Dialogue-based disambiguation
Koeling, Robert Wietse

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2002

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Download date: 25-04-2020
Chapter 4

Introduction to Maximum Entropy modeling

Noam Chomsky is well known for his scepticism about the value of statistics in language studies.

It must be recognized that the notion of a 'probability of a sentence' is an entirely useless one, under any interpretation of this term. (Chomsky, 1969)

Chomsky was the most influential authority in linguistics at that time, whose opinion had a big impact on the linguistics research agenda. As a consequence of Chomsky's opinion the idea of incorporating probability theory in linguistics did not receive serious attention for years.

Even though people might have difficulties with the intuition of the 'probability of a sentence', probability theory and language are much more easily intertwined when the point of view is slightly changed. About twenty years before Chomsky stated his opinion on this subject, the mathematician Claude Shannon wrote an influential paper with the title 'Prediction and the Entropy of Written English'. In this paper he introduces a task, often referred to as the Shannon game, of guessing the next letter in a text given the previous letters (Shannon, 1951). The average number of guesses that is needed to guess the correct letter that follows a randomly chosen string (of a fixed number of characters) from a text is used to estimate the information density (or entropy, see 4.1.5) of a particular text. By taking representative samples of text of a particular language it is possible to estimate the entropy of that language.

A variant of the game does not involve characters but words. Let a subject make a guess of the next word in a text given a string consisting of a fixed number of preceding words. It is easy to see that not only both probability and linguistic theory are issues here, but also that the combination of both notions can be more informative than either could be individually. Consider the example where the word that follows the string: 'The hungry cat chases the' has to be predicted. Not only will there be a preference for a word that
is semantically close to the preceding words, but also for a word that has the right function in the sentence. A noun like mouse would be very suitable and likewise an adjective like fat or frightened. A determiner or an unrelated noun like glasses might be possible, but is very unlikely. There is a strong correlation between some linguistic features of the candidates for the word and the probability of a particular word being chosen. A pure probabilistic language model can benefit from linguistic theory, because linguistic theory can rule out many candidates and therefore reduce the search space enormously. A language model based only on linguistic theory would benefit from probabilistic theory because it can take into consideration not only what is possible according to the model, but in case of many possibilities, also give preference for one over others.

In the previous chapter I have examined a knowledge-based theory to decide whether a particular hypothesis given by the speech recognizer is a suitable candidate in a certain dialogue state or not. The main problem with this approach was the fact that often several candidates are suitable, but some candidates are just preferable over others. In the next chapter I will investigate how a statistical approach might help to tackle that problem. In this chapter I will introduce Maximum Entropy (Maxent) Modeling, the statistical framework that I use here. First, I will say a few words about probabilistic modeling in general and explain why Maxent is a suitable framework to tackle this particular problem. Then I explain what a Maxent model is. I will conclude with pointing out its virtues and vices.

4.1 Probabilistic Preliminaries

4.1.1 Probabilistic modeling

First we will have to say something about what we want from probability theory. In this work (and in most statistical NLP) we want to use statistics to make observations about regularities in language use and learn how to use these regularities to make predictions about new observations. We want to learn from how language was used in the past and we will use text samples to create a simplified model of complex dependencies in language. For serious, real world applications, it will necessarily be a simplified model. The first reason for this is because we know that we will always generate our model from a very limited sample of the language we want to model. The second reason is that we will often forget about many more complex dependencies for practical (computational) reasons. We will not try to create a model that behaves like humans do, but merely one that mimics that behavior in some restricted ways.

An excellent explanation of why probability theory is a very desirable addition to any science that must deal with incomplete information or uncertain knowledge is given by the physicist R. Feynman (Feynman, 1963) and aptly quoted by Herman Ney in an introduction to statistical methods in speech and language processing (Young and Bloothooft, 1997):
By *chance*, we mean something like a guess. Why do we make guesses? We make guesses when we wish to make a judgment but have incomplete information or uncertain knowledge. We want to make a guess as to what things are, or what things are likely to happen. Often we wish to make a guess because we have to make a decision... Sometimes we make guesses because we wish, with our limited knowledge, to say as much as we can about some situation. Really, any generalization is in the nature of a guess. Any physical theory is a kind of guesswork. There are good guesses and there are bad guesses. The theory of probability is a system for making better guesses. The language of probability allows us to speak quantitatively about some situation which may be highly variable, but which does have some consistent average behavior.

Not much has to be added to this. The reasons why guesses often have to be made are clearly present in our situation. Furthermore, we do believe in the consistent average behavior of the language user and it is very attractive indeed to be able to speak in quantitive terms about it. However, we must always remember that we have to work with incomplete information and therefore we will always end up with a partial description of reality.

Some final observations about probabilistic modeling need to be mentioned here. First, what we want to do is to *generalize* over observations we make in the sample data. This means that in case we encounter a new situation, i.e. a situation not seen as such before, we want to be able to make a prediction using partial information in this situation that we have seen before. So, preferably we do not want to quantify over complete events, but try to split up the event in sub-events (properties of events), collect statistics about these properties and try to estimate the probability of an event using these statistics. In the case where (many of) the properties are described by the model, this strategy will allow us to estimate the probability of an event never seen before. Models are often evaluated in terms of *entropy* (see explanation of the Shannon Game above) or *perplexity*. Both measures are indications of the amount of uncertainty in the model. Adding information about the process we are modeling, will result in a model with lower perplexity. With this model, it will be more likely to make the right prediction in a situation we have never seen before (see sections 4.1.5 and 4.1.5).

A second issue is how to produce a probabilistic model. As said before, a basis for these models is the assumption that language users behave consistently (in the long term). We can translate this assumption into the idea that if you can model language as it was used in the past, it can be used for reliable predictions about language use in the future. In order to do so, we have to collect large samples of representative text and collect statistics about the events we are interested in in these texts. The next assumption is that if the samples are large enough, then the conclusions we draw about the distributions of events in the sample texts will also hold for unseen, new text. However, the dangerous
words in the previous sentence are 'large enough'. Language is so varied and
the lexicon we can use is so big (and always expanding), that we are sure that
for almost every application no sample will be large enough to include every
possible event. This problem is often referred to as the sparse data problem. It
brings us to the final issue we want to mention here: what to do with previously
unseen events? Obviously (because we already know that the sample we used
for our model was limited) it is too crude to conclude that just because we have
never seen this particular event before, the probability of seeing this event is
zero. We know for sure that we overestimate the probability of some events
and underestimate others. The art of redistributing some of the probability
mass in order to correct errors due to sparse data, is called smoothing and is an
important topic in the world of probabilistic modeling.

4.1.2 Language modeling

A classic task for statistical NLP is creating language models. A language model
is closely related to the Shannon game as described above: predict the next
word given the previous words. Although the most frequently used language
models, n-grams (see 4.1.2), are in fact straightforward implementations of the
Shannon game where only a very restricted number of preceding words is used
(normally not more than 3), it is not necessary to restrict the attention to
preceding words. Because the simple n-gram models work so well there is less
interest in looking for other information sources to include in the language
model. There is work however that suggests that looking beyond n-grams can
improve language models significantly (e.g. (Brill et al., 1998)). In the next
chapter we will follow this lead.

n-gram models

N-gram models are simple and very effective models and therefore widely used
in language modeling. The probability of a certain word is calculated by the
frequency of a word preceded by n-1 words in a large sample corpus. In general,
the probability of a particular word preceded by these n-1 words is given by:

\[ P(w_i) = \frac{c(w_{i-n+1}...w_i)}{c(w_{i-n+1}...w_{i-1})} \]

where \( c(x) \) is the count (i.e. how often \( x \) appears in relevant 'training material').
This is called the Maximum Likelihood Estimate (MLE) of the probability of
word \( w_i \) given the fact that \( w_i \) is preceded by \( w_{i-n+1}...w_{i-1} \).
They are based on the observation that the probability of a certain word in a
sentence depends on the preceding words:

\[ P(w_1, w_2, ..., w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1}...w_1) \]

That is, the probability that \( w_i \) is uttered depends on the probability that
the words \( w_1, w_2, ..., w_{i-1} \) were said previously. The good thing about this

\(^1\)Material from which probabilities are estimated.
observation is that it acknowledges the fact that even if words are far from each other they might depend on each other. The bad thing is the fact that it in practice is not possible to consider the complete history. For practical purposes we rely on the Markov Assumption. Under the Markov Assumption the values in any state are only influenced by the values of the state that directly preceded it\(^2\). We approximate the history by looking only at the previous \(n - 1\) words:

\[
P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i|w_{i-(n-1)}, \ldots, w_{i-1})
\]

The estimate of \(P(w_i|w_{i-(n-1)}\ldots w_{i-1})\) is based on the relative frequency of \(w_{i-(n-1)}\ldots w_i\) in a large text that is produced by the same process that we are trying to model. The estimate is calculated as:

\[
P(w_i) = \frac{c(w_{i-(n-1)}\ldots w_i)}{c(w_{i-(n-1)}\ldots w_{i-1})}
\]

where \(c(x)\) denotes the function that counts the occurrences of \(x\) in the sample corpus.

The next thing is of course to determine the value of \(n\). The trick is to find a balance between a value for \(n\) that results in a model that is acceptable in size and at the same time is accurate enough. Mostly \(n\) is restricted to 2 or 3. Even though this means that only a very restricted (and local) context is used, these models perform really well (see for example (Jelinek, 1998)). In this work we will mostly use trigrams:

\[
P(w_i) = \frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-2}, w_{i-1})}
\]

It is in fact surprising that trigram models work so well for modeling language that it is really hard to beat them. We know that language is full of long distance dependencies, which are impossible to capture when only local context is considered. Apparently local constraints are very strong in language. (Borthwick, 1997) cites a nice example that gives an idea why local constraints are so effective. He cites examples from a 139 million words corpus of transcripts of radio and television news shows. In this corpus the (unigram) probability of the word gates is very small (\(P('gates')=0.00001\)), but increases rapidly when it is known that it is preceded by certain words: \(P('gates'|'bill')=0.004\) and \(P('gates'|'chairman'|'bill')=0.47\). The word clinton has a probability of 0.001, but when preceded by the word bill the probability increases enormously: \(P('clinton'|'bill')=0.328\), but there are no examples found in the corpus of the word clinton preceded by the chairman and bill.

A very large corpus is used to estimate the probability of all trigrams that exist within the corpus. The qualification 'exist within the corpus' phrase already

\(^2\)State' can be of arbitrarily complexity. So, in our case, if we want to predict \(p(w_i)\) we do not have restrict ourselves to \(p(w_i|w_{i-1})\), but the previous state can include the previous \(n\) words
suggests the limitation of this approach. Even if we have a large (what is called a training) corpus at our disposal, it is very likely that many trigrams will not occur within this corpus and this approach will not be able to estimate the probability of these trigrams for future use.\textsuperscript{3} This indicates that it is not possible to rely on a big enough\textsuperscript{4} training corpus, but that we need instead to find a way to estimate the trigram probability in a different way when it was not found in the training corpus, or in other words: smoothing is an important issue. For n-grams this is mostly done by defining less informative models that nonetheless provide some information. In the case of trigram modeling, a bigram (i.e. $p(w_i|w_{i-1})$) and a unigram (i.e. $p(w_i)$) model are defined. These models are used to estimate the trigram probability. There is a choice: either we attempt to combine unigram, bigram and trigram estimates throughout the model's estimates, or we use the bigram estimate where too few trigrams have been seen, and we use unigrams where too few bigrams have been seen. Both approaches, called interpolated and backed off models respectively, will be introduced in section 4.1.4.

4.1.3 Classification

Another typical application for probabilistic modeling is classification. The job of a classifier is to decide to which class or category an item belongs given the fact that this item coincides with certain observations surrounding this item (context). The problem may be tackled using what is called a supervised learning technique. The training data for such an approach must be classified by a human expert to provide the examples needed to create the model. The model can then be used to estimate the probability of new data belonging to a certain class. Well known examples of classifiers are for example Part of Speech Taggers, classifiers that assign a part of speech tag (class) to a word using the context of the word (which might be the word itself, the surrounding words or features of these words), or language identifiers, classifiers that guess the language a text is written in (class) using information from this document (which might be word frequencies in this text or combinations of particular words, among other possibilities).

Classifiers and language models

The tasks of language modeling and classification are in fact very similar. So there is good reason to look at classification literature, because they address many problems that are also encountered when building language models. An (n-gram) language model has the form:

$$P(w_i|w_{i-n+1}...w_{i-1}) \text{ or } P(w_i|\text{context})$$

\textsuperscript{3}In section 4.3 of (Jelinek, 1998) Jelinek an early research on trigram models is described in which a trigram model is defined using a 1000 word lexicon. A corpus was divided into a 1,500,000 word training corpus and a 300,000 test corpus. They reported that 23% of the trigrams in the test corpus did not occur in the training corpus.

\textsuperscript{4}A corpus is be 'big enough' if it contains several instances of every event whose probability has to be estimated and is not zero.
and a classifier is specified by:

\[ P(\text{Class}|\text{context}) \]

When we are looking for the best path in a wordgraph, we always have a limited number of \( w_i \)'s to choose from. It does not require much imagination to look at the possible \( w_i \)'s as the classes that can be assigned to the to the \( i \)-th position of the string \( w_{i-n-1}...w_i \). The reason to look at work on classification is the fact that in that line of work it is much more common to use different kinds of information sources.

For the various classification tasks it is crucial to think carefully about what sort of context information gives a clue about the class a given object belongs to. It is the task of the modeller to identify the observations that are important for the process to be modeled. It will often be the case that information sources are completely different. It is easy enough to build separate statistical models for every information source, but we still need to come up with one model that uses all the information supplied by the various components. An important question is then how to combine the evidence given by the various knowledge sources in order to end up with a single estimate for the probability of an object belonging to a certain class.

### 4.1.4 Combining knowledge sources

The decisions in a classification process are based upon evidence found in the context of the item that is classified. The knowledge about the relation between the evidence found and the class that is assigned to the item is the only knowledge we have about and thus the only knowledge we can apply to the classification process. In statistical modeling we will always work with partial descriptions of the process we are trying to model. In most cases one particular information source is not enough to decide. In the previous section we have looked at \( n \)-gram language models. We saw that, although the previous word is a valuable indication for what the next word will be, in most cases it is not conclusive. Therefore a second or third information source might be consulted and the information gained from the various knowledge sources must be combined in a single probability estimate.

I will use the \( n \)-gram example in the rest of this section to show how traditionally the various information sources are combined in interpolated and backed off models. The \( n \)-gram example is used to show how combining models can help to tackle the smoothing problem. I will discuss the advantages and the disadvantages of interpolation and backing off. Especially the fact that these approaches have difficulties incorporating \emph{uncomparable} knowledge sources will be stressed. Later in this chapter we will see how the MaxEnt framework overcomes some of these problems. In the next chapter I will investigate how in this framework more and even very different information sources can be added to the same model.
Interpolation

Linear interpolation of different knowledge sources implements the idea that every knowledge source has something to say about the ultimate decision, but that some sources are more reliable than others and should therefore contribute more to the combined estimate. Every knowledge source is associated with a weight that reflects its importance. The general definition for combining $j$ different models is:

$$P_{\text{interpolation}}(w|h) = \sum_{i=1}^{j} \alpha_i P_i(w|h)$$

where $0 \leq \alpha_i \leq 1$, $\sum_{i=1}^{j} \alpha_i = 1$ and $w$ is an event (in case of language models often a word) whose likelihood is estimated and $h$ is the context (history) the models are based on.

In the case of $n$-gram language models, we use models less reliable than the (proper) $n$-gram model we seek not so much for reasons of incorporating completely different evidence, but more for tackling the sparse data problem. As was illustrated in the previous section, even a large text corpus will not guarantee good estimates of every single possible trigram. The basic idea of the interpolation approach is always to use all the available information sources and weight the contribution of the separate sources according to their reliability.

$$P_{\text{interp}}(w_i|h) = \alpha_1 P_{\text{uni}}(w_i) + \alpha_2 P_{bi}(w_i|w_{i-1}) + \alpha_3 P_{tr}(w_i|w_{i-2}, w_{i-1})$$

Some nice features of linear interpolation often make it the method of choice for combining knowledge sources. In the first place there exist algorithms to find the weights for a model that minimize the perplexity (of a set of held-out data). The software described in (Clarkson and Rosenfeld, 1997) implements the commonly used Estimation Maximization (EM) algorithm for this task. It just needs probability streams (i.e. a list of probabilities for the events in the held-out data) for the various models that are to be combined. It returns the weights for the knowledge sources that minimizes perplexity for this particular set of events (in case of a representative set of held-out data, the result will carry over to the general case). As this way of estimating the weights suggests, it does not matter what kind of knowledge sources are combined. Any model can be included, as long as it produces probabilities for the same events. The fact that any kind of information can be included makes linear interpolation very general. Finally, when EM is used to estimate the optimal weights, it is guaranteed that the resulting model will not perform worse than any of the components. The most radical case will be the combined model where one component will receive weight 1 and all other components weight 0.

Backing off

With interpolation all the available information sources are always used. The good thing is then that in case none of the information sources is completely
accurate, the combined estimate will be closer to the truth. The bad thing, however, is the fact that when one of the sources gives a much more reliable estimate than the others, the combined estimate will be further from the truth than necessary. In the backing off approach this is prevented by always taking the source that supplies the most reliable estimate. An backed-off n-gram model is defined as:

\[
P_{bo}(w_i|w_{i-n+1}...w_{i-1}) = \begin{cases} 
    d_{h(i)} \cdot \frac{C(w_{i-n+1}...w_i)}{C(w_{i-n+1}...w_{i-1})} & \text{if } C(w_{i-n+1}...w_i) > k \\
    \alpha_{h(i)} \cdot P_{bo}(w_i|w_{i-n+2}...w_{i-1}) & \text{otherwise}
\end{cases}
\]

where \(d_{h(i)}\) is a discounting factor that ensures that some probability mass is reserved for unseen \(n\)-grams (i.e. the cases where one backs off to a less specific model) and \(\alpha_{h(i)}\) is a normalizing factor that takes care for distributing only the left-over probability mass for the backed-off cases.

Somehow you want to account for the fact that you’ve never seen the trigram, while the bigram is very common. Just a simple backing off scheme would definitely be insufficient. It is also not enough to implement some discounting scheme to adjust the bigram probability. The fact that the bigram is very frequent and the trigram is not seen at all (especially when the added word is not rare), gives at lot more information about the frequency of this particular trigram. Many approaches have been suggested. A good overview and discussion of the differences can be found in Chen and Goodman (1996) or Chen and Goodman (1998).

**Problems with these approaches**

Even though linear interpolation and backing off models have proven to be very useful (and hard to beat) for language applications, there are some inherent problems with these approaches. There are many smoothing techniques applied to both approaches that address these problems in some extent, but they do not give a systematic answer (again, for a good overview see Chen and Goodman (1996) or Chen and Goodman (1998)).

One of the problems with either interpolation and backing off methods is the fact that it is difficult to account for interaction between information sources. Information sources are treated as being independent of each other. This is often not the case (e.g. the unigram, bigram and trigram components used for a trigram model are clearly not independent of each other) and by missing out on the interaction between the components we make suboptimal use of them. This can easily be shown with an example we have looked at before. In a particular corpus is found that the bigram probability \(P(’Clinton’|’Bill’) = 0.328\) and in the same corpus there are no examples at all of the case where the words ’Bill’ and Clinton are preceded by the word ’chairman’. There is important information in the fact that the bigram is seen so frequently and the trigram not at all. In the linear interpolation case, where the truth is sought somewhere in the middle of the contributions of the components, there is a big danger of
overestimating \( P('Clinton' | 'chairman', 'Bill') \). Especially when the weights for the components are determined globally (assuming independence of the components) there is no way for accounting for this local influence. The same danger exists for a backed off model. Backing off to the trigram model can, in this case, only be done if the discounting factor is (to a large extent) determined by local phenomena. Again, there are smoothing techniques available that are designed with this in mind.

Probabilistic models are designed from sample data. The sample data is all the information we have about the process we are modeling and therefore we want to stay as close to this data as possible (we do not know any better than that). Interpolated and backed off models will always be inconsistent with the training data. In case of linear interpolation this inconsistency with the sample data is inherent to the method, because none of the estimates given by one of the components is chosen (those estimates are consistent with the sample data). The resulting estimate is found somewhere in between them and will therefore always deviate from either of them. A backed off model will stay pretty close to the sample data when the most informative model applies (but not completely, because some of the probability mass is reserved for unseen items), but not anymore when one of the less informative components is used. It is not necessarily a bad thing that a model deviates from the sample data. We know that (in most cases) we will never have enough data to give a complete description of the process we are modeling. The partial description we come up with might be in need of corrections, but still, there must be a good motivated reason for these corrections.

Apparently these problems are not so grave that they prevent these approaches for combining models from being used in practice. Both interpolated and backed off models with appropriate smoothing techniques are widely and successfully used. They are well understood, computationally inexpensive to train and use and there are algorithms out there to find an optimal set of parameters. But the MaxEnt framework offers a systematic solution to both the problems I have described above, which opens the door to potentially better models. It combines the advantages of interpolated models (by always using all the information provided by the components) and backed off models (all the components always contribute to the final estimate; the most informative component is just one of them and will just be disregarded if it cannot supply any information in a particular case). Interaction between knowledge sources is accounted for and the model is guaranteed to be consistent with the sample data upon which the model is built.

4.1.5 Some information theory

In the next section where I introduce MaxEnt modeling I will use some measures from information theory. I introduce them here. For a thorough introduction the reader is referred to Cover and Thomas (1992), Jelinek (1998) or Shannon and Weaver (1963).
4.1. PROBABILISTIC PRELIMINARIES

Entropy

First we will need a way to express the amount of uncertainty in a model. The Shannon entropy is commonly used for this purpose and is defined as:

$$H(X) = - \sum_{x \in X} P(x) \log P(x)$$

where $X$ is a random variable with a probability distribution given by $P(X)$.

![Graph showing entropy function](image)

Figure 4.1: Entropy function for $H(p, 1-p)$. It reaches its maximum when $p = 1-p = 0.5$

Entropy measures the amount of uncertainty in a random variable and is normally measured in bits (i.e. it gives the number of bits needed to encode the information). Some important properties of entropy are:

- $H$ is a continuous function of $P$: Every little change in the uncertainty of a distribution $P$ should result in a change in $H$.

- $h(n) = H\left(\frac{1}{n}, \frac{1}{n}, ..., \frac{1}{n}\right)$, is a monotonic increasing function of $n$: If all events are equally likely, adding extra events will increase the uncertainty.

- $H(p, 1-p)$ is a convex function of $p$. It will reach its maximum at $p=0.5$ and will be zero if one of the probabilities in the distribution is 1 (see figure 4.1). In general, $H$ will reach its maximum when the distribution is uniform.

- $H \geq 0$ (and only equal to zero if $p(x)=1$ for some $x$): information (and therefore uncertainty) must be a positive quantity.

\[H_n(p_1, ..., p_n) = - \sum_{i=1}^{n} p_i \log p_i\]

, where $p_1 ... p_n$ is a probability distribution on the $c_1 ... c_n$ classes of events in a discrete model
Later we will be working with conditional probability distributions. Given two random variables \( Y \) and \( W \), a conditional probability distribution \( P(Y|W) \) and a probability distribution \( P(W) \), conditional entropy is defined as:

\[
H(Y|W) = - \sum_{w \in W, y \in Y} P(w)P(y|w)\log P(y|w)
\]

**Perplexity**

The perplexity of a probability distribution \( P(X) \) is a measure of the average number of possible values for a random variable \( X \). It is related to and can be defined in terms of entropy: \( PP(X) = 2^{H(X)} \)

**Kullback-Leibler divergence**

In order to decide whether the model we have defined is close enough to the model we are looking for (i.e. a model that fits the reference distribution), we need some measure of distance between two probability distributions. Often the *Kullback-Leibler Divergence* (or relative entropy) is used for this means. The Kullback-Leibler Divergence (KL) between distributions \( P \) and \( Q \) (over the same event space) is given by:

\[
D(P||Q) = \sum_{x \in X} P(x)\log \frac{P(x)}{Q(x)}
\]

where:

\[
D(P||Q) \geq 0
\]

and the distance will only be equal to 0 when \( P \) and \( Q \) are the same distribution. Although we use the KL divergence as a measure for distance between probability distributions, it does not give a proper distance in the mathematical sense. The reasons for this are the facts that it is not symmetric (i.e. \( D(P||Q) \neq D(Q||P) \)) and it does not satisfy the triangle inequality (i.e. for any three points \( x, y, z \): \( d(x,y) \leq d(x,z) + d(z,y) \)). But because we know that when the KL divergence between two distributions decreases the distributions get more similar and we will refer to this (informally) as a measure of distance.

**Cross entropy**

If we want to measure the entropy of a probability distribution according to a second distribution (for example the entropy of the distribution of a test set according to the distribution given by a model we have defined), *cross entropy* gives us an estimate:

\[
H(P, Q) = \sum_{x \in X} P(x)\log Q(x)
\]

Cross entropy also has its perplexity counterpart, which is defined as:

\[
PP(P, Q) = 2^{H(P, Q)}
\]
4.2 MaxEnt Models

In many classification problems evidence will come from various information sources. Data sparseness (as we saw in the n-gram example in the previous section) will often mean that alternative data sources will be brought to bear, but even if data were plentiful, it would still be the case that different kinds of knowledge can help to make the decision. In the previous section some inherent problems with traditional interpolation or backing off approaches to model combination were discussed. In the case where the information sources are similar (e.g. n-gram modeling) this task of combination is relatively simple. In the case of backing off models it is clear which source is more informative than the others (but it is still difficult to estimate the penalty for the backed off cases) and in the case of interpolated models, the Estimation Maximization Algorithm can be applied to estimate the weight of the different sources. But even then, the other disadvantages as sketched in section 4.1.4 can not be overcome.

The situation becomes even more difficult when information sources of a more divergent nature are used. In the case of the trigram example, it is clear that the most informative knowledge source (i.e. the model with the proper trigram estimates) should be trusted more than the less informative, and therefore the trigram is the first choice in the backing off approach and the source that contributes the most to the combined estimate in the interpolation approach. Rosenfeld (1994) describes a model for topic adaptation. In this model an extra information source is added to account for the fact that the word gates is more likely to follow the word bill in a text about computer software, whereas the word clinton would be more likely in a political text (There are of course cases where both topics overlap each other. For example, in texts about anti-trust cases both words might be equally likely). In this example it is difficult to decide how to weight information from the 'context' knowledge sources compared with the 'n-gram' knowledge. In the Maximum Entropy framework (MaxEnt) these problems are addressed: 'incomparable' information sources may be combined (the small data of the classification process will determine the weights of the various knowledge sources) and there exists an algorithm that guarantees that we find the optimal solution for the weights.

The last few years has seen much interest in the application of MaxEnt modeling in various scientific fields. MaxEnt models were originally introduced by E.T. Jaynes (Jaynes, 1957), but were not considered to be usable until recently because of the computational demands of the algorithms. Since the Computational Linguistics paper by DellaPietra et al. (Berger, DellaPietra, and DellaPietra, 1996) more and more people working in statistical NLP have become interested in applying MaxEnt models for their problems.

Before I explain how to tackle the problem introduced above within the MaxEnt framework, I will first give a short introduction to the technique. First I will give the general idea (a more thorough description can for example be found in Berger, DellaPietra, and DellaPietra (1996), Ratnaparkhi (1998), Borthwick
(1997) or Rosenfeld (1994)) Then I will work out an example along lines that will clarify many ideas. The example will be a simplified version of the model that I will describe later on in the next chapter.

### 4.2.1 Goal

We want to build a model of an unknown classification process. All the knowledge we have about this classification process is the sample output of the process. We decide what information in the data effects the classification and we will rely on these information sources (it is all we know about the classification process). Now we want to build a model that captures the information given by the information sources. We are looking for a single model as opposed to many other probabilistic approaches in which a multitude of models is built for the different information sources, and the task of combining them is not adequately addressed.

What we would like to have is a single model that uses all information sources, while keeping the estimates consistent with the relative frequencies found in the training data. And in case other information sources need to be added, we would still be able to create a single model that is consistent with the training data.

### 4.2.2 The MaxEnt principle

To meet this wish we will build a model according to the Maximum Entropy Principle as defined by Jaynes (Jaynes, 1957; Jaynes, 1996). Although the basic idea of the principle can be given in a few words, it is pretty hard to get the right feeling for what is going on exactly. I will first outline the intuition, then in the next section I will illustrate the MaxEnt principle with an example.

Maximizing the entropy of a statistical model sounds like a bad idea. The main objective of the modeler is normally to create a model that maximizes the certainty and therefore minimizes the entropy. In that respect there is not much new in the MaxEnt framework. When creating MaxEnt models, most of the effort is put in including a maximal amount of knowledge about the process that is modeled. The framework only acknowledges the fact that the resulting model will always be a partial description of the process. The term 'maximum entropy' only refers to what needs to be done with everything we don't know: the remaining probability mass that is not accounted for by the knowledge we have brought into the model! Here ancient wisdom that says that 'you should not decide on something you don't know' comes in. I will illustrate the intuition of this idea with a simple example.

Suppose someone is interested in the distribution of colors of cars. He asks the local car dealer which colors are available and learns that cars can only be bought in the colors: red, blue, yellow, green and silver. This knowledge is initially all we know about the distribution we are looking for and because we
know from probability theory that all probabilities should sum up to one, we can constrain our attention to only those distributions that have the following characteristic:

\[ P(\text{red}) + P(\text{blue}) + P(\text{yellow}) + P(\text{green}) + P(\text{silver}) = 1 \]

There are of course infinitely many probability distributions that obey this constraint. For example the model that assigns all the probability mass to the color red \( P(\text{red}) = 1 \) and nothing to the other colors obeys the constraint. The problem with this solution is, however, that it adds information to the model. It is of course always a good thing to add information to your model, but only if there is evidence for this knowledge. The Maximum Entropy solution requires in these cases that the most uniform distribution is chosen. The most uniform distribution here would be given by the model that assigns a probability of 0.2 to every single color. The intuition behind it is very reasonable: if you have information about the decision process, make it explicit and constrain attention to distributions with this characteristic, since everything you don’t know is equally likely.

Suppose extra information becomes available about the car-color distribution: you have noticed that there are more red cars out on the street and after taking a few samples you have found out that 3 out of 10 cars are red. The next constraint can be added:

\[ P(\text{red}) = 0.3 \]

Again, even though infinitely many probability distributions obey these two constraints, there is only one distribution that does not add any information not encoded in the constraints: the most uniform (i.e. \( P(\text{red}) = 0.3, P(\text{blue}) = P(\text{yellow}) = P(\text{green}) = P(\text{silver}) = 0.175 \))

Adding more constraints would lower the entropy of the model, but no matter how many constraints are added, as long as the constraints only supply a partial description of the decision process, the MaxEnt principle will always require that from all the models that are still possible, the one with highest entropy is chosen. If we were told that every second car sold is either blue or green, then this information would again remove uncertainty from the model. The most uniform distribution can again easily be identified: \( P(\text{red}) = 0.3; P(\text{blue}) = P(\text{green}) = 0.25; P(\text{yellow}) = P(\text{silver}) = 0.1 \). In this case it was always easy to find the most uniform distribution manually. It will get harder if more overlapping constraints are added.

The last thing that needs to be mentioned here, is the fact that one must always take care of keeping the constraints consistent. If for example the constraint \( P(\text{yellow}) = 0.25 \) is added, the set of constraints exclude a solution that will satisfy all of them.

When a dataset is available with sample output of the decision process, the basic idea can simply be implemented by following the next steps:
• Gather as many interesting events as possible about the process to be modeled. Examine the output data of this process (i.e. a training corpus) and try to find out what attributes in the history of a certain class trigger the prediction of that class (see 4.1.3). This will result in tuples consisting of a set of attributes and a class: \( \{ a_1, \ldots, a_m \}, w_i \).

• Count for every event \( \{ a_1, \ldots, a_m \}, w_i \) how often it appears in the training corpus. These facts are all that is known about the process to be modeled.

• Turn these facts into constraints by calculating the relative frequency of these events in the training data and state that the frequencies of these events are properties of the process.

• When looking for a model, restrict attention to only those models that obey these constraints (it is the only way to be certain that the model that is built conforms to the training data)

• Be sure that nothing is assumed other than the given facts about the data. We know that the information we have about the classification process gives us nothing more than a partial description. This means that we will not be able to completely determine the class and some uncertainty will remain. We now request that the remaining probability mass must be equally spread over the model. This can be achieved by finding the model (satisfying the constraints) which has a probability distribution which is as close to uniform as possible. Or to say it in a different way (Borthwick, 1997):

  - The probability distribution should be the one which has the greatest degree of uncertainty, given the constraints;
  - The model sought after is the one with the ‘flattest’ possible probability distribution given the constraints;
  - The distribution should be the one which has the lowest Kullback-Leibler distance from the uniform distribution; (see 4.1.5).
  - The desired model assumes a lack of higher-order interaction among the constraints except where specifically stated (i.e. if \( C_i \) and \( C_j \) do have interaction, there should be a constraint \( C_i; j \); see section 4.2.5).

### 4.2.3 The Maths

At the moment the constraints on the probability distribution are determined, the task of finding the MaxEnt solution consists not only of finding a model that satisfies all these constraints, but among all those models that fit that bill, we are looking for the model that comes with the most uniform probability distribution. Earlier we have seen that entropy is a measure of the 'amount of uncertainty' for probability distributions. The distribution \( P_i \) with the most uncertainty or the one that maximizes entropy is the one we are looking for:

\[
P_{\text{max}}(X) = \arg\max_i H(P_i(X))
\]
The entropy of a prior probability distribution $P(x)$ was defined as:

\begin{equation}
H(x) = - \sum_{x \in X} P(x) \cdot \log(P(x))
\end{equation}

And for conditional probabilities.

\begin{equation}
H(Y|W) = - \sum_{q \in W} P(q) \cdot \sum_{y \in Y} P(y|q) \cdot \log(P(y|q))
\end{equation}

It can be shown that there is a unique solution to equation 4.1 (Jaynes, 1957; Jaynes, 1996; Berger, DellaPietra, and DellaPietra, 1996) which has the form of an log-linear or exponential model:

\begin{equation}
P_{\text{max}}(y|h) = \prod_i \lambda_i^{f_i(h,y)} \frac{1}{Z_\lambda(h)}
\end{equation}

where: $Z_\lambda(h) = \sum_y \prod_i \lambda_i^{f_i(h,y)}$ is the normalization term.

In the next section I will examine in an example how this is applied.

### 4.2.4 How this works in an example

I will define here a simplified setup of the situation we will discuss in the next chapter. The form of the example I use here is taken from (Rosenfeld, 1996), but I have adapted it for this domain. I have collected 20 dialogues of a very

---

6or the equivalent:

$$P_{\text{max}}(y|h) = e^{\sum_i \lambda_i f_i(h,y)} \frac{1}{Z_\lambda(h)}$$

$$= \prod_i e^{\lambda_i f_i(h,y)}$$

where:

$$Z_\lambda(h) = \sum_y e^{\sum_i \lambda_i f_i(h,y)} \quad (\text{or} \quad \sum_y \prod_i e^{\lambda_i f_i(h,y)})$$

is the normalization term.
simple dialogue system. A typical dialogue looks like:

(4.5) Van waar naar waar wilt u reizen?
(From where to where do you want to travel?)
van A naar B
(from A to B)
Hoe laat wilt u van A naar B reizen?
(At what time do you want to travel from A to B?)
Om 10 uur
(At 10 o’clock)
U wilt dus om 10 uur ’s morgens uit A vertrekken?
(Do you want to depart from A at 10am?)
Ja
(Yes)
Wilt u nog een verbinding weten?
(Do you want to know about another connection?)
Graag
(Go please)

There are four types of questions: WHQ_LOC (locative wh-questions), WHQ_TEMP (temporal wh-questions), YNQ_CONF (request for confirmation) and YNQ_REG (‘normal’ ye-questions). In these 20 dialogues 22 words are used: {A, B, D, H, uur, ja, naar, om, van, nee, 10, 9, 7, graag, in, aankomen, 3, 2, terug, 6, 5, 12}. These 20 dialogues constitute all we know about the process we are trying to model, and we believe that it is a representative sample of the output produced by the process. Now we are going to build a model that helps us to find the best path in a wordgraph. The wordgraph in figure 4.2 is the user reply

![Wordgraph for the utterance Tien uur](image)

Figure 4.2: A wordgraph for the utterance *Tien uur*

to the system question: (*Hoe laat wilt u morgen vertrekken?* *(At what time do you want to leave tomorrow?*)). If we try to find the best path in the wordgraph in figure 4.2, we first have to decide between the words *tien* *(ten)* and *nee* *(no)* given the history (we have available the previous words (none or this is the first word of the utterance in this case) and the previous system question. That is,
we need an estimate of both \( P('tien'|h) \) and \( P('nee'|h) \) and compare them.

First we have to decide which items in the history of \( w_i \) are good predictors of \( w_i \). To keep this example as simple as possible we consider just two attributes: the previously predicted word (i.e. \( w_{i-1} \)) and the type of the previous system question. We say that these attributes trigger the occurrence of \( w_i \) and will therefore also refer to them as triggers. The attributes define the event templates, instantiations of these templates give the events. Figure 4.3 defines the attributes, event templates and a part of the event space for our example. All

- Attributes: \( w_{i-1}, qtype \)
- Event templates:
  - \( (w_{i-1},w_i) \)
  - \( (qtype,w_i) \)
- Events:

\[
\begin{align*}
(x_x, 'tien') \\
('om', 'tien') \\
(..., 'tien') \\
(x_x, 'nee') \\
('om', 'nee') \\
(..., 'nee') \\
(whq\_temp, 'tien') \\
(ymq\_reg, 'tien') \\
(..., ...) \\
\end{align*}
\]

Figure 4.3: Chosen attributes with resulting event templates and some events. The symbol '\( x_x \)' marks the beginning of an utterance; '...' is used to indicate that there exists more than one tuple with a different symbol at this position.

the knowledge we can get about the process comes from the relation between these attributes and the words we predict. We are interested in the relative frequencies of these combinations of attributes and 'words to predict' (i.e events) in the examples of the output of the process we have. To derive the estimates of the simple, unconditional (or marginal) probabilities of the events, we need a representative (and preferably large) training corpus to count how often these events occur. In this case the training corpus consists of actually uttered user replies with the corresponding system questions. This corpus together with the event templates defines our event space. The event space is divided into a number of equivalence classes

In figure 4.4 we have given a part of the event space (that part of the event space where the predicted word is 'tien' and the context can be everything).
CHAPTER 4. INTRODUCTION TO MAXIMUM ENTROPY MODELING

This part of the event space can again be divided into smaller classes of equivalent events. For example, every event where the predicted word is 'tien' and the previous word is 'om' (no matter what the words before \( w_{-1} \) were; no matter what the previous system question was and no matter what else happened in the history) belong to the same equivalence class.

<table>
<thead>
<tr>
<th>( w_i = \text{'tien'} )</th>
<th>( Q_s(h) = \text{WHQ_TEMP} )</th>
<th>( Q_s(h) = \text{YNQ_CONF} )</th>
<th>....</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{last_word}(h) ) is ( x \times )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{last_word}(h) ) is ( \text{'om'} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{last_word}(h) ) is ....</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.4: Part of the events space where \( w_i = \text{'tien'} \)

The event that that equivalence class constitutes has a particular relative frequency. So, every data sample that fits this event is put into this equivalence class and has therefore the same probability. In every row the \( w_{-1} \) in the history is different and thus defines a different equivalence class of events. The same thing holds for every column. Every data sample that fits in the leftmost column has the event that the predicted word is 'tien' and the type of the previous system question is \( \text{WHQ\_TEMP} \) in common. Every row and every column in this table represent an equivalence class in this event space. The first thing to do now is to calculate the probability estimates of the events in all the equivalence classes. These probabilities can easily be derived from the distribution of these events in the training data.

So what we have now is an estimate of the probability of events in these equivalence classes:

\[
P('tien'|W_{i-1}(h) = 'om') = K_{\{tien,om\}} \quad \text{and} \quad P('tien'|qtype(h) = \text{whq\_emp}) = K_{\{tien,whq\_emp\}}
\]

where \( K_{\{tien,om\}} \) def = \( \frac{C('tien,h \text{ ends in } 'om')}{C(h \text{ ends in } 'om')} \) and

\( K_{\{tien,whq\_emp\}} \) def = \( \frac{C('tien,Q_s(h)=\text{whq\_emp})}{C(Q_s(h)=\text{whq\_emp})} \) and \( Q_s(h) \) is a function that returns the type of the previous system question.

We consider the probabilities of these events as properties of the process we are trying to model and turn these properties into constraints. In our quest for the model that describes this process, we will pay attention only to models that have these properties.

The row marginals, all the events that belong to an equivalence class in a horizontal row, are assigned the same probability and also the column marginals,
the events in a vertical column. Now we are interested in the probability of an event that falls in the intersection of these classes (e.g. the event where the predicted word is 'tien', the previous word was 'om' and the previous system question was of type whQ-Temp). The probabilities of events in the two intersecting equivalence classes are normally independent, and therefore the two associated constraints will be inconsistent. How to estimate the probability of an event in this intersection?\footnote{We have to estimate the probability of events that belong to two (independent) equivalence classes (forming a new class: the intersection of the two classes it belongs to). Note further that for simplicity in this example only two classes intersect (2-dimensional space), so it might be possible to estimate the probability of an event within the intersection by just counting the occurrences in the training material. In the real world there will be many more (n-dimensional space) knowledge sources involved. Sparse data will cause problems very quickly when \( n \) increases.} Often the new estimate is calculated by interpolating the two probabilities. In a backing off model one of the two probabilities is chosen. Both solutions have the effect that in the resulting model it will (almost) always be the case that the estimate will not be the same as the one that was found in the training data. In the MaxEnt approach we avoid this inconsistency by relaxing the conditions imposed by the component sources. Here we no longer insist that \( P('tien'|w_{i-1}(h) = 'om') \) always has value \( K_{\{tien,om\}} \), but instead we require that it is equal to this value \textit{on average} in the training data. Of course this is a much weaker constraint. It makes sense to weaken the constraint in this way, because it acknowledges the fact that the probability of a certain event might be affected by other contextual facts.

Before the resulting form of the constraints can be shown, we first have to develop notation reflecting how we divide the event space into the equivalence classes. Therefore \textit{selector functions} (or indicator functions) are defined. These functions take two arguments: the history and the item whose probability we want to estimate \((w, j)\). The \( j \)th selector function looks like this:\footnote{Although real valued functions are possible, we only use binary valued functions in this work.}

\[
f_j(h_i, w_i) = \begin{cases} 1 & \text{if } w_{i-1}(h_i) = 'om' \wedge w_i = 'tien' \\ 0 & \text{otherwise} \end{cases}
\]

With these functions we have a means to note whether a particular history/item combination is in a certain equivalence class. Now we are able to formulate the constraints defined above:

\[
\sum_{h, w} (P(h, w) \cdot f_j(h, w)) = K_{h,w}
\]

where \( K_{h,w} \) is the count of observed \((h, w)\).

That is, we take the sum of all the (joint) probabilities of the events in an equivalence class as an estimate of future occurrences of events in the same equivalence class. But because we are trying to model conditional probabilities
rather than joint probabilities we can reformulate the constraint as:

$$\sum_{h,w} (P(h) \cdot P(w|h) \cdot f_i(h, w)) = K_{h,w}$$

Where $P(h)$ is the model probability of the context and $P(w|h)$ is the conditional probability assigned by our model. Because the space of possible contexts ($H$) will often be too big in practice, $P(h)$ cannot explicitly be normalized over the space of contexts. Therefore we will instead use:

$$\sum_{h,w} (\tilde{P}(h) \cdot P(w|h) \cdot f_i(h, w)) = K_{h,w}$$

where $\tilde{P}(h)$ is the empirical probability of $h$ (i.e. the relative frequency of $h$ observed in the training data) and $P(w|h)$ is the conditional probability assigned by our model.

Now we can produce all the constraints and calculate their expected values from the training data. The next step is to find the model satisfying these constraints that has highest entropy.

I have calculated the MaxEnt model for the example introduced earlier in this section. A sample with the relevant features of the model is given in figure 4.5. Using the features and their associated parameters we can now evaluate $P(w|h)$. Again, although the model does not tell us anything about complete paths through the word graph (it only helps us to decide between possible next words) it gives a good idea how the model can benefit from adding information sources other than $n$-gram. We will need to apply a search algorithm to find the best scoring path (as discussed in section 2).

I repeat here the formula for calculating $P(w|h)$ as given in 4.2.3:

$$P(w|h) = \prod_i \lambda_i^{f_i(h,w)} \frac{1}{Z_\lambda(h)}$$

where: $Z_\lambda(h) = \sum_w \prod_i \lambda_i^{f_i(h,w)}$ is the normalization term.

In order to apply this formula to calculate $P('tien'|h)$ we need to collect all the features that return a non zero value (i.e. the features that return zero will not contribute to the product ). In this case, where the previous system question was of type WHQ TEMP and the word to predict is the first word of the sentence (so $w_{-1} = x.x$), only the features $f_{\{'tien'\}}, f_{\{x.x\},tien'}, f_{\{(whq_{temp}),tien'\}}$ and $f_{\{x.x,whq_{temp},tien'\}}$ return 1:

$$P('tien'|w_{-1} = 'x.x', whq_{temp}) = 0.6362 \cdot 2.1572 \cdot 0.5938 \cdot 7.7187 \cdot \frac{1}{Z_\lambda(h)}$$
### 4.2. MAXENT MODELS

<table>
<thead>
<tr>
<th>Event</th>
<th>Feature (f_i)</th>
<th>Parameter (\lambda_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS, 10, 7</td>
<td>class(_,10)</td>
<td>0.6362</td>
</tr>
<tr>
<td>CLASS, nee, 15</td>
<td>class(_,nee)</td>
<td>0.9459</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLASS, om, 21</td>
<td>class(_,om)</td>
<td>1.3366</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QType : whq_temp, 10, 6</td>
<td>qtype((whq_temp,10)</td>
<td>2.1572</td>
</tr>
<tr>
<td>QType : whq_temp, nee, 1</td>
<td>qtype(whq_temp,nee)</td>
<td>0.3906</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QType : ynq_reg, 10, 0</td>
<td>qtype(ynq_reg,10)</td>
<td>2.7979</td>
</tr>
<tr>
<td>QType : ynq_reg, nee, 8</td>
<td>qtype(ynq_reg,nee)</td>
<td>2.7979</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word_1 : x_x, 10, 1</td>
<td>word_1(x_x,10)</td>
<td>0.5938</td>
</tr>
<tr>
<td>Word_1 : x_x, nee, 15</td>
<td>word_1(x_x,nee)</td>
<td>3.8934</td>
</tr>
<tr>
<td>Word_1 : x_x, om, 17</td>
<td>word_1(x_x,om)</td>
<td>4.2696</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word_1/QType : x_x/whq_temp, 10, 1</td>
<td>word_1_qtype(x_x/whq_temp,10)</td>
<td>7.7187</td>
</tr>
<tr>
<td>Word_1/QType : x_x/whq_temp, nee, 8</td>
<td>word_1_qtype(x_x/whq_temp,nee)</td>
<td>4.4683</td>
</tr>
</tbody>
</table>

Figure 4.5: Some features and their parameters (weights) from the example model

and

\[
P('nee'|w_{-1} = 'x_x', \text{whq_temp}) = 0.9459 \cdot 0.3906 \cdot 3.8934 \cdot 4.4683 \cdot \frac{1}{Z_\lambda(h)}
\]

where the normalization term \(\frac{1}{Z_\lambda(h)}\) will have the same value in both cases (it is the sum over all possible classes that could be predicted with this history).

\[
P('tien'|w_{-1} = 'x_x', \text{whq_temp}) = 6.2902576 \frac{1}{Z_\lambda(h)}
\]
\[
P('nee'|w_{-1} = 'x_x', \text{whq_temp}) = 6.4275996 \frac{1}{Z_\lambda(h)}
\]

Now that we know what information we want to use, how to extract the relevant information from the training material and how to represent the information, we need to think about whether we want to use all the information that we have. Once we decide what information we want to use, we need an algorithm to build the actual model.

#### 4.2.5 Constraint interaction

So the first thing to worry about is what information to use. Although it will not be too difficult to identify knowledge sources that might contribute to the
model, this issue is more complicated. When I gave a rough outline of the Maximum Entropy principle in the introduction of this section, I mentioned that the desired model should lack higher-order interaction among the constraints. The idea is that when two constraints are not independent, there is information about the process we are modeling in this dependency. Adding a new complex constraint that consists of both atomic constraints gives information about this interaction and should result in a decreasing divergence between the model and the reference distribution (i.e. the distribution we are modeling).

So when there are two constraints $C_i$ and $C_j$ that are not independent, there should also be a constraint $C_{ij}$. For example when defining a trigram model the next two features would be defined:

$$f_1(h_i, w_i) = \begin{cases} 
1 & \text{if } w_{i-1}(h_i) = w' \land w_i = w'_i \\
0 & \text{otherwise}
\end{cases}$$

which defines the relation between the current word and the previous word, and:

$$f_2(h_i, w_i) = \begin{cases} 
1 & \text{if } w_{i-2}(h_i) = w' \land w_i = w'_i \\
0 & \text{otherwise}
\end{cases}$$

which defines the relation between the current word and the word two to the left.

But because of the fact that words do not occur in random order, which means that there is an interaction between these two constraints, we need to express this explicitly and need therefore the combined constraint:

$$f_3(h_i, w_i) = \begin{cases} 
1 & \text{if } w_{i-1}(h_i) = w'_i \land w_{i-2}(h_i) = w'_2 \land w_i = w'_i \\
0 & \text{otherwise}
\end{cases}$$

which defines the relation between the current word and the previous two words.

This will introduce new difficulties however. In the example I just gave, it is pretty clear that there is interaction between the two constraint types, but it will not always be obvious. Especially when two constraints model information from completely different information sources, it will not always be clear when there is a significant interaction.

The situation is even more difficult when two constraint types (e.g. one based on the attribute $w_{i-1}$ and the other on $qtype$) interact in some instances, but not in others (e.g. in case $w_{i-1}(h_i)$='welk' (which) and $qtype(h_i)$=whqLEDtemp it is very likely that there is interaction, while in case $w_{i-1}(h_i)$='de' (the) and $qtype(h_i)$=whqLEDtemp it is unlikely).

The problem is not so much that these 'useless' combined constraints can hurt the model. If there is no interaction between two constraints, then the weights for the separate (atomic) constraints will be considerably higher than the weight
4.2. MAXENT MODELS

for the combined (complex) constraint. This is because of the fact that the combination does not say much about the process we are modeling and will therefore not be important for the model (which is reflected in a low weight associated with the feature). It is however not a good idea to include too many 'noisy' features in the model: the model will become large and expensive to train and use.

Adding these complex features might result in an enormous number of extra features. Many of the extra features might not be very useful for the model. Therefore it will be good to have a mechanism that helps to decide whether a feature should be included in the model or not. Feature selection will be discussed in the next section.

4.2.6 Feature selection

The next thing to worry about is the question of whether all the data from the training material is reliable. What we need is some way to decide which features (generated from the templates we have defined) we want to use. For example we have features of the 'bigram type':

\[
f_j(h_i, w_i) = \begin{cases} 
1 & \text{if } w_{i-1}(h_i) = w'_{i-1} \land w_i = w' \\
0 & \text{otherwise} 
\end{cases}
\]

If there are 2500 words in the lexicon of the speech recognizer, this feature template will, in the worst case, result in 6.25 million features. Most of them will however contribute little or nothing to the model. It is claimed that they won't harm the model, but the drawback of too many features will be a very large model (which will take ages to compute and will be computationally expensive to use).10

Berger, DellaPietra, and DellaPietra (1996) give a feature selection algorithm to select only those features that contribute the most to the model. It is a computationally very expensive algorithm, however. It starts with the smallest possible model with a uniform distribution. From the set of \( n \) potential features, one feature is chosen, added to the initial model and a new model that includes this feature is calculated. This process is repeated \( n \) times (for every potential feature). The feature that gives rise to the best performing model is added to the initial model. The whole process is repeated with the \( n - 1 \) remaining features until the best scoring feature adds less than some threshold value to the model. This means however that even if there exists a smart way of deciding which feature is the best candidate to add to the model in every step, a completely new MaxEnt model needs to be computed as many times as new features are added.

---

9Both issues are addressed in (Mikheev, forthcoming) in which a clustering technique for features is investigated.

10All the features that do not have a single event in the training corpus are obviously disregarded anyway, because no estimate of their probability can be derived from the training material. This will already reduce the number of features enormously (fewer than 100,000 remained in my experiments).
A much simpler feature selection is discussed in chapter 8 of Ratnaparkhi (1998). Here he proposes just to throw away all the features that are observed less than threshold times in the training material. Although this doesn't say anything about the real contribution of a certain feature to a model, it does have the desired effect of losing the unreliable features. Although Ratnaparkhi does not give a theoretical comparison of these methods, he reports experiments that suggest that noise is genuinely reduced. The advantage of the latter method of feature selection is obvious: it takes almost no time to reduce the feature set to more reasonable sizes. The drawback is however that it doesn't give any insight in which features are really important to describe the process to be modeled.

### 4.2.7 Generalized Iterative Scaling

And finally we need to have an idea of how to compute the model (i.e. how to compute the values for $\lambda_i$). The complete recipe a method called Generalized Iterative Scaling (GIS) is given in the Computational Linguistics article by Berger and the DellaPietra's (Berger, DellaPietra, and DellaPietra, 1996). Although the algorithm is straightforward, it is not trivial to implement it in a efficient way. There are however also some packages available for free use. For this work I used the package Maccent (Dehaspe, 1997).

In this section I will not explain in detail how GIS works, but just outline the general idea. The goal we have to work towards is given: we have calculated the empirical relative frequencies of the events before. The probabilities derived from these frequencies are the target expectations of the features. For all these features we want our model to predict values as close to these target expectations as possible. Therefore we will modify the weight (how much a particular feature contributes to the model) of the features. When all the features are assigned a weight, we can calculate the values for all the features and compare them to the target expectations. The values and the target expectations are compared and the weights of the features are adjusted so that the distance between the values the model predicts and the target expectations decreases. The algorithm is described in full detail in Berger, DellaPietra, and DellaPietra (1996) and in section 2.6 of Ratnaparkhi (1998).

### 4.2.8 Convergence of model during training

Ratnaparkhi (1998) describes (In chapter 9) a convergence problem that may occur while training the model. I have encountered the problem in my experiments and will therefore discuss briefly the cause and possible ways to deal with it here.

The problem occurs when we have to deal with unique contexts. In case there exists some history/item pair $(h_i, y_k)$ where history $h_i$ is unique, the desired probability of $P(y_k|h_i)$ will be equal to 1. The formula in 4.4 will consequently
try to get the value of

\[
p(y_i|h_i) = \prod_{j=1}^{k} f_j(h_j, y_j) \frac{1}{Z_{\lambda}(h_j)}
\]

as close to the expected value (i.e. 1) as possible. There is only one feature active, however. This feature \(i\) (it will be the numerator and of course one of the factors of the denominator) will therefore be the only feature whose weight \((\lambda_i)\) will contribute to the probability. All the other (non-active) features will however contribute 1 (i.e. \(\lambda_j^0\)) to the denominator. Ergo, the denominator will always be \(k - 1\) (number of non-active features with history \(h_i\)) bigger than the numerator and therefore the fraction will never be 1. It can come close to 1 when \(\lambda_i\) goes to \(+\infty\). In every iteration the value for \(\lambda_i\) will be increased.

This would not be so problematic if features with a non-ambiguous context only appeared in isolation. It becomes a serious problem though, when an non-ambiguous feature \(f_n\) is combined with a feature with ambiguous context \(f_a\). Because even when the non-ambiguous feature has only a few events in the training material and the ambiguous features many, the combined feature \(f_{na}\) will be completely dominated by \(f_n\), even while there is much more evidence for \(f_a\).\(^{11}\) Typically, very infrequent features are more likely to be coincidences and have unique contexts.

Using a threshold of 2 (as above) eliminates most of these features with non-ambiguous context, but not all of them. The iterative scaling algorithm used in the software I have used for our experiments stops at the moment when the distance between the model and the empirical distribution is lower than a certain value. Existence of these problematic features will have the effect that the model will not converge further to the empirical distribution. In our experiments I found that it was useful to use either a feature count cut-off to get rid of most of these features and use no more than 50 iterations (while the software suggests to continue).

This is clearly not a perfect solution, however. Even though it is often suggested otherwise, there is also a need to smooth MaxEnt models. People have investigated the issue, for example (Chen and Rosenfeld, 1999; Lau, 1994).

### 4.2.9 Pros and cons

In this chapter I have introduced a particular framework for statistical modeling. I have only compared it to two traditional methods of creating probabilistic models using several information sources and pointed out the advantages of

\(^{11}\)Remember, if we only have 3 features \(f_n, f_a\) and \(f_{na}\), the value for \(p(y|h)\) will be calculated by:

\[
p(y|h) = \frac{f_n^{\lambda_n} \cdot f_a^{\lambda_a} \cdot f_{na}^{\lambda_{na}}}{Z_{\lambda}(h)}
\]

will be more or less equal to \(\lambda_n\) when \(\lambda_n\) is much bigger then \(\lambda_a \cdot \lambda_{na}\).
the MaxEnt framework with respect to these two alternatives. There are many other approaches ((Manning and Schütze, 1999; Rosenfeld, 2000)). Even though I have good reason to believe that MaxEnt is particularly suitable to approach the kind of problem I am looking at here, I will not make the claim that it is the only suitable candidate. There are two main reasons that made me explore MaxEnt modeling. First, the fact that I want to experiment with models that integrate information from (potentially very) different knowledge sources and the MaxEnt framework promises a smooth integration of various information sources. The reported success of the framework when performance of models was compared with alternatives is an important second reason for giving it a closer look. There are good reasons for the good performance. The first reason, I have mentioned it above, is the smooth integration of different knowledge sources. Others are:

- The fact that the individual features (instead of the feature templates) are assigned weights accounts for optimal fit of the training data.

- There is a kind of 'built in' smoothing in the models. Not all the probability mass is accounted for by the constraints. Some of it is reserved for unseen cases, so even if a certain event is never seen before, it will always get a non-zero probability. I have argued before that the fact that features may overlap, allows for built in interpolation and backing off.

- Freedom in the choice of features: features are allowed to overlap (there is no assumption of independence of the features)

Finally, a nice property is that the generality of the framework allows that off-the-shelf software can be used to generate the model. The modeler can concentrate on choosing the knowledge sources (and make a judgment on which knowledge sources interact in such a way that the combined evidence is a useful addition to the model).

However, this does not mean that there are only virtues. One of the main problems with these models is that it takes quite a lot of computational power to create them. When it is clear which features are useful to include (hence, no fancy feature selection algorithms are needed), this will not be too problematic. The models I have created for the next chapter did not need more than a few hours to train. Since this is done off line, this will be acceptable for many applications.

The use of the model is potentially a bigger problem. When the created models are not too big (i.e. discrete models with a limited number of different events for which a probability must be computed), we can compute the complete model off-line and store it for future use (as we normally do with n-gram models). When the model gets bigger, we will run into efficiency problems. If we cannot store the model in terms of events with associated probabilities, but only in terms of features with their associated weights, we will have to apply formula 4.4 each time we need the probability of an event. Especially the normalisation function \(Z(h)\) is expensive. In the next chapter we will see that computing probabilities on-line can be problematic and needs extra attention.