A comparative study of structured prediction methods for sequence labeling

Autor: Aitor Palacios Cuesta

Director: Michael Minock

ESTOCOLMO, MAYO DE 2016
Resumen

Algunas tareas de aprendizaje automático tienen un resultado complejo, en lugar de un número real o una clase. Esos resultados están compuestos de elementos que tienen interdependencias y propiedades estructurales. Los métodos que tienen en cuenta la forma del resultado se conocen como técnicas de predicción estructurada. Este trabajo se centra en esas técnicas, evaluando su rendimiento sobre tareas de etiquetado de secuencias y comparándolas. En concreto, tareas pertenecientes al procesado del lenguaje natural son usadas como referencia.

El principal problema evaluado es el etiquetado gramatical. Conjuntos de datos de diferentes idiomas (inglés, español, portugués y holandés) y entornos (periódicos, twitter y chats) son usados para lograr un análisis general. También son examinadas las tareas de análisis sintáctico superficial y el reconocimiento de nombres de entidades. Los algoritmos usados son el perceptrón estructurado, campos aleatorios condicionales, máquinas de vectores de soporte estructuradas y modelos ocultos de Markov con trigramas. Esos algoritmos también son comparados con otros enfoques para resolver esos problemas.

Los resultados muestran que, en general, el perceptrón estructurado tiene el mejor rendimiento para el etiquetado de secuencias con las condiciones evaluadas. Sin embargo, con datos de entrenamiento más escasos, las máquinas de vectores de soporte estructuradas pueden lograr un rendimiento similar o superior. Además, los resultados para los campos aleatorios condicionales son cercanos a esos dos métodos. Los resultados relativos entre los algoritmos son similares para los diferentes conjuntos de datos, pero la precisión absoluta depende de sus particularidades.
Some machine learning tasks have a complex output, rather than a real number or a class. Those outputs are composed by elements which have interdependences and structural properties. Methods which take into account the form of the output are known as structured prediction techniques. This study focuses on those techniques, evaluating their performance for tasks of sequence labeling and comparing them. Specifically, tasks of natural language processing are used as benchmarks.

The principal problem evaluated is part-of-speech tagging. Datasets of different languages (English, Spanish, Portuguese and Dutch) and environments (newspapers, twitter and chats) are used for a general analysis. Shallow parsing and named entity recognition are also examined. The algorithms treated are structured perceptron, conditional random fields, structured support vector machines and trigram hidden Markov models. They are also compared to different approaches to solve these problems.

The results show that, in general, structured perceptron has the best performance for sequence labeling with the conditions evaluated. However, with few training examples, structured support vector machines can achieve a similar or superior accuracy. Moreover, the results for conditional ranom fields is near those two methods. The relative results of the algorithms are similar across different datasets, but the absolute accuracies are dependent on their specificities.
List of Figures

2.1 Bias-variance trade-off ........................................ 10
2.2 Comparison of conditional random fields .................. 12

4.1 Average token accuracy for the Brown corpus .............. 21
4.2 Average sentence accuracy for the Brown corpus ........... 21
4.3 Average token accuracy for the Spanish corpus ............. 22
4.4 Average sentence accuracy for the Spanish corpus .......... 22
4.5 Average token accuracy for the Dutch corpus ............... 22
4.6 Average sentence accuracy for the Dutch corpus .......... 22
4.7 Average token accuracy for the Floresta treebank ......... 23
4.8 Average sentence accuracy for the Floresta treebank ....... 23
4.9 Average token accuracy for the twitter corpus .......... 23
4.10 Average sentence accuracy for the twitter corpus ........ 23
4.11 Average token accuracy for the chat corpus ............... 24
4.12 Average sentence accuracy for the chat corpus ............ 24
4.13 Relative average computing time on the Brown corpus .... 24
4.14 Improvements on unigram with the Brill tagger for the Spanish and chat datasets 24
4.15 Token accuracy for a single iteration on the Penn Treebank 25
4.16 Average token accuracy for the Spanish NER corpus ...... 26
4.17 Average sentence accuracy for the Spanish NER corpus ... 26
4.18 Average token accuracy for the Dutch NER corpus ...... 26
4.19 Average sentence accuracy for the Dutch NER corpus ..... 26
4.20 Average token accuracy for the CoNLL-2000 chunk corpus 27
4.21 Average sentence accuracy for the CoNLL-2000 chunk corpus 27
# Contents

List of Figures ......................................................................................................................... 1

1 Introduction ......................................................................................................................... 4
   1.1 Problem statement ........................................................................................................... 5
   1.2 Organization of the report ............................................................................................... 5

2 Background .......................................................................................................................... 6
   2.1 Sequence labeling .......................................................................................................... 6
      2.1.1 Part-of-speech tagging .............................................................................................. 6
      2.1.2 Shallow parsing ......................................................................................................... 8
      2.1.3 Named entity recognition ......................................................................................... 8
   2.2 Structured prediction algorithms .................................................................................. 9
      2.2.1 Features .................................................................................................................. 10
      2.2.2 Structured perceptron ............................................................................................ 10
      2.2.3 Conditional random fields ..................................................................................... 11
      2.2.4 Structured support vector machines ....................................................................... 13
      2.2.5 Trigram TnT ......................................................................................................... 13
   2.3 Other approaches for sequence labeling ....................................................................... 14

3 Method .................................................................................................................................. 16
   3.1 Data ............................................................................................................................... 17
      3.1.1 Brown corpus ........................................................................................................... 17
      3.1.2 Spanish CoNLL-2002 corpus ................................................................................. 17
      3.1.3 Dutch CoNLL-2002 corpus .................................................................................... 18
      3.1.4 Floresta treebank .................................................................................................... 18
      3.1.5 Twitter bootstrapped dataset ................................................................................... 18
      3.1.6 NPS Chat corpus .................................................................................................... 18
      3.1.7 CoNLL-2000 corpus ............................................................................................... 18
   3.2 Features .......................................................................................................................... 19
   3.3 Performance measures .................................................................................................... 20

4 Results ................................................................................................................................. 21
   4.1 Part of speech tagging ................................................................................................. 21
   4.2 Shallow parsing and named entity recognition ............................................................ 25
5 Discussion
  5.1 Comparison of algorithms ..................................... 28
  5.2 Dataset performance ........................................... 29

6 Conclusions .......................... 31
  6.1 Future work .............................................. 31

7 Bibliography .................................. 33
1 Introduction

In the field of machine learning, from an abstract point of view, the objective is to learn a mapping from inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. This task can be based on examples of correct, or at least accurate if dealing with noise, pairs $(x, y)$ (supervised learning); or only based on inputs $x$ (unsupervised learning). Commonly, inputs have a form of a feature vector, a n-dimensional vector with numerical values. This is a convenient representation for many problems and allows for an easier numerical function to be predicted. For the majority of problems and algorithms, the output space of these functions is $\mathbb{R}$ (regression) or a finite subset of $\mathbb{N}$ (classification). However, some tasks could benefit from learning maps to an output space with a complex structure. Examples of these structures can be sentences, images, or graphs. It could be possible, and sometimes it is done in practice (e.g. naive Bayes classifier [34]), to treat these structures as a set of elements which can be predicted independently by means of usual functions with real or natural output spaces. Following the previous example, we could predict independently letters or pixels. However, disregarding the underlying structure can lead to a wrong or weak performance, because it ignores the dependencies and relationships that those elements have. Therefore, a more promising approach is to learn directly that structure, taking into account its specific properties.

There are different areas where this approach can be interesting. In computer vision, an image can be seen as a graph where pixels are nodes and edges represent certain properties, such as distances or difference in color values. Some examples of particular problems where structured prediction is suitable include image segmentation [5], human pose and gesture estimation [25] or object tracking [42]. Also, considering the structure of a word can be important when recognizing handwritten characters for a more accurate prediction (in English, “hello” is more likely to be written by a human than “hgllo”). The same idea applies to speech recognition. In computational biology, predicting the structure of a protein is inherently a structured prediction problem. Moreover, in natural language processing (NLP), structures are present in many typical problems [37] [15]. In part-of-speech tagging, the goal is to determine the category of a word in a sentence. Therefore, the output structure is a sequence of the same length as the input. Named-entity recognition tries to find and label strings referring to people, locations or organizations, which can be viewed as the same sequential structure. In document summarization, the function to learn maps a document to a shorter document. A parser transforms a sentence into an labeled tree. Machine translation, sentiment analysis and semantic representation also involve structures.

Some of these problems have been treated using different methods, such as deterministic
linguistic algorithms for NLP and feature-based matching for computer vision. However
the machine learning approach, especially with structured prediction methods, has shown
very promising recently and state-of-the-art results come from this field, either alone or in
combination with rule-based approaches. Therefore, it can be relevant to analyze which of
these structured prediction techniques is the best option and under which conditions.

In this document, the field of study will be supervised structured prediction. Specifically,
the evaluation of the algorithms will be done on tasks which objective is to learn a sequence.
The focus will be on part-of-speech tagging (POST), a classical and well studied problem in
NLP. Different algorithms for structure prediction will be examined with various datasets
that are not usually involved in these type of experiments, including variation in language
and environment. Named entity recognition (NER) and shallow parsing (or chunking) will
be also evaluated.

1.1 Problem statement

The goal of this study is to analyze and compare the performance of different structured
prediction algorithms for sequence labeling problems using a variety of datasets. Some
relevant questions that will be attempted to answer are:

• Which is, in general, the best algorithm for sequence labeling and under which condi-
tions?

• What is the influence of the characteristics of a particular dataset in the accuracy of
  the methods?

• What are the theoretical reasons that may have caused these results?

1.2 Organization of the report

The remaining document is structured as follows: Section 2 presents the sequence labeling
problems, structured prediction details and algorithms necessary to understand this study.
In section 3 it is detailed how the experiments are performed. Section 4 presents the results
obtained, and in section 5 these are discussed and compared. Finally, conclusions and pos-
sible new lines of study are suggested in section 6.
2 Background

This section introduces common notions relative to the tasks studied in the experiments, as well as it presents structured prediction concepts and algorithms that will be evaluated.

2.1 Sequence labeling

The tasks examined are part of a more general set of problems of sequence labeling. This set also includes optical character recognition or genome analysis and segmentation. It is a common test field for structured prediction algorithms, because it tries to learn sequences, possibly the simplest yet non-trivial structure. In a machine learning context, the abstract map we try to determine is \( f : \mathcal{X} \to \mathcal{Y} \), where \( \mathcal{X} \) is a sequence of variable length, and \( \mathcal{Y} \) is a sequence of labels. It holds that \( |x| \in \mathcal{X} = |y| \in \mathcal{Y} \), that is, sequences of both input and output spaces have the same length, as every position in the input sequence is labeled.

2.1.1 Part-of-speech tagging

In POST, \( \mathcal{X} \) is the set of sentences of a language, and \( \mathcal{Y} \) is the set of all possible sequence of grammatical tags. It is clear that \( |\mathcal{Y}| << |\mathcal{X}| \), because grammatical tags are abstractions of words according to their function in a sentence. This task is usually taught in schools, where in English and other Indo-European languages commonly 9 parts-of-speech are identified: noun, verb, article, adjective, preposition, pronoun, adverb, conjunction and intersection. However, in computational linguistics, the number of tags can be considerably larger, due to the labels having more specific categories than those (e.g., past tense verb, personal pronoun or comparative adverb). Typically, the cardinality of the tags used in popular datasets is bigger than 30 [35] [20]. Therefore, even though it is much smaller than the set of inputs as commented before, the set of outputs is very large: the number of possible output sequences for a reasonable input sentence of, for example, 20 words is bigger than \( 20^{30} > 10^{39} \).

A naive attempt to tackle this problem would be to treat it as a simple one-to-one correspondence between words and labels. Nevertheless, the same word can have a variable number of tags associated depending on the context. For example, the word “exciting” can be either a verb or an adjective. However, if it appears in the sentence “This competition is very exciting.”, it is fairly clear that its tag should be an adjective. Another preliminary attempt without using the structure of the output could be assigning to each word the most
common tag associated to that word in the specific dataset. This again ignores the possibility of a same word having different grammatical roles in the same text, and leads to a worse performance than methods which take into account the context. There are certain dependencies in the output space that are crucial to represent or learn. For example, an article can be followed by a substantive or an adjective, but it cannot be followed by a preposition or a verb. Moreover, to resolve ambiguities, sometimes it is also necessary to take into account words which are not the previous one we want to label. For example, in “I have a plastic bag”, we can determine that “plastic” is an adjective and not a substantive only if we consider the word “bag” coming after it. It is also important to notice that some ambiguities can require to consider words which are not neighbors of the one we want to tag. As an example, in “Sometimes one can think it is a good idea”, the function of the word “one” is undetermined unless we consider the word “think”. Instead of a substantive, it could be a cardinal adjective (e.g., in “Sometimes one can of meat is enough for lunch”). A wider range of problematic cases for tags used in the Penn Treebank Project dataset is described by Santorini (1991) [35].

Within structured prediction, unsupervised methods have also been used for this problem. However, as it can be expected, these have shown to have a worse performance than those supervised [4]; and might be more useful for datasets with partial information or to generate features for supervised methods.

Even though POST is a well studied area and it might not seem hard to solve compared to other NLP problems due to the simplicity of its structure, it is not yet a solved task. The best performances for English language by machine learning algorithms have an accuracy of above 97% per-word label success [40] [36] for specific datasets. Nonetheless, statistical methods that assign the most common label to each word can have an accuracy greater than 90% [27]. This can be explained because ambiguous cases as the ones presented above represent a small proportion of the total number of words present in a common document. Furthermore, if measured in per-sentence success, where a sentence tagging is only successful if every element is correct, the performance drops to less than 60% [40]. Also, the performance for other languages and domains that differ from news and academic documents is not as good [24]. Additionally, the specificity of the tagset can influence the accuracy a method achieves. The impact of these conditions will be analyzed in this study. It is suggested (e.g., Christopher D. Manning [27]) that additional progress might be hard to achieve only by improving algorithms from the machine learning perspective, and a linguistic analysis might be necessary. Nevertheless, state-of-the-art results come from structured prediction algorithms and it is a relatively new approach, so it is not unreasonable to expect new methods to produce further improvements on these results.

This problem is not only interesting by itself, but also serves as a fundamental base for more abstract NLP problems, such as word-sense disambiguation or syntactic parsing; which in turn are used for more abstract tasks such as sentiment analysis or machine translation. The two other problems treated in this study also use part-of-speech tags as an input. Therefore, improving the performance in this task might mean an improvement in those other areas.
2.1.2 Shallow parsing

Shallow parsing or chunking is the task of identifying word groups (or chunks) and can be seen as an intermediate step between part-of-speech tagging and parsing, as it identifies basic group of words but does not specify its relative role in the sentence. Commonly, part-of-speech tags are used as an input along with the sentences. The number of chunking tags are usually fewer than the ones of POST, so it also holds that $|\mathcal{Y}| << |\mathcal{X}|$. These tags are, for example “noun phrase” (e.g. “my great friend”), “verb phrase” (e.g. “wants to swim”) or “prepositional phrase” (e.g. “from now on”).

Even though identifying and labeling group of words might seem a different task, it can be transformed to sequence labeling by using two kind of labels for each chunking tag: one for the word which is the beginning of the chunk (e.g., “B-NP” for the beginning of a noun phrase) and one for the rest of words within the chunk (e.g., “I-NP”). Also, in the case of words outside any chunk, an additional tag (e.g., “O”) can be introduced. With this labeling, known as “IOB notation”, a bijection is determined between chunks and individual tags. Therefore it can be treated as a sequence labeling problem.

The same naive attempts as in the previous case can be used, but their problems are very similar, and the need of taking into account the structure of the output is even more clear. Intuitively, when the previous tag is the beginning of a chunk, the decision has very different conditions than when it is not. Also, the dependency between labels and near words in POST is similar to the dependency between chunking tags and near POST tags and words.

Similarly to the case of part-of-speech tagging, chunking is performed as a previous step for higher order other tasks such as machine translation or speech generation. It is also a closer step towards syntactic parsing. Therefore, it is an important problem in natural language processing that is interesting to solve.

2.1.3 Named entity recognition

In named entity recognition (NER) the goal is also to identify chunks in sentences. However, these chunks are very specific because they are only groups corresponding to direct mentions of entities (e.g., a person or a location), and they specify which kind of entity they are. As in shallow parsing, IOB notation can be used to transform it to a sequence labeling task. However, in this case, many “outside any chunk” labels are present; and the difficulty increases because it is necessary to identify noun groups, decide if they are direct references to entities and also determine the type of the entity. Usually, the number of entity classes is around 4 or 5, so the size of the tagset is similar to the one of chunking.

This task is commonly performed for information extraction and it is an intermediate step in entity tracking.
2.2 Structured prediction algorithms

As discussed in the introduction, common machine learning algorithms are not adequate for predicting structures considering their intrinsic properties. However, popular structured learning algorithms are mainly based on them, so it is good to have familiarity with those methods. Specifically, perceptron [6], logistic regression [3] and support vector machines (SVM)[14] are strongly related to them.

Also, important concepts of non-structured supervised machine learning are equally present in supervised structured prediction. The learning process is based on adapting a model to some correct examples of input-output pairs (training data). Usually, during that learning process the goal is to establish the parameters as to maximize the correctness of those examples. However, the ultimate goal is to predict unseen data correctly, and to that end the method has to generalize: not focusing too much on the particularities of the training dataset (usually called “overfitting”), but capturing the underlying map. This is often referred to as the bias-variance problem for an algorithm. A model with high variance, that is, sensitivity to the particular characteristics or noise of a training dataset, will lose capability to generalize and therefore it will perform worse over new data. A model with high bias, that is, inaccuracy with respect to the training data, can miss the relationship between inputs and outputs and will also have a poor performance. This is a trade-off problem and it is crucial to find an optimal configuration where the error over new unseen data is minimized. An illustration of it is found in figure 2.1. To evaluate the performance over unseen data, it is common to divide the available input-output data in a part used as training data, and a part (test data) used to evaluate the performance of the algorithm, not using its inputs on the training phase and testing whether, using the trained model, the given output equals the original one. During the training phase it is also possible to split the data, not involving a part of it in training and using it to tune parameters of the model (development data) as to maximize the accuracy over that partition. After that, the test data is use to evaluate the performance as described above.
Additionally, the “curse of dimensionality” is a common denomination for the problem of dealing with many dimensions as an input. Some methods cannot handle very large number of dimensions, whether it is for computational (e.g., the need of using much training data with increasing computational complexity) or accuracy (e.g., lost of meaning of euclidean distance in large dimensions) reasons. Therefore, it might be important to use feature vectors of a reasonable dimension for certain algorithms.

### 2.2.1 Features

Structured prediction algorithms use a different concept of feature function than that of unstructured machine learning. In the latter, only inputs \((x \in \mathcal{X})\) are used to determine the output. In the first, both inputs and also a hypothetical output \((y \in \mathcal{Y})\) are used. Abstractly, the feature function \(\Phi(x, y)\) maps that pair to a real vector. This function can represent, as an example, if the third word of the sentence is preceded by a word labeled as a pronoun or an article, or if the last symbol of the sequence is a quotation mark. This reflects the usage of the output’s structure in the learning process.

### 2.2.2 Structured perceptron

The structured perceptron is a variation of the perceptron for structured learning. Concretely, a variation of averaged perceptron is often used in practice [43]. The learning algorithm for that version is:
# Algorithm 1: Averaged structured perceptron

1: Input: Training data \( \{(x_1, y_1), (x_1, y_1), \ldots, (x_n, y_n)\} \), number of iterations \( I \)
2: Output: \( w_f \)
3: \( w \leftarrow (0, 0, \ldots, 0) \)
4: \( w_c \leftarrow (0, 0, \ldots, 0) \)
5: \( c \leftarrow 1 \)
6: \( \textbf{for } i = 1, \ldots, I \textbf{ do} \)
7: \( \textbf{for } j = 1, \ldots, N \textbf{ do} \)
8: \( \hat{y}_n \leftarrow \text{argmax}_{y \in Y} \quad w^t \Phi(x_n, y) \)
9: \( \textbf{if } \hat{y}_n \neq y_n \textbf{ then} \)
10: \( w \leftarrow w + \Phi(x_n, y_n) - \Phi(x_n, \hat{y}_n) \)
11: \( w_c \leftarrow w_c + c(\Phi(x_n, y_n) - \Phi(x_n, \hat{y}_n)) \)
12: \( c \leftarrow c + 1 \)
13: \( \textbf{return } w_f \leftarrow w - (w_c/c) \)

The idea of the algorithm is to iteratively learn a weight vector, updating it when an error in classification on a training example is made. This is done by increasing the importance of the weights corresponding to features of the true label and decreasing the importance of the ones corresponding to features of the wrong predicted label.

The only variation of the averaged version is the use of \( w_c \). Otherwise, the algorithm is the same for the normal version, which would return simply \( w \) as \( w_f \) (known as the weight vector). This averaging seeks a better generalization, avoiding dependency on the order of the training examples and overfitting. The weight vector that is learned during the training phase is later used for predicting the outcome of new examples, which is calculated as:

\[
\hat{y}_n = \text{argmax}_{y \in Y} \quad w^t \Phi(x_n, y)
\]  

(2.1)

The \textit{argmax} operation is not trivial because the combinations of the output structure for a given input can be huge, as we have seen in section 2.1. Therefore, an efficient search is necessary for this method to work. This is a common requirement in most of structured prediction algorithms. How to do this is out of scope in this study, but there are various alternatives present with exact (e.g., Viterbi algorithm) or approximate search (e.g., Beam search or greedy approaches) [23] [43] [11]. In the exact version, the training algorithm is guaranteed to converge after a certain number of steps, finding a perfect solution for the training set if possible. However, since this in practice might mean a much larger number of computations that do not improve the result on unseen data, it is common to use a fixed number of iterations.

## 2.2.3 Conditional random fields

Conditional random fields (CRF) are based in logistic regression, extending its concept to structured outputs. They define a conditional probability for an output structure, depending on the input and a weight vector \( w \), as:
\[ p(y|x, w) = \frac{\exp[w^t \cdot \Phi(x, y)]}{\sum_{y' \in (Y-\{y_c\})} \exp[w^t \cdot \Phi(x, y)]]} \]  

(2.2)

Where \( y_c \) is the correct output (the sum of the denominator is done over all incorrect outputs). In the training phase, the objective is to learn the weight vector \( w \), as in the previous method. However, instead of following an iterative approach, \( w \) is found by maximizing \( \sum_{D} p(y_t|x_t, w) \) over \( w \), where \( D \) is the set of training examples \((x_t, y_t)\). Sometimes, a prior distribution over the weight vector (a distribution expressing our belief over \( w \)) is introduced in that equation. This prior forces \( w \) to have smaller weights. This technique is also used in logistic regression, and it is a way to avoid overfitting because very large weights are usually an indicator of it. Once the model is trained, new outcomes are decided in a equivalent manner to the structured perceptron:

\[ \hat{y}_n = \arg\max_{y \in Y} p(y|x, w) \]  

(2.3)

Again, that search is not trivial and it requires of an efficient method. Also, the denominator of the conditional probability has to sum over the same vast space, so similar techniques are required for calculating it. Moreover, the maximization of the training phase has to be computed using general optimization strategies.

Figure 2.2 shows a comparison of conditional random fields and logistic regression with other similar models. Linear-chain CRF are commonly used for sequence labeling. Even though it might seem that their only difference with hidden Markov models is the direction of the dependencies, they also differ in that conditional random fields use features functions for each output node instead of simple words. Those features can have information about the rest of the sequence, so dependencies with the whole linear sequence can be represented. Additionally, the probability functions modeled are different.

![Figure 2.2: Comparison of conditional random fields (from An Introduction to Conditional Random Fields [38])](image-url)
2.2.4 Structured support vector machines

The structured support vector machines (SSVM) [41] technique is an extension of support vector machines for complex outputs. It formulates a quadratic programming problem as:

\[
\text{minimize } w, \xi \quad \frac{1}{2} ||w||^2 + C \sum_n \sum_y \xi_{n,y} \\
\text{subject to } w^t \Phi(x_n, y_n) - w^t \Phi(x_n, y') \geq 1 - \xi_{n,y'} \quad \forall n \forall y' \\\n\xi_{n,y'} \geq 0 \quad \forall n \forall y'
\]  

(2.4)

Where \( w \) is the weight vector, similar to the case of structured perceptron. It seeks to minimize the norm of the weight vector while maintaining a difference between the true label and the rest (known as margin). \( \xi \) is a “slack variable” that allows this difference to be smaller. \( C \) is a “hyperparameter” (a parameter that is defined before the training phase is carried out) of the model that controls the trade-off between minimizing the weight vector and minimizing the slack variable (which means maximizing the margin). The number of constraints is usually huge and cannot be handled by usual optimization procedures. Instead, an iterative procedure is used, where constraints are added at different steps as described in the original formulation [41].

After the optimization is done and the weight vector is found, the prediction of new examples are done exactly in the same way as in structured perceptron:

\[
\hat{y}_n = \arg\max_{y \in Y} w^t \Phi(x_n, y)
\]  

(2.5)

In the non-structured case, using the dual optimization problem and the “kernel trick” we can transform the weight vector and features to higher dimension spaces with non-linear functions. However, for structured prediction, this would make the problem extremely inefficient and for sequence labeling it would probably have small impact on the result.

2.2.5 Trigram TnT

Trigrams’n’Tags (TnT)[7] is a method that uses a hidden Markov model approach with a second-order Markov chain (using three consecutive instances, hence the name “trigrams”). It models tags as hidden states, and words as the corresponding outputs. In the learning phase, it estimates the probabilities of a tag given the two previous ones, \( P(t_i|t_{i-1}, t_{i-2}) \), and the probability of a word given its tag, \( P(w_i|t_i) \), calculating the relative frequency of these occurrences in the training data. Additionally, it uses the probability of finishing a sentence after the last tag, \( P(e|t_n) \), where \( e \) indicates end of sentence. After determining these probabilities, it predicts the sequence of tags of a new sentence as:
\[ T = t_1...t_n = \text{argmax}_{t_1...t_n} \left[ \prod_{i=1}^{n} P(t_i|t_{i-1}, t_{i-2})P(w_i|t_i) \right] P(e|t_n) \] (2.6)

It is noticeable that it does not use an explicit feature function. Instead, the probabilities of the tags and the word are used. However, we can see this as a feature function for each tag with the two preceding tags and the current word as input, \( \Phi(w_i, t_i, t_{i-1}, t_{i-2}) \). Nonetheless, this means that it is not possible to design an arbitrary feature function (e.g., if the previous word’s last letter is “s”). Moreover, it does not explicitly use tags following the current one to calculate those probabilities. Nevertheless, the \text{argmax} operation makes use of the joint probability of the whole sequence, and therefore the following tags also condition the output even though they are not present as features. This method can be adjusted to handle unseen words and unseen sequences of tags in the learning process to achieve better results. To solve the \text{argmax} operation, the Viterbi algorithm [18] can be used. However, for efficiency purposes, it can be interesting to use beam search to get an approximate solution reducing the computational time.

Furthermore, this is a generative model, whereas the previous ones are discriminative. This means that TnT can generate pairs of sentence-label examples, since it learns the joint probability distribution \( P(x, y) \). On the other hand, the three previous methods only model the conditional probability distribution \( P(y|x) \).

Another different characteristic of this algorithm is that it can only be applied to sequential structures, and consequently is less general than the three previous techniques, which can be extended to any type of structure.

### 2.3 Other approaches for sequence labeling

Historically, methods which are not based on structured prediction have been used for sequence labeling. As commented in chapter 2, the simplest method is to assign to each new word the most common tag it has in a specific training dataset. This is known as the unigram tagger.

Another technique, known as rule-based tagging, is to use known linguistic rules and properties. The most successful method of this kind is the Brill tagger [8], specially for POST. It is based on first assigning tags to sentences using any method, e.g., unigram tagger; and then applying a series of deterministic rules to predict the final tag. This rules can be predetermined by known linguistic properties or learned for a specific dataset based on some templates. This templates have the logic form “change label 1 to label 2 if condition”, where condition can be any feature of the sentence. Examples of this features are if the previous and the next label are nouns, the suffix of the word is “ed”, or the next word is “do”. The error-driven learning process [9] involves to try every possible rule over the training data and to determine which rules improve the result the most. In this sense, it is a learning algorithm based on trial and error. This method can be used on top of structured
prediction algorithms, first applying them to the data and then using the determined rules. Both methods will be evaluated and compared to the structured prediction algorithms in the experiments.
3 Method

For the experiments, the NLTK [2] 3.0 library for python was used. The implementation of CRF is a python binding of the version of Naoki Okazaki (2007) [31] which employs a linear-chain structure and the used average perceptron is the built-in implementation made by Matthew Honibbal, which uses a greedy approach that decides outputs word-by-word. The TnT method used is a library built-in implementation. The structured support vector machines method was not included in the NLTK library, so a C implementation by Thorsten Joachims that utilizes a chain structure similar to the one of hidden Markov models has been used. The prior feature extraction for SSVM was performed in python, specifically designed for the input format of the C program. Apart from those, the simple unigram tagger and a transformation-based tagger were also tested, to compare them with structured prediction methods. The implementation of the Brill Tagger is also built-in in NLTK. All of these implementations were subsequently adapted to handle the form of the input of NER and chunking, since they are originally written for POST.

For every experiment, the process was to first train the model with a subset of the desired dataset, and then test over another subset not used in the training phase. Separate datasets could be used for testing, but since the tagging convention varies for different datasets, this approach was not followed. To have an accurate evaluation of the performance, the training-testing process was repeated 5 times for each experiment, and then the mean value of the performance was calculated. Five different training set sizes were examined: 500, 1000, 2000, 4000 and 8000 sentences. The test set was composed of 500 sentences for every experiment. The training and testing subsets were randomly chosen in every iteration, to avoid any particular order. Moreover, for the structured prediction algorithms with the best performance, the mean training time was measured, as an approximation to the capacity of each algorithm to scale for large training sets.

For the case of the Brill tagger, it was trained over the same training set as the base tagger used and evaluated over the same test set. The template of rules is the same as the one described in the original paper [8]. This was done for every algorithm as the initial tagger, with two of the following the datasets (Spanish CoNLL-2002 Corpus and NPS Chat Corpus). The difference of per-word performance before and after applying the Brill tagger was measured.

In the NER and chunking experiments, the true part-of-speech tags were used as input features. It could be possible to use the result of the experimental tagging, but this could
introduce a dependency on the performance of those first experiments, so to have an independent analysis this was discarded. TnT was not evaluated over these tasks because it cannot handle both words and part-of-speech tags as features. Although the unigram tagger has the same characteristic, a single experiment with the largest training size was done to evaluate the intrinsic difficulty of each dataset.

3.1 Data

For POST, different datasets were used, involving different languages and domains. Apart from the following datasets, a subset of about 3900 available sentences of the Wall Street Journal dataset from the Penn Treebank [28] took part in a single POST experiment without randomization for each algorithm. It is the most used dataset for POST evaluations, and it was tested to compare the performance of these methods to that stated in other studies. However, it could not be evaluated following the proposed method because it is not publicly available, apart from those sentences. The 3400 first sentences were used as training data and the rest (approximately 500) were left as test data. Additionally, for the algorithm with the best performance, a single iteration with 50000 training sentences and a testing with 5000 sentences of the Brown Corpus was done, to evaluate the improvement that can be achieved using large training data. Also the unigram tagger was tested in this way to have a baseline to compare. Standard dataset were tested for shallow parsing and named entity recognition.

3.1.1 Brown corpus

The Brown corpus [20] is a text collection of English-language texts created in 1961 at Brown University. It is a compilation of more than one million words, taken from 500 different sources, and classified in diverse thematic categories (e.g., news, fiction, humor, editorial or religion). All the examples used are texts printed in the United States during that year. The part-of-speech annotations use 82 different categories. Hence, achieving a great accuracy might be more difficult than for other datasets with a smaller number of annotations, which are commonly used as tests for POST solutions (e.g., the Penn Treebank tagset)

3.1.2 Spanish CoNLL-2002 corpus

This corpus is a compilation of news articles in Spanish published in 2000 by the “EF E” news agency. It is composed of around 300000 tokens and more than 8500 sentences. The compilation and annotation was done by the Technical University of Catalonia and the University of Barcelona. Its tagset has 59 different categories. This corpus is also annotated for named entity recognition, and it was used to measure the performance on this task. The entities detected are divided in persons, organizations, locations and miscellanea.
3.1.3 Dutch CoNLL-2002 corpus

It is a set of Dutch news published by the Belgian newspaper “De Morgen” in 2000, put together by the University of Antwerp. It uses the universal tagset described by Petrov et al. (2012) [32], which has 12 tags very similar to the ones usually taught in school. It is also entity annotated, and therefore used for experiments of that task. The entities are divided as in the previous corpus.

3.1.4 Floresta treebank

The Floresta Treebank [1] is a compilation of more than nine thousand sentences from Portuguese news articles published in the “Publico” newspaper in 2000. Its tagset is formed by 17 simple categories.

3.1.5 Twitter bootstrapped dataset

This corpus [17] is a collection of English posts (or “tweets”) of Twitter, containing more than 1.5 million words. Its part of speech annotations have the same categories as the Penn Treebank [29], extending it with 4 additional tags (USR, HT, RT, URL) for specific parts of a tweet (40 tags in total). Instead of being manually annotated as the rest of corpus, it is a “bootstrapped” dataset. This means that the labels are automatically assigned by using a number of different methods trained on different datasets and including a sentence in the corpus if every method agrees on its labels. It is reported that, even though it is constructed this way, it has an estimated accuracy of 97.5%; which is similar to the percentage of agreed tags by humans for this tagset.

3.1.6 NPS Chat corpus

The NPS Chat Corpus, recorded by Eric Forsyth, Jane Lin, and Craig Martell [19]; consists of more than 10,000 posts of different age-specific chat rooms in English. It also uses the Penn Treebank tagset, but it does not extend it (36 categories). Because of their nature, these post have a smaller average word length than sentences of previous datasets, even smaller than the ones of the Twitter corpus.

3.1.7 CoNLL-2000 corpus

This dataset [39] has more than 10000 sentences, annotated for shallow parsing. It is the only dataset used as benchmark for that problem in this study. There are 11 different chunk categories, though three of them represent more than 90% of the chunks. It also has part-of-speech tags, which are used as features. Its POST tagset is the one of the Penn Treebank too.
3.2 Features

The features chosen for each method were the same for every experiment of POST. However, different methods had different features, due to their specific characteristics.

For structured perceptron, the features were: a bias feature that always has the same value for every word, the current and two previous tags, the prefix (of length 1) and suffix (of length 3) of the current and surrounding words, a combination of the two previous tags, and a combination of the previous tag and current word. It does not use following tags to allow for a greedy fast iterative search focused on a single word at a time. The input data was normalized, removing capital letters and setting a processed label for numbers (either “year” or “number”).

The features of CRF and SSVM were very similar between each other. They have fewer features per word compared to the previous case, but since these techniques treat the whole sentence as an input, the resulting number of features was bigger. For each word, they were its prefix and suffix, the word and the tag. Additionally, for CRF the data was not normalized and it also included features indicating if the word has numbers, punctuation or capital letters.

As we have seen in section 2.2.5, the TnT approach does not use explicit features, so there is no feature design involved in that method.

These features are all binary in practice. For example, there is a feature interpreted as “this word is play”, or “the suffix of the previous word is -ing”. Consequently, the dimension of the features is huge, and using a bigger number of feature categories can make the methods suffer from the curse of dimensionality, without effectively improving the performance. Even so, the structure of the feature space is binary and therefore the simplest one. These two characteristics might be important to analyze the performance of different algorithms.

For NER and chunking, the features were the same adding the information of part-of-speech tags. For perceptron, this information was the tag of the word and the two preceding tags. For CRF and SSVM, it was the tag of the word for each word in the sequence. In the case of the unigram tagger, only part-of-speech tags were used as features, because this method cannot handle both words and tags at the same time.

Some trials were attempted in order to check if that was the best configuration of features. For perceptron it turned out to be the most promising one, with no selection performing better. For CRF, normalizing the data and removing the case, punctuation and number sensitive features and adding prefix and suffix features, a combination of those and a “bias feature” for each word improved slightly the performance for some datasets, in the range of 1% accuracy increase. The same was true for SSVM with the prefix, suffix and bias features. However, they were not used as the number of features increased importantly for both algorithms because they use the whole sequence information, and it made these
methods extremely slow even for small datasets.

### 3.3 Performance measures

The most common performance measure in part-of-speech tagging is the accuracy per word (that is, which overall percentage of tags are correctly classified), which means adopting the Hamming loss. This measure was tested for every dataset. Additionally, for the same experiments, the accuracy per sentence was also measured. In this case, a sentence is considered correct only if the label of every word in that sentence is correct. Formally, they are respectively:

\[
P_w = \frac{\sum_{t=1}^{T} (y_t = \hat{y}_t)}{T}
\]

\[
P_s = \frac{\sum_{n=1}^{N} (\vec{y}_n = \vec{\hat{y}}_n)}{N}
\]

Where \( N \) is the number of sentences and \( T \) the total number of words. For the execution time, trying to establish an evaluation of scalability as independent from the hardware as possible, the time spent in the training of each size divided by the time spent in the smallest one was measured. Also the time spent in the smallest one was recorded as an approximation of the base complexity.

For NER and shallow parsing, the same measures were used. However, it is also common to evaluate the precision (proportion of identified chunks that are correct), recall (proportion of true chunks that are identified) and “f-measure” (the harmonic mean of the two previous measures) to evaluate result. Nevertheless, they do not seem appropriate to examine the algorithms. Intuitively, in the example of NER, if an entity that exists is guessed wrongly (e.g., it is a location but our guess is that it is a person) the cost affects both recall and precision so it is bigger than if it is missed, which only affects recall. This does not seem reasonable and therefore, considering this problems as strictly sequence labeling, the accuracy per word might be a better performance indicator.
4 Results

4.1 Part of speech tagging

The token and sentence average accuracy for the POST datasets with different training sizes for every method is shown in figures 4.1-4.12. The relative time performance of CRF, perceptron and SSVM is shown in figure 4.13. Improvements using the Brill tagger are shown on figure 4.14. Finally, performance for the single iteration on the Penn Treebank corpus is shown in figure 4.15.

![Figure 4.1: Average token accuracy for the Brown corpus](image1)

![Figure 4.2: Average sentence accuracy for the Brown corpus](image2)
Figure 4.3: Average token accuracy for the Spanish corpus

Figure 4.4: Average sentence accuracy for the Spanish corpus

Figure 4.5: Average token accuracy for the Dutch corpus

Figure 4.6: Average sentence accuracy for the Dutch corpus
Figure 4.7: Average token accuracy for the Floresta treebank

Figure 4.8: Average sentence accuracy for the Floresta treebank

Figure 4.9: Average token accuracy for the twitter corpus

Figure 4.10: Average sentence accuracy for the twitter corpus
Figure 4.11: Average token accuracy for the chat corpus

Figure 4.12: Average sentence accuracy for the chat corpus

Figure 4.13: Relative average computing time on the Brown corpus

Figure 4.14: Improvements on unigram with the Brill tagger for the Spanish and chat datasets
The first training average time used as a factor in figure 4.13 was 12.5 seconds for perceptron, 9.10 seconds for SSVM and 54.0 seconds for CRF. Training times on other datasets are not shown as they exhibit an almost identical behaviour. The training time for TnT was not numerically measured, but it was faster than those and scaled better. However, its testing time scaled badly, as opposed to the other methods which are similarly fast on testing phase and scale good.

Apart from the unigram tagger, no method was improved significantly (all improvements were smaller than 0.001) by combining them with the Brill tagger, so those results are not shown.

A single iteration of structured perceptron over 50000 sentences on the brown corpus resulted on a 0.962 token accuracy. The baseline (unigram token performance) for that same iteration was 0.896.

### 4.2 Shallow parsing and named entity recognition

Average performance of CRF, perceptron and SSVM for the NER and chunking datasets are shown in figures 4.16-4.21.
Figure 4.16: Average token accuracy for the Spanish NER corpus

Figure 4.17: Average sentence accuracy for the Spanish NER corpus

Figure 4.18: Average token accuracy for the Dutch NER corpus

Figure 4.19: Average sentence accuracy for the Dutch NER corpus
The performance of the unigram tagger was tested only for the largest training size, so those results are not included in the figures. The token accuracies were 0.879, 0.900, 0.180; and the sentence accuracies 0.252, 0.536, 0.002 respectively.
5 Discussion

5.1 Comparison of algorithms

The relative performance of the algorithms is similar across every dataset, with structured perceptron being in general the most successful method, followed by SSVM and CRF. TnT performs considerably worse than those methods, and it is closer to the accuracy of the unigram tagger. This is not surprising because as we have seen in section 2.2, the information of the sentence it handles is much more restricted than the other algorithms. It only involves frequencies in the training data of previous labels and the current word to determine each label, and it does not use any other type of feature (e.g., which is the next word). In addition, it does not optimize globally. We can observe that structured perceptron achieves the best token accuracy for the largest training sizes except for the POST Spanish and chat datasets, where SSVM outperforms it. SSVM has a slightly better relative performance regarding sentence accuracy than token accuracy, obtaining the best result for those two datasets and also for the Brown corpus. This means that a wrong decision can lead to higher number of mistakes in that sentence for SSVM. It is reasonable, since the structured perceptron tested has a greedy approach makes decisions more independently of remote words in the same sentence. However, this difference is not very pronounced. CRF also makes decisions sentence-wise, but this effect is not observed. An explanation can be that the implementation of CRF has a linear-chain structure, where the optimization is less correlated between distant parts, opposed to the more global optimization present in SSVM. Furthermore, SSVM performs relatively better with smaller training size, being the best method for the smallest training size for every POST dataset. In general, its accuracy grows slower when the number of sentences increases compared to perceptron and CRF. Hence, SSVM might be the best option for sequence labeling problems where training data is limited.

The Brill tagger is only able to significantly improve the results of the unigram tagger. The type of transformation rules it involves is very simple, so perceptron, CRF and SSVM are already able to extract an abstraction of those rules. TnT is restricted to the neighbor tag and word information, so it already does a similar evaluation, and it still keeps a similar or better performance than the combination of unigram and Brill tagger for the datasets evaluated. The combination approach was therefore concluded to be useless for the evaluated tasks.

The training time of CRF is particularly large because it scales much worse than the other methods. This does not affect during the tagging of new examples, and the training of
the algorithm can be done just once for some purposes. However, sometimes many different training iterations might be needed, involving a large quantity of data. It can cause CRF to be unfeasible, and there are recent studies on how to tackle this problem, e.g. Trevor Cohn (2007) [10]. Although TnT is a very fast method in the training phase, the testing phase is much slower and scales worse than the other methods. During prediction time, a lot of floating-point operations have to be done to estimate the probabilities of sentences, meanwhile in other methods that estimation is a simple multiplication between the weight and the feature vectors. It is also possible that its implementation is not properly optimized.

Even though it might be tempting to generalize the superiority of the perceptron within structured prediction techniques, this might not be true for others tasks. The features in these problems are very sparse, which can downgrade the performance of CRF and SSVM, because they use more sophisticated global optimization techniques. The features are also binary, which makes particularly easy to find a good weight vector in a linear space. For features living in the real space, CRF and SSVM might be able to learn better. Additionally, the structure in these problems has mainly local dependencies, which favors the greedy approach perceptron takes, while other problems might have more remote relationships that could be better represented by CRF and SSVM. Besides, that greedy approach makes perceptron able to include more features without making it terribly slow. The number of training examples is considerably large, so it can help perceptron to converge iteratively to a good solution, and the use of features from following words can make up for its weaker representation of output structure dependencies. Results in other fields have shown the other two methods as more promising, for example object recognition [33] or stereo vision [26] in computer vision.

The three methods with the best accuracy are conceptually not very different from each other. Structured perceptron and SSVM use the same strategy for labeling. Their difference is how they learn the weight vector. Both try to find a vector that classifies correctly the label of training examples, dividing the space of features accordingly. SSVM explicitly maximizes the margin between these examples and the rest. However, perceptron also does a similar work by averaging of the weight vector throughout iterations. Moreover, Kai Zaho (2014) [43] shows that structured perceptron can be viewed as an iterative and “hard search” version of CRF. Therefore, it is not surprising that their behavior is somewhat similar across different experiments.

5.2 Dataset performance

Observing the best accuracies achieved for different datasets of POST, it is clear that the performance is very dependent on the specificities of the corpus tested. For token accuracy, the chat corpus is the most difficult one, which can be explained by both its smaller average sentence length and its noisy and not formal nature. This also can explain the superiority of SSVM in that corpus. Nevertheless, its sentence accuracy is relatively high, due to that smaller sentence length. The twitter dataset also has a high sentence accuracy, but its token accuracy is the biggest achieved (more than 96.9%). Both have very similar tagsets. The conversational nature of chat might cause more problems, and the similarity between different
“tweets” is relatively high. Moreover, the 4 additional tags in the twitter dataset (USR, HT, RT, URL) are completely determined by the form of the word, and they frequently appear in the data. Additionally, its “bootstrapped” nature makes it not include some sentences in the dataset which are ambiguous and problematic. Generally, a smaller tagset should mean that the accuracy is higher. However, the Dutch corpus has a simple tagset and it does not get the best token performance, even though it gets superior sentence efficiency than the other “formal” datasets. Its average length is also smaller than those, so some very short sentence which are difficult to tag (such as sport results) might be included. The Portuguese (which uses 17 tags) and Spanish (which involves 59 tags) datasets have similar performance. However, the unigram tagger has quite superior performance on the Spanish corpus than in the others, which indicates that it might be intrinsically easier to tag. The Brown corpus does not get more than 93.8% token accuracy, being the second worst after the chat corpus and having the worst sentence performance. It has a very specific tagset (82 tags) and it also has low accuracy for the unigram tagger.

The results were relatively similar for the Dutch and Spanish NER datasets. The baseline token accuracy given by the unigram tagger is slightly higher for the Dutch corpus and the best result is consequently slightly higher. As explained before, the Dutch corpus has shorter length of sentences, so it is not surprising that its sentence accuracy is superior, both for the baseline and the best performances. As opposed to the around 90% baseline accuracy of the NER datasets (because most of the words are not entities), the chunking dataset has less than 20% for the unigram tagger. Nevertheless, the other scores for this dataset are in line with the NER experiments. This fact can be somehow surprising, as this seems in principle a more difficult task than POST, but the performance is similar to the best of POST news datasets. The fact that the information used is both words and correct part-of-speech tags can be determinant for it.

The training iteration with 50000 sentences of the Brown corpus with structured perceptron gave an increase of almost 3% accuracy respect to the one with 8000 sentences. The baseline accordingly improves more than 4%. Therefore, although the increase of accuracy seems to asymptotically decrease respect to the training size in the experiments, it is reasonable to also expect general increases of this proportion using larger training sizes for other algorithms and datasets. Moreover, the performances for the Penn Treebank single iteration are similar to other experiments performed on this dataset with corresponding training size [30]. With the whole dataset (composed of more than 38000 training sentences), it is reasonable to expect an increase of accuracy that achieves a very close performance to state of the art reported results (around 97.3%), as argued above.
6 Conclusions

It has been shown that, out of the evaluated methods, structured perceptron has the best performance in general for sequence labeling on the largest datasets. It has also been observed that SSVM is superior for smaller and noisy datasets. Nonetheless, it is argued that this may not be true for other structured prediction problems because there are some specific characteristics that might provoke these results. The sparseness and binary representation of the features, local dependencies between words and large training size might be the factors influencing this outcome. The Brill tagger combination with structured prediction methods has not been successful. Although the variation between datasets is huge, the comparative performance between methods is relatively similar. It also can be concluded that the reported absolute accuracies are very dependent on the datasets, their type of language and environment, their number and complexity of tags, and the training size used. Therefore, it is meaningless to simply state a certain accuracy without specifying those conditions. It has also been observed that further accuracy improvement is possible using very large training sizes.

Languages and environments that are not usually tested as a benchmark for part-of-speech tagging have been evaluated. The chat and twitter evaluations can be particularly interesting nowadays, as an example of data produced on the Internet that is noisy, with misspellings and frequent use of slang or that includes conversational speech. The accuracy for the twitter dataset has been superior than other experiments reported for this type of data [21] [17]. However, as previously discussed, that may be due to particular conditions of this experiment.

General structured prediction algorithms have shown the best performance, in line with state of the art reported results where comparable. It can be concluded that they are a very promising approach for sequence labeling, and it may be interesting to apply them to other problems where structures with interdependent elements are present.

6.1 Future work

CRF and SSVM could be tested using a greater number of features or more computationally expensive configurations, such as non-linear kernels for SSVM or more connected linear structures for CRF; possibly using large scale computational power. Those test could be useful to see if those configurations are able to improve the accuracies significantly. For the
analysis of the performance, new accuracy measures can be used for POST tagsets which are very specific (such as the tagset used in the Brown Corpus), in which assigning a wrong tag which belongs to the same category class as the original one is not as bad as assigning a completely uncorrelated tag. It can be of interest to also have an evaluation of this performance because other tasks where POST is an input often use a more general tagset (e.g., chunking). These measures would have to be designed and adapted for each different tagset. Also, f-measure, recall and precision could be evaluated for NER and chunking.

Other structured prediction algorithms that use different concepts can also be worth to study over the sequence labeling problem. Ensemble methods for structured prediction [13] use the popular idea in “unstructured” machine learning of combining the output of various algorithms to improve the performance. There are different ways to do this (e.g., majority vote, bagging or boosting) and this can be a useful method for structured prediction too. Search-based structured prediction [16] is a meta-algorithm that transforms the structured prediction problem into a series of binary classification problems for which any supervised learning algorithm can be applied. It does so by assigning costs for different individual problems based on the structure, thus transferring the dependencies to the binary classification tasks. This technique in combination with different binary-classification algorithms is another possibility for further evaluation, specially interesting for problems in which Hamming loss is not appropriate. Furthermore, “SENN” [12] is based on the paradigm of also learning features automatically with a multilayer neural network, instead of designing them by hand and using them as an input. This technique has shown great performance in many other machine learning tasks, and might also be promising for structured learning.

Moreover, an evaluation over datasets of not Indo-European languages could be also interesting. This would probably require quite different feature models and tagsets to cope with the specific linguistic properties of languages such as Chinese or Arabic. It would also require large datasets, which are not very common, for testing these fully supervised methods.

Also, other sequence labeling problems can be worth to study, such as optical character recognition. Furthermore, other tasks of NLP which outputs are not sequences (e.g., parsing or document summarization) can be interesting for examining these algorithms. Nevertheless, these problems usually require of linguistic techniques in combination with machine learning algorithms for a reasonable performance. Consequently, this type of study would need a standardized linguistic environment, so the results would only depend on these algorithms for an objective analysis.
7 Bibliography


[24] Jin Kiat Low Hwee Tou Ng. Chinese part-of-speech tagging: One-at-a-time or all-at-once? word-based or character-based?


