The hinge loss (also called the margin loss), which is optimized by the SVM, is a ramp function that has slope -1 when $yf(x) < 1$ and is zero otherwise. Two other loss functions—squared loss and exponential loss—are also shown; these are used in other learning algorithms such as neural networks (Bishop, 1995) and boosting (Schapire, 2003; Lebanon and Lafferty, 2002). Each of these loss functions has different advantages and disadvantages; these are too deep and off-topic to attempt to discuss in the context of this thesis. The interested reader is directed to (Bartlett, Jordan, and McAuliffe, 2005) for more in-depth discussions.

2.2 Structured Prediction

The vast majority of prediction algorithms, such as those described in the previous section, are built to solve prediction problems whose outputs are "simple." Here, "simple" is intended to include binary classification, multiclass classification and regression. (I note in passing that some of the aforementioned algorithms are more easily adapted to multiclass classification and/or regression than others.) In contrast, the problems I am interested in solving are "complex." The family of generic techniques for solving such "complex" problems are generally known as *structured prediction algorithms* or *structured learning algorithms.* To date, there are essentially four state-of-the-art structured prediction algorithms (with minor variations), each of which I briefly describe in this section. However, before describing these algorithms in detail, it is worthwhile to attempt to formalize what is meant by "simple," "complex" and "structure." It turns out that defining these concepts is remarkably difficult.

2.2.1 Defining Structured Prediction

Structured prediction is a very slippery concept. In fact, of all the primary prior work that proposes solutions to the structured prediction problem, none explicitly defines the problem (McCallum, Freitag, and Pereira, 2000; Lafferty, McCallum, and Pereira, 2001; Punyakanok and Roth, 2001; Collins, 2002; Taskar, Guestrin, and Koller, 2003; McAllester, Collins, and Pereira, 2004; Tsochantaridis et al., 2005). In all cases, the problem is explained and motivated purely by means of examples. These examples include the following problems:

- Sequence labeling: given an input sequence, produce a label sequence of equal length. Each label is drawn from a small finite set. This problem is typified in NLP by part-of-speech tagging.
- Parsing: given an input sequence, build a tree whose yield (leaves) are the elements in the sequence and whose structure obeys some grammar. This problem is typified in NLP by syntactic parsing.
- Collective classification: given a graph defined by a set of vertices and edges, produce a labeling of the vertices. This problem is typified by relation learning problems, such as labeling web pages given link information.

- Bipartite matching: given a bipartite graph, find the best possible matching. This problem is typified by (a simplified version of) word alignment in NLP and protein structure prediction in computational biology.

There are many other problems in NLP that do not receive as much attention from the machine learning community, but seem to also fall under the heading of structured prediction. These include entity detection and tracking, automatic document summarization, machine translation and question answering (among others). Generalizing over these examples leads us to a partial definition of structured prediction, which I call Condition 1, below.

Condition 1. *In a structured prediction problem, output elements $y \in \mathcal{Y}$ decompose into variable length vectors over a finite set. That is, there is a finite $M \in \mathbb{N}$ such that each $y \in \mathcal{Y}$ can be identified with at least one vector $v_y \in M^{T_y}$, where T_y is the length of the vector.*

This condition is likely to be deemed acceptable by most researchers who are active in the structured prediction community. However, there is a question as to whether it is a sufficient condition. In particular, it includes many problems that would *not* really be considered structured prediction (binary classification, multitask learning (Caruana, 1997), etc.). This leads to a second condition that hinges on the form of the loss function. It is natural to desire that the loss function does not decompose over the vector representations. After all, if it does decompose over the representation, then one can simply solve the problem by predicting each vector component independently. However, it is always possible to construct *some* vector encoding over which the loss function decomposes[3] This means that we must therefore make this conditions stronger, and require that there is no polynomially sized encoding of the vector over which the loss function decomposes.

Condition 2. *In a structured prediction problem, the loss function does* not *decompose over the vectors v_y for $y \in \mathcal{Y}$. In particular, $l(x, y, \hat{y})$ is* not *invariant under identical permutations of y and $\hat{y}$. Formally, we must make this stronger: there is* no *vector mapping $y \mapsto v_y$ such that the loss function decomposes, for which $|v_y|$ is polynomial in $|y|$.*

Condition 2 successfully excludes problems like binary classification and multitask learning from consideration as structured prediction problems. Importantly, it excludes standard classification problems and multitask learning. Interestingly, it also excludes problems such as sequence labeling under Hamming loss (discussed further in Chapter 4). Hamming loss (per-node loss) on sequence labeling problems *is* invariant over permutations. This condition also excludes collective classification under zero/one loss on the nodes. In fact, it excludes virtually any problem that one could reasonably hope to solve by using a collection of independent classifiers (Punyakanok and Roth, 2001).

[3]To do so, we encode the true vector in a very long vector by specifying the exact location of each label using products of prime numbers. Specifically, for each label k, one considers the positions $i_1, \ldots, i_Z$ in which k appears in the vector. The encoded vector will contain $p_1^{i_1} p_2^{i_2} \cdots p_Z^{i_Z}$ copies of element k, where $p_1, \ldots$ is an enumeration of the primes. Given this encoding it is always possible to reconstruct the original vector, yet the loss function will decompose.

The important aspect of Condition 2 is that it hinges on the notion of the loss function rather than the features. For instance, one can argue that even when sequence labeling is performed under Hamming loss, there is still important structural information. That is, we "know" that by including structural *features* (such as Markov features), we can solve most sequence labeling tasks better.[4] The difference between these two perspectives is that under Condition 2 the *loss* dictates the structure, while otherwise the *features* dictate the structure. Since when the world hands us a problem to solve, it hands us the loss but not the features (the features are part of the *solution*), it is most appropriate to define the structured prediction *problem* only in terms of the loss.

Current generic structured prediction algorithms are not built to solve problems under which Condition 2 holds. In order to facilitate discussion, I will refer to problems for which both conditions hold as "structured prediction problem" and those for which only Condition 1 holds as "decomposable structured prediction problems." I note in passing that this terminology is nonstandard.

2.2.2 Feature Spaces for Structured Prediction

Structured prediction algorithms make use of an extended notion of feature function. For structured prediction, the feature function takes as input *both* the original input $x \in \mathcal{X}$ and a hypothesized output $y \in \mathcal{Y}$. The value $\Phi(x, y)$ will again be a vector in Euclidean space, but which now depends on the output. In particular, in part of speech tagging, an element in $\Phi(x, y)$ might be the number of times the word "the" appears and is labeled as a determiner and the next word is labeled as a noun.

All structured prediction algorithms described in this Chapter are only applicable when Φ admits efficient search. In particular, after learning a weight vector $\boldsymbol{w}$, one will need to find the best output for a given input. This is the "argmax problem" defined in Eq (2.11).

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \boldsymbol{w}^\top \Phi(x, y) \tag{2.11}$$

This problem will not be tractable in the general case. However, for very specific $\mathcal{Y}$ and very specific Φ, one can employ dynamic programming algorithms or integer programming algorithms to find efficient solutions. In particular, if Φ decomposes over the vector representation of $\mathcal{Y}$ such that no feature depends on elements of y that are more than k positions away, then the Viterbi algorithm can be used to solve the argmax problem in time $\mathcal{O}(M^k)$ (where M is the number of possible labels, formally from Condition 1). This case includes standard sequence labeling problems under the Markov assumption as well as parsing problems under the context-free assumption.

2.2.3 Structured Perceptron

The structured perceptron is an extension of the standard perceptron (Section 2.1.1) to structured prediction (Collins, 2002). Importantly, it is only applicable to the problem

[4] This is actually not necessarily the case; see Section 4.2 for an extended discussion.

Algorithm AVERAGEDSTRUCTUREDPERCEPTRON($x_{1:N}, y_{1:N}, I$)

1: $\boldsymbol{w}_0 \leftarrow \langle 0, \ldots, 0 \rangle$
2: $\boldsymbol{w}_a \leftarrow \langle 0, \ldots, 0 \rangle$
3: $c \leftarrow 1$
4: **for** $i = 1 \ldots I$ **do**
5: **for** $n = 1 \ldots N$ **do**
6: $\hat{y}_n \leftarrow \arg\max_{y \in \mathcal{Y}} \boldsymbol{w}_0{}^\top \Phi(x_n, y_n)$
7: **if** $y_n \neq \hat{y}_n$ **then**
8: $\boldsymbol{w}_0 \leftarrow \boldsymbol{w}_0 + \Phi(x_n, y_n) - \Phi(x_n, \hat{y}_n)$
9: $\boldsymbol{w}_a \leftarrow \boldsymbol{w}_a + c\Phi(x_n, y_n) - c\Phi(x_n, \hat{y}_n)$
10: **end if**
11: $c \leftarrow c + 1$
12: **end for**
13: **end for**
14: **return** $\boldsymbol{w}_0 - \boldsymbol{w}_a / c$

Figure 2.3: The averaged structured perceptron learning algorithm.

of 0/1 loss over $\mathcal{Y}$: that is, $l(x, y, \hat{y}) = \mathbf{1}(y \neq \hat{y})$. As such, it only solves decomposable structured prediction problems (0/1 loss is trivially invariant under permutations). Like all the algorithms we consider, the structured perceptron will be parameterized by a weight vector $\boldsymbol{w}$. The structured perceptron makes one significant assumption: that Eq (2.11) can be solved efficiently.

Based on the argmax assumption, the structured perceptron constructs the perceptron in nearly an identical manner as for the binary case. While looping through the training data, whenever the predicted $\hat{y}_n$ for x_n differs from y_n, we update the weights according to Eq (2.12).

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Phi(x_n, y_n) - \Phi(x_n, \hat{y}_n) \tag{2.12}$$

This weight update serves to bring the vector closer to the true output and further from the incorrect output. As in the standard perceptron, this often leads to a learned model that generalizes poorly. As before, one solution to this problem is weight averaging. This behaves identically to the averaged binary perceptron and the full training algorithm is depicted in Figure 2.3.

The behavior of the structured perceptron and the standard perceptron are virtually identically. The major changes are as follow. First, there is no bias b. For structured problems, a bias is irrelevant: it will increase the score of all hypothetical outputs by the same amount. The next major difference is in step (6): the best scoring output $\hat{y}_n$ for the input x_n is computed using the arg max. After checking for an error, the weights are updated, according to Eq (2.12), in steps (8) and (9).

2.2.4 Incremental Perceptron

The *incremental perceptron* (Collins and Roark, 2004) is a variant on the structured perceptron that deals with the issue that the arg max in step 6 may not be analytically available. The idea of the incremental perceptron (which I build on significantly in Chapter 3) is to replace the arg max with a beam search algorithm. Thus, step 6 becomes "$\hat{y}_n \leftarrow \text{BeamSearch}(x_n, \boldsymbol{w}_0)$". The key observation is that it is often possible to detect in the process of executing search whether it is possible for the resulting output to ever be correct. For instance, in sequence labeling, as soon as the beam search algorithm has made an error, we can detect it without completing the search (for standard loss function and search algorithms). The incremental perceptron *aborts* the search algorithm as soon as it has detected that an error has been made. Empirical results in the parsing domain have shown that this simple modification leads to much faster convergence and superior results.

2.2.5 Maximum Entropy Markov Models

The maximum entropy Markov model (MEMM) framework, pioneered by McCallum, Freitag, and Pereira (2000) is a straightforward application of maximum entropy models (aka logistic regression models, see Section 2.1.2) to sequence labeling problems. For those familiar with the hidden Markov model framework, MEMMs can be seen as HMMs where the conditional "observation given state" probabilities are replaced with direct "state given observation" probabilities (this leads to the ability to include large numbers of overlapping, non-independent features). In particular, a first-order MEMM places the conditional distribution shown in Eq (2.13) on the nth label, y_n, given the full input x, the previous label, y_{n-1}, a feature function Φ and a weight vector $\boldsymbol{w}$.

$$p\left(y_n \mid x, y_{n-1}; \boldsymbol{w}\right) = \frac{1}{Z_{x,y_{n-1};\boldsymbol{w}}} \exp\left[\boldsymbol{w}^\top \Phi(x, y_n, y_{n-1})\right] \tag{2.13}$$
$$Z_{x,y_{n-1};\boldsymbol{w}} = \sum_{y' \in \mathcal{Y}^n} \exp\left[\boldsymbol{w}^\top \Phi(x, y', y_{n-1})\right]$$

The MEMM is trained by tracing along the true output sequences for the training data and using the *true* y_{n-1} to generate training examples. This process simply produces multiclass classification examples, equal in number to the number of labels in all of the training data. Based on this data, the weight vector $\boldsymbol{w}$ is learned exactly as in standard maximum entropy models.

At prediction time, one applies the Viterbi algorithm, as in the case of the structured perceptron, to solve the "arg max" problem. Importantly, since the true values for y_{n-1} are not known, one uses the *predicted* values of y_{n-1} for making the prediction about the nth value (albeit, in the context of Viterbi search). As I will discuss in depth in Section 3.4.5, this fact can lead to severely suboptimal results.

2.2.6 Conditional Random Fields

While successful in many practical examples, maximum entropy Markov models suffer from two severe problems: the "label-bias problem" (both Lafferty, McCallum, and Pereira (2001) and Bottou (1991) discuss the label-bias problem in depth) and a limitation to sequence labeling. Conditional random fields are an alternative extension of logistic regression (maximum entropy models) to structured outputs (Lafferty, McCallum, and Pereira, 2001). Similar to the structured perceptron, a conditional random field does not employ a loss function. It optimizes a log-loss approximation to the 0/1 loss over the entire output. In this sense, it is also a solution only to a decomposable structured prediction problems.

The actual formulation of conditional random fields is identical to that for multiclass maximum entropy models. The CRF assumes a feature function $\Phi(x, y)$ that maps input/output pairs to vectors in Euclidean space, and uses a Gibbs distribution parameterized by $\boldsymbol{w}$ to model the probability, Eq (2.14).

$$p\left(y \mid x; \boldsymbol{w}\right) = \frac{1}{Z_{x;\boldsymbol{w}}} \exp\left[\boldsymbol{w}^\top \Phi(x, y)\right] \tag{2.14}$$

$$Z_{x;\boldsymbol{w}} = \sum_{y' \in \mathcal{Y}} \exp\left[\boldsymbol{w}^\top \Phi(x, y')\right] \tag{2.15}$$

Here, $Z_{x,\boldsymbol{w}}$ (known as the "partition function") is the sum of responses of all *incorrect* outputs. Typically, this set will be too large to sum over explicitly. However, if Φ is chosen properly and if $\mathcal{Y}$ is a simple linear-chain structure, this sum can be computed using dynamic programming techniques (Lafferty, McCallum, and Pereira, 2001; Sha and Pereira, 2002). In particular, Φ must be chosen to obey the Markov property: for a Markov length of l, no feature can depend on elements of y that are more than l positions apart. The algorithm associated with the sum is nearly identical to the forward-backward algorithm for hidden Markov models (Baum and Petrie, 1966) and scales as $\mathcal{O}(NK^l)$, where N is the length of the sequence, K is the number of labels and l is the "Markov order" used by Φ.

Just as in maximum entropy models, the weights $\boldsymbol{w}$ are regularized by a Gaussian prior and the log posterior distribution over weights is as in Eq (2.16).

$$\log p\left(\boldsymbol{w} \mid D; \sigma^2\right) = -\frac{1}{\sigma^2} ||\boldsymbol{w}||^2 + \sum_{n=1}^{N} \left[\boldsymbol{w}^\top \Phi(x_n, y_n) - \log \sum_{y' \in \mathcal{Y}} \exp\left[\boldsymbol{w}^\top \Phi(x_n, y')\right]\right] \tag{2.16}$$

Finding optimal weights can be solved either using iterative scaling methods (Lafferty, McCallum, and Pereira, 2001) or more complex optimization strategies such as BFGS (Sha and Pereira, 2002; Daumé III, 2004b) or stochastic meta-descent (Schraudolph and Graepel, 2003; Vishwanathan et al., 2006). In practice, the latter two are much more efficient. In practice, in order for full CRF training to be practical, we must be able to efficiently compute both the arg max from Eq (2.11) and the log normalization constant from Eq (2.17).

$$\log Z_{x;\boldsymbol{w}} = \log \sum_{y' \in \mathcal{Y}} \exp \left[\boldsymbol{w}^\top \Phi(x, y') \right] \tag{2.17}$$

So long as we can compute these two quantities, CRFs are a reasonable choice for solving the decomposable structured prediction problem under the log-loss approximation to 0/1 loss over $\mathcal{Y}$. See (Sutton and McCallum, 2006) and (Wallach, 2004) for in-depth introductions to conditional random fields.

2.2.7 Maximum Margin Markov Networks

The Maximum Margin Markov Network (M^3N) formalism considers the structured prediction problem as a quadratic programming problem (Taskar, Guestrin, and Koller, 2003; Taskar et al., 2005), following the formalism for the support vector machine for binary classification. Recall from Section 2.1.3 that the SVM formulation sought a weight vector with small norm (for good generalization) and which achieved a margin of at least one on all training examples (modulo the slack variables). The M^3N formalism extends this to structured outputs under a given loss function l by requiring that the *difference* in score between the true output y and any incorrect output $\hat{y}$ is at least the loss $l(x, y, \hat{y})$ (modulo slack variables). That is: the M^3N framework scales the *margin* to be proportional to the loss. This is given formally in Eq (2.18).

$$\begin{aligned} \text{minimize}_{\boldsymbol{w}} \quad & \frac{1}{2} ||\boldsymbol{w}||^2 + C \sum_{n=1}^{N} \sum_{\hat{y}} \xi_{n,\hat{y}} & \\ \text{subject to} \quad & \boldsymbol{w}^\top \Phi(x_n, y_n) - \boldsymbol{w}^\top \Phi(x_n, \hat{y}) \geq l(x_n, y_n, \hat{y}) - \xi_{n,\hat{y}} & \forall n, \forall \hat{y} \in \mathcal{Y} \\ & \xi_{n,\hat{y}} \geq 0 & \forall n, \forall y' \in \mathcal{Y} \end{aligned} \tag{2.18}$$

One immediate observation about the M^3N formulation is that there are too many constraints. That is, the first set of constraints is instantiated for every training instance n and for every incorrect output $\hat{y}$. Fortunately, under restrictions on $\mathcal{Y}$ and Φ, it is possible to replace this exponential number of constraints with a polynomial number. In particular, for the special case of sequence labeling under Hamming loss (a decomposable structured prediction problem), one needs only one constraint per element in an example.

In the original development of the M^3N formalism (Taskar, Guestrin, and Koller, 2003), this optimization problem was solved using an active set formulation similar to the SMO algorithm (Platt, 1999). Subsequently, more efficient optimization techniques have been proposed, including ones based on the exponentiated gradient method (Bartlett et al., 2004), the dual extra-gradient method (Taskar et al., 2005) and the sub-gradient method (Bagnell, Ratliff, and Zinkevich, 2006). Of these, the last two appear to be the most efficient. In order to employ these methods in practice, one must be able to compute both the arg max from Eq (2.11) as well as a so-called "loss-augmented search" problem given in Eq (2.19).

$$S(x, y) = \arg\max_{\hat{y} \in \mathcal{Y}} \boldsymbol{w}^\top \Phi(x, \hat{y}) + l(x, y, \hat{y}) \tag{2.19}$$

In order for this to be efficiently computable, the loss function is forced to decompose over the structure. This implies that M^{3}Ns are only (efficiently) applicable to decomposable structured prediction problems. Nevertheless, they are applicable to a strictly wider set of problems than CRFs for two reasons. First, M^{3}Ns do not have a requirement that the log normalization constant (Eq (2.17)) be efficiently computable. This alone allows optimization in M^{3}Ns for problems that would be F#P-complete for CRFs (Taskar et al., 2005). Second, M^{3}Ns can be applied to loss functions other than 0/1 loss over the entire sequence. However, in practice, they are essentially only applicable to a hinge-loss approximation to Hamming loss over $\mathcal{Y}$.

2.2.8 SVMs for Interdependent and Structured Outputs

The Support Vector Machines for Interdependent and Structured Outputs (SVM$^{\text{struct}}$) formalism (Tsochantaridis et al., 2005) is strikingly similar to the M^{3}N formalism. The difference lies in the fact that the M^{3}N framework scales the *margin* by the loss, while the SVM$^{\text{struct}}$ formalism scales the *slack variables* by the loss. The quadratic programming problem for the SVM$^{\text{struct}}$ is given as:

$$\begin{aligned}
\text{minimize}_{\boldsymbol{w}} \quad & \frac{1}{2}\,||\boldsymbol{w}||^2 + C\sum_n \sum_{\hat{y}} \xi_{n,\hat{y}} && (2.20)\\
\text{subject to} \quad & \boldsymbol{w}^\top \Phi(x_n, y_n) - \boldsymbol{w}^\top \Phi(x_n, y') \geq 1 - \frac{\xi_{n,y'}}{l(x_n, y_n, y')} && \forall n, \forall y' \in \mathcal{Y}\\
& \xi_{n,y'} \geq 0 && \forall n, \forall y' \in \mathcal{Y}
\end{aligned}$$

The objective function is the same in both cases; the only difference is found in the first constraint. Dividing the slack variable by the corresponding loss is akin to multiplying the slack variables in the objective function by the loss (in the division, we assume $0/0 = 0$). Though, to date, the SVM$^{\text{struct}}$ framework has generated less interest than the M^{3}N framework, the formalism seems more appropriate. It is much more intuitive to scale the training error (slack variables) by the loss, rather than to scale the margin by the loss. This advantage is also claimed by the original creators of the SVM$^{\text{struct}}$ framework, in which they suggest that their formalism is superior to the M^{3}N formalism because the latter will cause the system to work very hard to separate very lossful hypotheses, even if they are not at all confusable for the truth.

In addition to the difference in loss-scaling, the optimization techniques employed by the two techniques differ significantly. In particular, the decomposition of the loss function that enabled us to remove the exponentially many constraints does not work in the SVM$^{\text{struct}}$ framework. Instead, Tsochantaridis et al. (2005) advocate an iterative optimization procedure, in which constraints are added in an "as needed" basis. It can be shown that this will converge to a solution within ϵ of the optimal in a polynomial number of steps.

The primary disadvantage to the SVM$^{\text{struct}}$ framework is that it is often difficult to optimize. However, unlike the other three frameworks described thus far, the SVM$^{\text{struct}}$ does *not* assume that the loss function decomposes over the structure. However, in exchange

for this generality, the loss-augmented search problem for them SVM$^{\text{struct}}$ framework becomes more difficult. In particular, while the M^{3}N loss-augmented search (Eq (2.19)) assumes decomposition in order to remain tractable, the loss-augmented search (Eq (2.21)) for the SVM$^{\text{struct}}$ framework is often never tractable.

$$S(x, y) = \arg\max_{\hat{y} \in \mathcal{Y}} \left[\boldsymbol{w}^\top \Phi(x, \hat{y}) \right] l(x, y, \hat{y}) \tag{2.21}$$

The difference between the two requirements is that in the M^{3}N case, the loss appears as an additive term, while in the SVM$^{\text{struct}}$ case, the loss appears as a multiplicative term. In practice, for many problems, this renders the search problem intractable.

2.2.9 Reranking

Reranking is an increasingly popular technique for solving complex natural language processing problems. The motivation behind reranking is the following. We have access to a method for solving a problem, but it is difficult or impossible to modify this method to include features we want or to optimize the loss function we want. Assuming that this method can produce a "n-best" list of outputs (instead of just outputting what it thinks is the *single* best output, it produces many best outputs), we can attempt to build a *second* model for picking an output from this n-best list. Since we are only ever considering a constant-sized list, we can incorporate features that would otherwise render the argmax problem intractable. Moreover, we can often optimize a reranker to a loss function closer to the one we care about (in fact, we can do so using techniques described in Section 2.3). Based on these advantages, reranking has been applied in a variety of NLP problems including parsing (Collins, 2000; Charniak and Johnson, 2005), machine translation (Och, 2003; Shen, Sarkar, and Och, 2004), question answering (Ravichandran, Hovy, and Och, 2003), semantic role labeling (Toutanova, Haghighi, and Manning, 2005), and other tasks. In fact, according to the ACL anthology[5], in 2005 there were 33 papers that include the term "reranking," compared to ten in 2003 and virtually none before 2000.

Reranking is an attractive technique because it enables one to quickly experiment with new features and new loss functions. There are, however, several drawbacks to the approach. Some of these are enumerated below:

1. Close ties to original model. In order to rerank, one must have a model whose output can be reranked. The best the reranking model can do is limited by the original model: if it cannot find the best output in an n-best list, then neither will the reranker. This is especially concerning for problems with enormous $\mathcal{Y}$, such as machine translation.

2. Segmentation of training data. One should typically not train a reranker over data that the original model was trained on. This means that one must set aside a held-out data set for training the reranker, leading to less data on which one can train the original model.

[5] `http://acl.ldc.upenn.edu`