5 Supervised Classification

5.1 Introduction

Supervised learning is a very popular approach in any text classification task. Many different algorithms are available that learn classification patterns from a set of labeled or classified examples. Given this sample of examples, the task is to model the classification process and to use the model to predict the classes of new, previously unseen examples. In information retrieval the supervised techniques are very popular for the classification of documents into subject categories (e.g., the classification of news into financial, political, cultural, sports, …) using the words of a document as main features. In information extraction usually smaller content units are classified with a variety of classification schemes ranging from rather generic categories such as generic semantic roles of sentence constituents to very specific classes such as the number of heavily burned people in a firework accident.

As in text categorization, the *amount of training examples is often limited*, or training examples are expensive to build. When the example set is small, it very often represents incomplete knowledge about the classification model sought. This danger is especially present in natural language data where a large variety of patterns express the same content.

Parallel to text categorization, *the amount of different features* is large and the feature set could include some noisy features. There are many different words, syntactic, semantic and discourse patterns that make up the context of an information element. But, compared to text categorization lesser features in the context are indicative of the class sought. Often, the features behave dependently and the dependency is not always restricted to co-occurrence of certain feature values, it sometimes also demands that feature values occur in a certain order in the text. When different classes are to be assigned to text constituents, the class assignment might also be dependent on classes previously assigned.

Chap. 2 gave an extensive historical overview of machine learning approaches that have been used to extract information from text and relevant references were cited. In this chapter we dig deeper into the current and most successful algorithms for information extraction that use a supervised learning approach. The *chosen classifiers* allow dealing with incomplete data and with a large set of features that on occasion might be noisy. They also have the potential to be used for weakly supervised learning (described in the next chapter), and incorporate dependencies in their models.

As seen in Chap. 4 a feature vector $\boldsymbol{x}$ is described by a number of features (see Eq. (4.1)) that may refer to the information element to be classified, to the close context of the element, to the more global context of the document in which the element occurs, and perhaps to the context of the document collection. The goal is to assign a label y to a new example. Among the statistical learning techniques a distinction is often made between generative and discriminative classifiers (Vapnik, 1988; Ng and Jordan, 2002). Given inputs $\boldsymbol{x}$ and their labels y, a *generative classifier* learns a model of the joint probability, $p(\boldsymbol{x},y)$ and makes its predictions by using Bayes' rule to calculate $p(y|\boldsymbol{x})$[1] and then selects the most likely label y. An example of a generative classifier that we will discuss in this chapter is a *hidden Markov model.* A *discriminative classifier* models the posterior probability $p(y|\boldsymbol{x})$ directly and selects the most likely label y, or learns a direct map from inputs $\boldsymbol{x}$ to the class labels. An example is the *maximum entropy model*, which is very often used in information extraction. Here, the joint probability $p(\boldsymbol{x},y)$ is modeled directly from the inputs $\boldsymbol{x}$. Another example is a *Support Vector Machine,* which is a quite popular learning technique for information extraction. Some of the classifiers adhere to the *maximum entropy principle*. This principle states that, when we make inferences based on incomplete information, we should draw them from that probability distribution that has the maximum entropy permitted by the information we have (Jaynes, 1982). Two of the discussed classifiers adhere to this principle. They are the *maximum entropy model* and *conditional random fields*.

In information extraction sometimes a relation exists between the various classes. In such cases it is valuable not to classify a feature vector separately from other feature vectors in order to obtain a more accurate

[1] With a slight abuse in notation in the discussion of the probabilistic classifiers, we will also use $p(y|\boldsymbol{x})$ to denote the entire conditional probability distribution provided by the model, with the interpretation that y and $\boldsymbol{x}$ are placeholders rather than specific instantiations. A model $p(y|\boldsymbol{x})$ is an element of all conditional probability distributions. In the case that feature vectors take only discrete values, we will denote the probabilities by the capitalized letter P.

classification of the individual extracted information. This is referred to as *context-dependent classification* as opposed to context-free classification. So, the class to which a feature vector is assigned depends on 1) the feature vector itself; 2) the values of other feature vectors; and 3) the existing relation among the various classes. In information extraction, dependencies exist at the level of descriptive features and at the level of classes, the latter also referring to classes that can be grouped in higher-level concepts (e.g., scripts such as a **bank robbery** script). We study two *context-dependent classifiers*, namely a *hidden Markov model* and one based on *conditional random fields*.

Learning techniques that present the learned patterns in representations that are well conceivable and interpretable by humans are still popular in information extraction. *Rule and tree learning* is the oldest of such approaches. When the rules learned are represented in logical propositions or first-order predicate logic, this form of learning is often called *inductive logic programming* (ILP). The term *relational learning* refers to learning in any format that represents relations, including, but not limited to logic programs, graph representations, probabilistic networks, etc. In the last two sections of this chapter we study respectively rule and tree learning, and relational learning.

The selection of classifiers in this chapter by no means excludes other supervised learning algorithms for which we refer to Mitchell (1997) and Theodoridis and Koutroumbas (2003) for a comprehensive overview of the supervised classification techniques.

When a binary classifier is learned in an information extraction task, we are usually confronted with an *unbalanced example set*, i.e., there are usually too many negative examples as compared to the positive examples. Here techniques of active learning discussed in the next chapter might be of help to select a subset of negative examples.

When using *binary classification* methods such as a Support Vector Machine, we are usually confronted with the *multi-class problem*. The larger the number of classes the more classifiers need to be trained and applied. We can handle the multi-class problem by using a one-vs-rest (one class versus all other classes) method or a pair wise method (one class versus another class). Both methods construct multi-class SVMs by combining several binary SVMs. When classes are not mutually exclusive, the one-vs-rest approach is advisable.

In information extraction we are usually confronted with a *complex problem*. For instance, on one hand there is the detection of the boundaries of the information unit in the text. On the other hand there is the classification of the information unit. One can tackle these problems separately, or learn and test the extractor in one run. Sometimes the semantic classes to be assigned are hierarchically structured. This is often the case for entities to be recognized in biomedical texts. The hierarchical structure can be exploited both in an efficient training and testing of the classifier by assuming that one class is subsumed by the other. As an alternative, in relational learning one can learn class assignment and relations between classes. A similar situation occurs where components of a class and their chronological order signal a superclass. An important motivation for separating the classification tasks is when they use a different feature set. For instance, with the boundary recognition task, the orthographic features are important, while in the classification task the context words are valuable. The issue of separating the learning tasks or combining them in one classifier will be tackled in Chap. 10.

5.2 Support Vector Machines

Early machine learning algorithms aimed at learning representations of simple symbolic functions that could be understood and verified by experts. Hence, the goal of learning in this paradigm was to output a hypothesis that performed the correct classification of the training data, and the learning algorithms were designed to find such an accurate fit to the data. The hypothesis is *complete* when it covers all positive examples, and it is *consistent* when it does not cover any negative ones. It is possible that a hypothesis does not converge to a (nearly) complete and (nearly) consistent one, indicating that there is no rule that discriminates between the positive and the negative examples. This can occur either for noisy data, or in case where the rule language is not sufficiently complex to represent the dichotomy between positive and negative examples.

This situation has fed the interest in learning a *mathematical function* that discriminates the classes in the training data. Among these, linear functions are the best understood and the easiest to apply. Traditional statistics and neural network technology have developed many methods for discriminating between two classes of instances using linear functions. They can be called linear learning machines as they use hypotheses that form linear combinations of the input variables.

In general, complex real-world applications require more expressive hypothesis spaces than the ones formed by linear functions (Minsky and Papert, 1969). Frequently, the target concept cannot be expressed as a simple linear combination of the given features, but requires that more abstract features of the data are exploited. *Kernel representations* offer an alternative solution by mapping the data into a high dimensional feature space where a linear separation of the data becomes easier. In natural language classification, it is often not easy to find a complete and consistent hypothesis that fits the training data. And in some cases linear functions are insufficient in order to discriminate the examples of two classes. This is because natural language is full of exceptions and ambiguities. We may not capture sufficient training data, or the training data might be noisy in order to cope with these phenomena, or the function we are trying to learn does not have a simple logical representation.

In this section we will lay out the principles of a Support Vector Machine for data that are linearly or nearly linearly separable. We will also introduce kernel methods because we think they are a suitable technology for certain information extraction tasks.

The technique of Support Vector Machines (Cristianini and Shawe-Taylor, 2000) is a method that finds a function that discriminates between two classes. In information extraction tasks the two classes are often the positive and negative examples of a class. In the theory discussed below we will use the terms positive and negative examples. This does not exclude that any two different semantic classes can be discriminated.

We will first discuss the technique for example data that are linearly separable and then generalize the idea to data that are not necessarily linearly separable and to examples that cannot be represented by linear decision surfaces, which leads us to the use of kernel functions.

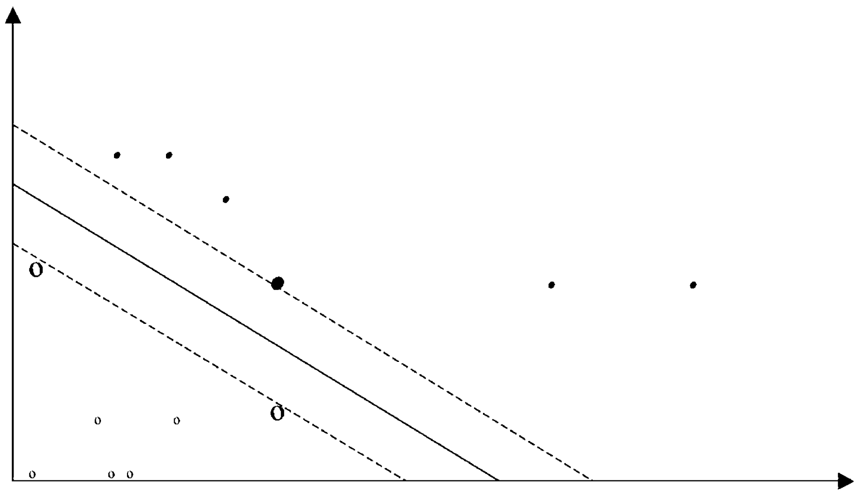

Fig. 5.1. A maximal margin hyperplane with its support vectors highlighted (after Christianini and Shawe-Taylor, 2000).

In a classical linear discriminant analysis, we find a linear combination of the features (variables) that forms the hyperplane that discriminates between the two classes (e.g., line in a two-dimensional feature space, plane in a three-dimensional feature space). Generally, many different hyperplanes exist that separate the examples of the training set in positive and negative examples among which the best one should be chosen. For instance, one can choose the hyperplane that realizes the maximum margin between the positive and negative examples. The hope is that this leads to a better generalization performance on unseen examples. Or in other words, the hyperplane with the margin *d* that has the maximum Euclidean distance to the closest training examples (support vectors) is chosen. More formally, we compute this hyperplane as follows:

Given the set *S* of *n* training examples:

$$S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$$

where $\boldsymbol{x}_i \in \Re^p$ (*p*-dimensional space) and $y_i \in \{-1,+1\}$ indicating that $\boldsymbol{x}_i$ is respectively a negative or a positive example.

When we train with data that are linearly separable, it is assumed that some hyperplane exists which separates the positive from the negative examples. The points which lie on this hyperplane satisfy:

$$\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b = 0 \tag{5.1}$$

where $\boldsymbol{w}$ defines the direction perpendicular to the hyperplane (normal to the hyperplane). Varying the value of *b* moves the hyperplane parallel to itself. The quantities $\boldsymbol{w}$ and *b* are generally referred to as respectively *weight vector* and *bias*. The perpendicular distance from the hyperplane to the origin is measured by:

$$\frac{|b|}{\|\boldsymbol{w}\|} \tag{5.2}$$

where $\|\boldsymbol{w}\|$ is the Euclidean norm of $\boldsymbol{w}$.

Let d_+ (d_-) be the shortest distance from the separating hyperplane to the closest positive (negative) example. d_+ and d_- thus define the margin to the hyperplane. The task is now to find the hyperplane with the largest margin.

Given the training data that are linearly separable and that satisfy the following constraints:

$$\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b \geq +1 \qquad \text{for } y_i = +1 \tag{5.3}$$

$$\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b \leq -1 \qquad \text{for } y_i = -1 \tag{5.4}$$

which can be combined in 1 set of inequalities:

$$y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) - 1 \geq 0 \qquad \text{for } i = 1,\ldots, n \tag{5.5}$$

The hyperplane that defines one margin is defined by:

$$H_1 : \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b = 1 \tag{5.6}$$

with perpendicular distance from the origin:

$$\frac{|1-b|}{\|\boldsymbol{w}\|} \tag{5.7}$$

The hyperplane that defines the other margin is defined by:

$$H_2 : \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b = -1 \tag{5.8}$$

with perpendicular distance from the origin:

$$\frac{|-1-b|}{\|\boldsymbol{w}\|} \tag{5.9}$$

Hence $d_+ = d_- = \frac{1}{\|\boldsymbol{w}\|}$ and the margin $= \frac{2}{\|\boldsymbol{w}\|}$.

In order to maximize the margin the following objective function is computed:

$$\begin{aligned} &\text{Minimize}_{w,b}\ \langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle \\ &\text{Subject to } y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) - 1 \geq 0, \quad i = 1,\ldots,n \end{aligned} \tag{5.10}$$

Linear learning machines can be expressed in a *dual representation*, which turns out to be easier to solve than the primal problem since handling inequality constraints directly is difficult. The dual problem is obtained by introducing *Lagrange multipliers* λ_i, also called dual variables. We can transform the primal representation into a dual one by setting to zero the derivatives of the Lagrangian with respect to the primal variables, and substituting the relations that are obtained in this way back into the Lagrangian, hence removing the dependence on the primal variables. The resulting function contains only dual variables and is maximized under simpler constraints:

$$\begin{aligned} &\text{Maximize } W(\lambda) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{n} \lambda_i \lambda_j y_i y_j \langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \rangle \\ &\text{Subject to: } \lambda_i \geq 0 \\ &\qquad \sum_{i=1}^{n} \lambda_i y_i = 0, \; i = 1,\dots,n \end{aligned} \tag{5.11}$$

It can be noticed that training examples only need to be inputted as *inner products* (see Eq. (5.11)), meaning that *the hypothesis can be expressed as a linear combination of the training points*. By solving a quadratic optimization problem, the decision function $h(\boldsymbol{x})$ for a test instance $\boldsymbol{x}$ is derived as follows:

$$h(\boldsymbol{x}) = sign(f(\boldsymbol{x})) \tag{5.12}$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \lambda_i y_i \langle \boldsymbol{x}_i \cdot \boldsymbol{x} \rangle + b \tag{5.13}$$

The function in Eq. (5.13) only depends on the support vectors for which $\lambda_i > 0$. Only the training examples that are support vectors influence the decision function. Also, the decision rule can be evaluated by using just *inner products* between the test point and the training points.

We can also train a soft margin Support Vector Machine which is able to deal with some noise, i.e., classifying examples that are linearly separable while taking into account some errors. In this case, the amount of training error is measured using slack variables ξ_i, the sum of which must not exceed some upper bound.

The hyperplanes that define the margins are now defined as:

$$H_1 : \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b = 1 - \xi_i \tag{5.14}$$

$$H_2 : \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b = -1 + \xi_i \tag{5.15}$$

Hence, we assume the following objective function to maximize the margin:

$$\begin{aligned} &\text{Minimize}_{\xi, \boldsymbol{w}, b} \ \langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle + C \sum_{i=1}^{n} \xi_i^2 \\ &\text{Subject to } y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) - 1 + \xi_i \geq 0 \ , \quad i = 1,\ldots,n \end{aligned} \tag{5.16}$$

where	$\sum_{i=1}^{n} \xi_i^2$	=	penalty for misclassification
	C	=	weighting factor.

The decision function is computed as in the case of data objects that are linearly separable (cf. Eq. (5.13)).

When classifying natural language data, it is not always possible to linearly separate the data. In this case we can map them into a feature space where they are linearly separable (see Fig. 5.2). However, working in a high dimensional feature space gives computational problems, as one has to work with very large vectors. In addition, there is a generalization theory problem (the so-called *curse of dimensionality*), i.e., when using too many features, we need a corresponding number of samples to insure a correct mapping between the features and the classes. However, in the dual representation the data appear only inside inner products (both in the training algorithm shown by Eq. (5.11) and in the decision function of Eq. (5.13)). In both cases a kernel function (Eq. (5.19)) can be used in the computations.

A Support Vector Machine is a kernel based method. It chooses a kernel function that projects the data typically into a high dimensional feature space where a linear separation of the data is easier.

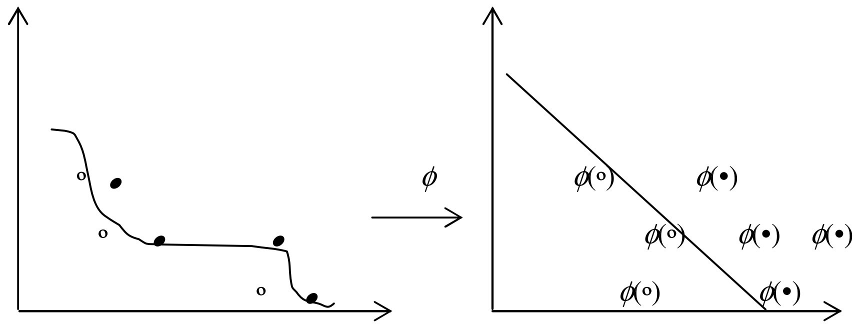

Fig. 5.2. A mapping of the features can make the classification task more easy (after Christianini and Shawe-Taylor 2000).

Formally, a kernel function K is a mapping K: S x $S \rightarrow [0, \infty]$ from the instance space of training examples S to a similarity score:

$$K(\boldsymbol{x}_i,\boldsymbol{x}_j)=\sum_k \phi_k(\boldsymbol{x}_i)\phi_k(\boldsymbol{x}_j)=\langle\phi(\boldsymbol{x}_i)\cdot\phi(\boldsymbol{x}_j)\rangle \tag{5.17}$$

In other words a kernel function is an inner product in some feature space (this feature space can be potentially very complex). The kernel function must be symmetric $[K(\boldsymbol{x}_i,\boldsymbol{x}_j) = K(\boldsymbol{x}_j,\boldsymbol{x}_i)]$ and positive semi-definite. By semi-definite we require that if $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n \in S$, then the n x n matrix G defined by $G_{ij} = K(\boldsymbol{x}_i,\boldsymbol{x}_j)$ is positive semi-definite[2]. The matrix G is called the *Gram matrix* or the *kernel matrix*. Given G, the support vector classifier finds a hyperplane with maximum margins that separates the instances of different classes. In the decision function $f(\mathbf{x})$ we can just replace the dot products with kernels $K(\boldsymbol{x}_i,\boldsymbol{x}_j)$.

$$h(\boldsymbol{x})=sign(f(\boldsymbol{x})) \tag{5.18}$$

$$f(\boldsymbol{x})=\sum_{i=1}^{n}\lambda_i y_i\langle\phi(\boldsymbol{x}_i)\cdot\phi(\boldsymbol{x})\rangle+b \tag{5.19}$$

Or

$$f(\boldsymbol{x})=\sum_{i=1}^{n}\lambda_i y_i K(\boldsymbol{x}_i,\boldsymbol{x})+b$$

[2] A matrix $A \in \Re^{pxp}$ is a positive semi-definite matrix if $\forall x\in\Re^p \Rightarrow x^T A x \geq 0$. A positive semi-definite matrix has non-negative eigenvalues.

To classify an unseen instance $\boldsymbol{x}$, the classifier first projects $\boldsymbol{x}$ into the feature space defined by the kernel function. Classification then consists of determining on which side of the separating hyperplane $\boldsymbol{x}$ lies. If we have a way of efficiently computing the inner product $\langle \phi(x_i) \cdot \phi(\boldsymbol{x}) \rangle$ in the feature space as a function of the original input points, the decision rule of Eq. (5.19) can be evaluated by at most n iterations of this process.

An example of a simple kernel function is the *bag-of-words kernel* used in text categorization where a document is represented by a binary vector, and each element corresponds to the presence or absence of a particular word in the document. Here, $\phi_k(\boldsymbol{x}_i) = 1$ if word w occurs in document $\boldsymbol{x}_i$ and word order is not considered. Thus, the kernel function $K(\boldsymbol{x}_i,\boldsymbol{x}_j)$ is a simple function that returns the number of words in common between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

Kernel methods are effective at reducing the feature engineering burden for structured objects. In natural language processing tasks, the objects being modeled are often strings, trees or other discrete structures. By calculating the similarity between two such objects, kernel methods can employ dynamic programming solutions to efficiently enumerate over substructures that would be too costly to explicitly include as features.

Another example that is relevant in information extraction is the tree kernel. Tree kernels constitute a particular case of more general kernels defined on a discrete structure (*convolution kernels*) (Collins and Duffy, 2001). The idea is to split the structured object in parts and to define a kernel on the "atoms" and to recursively compute the kernel over larger parts in order to get the kernel of the whole structure.

The property of kernel methods to map complex objects in a feature space where a more easy discrimination between objects can be performed and the capability of the methods to efficiently consider the features of complex objects make them also interesting for information extraction tasks. In information extraction we can combine parse tree similarity with a similarity based on feature correspondence of the nodes of the trees. In the feature vector of each node additional attributes can be modeled (e.g., POS, general POS, entity type, entity level, WordNet hypernyms). Another example in information extraction would be to model script tree similarity of discourses where nodes store information about certain actions and their arguments.

We illustrate the use of a tree kernel in an entity relation recognition task (Zalenko et al., 2003; Culotto and Sorensen, 2004). More specifically the purpose of this research is to find relations between entities that are already recognized as persons, companies, locations, etc. (e.g., John **works for** Concentra).

In this example, the training set is composed of parsed sentences in which the sought relations are annotated. For each entity pair found in the same sentence, a dependency tree of this training example is captured based on the syntactic parse of the sentence. Then, a tree kernel can be defined that is used in a SVM to classify the test examples.

The kernel function incorporates two functions that consider attribute correspondence of two nodes t_i and t_j: A matching function $m(t_i,t_j) \in \{0, 1\}$ and a similarity function $s(t_i,t_j) \in [0,\infty]$. The former just determines whether two nodes are matchable or not, i.e., two nodes can be matched when they are of compatible type. The latter computes the correspondence of the nodes t_i and t_j based on a similarity function that operates on the nodes' attribute values.

For two dependency trees T_1 and T_2 the tree kernel $K(T_1,T_2)$ can be defined by the following recursive function:

$$K(t_i,t_j) = \begin{cases} 0, & \text{if } m(t_i, t_j) = 0 \\ s(t_i,t_j) + K_c(t_i[c], t_j[c]) & \text{otherwise} \end{cases} \tag{5.20}$$

where K_c is a kernel function that defines the similarity of the tree in terms of children subsequences. Note that two nodes are not matchable when one of them is nil. Let a and b be sequences of indices such that a is a sequence $a_1 \leq a_2 \leq \ldots a_k$ and likewise for b. Let $d(a) = a_k - a_1 + 1$ and $l(a)$ be the length of a. Then K_c can be defined as:

$$K_c(t_i[c], t_j[c]) = \sum_{a,b,l(a)=l(b)} \lambda^{d(a)} \lambda^{d(b)} K(t_i[a], t_j[b]) \tag{5.21}$$

The constant $0 < \lambda < 1$ is a decay factor that penalizes matching subsequences that are spread out within the child sequences.

Intuitively, whenever we find a pair of matching nodes, the model searches for all matching subsequences of the children of each node. For each matching pair of nodes (t_i,t_j) in a matching subsequence, we accumulate the result of the similarity function $s(t_i,t_j)$ and then recursively search for matching subsequences of their children $t_i[c]$ and $t_j[c]$. Two types of tree kernels are considered in this model. A contiguous kernel only matches child subsequences that are uninterrupted by non-matching nodes. Therefore, $d(a) = l(a)$. On the other hand, a sparse tree kernel, allows non-matching nodes within matching subsequences.

The above example shows that kernel methods have a lot to offer in information extraction. Complex information contexts can be modeled in a kernel function, and problem-specific kernel functions can be drafted. The problem is then concentrated on finding the suitable kernel function. The use of kernels as a general technique for using linear machines in a non-linear fashion can be exported to other learning systems (e.g., nearest neighbor classifiers).

Generally, Support Vector Machines are successfully employed in named entity recognition tasks (e.g., Isozaki and Kazawa, 2002), noun phrase coreferent resolution (e.g., Isozaki and Hirao, 2003) and semantic role recognition (e.g., Zhang and Lee 2003; Mehay et al., 2005) and in entity relation recognition (Culotto and Sorensen, 2004). As explained above Support Vector Machines have the advantage that they can cope with many (sometimes) noisy features without being doomed by the curse of dimensionality.

5.3 Maximum Entropy Models

The *maximum entropy model* (sometimes referred to as MAXENT) computes the probability distribution $p(\boldsymbol{x},y)$ with maximum entropy that satisfies the constraints set by the training examples (Berger et al., 1996). Among the possible distributions that fit the training data, the one is chosen that maximizes the entropy. The concept of entropy is known from Shannon's information theory (Shannon, 1948). It is a measure of uncertainty concerning an event, and from another viewpoint a measure of randomness of a message (here a feature vector).

Let us first explain the maximum entropy model with a simple example of named entity recognition. Suppose we want to model the probability of a named entity being a disease or not when it appears in three very simple contexts. In our example the contexts are composed of the word that is to be classified being one of the set {**neuromuscular**, **Lou Gerigh**, **person**}. In other words, the aim is to compute the joint probability distribution p defined over {*neuromuscular, Lou Gerigh, person*} x {*disease, nodisease*} given a training set S of n training examples:

$$S = \{(\boldsymbol{x}, y)_1,\ (\boldsymbol{x}, y)_2\ ,\ldots,\ (\boldsymbol{x},y)_n\}.$$

Because p is a probability distribution, a first constraint on the model is that:

$$\sum_{\boldsymbol{x},y} p(\boldsymbol{x},y) = 1 \qquad (5.22)$$

or
p(*neuromuscular*, *disease*) + *p*(*Lou Gerigh*, *disease*) + *p* (*person*, *disease*) + *p*(*neuromuscular*, *nodisease*) + *p*(*Lou Gerigh*, *nodisease*) + *p*(*person*, *nodisease*) = 1

It is obvious that numerous distributions satisfy this constraint, as seen in the Tables 5.1 and 5.2.

The training set will impose additional constraints on the distribution. In a maximum entropy framework, constraints imposed on a model are represented by k binary-valued[3] features known as feature functions. A feature function f_j takes the following form:

$$f_j(\boldsymbol{x}, y) = \begin{cases} 1 & \text{if } (\boldsymbol{x}, y) \text{ satisfies a certain constraint} \\ 0 & \text{otherwise} \end{cases} \tag{5.23}$$

From the training set we learn that in 50% of the examples in which a disease is mentioned the term **Lou Gerigh** occurs and that 70% of the examples of the training set are classified as *disease* imposing the following constraints expressed by the feature functions:

Table 5.1. An example of a distribution that satisfies the constraint in Eq. (5.22).

	disease	*nodisease*	
neuromuscular	1/4	1/8	
Lou Gerigh	1/8	1/8	
person	1/8	1/4	
Total			1.0

Table 5.2. An example of a distribution that in the most uncertain way satisfies the constraint in Eq. (5.22).

	disease	*nodisease*	
neuromuscular	1/6	1/6	
Lou Gerigh	1/6	1/6	
person	1/6	1/6	
Total			1.0

[3] The model is not restricted to binary features. For binary features efficient numerical methods exist for computing the model parameters of Eq. (5.35).

$$f_{LouGehrig}(\boldsymbol{x}, y) = \begin{cases} 1 & \text{if } x_1 = \textit{Lou Gerigh} \text{ and } y = \textit{disease} \\ 0 & \text{otherwise} \end{cases} \tag{5.24}$$

$$f_{disease}(\boldsymbol{x}, y) = \begin{cases} 1 & \text{if } y = \textit{disease} \\ 0 & \text{otherwise} \end{cases} \tag{5.25}$$

In this simplified example, our training set does not give any information about the other context terms. The problem is how to find the most uncertain model that satisfies the constraints. In Table 5.3 one can again look for the most uniform distribution satisfying these constraints, but the example makes it clear that the choice is not always obvious. The maximum entropy model offers here a solution. Thus, when training the system, we choose the model p^* that preserves as much uncertainty as possible, or which maximizes the entropy $H(p)$ between all the models $p \in P$ that satisfy the constraints enforced by the training examples.

$$H(p) = -\sum_{(x,y)} p(x, y) \log p(x, y) \tag{5.26}$$

$$p^* = \arg\max_{p \in P} H(p) \tag{5.27}$$

In the examples above we have considered an input training example characterized by a certain label y and a feature vector $\boldsymbol{x}$**,** containing the context of the word (e.g., as described by surrounding words and their POS tag). We can collect n number of training examples and summarize the training sample S in terms of its empirical probability distribution: $\tilde{p}$ defined by:

Table 5.3. An example of a distribution that satisfies the constraints in Eqs. (5.22), (5.24) and (5.25).

	disease	*nodisease*	
neuromuscular	?	?	
Lou Gerigh	0.5	?	
person	?	?	
Total	0.7		1.0

$$\tilde{p}(\boldsymbol{x},y) \equiv \frac{no}{n} \tag{5.28}$$

where no = number of times a particular pair $(\boldsymbol{x},y)$ occurs in S and $no \geq 0$.

We want to compute the expected value of the feature function f_j with respect to the empirical distribution $\tilde{p}(\boldsymbol{x},y)$.[4]

$$E_{\tilde{p}}(f_j) = \sum_{\boldsymbol{x},y} \tilde{p}(\boldsymbol{x},y) f_j(\boldsymbol{x},y) \tag{5.29}$$

The statistics of a feature function are captured and it is required that the model that we are building accords with it. We do this by constraining the expected value that the model assigns to the corresponding feature function f_j. The expected value of f_j with respect to the model $p(y|\boldsymbol{x})$ is:

$$E_p(f_j) = \sum_{\boldsymbol{x},y} \tilde{p}(\boldsymbol{x}) p(y|\boldsymbol{x}) f_j(\boldsymbol{x},y) \tag{5.30}$$

where $\tilde{p}(\boldsymbol{x})$ is the empirical distribution of x in the training sample. We constrain this expected value to be the same as the expected value of f_j in the training sample, i.e., the empirical expectation of f_j. That is we require:

$$E_p(f_j) = E_{\tilde{p}}(f_j) \tag{5.31}$$

Combining Eqs. (5.29), (5.30) and (5.31) yields the following constraint equation:

$$\sum_{\boldsymbol{x},y} \tilde{p}(\boldsymbol{x}) p(y|\boldsymbol{x}) f_j(\boldsymbol{x},y) = \sum_{\boldsymbol{x},y} \tilde{p}(\boldsymbol{x},y) f_j(\boldsymbol{x},y) \tag{5.32}$$

By restricting attention to these models $p(y|\boldsymbol{x})$ for which Eq. (5.31) holds, we are eliminating from consideration those models that do not agree with the training samples. In addition, according to the principle of maximum entropy we should select the distribution which is most uniform. A

[4] The notation is sometimes abused: $f_j(\boldsymbol{x},y)$ will both denote the value of f_j for a particular pair $(\boldsymbol{x},y)$ as well as the entire function f_j.

mathematical measure of the uniformity of a conditional distribution $p(y|\boldsymbol{x})$ is provided by the conditional entropy. The conditional entropy $H(Y|X)$ measures how much entropy a random variable Y has remaining, if we have already learned completely the value of a second random variable X. The conditional entropy of a discrete random Y given X:

$$H(Y|X) = \sum_{x \in X} p(x) H(Y|X = x) \tag{5.33}$$

$$H(Y|X) = -\sum_{x \in X} p(x) \sum_{y \in Y} p(y|x) \log p(y|x) \tag{5.34}$$

or[5]

$$H(p) \equiv -\sum_{\boldsymbol{x} \in X, y \in Y} \tilde{p}(\boldsymbol{x}) p(y|\boldsymbol{x}) \log p(y|\boldsymbol{x})$$

Note that $\tilde{p}(\boldsymbol{x})$ is estimated from the training set and $p(y|\boldsymbol{x})$ is the learned model. When the model has no uncertainty at all, the entropy is zero. When the values of y are uniformly distributed, the entropy is $\log |y|$. It has been shown that there is always a unique model $p^*(y|\boldsymbol{x})$ with maximum entropy that obeys the constraints set by the training set. Considering the feature vector $\boldsymbol{x}$ of a test example, this distribution has the following exponential form:

$$p^*(y|\boldsymbol{x}) = \frac{1}{Z} \exp\Big(\sum_{j=1}^{k} \lambda_j f_j(\boldsymbol{x}, y)\Big), \quad 0 < \lambda_j < \infty \tag{5.35}$$

where $f_j(\boldsymbol{x}, y)$ is one of the k binary-valued feature functions
λ_j = parameter adjusted to model the observed statistics
Z = normalizing constant computed as:

$$Z = \sum_{y} \exp\Big(\sum_{j=1}^{k} \lambda_j f_j(\boldsymbol{x}, y)\Big) \tag{5.36}$$

So, the task is to define the parameters λ_j in p which maximize $H(p)$. In simple cases we can find the solution analytically, in more complex cases

[5] Following Berger et al. (1996) we use here the notation $H(p)$ in order to emphasize the dependence of the entropy on the probably distribution p instead of the common notation $H(Y|X)$ where Y and X are random variables.

we need numerical methods to derive λ_j given a set of constraints. The problem can be considered as a constrained optimization problem, where we have to find a set of parameters of an exponential model, which maximizes its log likelihood. Different numerical methods can be applied for this task among which are generalized iterative scaling (Darroch and Ratcliff, 1972), improved iterative scaling (Della Pietra et al., 1997), gradient ascent and conjugate gradient (Malouf, 2002).

We also have to efficiently compute the expectation of each feature function. Eq. (5.30) cannot be efficiently computed, because it would involve summing over all possible combinations of $\boldsymbol{x}$ and y, a potentially infinite set. Instead the following approximation is used, which takes into account the n training examples $\boldsymbol{x}_i$:

$$E_p(f_j) = \frac{1}{n} \sum_{i=1}^{n} \sum_{y} p(y \mid \boldsymbol{x}_i) f_j(\boldsymbol{x}_i, y) \tag{5.37}$$

The maximum entropy model has been successfully applied to natural language tasks in which context-sensitive modeling is important (Berger et al., 1996; Ratnaparkhi, 1998) among which is information extraction. The model has been used in named entity recognition (e.g., Chieu and Hwee 2002), coreference resolution (e.g., Kehler, 1997) and semantic role recognition (Fleischman et al., 2003; Mehay et al., 2005). The maximum entropy model offers many *advantages*. The classifier allows to model dependencies between features, which certainly exist in many information extraction tasks. The classifier has the advantage that there is no need for an a priori feature selection, as features that just are randomly associated with a certain class, will keep their randomness in the model. This has the advantage that you can train and experiment with many context features in the model, in an attempt to decrease the ambiguity of the learned patterns. Moreover, the principle of maximum entropy states that when we make inferences based on incomplete information, we should draw them from a probability distribution that has the maximum entropy permitted by the information that we do have (Jaynes, 1982). In many information extraction tasks, our training set is often incomplete given the large variety of natural language patterns that convey the semantic classes sought. Here, the maximum entropy approach offers a satisfactory solution.

The above classification methods assume that there is no relation between various classes. In information extraction in particular and in text understanding in general, content is often dependent. For instance, when there is no grant approved, there is also no beneficiary of the grant. Or, more formally one can say: There is only one or a finite number of ways in which information can be sequenced in a text or in a text sentence in order to convey its meaning. The scripts developed by Schank and his school in the 1970s and 1980s are an illustrative example (e.g., you have to get on the bus before you can ride the bus). But also, at the more fine-grained level of the sentence the functional position of an information unit in dependency with the other units defines the fine-grained meaning of the sentence units (e.g., semantic roles). In other words, information contained in text often has a certain dependency, one cannot exist without the other, or it has a high chance to occur with other information. This dependency and the nature of the dependency can be signaled by lexical items (and their co-occurrence in a large corpus) and refined by the syntactical constructs of the language including the discourse structure.

In pattern recognition there are a number of algorithms for context-dependent classification. In these models, the objects are described by feature vectors, but the features and their values stored in different feature vectors together contribute to the classification. In order to reduce the computational complexity of the algorithms the vectors are often processed in a certain order and the dependency upon vectors previously processed is limited. The class to which a feature vector is assigned depends on its own value, on the values of the other feature vectors and on the existing relation among the various classes. In other words, having obtained the class c_i for a feature vector $\boldsymbol{x}_i$, the next feature vector could not always belong to any other class. In the following sections we will discuss two common approaches to context-dependent information recognition: Hidden Markov models and conditional random fields. We foresee that many other useful context dependent classification algorithms will be developed in text understanding. In context-dependent classification, feature vectors are often referred to as observations. For instance, the feature vector $\boldsymbol{x}_i$ occurs in a sequence of observations $X = (\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T)$.

5.4 Hidden Markov Models

In Chap. 2 we have seen that finite state automata quite successfully recognize a sequence of information units in a sentence or a text. In such a model a text is considered as a sequence of symbols and not as an unordered set. The task is to assign a *class sequence* $Y= (y_1,\ldots,y_T)$ to the *sequence of observations* $X = (\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T)$. Research in information extraction has recently investigated probabilistic sequence models, where the task is to assign the most probable sequence of classes to the chain of observations. Typically, the model of the content is implemented as a *Markov* *chain of states*, in which transition probabilities between states and the probabilities of emissions of certain symbols of the alphabet are modeled. In Fig. 5.3 the content of a Belgian criminal court decision is modeled as a Markov chain.

The states are shown as circles and the start state is indicated as *start.* Possible transitions are shown by edges that connect states, and an edge is labeled with the probability of this transition. Transitions with zero probability are omitted from the graph. Note that the probabilities of the edges that go out from each state sum to 1. From this representation, it should be clear that a Markov model can be thought as a (non-deterministic) finite state automaton with probabilities attached to each edge.

The probability of a sequence of states or classes $Y = (y_1,\ldots,y_T)$ is easily calculated for a Markov chain:

$$P(y_1,\ldots,y_T) = P(y_1)P(y_2|y_1)\, P(y_3|y_1, y_2) \ldots \mathrm{P}\, (y_T|y_1,\ldots,y_{T-1}) \tag{5.38}$$

A first order Markov model assumes that class dependence is limited only within two successive classes yielding:

$$P(y_1,\ldots,y_T) = P(y_1)P(y_2|y_1)\, P(y_3|y_2)\ldots P\, (y_T|\, y_{T-1}) \tag{5.39}$$

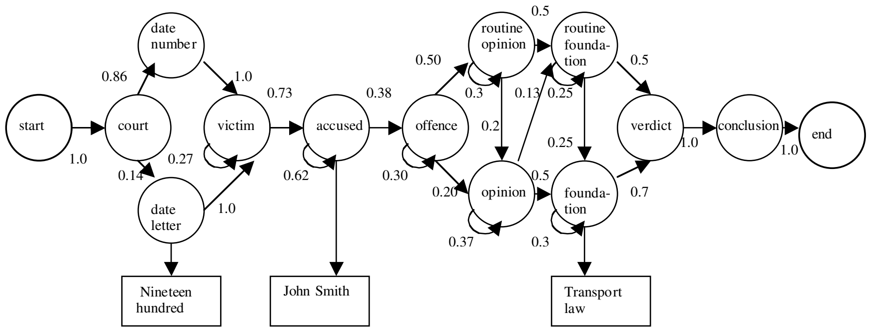

Fig. 5.3. An example Markov model that represents a Belgian criminal court decision. Some examples of emissions are shown without their probabilities.

$$= P(y)\prod_{i=2}^{T} P(y_i|y_{i-1}) \tag{5.40}$$

In Fig. 5.3 only some of the emission symbols are shown. The models that we consider in the context of information extraction have a discrete output, i.e., an observation outputs discrete values.

A first order Markov model is composed of a set of states Y with specified initial and final states y_1 and y_T, a set of transitions between states, and a discrete vocabulary of output symbols $\Sigma = \{\sigma_1, \sigma_2,\ldots,\sigma_k\}$. In information extraction the emission symbols are usually words. The model generates an observation $X = (\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T)$ by beginning in the initial state, transitioning to a new state, emitting an output symbol, transitioning to another state, emitting another symbol, and so on, until a transition is made into the final state. The parameters of the model are the transition probabilities $P(y_i|y_{i-1})$ that one state follows another and the emission probabilities $P(\boldsymbol{x}_i|y_i)$ that a state emits a particular output symbol.[6]

Classification regards the recognition of the most probable path in the model. For the task of information extraction this translates into the following procedure. Having observed the following sequence of feature vectors $X = (\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T)$, we have to find the respective sequence of classes or states $Y = (y_1,\ldots,y_T)$ that is most probably followed in the model. We compute Y^* for which

$$Y^* = \underset{Y}{argmax}\, P(Y|X) \tag{5.41}$$

$$P(Y|X) = P(y_1)P(\boldsymbol{x}_1|y_1)\prod_{i=2}^{T} P(y_i|y_{i-1})P(\boldsymbol{x}_i|y_i) \tag{5.42}$$

In order to compute the most probable path the *Viterbi algorithm* is used. Instead of a brute-force computation, by which all possible paths are computed, the Viterbi algorithm efficiently computes a subset of these paths. It is based on the observation that, if you look at the best path that goes through a given state y_i at a given time t_i, the path is the concatenation of the best path that goes from state y_1 to y_i (while emitting symbols corresponding to the feature vectors $\boldsymbol{x}_1$ to $\boldsymbol{x}_i$ respectively at times t_1 to t_i) with the best path from state y_i to the final state y_T (while emitting symbols corresponding to the feature vectors $\boldsymbol{x}_{i+1}$ to $\boldsymbol{x}_T$ respectively at times t_{i+1} to t_T). This is because the probability of a path going through state y_i is simply the product of the probabilities of the two parts (before and after y_i), so that

[6] We mean here the discrete symbol that is represented by the feature vector $\boldsymbol{x}$.

the maximum probability of the global path is obtained when each part has a maximum probability.

When we want to *train a Markov model* based on labeled sequences X_{all} there are usually two steps. First, one has to define the model of states or classes, which is called the topology of the model. Secondly, one has to learn the emission and transition probabilities of the model. The first step is usually drafted by hand when training an information extraction system (although at the end of this section we will mention some attempts to learn a state model). In the second step, the probabilities of the model are learned based on the classified training examples. The task is learning the probabilities of the initial state, the state transitions and the emissions of a model μ.

In a *visible Markov model* (Fig. 5.4), the state sequence that actually generated an example is known, i.e., we can directly observe the states and the emitted symbols. If we can identify the path that was taken inside the model to produce each training sequence, we are able to estimate the probabilities by the relative frequency of each transition from each state and of emitting each symbol. The labeling is used to directly compute the probabilities of the parameters of the Markov model by means of maximum likelihood estimates in the training set X_{all}. The transition probabilities $P(y'|y)$ and the emission probabilities $P(\boldsymbol{x}|y)$ are based on the counts of respectively the class transitions $\xi(y\text{->}y')$ or $\xi(y,y')$ and of the emissions occurring in a class $\gamma(y)$ where $y\uparrow \boldsymbol{x}_i$ considered at the different times t:

$$P(y'|y) = \frac{\sum_{t=1}^{T-1} \xi_t(y,y')}{\sum_{t=1}^{T-1} \gamma_t(y)} \tag{5.43}$$

$$P(\boldsymbol{x}|y) = \frac{\sum_{t=1 \text{ and } y\uparrow \boldsymbol{x}}^{T} \gamma_t(y)}{\sum_{t=1}^{T} \gamma_t(y)} \tag{5.44}$$

In a *hidden Markov model* (Rabiner, 1989) (Fig. 5.5) you do not know the state sequence that the model passed through when generating the training examples. The states of the training examples are not fully observable. This is often the case in an information extraction task from a sequence of words. Each state (e.g., word) is associated with a class that we want to extract. Some states are just background states, when they represent information not to be extracted or semantically labeled. As a result some of the words are observed as emission symbols and have an unknown class or state.

In this case the transition and emission probabilities are inferred from a sequence of observations via some optimization function that is iteratively computed. The training of parameters is usually performed via the Baum-Welch algorithm, which is a special case of the Expectation-Maximization algorithm (EM) (Dempster et al., 1977).

The task is learning the probabilities of the initial state, the state transitions and the emissions of the model μ. The Baum-Welch approach is characterized by the following steps:

1. Start with initial estimates for the probabilities chosen randomly or according to some prior knowledge.
2. Apply the model on the training data:

   Expectation step (E): Use the current model and observations to calculate the expected number of traversals across each arc and the expected number of traversals across each arc while producing a given output.

   Maximization step (M): Use these calculations to update the model into a model that most likely produces these ratios.

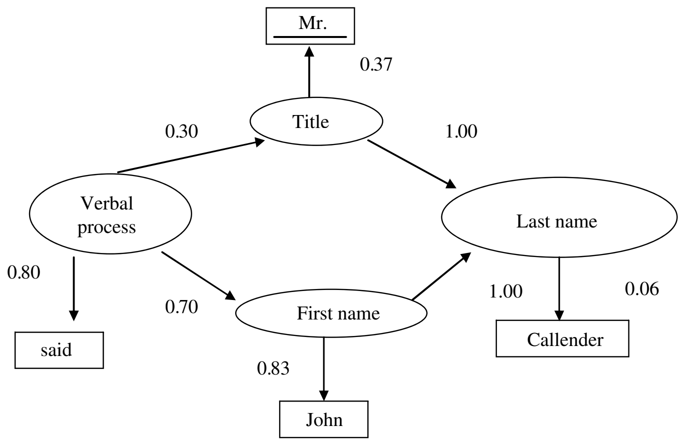

Fig. 5.4. Example of a visible Markov Model for a named entity recognition task.

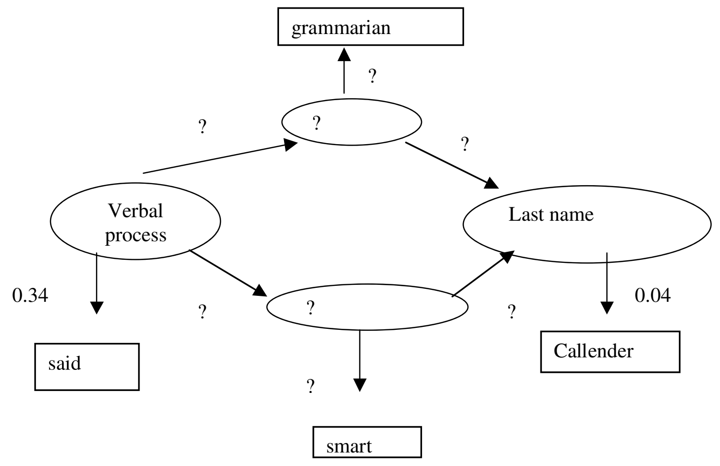

Fig. 5.5. Example of a hidden Markov model for a named entity recognition task.

3. Iterate step 2 until a convergence criterion is satisfied (e.g., when the differences of the values with the values of a previous step are smaller than a threshold value ε).

Expectation step (E)

We consider the number of times that a path passes through state *y* at time *t* and through state *y'* at the next time *t* + 1 and the number of times this state transition occurs while generating the training sequences X_{all} given the parameters of the current model μ. We then can define:

$$\xi_t(y,y') \equiv \xi_t(y,y'|X_{all},\mu) = \frac{\xi_t(y,y',X_{all}|\mu)}{P(X_{all}|\mu)} \tag{5.45}$$

$$= \frac{\alpha(y_t = y)P(y'|y)P(\boldsymbol{x}_{t+1}|y')\beta(y_{t+1} = y')}{P(X_{all}|\mu)} \tag{5.46}$$

where $\alpha(y_t = y)$ represents the path history terminating at time *t* and state *y* (i.e., the probability of being at state *y* at time *t* and outputting the first *t* symbols) and $\beta(y_{t+1} = y')$ represents the future of the path, which at time *t* + 1 is at state *y'* and then evolves unconstrained until the end (i.e., the probability of being at the remaining states and outputting the remaining symbols). We define also the probability of being at time *t* at state *y*:

$$\gamma_t(y) \equiv \gamma_t(y|X_{all},\mu) = \frac{\alpha(y_t = y)\beta(y_t = y)}{P(X_{all}|\mu)} \tag{5.47}$$

$\sum_{t=1}^{T-1} \gamma_t(y)$ can be regarded as the expected number of transitions from state *y* given the model μ and the observation sequences X_{all}.

$\sum_{t=1}^{T-1} \xi_t(y, y')$ can be regarded as the expected number of transitions from state *y* to state *y'*, given the model μ and the observation sequences X_{all}.

Maximization step (M)

During the M-step the following formulas compute reasonable estimates of the unknown model parameters:

$$\overline{P}(y'|y) = \frac{\sum_{t=1}^{T-1} \xi_t(y, y')}{\sum_{t=1}^{T-1} \gamma_t(y)} \tag{5.48}$$

$$\overline{P}(\boldsymbol{x}|y) = \frac{\sum_{t=1 \text{ and } y\uparrow\boldsymbol{x}}^{T} \gamma_t(y)}{\sum_{t=1}^{T} \gamma_t(y)} \tag{5.49}$$

$$\overline{P}(y) = \gamma_1(y) \tag{5.50}$$

Practical implementations of the HMM have to cope with problems of zero probabilities as the values of $\alpha(y_t)$ and $\beta(y_t)$ are smaller than one and when used in products tend to go to zero, which demands for an appropriate scaling of these values.

A hidden Markov model is a popular technique to detect and *classify a linear sequence of information in text*. The first information extraction systems that used HMM technology were developed by Leek (1997), whose system extracted gene names and locations from scientific abstracts, and by Bikel et al. (1997) who used this technology for named entity recognition. McCallum et al. (1999) extracted document segments that occur in a fixed or partially fixed order, such as the title, author, and journal from both the headers and reference sections of papers. Ray and Craven (2001) apply HMMs to Medline texts to extract the proteins, locations, genes and disorders and their relationships. Zhang et al. (2004) use also HMMs for the recognition of biomedical named entities.

The disadvantage of using HMM for information extraction is that we need large amounts of training data for guaranteeing that all state transitions will appear a sufficient number of times in order to learn the probabilities in a satisfactory way. Content can be expressed in many linguistic variant forms, not at least if one just considers the words of a text. In addition there is the need for an a priori notion of the model's topology (the possible sequences of states) or that this topology should automatically be learned. Existing work has generally used a handcrafted topology, in which states are connected manually in a reasonable way after evaluating the training corpus. There have been several attempts to automatically learn an appropriate topology for information extraction tasks. Examples can be found in Seymore et al. (1999) and McCallum et al. (1999).

5.5 Conditional Random Fields

Conditional random fields (CRF) regard a statistical method based on undirected graphical models. The method exhibits a number of properties that makes it very well suited for information extraction tasks. Like the discriminative learning models it can accommodate many statistically correlated features of the input example. This contrasts with generative models, which often require conditional independent assumptions in order to make the computations tractable. Nonetheless, the discriminative methods seen so far do not incorporate context dependency of classes unless they resort to some heuristics to find an acceptable combination of classes. Conditional random fields incorporate both the possibility of incorporating *dependent features* and the possibility of *context-dependent learning*, making the technique as one of the best current approaches to information extraction in empirical evaluations (Lafferty et al., 2001). This method can be thought of a generalization of both the maximum entropy model and the hidden Markov model.

Let $\boldsymbol{X}$ be a random variable over data sequences to be labeled and $\boldsymbol{Y}$ a random variable over corresponding label sequences. All components $\boldsymbol{Y_i}$ of $\boldsymbol{Y}$ are assumed to range over a finite label alphabet Σ. For example, in an information extraction task, $\boldsymbol{X}$ might range over the sentences of a text, while $\boldsymbol{Y}$ ranges over the semantic classes to be recognized in these sentences. A conditional random field is viewed as an undirected graphical model or Markov random field, conditioned on $\boldsymbol{X}$ (Jordan, 1999, Wallach, 2004). We define $G = (V, E)$ to be an undirected graph such that there is a node $v \in V$ corresponding to each of the random variables representing an element Y_v of $\boldsymbol{Y}$. If each random variable Y_v obeys the Markov property with respect to G (e.g., in a first order model the transition probability depends only on the neighboring state), then the model $(\boldsymbol{Y}, \boldsymbol{X})$ is a conditional random field. In theory the structure of graph G may be arbitrary, however, when modeling sequences, the simplest and most common graph structure encountered is that in which the nodes corresponding to elements of $\boldsymbol{Y}$ form a simple first-order Markov chain. In this case the conditional random field forms a probabilistic finite state automaton.

In information extraction conditional random fields are often used to label sequential data, although the method can also be used in other settings. We focus here on a conditional random field that represents a sequence of extraction classes. Such a CRF defines a conditional probability distribution $p(\boldsymbol{Y}|\boldsymbol{X})$ of label sequences given input sequences. We assume that the random variable sequences $\boldsymbol{X}$ and $\boldsymbol{Y}$ have the same length and use $\boldsymbol{x} = (x_1,\ldots,x_T)$ and $y = (y_1,\ldots,y_T)$ for an input sequence and label sequence respectively.[7] Instead of defining a joint distribution over both label and observation sequences, the model defines a conditional probability over labeled sequences. A novel observation sequence $\boldsymbol{x}$ is labeled with y, so that the conditional probability $p(y|\boldsymbol{x})$ is maximized.

Comparable to the maximum entropy model, we define a set of k binary-valued[8] features or *feature functions* that each express some characteristic of the empirical distribution of the training data that should also hold in the model distribution. Each local feature is either a state feature $s(y_i, \boldsymbol{x}, i)$ or a transition feature $t(y_{i-1}, y_i, \boldsymbol{x}, i)$, where y_{i-1} and y_i are class labels, $\boldsymbol{x}$ an input sequence, and i an input position. When i is 1 (start state of the sequence), $t(y_{i-1}, y_i, \boldsymbol{x}, i) = 0$. Examples of such features are:

[7] Note that we represent here an instantiation of an observation sequence as $\boldsymbol{x}$ in contrast with the rest of this book where we use $\boldsymbol{x}$ as an instantiation of a feature vector. Analogically, we use y for the representation of a label sequence.

[8] See footnote 3.

$$s_j(y_i, \boldsymbol{x}, i) = \begin{cases} 1 \text{ if the observation at position } i \text{ is the word "say"} \\ 0 \text{ otherwise} \end{cases} \tag{5.51}$$

$$t_j(y_{i-1}, y_i, \boldsymbol{x}, i) = \begin{cases} 1 \text{ if } y_{i-1} \text{ has tag "title" and } y_i \text{ has POS tag "NNP"} \\ 0 \text{ otherwise} \end{cases} \tag{5.52}$$

Feature functions thus depend on the current state (in case of a state feature function) or on the previous and current states (in the case of a transition feature function). We can use a more global notation f_j for a feature function where $f_j(y_{i-1}, y_i, \boldsymbol{x}, i)$ is either a state function $s_j(y_i, \boldsymbol{x}, i) = s_j(y_{i-1}, y_i, \boldsymbol{x}, i)$ or a transition function $t_j(y_{i-1}, y_i, \boldsymbol{x}, i)$.

The CRF's global feature vector $F_j(\boldsymbol{x},y)$ for the input sequence $\boldsymbol{x}$ and label sequence y is given by:

$$F_j(\boldsymbol{x}, y) = \sum_{i=1}^{T} f_j(y_{i-1}, y_i, \boldsymbol{x}, i) \tag{5.53}$$

where i ranges over input positions (such as a sequence of words in a document) or in terms of the graphical model over the values on T input nodes. Considering k feature functions, the conditional probability distribution defined by the CRF is then:

$$p(y|\boldsymbol{x}) = \frac{1}{Z}\exp\Big(\sum_{j=1}^{k} \lambda_j F_j(\boldsymbol{x}, y)\Big)$$

or

$$p(y|\boldsymbol{x}) = \frac{1}{Z}\exp\Big(\sum_{j=1}^{k}\sum_{i=1}^{T} \lambda_j f_j(y_{i-1}, y_i, \boldsymbol{x}, i)\Big) \tag{5.54}$$

where λ_j = parameter adjusted to model the observed statistics
Z = normalizing constant computed as:

$$Z = \sum_{y \in \boldsymbol{Y}} \exp\Big(\sum_{j=1}^{k} \lambda_j F_j(\boldsymbol{x}, y)\Big)$$

Z is a normalization factor for observation sequence $\boldsymbol{x}$ computed over different possible state sequences and f_j ranges over all k feature functions.

The most probable label sequence y^* for input sequence $\boldsymbol{x}$ is:

$$y^* = \underset{y}{argmax}\, p(y|\boldsymbol{x}) \tag{5.55}$$

For a chain-structured conditional random field, the probability $p(y|\boldsymbol{x})$ of label sequence y given an observation sequence $\boldsymbol{x}$ can be easily computed by using matrices and relying on algorithms for solving path problems in graphs. To simplify the expressions we add a *start* and *end* state, respectively represented by y_o and y_{T+1}. Let Σ be de alphabet from which labels are drawn and y and y' be labels drawn from this alphabet, we define a set of $T + 1$ matrices $\{M_i(\boldsymbol{x}) \mid \mathrm{i} = 1,\ldots, T + 1\}$, where each $M_i(\boldsymbol{x})$ is a $|\Sigma| \mathrm{x} |\Sigma|$ matrix with elements of the form:

$$M_i(y', y|\boldsymbol{x}) = \exp\Big(\sum_{j=1}^{k} \lambda_j f_j(y', y, \boldsymbol{x}, i)\Big) \tag{5.56}$$

The conditional probability of a label sequence y given observation sequence $\boldsymbol{x}$ can be written as:

$$p(y|\boldsymbol{x}) = \frac{1}{Z} \prod_{i=1}^{T+1} M_i(y_{i-1}, y_i|\boldsymbol{x}) \tag{5.57}$$

The normalization factor Z for observation sequence $\boldsymbol{x}$, may be computed from the set of $M_i(\boldsymbol{x})$ matrices. Z is given by the (*start*, *end*) entry of the product of all $T + 1$ $M_i(\mathbf{x})$ matrices.

$$Z = \left[\prod_{i=1}^{T+1} M_i(x)\right]_{start,\, end} \tag{5.58}$$

The conditional random field as defined by Eq. (5.54) is heavily motivated by the *principle of maximum entropy*. As seen earlier in this chapter the entropy of a probability distribution is a measure of uncertainty and is maximized when the distribution in question is as uniform as possible, subject to the constraints set by the training examples. The distribution that is as uniform as possible while ensuring that the expectation of each feature function with respect to the empirical distribution of the training data equals the expected value of the feature function with respect to the model distribution.

As for the maximum entropy model, we need numerical methods in order to derive λ_j given the set of constraints. The problem can be considered as a constrained optimization problem, where we have to find a set of parameters of an exponential model, which maximizes its log likelihood. We refer here to the references given above on numerical methods for determining the model parameters for the maximum entropy model. Here also, we are confronted with the problem of efficiently calculating the *expectation of each feature function* with respect to the CRF model distribution for every observation sequence $\boldsymbol{x}$ in the training data. Fortunately, dynamic programming techniques that are similar to the Baum-Welch algorithm that is commonly used for estimating the parameters of a hidden Markov model, can be used here for parameter estimation (Lafferty et al., 2001).

Conditional random fields have been implemented for named entity recognition (McCallum and Li, 2003) and timex recognition and normalization (Ahn et al., 2005). They allow representing dependencies on previous classifications in a discourse. While adhering to the maximum entity principle, they offer a valid solution when learning from incomplete information. Given that in information extraction tasks, we often lack an annotated training set that covers all extraction patterns, this is a valuable asset.

Conditional random fields are a restricted class of *undirected graphical models* (Jordan, 1999). The advantage is that the feature functions can model many characteristics of the texts not only with regard to an input sequence, its terms and their characteristics, but they can also take into account other discourse features that occur in previous sentences. Conditional random fields have here been illustrated with the case of a linear sequence of observations. Other graph models can be valuable for information extraction tasks.

For instance, a *relational Markov network* can represent arbitrary dependencies between extractions (e.g., Taskar et al., 2004). This model allows for a collective classification of a set of related entities by integrating information from features of individual entities as well as the relations between them. For example, in a protein named entity recognition task, repeated references to the same protein are common. If the context surrounding one occurrence of a name offers good evidence that the name is a protein, then this should influence the tagging of another occurrence of the same name in a different ambiguous context, if we assume the one sense per discourse heuristic (Bunescu and Mooney, 2004).

5.6 Decision Rules and Trees

Learning of rules and trees aims at inducing classifying expressions in the form of decision rules and trees from example cases. These are one of the oldest approaches to machine learning and were also part of one of the oldest applications of machine learning in information extraction. Each decision rule is associated with a particular class, and a rule that is satisfied, i.e., evaluated as true, is an indication of its class. Thus, classifying new cases involves the application of the learned classifying expressions and assignment to the corresponding class upon positive evaluation.

The rules are found by searching these combinations of features or of feature relations that are discriminative for each class. Given a set of positive examples and a set of negative examples (if available) of a class, the training algorithms generate a rule that covers all (or most) of the positive examples and none (or fewest) of the negative examples. Having found this rule, it is added to the rule set, and the cases that satisfy the rule are removed from further consideration. The process is repeated until no more example cases remain to be covered.

The paradigm of searching possible hypotheses also applies to tree and rule learning. There are two major ways for accessing this *search space* (Mitchell, 1977). *General-to-specific* methods search the space from the most general towards the most specific hypothesis. One starts from the most general rule possible (often an empty clause), which is specialized at the encounter of a negative example that is covered. The principle is to add features to the rule. *Specific-to-general* methods search the hypothesis space from the most specific towards the most general hypothesis and will progressively generalize examples. One starts with a positive example, which forms the initial rule for the definition of the concept to be learned. This rule is generalized at the encounter of another positive example that is not covered. The principle is to drop features. The combination of the general-to-specific and the specific-to-general methods is the so-called *version spaces method*, which starts from two hypotheses (Mitchell, 1977). Negative examples specify the most general hypothesis. Positive examples generalize the most specific hypothesis. The version spaces model suffers from practical and computational limitations. To test all possible hypotheses is most of the time impossible given the number of feature combinations.

The most widely used method is *tree learning*. The vectors of the training examples induce classification expressions in the form of a decision tree. A decision tree can be translated in if-then rules to improve the readability of the learned expressions. A decision tree consists of nodes and branches. Each node, except for terminal nodes or leaves, represents a test or decision and branches into subtrees for each possible outcome of the test. The tree can be used to classify an object by starting at the root of the tree and moving through it until a leaf (class of the object) is encountered.

Basic algorithms (e.g., C4.5 of Quinlan, 1993) construct the *trees* in a top-down, greedy way by selecting the most discriminative feature and use it as a test to the root node of the tree. A descendant node is then created for each possible value of this feature, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this feature). The entire process is then repeated using the training examples associated with each descendant node to select the best feature to test at that point in the tree. This forms a greedy search for an acceptable decision tree, in which the algorithm never backtracks to reconsider earlier choices. In this way not all the hypotheses of the search space are tested. Additional mechanisms can be incorporated. For instance, by searching a rule or tree that covers most of the positive examples and removal of these examples from further training, the search space is divided into subspaces, for each of which a covering rule is sought. Other ways for reducing the search space regard preferring simple rules above complex ones and by branching and bounding the search space when the method will not consider a set of hypotheses if there is some criterion that allows assuming that they are inferior to the current best hypothesis. The selection of the most discriminative feature at each node except for a leave node, is often done by selecting the one with the largest *information gain*, i.e., the feature that causes the largest reduction in entropy when the training examples are classified according to the outcome of the test at the node. As seen above, entropy is a measure of uncertainty.

More specifically, given a collection S of training examples, if the classification can take on k different values, then the entropy of S relative to the k classifications is defined as:

$$Entropy(S) \equiv \sum_{i=1}^{k} -p_i \log_2 p_i \tag{5.59}$$

where p_i is the proportion of S belonging to class k. The information gain of a feature f is the expected reduction in entropy caused by partitioning the examples according to this feature.

$$Gain(S,f) \equiv Entropy(S) - \sum_{v \in Values(f)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{5.60}$$

where $Values(f)$ = set of all possible values of feature f
S_v = subset of S for which feature f has value v.

Rule and tree learning algorithms were the first algorithms that have been used in information extraction, and they are still popular learning techniques for information extraction. The factors that play a role in their popularity are their expressive power, which makes them compatible with human-engineered knowledge rules and their easy interaction with other knowledge resources. Because of their greedy nature the algorithms usually perform better when the feature set is limited. Information extraction tasks sometimes naturally can make use of a limited set of features that exhibit some dependencies between the features (e.g., in coreference resolution).

Induction of rules and trees was a very popular information extraction technique in the 1990s. It has been applied among others to information extraction from semi-structured text (Soderland, 1999) and it continues to be a popular and successful technique in coreference resolution (McCarthy and Lehnert 1995; Soon et al., 2001; Ng and Cardie, 2002).

5.7 Relational Learning

When the learned rules are written in a logical formalism, the learning is often referred to as *inductive logic programming* (ILP) (Mooney, 1997). The most simple rules are expressed in propositional logic, but often the learner will also acquire expressions in first-order predicate logic. The classifier learns small programs containing predicates, constants and variables, which can be used to make inferences, hence the term inductive logic programming.

Inductive logic programming is a subcategory of relational learning. Unless rule representation is severely restricted, the learning is often intractable. In order to counter this problem for a specific extraction problem, domain-specific heuristics are implemented. However, we lack generic ILP methods that could be applicable to a variety of information extraction problems. *Relational learning* refers to all techniques that learn structured concept definitions from structured examples. Relational learning is concerned with the classification of patterns whose presence signifies that certain elements of a structure are in a particular relation to one another. The structure of the instances can have different formats (e.g., logical programs, Bayesian networks, graphs). The learning algorithm receives input examples of which the complete structure is classified.

In information extraction relational learning that learns first-order predicates has been implemented for extracting rather structured information such as information in job postings (Califf and Mooney, 1997) and in seminar announcements (Roth and Yih, 2001). In addition, there exist relational models based on statistics. The kernel methods, the hidden Markov models and conditional random fields can be seen as relational learning models. In these cases, the relational model is chosen because the propositional, nominal or ordinal representations might become too large, or could loose much of the inherent domain structure.

Many questions have still to be solved and appropriate algorithms for relational learning should be drafted. Relational learning could offer suitable solutions to recognize information in texts.

5.8 Conclusions

Supervised pattern recognition techniques are very useful in information extraction. Many useful approaches exist. As we will see in Chap. 9 they currently constitute the most successful techniques. However, there is the bottleneck of acquiring sufficient annotated examples. In the next chapter it is shown how unsupervised learning techniques aid in resolving this problem.

Information extraction techniques recognize rather simple patterns that classify information in a particular semantic class. As we will discuss in the final chapter, there is a need for more advanced recognition of content, where meaning is assigned based on a conglomerate of different concepts and their relations found in the unstructured sources.

5.9 Bibliography

Ahn, David, Sisay F. Adafre and Maarten de Rijke (2005). Extracting temporal information from open domain text. In *Proceedings of the 5th Dutch-Belgian Information Retrieval Workshop (DIR'05)*. Twente.

Berger, Adam, Stephen A. Della Pietra and Vincent J. Della Pietra (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, 22 (1), 39-71.

Bikel, Daniel M., Scott Miller, Richard Schwartz and Ralph Weischedel (1997). Nymble: A high-performance learning name-finder. In *Proceedings Fifth Conference on Applied Natural Language Processing* (pp. 194-201). Washington, DC.

Bunescu, Razvan and Raymond J. Mooney (2004). Collective information extraction with relational Markov networks. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics* (pp. 439-446). East Stroudsburgh, PA: ACL.

Califf, Mary E. and Raymond J. Mooney (1997). Relational learning of pattern-matching rules for information extraction. In T.M. Ellison (Ed.), *CoNLL: Computational Natural Language Learning* (pp. 9-15). ACL.

Chieu, H.L. and Ng Hwee Tou (2002). Named entity recognition: A maximum entropy approach using global information. In *COLING 2002. Proceedings of the 19th International Conference on Computational Linguistics* (pp. 190-196). San Francisco: Morgan Kaufmann.

Christianini, Nello and John Shawe-Taylor (2000). *An Introduction to Support Vector Machines and Other Kernel Based Learning Methods.* Cambridge, UK: Cambridge University Press.

Collins, Michael and Nigel Duffy (2001). Convolution kernels for natural language. In Thomas G. Dietterich, Sue Becker and Zoubin Ghahramani (Eds.), *Advances in Neural Information Processing Systems* 14 (pp. 625-632). Cambridge, MA: The MIT Press.

Culotto, Aron and Jeffrey Sorenson (2004). Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics* (pp. 424-430). East Stroudsburg, PA: ACL.

Darroch, J.N. and D. Ratcliff (1972). Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43, 1470-1480.

Dempster, A.P., N.M. Laird and D.B. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal Royal Statistical Society Series B 39*, 1-38.

Della Pietra, Stephen, Vincent Della Pietra and John Lafferty (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19, 380-393.

Fleischman, Michael, Namhee Kwon and Eduard Hovy (2003). A maximum entropy approach to FrameNet tagging. In *Proceedings of the Human Language Technology Conference of the North American Chapter for Computational Linguistics*. East Stroudsburg, PA: ACL.

Isozaki, Hideki and Hideto Kazawa (2002). Efficient support vector classifiers for named entity recognition. In *COLING 2002. Proceedings of the 19th International Conference on Computational Linguistics* (pp. 390-396). San Francisco, CA: Morgan Kaufmann.

Isozaki, Hideki and Tsutomu Hirao (2003). Japanese zero pronoun resolution based on ranking rules and machine learning. In *Proceedings of EMNLP-2003* (pp. 184-191). ACL.

Jaynes, Edwin T. (1982). On the rationale of maximum-entropy models. *Proceedings of the IEEE*, 70 (9), 939-952.

Jordan, Michael I. (1999). *Learning in Graphical Models.* Cambridge, MA: The MIT Press.

Kehler, Andrew (1997). Probabilistic coreference in information extraction. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing* (pp. 163-173). Somerset, NJ: ACL.

Lafferty, John, Andrew McCallum and Fernando C.N. Pereira (2001). Conditional random fields: Probabilistic models for segmenting and labelling sequence data. In *Proceedings of the 18th International Conference on Machine Learning* (pp. 282-289). San Francisco, CA: Morgan Kaufmann.

Leek, Timothy Robert (1997). *Information Extraction using Hidden Markov Models*. Master thesis, University of California San Diego.

Malouf, Robert (2002). A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Conference on Natural Language Learning (CoNLL-2002)* (pp. 49-55). San Francisco, CA: Morgan Kaufmann.

McCallum, Andrew, Kamal Nigam, Jason Rennie and Kristie Seymore (1999). A machine learning approach to building domain-specific search engines. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 662-667). San Mateo, CA: Morgan Kaufmann.

McCallum, Andrew, Andrew Ng and Michael I. Jordan (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In Thomas Dietterich, Suzanna Becker and Zoubin Ghahramani (Eds.), *Advances in Neural Information Processing Systems 14* (pp. 609-616). Cambridge, MA: The MIT Press.

McCallum, Andrew and Wei Li (2003). Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the Seventh Conference on Natural Language Learning* (CoNLL). East Stroudsburg, PA: ACL.

McCarthy, Joseph and Wendy G. Lehnert (1995). Using decision trees for coreference resolution. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1050-1055). San Mateo, CA: Morgan Kaufmann.

Mehay Dennis N., Rik De Busser and Marie-Francine Moens (2005). Labeling generic semantic roles. In Harry Bunt, Jeroen Geertzen and Elias Thyse (Eds.), *Proceedings of the Sixth International Workshop on Computational Semantics* (*IWCS-6*) (pp. 175-187). Tilburg, The Netherlands: Tilburg University.

Minsky, Marvin L. and Seymour A. Papert (1969). *Perceptrons*. The MIT Press.

Mitchell, Tom (1977). Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence* (pp. 305-310). Cambridge, MA: William Kaufmann.

Mitchell, Tom (1997). *Machine Learning*. MacGraw Hill.

Mooney, Raymond (1997). Inductive logic programming for natural language processing. In *Inductive Logic Programming, volume 1314 of LNAI* (pp. 3-24). Berlin: Springer.

Ng, Andrew Y. and Michael Jordan (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naïve Bayes. *Neural Information Processing 2002.*

Ng, Vincent and Claire Cardie (2002). Improving machine learning approaches to coreference resolution. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics* (*ACL-2002*). San Francisco, CA: Morgan Kaufmann.

Quinlan, J. Ross (1993). *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Rabiner, Lawrence L. (1989). A tutorial on hidden Markov models and selected applications. In *Proceedings of the IEEE* 77 (pp. 257-285). Los Alamitos, CA: The IEEE Computer Society.

Ratnaparkhi, Adwait (1998). *Maximum Entropy Models for Natural Language Ambiguity Resolution.* Ph.D. thesis, University of Pennsylvania.

Ray, Soumya and Mark Craven (2001). Representing sentence structure in hidden Markov models for information extraction. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence, Seattle, WA.* San Francisco, CA: Morgan Kaufmann.

Roth, Dan and Wen-tau Yih (2001). Relational learning via propositional algorithms: An information extraction case study. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence* (pp. 1257-1263). San Francisco, CA: Morgan Kaufmann.

Seymore, Kristie, Andrew McCallum and Ronald Rosenfeld (1999). Learning hidden Markov model structure for information extraction. In *Proceedings of the AAAI 99 Workshop on Machine Learning for Information Extraction.*

Shannon, Claude E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27, 379-423, 623-656.

Soderland, Stephen (1999). Learning information extraction rules for semi-structured and free text. *Machine Learning,* 1-3, 233-272.

Soon, Wee Meng, Hwee Tou Ng and Daniel Chung Yong Lim (2001). A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27 (4), 521-544.

Taskar, Ben, Vassil Chatalbashev and Daphne Koller (2004). Learning associative Markov networks. In *Proceedings of the Twenty-First International Conference on Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Theodoridis, Sergios and Konstantinos Koutroumbas (2003). *Pattern Recognition.* Amsterdam, The Netherlands: Academic Press.

Vapnik, Vladimir N. (1988). *Statistical Learning Theory.* New York: John Wiley and Sons.

Wallach, Hanna M. (2004). Conditional random fields: An introduction. University of Pennsylvania CIS Technical Report MS-CIS-04-21.

Zalenko, Dimitry, Chinatsu Aone and Antony Richardella (2003). Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3, 1083-1106.

Zhang, Dell and Wee Sun Lee (2003). Question classification using Support Vector Machines. In *Proceedings of the Twenty-Sixth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 26-31). New York: ACM.

Zhang, Jie, Dan Shen, Guodong Zu, Su Jian and Chew-Lim Tan (2004). Enhancing HMM-based biomedical named entity recognition by studying special phenomena. *Journal of Biomedical Informatics,* 37, 411-422.