VII

Probabilistic Models for Information Extraction

Several common themes frequently recur in many tasks related to processing and analyzing complex phenomena, including natural language texts. Among these themes are classification schemes, clustering, probabilistic models, and rule-based systems.

This section describes some of these techniques generally, and the next section applies them to the tasks described in Chapter VI.

Research has demonstrated that it is extremely fruitful to model the behavior of complex systems as some form of a random process. Probabilistic models often show better accuracy and robustness against the noise than categorical models. The ultimate reason for this is not quite clear and is an excellent subject for a philosophical debate.

Nevertheless, several probabilistic models have turned out to be especially useful for the different tasks in extracting meaning from natural language texts. Most prominent among these probabilistic approaches are hidden Markov models (HMMs), stochastic context-free grammars (SCFG), and maximal entropy (ME).

VII.1 HIDDEN MARKOV MODELS

An HMM is a finite-state automaton with stochastic state transitions and symbol emissions (Rabiner 1990). The automaton models a probabilistic generative process. In this process, a sequence of symbols is produced by starting in an initial state, emitting a symbol selected by the state, making a transition to a new state, emitting a symbol selected by the state, and repeating this transition–emission cycle until a designated final state is reached.

Formally, let $O = \{o_1, \ldots o_M\}$ be the finite set of observation symbols and $Q = \{q_1, \ldots q_N\}$ be the finite set of states. A first-order Markov model λ is a triple (π, A, B), where $\pi : Q \to [0, 1]$ defines the starting probabilities, $A : Q \times Q \to [0, 1]$ defines the transition probabilities, and $B : Q \times O \to [0, 1]$ denotes the emission probabilities. Because the functions π, A, and B define true probabilities, they must satisfy

$$\Sigma_{q \in Q}\, \pi(q) = 1,$$
$$\Sigma_{q' \in Q} A(q, q') = 1 \quad \text{and} \quad \Sigma_{o \in O} B(q, o) = 1 \text{ for all states } q.$$

A model λ together with the random process described above induces a probability distribution over the set O^* of all possible observation sequences.

VII.1.1 The Three Classic Problems Related to HMMs

Most applications of hidden Markov models can be reduced to three basic problems:

1. Find $P(T \mid \lambda)$ – the probability of a given observation sequence T in a given model λ.
2. Find $\text{argmax}_{S \in Q}{}^{|T|} P(T, S \mid \lambda)$ – the most likely state trajectory given λ and T.
3. Find $\text{argmax}_{\lambda} P(T, \mid \lambda)$ – the model that best accounts for a given sequence.

The first problem allows us to compute the probability distribution induced by the model. The second finds the most probable states sequence for a given observation sequence. These two tasks are typically used for analyzing a given observation.

The third problem, on the other hand, adjusts the model itself to maximize the likelihood of the given observation. It can be viewed as an HMM training problem.

We now describe how each of these three problems can be solved. We will start by calculating $P(T \mid \lambda)$, where T is a sequence of observation symbols $T = t_1 t_2 \ldots t_k \in O^*$. The most obvious way to do that would be to enumerate every possible state sequence of length $|T|$. Let $S = s_1 s_2 \ldots s_{|T|} \in Q^{|T|}$ be one such sequence. Then we can calculate the probability $P(T \mid S, \lambda)$ of generating T knowing that the process went through the states sequence S. By Markovian assumption, the emission probabilities are all independent of each other. Therefore,

$$P(T \mid S, \lambda) = \pi_{i=1..|T|} B(s_i, t_i).$$

Similarly, the transition probabilities are independent. Thus the probability $P(S|\lambda)$ for the process to go through the state sequence S is

$$P(S \mid \lambda) = \pi(s_1) \cdot \pi_{i=1..|T|-1} A(s_i, s_{i+1}).$$

Using the above probabilities, we find that the probability $P(T|\lambda)$ of generating the sequence can be calculated as

$$P(T \mid \lambda) = \Sigma_{S \in Q}^{|T|} P(T \mid S, \lambda) \cdot P(S \mid \lambda).$$

This solution is of course infeasible in practice because of the exponential number of possible state sequences. To solve the problem efficiently, we use a dynamical programming technique. The resulting algorithm is called the *forward–backward* procedure.

VII.1.2 The Forward–Backward Procedure

Let α_m(q), the *forward variable*, denote the probability of generating the initial segment $t_1 t_2 \ldots t_m$ of the sequence T and finishing at the state q at time m. This forward variable can be computed recursively as follows:

1. $\alpha_1(q) = \pi(q) \cdot B(q, t_1)$,
2. $\alpha_{n+1}(q) = \Sigma_{q' \in Q} \alpha_n(q') \cdot A(q', q) \cdot B(q, t_{n+1})$.

Then, the probability of the whole sequence T can be calculated as

$$P(T \mid \lambda) = \Sigma_{q \in Q} \alpha_{|T|}(q).$$

In a similar manner, one can define $\beta_m(q)$, the *backward variable*, which denotes the probability of starting at the state q and generates the final segment $t_{m+1} \dots t_{|T|}$ of the sequence T. The backward variable can be calculated starting from the end and going backward to the beginning of the sequence:

1. $\beta_{|T|}(q) = 1$,
2. $\beta_{n-1}(q) = \Sigma_{q' \in Q} A(q, q') \cdot B(q', t_n) \cdot \beta_n(q')$.

The probability of the whole sequence is then

$$P(T \mid \lambda) = \Sigma_{q \in Q}\, \pi(q) \cdot B(q, t_1) \cdot \beta_1(q).$$

VII.1.3 The Viterbi Algorithm

We now proceed to the solution of the second problem – finding the most likely state sequence for a given sequence T. As with the previous problem, enumerating all possible state sequences S and choosing the one maximizing $P(T, S \mid \lambda)$ is infeasible. Instead, we again use dynamical programming, utilizing the following property of the optimal states sequence: if T' is some initial segment of the sequence $T = t_1 t_2 \dots t_{|T|}$ and $S = s_1 s_2 \dots s_{|T|}$ is a state sequence maximizing $P(T, S \mid \lambda)$, then $S' = s_1 s_2 \dots s_{|T'|}$ maximizes $P(T', S' \mid \lambda)$ among all state sequences of length $|T'|$ ending with $s_{|T|}$. The resulting algorithm is called the *Viterbi* algorithm.

Let $\gamma_n(q)$ denote the state sequence ending with the state q, which is optimal for the initial segment $T_n = t_1 t_2 \dots t_n$ among all sequences ending with q, and let $\delta_n(q)$ denote the probability $P(T_n, \gamma_n(q) \mid \lambda)$ of generating this initial segment following those optimal states. Delta and gamma can be recursively calculated as follows:

1. $1.\delta_1(q) = \pi(q) \cdot B(q, t_1),\ \gamma_1(q) = q$,
2. $\delta_{n+1}(q) = \max_{q' \in Q} \delta_n(q') \cdot A(q', q) \cdot B(q, t_{n+1}),\ \gamma_{n+1}(q) = \gamma_1(q')q$,
   where $q' = \operatorname{argmax}_{q' \in Q} \delta_n(q') \cdot A(q', q) \cdot B(q, t_{n+1})$.

Then, the best states sequence among $\{\gamma_{|T|}(q) : q \in Q\}$ is the optimal one:

$$\operatorname{argmax}_{S \in Q}{}^{|T|} P(T, S \mid \lambda) = \gamma_{|T|}(\operatorname{argmax}_{q \in Q} \delta_{|T|}(q)).$$

Example of the Viterbi Computation

Using the HMM described in Figure VII.1 with the sequence (a, b, a), one would take the following steps in using the Viterbi algorithm:

$$\pi_i = \begin{pmatrix} 0.5 & 0 & 0.5 \end{pmatrix}, \qquad A_{ij} = \begin{pmatrix} 0.1 & 0.4 & 0.4 \\ 0.4 & 0.1 & 0.5 \\ 0.4 & 0.5 & 0.1 \end{pmatrix},$$

$$B_i(a) = \begin{pmatrix} 0.5 & 0.8 & 0.2 \end{pmatrix}, \qquad B_i(b) = \begin{pmatrix} 0.5 & 0.2 & 0.8 \end{pmatrix}$$

First Step (a):

- $\delta_1(S1) = \pi(S1) \cdot B(S1, a) = 0.5 \cdot 0.5 = 0.25$

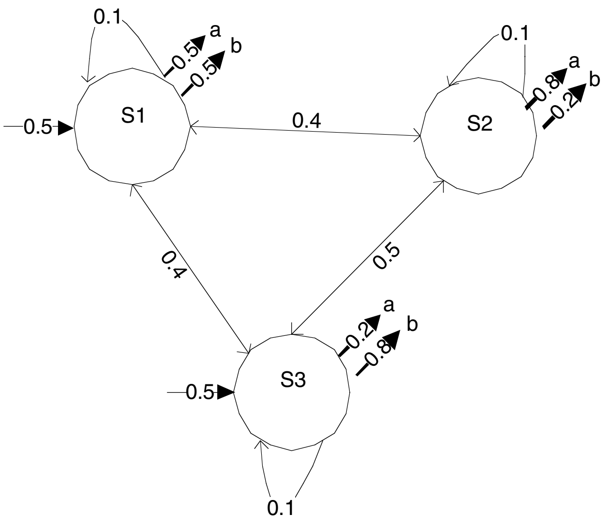

Figure VII.1. A sample HMM.

- $\delta_1(S2) = \pi(S2) \cdot B(S2, a) = 0$
- $\delta_1(S3) = \pi(S3) \cdot B(S3, a) = 0.5 \cdot 0.2 = 0.1$

Second Step (b):

- $$\begin{aligned}\delta_2(S1) &= \max_{q' \in Q} \delta_1(q') \cdot A(q', S1) \cdot B(S1, b) \\ &= \max(\delta_1(S1) \cdot A(S1, S1) \cdot B(S1, b), \\ &\qquad \delta_1(S2) \cdot A(S2, S1) \cdot B(S1, b), \\ &\qquad \delta_1(S3) \cdot A(S3, S1) \cdot B(S1, b)) \\ &= \max(0.25 \cdot 0.1 \cdot 0.5, \\ &\qquad 0, \\ &\qquad 0.1 \cdot 0.4 \cdot 0.5) \\ &= \max(0.0125, 0, 0.02) = 0.02\end{aligned}$$
- $\gamma_2(S1) = S3$

In a similar way, we continue to calculate the other δ and γ factors. Upon reaching t_3 we can see that $S1$ and $S3$ have the highest probabilities; hence, we trace back our steps from both states using the γ variables. We have in this case two optimal paths: $\{S1, S3, S1\}$ and $\{S3, S2, S3\}$. The diagram of the computation of the Viterbi Algorithm is shown in Figure VII.2.

Note that, unlike the forward–backward algorithm described in Section VII.1.2 the Viterbi algorithm does not use summation of probabilities. Only multiplications are involved. This is convenient because it allows the use of logarithms of probabilities instead of the probabilities themselves and to use summation instead of multiplication. This can be important because, for large sequences, the probabilities soon become infinitesimal and leave the range of the usual floating-point numbers.

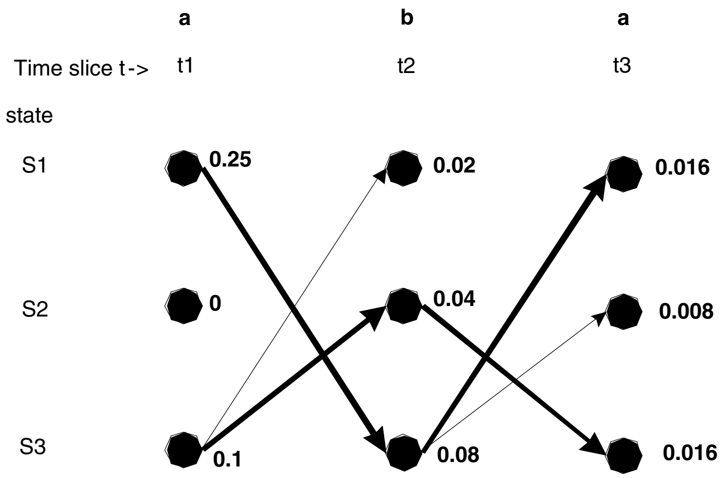

Figure VII.2. Computation of the optimal path using the Viterbi algorithm.

VII.1.4 The Training of the HMM

The most difficult problem of the three involves training the HMM. In this section, only the problem of estimating the parameters of HMM is covered, leaving the topology of the finite-state automaton fixed.

The training algorithm is given some initial HMM and adjusts it so as to maximize the probability of the training sequence. However, the set of states is given in advance, and the transition and emission probabilities, which are initially zero, remain zero. The adjustment formulas are called *Baum–Welsh reestimation formulas*.

Let $\mu_n(q)$ be the probability $P(s_n = q \mid T, \lambda)$ of being in the state q at time n while generating the observation sequence T. Then $\mu_n(q) \cdot P(T \mid \lambda)$ is the probability of generating T passing through the state q at time n. By definition of the forward and backward variables presented in Section VII.1.2, this probability is equal to $\alpha_n(q) \cdot \beta_n(q)$. Thus,

$$\mu_n(q) = \alpha_n(q) \cdot \beta_n(q) \,/\, P(T \mid \lambda).$$

Also let $\varphi_n(q, q')$ be the probability $P(s_n = q, s_{n+1} = q' \mid T, \lambda)$ of passing from state q to state q' at time n while generating the observation sequence T. As in the preceding equation,

$$\varphi_n(q, q') = \alpha_n(q) \cdot A(q, q') \cdot B(q', o_{n+1}) \cdot \beta_n(q)/P(T \mid \lambda).$$

The sum of $\mu_n(q)$ over all $n = 1 \ldots \mid T \mid$ can be seen as the expected number of times the state q was visited while generating the sequence T. Or, if one sums over $n = 1 \ldots \mid T \mid -1$, the expected number of transitions out of the state q results because there is no transition at time $|T|$. Similarly, the sum of $\varphi_n(q, q')$ over all $n = 1 \ldots \mid T \mid -1$ can be interpreted as the expected number of transitions from the state q to q'.

The Baum–Welsh formulas reestimate the parameters of the model λ according to the expectations

$$\pi'(q) := \mu_1(q),$$
$$A'(q, q') := \Sigma_{n=1..|T|-1}\varphi_n(q, q')/\Sigma_{n=1..|T|-1}\mu_n(q),$$
$$B'(q, o) := \Sigma_{n:Tn=o}\mu_n(q)/\Sigma_{n=1..|T|}\mu_n(q).$$

It can be shown that the model $\lambda' = (\pi', A', B')$ is equal either to λ, in which case the λ is the critical point of the likelihood function $P(T \mid \lambda)$, or λ', which better accounts for the training sequence T than the original model λ in the sense that $P(T \mid \lambda') > P(T \mid \lambda)$. Therefore, the training problem can be solved by iteratively applying the reestimation formulas until convergence.

VII.1.5 Dealing with Training Data Sparseness

It is often the case that the amount of training data – the length of the training sequence T – is insufficient for robust estimation of parameters of a complex HMM. In such cases, there is often a trade-off between constructing complex models with many states and constructing simple models with only a few states.

The complex model is better able to represent the intricate structure of the task but often results in a poor estimation of parameters. The simpler model, on the other hand, yields a robust parameter estimation but performs poorly because it is not sufficiently expressive to model the data.

Smoothing and *shrinkage* (Freitag and McCallum 1999) are the techniques typically used to take the sting out of data sparseness problems in probabilistic modeling. This section describes the techniques with regard to HMM, although they apply in other contexts as well such as SCFG.

Smoothing is the process of flattening a probability distribution implied by a model so that all reasonable sequences can occur with some probability. This often involves broadening the distribution by redistributing weight from high-probability regions to zero-probability regions. Note that smoothing may change the topology of an HMM by making some initially zero probability nonzero.

The simplest possible smoothing method is just to pretend that every possible training event occurrs one time more than it actually does. Any constant can be used instead of "one." This method is called *Laplace smoothing*. Other possible methods may include back-off smoothing, deleted interpolation, and others.[1]

Shrinkage is defined in terms of some hierarchy representing the expected similarity between parameter estimates. With respect to HMMs, the hierarchy can be defined as a tree with the HMM states for the leaves – all at the same depth.

This hierarchy is created as follows. First, the most complex HMM is built and its states are used for the leaves of the tree. Then the states are separated into disjoint classes within which the states are expected to have similar probability distributions. The classes become the parents of their constituent states in the hierarchy. Note that the HMM structure at the leaves induces a simpler HMM structure at the level of

[1] Full details outlining the smoothing technique can be found in Manning and Schutze (1999).

the classes. It is generated by summing the probabilities of emissions and transitions of all states in a class. This process may be repeated until only a single-state HMM remains at the root of the hierarchy.

Training such a hierarchy is straightforward. The emission and transition probabilities of the states in the internal levels of the hierarchy are calculated by summing the corresponding probabilities of their descendant leaves. Modeling using the hierarchy is also simple. The topology of the most complex HMM is used. However, the transition and emission probabilities of a given state are calculated by linearly interpolating between the corresponding probabilities for all ancestors of the state in the shrinkage hierarchy. The weights of the different models in the interpolation can be fixed at some reasonable value, like ½, or can be optimized using held-out training data.

VII.2 STOCHASTIC CONTEXT-FREE GRAMMARS

An SCFG is a quintuple $G = (T, N, S, R, P)$, where T is the alphabet of terminal symbols (tokens), N is the set of nonterminals, S is the starting nonterminal, R is the set of rules, and $P : R \rightarrow [0.1]$ defines their probabilities. The rules have the form

$$n \rightarrow s_1 s_2 \ldots s_k,$$

where n is a nonterminal and each s_i is either a token or another nonterminal. As can be seen, SCFG is a usual context-free grammar with the addition of the P function.

As is true for a canonical (nonstochastic) grammar, SCFG is said to *generate* (or *accept*) a given string (sequence of tokens) if the string can be produced starting from a sequence containing just the starting symbol S and expanding nonterminals one by one in the sequence using the rules from the grammar. The particular way the string was generated can be naturally represented by a *parse tree* with the starting symbol as a root, nonterminals as internal nodes, and the tokens as leaves.

The semantics of the probability function P are straightforward. If r is the rule $n \rightarrow s_1 s_2 \ldots s_k$, then $P(r)$ is the frequency of expanding n using this rule, or, in Bayesian terms, if it is known that a given sequence of tokens was generated by expanding n, then $P(r)$ is the a priori likelihood that n was expanded using the rule r. Thus, it follows that for every nonterminal n the sum $\sum P(r)$ of probabilities of all rules r headed by n must be equal to one.

VII.2.1 Using SCFGs

Usually, some of the nonterminal symbols of a grammar correspond to meaningful language concepts, and the rules define the allowed syntactic relations between these concepts. For instance, in a parsing problem, the nonterminals may include S, NP, VP, and others, and the rules would define the syntax of the language. For example, $S \rightarrow NP\ VP$. Then, when the grammar is built, it is used for parsing new sentences.

In general, grammars are ambiguous in the sense that a given string can be generated in many different ways. With nonstochastic grammars there is no way to compare different parse trees, and thus the only information we can gather for a given sentence is whether or not it is *grammatical* – that is whether it can be produced by any parse. With SCFG, different parses have different probabilities; therefore, it is possible to find the best one, resolving the ambiguity.

In designing preprocessing systems around SCFGs, it has been found neither necessary nor desirable (for performance reasons) to perform a full syntactic parsing of all sentences in the document. Instead, a very basic "parsing" can be employed for the bulk of a text, but within the relevant parts the grammar is much more detailed. Thus, the extraction grammars can be said to define *sublanguages* for very specific domains.

In the classical definition of SCFG it is assumed that the rules are all independent. In this case it is possible to find the (unconditional) probability of a given parse tree by simply multiplying the probabilities of all rules participating in it. Then the usual parsing problem is formulated as follows: Given a sequence of tokens (a *string*), find the most probable parse tree that could generate the string. A simple generalization of the Viterbi algorithm is able to solve this problem efficiently.

In practical applications of SCFGs, it is rarely the case that the rules are truly independent. Then, the easiest way to cope with this problem while leaving most of the formalism intact is to let the probabilities $P(r)$ be conditioned on the context where the rule is applied. If the conditioning context is chosen reasonably, the Viterbi algorithm still works correctly even for this more general problem.

VII.3 MAXIMAL ENTROPY MODELING

Consider a random process of an unknown nature that produces a single output value y, a member of a finite set Y of possible output values. The process of generating y may be influenced by some contextual information x – a member of the set X of possible contexts. The task is to construct a statistical model that accurately represents the behavior of the random process. Such a model is a method of estimating the conditional probability of generating y given the context x.

Let $P(x, y)$ be denoted as the unknown true joint probability distribution of the random process, and let $p(y \mid x)$ be the model we are trying to build taken from the class $\wp$ of all possible models. To build the model we are given a set of training samples generated by observing the random process for some time. The training data consist of a sequence of pairs (x_i, y_i) of different outputs produced in different contexts.

In many interesting cases the set X is too large and underspecified to be used directly. For instance, X may be the set of all dots "." in all possible English texts. For contrast, the Y may be extremely simple while remaining interesting. In the preceding case, the Y may contain just two outcomes: "SentenceEnd" and "NotSentenceEnd." The target model $p(y \mid x)$ would in this case solve the problem of finding sentence boundaries.

In such cases it is impossible to use the context x directly to generate the output y. There are usually many regularities and correlations, however, that can be exploited. Different contexts are usually similar to each other in all manner of ways, and similar contexts tend to produce similar output distributions.

To express such regularities and their statistics, one can use *constraint functions* and their expected values. A constraint function $f: X \times Y \to R$ can be any real-valued function. In practice it is common to use binary-valued *trigger* functions of the form

$$f(x, y) = \begin{cases} 1, & \text{if } C(x) \text{ and } y = y_i, \\ 0, & \text{otherwise.} \end{cases}$$

Such a trigger function returns one for pair (x, y) if the context x satisfies the condition predicate C and the output value y is y_i. A common short notation for such a trigger function is $C \to y_i$. For the example above, useful triggers are

previous token is “Mr” $\to$ NotSentenceEnd,
next token is capitalized $\to$ SentenceEnd.

Given a constraint function f, we express its importance by requiring our target model to reproduce f’s expected value faithfully in the true distribution:

$$p(f) = \Sigma_{x,y} p(x, y) f(x, y) = P(f) = \Sigma_{x,y} P(x, y) f(x, y).$$

In practice we cannot calculate the true expectation and must use an *empirical* expected value calculated by summing over the training samples:

$$p_E(f) = \Sigma_{i=1..N} \Sigma_{y \in Y} p(y \mid x_i) f(x_i, y)/N = P_E(f) = \Sigma_{i=1..N} f(x_i, y_i)/N.$$

The choice of feature functions is of course domain dependent. For now, let us assume the complete set of features $F = \{f_k\}$ is given. One can express the completeness of the set of features by requiring that the model agree with all the expected value constraints

$$p_E(f_k) = P_E(f_k) \text{ for all } f_k \in F$$

while otherwise being as uniform as possible. There are of course many models satisfying the expected values constraints. However, the uniformity requirement defines the target model uniquely. The degree of uniformity of a model is expressed by its *conditional entropy*

$$H(p) = -\sum_{x,y} p(x) \cdot p(y \mid x) \cdot \log p(y \mid x).$$

Or, empirically,

$$H_E(p) = -\Sigma_{i=1..N} \Sigma_{y \in Y} p(y \mid x_i) \cdot \log p(y \mid x_i)/N.$$

The constrained optimization problem of finding the maximal-entropy target model is solved by application of Lagrange multipliers and the Kuhn–Tucker theorem. Let us introduce a parameter λ_k (the Lagrange multiplier) for every feature. Define the Lagrangian $\Lambda(p, \lambda)$ by

$$\Lambda(p, \lambda) \equiv H_E(p) + \Sigma_k \lambda_k (p_E(f_k) - P_E(f_k)).$$

Holding λ fixed, we compute the unconstrained maximum of the Lagrangian over all $p \in \wp$. Denote by p_λ the p where $\Lambda(p, \lambda)$ achieves its maximum and by $\Psi(\lambda)$ the value of Λ at this point. The functions p_λ and $\Psi(\lambda)$ can be calculated using simple calculus:

$$p_\lambda(y \mid x) = \frac{1}{Z_\lambda(x)} \exp\left(\sum_k \lambda_k f_k(x, y)\right),$$
$$\Psi(\lambda) = -\sum_{i=1..N} \log Z_\lambda(x)/N + \sum_k \lambda_k P_E(f_k),$$

where $Z_\lambda(x)$ is a normalizing constant determined by the requirement that $\Sigma_{y \in Y} p_\lambda(y \mid x) = 1$. Finally, we pose the *dual* optimization problem

$$\lambda^* = \operatorname{argmax}_\lambda \Psi(\lambda).$$

The Kuhn–Tucker theorem asserts that, under certain conditions, which include our case, the solutions of the primal and dual optimization problems coincide. That is, the model p, which maximizes $H_E(p)$ while satisfying the constraints, has the parametric form p_{λ^*}.

It is interesting to note that the function $\Psi(\lambda)$ is simply the log-likelihood of the training sample as predicted by the model p_λ. Thus, the model p_{λ^*} maximizes the likelihood of the training sample among all models of the parametric form p_λ.

VII.3.1 Computing the Parameters of the Model

The function $\Psi(\lambda)$ is well behaved from the perspective of numerical optimization, for it is smooth and concave. Consequently, various methods can be used for calculating λ^*. Generalized iterative scaling is the algorithm specifically tailored for the problem. This algorithm is applicable whenever all constraint functions are non-negative: $f_k(x, y) \geq 0$.

The algorithm starts with an arbitrary choice of λ's – for instance $\lambda_k = 0$ for all k. At each iteration the λ's are adjusted as follows:

1. For all k, let $\Delta\lambda_k$ be the solution to the equation

   $$P_E(f_k) = \Sigma_{i=1..N} \Sigma_{y \in Y} p_\lambda(y \mid x_i) \cdot f_k(x_i, y) \cdot \exp(\Delta\lambda_k f^\#(x_i, y))/N,$$

   where $f^\#(x, y) = \Sigma_k f_k(x, y)$.
2. For all k, let $\lambda_k := \lambda_k + \Delta\lambda_k$.

In the simplest case, when $f^\#$ is constant, $\Delta\lambda_k$ is simply $(1/f^\#) \cdot \log P_E(f_k)/p_{\lambda E}(f_k)$. Otherwise, any numerical algorithm for solving the equation can be used such as Newton's method.

VII.4 MAXIMAL ENTROPY MARKOV MODELS

For many tasks the conditional models have advantages over generative models like HMM. Maximal entropy Markov models (McCallum, Freitag, and Pereira 2000), or MEMM, is one class of such a conditional model closest to the HMM.

A MEMM is a probabilistic finite-state acceptor. Unlike HMM, which has separate transition and emission probabilities, MEMM has only transition probabilities, which, however, depend on the observations. A slightly modified version of the

Viterbi algorithm solves the problem of finding the most likely state sequence for a given observation sequence.

Formally, a MEMM consists of a set $Q = \{q_1, \ldots, q_N\}$ of states, and a set of transition probabilities functions $A_q : X \times Q \to [0, 1]$, where X denotes the set of all possible observations. $A_q(x, q')$ gives the probability $P(q' \mid q, x)$ of transition from q to q', given the observation x. Note that the model does not generate x but only conditions on it. Thus, the set X need not be small and need not even be fully defined. The transition probabilities A_q are separate exponential models trained using maximal entropy.

The task of a trained MEMM is to produce the most probable sequence of states given the observation. This task is solved by a simple modification of the Viterbi algorithm. The forward–backward algorithm, however, loses its meaning because here it computes the probability of the observation being generated by *any* state sequence, which is always one. However, the forward and backward variables are still useful for the MEMM training. The forward variable [Ref->HMM] $\alpha_m(q)$ denotes the probability of being in state q at time m given the observation. It is computed recursively as

$$\alpha_{n+1}(q) = \Sigma_{q' \in Q} \alpha_n(q') \cdot A_q(x, q').$$

The backward variable β denotes the probability of starting from state q at time m given the observation. It is computed similarly as

$$\beta_{n-1}(q) = \Sigma_{q' \in Q} A_q(x, q') \cdot \beta_n(q').$$

The model A_q for transition probabilities from a state is defined parametrically using constraint functions. If $f_k : X \times Q \to \mathbf{R}$ is the set of such functions for a given state q, then the model A_{q} can be represented in the form

$$A_q(x, q') = Z(x, q)^{-1} \exp(\Sigma_k \lambda_k f_k(x, q')),$$

where λ_k are the parameters to be trained and $Z(x, q)$ is the normalizing factor making probabilities of all transitions from a state sum to one.

VII.4.1 Training the MEMM

If the true states sequence for the training data is known, the parameters of the models can be straightforwardly estimated using the GIS algorithm for training ME models.

If the sequence is not known – for instance, if there are several states with the same label in a fully connected MEMM – the parameters must be estimated using a combination of the Baum–Welsh procedure and iterative scaling. Every iteration consists of two steps:

1. Using the forward–backward algorithm and the current transition functions to compute the state occupancies for all training sequences.
2. Computing the new transition functions using GIS with the feature frequencies based on the state occupancies computed in step 1.

It is unnecessary to run GIS to convergence in step 2; a single GIS iteration is sufficient.

VII.5 CONDITIONAL RANDOM FIELDS

Conditional random fields (CRFs) (Lafferty, McCallum, et al. 2001) constitute another conditional model based on maximal entropy. Like MEMMs, which are described in the previous section, CRFs are able to accommodate many possibly correlated features of the observation. However, CRFs are better able to trade off decisions at different sequence positions. MEMMs were found to suffer from the so-called label bias problem.

The problem appears when the MEMM contains states with different output degrees. Because the probabilities of transitions from any given state must sum to one, transitions from lower degree states receive higher probabilities than transitions from higher degree states. In the extreme case, transition from a state with degree one always gets probability one, effectively ignoring the observation.

CRFs do not have this problem because they define a single ME-based distribution over the whole label sequence. On the other hand, the CRFs cannot contain "hidden" states – the training data must define the sequence of states precisely. For most practical sequence labeling problems this limitation is not significant.

In the description of CRFs presented here, attention is restricted to their simplest form – linear chain CRFs, which generalize finite-state models like HMMs and MEMMs. Such CRFs model the conditional probability distribution of sequences of labels given the observation sequences. More general formulations are possible (Lafferty et al. 2001; McCallum and Jensen 2003).

Let X be a random variable over the observation sequences and Y a random variable over the label sequences. All components Y_i of Y are assumed to range over a finite set L of labels. The labels roughly correspond to states in finite-state models. The variables X and Y are jointly distributed, but CRF constructs a conditional model $p(Y \mid X)$ without explicitly modeling the margin $p(X)$.

A CRF on (X, Y) is specified by a vector $\mathbf{f} = (f_1, f_2, \ldots f_m)$ of *local features* and a corresponding *weight vector* $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots \lambda_m)$. Each local feature $f_j(\mathbf{x}, \mathbf{y}, i)$ is a real-valued function of the observation sequence $\mathbf{x}$, the labels sequence $\mathbf{y} = (y_1, y_2, \ldots y_n)$, and the sequence position i. The value of a feature function at any given position i may depend only on y_i or on y_i and y_{i+1} but not on any other components of the label sequence $\mathbf{y}$. A feature that depends only on y_i at any given position i is called a *state feature*, and if it depends on y_i and y_{i+1} it is called a *transition feature*.

The *global feature vector* $\mathbf{F}(\mathbf{x}, \mathbf{y})$ is a sum of local features at all positions:

$$\mathbf{F}(\mathbf{x}, \mathbf{y}) = \Sigma_{i=1..n}\mathbf{f}(\mathbf{x}, \mathbf{y}, i).$$

The conditional probability distribution defined by the CRF is then

$$p_\lambda(\mathbf{y} \mid \mathbf{x}) = Z_\lambda(\mathbf{x})^{-1}\exp(\boldsymbol{\lambda} \cdot \mathbf{F}(\mathbf{x}, \mathbf{y})),$$

where

$$Z_\lambda(\mathbf{x}) = \Sigma_y \exp(\boldsymbol{\lambda} \cdot \mathbf{F}(\mathbf{x}, \mathbf{y})).$$

It is a consequence of a fundamental theorem about random Markov fields (Kindermann and Snell 1980; Jain and Chellappa 1993) that any conditional distribution $p(\mathbf{y}/\mathbf{x})$ obeying the Markov property $p(y_i \mid x, \{y_j\}_{j \neq i}) = p(y_i \mid x, y_{i-1}, y_{i+1})$

can be written in the exponential form above with a suitable choice of the feature functions and the weights vector.

Notice also that any HMM can be represented in the form of CRF if its set of states Q coincide with the set of labels L. If $A : L \times L \to [0, 1]$ denotes the transition probability and $B : L \times O \to [0, 1]$ denotes is the emission probability functions, the corresponding CRF can be defined by the set of state features

$$f_{yo}(\mathbf{x}, \mathbf{y}, k) \equiv (y_k = y) \text{ and } (x_k = o)$$

and transition features

$$f_{yy'}(\mathbf{x}, \mathbf{y}, k) \equiv (y_k = y) \text{ and } (y_{k+1} = y')$$

with the weights $\lambda_{yo} = \log B(y, o)$ and $\lambda_{yy'} = \log A(y, y')$.

VII.5.1 The Three Classic Problems Relating to CRF

As with HMMs, three main problems are associated with CRFs:

1. Given a CRF λ, an observation sequence $\mathbf{x}$, and a label sequence $\mathbf{y}$, find the conditional probability $p_\lambda(\mathbf{y} \mid \mathbf{x})$.
2. Given a CRF λ and an observation sequence $\mathbf{x}$, find the most probable label sequence $\mathbf{y} = \text{argmax}_{\mathbf{y}}\, p_\lambda(\mathbf{y} \mid \mathbf{x})$.
3. Given a set of training samples $(\mathbf{x}^{(k)}, \mathbf{y}^{(k)})$, find the CRF parameters λ that maximize the likelihood of the training data.

At least a basic attempt will be made here to explain the typical approaches for each of these problems.

VII.5.2 Computing the Conditional Probability

For a given $\mathbf{x}$ and a given position i define a $|L| \times |L|$ *transition matrix* $M_i(\mathbf{x})$ by

$$M_i(\mathbf{x})[y, y'] = \exp\,(\boldsymbol{\lambda} \cdot \mathbf{f}(\mathbf{x}, \{y_i = y, y_{i+1} = y'\}, i)).$$

Then, the conditional probability $p_\lambda(\mathbf{y} \mid \mathbf{x})$ can be decomposed as

$$p_\lambda(\mathbf{y} \mid \mathbf{x}) = Z_\lambda(\mathbf{x})^{-1} \pi_{i=1..n} M_i(\mathbf{x})[y_i, y_{i+1}].$$

The normalization factor $Z_\lambda(\mathbf{x})$ can be computed by a variant of the forward–backward algorithm. The forward variables $\alpha_i(\mathbf{x}, y)$ and the backward variables $\beta_i(\mathbf{x}, \mathbf{y})$, for $y \in L$, can be computed using the recurrences

$$\begin{aligned}
&\alpha_0(x, y) = 1, \\
&\alpha_{i+1}(\mathbf{x}, y) = \Sigma_{y' \in L} \alpha_i(\mathbf{x}, y') M_i(y', y, \mathbf{x}), \\
&\beta_n(\mathbf{x}, y) = 1, \\
&\beta_{i-1}(\mathbf{x}, y) = \Sigma_{y' \in L} M_{i-1}(y, y', \mathbf{x}) \beta_i(\mathbf{x}, y').
\end{aligned}$$

Finally, $Z_\lambda(\mathbf{x}) = \Sigma_{y \in L} \alpha_n(\mathbf{x}, y)$.

VII.5.3 Finding the Most Probable Label Sequence

The most probable label sequence $\mathbf{y} = \text{argmax}_{\mathbf{y}}\ p_\lambda(\mathbf{y} \mid \mathbf{x})$ can be found by a suitable adaptation of the Viterbi algorithm. Note that

$$\text{argmax}_y p_\lambda(\mathbf{y} \mid \mathbf{x}) = \text{argmax}_y\ (\boldsymbol{\lambda} \cdot \mathbf{F}(\mathbf{x}, \mathbf{y}))$$

because the normalizer $Z_\lambda(\mathbf{x})$ does not depend on $\mathbf{y}$. $\mathbf{F}(\mathbf{x}, \mathbf{y})$ decomposes into a sum of terms for consecutive pairs of labels, making the task straightforward.

VII.5.4 Training the CRF

CRF is trained by maximizing the log-likelihood of a given training set $\{(\mathbf{x}^{(\mathrm{k})}, \mathbf{y}^{(\mathrm{k})})\}$:

$$L(\lambda) = \Sigma_k \log p_\lambda(\mathbf{y}^{(k)} \mid \mathbf{x}^{(k)}) = \Sigma_k[\lambda \cdot \mathbf{F}(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}) - \log Z_\lambda(\mathbf{x}^{(k)})].$$

This function is concave in λ, and so the maximum can be found at the point where the gradient L is zero:

$$0 = \nabla L = \Sigma_k[\mathbf{F}(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}) - \Sigma_y \mathbf{F}(\mathbf{x}^{(k)}, \mathbf{y}) p_\lambda(\mathbf{y} \mid \mathbf{x}^{(k)})].$$

The left side is the empirical average of the global feature vector, and the right side is its model expectation. The maximum is reached when the two are equal:

$$(^*)\Sigma_k \mathbf{F}(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}) = \Sigma_k \Sigma_y \mathbf{F}(\mathbf{x}^{(k)}, \mathbf{y}) p_\lambda(\mathbf{y} \mid \mathbf{x}^{(k)}).$$

Straightforwardly computing the expectations on the right side is infeasible, because of the necessity of summing over an exponential number of label sequences $\mathbf{y}$. Fortunately, the expectations can be rewritten as

$$\Sigma_y \mathbf{F}(\mathbf{x}, \mathbf{y}) p_\lambda(\mathbf{y} \mid \mathbf{x}) = \Sigma_{i=1,n} \Sigma_{y, y' \in L} p_\lambda(y_i = y, y_{i+1} = y' \mid \mathbf{x}) \mathbf{f}(\mathbf{x}, \mathbf{y}, i),$$

which brings the number of summands down to polynomial size. The probabilities $p_\lambda(y_i = y, y_{i+1} = y' \mid \mathbf{x})$ can be computed using the forward and backward variables:

$$p_\lambda(y_i = y, y_{i+1} = y' \mid \mathbf{x}) = Z(\mathbf{x})^{-1} \alpha_i(\mathbf{x}, y) M_i(y', y, \mathbf{x}) \beta_{i+1}(\mathbf{x}, y').$$

GIS can be used to solve the equation (*). A particularly simple form of it further requires that the total count of all features in any training sequence be constant. If this condition does not hold, a new *slack* feature can be added, making the sum equal to a predefined constant S:

$$s(\mathbf{x}, \mathbf{y}, i) = S - \Sigma_i \Sigma_j f_j(\mathbf{x}, \mathbf{y}, i).$$

If the condition holds, the parameters λ can be adjusted by

$$\lambda: = \lambda + \Delta\lambda,$$

where the $\boldsymbol{\Delta\lambda}$ are calculated by

$$\Delta\lambda_j = S^{-1} \log\ (\text{empirical average of } f_j / \text{ modelexpectation of } f_j).$$

VII.6 FURTHER READING

Section VII.1

For a great introduction on hidden Markov models, refer to Rabiner (1986) and Rabiner (1990).

Section VII.2

Stochastic context-free grammars are described in Collins (1997) and Collins and Miller (1998).

Section VII.3

The following papers elaborate more on maximal entropy with regard to text processing: Reynar and Ratnaparkhi (1997); Borthwick (1999); and Charniak (2000).

Section VII.4

Maximal entropy Markov models are described in McCallum et al. (2000).

Section VII.5

Random markov fields are described in Kindermann and Snell (1980) and Jain and Chellappa (1993). Conditional random fields are described in Lafferty et al. (2001) and Sha and Pereira (2003).