Statistical NLP: course notes

Çağrı Çöltekin — SfS / University of Tübingen

2020-07-13

These notes are prepared for the class *Statistical Natural Language Processing* taught in Seminar für Sprachwissenschaft, University of Tübingen.

9 *Sequence learning*

The machine learning methods we discusses so far are built on the assumption that the training instances are *independent and identically distributed* (i.i.d). When this assumption is reasonable, each prediction can be made independently of other predictions. For example, whether current email is spam or not unlikely to depend on the prediction on the previous email. In most realistic scenarios, it is fairly reasonable to make each decision in isolation.

In some problems, however, individual predictions are not independent of each other. For example, consider the task of predicting the part-of-speech (POS) tagging where the aim is to assign part-of-speech tags, such as `NOUN` or `VERB` to words. POS assignment for a large number of words is ambiguous. For example the word *book* is a `NOUN` in *He read a book*, while it is a `VERB` *Please book a flight*. An important fact to note for our purposes here is that the choice of POS tag depends on (the POS tags of) the other words in the sentence. For example, if the previous word is *a*, or more generally, if it is a determiner (`DET`), the word is most likely a `NOUN`. The correct prediction cannot be made individually for each word.

For another example, consider the example sentence in (1).[1]

(1)

The	*old*	*man*	*the*	*boat*
DET	ADJ	NOUN	DET	NOUN
DET	NOUN	VERB	DET	NOUN

Among others, POS tag assignment to the words *old* and *man* have multiple options, resulting in two alternative POS tag sequences displayed above. The first sequence, with *old* identified as an adjective and *man* identified as noun is more likely if we consider POS assignment in isolation. The word *old* is used much more frequently as an adjective than as a noun. Similarly, the word *man* is a noun (rather than a verb) in its typical usage. However, the second sequence of POS tags are much more likely than the first one.[2] The correct POS assignment is only possible if we pay attention to the whole sequence.

The methods we discuss in this lecture are methods that are suitable for learning sequences with dependences. We will use the POS tagging as our running example throughout this lecture. However, these methods have wider application areas in NLP and also in many other related or unrelated fields. Examples include,

- Predicting weather conditions. For example, the fact that it rained

[1] This is an example of a *garden path* sentence, where people also have difficulties interpreting them at first.

[2] We can even say that the first sequence is not possible in a grammatical sentence.

today is an important predictor for rain tomorrow since rainy days often follow each other.
- Gene predictions from a genome also is an example where predictions are not independent of each other.
- Getting closer to home, optical character recognition also makes use of dependencies between the letters predicted.
- Some other problems that does not look like sequence prediction at first sight may also be cast as sequence prediction tasks. An example we will revisit later is tokenization or segmentation, where the task is formulated as labeling each characters as being at a boundary or in a word-internal position. Again, the prediction, whether an item is at a boundary or not depends on earlier predictions. Hence, modeling the dependencies between the labels is useful.

Hopefully, the examples above are enough to convince you that in these problems, best independent predictions are likely to lead wrong predictions. One way to solve the problem is to consider each possible sequence of labels as an independent label.[3] That is, for a given sentence of length five as in our example, consider each possible POS tag sequence of length five as an atomic, indivisible label. However, this clearly makes it difficult to use the information based on individual words. Furthermore, expands the label space exponentially. If we had 10 POS tags, for example, we would have 10^5 possible possible POS tag sequences for a 5-word sentence, growing exponentially with the length of the sequence.[4]

[3] For example, `DET-ADJ-NOUN-DET-NOUN` in our example can be considered a single label.

[4] This exponential growth is an important consideration for the models we will discuss in the rest of this lecture.

In this lecture, we will discuss some 'traditional' methods, mainly *hidden Markov models* (HMMs) for modeling sequences. We will also introduces so-called *maximum-entropy Markov models* (MEMMs) and *conditional random fields* (CRFs) briefly, mainly by pointing out the differences between them and the HMMs. The typical ways to model sequences using neural networks will be discussed later.

The problems noted above have two parallel sequences. For example, in POS tagging, we need to model the relation between the words and POS tags. Often one of the sequences is *observed*, e.g., the words, while the other is *hidden* or *latent*, e.g., the POS tags are not something we directly observe in natural language sentences. Hence, as well as being sequence models, the models we listed above are all *latent variable* models. To make the introduction easier, we will start with an approach to model sequences without hidden variables, then extend our discussion to the models with latent variables.

9.1 Markov chains

Markov chains or *Markov models* are probabilistic models of sequences where the observation at time t only depends on a limited history of previous observations (as opposed to all of the previous observations).[5]

[5] Since temporal sequences are one of the most common sequences, we often refer to observation at certain time steps. However, the sequences modeled do not have to be temporal sequences. The models we discuss here are equally applicable if the sequence does not have any temporal interpretation.

We introduce Markov chains using, *language models*, one of the basic tools in natural language processing. The aim of a language

model is to assign probabilities to sequences, most typically to the sequences of words. Returning to the our earlier example, we want to assign a probability to a sentence like *The old man the boat*. Intuitively, the probability of this sentence should not be as high as less 'perplexing' examples like *The old man sleeps*, or *The old man painted his boat*. However, the probability should also be (much) higher than ungrammatical examples like **The old woman the boat*, or semantically (more) implausible examples like *The old man the book*.

In summary, we want to know $\mathrm{P}(\textit{The old man the boat})$. As a brute-force solution to find the maximum-likelihood estimate, we may consider counting the number of times this sentence occurs in a large corpus, and divide it to the total number of sentences. However, this will clearly not work as many reasonably high-probability sentences will never occur even in very large corpora.[6] The solution is, factoring/decomposing the probability of a sentence in terms of probabilities of words in it. Clearly, the probability of observing a word in a sentence is not independent of other words. Nevertheless, we can factor $\mathrm{P}(\textit{The old man the boat})$ as[7]

$$\mathrm{P}(\textit{The}) \times \mathrm{P}(\textit{old} \mid \textit{The}) \times \mathrm{P}(\textit{man} \mid \textit{The old}) \times \mathrm{P}(\textit{the} \mid \textit{The old man}) \times \mathrm{P}(\textit{boat} \mid \textit{The old man the}) \,.$$

This factorization of the probability is simply an application of the chain rule of probabilities. Like any other factorization based on the chain rule, the result is still the exact joint probability. However, estimating the final conditional probability still requires observing the whole sentence.

If we model the process using a Markov chain, in particular, if we assert the conditional independence (assumption) that the probability of a word is independent from other words given the previous word, the same factorization becomes,[8]

$$\mathrm{P}(\textit{The}) \times \mathrm{P}(\textit{old} \mid \textit{The}) \times \mathrm{P}(\textit{man} \mid \textit{old}) \times \mathrm{P}(\textit{the} \mid \textit{man}) \times \mathrm{P}(\textit{boat} \mid \textit{the}) \,.$$

In such a *first-order* Markov model for modeling sentences, we need estimation of conditional probabilities of words given only the previous word. These conditional probabilities are much easier to estimate from a corpus, since it only depends on observations of bigrams (two-word sequences).[9]

More formally, a first-order Markov chain is defined by

- a set of states $\mathrm{Q} = \{q_0, q_1, \ldots, q_k\}$. Each state corresponding to a word type in the lexicon for our language model example.
- q_0 is a special state *start state*
- and a matrix A,

$$\boldsymbol{A} = \begin{bmatrix} a_{01} & a_{02} & \ldots & a_{0k} \\ a_{11} & a_{12} & \ldots & a_{1k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \ldots & a_{kk} \end{bmatrix}$$

[6] Theoretically, the number of sentences in a language is unbounded. However, even without reference to potentially infinite number of sentences, the combinatorial nature of forming sentences by combining words from a lexicon results in a very large space (exponential in the length of the sequence) of possible sentences that observing a particular sentence is always going to be a very rare event.

[7] We are taking some (notational) shortcuts here. For example, $\mathrm{P}(\textit{The})$ here means the probability of observing *The* as the first word. Similarly, $\mathrm{P}(\textit{old} \mid \textit{The})$ means observing *old* given the previous word is *The*. A less ambiguous notation would be, $\mathrm{P}(w_1 = \textit{The})$ and $\mathrm{P}(w_2 = \textit{old} \mid w_1 = \textit{The})$, where w_1 and w_2 refer to first and the second word in a bigram respectively.

[8] The assumption in this case is clearly not correct. The usage/probability of words depend on more than a few previous words. However, as in any model, the goal is finding 'useful' models, and even the simple bigram models have been useful in many tasks.

[9] As a result, a first-order Markov model over words is called a *bigram language model*. *Trigram* language models are also common in practice, and n-gram models of 'n' up to five are sometime used by combining them with lower-order n-grams. We will return to the discussion of the n-gram language models in more detail later in this course.

where each element of matrix a_{ij} is the transitions probability from state q_i to state q_j. Note that, the first column of the matrix is all 0s (there are no transitions to q_0), and not included in the above matrix.

Given a set of sequences, it is relatively trivial to estimate the probabilities in matrix A, for example by using maximum-likelihood estimation, we count the number of times transitions from q_i to q_j occurs, and divide it to the number of times q_i is visited. For higher-order Markov models, the rows of the transition matrix A correspond to sequences of states. However, we still need as many columns as the number of states. For example, the paramters of a second-order Markov model with n states is defined by a matrix of $n^2 \times n$.

9.2 *Hidden Markov models*

Hidden Markov models (HMMs) are models of two sequences, one of the sequences are *hidden*. The sequence of hidden variables in an HMM forms a Markov chain. The non-hidden, or *observed* sequence is assumed to be independent of the other variables in the model given the associated hidden variable.

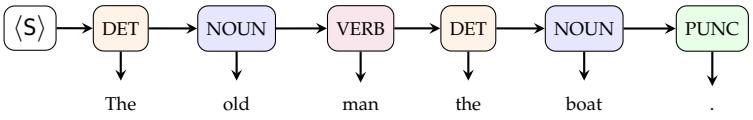


Figure 9.1: An example HMM POS tagger. The states are the POS tags, and the observed sequence is the words (tokens). The state ⟨S⟩ is a special 'start state'.

Figure 9.1 presents a visualization of a first-order HMM for POS tagging. The hidden states (POS tags) form a Markov chain. Each state is independent of others given the previous state, and each observation (word) depends only on the corresponding state. The general intuition here is that in this sentence, we can replace a particular word with another one as long as they belong to the same category. For example, we can replace the word *old* with another adjective, e.g., *young* or *strong*. The probability assigned to the sentence will, then, change based only on probability of the particular word given the HMM is in the `ADJ` state. Clearly, not all independence assumptions in the model is realistic. The model may assign even a higher probability of the sentence *The old man the book*, assuming that *book* is a more common noun than *boat*. This is due to the fact that the dependence between the `VERB` *man* and the noun is not part of this model. Despite the shortcomings, HMMs are useful models for many (NLP) problems.

From this informal introduction, you should have already gathered that HMMs are probabilistic models. In particular, they model the joint probability $P(\mathbf{o}, \mathbf{q})$ distribution over a sequence of hidden states, $\mathbf{q}$, and a sequence of corresponding observations $\mathbf{o}$. This joint distribution is factorized such that $P(\mathbf{q}, \mathbf{o}) = P(q)P(\mathbf{o} \mid \mathbf{q})$. This factorization is simply application of chain rule of probabilities, and model any pair of sequences without loos of generality. The interest-

ing part of the HMMs are simplification of this model because of the independence assumptions asserted by the models structure. Making the independence assumptions of an HMM model explicit, the joint probability becomes

$$\begin{aligned} P(\mathbf{o}, \mathbf{q}) &= P(o_1, \ldots, o_n, q_0, \ldots, q_n) \\ &= \left[\prod_1^n P(q_t|q_{t-1})\right] \prod_1^n P(o_t|q_t). \end{aligned} \quad (9.1)$$

This formulation also clarifies the parameters of an HMM model. More formally, an HMM is defined by

- A set of states $Q = \{q_0, \ldots, q_k\}$, where q_0 is a special 'start state'
- A set of possible observations $O = \{o_1, \ldots, o_m\}$
- A *transition* probability matrix

$$\mathbf{A} = \begin{bmatrix} a_{01} & a_{02} & \ldots & a_{0k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \ldots & a_{kk} \end{bmatrix}$$

  where a_{ij} is the probability of transition from state q_i to state q_j. Sometimes the first row of $\mathbf{A}$, the transitions from the start state to each state is indicated with a separate vector of probabilities $\pi = \{P(q_1), \ldots, P(q_n)\}$, in which case the matrix $\mathbf{A}$ above would be a $k \times k$ matrix.

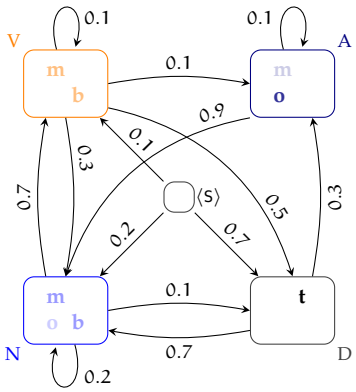
- A matrix of *emission* probabilities

$$\mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & \ldots & b_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \ldots & b_{mn} \end{bmatrix}$$

  where b_{ij} is the probability of emitting output o_i at state q_j.

To make this definition more concrete, a toy HMM POS tagger defined over four part of speech categories, (A)djective, (N)oun, (V)erb and (D)eterminer, and four words, (m)an, (b)oat, (o)ld and (t)he, is depicted in Figure 9.2. The upper part of figure shows a state diagram, much like a weighted finite-state automaton. When the automaton visits a state, picking a path probabilistically, it also emits an output according to the probability distribution given in $\mathbf{B}$. Even if our aim is to classify the items in a sequence, it is generally, intuitive to think of an HMM as a *generative* device.

In our definition, the rows of the matrix $\mathbf{A}$ define conditional distribution of the next state given the current state. The columns of $\mathbf{B}$ define conditional probability distributions of words given POS tags. Since both rows of $\mathbf{A}$ and columns of $\mathbf{B}$ are proper probability distributions (they sum to one), the model is define only on these four POS tags and four words. Also note that, the probability distributions defined in $\mathbf{B}$ are not probability of POS tags given the probability of the



$$\mathbf{A} = \begin{array}{c} \begin{matrix} \texttt{A} & \texttt{N} & \texttt{V} & \texttt{D} \end{matrix} \\ \begin{bmatrix} 0.0 & 0.2 & 0.1 & 0.7 \\ 0.1 & 0.9 & 0.0 & 0.0 \\ 0.0 & 0.2 & 0.7 & 0.1 \\ 0.1 & 0.3 & 0.1 & 0.5 \\ 0.3 & 0.7 & 0.0 & 0.0 \end{bmatrix} \end{array} \begin{array}{l} \langle \texttt{s} \rangle \\ \texttt{A} \\ \texttt{N} \\ \texttt{V} \\ \texttt{D} \end{array}$$

$$\mathbf{B} = \begin{array}{c} \begin{matrix} \texttt{A} & \texttt{N} & \texttt{V} & \texttt{D} \end{matrix} \\ \begin{bmatrix} 0.2 & 0.4 & 0.5 & 0.0 \\ 0.8 & 0.2 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.4 & 0.5 & 0.0 \end{bmatrix} \end{array} \begin{array}{l} \texttt{m} \\ \texttt{o} \\ \texttt{t} \\ \texttt{b} \end{array}$$

Figure 9.2: An example HMM state diagram (top) and corresponding transition ($\mathbf{A}$) and emission ($\mathbf{B}$) probabilities for a toy POS tagger. The letter in each state represents the words, with darker shades indicating higher probability. The labels on the arcs are transition probabilities. The transitions with probability 0 are not drawn.

words. For example, $b_{13} = P(\mathtt{m}|\mathtt{V}) = 0.5$ and $b_{12} = P(\mathtt{m}|\mathtt{N}) = 0.4$. It is important to notice that does not mean that *man* is more likely to be a verb. It is the inverse conditional probability: it means it more likely to output *man* if we are in state V in comparison to when we are in state N. To reverse these probabilities, we need to apply the Bayes' rule.

As presented here, however, the HMM defines multiple probability distributions over equal-length sequences. To define a probability distribution over sequences of any length, we typically introduce another special symbol marking the end of sequences.[10]

[10] We will discuss this issue further when we introduce language models.

From the definition of the HMM above, it is clear how to assign probabilities to pairs of state and output of sequences. If we have both sequences, and have an HMM defined by the $\mathbf{A}$ and $\mathbf{B}$ matrices in Figure 9.2, all we need to do is put the relevant quantities in Equation 9.1 and calculate it. This calculation is also linear in the length of the sequence (its complexity is $O(n)$, since it requires $2n + 1$ multiplications). However, assigning probabilities to pairs of sequences is not the typical usage of HMMs.

HMMs are used for two common scenarios. First, they can be used for sequence labeling tasks as in our POS tagging example. This is also called the *decoding* problem. Here, we only have access to the output (the words), and we want to determine the most likely state sequence (POS tags). Second common usage of HMMs, sometimes called *evaluation*, is to assign probabilities to observation sequences. Again, we only have access to the observation sequence, and calculate the probability of the sequence by summing the probabilities over all possible state sequences. Both tasks can be computationally expensive. And, the same problems will surface when we want to *learn* the parameters of an HMM from data.

To appreciate the computational complexity, and discuss the solutions to the problems above, another way of visualizing an HMM is by 'unfolding' the sequence is useful. The unfolded version of the HMM in Figure 9.2 is presented in Figure 9.3. In this representation, each column of nodes represent one time step in the sequence. We will use this representation for discussing the use cases noted above.

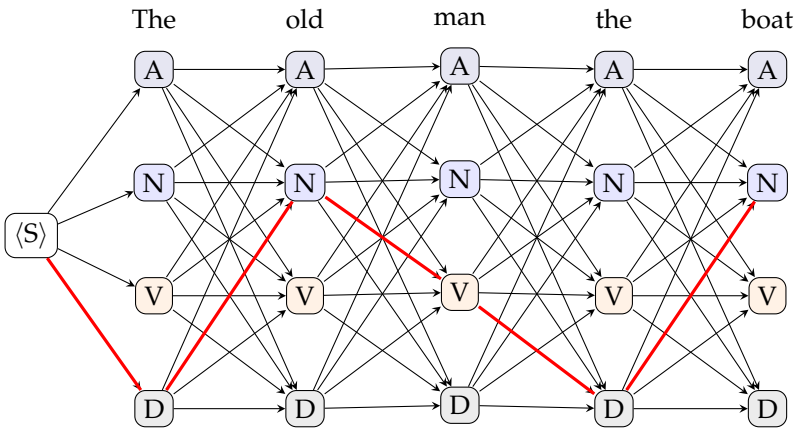


Figure 9.3: An example HMM state lattice (or trellis). The path representing the state sequence of the example in Figure 9.1 is indicated with thick red lines.

9.2.1 *Decoding*

In the decoding problem, given an observation sequence, we are interested in finding the most likely state sequence according to the model. More formally, the task we want to solve is to find the best sequence $\hat{\mathbf{q}}$ such that

$$\hat{\mathbf{q}} = \arg\max_{\mathbf{q}} P(\mathbf{o}, \mathbf{q}) .$$

The solution requires a search for the best state sequence among all state sequences. A naive attempt to solve this problem would enumerate all possible POS tag sequences, and select the sequence with the highest $P(\mathbf{o}, \mathbf{q})$, which corresponds to following all possible paths in the lattice in Figure 9.3, and selecting the path with the highest probability. However, it is easy to see that there are exponentially many paths in the lattice. With k possible states, the number of all all possible hidden state sequences (or equivalently all paths in the trellis) is k^n for an observation sequence of length n. For our example, there are $4^5 = 1\,024$ possible POS tag sequences, and every additional item in the sequence increases this number by four. For example, for a sequence of 20 words we get $4^{20} = 1\,099\,511\,627\,776$ alternative sequences. Like any problem with exponential complexity, this makes the naive solution intractable.

Fortunately, there is a dynamic programming solution, commonly known as the *Viterbi algorithm*, that makes the decoding problem tractable. The solution is possible because of the structure of the model. Notice that the naive approach recalculates the probabilities of sequences leading to a particular state for each possible state at each point. For example to find the best POS tag at the end of the sequence in Figure 9.3, we need compare probability of sequences leading to the four possible POS tags. However, the highest-probability sequence leading to each of the previous states (POS tags) are the same for all four final tag options. Hence, we only need to know what is the best sequence leading to each of the previous POS tags. As a result, if we record the best probability leading to each node, we do not need to re-calculate all the paths leading to this node again for determining the best probabilities of later nodes.

Formally, at each time step t the maximum likelihood of the sequence with state j is

$$v(t, j) = \max_{i \in Q} v(t-1, i) a_{ij} b_{j o_t}$$

where Q is the set of states (POS tags) a_{ij} is the transition from state i to state j, and $b_{j o_t}$ is the emission probability of observation o_t given the state is j. Except $v(t-1, i)$, which the maximum likelihood leading to state i in the previous time step, the other values in the right hand side of the formula are the values from the $\mathbf{A}$ and $\mathbf{B}$ matrices defining the HMM. If we store these likelihoods at every node of the HMM trellis, we can simply perform the calculations in $O(n \times k)$ for sequence length n and number of states k. The cost is

an additional space complexity of $O(n \times k)$, since we need to store these likelihoods. The best sequence is the sequence with the highest likelihood at the final step. However, note that to recover the exact sequence that leads to the final maximum likelihood, we also need to store 'backlinks', the state in time step $t-1$ that leads to the maximum likelihood at time step t.

If we, calculate the maximum probability leading to the output *old* as a noun in our example, we need the maximum probabilities leading tho each of the POS tags in the first step with output *t*. We can easily calculate these by multiplying $a_{0,i}$ and $b_{i,t}$ for each POS tag i. These likelihoods will be 0.00 for all tags except for determiner in our example since probability of observing *the* in other states is 0.00 for POS tags (states) other than D according to the emission probabilities in **B**. The likelihood of D in step 1 is $0.7 \times 1.0 = 0.7$. We store all these values, and then we can calculate the likelihoods for time step 2, without recalculating any of the previous steps. Now, if we want to calculate the likelihood N in the second time step, we simply multiply values stored earlier, transitions probabilities from each of the earlier stats to N and pick the maximum probability among them: $0.7 \times 0.7 \times 0.2$. Note that likelihood of A in this time step is higher ($0.7 \times 0.3 \times 0.8$). We store all these values, including the previous state that leads to the maximum likelihood, and continue processing until the end of the string. An important aspect of these calculations is that even though an initial sequence of tags may be much more likely than others, a later transition may change the sequence with the maximum likelihood. For example, the most likely sequence at step 3 is DAN in our example. However the low transition probability from N to D may change the best sequence at time step 4 DNVD.[11]

$$\mathbf{A} = \begin{array}{c} \begin{array}{cccc} \texttt{A} & \texttt{N} & \texttt{V} & \texttt{D} \end{array} \\ \begin{bmatrix} 0.0 & 0.2 & 0.1 & 0.7 \\ 0.1 & 0.9 & 0.0 & 0.0 \\ 0.0 & 0.2 & 0.7 & 0.1 \\ 0.1 & 0.3 & 0.1 & 0.5 \\ 0.3 & 0.7 & 0.0 & 0.0 \end{bmatrix} \end{array} \begin{array}{c} \langle \texttt{S} \rangle \\ \texttt{A} \\ \texttt{N} \\ \texttt{V} \\ \texttt{D} \end{array}$$

$$\mathbf{B} = \begin{array}{c} \begin{array}{cccc} \texttt{A} & \texttt{N} & \texttt{V} & \texttt{D} \end{array} \\ \begin{bmatrix} 0.2 & 0.4 & 0.5 & 0.0 \\ 0.8 & 0.2 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.4 & 0.5 & 0.0 \end{bmatrix} \end{array} \begin{array}{c} \texttt{m} \\ \texttt{o} \\ \texttt{t} \\ \texttt{b} \end{array}$$

Figure 9.4: The example HMM parameters from Figure 9.2. Repeated for convenience.

[11] You need to perform the calculations to see if this is true for our example parameters in Figure 9.2.

In some applications, we may want to obtain not only the best sequence according to the model, but a ranked list of highly-probable sequences. In this case, instead of storing only a single best likelihood at each node, we can store a pre-defined number of best likelihood values, and enumerate the best options at the end of the output sequence. This is an example of a *beam search* that finds not only the best result, but n-best result according to the model. A very common practice in the field is to use a generative model like an HMM to produce a ranked list, and re-rank this list with a discriminative model using other external or global features that are difficult to incorporate into a generative model's input.

9.2.2 *Assigning probabilities to output sequences*

Another common use of HMMs is to assign probabilities to output sequences. This can, for example, be useful for assigning probabilities to utterances or sentences in speech recognition and machine translation formulated as noisy channel problems.

The problem, and the solution, is similar to decoding. Instead of storing the maximum probability leading to a particular node, we store *forward probabilities* $\alpha_{t,i}$ at each node, which are the total

probability of observing the output o_t and being in state i. More formally,

$$\alpha_{t,i} = \sum_{j=1}^{k} \alpha_{t-1,j} P(q_i|q_j) P(o_t|q_i)$$

where k is the number of states. Note that calculation of forward probabilities at teach step only depend on the previous one. Hence, storing these values, as in decoding, makes it a tractable problem. This version of the dynamic programming algorithm is sometimes called the *forward* algorithm.

Once we have the forward probabilities, the probability of a particular observation sequence can be calculated using the forward probabilities at he final time step n.

$$P(\mathbf{o}|M) = \sum_{j=1}^{|Q|} \alpha_{n,j}$$

9.2.3 *Learning HMMs*

If we have supervised input, that is pairs of observation and (hidden) state sequences, estimating the HMM parameters is easy. Using maximum likelihood estimation, we can simply count and divide the appropriate values:

$$a_{ij} = \frac{C(q_i \rightarrow q_j)}{\sum_k C(q_i \rightarrow q_k)}$$
$$b_{ij} = \frac{C(q_i \rightarrow o_j)}{\sum_k C(q_i \rightarrow o_k)}$$

where $C(\cdot)$ the number of times a particular transition occurs for the first formula, and the number of times the specified output is emitted in a given state for the second formula.

The above formulation defines the maximum likelihood estimation. A known problem with maximum likelihood that it overfits the training data. Particularly, the model will assign zero probabilities for transitions and emissions that are not observed in the training data. In practice, it is common to use *smoothed* probability estimates, so that some of the probability mass is reserved for unobserved transition and/or emission events. We will discuss smoothing in detail while introducing the language models.

HMMs can also be trained in an unsupervised manner. We will not go into detailed description of unsupervised estimation of HMMs. However, the most common procedure is using a version of the EM algorithm to find a model that assigns maximum likelihood to the given output sequences. In case of HMMs, the complication again comes from the dependencies between the variables, which makes the computation of E/M steps expensive. However, we already saw the forward algorithm which calculates the likelihood of an observation sequence (this is the main part of the E step). For the M

step, a similar dynamic programming algorithm is used for efficient calculation of the model parameters. The algorithm is known as *Baum–Welch* or *forward-backward* algorithm. Interested readers can find references to descriptions of the algorithm at the end of the chapter.

9.3 Graphical models: a brief divergence

Hidden Markov models are probabilistic models with certain structure. The structure of the model is imposed by the model's (conditional) independence assumptions between the random variables involved. In particular, for HMMs, all output variables are assumed to be conditionally independent of all other given the associated state, and the states depend only on a limited history of earlier states. *Probabilistic graphical models* provide a way to define and display the structure of such models with an intuitive, visual formalism. A brief introduction to graphical models will highlight the differences and similarities between the HMMs introduced in some detail above, and the other alternative models which we will discuss briefly.

A graphical model expresses the structure of a joint probability distribution. Remember that a joint distribution can be factorized in different ways. For example, for three random variables x, y, and z, we can factorize the joint distribution six different ways. Figure 9.5 shows three of these different factorizations and the directed graphical model corresponding each factorization. In graphical models, we represent the random variables with nodes. The directed graphical models are represented with directed graphs as in Figure 9.5. Each variable (node) in a directed graphical model depends only on its parents. As a result, the term $P(x \mid y)$ in the factorized expression results in a directed edge from node y to node x.

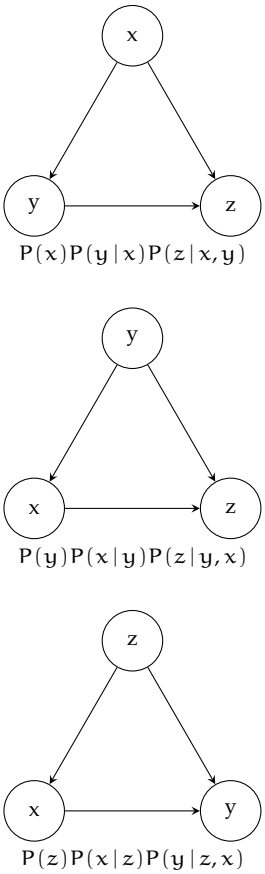


Figure 9.5: Three different ways to factorize the joint distribution of three variables x, y and z, and the directed graphical model describing each factorization.

Directed graphical models provide a visual, intuitive way to represent the structure of a joint probability distribution, when the relations between the variables are causal. For example, the topmost graph in Figure 9.5 indicates that the value of x is (partially) responsible form values we obtain from y and z, and similarly y also has an effect on z. In such models, we can generate data from the model by starting at the node(s) with no incoming edges, and sampling values from these nodes, and then sampling from the child nodes given the samples from the parent node(s). In some applications, it is useful to be able to generate data from the model in this way. However, even if our application does require generating data from the model, thinking about the model in terms of a 'generative story' is gives a better intuition about the model.

As a concrete example, think about the naive Bayes classifier we introduced earlier. The conditional assumption of the naive Bayes classifier is that given the class y, the features $\mathbf{x}$ are independent of each other. Figure 9.6 shows the representation of a naive Bayes classifier as a directed graphical model. The figure also demonstrates two conventions used in graphical models. The observed variables

are indicated by shaded nodes, and when there are many variables of the same independence properties, only one of them are drawn within a square box (called a 'plate') which also indicates the number of variables.

The graph representation in Figure 9.6 indicates the conditional independence assumption of the naive Bayes classifier in a visual, intuitive way. This also defines a generative story: we first determine the class label by sampling from $p(y)$ (e.g., whether to create a spam or non-spam document), and then we sample features (e.g., words) from the distribution $p(x \mid y)$. We are, in most cases, interested in the inverse conditional probability, $p(y \mid x)$. However, the generative point of view is often useful for understanding the model.

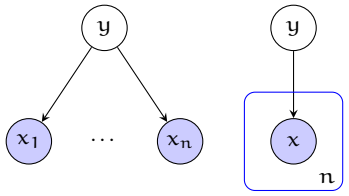


Figure 9.6: Representation of a naive Bayes classifier with n features as a graphical model. Both graphs show the same model. The one on the left lists the multiple variables (features) explicitly, while the one on the right uses the 'plate notation' for indicating multiple variables with the same independence properties in a compact form.

Now we are ready to define HMMs as graphical models. Figure 9.7 presents an HMM following the conventions we described above. In words, each state (q_i) only depends on the previous state, and the output items (o_i) only depend on the corresponding states. Note that we can easily represent additional dependencies. For example we can form a second-order Markov chain by drawing additional directed edges to each node from their 'grandparent' in Figure 9.7.

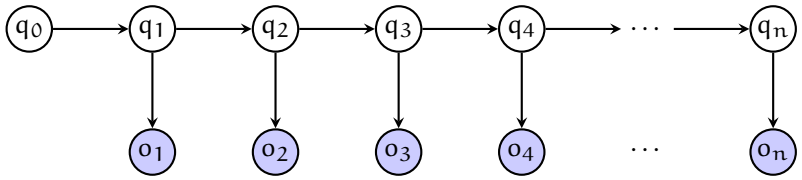


Figure 9.7: Representation of HMMs as graphical models.

In all the directed graphical model examples above, the model assume a sort of order, or causal relationship, between the variables. Sometimes there is no such clear causal relationship. In that case *undirected graphical models*, also called *Markov random fields* is are appropriate. Similar to directed graphical models, undirected graphical models also define (in)dependence relations between individual variables. However, unlike conditional probabilities, we use symmetric functions, called *potential functions*, that indicate the dependence between individual variables. The model, similar to the directed models, define a joint distribution over all variables, factorized based on the model structure.[12]

[12] In general, the factorization of joint probability in undirected graphs are based on subsets of nodes (called cliques). We will not discuss the details of the way the joint probability distribution of a given graph is factorized. Interested readers are referred to the references at the end of the chapter.

Figure 9.8 shows an example undirected graphical model similar to the naive Bayes example from Figure 9.6. The difference here is that the dependences between variables are now specified with a symmetric function $\phi(\cdot)$, rather than conditional probabilities. The joint probability defined by this networks is,

$$p(y, x_1, \ldots, x_n) = \frac{1}{Z} \prod_i^n \phi(y, x_i) \,.$$

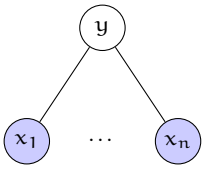


Figure 9.8: An example undirected graphical model.

Since we still have the assumption that the features (x_i) are independent given the class (y), the relations between them do not factor into to expression of the joint distribution. The constant Z normalizes the

joint distribution globally, it is simply a sum over the whole space of variables. Except for small problems, calculation of Z is intractable. As a result, most practical applications involve approximate estimation of the joint probability.

The only requirement on the choice of potential functions is that they have to return positive numbers. Otherwise, one is rather free to chose the function, and the features that affect the value of the potential function. In practice, the most common choice is exponential functions, since they return positive values.

The undirected models are, in general, more flexible, and considered more powerful. This also means that they are computationally more expensive to train. The sets of joint distributions that can be expressed by directed and undirected graphical models are not the same, but they intersect. Some distributions can be expressed by both, some only by directed or undirected graphical models.

9.4 *Alternative models for sequence labeling*

Hidden Markov models are well-known, well-studied models for sequential data. They have been used in many fields, including computational linguistics. Since they are generative models, they can also generate data, or more importantly, assign probabilities to the observation sequences. However, as in our part of speech tagging example, their most common usage has been sequence labeling. In sequence labeling we do not need to model the joint distribution. What we really need is $p(\mathbf{q} \mid \mathbf{o})$, probability of the state sequence given the observation. Hence, as noted earlier during the discussion of differences between generative and discriminative models, they put some extra modeling effort that is not required for sequence modeling. As a result, there are a number of discriminative sequence labeling, which often perform better than HMMs on sequence labeling tasks. We will briefly discuss two of these alternatives, *maximum-entropy Markov models* and *conditional random fields*, briefly. Now that we have a basic understanding of graphical models, it will be easy to describe these models based on what we know about HMMs.

9.4.1 *MaxEnt HMMs*

Maximum-entropy (hidden) Markov models (MEMMs), or conditional Markov models, are very similar to the HMMs. MEMMs also model the state sequence as a Markov chain. However, instead of estimating $p(\mathbf{o} \mid \mathbf{q})$, they directly estimate $p(\mathbf{o} \mid \mathbf{q})$. Figure 9.9 shows the graphical model for a simple MEMM. The difference between an

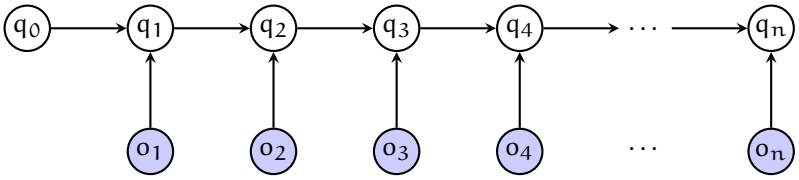


Figure 9.9: A simple maximum-entropy hidden Markov model (MEMM).

HMM and MEMM is simply the direction of the conditional dependencies between the states and the observations. The advantage of using an MEMM, however, is not directly visible from this picture. The advantage comes from the fact that we now can estimate the conditional probability $p(\mathbf{q} \mid \mathbf{o})$ directly using a discriminative probabilistic model. In MEMMs, *maximum entropy* models, or multi-class logistic regression, is used for this task. Once we do that, we can use any feature we can extract from the observation sequence as a predictor. For example, if a word starts with a capital letter, this is a good cue for its POS tag in many languages. However, it is not easy to include such features in an HMM model without increasing the model complexity. In an MEMM, any feature we can extract from any part of the observation sequence can be included as feature for the logistic regression model predicting $p(\mathbf{q} \mid \mathbf{o})$. As a result, a more representative graphical model describing the MEMM models in practice is given in Figure 9.10.

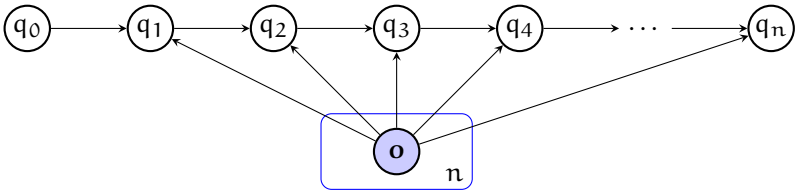


Figure 9.10: Another way to represent an MEMM as a graphical model. Note that now we can include features from all items from the observed sequence.

Overall, an MEMM models the conditional probability distribution

$$p(\mathbf{q} \mid \mathbf{o}) = \prod_i p(q_t | q_{t-1}, o_t)$$

The conditional probability $p(q_t | q_{t-1}, o_t)$ is estimated using a maximum entropy (multi-class logistic regression, or softmax) classifier. This means estimating the parameters w_i in [13]

$$p(q_t | q_{t-1}, o_t) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(q_t, q_{t-1}, \mathbf{o})\right) \tag{9.2}$$

where Z is the normalization constant that makes sure that the probabilities of all classes sum to one.

MEMMs are discriminative models. Since they have the flexibility of including extra surface features, and the fact that they do model the data unnecessarily (for sequence labeling), they are generally better than HMMs in sequence labeling tasks. However, this also means that they cannot be used for tasks where assigning probabilities to observations is required.

[13] In practice, all we need to do is, to fit a multi-class logistic regression model. The formula is included here as it provides some insights into the relation between the logistic regression, and the CRFs we will discuss next.

9.4.2 *Conditional random fields*

Conditional random fields (CRFs) are another popular model for sequence labeling. CRFs are related to the Markov random fields described above. However, instead of modeling the joint distribution $P(\mathbf{o}, \mathbf{q})$, CRFs model the conditional distribution $P(\mathbf{q} \mid \mathbf{o})$. CRFs can

specify different independence structures depending on the problem to be modeled. For sequence labeling problems, we are interested in modeling a linear chain structure, which are called *linear chain* CRFs. Since this structure is prevalent in NLP, the term CRF almost exclusively refers to a linear chain CRF. A graphical representation of a linear chain CRF is given in Figure 9.11.

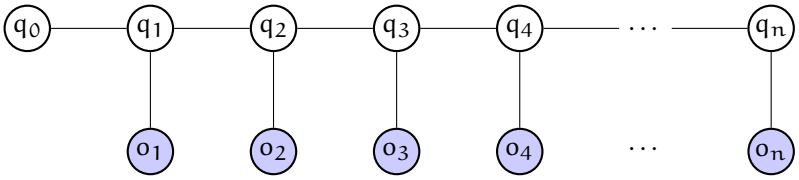


Figure 9.11: A linear chain conditional random filed model.

Given the above representation, the *conditional probability* modeled by the CRF is factorized as

$$P(\mathbf{q} \mid \mathbf{o}) = \frac{1}{Z} \prod_{i}^{n} \phi(q_{t-1}, q_t, o_t) .$$

In theory, similar to Markov random fields, we can use any positive function $\phi(\cdot)$, in the above formulation. However, in practice, an exponential function of linear combination of features is used, resulting in

$$P(\mathbf{q} \mid \mathbf{o}) = \frac{1}{Z} \prod_{i}^{n} \exp\left(\sum_{j} w_j f_j(y_{i-1}, y_i, \mathbf{o}) \right) .$$

where $f_j(\cdot)$ are feature functions (often binary features extracted from the observation sequence and the past states); w_j is the weight associated with the corresponding feature j; j ranges over feature indexes; and i ranges over each step in the sequence. Note that, like in MEMMs, we can use features from the whole observation sequence.

You should have also note the similarity with the multi-class logistic regression, and hence the MEMM model defined in Equation 9.2. There is a subtle difference: the CRF model define above normalizes the conditional distribution globally, while MEMM models normalize it locally (the values multiplied are already probabilities in MEMM). This seems to make a difference in practice. The MEMMs exhibit a behavior called 'label bias'. If there is a high-probability transition from a particular label to another, MEMMs tend to disregard the observation and follow this transition, while CRFs are not sensitive to the this problem.

Similar to MEMMs, CRFs are discriminative models for sequence labeling. They often perform better than both HMMs and MEMMs on sequence labeling tasks. However, they also are computationally more demanding. Again, since they are discriminative models, they are not suitable for tasks where we need to assign probabilities to observation sequences.

Summary

In this lecture we covered a number of models designed particularly processing sequences. Hidden Markov models are generative models that model the joint probability of a hidden state sequence, and an observation sequence. They are important tools in many disciplines, including in various tasks in NLP. Since HMMs model the joint distribution of the observation and the hidden state sequence, they can assign probabilities to the observed data. This is useful for applications such as language models for speech recognition or machine translation where estimating probabilities of utterances or sentences are important. For sequence labeling, however, the strong independence assumptions made by HMMs cause low performance in many tasks.

We reviewed two other models, MEMMs and CRFs which are in general better sequence labeling models than HMMs. Particularly CRFs has been very influential since their introduction (Lafferty, McCallum, and Pereira 2001). Even though the common choice for many sequence labeling tasks nowadays are (recurrent) neural networks, it is also common to use CRFs in combination with neural networks as well. We will discuss neural networks for sequence processing in another lecture.

We covered some of the computational issues with sequence processing only in the context of HMMs. Most of the ideas, e.g., use of Viterbi algorithm for decoding, is applicable to other methods we discusses as well. For a more detailed and formal introduction, the reader is referred to machine learning textbooks such as Bishop (2006) and Murphy (2012).

Bibliography

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer. ISBN: 978-0387-31073-2.

Lafferty, John D, Andrew McCallum, and Fernando CN Pereira (2001). "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 282–289.

Murphy, Kevin P. (2012). *Machine Learning: A Probabilistic Perspective*. Adaptive computation and machine learning series. MIT Press. ISBN: 9780262018029.