disambiguation between `FIN` and `RIV` as a classification task we could start by regarding each concordance as an instance $d \in \mathcal{D}$ to be represented as a feature vector. The values of these features could be assigned, for example, as in our multi-variate Bernoulli text representation scheme, with 1 corresponding to one or more occurrences of a word in the left or right context of the keyword and 0 corresponding to non-occurrence. Alternatively, we could assign numeric values according to the distance between the word (feature) and the keyword to be disambiguated.

Adopting the multi-variate Bernoulli approach, we have trained a decision tree classifier using Algorithm 2.3 on a small set of concordances of the word "bank" and tested it through 10-fold cross validation. The data vectors were created by scanning the left and right contexts up to 10 words away from the keyword, excluding stop words. The feature set thus created, excluding low frequency words, consisted of the following terms: *small, money, on, to, river, from, in, his, accounts, when, by, other, estuary, some, with.* The decision tree produced is shown in Figure 2.8. Even with this simple decision tree we have managed to obtain F_1 scores of 0.941 for `RIV` and 0.944 for `FIN`.

A 3-NN classifier for the same data using numeric vectors with values corresponding to the distance of the various features to the keyword (negative integers for words to the left of the keyword, positive integers for words to the right of the keyword) obtained F_1 scores of 0.88 and 0.91, respectively. This is again a fairly decent performance, if one considers the small amount of available training data and the simplicity of the classification strategy employed.

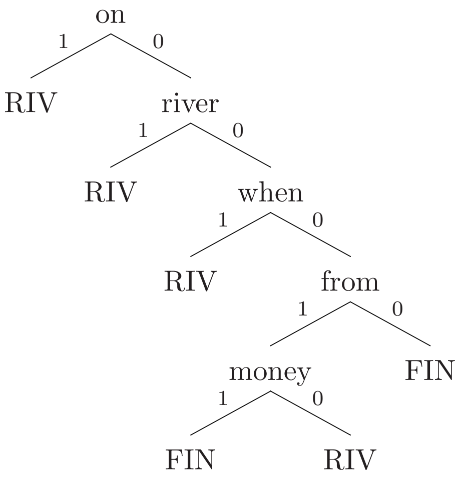


Figure 2.8: Decision tree for disambiguation of the word "bank"

2.4 Practical activities

The best way to get a good idea of how the above described techniques work is to apply them in practice. We have developed a suite of text categorisation utilities and libraries, available as Free Software (GPL) from the modnlp web site[10] which (although it is pretty much work in progress) should provide you with the basic functionality you need to represent documents, train NB classifiers, run classification experiments and evaluate the results. The software handles parsing of (the XML version of) the REUTERS-21578 and Ling-Spam[11] corpora, stop-word removal, feature selection (including implementation of many of the measures listed in Table 2.2), classification and exporting to the ARFF format used by the Weka machine learning toolkit[12].

We suggest you try running the feature selection and classification utilities of modnlp-tc. You will find instructions on how to do so on the project web page, along with links to a couple of text categorisation corpora. Alternatively, if you don't mind doing some hands-on Java programming, you could follow our step-by-step tutorial on building a simple (but complete) Naïve Bayes text categorisation system. The tutorial is available on the course website[13] and contains instructions on how to download the required software.

On `http://ronaldo.cs.tcd.ie/esslli07/` you will also find a larger concordance set for the word "bank" and Perl scripts that generate the basic Boolean and numeric vector representations described in section 2.3. Try

[10] `http://modnlp.berlios.de/tc.html`

[11] `http://www.aueb.gr/users/ion/data/lingspam_public.tar.gz`

[12] `http://www.cs.waikato.ac.nz/ml/weka/`

[13] http://ronaldo.cs.tcd.ie/esslli07/practicals/tcsystem.pdf

labelling the entire list, generating ARFF files for it with the Perl scripts and using Weka to train a classifier to disambiguate the instances. Feel free to modify the scripts to produce different representations (by, for instance, scanning shorter or longer contexts, assigning nominal values from a wider range to features, etc) and test the effect your changes might have on disambiguation performance.

Lecture 3

Unsupervised learning and applications

In lecture 2 we examined classification tasks learnt through supervised learning. Learning there was based on a training set where the labelling of instances defined the target function of which the classifier implemented an approximation. The input was an unlabelled instance and the output a classification decision. This lecture introduces *unsupervised learning*, where the concept of "learning" is somewhat different. As before, the learning algorithm seeks to produce a generalisation but this time no explicit approximation of a target function is built[1]. The generalisation sought consists in revealing natural groupings within a data set. These groupings are discovered exclusively through processing of unlabelled instances.

Unsupervised learning has been used in exploratory data analysis (data mining), where clustering can reveal patterns of association in the data, thus forming the basis for visualisation of association patterns. Dendrograms and self-organising maps are typical examples of such information visualisation techniques. Unsupervised learning has also been employed in several areas of natural language processing, including information retrieval, object and character recognition, and dimensionality reduction for text categorisation.

3.1 Data representation

As with supervised learning, it is necessary in unsupervised learning to adopt a uniform data representation model. The general vector space model described in lecture 2 can also be adopted here. As before, the composition of the feature set and the way values are assigned is specific to the learning task. Document clustering, for instance, could be based on real-valued feature vectors built as described in section 2.2.4. Word clustering could be

[1] This is, incidentally, a feature unsupervised learning methods share with supervised instance-based learners

$$\begin{array}{lcccccl}
 & \text{lecture} & \text{we} & \text{examined} & \text{clustering} & \text{groups} & \ldots \\
\text{lecture} \quad = \langle & 2, & 2, & 1, & 2, & 0 & ,\ldots\ \rangle \\
\text{we} \quad = \langle & 2, & 2, & 1, & 2, & 0 & ,\ldots\ \rangle \\
\text{examined} = \langle & 1, & 1, & 1, & 2, & 0 & ,\ldots\ \rangle \\
\text{clustering} = \langle & 2, & 2, & 1, & 3, & 1 & ,\ldots\ \rangle \\
\text{groups} \quad = \langle & 0, & 0, & 0, & 1, & 1 & ,\ldots\ \rangle \\
\quad\vdots & & & \vdots & & &
\end{array}$$

Figure 3.1: Co-occurrence vector representation for words

based on vectors of word co-occurrence in documents, where the feature set is the set of types and the entire data set can be seen as a co-occurrence matrix.

If we took, for instance, section 3, section 3.2 and this section up to this sentence to be our document set, adopting a co-occurrence vector representation for words would give us a representation along the lines shown in Figure 3.1. Intuitively, one can see that the word “groups” is somehow the odd one out and that its closest relative appears to be the word “clustering”. Unsupervised learning essentially aims to uncover this sort of relationship.

3.2 Main algorithms

Clustering algorithms form the main class of unsupervised learning techniques. Jain et al. (1999) classify them into two major groups: hierarchical clustering and partitional clustering. Hierarchical clustering techniques differ with respect to the way distances between clusters are computed during the clustering process. Single-link, complete-link and average-link are the most often used hierarchical clustering algorithms. Partitional clustering includes k-means clustering (an instance of the Expectation Maximisation algorithm), graph theoretic clustering, mode seeking, etc. An orthogonal distinction between *agglomerative* and *divisive* clustering can also be established with respect to the cluster construction algorithm employed (Jain et al., 1999). Agglomerative techniques create hierarchical structures from the bottom up, starting with each instance in a distinct cluster and building up a structure through successive merging operations. Divisive clustering, on the other hand, starts with all instances in a single cluster and builds the structure by splitting clusters until a stopping criterion is met.

Grouping data instances implies assessing how “close” instances are to each other. In other words, it involves calculating *distances* between two instances. In hierarchical clustering, the criterion for joining groups of instances into larger groups is the *similarity* between groups, which is calculated as a function of the distances between pairs of instances, one from each group.

Algorithm 3.1: Simple agglomerative hierarchical clustering

```
hclust($\mathcal{D}$: set of instances): tree
  var: $C$,  /* set of clusters */
       $M$  /* matrix containing distances between */
           /* pairs of clusters                    */
  for each $d \in \mathcal{D}\}$ do
     make $d$ a leaf node in $C$
  done
  for each pair $a, b \in C$ do
     $M_{a,b} \leftarrow d(a, b)$
  done
  while (not all instances in one cluster) do
    Find the most similar pair of clusters in $M$
    Merge these two clusters into one cluster.
    Update $M$ to reflect the merge operation.
  done
  return $C$
```

3.2.1 Distance and dissimilarity measures

As mentioned in the previous section, clustering implies a notion of "distance" between objects in the data set. This notion is formalised below.

Given instances a, b and c represented as real-valued vectors, as described above, we define a *distance* between a and b as a function $d(a, b)$ satisfying the following properties:

$$d(a,b) \geq 0 \tag{3.1}$$

$$d(a,a) = 0 \tag{3.2}$$

$$d(a,b) = d(b,a) \tag{3.3}$$

$$d(a,b) \leq d(a,c) + d(b,c) \tag{3.4}$$

If condition (3.4), the so called triangular inequality, is not satisfied, d is called a *dissimilarity*. Clustering algorithms use these measures in order to group (in the case of agglomerative techniques) objects (instances and clusters) together within a tree structure.

3.2.2 Hierarchical clustering

The output of a hierarchical clustering algorithm is a tree structure called a *dendrogram*, in which links between (sister) nodes indicate similarity between clusters and the height of the links indicate their degree of similarity. Algorithm 3.1 describes a simple and general agglomerative clustering strategy.

Figure 3.2 illustrates an application of an instance of Algorithm 3.1 (complete-link clustering) to seven instances on a 2-dimensional space. The

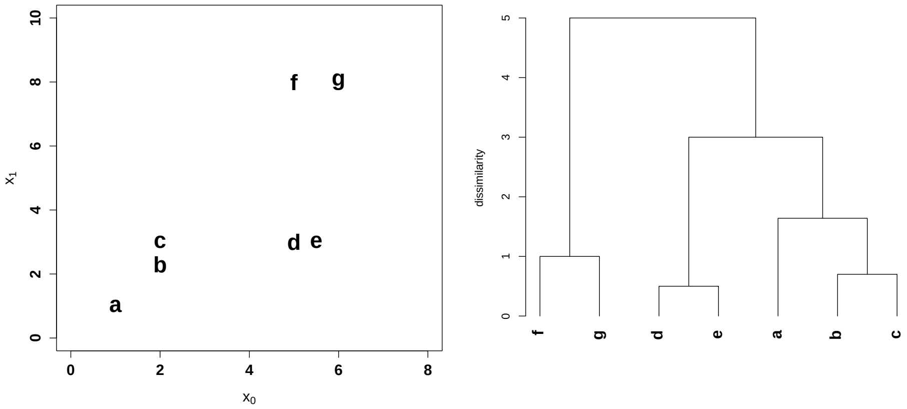


Figure 3.2: Clustering in 2-dimensional space

location of the instances on the plane is plotted on the left and the resulting dendrogram is shown on the right. Note how the linking patterns reflect the geometric distance between the instances on the plane.

The final result of the clustering algorithm varies depending on how one defines the distance function and the notion of *similarity* (line 12 of Algorithm 3.1). There are various ways to define distance and similarity. Commonly used distance functions are Hamming distance, Manhattan distance (1 norm), Euclidean distance (2 norm) and maximum distance (supremum norm). The choice here is largely dependent on domain and data representation. In the example of Figure 3.2 we used Euclidean distance, which is defined as $d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{|\vec{x}|}(x_i - y_i)^2}$. Different definitions of similarity give rise to three widely used hierarchical clustering methods, namely: *single-link* , *complete-link* and *average-link* clustering.

In single-link clustering similarity is defined in terms of the *minimum* of the distances between all pairs of instances from the two clusters. That is, if *dissimilarity* is given by minimum distance, similarity can be defined as follows:

$$sim_s(c_1, c_2) = \frac{1}{1 + \min_{x_1 \in c_1, x_2 \in c_2} d(x_1, x_2)} \tag{3.5}$$

In complete-link clustering, similarity between two clusters is defined in terms of the *maximum* distance between any two instances drawn from each cluster:

$$sim_c(c_1, c_2) = \frac{1}{1 + \max_{x_1 \in c_1, x_2 \in c_2} d(x_1, x_2)} \tag{3.6}$$

Finally, in average-link clustering, similarity is inversely proportional to the mean distance between all pairs of instances:

$$sim_a(c_1, c_2) = \frac{1}{1 + \frac{1}{|c_1||c_2|}\sum_{x_1 \in c_1}\sum_{x_2 \in c_2} d(x_1, x_2)} \tag{3.7}$$

Single-link is probably the most versatile of these techniques, though it tends to produce elongated clusters, which might lead to incorrect clustering in the presence of data clusters connected by noise. An example of this undesirable behaviour is shown in Figure 3.3, where the ellipses represent clusters of instances of types A and B connected by "noise" instances represented in the graph by asterisks (Jain et al., 1999). Complete-link tend to produce more compact groups but is more sensitive to the presence of outliers.

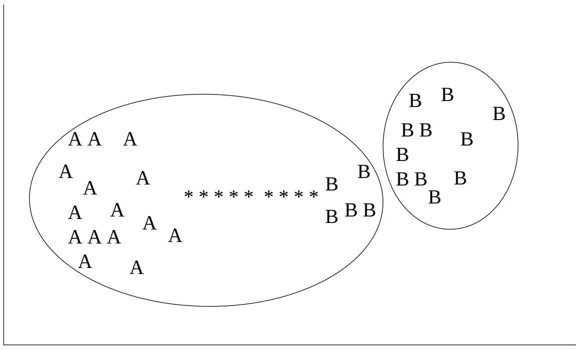
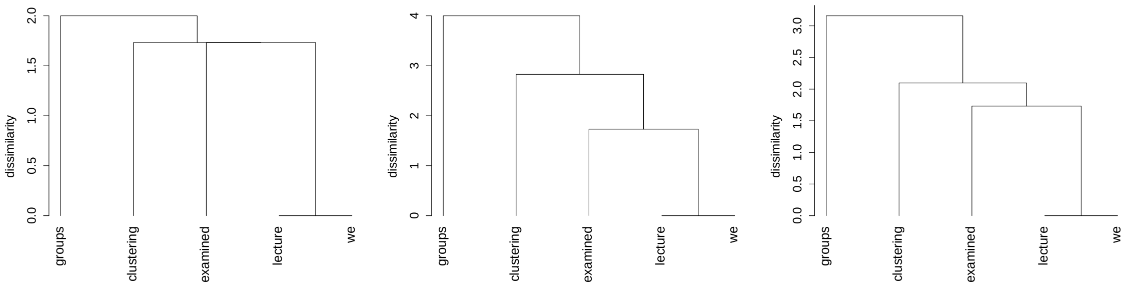


Figure 3.3: Straggly single-link clusters.

In order to further illustrate the differences in linkage criteria for hierarchical clustering, we plotted (Figure 3.4) the dendrograms that result from applying each of the three clustering methods to the co-occurrence word vectors of Figure 3.1. The single link method produces, as mentioned above, a flat elongated cluster containing most words while the other methods discriminate the word "clustering".



(a) (b) (c)

Figure 3.4: Clustering of word vectors of Figure 3.1: (a) single linkage, (b) complete linkage, (c) average linkage

Algorithm 3.2: K-means clustering

```
k-means($X = \{\vec{d}_1, \dots, \vec{d}_n\} \subseteq \mathbb{R}^m$, $k$): $2^{\mathbb{R}}$
    $C : 2^{\mathbb{R}}$  /* $\mu$ a set of clusters */
    $d : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$ /* distance function */}
    $\mu : 2^{\mathbb{R}} \rightarrow \mathbb{R}$ /* $\mu$ computes the mean of a cluster */}
  select $C$ with $k$ initial centres $\vec{f}_1, \dots, \vec{f}_k$
  while stopping criterion not true do
    for all clusters $c_j \in C$ do
       $c_j \leftarrow \{\vec{d}_i | \forall f_l d(\vec{d}_i, f_j) \leq d(\vec{d}_i, f_l)\}$
    done
    for all means $\vec{f}_j$ do
       $\vec{f}_j \leftarrow \mu(c_j)$
    done
  done
  return $C$
```

3.2.3 K-means clustering

The k-means algorithm is one of the best known partitional clustering methods. The strategy it employs, detailed in Algorithm 3.2, consists essentially in iterating through the set of instances $d_1, \dots, d_n$ assigning instances to the clusters with the nearest means (centroids), updating the cluster means and continuing the reassignments until a stopping (or convergence) criterion is met. A natural stopping criterion is to stop when no new reassignments take place.

Unlike hierarchical clustering, k-means starts off with a target number k of clusters and generates a flat set of clusters. As before, a number of options are available as distance metrics. Again, Euclidean distance is the most common choice. The cluster mean is calculated, in the obvious way, as follows:

$$\mu(c) = \frac{1}{|c|} \sum_{\vec{x} \in c} \vec{x} \tag{3.8}$$

One of the main advantages of partitional methods over hierarchical clustering is the lower computational costs incurred by the former. Hierarchical clustering algorithms of the kinds described in section 3.2.2 generally have $O(n^2)$ worst-case run time (Murtagh, 1984) while the run time of k-means is $O(n)$. This computational advantage is particularly relevant in the domain of natural language processing, where the volumes of data to be processed tend to be very large. The trade-off is that, as in other partitional methods, one needs to decide in advance the number of clusters to be created. Although strategies designed to optimise that choice have been devised, these strategies have the effect of increasing the overall complexity (Jain et al., 1999) of the method.

Finally, we point out that K-means can be seen as a specialisation of the expectation maximisation (EM) algorithm, other instances of which will be presented in Lectures 4 and 5. In Algorithm 3.2, the iteration stating on line 7 broadly corresponds to the expectation step, while the iteration starting on line 10 corresponds to maximisation.

3.3 Applications to Natural Language Processing

As mentioned above, in addition to being readily applicable to exploratory data analysis, unsupervised learning techniques have been used in a number of natural language processing applications. Unsupervised methods have been extensively used in the area of information retrieval for keyword (Sparck Jones and Jackson, 1970) and document (van Rijsbergen, 1979) clustering. The latter has been investigated since the early days of information retrieval in connection with both retrieval efficiency (Salton, 1968) and effectiveness (van Rijsbergen, 1979). Retrieval efficiency (in terms of computational cost) can be improved by clustering the documents in a collection, selecting a vector representative of each cluster, and initially restricting the search to those representative vectors. Although Salton (1968) conjectures that employing clustering this way reduces retrieval effectiveness (in terms of the relevance of the retrieved matches), the opposite is suggested by (Jardine and van Rijsbergen, 1971) who propose a method for information retrieval which uses (single-link) hierarchical clustering with the aim of improving effectiveness. They put forward a *cluster hypothesis* which states that "closely associated documents tend to be relevant to the same requests" (van Rijsbergen, 1979, p. 30) and show that clustering can produce promising results. More recently, clustering techniques have been employed in order to identify and make use of homogeneous subsets of corpora for language modelling in information retrieval (Kurland and Lee, 2004).

Another application of unsupervised learning to information retrieval is described by Jain et al. (1999). The test data they used are a set of books pre-classified according to the ACM Computing Reviews classification hierarchy. Unlike previous approaches, a "knowledge-based" representation (Murty and Jain, 1995) is adopted which consists basically in using the ACM category codes to produce descriptors in the form of generalised lists (conjunctions of weighted disjunctions of category codes). A new distance metric based on the lengths of the largest common prefixes of the category codes is defined and complete-link clustering is used to produce a dendrogram encoding the relationships between books. A flat set of categories is then generated by imposing an empirically derived threshold on the hierarchy. Users searching for specific category codes have their requests mapped to the matching clusters.

Other applications of clustering techniques to natural language processing have been described in the literature. They include the use of clustering for improving language models (Manning and Schütze, 1999b), character recognition (Jain et al., 1999), and dimensionality reduction (Sebastiani, 2002).

In the following section, we present another practical application of unsupervised learning in natural language processing, namely, a simple approach to feature extraction which implements dimensionality reduction for text categorisation.

3.3.1 Feature extraction for text categorisation

In lecture 2 we described the feature selection approach to dimensionality reduction. Feature selection reduces dimensionality by filtering out of the feature set (the vocabulary, in the case of text categorisation) those terms which are deemed to be the least informative according to some quantitative criterion. Feature extraction, in contrast, builds an entirely new (but considerably smaller) feature set out of the original features. The elements of this new set are usually objects of a different kind to those in the original feature set. Feature extraction therefore requires the text categorisation system to implement the extra step of converting the tokens of its input documents to values compatible with the new feature set.

Feature extraction can be implemented in different ways. One could take an input consisting, as in section 3.1, of co-occurrence vectors and apply, for instance, k-means clustering to produce a reduced set of k basic features represented by each cluster's centroid. Given a document for categorisation, the system would match the document's tokens against the clusters in some way and build a corresponding vector representation. This matching operation could be, for instance, a cluster membership test for a Boolean vector representation, or maybe a measure of the distance between each feature (cluster centroid) and a mapping of the document to the original (i.e. prior to feature extraction) feature space, for a real-valued vector representation.

As an example, consider the co-occurrence matrix shown in Table 3.1. The choice of words there is somewhat contrived[2] and common pre-processing steps such as stemming were bypassed, but the co-occurrence patterns were actually extracted from a 3,000-document subset of REUTERS-21578. In order to extract a reduced set of, say, five features, one could apply Algorithm 3.2 ($k = 5$) to obtain the clusters shown in Table 3.2. Note that despite its simplicity, this feature extraction method is capable of singling out highly discriminative terms (with respect to target category `acq`) and assigning terms one would think have strongly related meanings, such as "usair" and "twa" (both names of airline companies), to the same cluster.

[2] In order to keep the example understandable the terms in Table 3.1 were pre-selected according to an odds ratio criterion (see section 2.2.3) targeting a single category (`acq`).

Table 3.1: Sample co-occurrence matrix for a subset of REUTERS-21578

usair	20	2	0	1	0	4	1	0	0	0	1	0	2	0	3	0	0	0	1	0	1	1	0	0	0	0	0	0	0	2	14	1	3
voting	2	10	0	2	0	1	0	0	0	0	0	0	2	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	2	0	0
buyout	0	0	8	1	0	2	0	0	0	0	0	0	1	1	0	0	0	0	3	0	0	0	0	0	0	0	0	1	0	1	0	0	0
stake	1	2	1	62	0	0	0	1	1	0	0	1	2	0	0	0	0	2	0	1	0	0	1	0	1	2	1	0	0	0	1	0	0
santa	0	0	0	0	7	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	2	0	0	1	0
merger	4	1	2	0	3	48	0	1	3	0	2	0	1	2	4	1	0	0	1	1	2	0	0	0	2	0	0	1	0	5	4	3	1
ownership	1	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
rospatch	0	0	0	1	0	1	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rexnord	0	0	0	1	0	3	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
designs	0	0	0	0	0	0	0	0	0	5	0	0	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	1	1	0	0	0
pie	1	0	0	0	0	2	0	0	0	0	5	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	1
recommend	0	0	0	1	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
acquire	2	2	1	2	0	1	0	0	0	1	0	0	32	1	0	0	2	0	0	1	0	0	0	0	0	0	0	0	4	9	2	0	0
definitive	0	0	1	0	1	2	0	0	0	0	0	0	1	8	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	2	0	0	0
piedmont	3	0	0	0	0	4	0	0	0	0	4	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	4	1	1
consent	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
boards	0	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0
dome	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
obtain	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	1	0	0
leveraged	0	0	3	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0
comply	1	1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	1	0	0
phoenix	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	1	0	0	0	0	0
core	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	1	0	0	0	0
manufactures	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0
midnight	0	0	0	1	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0
islands	0	0	0	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0
axp	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	1	0	0	0
attractive	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	4	0	0	0	0	0
undisclosed	0	0	0	0	2	0	0	0	0	1	0	0	4	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	24	4	0	0	0
acquisition	2	0	1	0	0	5	0	0	0	1	1	1	9	2	2	1	2	0	0	0	0	0	0	0	0	0	1	0	4	60	2	0	0
twa	14	2	0	1	0	4	1	0	0	0	2	0	2	0	4	0	0	0	1	0	1	0	0	0	0	0	0	0	0	2	20	1	1
interested	1	0	0	0	1	3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	6	0
trans	3	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	6

One could also employ hierarchical clustering to further analyse the relationships between terms as well as extract features. With hierarchical clustering, features can be extracted by choosing a dissimilarity threshold (cut) and picking the subtrees under the threshold as the reduced feature set. Figure 3.5 shows the dendrogram for a complete-link clustering of Table 3.2. The rectangles enclosed the feature clusters for a dissimilarity threshold of 38. As with k-means, the main words are singled out and words that seem intuitively related such as “twa” and “usair” or “designs” and “manufactures” are linked. The choice of threshold value in this kind of approach important. In the example, a threshold of 18 would create two extra features, one for the word “undisclosed” and one for the group “twa” and “usair”.

Approaches to feature (term) extraction in text categorisation have employed hierarchical and flat clustering, similarly to the strategy described

Table 3.2: K-means ($k = 5$) clustering of the words in Table 3.1

Cluster	elements
1	stake
2	usair, merger, twa
3	acquisition
4	acquire
5	voting, buyout, santa, ownership, rospatch, rexnord, designs, pie, recommend, definitive, piedmont, consent, boards, dome, obtain, leveraged, comply, phoenix, core, manufactures, midnight, islands, axp, attractive, undisclosed, interested, trans

above, as well as other unsupervised techniques such as latent semantic indexing (LSI) and principal component analysis (PCA). Li and Jain (1998) experiment with PCA and complete-link hierarchical clustering and report promising results, though some consider the size of the corpus used too small to produce conclusive results (Sebastiani, 2002). Li and Jain (1998) employ hierarchical clustering to generate word groups (through thresholding, as in the example above) which are taken to represent the reduced feature set. (Manning and Schütze, 1999b, ch. 15) describe the use of LSI to project a high dimensional vector space of words onto a reduced space of independent dimensions by applying singular value decomposition to the original document matrix.

PCA is similar to LSI in that it also employs singular value decomposition. Principal components are found by performing eigenvalue decomposition of the covariance matrix constructed from the training data. Dimensionality reduction is achieved by re-expressing the original data as a small number of principal components, i.e. those with the largest associated eigenvalues.

The advantage of unsupervised learning for dimensionality reduction is that no labelling information is needed to generate a reduced training set. This is particularly important if the text categorisation process itself can only rely on a small number of labelled documents, as is the case in active learning (Davy and Luz, 2007).

3.3.2 Word senses revisited

As a final example, we revisit the problem of word sense disambiguation, which was approached from a supervised learning perspective in lecture 2. Given concordances of the kind shown in Figure 2.7, one can construct co-occurrence vectors of the kind used above for feature extraction, or document

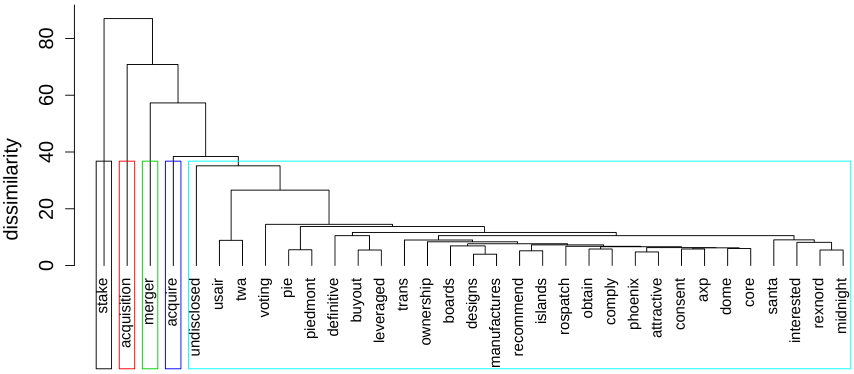


Figure 3.5: Hierarchical clustering (complete-link) of the words in Table 3.1

(i.e. concordance line) by word position vectors, encoding the position of each word relative to the keyword, or simply Boolean vectors, encoding word occurrence per line. Assume, for simplicity, that the last option has been adopted, and that each concordance line is identified by a symbol `finN` or `rivN`, where `N` is an integer and `finN` identifies the financial institution sense whereas `rivN` identifies the river sense of the word "bank".

Applying k-means clustering to a set of 35 concordances[3] yields (with some luck, if the algorithm manages to avoid converging to a bad local minimum) the two clusters shown in Table 3.3.

Table 3.3: K-means clustering of senses of the word "bank"

1	fin1, riv3, fin4, fin6, fin7, fin9, fin10, riv15, riv16, fin19, fin20, fin22, fin23, fin24, fin25, fin26, riv27, fin28, fin29, fin32, fin33, fin34
2	riv2, riv5, riv8, riv11, riv12, riv13, riv14, riv17, riv18, riv21, riv30, riv31, riv35

Single-link hierarchical clustering produces the dendrogram of Figure 3.6, which is displayed with leaves hanging nearer the branches for clarity.

Even though these examples cover a very small amount of data and use a very basic form of representation, they hopefully show that the unsupervised approach to word sense disambiguation is viable. In fact, unsupervised techniques have been employed in this area (Yarowsky, 1995) and figure still among the best performing disambiguation methods.

[3] The full set is at `http://www.cs.tcd.ie/~luzs/esslli07/data/bankselected.csv`

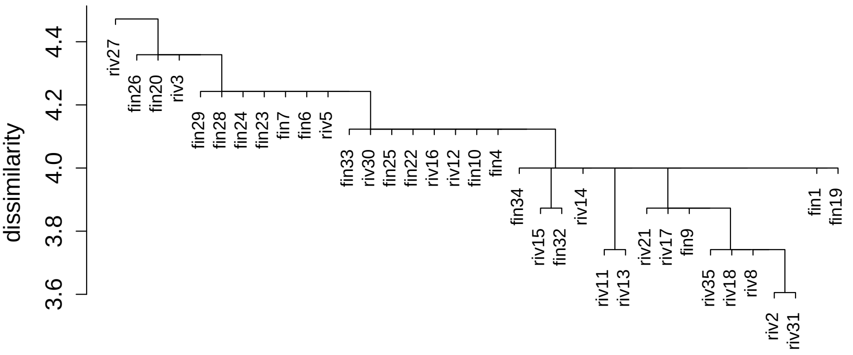


Figure 3.6: Hierarchical clustering (single-link) of senses of the word "bank"

3.4 Practical activities

Consider creating a program (perhaps using the modnlp-tc[4] libraries) to generate a co-occurrence matrix for the entire REUTERS-21578 corpus, employing clustering to extract a reduced set of features, and running the classifier on a test set represented according to this extracted feature set. Compare the results with results for the same documents represented as vectors of the same number of features obtained through supervised feature set reduction (term filtering).

An easier activity might be to experiment with clustering different encodings and larger concordance line sets for the word "bank" and see if the resulting clusters capture the different senses.

[4] `http://modnlp.berlios.de/tc.html`

Lecture 4

Learning Sequence-to-Sequence Maps: Hidden Markov Models

Sequence-to-sequence mappings arise in a variety of settings. In speech-recognition, one sequence consists of acoustic data, the other consists of phones and/or words. In optical character recognition, one sequence consists of optical data from scanning, and the other sequences consists of letters and/or words. In POS-tagging[1], one sequence consists of words, the other of POS tags.

Hidden Markov Models are one approach widely used in the machine-learning of such mappings, and this lecture will attempt to provide an introduction to this, giving some emphasis to the use of this technique in the case of POS-tagging.

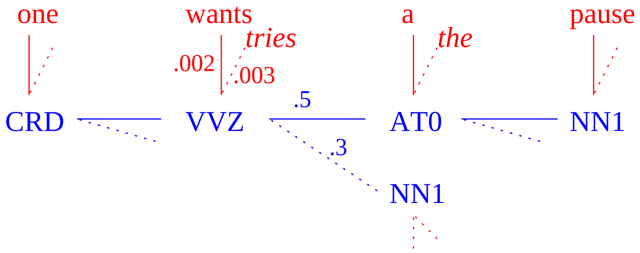
4.1 HMMs: the basic definitions

A **Markov model** is defined by a set of states, a probabilistic transition relation between the states and an initial-state probability assignment function, giving the likelihood of starting in a given state.

A **Hidden Markov model** (HMM) is defined by having an additional probabilisitic output function, according to which observation symbols are output from a state.

The picture below shows a sub-part of an example of a HMM:

[1] **P**art **o**f **S**peech tagging



In this example, the states (shown in blue) of the HMM are part-of-speech tags. The observation symbols (shown in red) are words. The blue lines indicate transitions from state to state, and red lines indicate observation symbols that are output from a state.

A few of the probabilities are shown. For example amongst the possible successors of the VVZ state are AT0, with probability 0.5, and NN1, with probabilit 0.3. Amongst the possible observation symbols in state VVZ are wants, with prob 0.002, and tries, with prob 0.003.

Fixing a notation, we will use

set of possible hidden states represented by integers $1 \ldots N$

$\mathcal{S}$: for a sequence of hidden states

$\mathcal{O}$: for a sequence of observation symbols

π: assigning to i the probability of starting in i

A: assigning to i and j the probability $P(s_t = j | s_{t-1} = i)$, abbreviated a_{ij}

B: assigning to i and observation symbol k the probability $P(o_t = k | s_t = i)$, abbreviated $b_i(k)$

An HMM fundamentally assigns a **joint probability** to a state+observation sequence:

$$P(\mathcal{O}_{1,T}, \mathcal{S}_{1,T}) = P(s_1)P(o_1|s_1) \times \prod_{t=2}^{T} P(o_t|s_t)P(s_t|s_{t-1}) \tag{4.1}$$

A sequence-to-sequence mapping can be defined on the basis of this by seeking the most probable hidden state sequence for given sequence of visible observations – conventionally referred to as **decoding**. In the case of HMM's used for POS-tagging task, this amounts to making the best guess for a tag sequence given a sequence of words. The equations for this are:

$$
\begin{aligned}
decode(\mathcal{O}_{1,T}) &= \arg\max_{\mathcal{S}_{1,T}} P(\mathcal{S}_{1,T}|\mathcal{O}_{1,T}) && (4.2)\\
&= \arg\max_{\mathcal{S}_{1,T}} P(\mathcal{S}_{1,T}, \mathcal{O}_{1,T})/P(\mathcal{O}_{1,T}) && (4.3)\\
&= \arg\max_{\mathcal{S}_{1,T}} P(\mathcal{S}_{1,T}, \mathcal{O}_{1,T}) && (4.4)
\end{aligned}
$$

with the second equation coming via the definition of conditional probability, and the third because the denominator is constant across the compared $\mathcal{S}_{1,T}$.

Relevant later to the issue of unsupervised training is the fact that on the basis of the joint probability, a probability is defined for an observation sequence:

$$P(\mathcal{O}_{1,T}) = \sum_{all\ \mathcal{S}_{1,T}} P(\mathcal{O}_{1,T}, \mathcal{S}_{1,T}) \tag{4.5}$$

So at the highest level, HMMs approach sequence-to-sequence mappings via a *joint probability* of observation+state sequences, on the basis of which a most-probable hidden-state sequence is defined for a given observation sequence. This is to be distinguished from what are sometimes called *discriminative* approaches, in which a *conditional probability* of a target sequence given input sequence is directly modelled.

4.2 Best path through an HMM: Viterbi Decoding

Given an HMM, the most likely use for it is in assigning a most-probable hidden state sequence to a given sequence of visible observations. 4.4 gave the equation for this **decoding** task, which re-expressed in terms of π, A and B is:

$$decode(\mathcal{O}_{1,T}) = \arg\max_{\mathcal{S}_{1,T}}[(\pi(s_1)b_{s_1}(o_1) \times \prod_{t=2}^{T} b_{s_t}(o_t)a_{s_{t-1}s_t})]$$

If there are N different possible states, then the number of possible state sequences to be compared is N^T, so it is not computationally feasible to simply enumerate all the possible state sequences.

The **Viterbi** algorithm is an efficient method for avoiding the enumeration of each possible path. Let $best\_path(t, i)$ be the best-path through the HMM accounting for the first t observation symbols and which ends with hidden state i. Note the parameter i, giving the hidden state at the end of the path. A different function would be $abs\_best\_path(t)$, simply giving the best-path through the HMM accounting for the first t observation symbols. Clearly

Algorithm 4.1: Viterbi

```
Initialisation :

for ($i = 1$; $i \leq N$; $i$++) {
 $best\_path(1, i) = \pi(i) b_i(o_1)$;
}

Iteration :

for ($t = 2$; $t \leq T$; $t$++) {
   for ($j = 1$; $j \leq N$; $j$++) {
    $max = 0$;
    for ($i = 1$; $i \leq N$; $i$++) {
     $p = best\_path(t-1, i).prob \times a_{ij} b_i(o_t)$;
     if ($p > max$)
        { $max = p$; $prev\_state = i$; }
    }
    $best\_path(t, j).prob = max$;
    $best\_path(t, j).prev\_state = prev\_state$
 }
}
```

$$abs\_best\_path(t) = \underset{1 \leq i \leq N}{max} best\_path(t, i)$$

so having $best\_path(t, i)$ for all i is sufficient to calculate $abs\_best\_path(t)$, which is what you eventually want.

The values of $best\_path(t, .)$ can be easily calculated from the values of $best\_path(t-1, .)$. For a given state j at time t, the idea is consider every possible *immediate predecessor* i for j. For each such i the probability of the best path to i, $best\_path(t-1, i)$, is multiplied with the probability of the next step on the path, $a_{ij} b_j(o_t)$, and the maximum of these is taken to give $best\_path(t, i)$.

Its most economical to formalize and indeed implement in terms of a function giving the probability of the best-path to i at t, and a pointer back to precedecessor on the best-path: see Algorithm 4.1

The cost of this algorithm will be of the order of $N^2 T$, compared to the brute force cost N^T.

4.3 Unsupervised Learning of an HMM: the Baum-Welch algorithm

Given an observation sequence $\mathcal{O}$ there is a method to iteratively refine the parameters of an HMM known as **Baum-Welch re-estimation** ((Baum et al., 1970)).

For the observation sequence $\mathcal{O}$, the model assigns a probability to every path through the model that is consistent with $\mathcal{O}$, from the very likely paths, to every unlikely ones. For any event on a path involving the hidden states, these probabilities give rise to an *expected* count for that event. For example, there will be an expected count for an ij transition: $E(ij)$. To calculate it, in principle you just need to consider each path in turn, and add $N \times p$ to a running total, where N is the number of times ij occurs on the path, and p is the probability of the path (given $\mathcal{O}$). Similarly there will be an expected count for the occurence of i on a path, without regard to its successor: $E(i)$. From these two expected counts, the transition probability a_{ij} can be *re-estimated*

$$\hat{a}_{ij} = E(ij)/E(i)$$

Something similar can in principle be done to re-estimate every parameter of the model. Essentially the Baum-Welch procedure consists of *iterating* this re-estimation of model parameters till a fixed-point is reached.

It is an instance of an **E**xpectation/**M**aximisation algorithm (Dempster et al., 1977), alternating between an **E** phase, using the observed data and current parameters to calculate expectations for events involving the hidden states, and a **M** phase, which re-estimates parameters to maximise the likelihood of the distribution of event expectations just obtained – which in this case consists of calculating the conditional probabilities by taking ratios in the natural way.[2]

If an interation of re-estimation transforms the parameters from λ to λ', it can be shown that

$$P_{\lambda}(\mathcal{O}) \leq P_{\lambda'}(\mathcal{O})$$

so the revised model finds the observations *more likely* than before.

Idealising for a moment, if the observation sequence $\mathcal{O}$ is really a sequence generated by the original model, λ, then in the limit as you make the se-

[2] The K-means algorithm seen in Chapter 3 is an instance of an algorithm from the same family, alternating between an assignment phase, using current means $f_1 \dots f_k$, to generate clusters $c_1 \dots c_k$, and an update phase, generating new means $\hat{f}_1 \dots \hat{f}_k$ by taking the centroids of the clusters $c_1 \dots c_k$

quence longer and longer, this re-estimation procedure will return unchanged parameters.

Another possibility is that the observation sequence $\mathcal{O}$ is really a sequence generated by a model λ' which is different to the initial model λ. In this case the re-estimation procedure will generate a new model which is a better approximation of λ' than the initial model.

There are some points to be borne in mind about what may seem a somewhat magical procedure. Firstly, if the observation sequence is not long enough, the re-estimation will tend to give a poor performance on unseen data – the classic over-fitting problem. Secondly, there may be many local maxima in the space of possible models, so there is no guarantee that the re-estimation will converge to *the* best model. Thirdly, though the re-estimation is mathematically guaranteed to maximise the probability of the observation sequence, what one is probably seeking is to maximise the correctness of best-path choice on unseen data.

Setting aside these caveats, there is a more pressing practical problem in the way of carrying out the re-estimation procedure literally as stated: with an observation sequence of length T, and N states, there are N^T paths, so its prohibitively computationally expensive to compute the expected counts via an emumeration of possible paths.

This problem is cleverly solved by essentially splitting the overall expected counts into T individual clock-tick versions. So instead of an overall expected count for an ij transition, you have for each clock tick t, a probability for having an ij transition at that clock tick. Summing over all clock ticks, then gives back the overall expected count. These clock-tick versions are determined by a pretty ingenious bit of dynamic programming, which relies on a factorisation of the calculations into two functions, α and β, with α working *forward* through time and β *backwards*. At each time tick, these two functions can be combined to give a joint probability of the observatoins and an event at that time tick.

$\alpha_t(i)$ is a function to give the probability of being in state i having emitted the observation symbols $O_{1,t}$:

$$\alpha_t(i) = P(o_1 \ldots o_t, s_t = i)$$

$\beta_t(i)$ is a function to give the probability of emitting observation symbols $O_{t+1,T}$, given being in state i at time t:

$$\beta_t(i) = P(o_{t+1} \ldots o_T | s_t = i)$$

Taking some intermediate time t in the observation sequence, multiplying

$\alpha_t(i)$ and $\beta_t(i)$, gives[3]:

$$\alpha_t(i)\beta_t(i) = P(o_1 \dots o_t, x_t = i, o_{t+1} \dots o_T)$$

α is refered to as the 'forward' probability, and β as the 'backward' probability. The recursive definitions of α and β (which explain the names), and the remaining formulae for the re-estimation procedure are given in Table 4.1

'forward probability' $\alpha_t(i)$		$= P(o_1 \dots o_t, s_t = i)$
'backward probability' $\beta_t(i)$		$= P(o_{t+1} \dots o_T \mid s_t = i)$
$\alpha_t(i)\beta_t(i)$		= the joint prob of the obs and $s_t = i$
		$= P(o_1 \dots o_t, x_t = i, o_{t+1} \dots o_T)$
$\gamma_t(i)$	= the probability of state i at t given $\mathcal{O}$	
	$= \alpha_t(i)\beta_t(i)/P(\mathcal{O})$	
$\xi_t(i,j)$	= the probability of transition ij at t given $\mathcal{O}$	
	$= \alpha_{t-1}(i)a_{ij}b_j(o_t)\beta_t(j)/P(\mathcal{O})$	
$P(\mathcal{O})$	$= \sum_i \alpha_t(i)\beta_t(i)$ (any t can be chosen)	
restimation for a transition	$\hat{a}_{ij}$	$= \frac{\sum_{t=2}^T \xi_t(i,j)}{\sum_{t=1}^T \gamma_t(i)}$
restimation for output	$\hat{b}_j(k)$	$= \frac{\sum_{t=1}^T \|_{o_t=k}\gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)}$
Recursion for β		
base	$\beta_T(i) = 1$	
recursive	$\beta_t(i) = \sum_{i=1}^N a_{ij}b_j(o_{t+1})\beta_{t+1}(j), \quad \text{for } t = T-1, \dots, 1$	
Recursion for α		
base	$\alpha_1(i) = \pi(i)b_i(o_1)$	
recursive	$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i)a_{ij}b_j(o_t), \quad \text{for } t = 2, \dots, T$	

Table 4.1: Formulae involved in re-estimating HMM probabilities from an observation sequence $\mathcal{O}$ (the Baum-Welch algorithm)

Forbidding though the re-estimation formuale may seem, each says something relatively straightforward. For example, the re-estimation formula for $\hat{a}_{ij}$ is simply taking a ratio of the expected count for ij events to i events.

The formulae given are for the case of a single, presumably very long, observation sequence $\mathcal{O}$. The expressions for $\gamma_t(i)$ and $\xi_t(i,j)$ have a $P(\mathcal{O})$ denominator, which is not strictly necessary, as always ratios are taken. However, there is a generalisation to the case of multiple observations sequences, $\mathcal{O}^1 \dots \mathcal{O}^Q$, in which these denominators matter. In this case, re-

[3] this relies on the fact that $P(o_{t+1} \dots o_T \mid s_t = i) = P(o_{t+1} \dots o_T \mid o_1 \dots o_t, s_t = i)$

estimation formulae for $\hat{a}_{ij}$ and $\hat{b}_j(k)$ have numerators and denominators in which there is a summation over each observation sequence.

$$\hat{a}_{ij} = \frac{\sum_q E^{\mathcal{O}^q}(ij)}{\sum_q E^{\mathcal{O}^q}(i)} = \frac{\sum_q \sum_{t=2}^{T_q} \xi_t^{\mathcal{O}^q}(i,j)}{\sum_q \sum_{t=1}^{T_q} \gamma_t^{\mathcal{O}^q}(i)}$$

$$\hat{b}_j(k) = \frac{\sum_q E^{\mathcal{O}^q}(ik)}{\sum_q E^{\mathcal{O}^q}(i)} = \frac{\sum_q \sum_{t=1}^{T_q} |_{o_t=k} \gamma_t^{\mathcal{O}^q}(j)}{\sum_q \sum_{t=1}^{T_q} \gamma_t^{\mathcal{O}^q}(j)}$$

4.3.1 A worked example of the (brute-force) re-estimation algorithm

Suppose an HMM is initialised with probabilities deriving from the relative frequencies in the sample below:

the/AT wins/N stopped/V ./STOP (= tags0)
the/AT cup/N wins/N ./STOP (= tags2)
it/PRO wins/V the/AT cup/N ./STOP (= tags3)

then the following tables for π, A and B would be obtained:

π	A	B
$P(s_1 = AT) = 0.666$	$P(N\|AT) = 1$	$P(the\|AT) = 1$
$P(s_1 = PRO) = 0.333$	$\mathbf{P(V\|N) = 0.25}$	$\mathbf{P(wins\|N) = 0.5}$
	$\mathbf{P(N\|N) = 0.25}$	$P(cup\|N) = 0.5$
	$\mathbf{P(STOP\|N) = 0.5}$	$P(it\|PRO) = 1$
	$P(V\|PRO) = 1$	$P(.\|STOP) = 1$
	$\mathbf{P(STOP\|V) = 0.5}$	$P(stopped\|V) = 0.5$
	$P(AT\|V) = 0.5$	$\mathbf{P(wins\|V) = 0.5}$

Whilst the observed symbols, $\mathcal{O}^0$ and $\mathcal{O}^2$ of $tags0$ and $tags3$ each have only possible tagging, for the observed symbols $\mathcal{O}^1$ of $tags2$ there are two possible taggings:

the/AT cup/N wins/V ./STOP (= tags1)
the/AT cup/N wins/N ./STOP (= tags2)

which are exactly equiprobable, which can be seen by considering the ratio of their probabilities:

$$\frac{P(tags1)}{P(tags2)} = \frac{P(V|N)P(wins|V)P(STOP|V)}{P(N|N)P(wins|N)P(STOP|N)} \tag{4.6}$$

Another way of saying this is that the conditional probabilities for each tag sequence given its observation sequence are as follows

$$P(tags0|\mathcal{O}^0) = 1 = P(tags3|\mathcal{O}^2) \quad P(tags1|\mathcal{O}^1) = 0.5 = P(tags2|\mathcal{O}^1)$$

Now suppose a training corpus for re-estimation has the observation sequences $\mathcal{O}^0$, $\mathcal{O}^1$ and $\mathcal{O}^2$ in the ratios $1:1:10$. Assuming the tag sequences are distributed in the training corpus according to the above probabilities, the expectations and derived re-estimated probabilities are (considering just the factors contributing to $P(tags1)$ vs $P(tags2)$):

π

$E(s_1 = AT) = 2 \quad P(s_1 = AT) = 2/12$

A

$E(AT, N) = 12$	$E(AT) = 12$	$P(N\|AT) = 12/12$
$E(N, V) = 1.5$	$E(N) = 12.5$	$\mathbf{P(V\|N) = 1.5/12.5}$
$E(N, N) = 0.5$		$\mathbf{P(N\|N) = 0.5/12.5}$
$E(N, STOP) = 10.5$		$\mathbf{P(STOP\|N) = 10.5/12.5}$
$E(V, STOP) = 1.5$	$E(V) = 11.5$	$\mathbf{P(STOP\|V) = 1.5/11.5}$
$E(V, AT) = 10$		$P(AT\|V) = 10/11.5$

B

$E(AT, the) = 12$	$E(AT) = 12$	$P(the\|AT) = 12/12$
$E(N, wins) = 1.5$	$E(N) = 12.5$	$\mathbf{P(wins\|N) = 1.5/12.5}$
$E(N, cup) = 11$		$\mathbf{P(cup\|N) = 11/12.5}$
$E(STOP, .) = 12$	$E(STOP) = 12$	$P(.\|STOP) = 12/12$
$E(V, stopped) = 1$	$E(V) = 11.5$	$P(stopped\|V) = 1/11.5$
$E(V, wins) = 10.5$		$\mathbf{P(wins\|V) = 10.5/11.5}$

Plugging these numbers into 4.6, the result is that now $P(tags1) > P(tags2)$ (ie. the V tagging of *wins* gets preferred):

$$\frac{P(the/AT\ cup/N\ wins/V\ ./STOP)}{P(the/AT\ cup/N\ wins/N\ ./STOP)} = \frac{1.5/12.5 \times 10.5/11.5 \times 1.5/11.5}{0.5/12.5 \times 1.5/12.5 \times 10.5/12.5} = \frac{0.0143}{0.0040}$$

Further iterations will magnify this trend.

4.4 Empirical Outcomes on re-estimating HMMs

HMMs and their associated algorithms were developed by researchers working the speech-recogition, and it is fair to say that is there that they have achieved their most notable successes. In this section we note some of the findings in applying these techniques to the problem of POS-tagging.

Merialdo (1994), Elworthy (1994) and Wang and Schuurmans (2005) all report work evaluating *unsupervised* learning of HMMs for POS-tagging.

Merialdo's work was based on an annotated corpus, with approx 40k sentences used for training (stripped of their annotation) and a disjoint 2k tagged sentences for evaluation. For the output probabilities, B, $b_i(w)$ was initialised to be uniform for each w which appears at least once with tag i

in the full annotated corpus. This amounts to assuming just that one knows for which (i, w), $P(i, w) = 0$. The transition probabilities a_{ij} were set to be uniform for all ij. The Baum-Welch procedure reduced an initial error rate from 23% to 13%.

Wang et al's work is a replication of this for the PTB, with approx. 45k sentences used for training, and a disjoint subset used for testing, approximately 1/10th the size of the training set. With the same initialisation as Merialdo, which they term the *full lexicon* condition, the Baum-Welch algorithm reduced the error rate from 34% to 18.7%. They also considered the *ablated lexicon* condition, where $b_i(w)$ is additionally given non-zero values only for those i and w for which $P(i|w) > 0.1$. The error rate then at the end of the Baum-Welch re-estimation is 6.2%.

Arguably, these figures seem quite good, for an unsupervised method, and it is interesting to see that the re-estimation did increase the accuracy, even though it is only mathematically guaranteed to increase the observed sequence likelihood. On the other hand, both the full and ablated lexicon initialisations of B make use of annotated data in the PTB, in fixing the initial zeroes of this distribution.

Another finding of Merialdo (1994) and Elworthy (1994) was that running the unsupervised Baum-Welch procedure on an HMM obtained by supervised training tends to lower the accuracy, at least if the amount of training data for the supervised training is substantial. Elworthy (1994) investigates also in more detail the situations in which an initial maximum might occur, or a maximum after a few iterations.

As a final finding concerning HMM re-estimation, Kazama et al. (2001) report work in a tagging architecture in which unsupervised HMM training is a component and claim that, especially in the case where only a small amount of annotated data is available, the HMM re-estimation procedure does boost their overall performance.

4.5 Supervised learning and higher order models

In the *supervised* variant of HMMs, direct evidence for events involving the hidden states is assumed to be involved. This might be evidence purely about the hidden state sequences: in the application of HMMs to speech recognition, this might take the form simply of text, providing evidence for word-to-word transitions. For POS tagging, this evidence is typically a POS-tagged corpus, providing direct evidence thereby for the tag-to-tag and tag-to-word distributions.

As an example of such an approach, Brants (2000) uses the PTB for training. The equation underlying the model is a slight variant of 4.1, according to which a state is condition on the preceding *two* states

$$P(\mathcal{O}_{1,T}, \mathcal{S}_{1,T}) = P(s_1)P(o_1|s_1)P(s_2|s_1)P(o_2|s_2) \times \prod_{t=3}^{T} P(o_t|s_t)P(s_t|s_{t-2}s_{t-1}) \tag{4.7}$$

A 96.7% accuracy is achieved. This performance is typical of that attained by a number of people with broadly similar approaches.

4.5.1 Sparsity, Smoothing, Interpolation

Inevitably, even with a large corpus, the real probabilities are misrepresented by the observed relative frequencies. One aspect of this is words that were not seen in the training data, for which simple counting will set the estimate for output B probabilities to 0. Another aspect of this is POS trigrams whose raw counts may misrepresent reality, the extreme case of which is trigrams xyz with 0 counts. In the above described model, the simple counting will set the estimated transition A probabilities $P(z|xy)$ for these to 0.

Techniques to combat this situation – widely refered to as *data sparsity* – go under the headings of *smoothing* and *interpolation*. Smoothing refers to methods which raise the probability of the apparently impossible, while inevitably lowering the probability of the apparently frequent, and there are very many of these. Interpolation refers to methods which basically combine models which use successively less conditioning, and again there are many methods for this. Space precludes making more than few cursory remarks about this: Chen and Goodman (1998) gave a comprehensive survey.

Interpolation

For the state transition probabilities, one approach is to combine a raw count-based estimate – $\hat{P}_0(z|xy)$ – with estimates of the probability of z with less conditioning information, specifically with raw count-based estimates $\hat{P}_1(z|y)$ and $\hat{P}_2(z)$. In Brants (2000) the HMM's A parameters are defined as a *linear interpolation* of these:

$$A(xyz) = \lambda_0\hat{P}_0(z|xy) + \lambda_1\hat{P}_1(z|y) + \lambda_2\hat{P}_2(z)$$

such that λs sum to 1.

For the case of 0-count trigrams, the above equation will set the HMM's A parameter not to 0, but to a combination of the probabilities based on bigrams and unigrams.

There have been quite a number of proposed methods for setting the values of the λs in this kind of interpolation to estimate an HMM's transition probabilities. One method, which ties in with what was said earlier concerning unsupervised HMM training, is a variant of the Baum-Welch algorithm due to Jelinek and Mercer (1980) to allow the values for the λs to be automatically learned from the annotated data: the hidden variables in this case become not the state transitions, but instead the interpolation coefficients.

Another, simpler method takes each triple xyz in turn, determines which of $\hat{P}_0(z|xy)$, $\hat{P}_1(z|y)$ and $\hat{P}_2(z)$ is the greatest, *based on the corpus with the occurrence of the triple deleted*. Whichever is greatest counts as a vote of 1 for the relevant λ. Once the whole corpus has been processed in this way the accumulated votes for the λs are used to set the λs in the equation. This is the method used in the Brants (2000) tagger mentioned earlier.

There are versions where the λs are not constants of the model, but instead have values depending on the conditioning event. There is a *Witten-Bell* variant of this, details of which will be given in the lecture on probabilistic grammars, Chapter 5.

Backoff

In so-called *Backoff* approaches the idea of using a succession of less and less specific events is also used, but in a subtley different way. If there are non-zero counts available for a specific event, then the estimate is based on counts for that event, and the less specific events are ignored. On the other hand, if there are zero counts for the specific event, the estimate is entirely based on its less specific sub-events: one *backs-off* to the next, less specific event.

Interwoven with this is the notion of *discounting*: a non-zero count C for a specific event is not taken at face-value, but subjected instead to some kind of discounting, to give C^*, in such a way as to leave aside an expected count for all the unseen events, treated as a single class. Basically the rationale of backoff methods is to distribute this expected count for the unseen event class over the different specific unseen events according to the probabilites of their less specific sub-events.

For example, a backoff version of transtion probabilities might have

$$P_{back}(y|x) = \begin{cases} C^*(xy)/C(x) \text{ if } C(xy) > 0 \\ \alpha(x)C(y)/N \text{ otherwise} \end{cases}$$

where $\alpha(x)$ is set to make sure that the probabilities sum to 1. The formulae for these back-off methods become quite intricate, so we cannot pursue this much further in this context, save at least to describe the best known method for obtaining the discounted counts C^*: the *Good-Turing* method. Where $r = C(X)$, where X is some event, and n_r is the number of events also with frequency r, the Good-Turing value for $C^*(X)$ is

$$C^*(X) = (r+1)\frac{n_{r+1}}{n_r}$$

Summing the probabilities based on these discounted C^* scores does not give 1, and this corresponds to an amount of probability that has been set aside for events which had 0 observed counts. The probability that the next observation will be a novel event is effectively estimated by this method to be

$$n_1/N$$

The table below is an excerpt from Church and Gale (1991), which reports work which shows how well this Good-Turing method seems to work

r	n_r	f_{emp}	f_{GT}
1	$2,018,046$	0.448	0.446
2	$449,721$	1.25	1.26
3	$188,933$	2.24	2.24
$\vdots$	$\vdots$	$\vdots$	
8	$37,710$	7.21	7.24
9	$22,280$	8.26	8.25

Each bigram from a 44 million word corpus was randomly assigned to one of two sets, $\mathcal{C}_1$ and $\mathcal{C}_2$. The r column gives possible values taken on by bigram frequencies in the first of these sets $\mathcal{C}_1$. Many different bigrams could share a particular frequency and n_r gives the number of different bigrams in the first set $\mathcal{C}_1$ with frequency r, which decreases as r increases. Under f_{emp} is an empirical value for the average observed frequency in the second set of bigrams $\mathcal{C}_2$ for a bigram x occurring r times in the 1st set $\mathcal{C}_1$. Under f_{GT} is the Good-Turing estimate for this quantity.

For the observation probabilities, B, of an HMM, the same interpolation and backoff techniques might be applied. The very simplest approach is to treat all unknown words as alike, and for each tag x to find an estimate for

$$P(UNK|x)$$

The Good-Turing perspective on this leads to estimating this from the observations seen only once – the so called *hapax legomena*

$$\hat{P}(UNK|x) = \hat{P}(HAPAX|x)$$

The natural notion of sub-event in this case are spelling features of the word, such as capitalisation or suffixes. These are for example combined by Weischedel et al. (1993) to give a estimate of the form

$$\hat{P}(w \in UNK|x) = \hat{P}(HAPAX|x) \times \hat{P}(endings(w)|x) \times \hat{P}(caps(w)|x)$$

The approach taken in the Brants tagger above is to estimate $P(x|suffix)$ for some chosen longest suffix via interpolating estimates for a succession of smaller suffixes, and then finally to use Bayesian inversion to obtain $P(suffix|x)$.

4.6 Further Reading

HMMs, and the machine learning of them, are a foundation of much work in speech recognition and optical character recognition. For some of the further details see Jurafsky and Martin (2000). Further details of the application of HMM to POS-tagging can also be found in the same reference. HMMs have been used for Named-Entity Recognition (eg. Zhou and Su, 2002), and for Information Extraction (eg. Freitag and McCallum, 1999; Gu and Cercone, 2006). They have even been put forward as models of psycholinguistic phenomena, such as garden-path effects (eg. Park and Brew, 2006).

The decoding equation for an HMM can be expressed

$$\begin{aligned} decode(\mathcal{O}_{1,T}) &= \arg\max_{\mathcal{S}_{1,T}} P(\mathcal{O}_{1,T}, \mathcal{S}_{1,T}) \\ &= \arg\max_{\mathcal{S}_{1,T}} P(\mathcal{S}_{1,T}) \times P(\mathcal{O}_{1,T}|\mathcal{S}_{1,T}) \end{aligned}$$

and so in a general sense, HMM are related to the Bayesian approach to classification described in earlier lectures. This Bayesian, or 'noisy-channel' perspective is also the basis of stochastic Machine translation, where French f is 'decoded' to English e by

$$\arg\max_{e} P(f, e) = \arg\max_{e}(P(e) \times P(f|e))$$

For further information about these various sequence-to-sequence problems and the techniques that have been proposed to handle them see Jurafsky and Martin (2000) and Manning and Schütze (1999a).

Lecture 5

Learning Sequence-to-Tree Maps: Probabilistic Grammars

5.1 The PCFG Model

A wide variety of probabilistic grammars have been the focus of intensive investigation in the last 20 years. The simplest model is the Probabilistic Context Free Grammar (PCFG), which augments a CFG by (i) a function A assigning probability to each grammar rule, with the constraint that the probabilities assigned to the expansions of a particular category sum to 1[1]

$$\sum_{rhs} A(M \rightarrow rhs) = 1$$

and (ii) a function π assigning an initial probability to each category, quantifying its likelihood of being a root of a tree[2]. Under this model, the probability of a tree is a product of the probabilities of the rules used (and root category): see Figure 5.1 for an example.

Treating tree probability this way represents independence assumptions. In particular looking at a local tree $M \rightarrow rhs$, probability of daughters rhs is independent of everything above, left and right of the mother M.

The PCFG model assigns a probability to a complete tree, words included. The parsing task is then conceptualised as that of finding the most probable tree, given the words

[1] and thus express a conditional probability of the dtrs given the mother

[2] this is optional

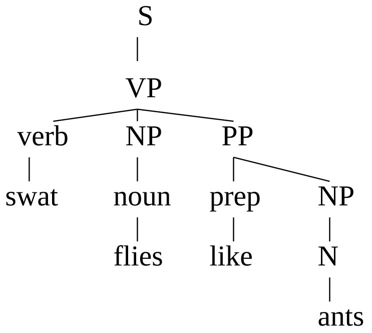
π

S	1.0

A

$S \rightarrow NP\ VP$	0.8
$S \rightarrow VP$	0.2
$NP \rightarrow noun$	0.4
$NP \rightarrow noun\ PP$	0.4
$NP \rightarrow noun\ NP$	0.2
$VP \rightarrow verb$	0.3
$VP \rightarrow verb\ NP$	0.3
$VP \rightarrow verb\ PP$	0.2
$VP \rightarrow verb\ NP\ PP$	0.2
$PP \rightarrow prep\ NP$	1.0
$prep \rightarrow like$	1.0
$verb \rightarrow swat$	0.2
$verb \rightarrow flies$	0.4
$verb \rightarrow like$	0.4
$noun \rightarrow swat$	0.05
$noun \rightarrow flies$	0.45
$noun \rightarrow ants$	0.5

$tree_1$ for *swat flies like ants* is



$$\begin{aligned} P(tree_1) &= 0.2 \times 0.2 \times 0.2 \times 0.4 \times 0.45 \\ &\quad \times 1.0 \times 1.0 \times 0.4 \times 0.5 \\ &= 2.88 \times 10^{-4} \end{aligned}$$

Figure 5.1: Illustrating the PCFG probability of a tree

$$\begin{aligned} parse(\mathcal{O}) &= \arg\max_{T} P(T|\mathcal{O}) \\ &= \arg\max_{T, yield(T)=\mathcal{O}} P(T)/P(\mathcal{O}) \\ &= \arg\max_{T, yield(T)=\mathcal{O}} P(T) \end{aligned}$$

Relevant later to the issue of unsupervised training is the fact that on the basis of the tree probability, a probability is defined for an observation sequence:

$$P(\mathcal{O}) = \sum_{all\ \ T, yield(T)=\mathcal{O}} P(T) \tag{5.1}$$

A sequence of words to be parsed can be seen as analogous to the observation sequence of an HMM. The non-terminal nodes and their linking are analogous to the hidden state sequence of an HMM.

The model of tree probability embodied by a PCFG is very simple, with nothing conditioned on anything distant – entirely analogously to HMMs. Because of this, given a PCFG, finding the most probable parse given the input is relatively straightforward to do, by a simple adaptation of a bottom-up chart-parsing algorithm, such as the CKY algorithm: just as with the Viterbi algorithm an HMMs, the key is to have each cell in the table record the best parse for each possible constituent at the point, before finally picking an overall best from the cell spanning the entire input.

The analogy with HMMs is very strong, and for grammars with only $C^i \rightarrow w$ and $C^i \rightarrow w, C^j$ type rules, the PCFG case collapses into a HMM.

Also there is generalisation of the Baum-Welch re-estimation procedure for HMMs to PCFGs (Baker, 1979). This *unsupervised* approach to parameter estimation is outlined below.

5.2 Unsupervised Learning of a PCFG: the Inside-Outside algorithm

Given an observation sequence there are all the possible trees which are consistent with it, each assigned a probability. For any event in a tree, these probabilities give rise to an *expected* count for that event. For example, there will be an expected count for the use of a particular rule $M \rightarrow D_1, \ldots, D_n$: $E(M \rightarrow D_1, \ldots, D_n)$. To calculate it, in principle you just need to consider each tree in turn, and add $N \times p$ to a running total, where N is the number of times $M \rightarrow D_1, \ldots, D_n$ occurs in the tree, and p is the probability of the tree, given the observation sequence. Similarly there will be an expected count for an occurence of M in the tree: $E(M)$. From these two expected counts, the rule probability can be re-estimated:

$$\hat{P}(D_1, \ldots, D_n|M) = E(M \rightarrow D_1, \ldots, D_n)/E(M)$$

This procedure could be iterated till a fixed-point is reached. As was the case for HMMs, this is an instance of an Expectation/Maximisation algorithm

It can be shown that if an iteration of re-estimation using $\mathcal{O}$ changes the models parameters from λ to λ' then

$$P_\lambda(\mathcal{O}) \leq P_{\lambda'}(\mathcal{O})$$

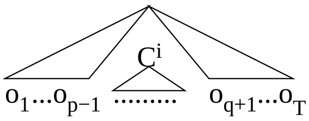
The caveats mentioned in the discussion of the Baum-Welch algorithm pertain here also: training on a small of amount of observations will give poor performance on unseen data; there will typically be local maxima, so the algorithm is not guaranteed to converge to *the* best model; the iterations increase $P(\mathcal{O})$, but you probably really want to increase $accuracy(parse(\mathcal{O}))$.

As in the case of HMMs, this procedure would be exponentially costly to carry out in practice. The **Inside-Outside** algorithm (Baker, 1979) is the feasible alternative, breaking up the expected counts into expectations per span of words, summing ultimately over all spans of words, and again analogously to the HMM case, the span-specific calculations are done by an ingenious factorisation into two functions, concerning disjoint spans, which are multiplied together.

Let C^i_{pq} mean a node C^i dominating terminals starting at position p and ending at position q.

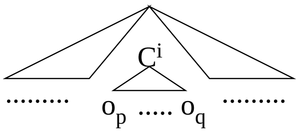
$\alpha_{pq}(i)$ — the 'outside probability' — is the probability of a tree whose terminal sequence is $o_1, \ldots, o_{p-1} X o_{q+1}, \ldots, o_T$ and has a category C^i, dominating the sequence X

$$\alpha_{pq}(i) = P(o_1, \ldots, o_{p-1}, C^i_{pq}, o_{q+1}, \ldots, o_T)$$



$\beta_{pq}(i)$ — the 'inside probability' – is the probability of tree having terminal sequence $o_p, \ldots, o_q$, given mother C^i:

$$\beta_{pq}(i) = P(o_p, \ldots, o_q|C^i_{pq})$$



$$\begin{aligned}
&\text{'outside probability' } \alpha_{pq}(i) && = P(o_1, \ldots, o_{p-1}, C^i_{pq}, o_{q+1}, \ldots, o_T) \\
&\text{'inside probability' } \beta_{pq}(i) && = P(o_p, \ldots, o_q | C^i_{pq}) \\
&\alpha_{pq}(i)\beta_{pq}(i) && = \text{the joint prob of the obs and } C^i_{pq} \\
& && = P(o_1, \ldots, o_T, C^i_{pq})
\end{aligned}$$

$$\begin{aligned}
\gamma_{pq}(i) &= \text{the probability of } C^i_{pq} \text{ given } \mathcal{O} \\
&= \alpha_{pq}(i)\beta_{pq}(i)/P(\mathcal{O}) \\
\\
\xi_{pq}(i,j,k) &= \text{the probability of the use of } C^i \rightarrow C^j C^k \text{ spanning } p,q \text{ given } \mathcal{O} \\
&= \tfrac{1}{P(\mathcal{O})}\alpha_{pq}(i)P(C^i \rightarrow C^j C^k)(\textstyle\sum_{d=p}^{q-1} \beta_{p,d}(j)\beta_{d+1,q}(k)) \\
\\
P(\mathcal{O}) &= \textstyle\sum_i \alpha_{pq}(i)\beta_{pq}(i), \quad \text{any } p, q, \ p < q
\end{aligned}$$

$$\begin{aligned}
&\text{restimation for a binary rule} \quad && \hat{P}(C^i \rightarrow C^j, C^k) && = \frac{\sum_{p=1}^{T-1}\sum_{q=p+1}^{T}(\xi_{pq}(i,j,k))}{\sum_{p=1}^{T}\sum_{q=p}^{T}(\gamma_{pq}(i))} \\
\\
&\text{restimation for a terminal rule} \quad && \hat{P}(C^i \rightarrow w) && = \frac{\sum_{p=1}^{T} |_{o_t=w}\gamma_{pp}(i)}{\sum_{p=1}^{T}\sum_{q=p}^{T}(\gamma_{pq}(i))}
\end{aligned}$$

$$\begin{aligned}
&\text{Recursion for } \beta \\
&\text{base} && \beta_{pp}(i) = P(C^i \rightarrow o_p) \\
&\text{recursive} && \beta_{pq}(i) = \textstyle\sum_{C^j C^k}\sum_{d=p}^{q-1} P(C^i \rightarrow C^j C^k)\beta_{pd}(j)\beta_{d+1,q}(k) \\
&\text{Recursion for } \alpha \\
&\text{base} && \alpha_{1T}(i) = \pi(C^i) \\
&\text{recursive} && \alpha_{pq}(i) = [\textstyle\sum_{k,j \neq i}\sum_{s=1}^{T} \alpha_{p,q+s}(k)P(C^k \rightarrow C^i C^j)\beta_{q+1,q+s}(j)] \quad (\textit{– left}) \\
& && \qquad\quad +[\textstyle\sum_{k,j}\sum_{s=1}^{p-1} \alpha_{p-s,q}(k)P(C^k \rightarrow C^j C^i)\beta_{p-s,p-1}(j)] \quad (\textit{– right}
\end{aligned}$$

Table 5.1: Formulae involved in re-estimating PCFG probabilities from an observation sequence $\mathcal{O}$ (the Inside-Outside algorithm)

Multiplying the two together gives the joint probability of the observation sequence and an occurence of C^i_{pq}:

$$\alpha_{pq}(i)\beta_{pq}(i) = P(o_1, \ldots, o_T, C^i_{pq})$$

The recursive definitions of α and β, and the remaining formulae for the re-estimation procedure are given in Table 5.1.

As for the earlier table giving the formulae for the Baum-Welch algorithm, the formulae here are for the case of a single observation sequence $\mathcal{O}$. For multiple observations sequences, $\mathcal{O}^1 \ldots \mathcal{O}^M$, in the re-estimation formulae for rule probabilities, the numerators and denominators should have a summation over each observation sequence.

$$\hat{P}(C^j, C^k | C^i) = \frac{\sum_m \sum_{p=1}^{T_m - 1}\sum_{q=p+1}^{T_m}(\xi^{\mathcal{O}^m}_{pq}(i,j,k))}{\sum_m \sum_{p=1}^{T_m}\sum_{q=p}^{T_m}(\gamma^{\mathcal{O}^m}_{pq}(i))}$$

$$\hat{P}(w|C^i) = \frac{\sum_m \sum_{p=1}^{T_m} |_{o_t=w} \gamma_{pp}^{\mathcal{O}^m}(i)}{\sum_m \sum_{p=1}^{T_m} \sum_{q=p}^{T_m} (\gamma_{pq}^{\mathcal{O}^m}(i))}$$

This procedure inherits the cubic time complexity of chart-parsing for CFGs. Since many inputs have to be parsed on an iteration of re-estimation, and there are potentially many iterations, this algorithm is still computationally intensive.

The above gives the bare bones of the re-estimation procedure. There are a number of variants on this basic set up which will be touched on below.

5.3 Empirical Outcomes on re-estimating PCFGs

What is empirically known about this re-estimation procedure ?

Artificial grammars and corpora: first of all there are experiments which have been done on artificial created data.

For example in both Pereira and Schabes (1992) and Benedi and Sanchez (2005), a PCFG for the palindrome language is assumed:

$$\begin{array}{ll}
S \rightarrow AC & [0.4] \\
S \rightarrow BD & [0.4] \\
S \rightarrow AA & [0.1] \\
S \rightarrow BB & [0.1] \\
C \rightarrow SA & [1.0] \\
D \rightarrow SB & [1.0] \\
A \rightarrow a & [1.0] \\
B \rightarrow b & [1.0]
\end{array}$$

then used to stochastically generate a training corpus, to give in turn as input to the Inside-Outside procedure. Pereira and Schabes (1992) generate a training set of 100 sentences. The back-bone CFG for the learning experiment has every possible Chomsky normal-form rule, using the symbols of the generating PCFG (135 rules), which are then assigned random weights. After 40 iterations of the Inside-Outside procedure, in their words, 'no useful solution was found' . However, in the similar experiment in Benedi and Sanchez (2005), a training set of 1000 strings was generated, and the Inside-Outside procedure iterated 340 times. They quantify the success of this as follows: 76% of randomly generated strings by the resulting grammar are palindromes. This seems to indicate that Pereira and Schabes underestimated the necessary amounts of training data and iterations.

First reported in Charniak and Carroll (1992) and discussed further in Charniak (1993) is work where an artificial, though modestly linguistically realistic PCFG was created and used to randomly generate a corpus of observation symbols (there are approx 30 rules, and about 9000 words), with a view to learning a grammar using the Inside-Outside algorithm. The rules were in a format that has been termed DEP-PCFG, which has rules of the form

$$\overline{t_i} \rightarrow \overline{t_0} \ldots \overline{t_{i-1}}\ t_i\ \overline{t_{i+1}} \ldots \overline{t_n}$$

where the t_i are the terminal symbols[3]. The back-bone CFG for the learning experiment was taken to consist of all DEP-PCFG rules, using the symbols of the generating grammar, and subject to a length 4 maximum for a right-hand side. Their finding was that in 300 runs, each time setting the probabilities to different random values, 300 different grammars were learnt. One was apparently very similar to the generating PCFG and assigned the corpus similar probability. The 299 others assigned worse probabilities. This result has been widely noted as indicating that the Inside-Outside algorithm is especially vulnerable to the problem of local maxima. See below, however, for work revisiting this, or a very similar scenario.

Realistic grammars and corpora

Moving to more realistic empirical work, Pereira and Schabes (1992) took the POS sequences of 700 sentences of the ATIS corpus as a training material, and a randomly initialised grammar with all possible CNF rules (15 non-terminals, 48 POS tags). Testing on a held out set of 70 treebank trees, in the parses from the resulting trained grammar, 35% of constituents are 'consistent' with treebank bracketing[4].

Klein and Manning (Klein and Manning, 2001, 2004; Klein, 2005)), in the context of benchmarking their own learning model, revisit the above mentioned DEP-PCFG work (Charniak and Carroll, 1992; Charniak, 1993). Klein and Manning take a binary-branching version of the DEP-PCFG format, and rather than initializing with random probabilities, initialize to a *uniform* distribution. Working with the length ≤ 10 subcorpus of the PTB[5] they report a more encouraging outcome: the trained grammar achieves F1: 48.2%, for unlabelled precision/recall[6] when evaluated against the treebank parses. They also argue that the model space represented by DEP-PCFG grammars is intrinsically a poor one, pointing out that if the model is used in a supervised setting – that is reading the probabilities from a treebank – it only reaches F1: 53.5%, which puts the unsupervised score in a different light.

[3] its called this because there is an isomorphism between these trees and dependency structures

[4] see below for their further work on semi-supervised case

[5] containing 7422 sentences

[6] F1 is $\frac{2PR}{R+P}$

5.4 Some variant applications of the Inside-Outside algorithm

5.4.1 The Consituent/Context Model

The above-mentioned reconsideraton of the re-estimation of DEP-PCFG grammars is carried out by Klein and Manning in the context of developing their own *Constituent/Context Model* (CCM). Their CCM learning procedure is an EM re-estimation procedure, but differs somewhat from the classic PCFG re-estimation procedure.

A major difference is that theirs is a model of *unlabeled* tree structure, whereas a PCFG produces *labeled* tree structures. More particularly, they work within a space of *binary-branching* unlabelled trees, which they represented indirectly with a span-table, identifying for each span (p, q), $q >= p$, whether it is a constituent or not – not being a constituent is termed being a *distituent*. Such a table[7] is termed a *bracketing*. The intuition broadly speaking seems to be that given a sequence of terminals, there are 2 likelihoods: that of being a constituent, and that of being a distituent. The model seeks to track these with inverted probabilities $P(\mathcal{O}_{pq}|B_{pq} = 1)$ and $P(\mathcal{O}_{pq}|B_{pq} = 0)$. Also given a *context* pair of sequences of terminals $(o_1 \ldots o_{p-1} \sim o_{q+1} \ldots o_T)$, there are 2 likelihoods: that these frame a constituent, and that these frame a distituent. The model seeks to track these with inverted probabilities $P(\mathcal{O}_{1,p-1} \sim \mathcal{O}_{q+1,T}|B_{pq} = 1)$ and $P(\mathcal{O}_{1,p-1} \sim \mathcal{O}_{q+1,T}|B_{pq} = 0)$.

They formulate a model of the joint probability of observations $\mathcal{O}$ and a *bracketing* B:

$$P(\mathcal{O}_{1T}, B) = P(B)P(\mathcal{O}_{1T}|B) = P(B)\times\prod_{pq} P(\mathcal{O}_{pq}|B_{pq})P(\mathcal{O}_{1,p-1} \sim \mathcal{O}_{q+1,T}|B_{pq})$$

In the term for $P(\mathcal{O}_{1T}|B)$, each span (p, q) is taken in turn, including both the constituent *and* 'distituent' spans of the bracketing.

They then use this model in an EM algorithm, first using current settings to give expectations concerning the distribution of bracketings, given the observation sequences, then using these expectations to restimate parameters, making the summation over bracketings likelier: for the further details see the appendix in (Klein, 2005).

Working with length $\leq$ 10 sub-corpus of the PTB, taking POS tags as the observation symbols. They report attaining F1: 71%, on unlabeled precision/recall[8]

[7] they have conditions to so that such a table truly does correspond to a binary tree

[8] Given the fact that PTB trees are not uniformly binary branching, the upper-bound

Some further details. At each expectation step in the induction algorithm, the term $P(B)$ is held to a constant P_{BIN}, which is uniform over over all bracketings corresponding to binary-branching trees, and 0 elsewhere. However, at the very outset of the algorithm, they take a different bracket distribution P_{SPLIT}, and use this alone in the first round of determining expectations. This is the distribution arrived at by assuming trees to be generated by a process of recursively randomly choosing split points. They claim that compared to P_{BIN}, P_{SPLIT} puts relatively less probability on 'central' spans – (p, q) where $p \neq 1$, $q \neq T$. If instead of P_{SPLIT}, P_{BIN} is used at the every outset of the algorithm, performance drops by 10% (Klein, 2005). Puzzlingly, though, they also mention that if P_{SPLIT} is used throughout:

> *It seemed to bias too strongly against balanced structures, and led to entirely linear-branching structures*

It seems fair to say that, although the ultimate performance is quite encouraging, there is a lot that remains to be understood about what is really going on this model. Space precludes further discussion, but it should be noted that the authoers develop a further probabilistic *dependency* model, and a model combining this with the CCM model, obtaining better performance (Klein and Manning, 2004; Klein, 2005).

5.4.2 Latent Tree Annotation

There have also been a number of pieces of work in which the observed/hidden status is varied from the basic set-up. The place of an observation sequence is taken by some kind of tree structure, and hidden side is taken by some kind of elaboration of that tree structure.

In one such Petrov et al. (2006) an interesting approach is described. They take a basic PCFG grammar obtained directly from a tree-bank and then try to use EM-based machine learning to infer a better grammar, in which the PTB node labels are *split* into sub-cases – they refer to these as *subsymbols*, and to a tree featuring these subsymbols as an *annotated tree*. In working predating Petrov et al. (2006), Klein and Manning (2003) carried out what is in effect a manual version of this splitting, where they decided which symbols to split, with some kind of straightforward procedure to deterministically carry out the split on all occurrences – an example might be annotating with a mother category. Their base-line model (which uses some notions which will be described more fully in Section 5.5[9]) achieved around F1:

F1 for this kind of work is not 100%: they cite 87%.

[9] Anticipating the notions of Section 5.5, their base-line model is supervised, identifies a head daughter, generates daughters outwards from it, allows conditioning on grandparent

78%, and they were able to show that judicious use of such splits gave an improved performance: F1 86.3%.

Petrov et al. (2006) differ in using machine learning techniques to decide the subsymbols, and on their distribution in annotated trees: the probabilities on rules using the subsymbols are infered by a version of the Inside-Outside algorithm in which the plain PTB trees stand in the role of the observation sequence: each plain PTB tree is regarded as the visible manifestation of a hidden, or latent, annotated tree [10].

In outline, they execute a small number of *split-merge* stages $SM_1, \ldots SM_n$, at each of which which all current labels are *split*, and then some *merged* back. In outline a split-merge stage SM_i has

- an intitialisation, where the probality of the un-split rules is shared amongst the split versions, with a small amount of noise to break symmetry
- an EM-phase repeatedly using the current probabilities and PTB trees to get expected counts for events using split symbols and then re-estimatig the split rule probalities
- a closing merge phase, whereby a newly introduced split, say A into A_1, A_2, is discarded if there is a small estimated loss in likelihood by making the merge.

They estimate the loss in likelihood of retracting the split of A into A_1, A_2, by considering each occurrence n of the merge candidate in turn and recalculating a tree probability if the split were discarded at that point $P^n(T)$. Where $P(T)$ is the likelihood of the tree under the re-estimated model given the split, the ratio $P^n(T)/P(T)$ gives an indication the loss of likelihood if not making the split at n. They approximate the overall loss in likelihood due to merging A_1 and A_2 by taking the product of this ratio for each occurrence.

Finally also they include a *smoothing* aspect, to force 'the production probabilities from annotations of the same non-terminal to be similar'. Where a re-estimation procedure gives a raw estimate $\hat{p}(A_x \rightarrow B_y, C_z)$ to a latent version of $A \rightarrow B, C$, the final estimate is obtained by linearly interpolating with the average of raw estimates for all subsymbols of A:

$$p'(A_x \rightarrow B_y, C_z) = (1-\alpha)(\hat{p}(A_x \rightarrow B_y, C_z)) + \alpha(\frac{1}{n}\sum_{x'}\hat{p}(A_{x'} \rightarrow B_y, C_z)$$

as well as parent, does not refer to inherited head features, such as words or POS tags, and uses no smoothing.

[10] A consequence of moving the algorithm to this setting is that its linear rather than cubic in the length of the sentence

They start with a PCFG with 98 symbols, directly estimated from the PTB, whose performance is poor: F1 63.4%. After 6 split-merge re-estimation rounds, where 50% of newly split symbols are merged, they obtain a grammar of 1043 symbols whose performance is very high: F1 90.2 %.

Another interesting aspect of this work is that they claim that many of the refinements learnt are linguistically interpretable, and indeed re-discover the manually created refinements of Klein and Manning (2003). For example, an original single pre-terminal category DT ends up divided into sub-symbols recognisable as definite determiners (eg. the), indefinite (eg. a), demonstratives (eg. this) and quantificational elements (eg. some). As another example, below is an excerpt from the table of the likeliest words in the sub-symbol for VBZ (third person singular verb), where the final column gives a potential linguistic ’interpretation’ of the category

VBZ-0	gives	sells	takes	ditransitives ?
VBZ-4	says	adds	Says	sentence complement, communication ?
VBZ-5	believes	means	thinks	sentence complement, cognitive ?
VBZ-6	expects	makes	calls	control verbs ?

See the cited paper for more examples.

5.5 Supervised learning and more complex models

Touched on a few times in the above was the in fact much more widespead approach of using a tree-bank to train a parsing model: the *supervised* variant of machine-learning a parsing model. This section will attempt to say something about this vast area.

The general approach is clear: define a model of tree probability, use (a subset of) tree-bank evidence to estimate the parameters of that model, evaluate the resulting most-probable parses against (a disjoint subset of) tree-bank evidence. The art is in deciding the model and the methods of estimating the parameters.

The simplest possible approach is to use the PCFG model, and estimate probabilities of rules by counting in local trees of the tree-bank, with no smoothing of the estimates: this has come to be refered to as computing the *treebank PCFG*.

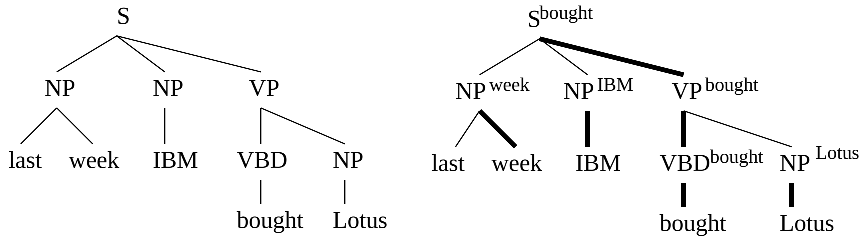
The consensus seems to be that this works surprisingly well. This is certainly the sentiment of Charniak (1996). He considers the case of the treebank with lexical items discarded – so terminals are POS tags, and reports that this approach achieves F1: 79.6 labelled precision/recall. In Klein and Manning (2001) the same is done for the subset of 10 word sentences, with a report F1 82%. For the case where the lexical items are not discarded, for example

Klein and Manning (2003) report F1 72.6%.

The reason for thinking that these numbers ought to be lower is the *data-sparsity of the treebank*. There is an expectation that the method will not work well because it would be quite likely that the best parse of a test item will need a rule that has never been seen in the training data. Collins in his thesis elaborates on this: in the training set, about 15% of trees use a rule which occurs only once, which would lead to an expectation of something similar in the test set. This missing-rules problem turns out not to be quite as punitive as might have been thought. Although it is often the case that the correct parse cannot be found (its assigned 0 probability), somewhat surprisingly a parse can be found which, although it does something very weird around the place where the unseen rule should be, elsewhere does sensible stuff. This is where the typical scoring procedures have to be borne in mind: 80% does not mean getting a completely correct parse 80% of the time, it means getting 80% of the parts of the parse right.

Two general trends can be seen in the work that has gone on to achieve parsers exceeding this treebank-grammar base-line. The first is the assumption that the treebank data can be effectively annotated with supplementary information, prior to parameter estimation, and that the parsing model should refer to this information. The second is that instead of treating a local tree as an indivisible unit, assigned a probability in its entirety, this probability should be *factorised*.

The best example of the first idea, that of supplementary annotation, is supplementation with *head* information. For each local tree, one of the daughters is designated the head daughter. Head features (such as head word, or head POS) are recursively percolated up a tree, by setting the head features of a mother equal to the head features of the head daughter. In most approaches the head daughter annotation is obtained by running hand-written head-placing heuristics over the tree-bank[11]. The tree to the right below is a head-annotated version of the tree to the left:



[11] This might be regarded as a stop-gap measure, pending either clever probabilistic ways to add the annotation, or further manual annotation of large resources. It should be noted that linguistic controversies attach to the identification of head daughters: which daughter it is, whether there is just one, whether there is always at least one.

It is an intuitive idea to try make the model refer to the items that linguistic wisdom indicates to be heads: the linguistic wisdom after all is roughly that these are the items that control the construction.

For example, the basic PCFG model derived straight from the Penn treebank trees will simply have for VP a distribution of the possible daughter sequences, the vocabulary of which makes no reference to particular verbs, or kinds of verbs. But for different head words, one would expect the empirical distribution of daughter sequences to be quite different. The table below[12] gives the frequences of some different VP expansions for different verbs:

	come	take	think	want
$VP \rightarrow V$	9.5%	2.6%	4.6%	5.7%
$VP \rightarrow V\ NP$	1.1%	31.1%	0.2%	13.9%
$VP \rightarrow V\ PP$	34.5%	3.1%	7.1%	0.3%
$VP \rightarrow V\ SBAR$	6.6%	0.3%	73.0%	0.2%
$VP \rightarrow V\ S$	2.2%	1.3%	4.8%	70.8%
⋮	⋮	⋮	⋮	⋮

Each column indicates a distribution over particular expansions, and one can see that they seem to strongly differ.

Such an annotation just increases the size of the vocabulary of node labels and in principle one could assign probabilities to such a tree in the standard PCFG fashion, essentially treating each local tree as an indivisible unit. Instead of this, frequently some *factorisation* is adopted, treating daughters right and left of the head daughter independently:

$$\begin{aligned} P(\mathit{mother,\ dtrs}) &= P(\mathit{mother}) \times P_H(\mathit{head\ dtr}|\mathit{mother}) \\ &\quad \times P_L(\mathit{dtrs\ left\ of\ head}|\mathit{head\ dtr\ and\ mother}) \\ &\quad \times P_R(\mathit{dtrs\ right\ of\ head}|\mathit{head\ dtr\ and\ mother}) \end{aligned}$$

P_L and P_R are often designed to treat the daughter sequences as some kind of Markov process. All such approaches part company with a PCFG model in thereby not setting a limit to the length of daughter sequences, assigning a non-zero probability to daughter sequences of all lengths. This is somewhat in accord with the PTB trees, which are often wide and flat, especially in the treatment of optional items, such as adjectives and adverbs. As an example, treating the daughter sequences as 0-order Markov processes gives a model which treats the sub-tree rooted at S^{bought} in the above example thus (Collins, 2003, p.595):

[12] an excerpt from one in Manning and Schütze (1999a)

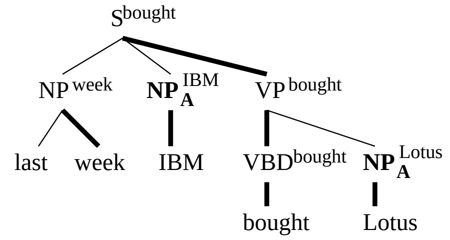
$$\begin{aligned} P(S^{bought}) \quad & \times P_H(VP^{bought}|S^{bought}) \\ & \times P_L(NP^{IBM}|Cond) \times P_L(NP^{week}|Cond) \times P_L(STOP_L|Cond) \\ & \times P_R(STOP_R|Cond) \end{aligned}$$

where *Cond* is

$$head\text{-}dtr = VP^{bought}, mother = S^{bought}$$

$STOP_L$ and $STOP_R$ are pseudo-symbols. The $STOP_R$ probability represents how often, when reading daughters right-wards from the head, there are no more right-ward daughters. This is basically Collins' Model 1 (Collins, 2003)[13]

Besides the head-notion, another kind of enhancement over the basic information present in the PTB trees is an *argument vs. adjunct* distinction. This is used in Collins' widely cited Model 2 (Collins, 2003). Again the assumption is that the annotation itself is again arrived at in the treebank trees via hand-written heuristics. Showing arguments with subscript A, an example would be:



This annotation is deployed by associating a head with a *subcat* list, divided into left and right parts, LC and RC. The first *argument* daughter *right* of the head is conditioned on RC. This is then struck out of the subcat list on which the next rightward argument daughter is conditioned, if any. By giving zero probability to any candidate argument daughter not on the relevant list, and to $STOP_L$ and $STOP_R$ whilst the relevant subcat is not empty, only a sub-tree with all and only the required arguments can get non-zero probability. Non-argument daughters are not conditioned on any subcat list. For the above example, the probability of the local tree rooted at S^{bought} then is[14]:

[13] One of the simplifications we have made is that in Collin's work instead of $P((category, headword)|\ldots)$, there is $P((category, headtag, headword)|\ldots)$, factorised as $P((category, headtag)|\ldots)$ multiplied by $P(headword|category, headtag, \ldots)$.

[14] Note this treats the VP daughter has having its own one-part subcat. Within the VP, the VBD daughter would be treated as having its own one-part subcat

$$\begin{array}{ll}
P(S^{bought}) & \times P_H(VP^{bought}|S^{bought}) \\
P_{lc}(\{NP_A\}|Cond) & \times P_L(NP_A^{IBM}|Cond,\{NP_A\}) \times P_L(NP^{week}|Cond,\{\}) \times P_L(STOP_L|Cond,\{\}) \\
P_{rc}(\{\}|Cond) & \times P_R(STOP_R|Cond,\{\})
\end{array}$$

The table below summarises the performance

	details	labelled prec/rec
Klein and Manning 2003	treebank PCFG	F1: 72.6
Collins 2003	model 1: heads, markovisation	F1: 88
Collins 2003	model 2: heads, markovisation subcat	F1: 88.5

5.5.1 Witten-Bell interpolation

A vital element in the success of model such as this is the use of *smoothing* in the estimation of the probabilities, to address the data sparsity problems, just was the case for HMM taggers. The techniques mentioned in relation to HMMs in Section 4.5.1 could all potentially be turned to here. We will discuss here the technique left pending from that discussion, which is the one used by Collins (2003): *Witten-Bell* interpolation.

Witten-Bell interpolation is a variant of the linear interpolation schemes mentioned in Section 4.5.1. As there, the general scheme is to estimate $P(A|B)$, by interpolating estimates of $\hat{p}(A|\Phi_i(B))$ where the $\Phi_i(B)$ are less and less specific versions of B

$$\Phi_0(B) = B \subset \Phi_1(B) \ldots \subset \Phi_n(B)$$

The *Witten-Bell* approach is distinguished by having for each i in this series of less specific conditions a raw estimate e_i, and a smoothed smoothed estimate $\tilde{e}_i$, with a recursive equation defining the i-th smoothed estimate $\tilde{e}_i$ as a weighted combination of the i-th raw esimate e_i and $(i+1)$-th smoothed estimate $\tilde{e}_{i+1}$:

$$\begin{array}{rcl}
\tilde{e}_i & = & \lambda_i e_i + (1-\lambda_i)\tilde{e}_{i+1} \quad \text{for } 0 \leq i < n \\
\tilde{e}_n & = & e_n
\end{array}$$

e_i is based on raw counts in the usual way

$$e_i = \mathit{count}(A, \Phi_i(B))/\mathit{count}(\Phi_i(B)$$

In the Witten-Bell method the λ_i are not constant, but instead are dependent on the particular conditioning event Φ_i. To get the weighting parameter λ_i, where f_i is the count of the conditioning event $\Phi_i(B)$, the *Witten-Bell* method considers the number u_i of distinct outcomes in these f_i conditioning events: A is one particular value of some variable (eg. head-word of a node), and u_i measures the observed size of the range of values of this variable in the context specified by $\Phi_i(B)$. λ_i is then defined

$$\lambda_i = f_i/(\alpha u_i + f_i)$$

where α is a further constant to be set: Collins used $\alpha = 5$.

This has the effect that when weights are assigned to probabilities referring to 2 different conditions, and the size of the range of outcomes in these two conditions are the same, then more weight is given to the probability based on the more frequently observed condition. It also has the effect that if the 2 different conditions are equally frequent, more weight is given to the probability with the smaller range of outcomes. This addresses a potential weakness of the simplest kind of interpolation which is that when there is a lot of evidence going into a highly specific probability, its not going to be optimal to mix this with the less specific probabilities.

For example, in Collins' Model 1, a sibling is conditioned on a head-sibling, with head-word,head-tag annotation, and a particular mother category (with head-word, head-tag annotation). There will be a raw-estimate of this based on counts for trees featuring exactly the mentioned features:

$$\hat{P}^0_L(NP^{week}|head - dtr = VP^{bought,VBD}, mother = S^{bought,VBD})$$

But the final estimate will not be based just on this, but also on trees which are unspecific for the head-word, though specific about its tag:

$$\hat{P}^1_L(NP^{week}|head - dtr = VP^{VBD}, mother = S^{VBD})$$

and also on trees unspecific for the head-tag:

$$\hat{P}^2_L(NP^{week}|head - dtr = VP, mother = S)$$

The final estimate for the desired probability is obtained by combining these according to

$$\lambda_0 \hat{P}^0 + (1 - \lambda_0)(\lambda_1 \hat{P}^1 + (1 - \lambda_1)(\hat{P}^2))$$

Factoring into the calculation of λ_0 in this case will be the frequency of the conditioning event – how often are sub-trees seen with $head - dtr =$

$VP^{bought,VBD}$ and $mother = S^{bought,VBD}$ – and how many different outcomes are possible for the label of a left sibling.

There is an interesting discussion in Collins (2003) of how the design of the model and the methods for estimating the parameters can interact. Considering VPs headed by *told*, one way to estimate the likelihood of a given expansion $dtrs$ given $(VP, told)$ would be (see Charniak, 1997)

$$P(dtrs|(VP, told)) = \lambda_0 \hat{P}^0(dtrs|(VP, told)) + (1 - \lambda_0)\hat{P}^1(dtrs|VP)$$

where the $\hat{P}^0$ and $\hat{P}^1$ are simply based on counts: thus the lexically specific estimate is interpolated with the lexically unspecific. Given $(VP, told)$ there are 27 different observed values for $dtrs$ in the tree-bank, which would lead to the value for λ_0 being set rather low, indicating the seemingly high chance of encountering further different expansions in unseen data. However, once only the *argument* daughters are retained from the possible expansions, there are only 5 observed different possibilities. Thus in estimating a subcategorisation frame given $(VP, told)$,

$$P(RC|(VP, told)) = \lambda_0 \hat{P}^0(RC|(VP, told)) + (1 - \lambda_0)\hat{P}^1(RC|VP)$$

λ_0 would be set much higher, and much less significance given to the lexically unspecific estimate.

5.6 Further reading

Concerning *supervised* parsing methods we have tried to gives some of the details of the parsing models considered by Collins (2003). This is hopefully worthwhile because it has been quite influential, and there are quite a number of other parsing models that make use of similar techniques (head annotation, factorisation). This barely scratches the surface, however, of what is a very large body of work on supervised learning of statistical parsing models. Just to mention three further areas

- dependency grammars: the topic of one of the course at this summer school
- so-called *Data Oriented Parsing*, inferring tree probabilities by combining observed tree fragments, where these fragments do not necessarily correspond to grammar rules
- re-ranking approaches, in which a probabilistic parser delivers its N-best parses, and then these parses are then re-ordered possibly referring to quite global features which are quite difficult to incorporate directly into the parser.

- semi-supervised learning, combining annotated and unannotated training data.

We also tried to give some impression of techniques that have been used in *unsupervised* learning of statistical parsing models.

For further information concerning machine learning and parsing see Jurafsky and Martin (2000) and Manning and Schütze (1999a).

Bibliography

D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1022689900470.

C. Apté, F. Damerau, and S. M. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12(3):233–251, 1994. ISSN 1046-8188. doi: http://doi.acm.org/10.1145/183422.183423.

J. Baker. Trainable grammars for speech recognition. In D.H.Klatt and J.J.Wolf, editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550. 1979.

L. Baum, T. Petrie, G. Soules, and N. Weiss. A maximizatio technique occurring in the statistical analysis of the probabilistic functions of markov chains. *Annals of Mathematical Statistics*, 41:164–171, 1970.

J. Benedi and J. Sanchez. Estimation of stochastic context-free grammars and their use as language models. *Computer Speech and Language*, July 2005.

T. Brants. Tnt: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, 2000.

E. Charniak. *Statistical Language Learning*. MIT Press, 1993.

E. Charniak. Tree-bank grammars. Technical report, Department of Computer Science, Brown University, 1996.

E. Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1997.

E. Charniak and G. Carroll. Two experiments on learning probabilistic dependency grammars from corpora. Technical Report CS-92-16, Brown University, 1992.

S. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard Computer Science, 1998.

K. W. Church and W. A. Gale. A comparison of the enhanced good-turing and deleted estimation methods for estimating probabilities of english bigrams. *Computer Speech and Language*, 5:19–54, 1991.

M. Collins. Head-driven statistical models for natural language parsing. *Computational Linguitics*, 2003.

M. Collins and N. Duffy. Convolution kernels for natural language. *Advances in Neural Information Processing Systems*, 14:625–632, 2002.

M. Collins and T. Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–69, 2005.

W. Daelemans and A. van den Bosch. *Language-independent data-oriented grapheme to phoneme conversion*. Springer, 1996.

W. Daelemans, J. Zavrel, P. Berck, and S. Gillis. MBT: A memory-based part of speech tagger generator. In *Proc. of Fourth Workshop on Very Large Corpora*, pages 14–27, 1996.

M. Davy and S. Luz. Dimensionality reduction for active learning with nearest neighbour classifier in text categorisation problems, 2007. Submitted.

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *J. Royal Statistical Society*, B 39:1–38, 1977.

P. Domingos and M. J. Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In *International Conference on Machine Learning*, pages 105–112, 1996. URL `citeseer.nj.nec.com/domingos96beyond.html`.

D. Elworthy. Does baum-welch re-estimation help taggers? In *Proceedings of the Fourth ACL Conference on Applied Natural Language Processing*, 1994.

G. Escudero, L. Màrquez, and G. Rigau. Boosting applied to word sense disambiguation. In R. L. D. Mántaras and E. Plaza, editors, *Proceedings of ECML-00, 11th European Conference on Machine Learning*, pages 129–141, Barcelona, 2000. Springer Verlag.

G. Forman. An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*, 3:1289–1305, 2003. ISSN 1533-7928.

D. Freitag and A. McCallum. Information extraction with hmms and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Informaation Extraction*, 1999.

P. Graham. A plan for spam, 2002. Downloaded from `http://www.paulgraham.com/spam.html`.

Z. Gu and N. Cercone. Segment-based hidden markov models for information extraction. In *Proceedings of COLING 2006*, pages 481–488, 2006.

M. Haruno, S. Shirai, and Y. Ooyama. Using decision trees to construct a practical parser. *Machine Learning*, 34(1):131–149, 1999.

P. J. Hayes and S. P. Weinstein. Construe-TIS: A system for content-based indexing of a database of news stories. In A. Rappaport and R. Smith, editors, *Proceedings of the IAAI-90 Conference on Innovative Applications of Artificial Intelligence*, pages 49–66. MIT Press, 1990. ISBN 0-262-68068-8.

D. Hogan. Coordinate noun phrase disambiguation in a generative parsing model. *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 680–687, 2007.

A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999. URL `citeseer.ist.psu.edu/jain99data.html`.

N. Jardine and C. J. van Rijsbergen. The use of hierarchic clustering in information retrieval. *Information Storage and Retrieval*, 7:217–240, 1971.

F. Jelinek and R. L. Mercer. Interpolated estimation of markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, 1980.

T. Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142, Chemnitz, 1998.

G. H. John and P. Langley. Estimating continuous distributions in Bayesian classifiers. In Besnard, Philippe and S. Hanks, editors, *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI'95)*, pages 338–345, San Francisco, CA, USA, Aug. 1995. Morgan Kaufmann Publishers.

D. Jurafsky and J. H. Martin. *Speech and Language Processing*. Prentice-Hall, 2000.

J. Kazama, Y. Miyao, and J. Tsujii. A maximum entropy tagger with unsupervised hidden markov models. In *Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium (NLPRS2001)*, pages 333–340, 2001.

D. Kelleher and S. Luz. Automatic hypertext keyphrase detection. In L. P. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1608–1609, Edinburgh, Scotland, 2005. IJCAI.

D. Klein. *The Unsupervised Learning of Natural Language Structure*. PhD thesis, Department of Computer Science, Stanford University, 2005.

D. Klein and C. D. Manning. Accurate unlexicalised parsing. In *Proceedings ACL 2003*, pages 423–430, 2003.

D. Klein and C. D. Manning. A generative constituent-context model for improved grammar induction. In *Proceedings fo the 40th Annual Meeting of the Association for Computational Linguistics*, pages 128–135, 2001.

D. Klein and C. D. Manning. Corpus-based induction of syntactic structure: models of dependency and constituency. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2004.

O. Kurland and L. Lee. Corpus structure, language models, and ad hoc information retrieval. In M. Sanderson, K. Järvelin, J. Allan, and P. Bruza, editors, *SIGIR 2004: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 194–201, Sheffield, UK, 2004. ACM. URL `http://doi.acm.org/10.1145/1008992.1009027`.

D. Lewis. Reuters-21578 text categorisation corpus. Avaible from `http://www.daviddlewis.com/resources/`, 1997.

D. Lewis and M. Ringuette. A comparison of two learning algorithms for text categorization. In *Third Annual Symposium on Document Analysis and Information Retrieval*, pages 81–93, 1994.

D. D. Lewis. Evaluating and optimizing autonomous text classification systems. In *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 246–254, New York, NY, USA, 1995. ACM Press. ISBN 0-89791-714-6. doi: http://doi.acm.org/10.1145/215206.215366.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

Y. H. Li and A. K. Jain. Classification of Text Documents. *The Computer Journal*, 41(8):537–546, 1998. doi: 10.1093/comjnl/41.8.537.

D. M. Magerman. Statistical decision-tree models for parsing. In *Meeting of the Association for Computational Linguistics*, pages 276–283, 1995. URL `citeseer.ist.psu.edu/magerman95statistical.html`.

C. Manning and H. Schütze. *Foundations of Statistical Language Processing*. 1999a.

C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999b.

A. McCallum and K. Nigam. A comparison of event models for naive Bayes text classification. In *AAAI/ICML-98 Workshop on Learning for Text Categorization*, pages 41–48. AAAI Press, 1998.

B. Merialdo. Tagging english text with a probabilistic model. *Computational Linguistics*, 20:155–171, 1994.

T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

F. Murtagh. Complexities of hierarchic clustering algorithms: state of the art. *Computational Statistics Quarterly*, 1(2):101–113, 1984.

M. N. Murty and A. K. Jain. Knowledge-based clustering scheme for collection management and retrieval of library books. *Pattern Recognition*, 28(7), 1995.

J. Park and C. Brew. A finite-state model of human sentence processing. In *Proceedings of COLING 2006*, pages 49–56, 2006.

J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.

T. Pedersen. A simple approach to building ensembles of Naive Bayesian classifiers for word sense disambiguation. pages 63–69, 2000.

F. Pereira and Y. Schabes. Inside-outside reestimation from partially bracketed corpora. In *27th Annual Meeting of the Association for Computational Linguistics*, pages 128–135, 1992.

S. Petrov, L. Barret, R. Thibaux, and D. Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of COLING/ACL 2006*, pages 433–440, 2006.

R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(33):21–45, 2006.

J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1: 81–106, March 1986. URL `http://dx.doi.org/10.1007/BF00116251`. 10.1007/BF00116251.

S. E. Robertson and K. S. Jones. Relevance weighting of search terms. pages 143–160, 1988.

G. Salton. *Automatic Information Organization and Retrieval*. McGraw-Hill, New York, 1968.

R. Schapire. The strength of weak learnability. *Machine Learning*, 5(2): 197–227, 1990.

F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 2002. Forthcoming.

L. Shen and A. Joshi. Ranking and reranking with perceptron. *Machine Learning*, 60(1-3):73–96, September 2005. ISSN 0885-6125.

K. Sparck Jones and D. Jackson. The use of automatically-obtained keyword classifications for information retrieval. *Information Storage and Retrieval*, 5:175–201, 1970.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

C. J. van Rijsbergen. *Information Retrieval*. Butterworths, 1979. URL `http://www.dcs.gla.ac.uk/Keith/`.

Q. I. Wang and D. Schuurmans. Improved estimation for unsupervised part-of-speech tagging. In *Proceedings of the 2005 IEEE International Conference on Natural Language Processing and Knowledge Engineering (IEEE NLP-KE'2005)*, 2005.

R. Weischedel, M. Meteer, R. Schwartz, L. Ramshaw, and J. Palmucci. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 1993.

I. H. Witten, G. W. Paynter, E. Frank, C. Gutwin, and C. G. Nevill-Manning. KEA: Practical automatic keyphrase extraction. In *DL'99: Proceedings of the 4th ACM International Conference on Digital Libraries*, pages 254–255, 1999.

Y. Yang. A study on thresholding strategies for text categorization. In W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, editors, *Proceedings* *of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-01)*, pages 137–145, New York, Sept. 9–13 2001. ACM Press.

Y. Yang and C. G. Chute. An example-based mapping method for text categorization and retrieval. *ACM Transaction on Information Systems*, 12(3):252–277, 1994.

Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 42–49, 1999.

Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In D. H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 412–420, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.

D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. *Proceedings of the 33rd conference on Association for Computational Linguistics*, pages 189–196, 1995.

J. Zavrel and W. Daelemans. Feature-rich memory-based classification for shallow NLP and information extraction. In *Text Mining*, pages 33–54. 2003.

G. Zhou and J. Su. Named entity recognition using an hmm-based chunk tagger. In *Proceedings of ACL'02*, pages 473–480, 2002.

Index