SCIENTIFIC PAPER UDC 004.912:811.163.41’322

TAGGERS APPLIED ON TEXTS IN SERBIAN*

Zoran Popović**
Hemofarm, STADA

Abstract: This paper provides a comparative overview of existing language tools based on taggers and machine learning methods, with practical tests and results about different taggers applied on texts in Serbian. For that purpose some already prepared annotated corpora were used, and 10-fold cross validation was used as the testing framework with a specially devised and developed environment of automated testing based on unix scripting (bash, perl, awk) – TnT has shown best performance, while Tree Tagger and SVMTool taggers have shown somewhat better performance in special cases. A possibility of combining different tagging methods and tools (programs) and integration with other NLP environments opens a wide area for further investigations and experiments about these solutions.

Keywords: tagging, tagger, PoS, machine learning, NLP, Computational Linguistics, CL

*This paper overviews the results presented in the Master Thesis submitted at the Faculty of Mathematics, University of Belgrade

**shoom013@gmail.com

1. Introduction – two paradigms of Computational Linguistics

NLP (Natural Language Processing) as an area of Computational Linguistics usually implies very complex processes in terms of computability and time needed for the processing. It consists of phases such as lexical analysis (segmentation and tokenization of input speech or text, which starts with finding beginnings and ends of sentences or words, and detecting general lexical categories – future lexemes or tokens: numbers, punctuation characters, words, HTML tags, and similar), morpho-syntactic analysis (structure of a word, sentence or text), and finally, semantic analysis or even pragmatic analysis. Parsers, programs that do syntax analysis, have a highly complex task to cover all the rules and properties of a natural language. The traditional approach is "top-down", where with this complex analysis in a conjoint process one also gets more simple properties like lexical properties. The goal and basic result of the analysis are structures and grammatical rules like formal grammar rules of Noam Chomsky (syntax tree of a sentence). This approach is aligned with the unfulfilled goal of describing a whole natural language with an appropriate first-order predicate calculus theory.

The second quantitative approach to these problems which fits more with a reverse ("bottom-up") approach in which the process starts with efficient and fast algorithms of statistical nature which are not exact in the former traditional sense and which are used for more simple lexical tasks, gave unexpectedly good results and significant contributions to new NLP solutions. A class of such programs, called taggers, is concerned about discovering categories of words in a sentence. These programs put simpler, for each word in a text based upon its morphological, syntactical and other roles, associate the word with an appropriate tag which describes association with subclasses of lexical classes like: nouns, pronouns, verbs, prepositions, etc. The number of these subclasses and their categories can be extensively large if other properties of words are also taken into account: gender, number, person, case, and similar. Their number is determined by the complexity of a given language model built in that way, but also by the language itself. Somewhere during the 80-ties first practical results of European researchers with CLAWS program (Leech 1987), which is partly based on the Hidden Markov Model (HMM), have turned further interest for these and similar problems, and also specially for taggers and further development of the statistical NLP, and for the application of machine learning in NLP.

These two approaches represent two different paradigms which could be also identified in related areas of computer science, and in the different historical periods these approaches have developed differently and sometimes had conflicts: the symbolic against the quantitative paradigm, which can be traced up to semiotic conflicts of rationalists and empiricists, in terms of NLP or behavioristic debates. On one side there is an intuitive analysis and often "manually" created symbolic rules – and on the other, an automatic rule generation given through corrections of numerical parameters of a statistical model. The first paradigm has deeper and often more intuitively clear linguistic knowledge of the natural language, while the second paradigm is more robust and ready to overcome an erroneous input and ambiguities, having better generalization capacity in practice, while often showing more superficial and shallow linguistic knowledge. In the domain of text engineering (a notion met in commercial applications like machine translation systems and other NLP implementations) it is often more practical to use automatically generated rules, maybe not so descriptive and intuitive, but realized in less people-days and other resources, while offering results with a quite good performance. Of course, specific areas of application are not the final reach of these paradigms, and

their research activities are continuing to develop in quite a complementary manner. The well-known IBM's team under Frederick Jelinek's guidance has built a speech recognition system in the mid 70-ties based on statistical methods. This was done in the times of the great crises of artificial intelligence researches and symbolic paradigm in general, after the famous Lighthill report in year 1973 (Dreyfus 1974) which was well-inspired by poor results of machine translation at that time and which has undermined the financing of such researches for at least a decade. During 1990-1994 period in the DARPA MTEval initiative, Jelinek's machine translation system that was statistically-oriented and with which the language structure had no importance in the beginning, CANDIDE, has never shown better performance neither from the elder symbolic-oriented SYSTRAN, nor from the "opponent" Pangloss[4] system in that manner, which can be explained with the insufficient syntactical and semantic knowledge of that system. In the end, both CANDIDE and Pangloss became hybrid solutions in order to remain sufficiently robust and advanced. There is no general machine learning system equally good for all applications and domains[3] (Schaffer 1994) and more significant than specific solutions at the same time – that is why the implicit knowledge is important for machine learning (bias) which is present in any symbolic theory, or in the conceptual knowledge given by well chosen grammatical rules in NLP. From the mid 90-ties, taggers and statistical parsing had an additional expansion also just by simple growth of the raw hardware resources power.

2. Language resources and the quantitative approach

The common property of these paradigms are language resources like corpus and lexicon. The length of the corpus is determined as the length of a sequence of words given the sequence of words constituting it, and the size of its lexicon is measured as the total number of different words. An N-gram is a subsequence of length N in a sequence of words, and the same N-gram can appear many times in that sequence, which is the frequency of this N-gram in the text given as a sequence of words. Objects of the initial language processing can be elements of a document such as generalized text, punctuation characters or HTML tags for instance, or tokens as a generalization of notion of words. The complexity of annotated corpus, in which every word is associated with an appropriate tag, is influenced by the number of categories in its tagset, which is usually determined by its structure which is well illustrated in (Džeroski et al. 2000). The structure of tagset is determined by the expressive richness of a corpus, and it is directly influencing the complexity of the tagging problem itself, both manual, or automatic with taggers. For taggers and computational linguistics in general it is very important to have an annotated corpus in a digital form (a good example of a development of such a corpus is given in the MULTEXT-East[14] project). This corpus can serve as a "gold standard" by which a tagger is trained, and by which performance of a tagger is also being measured in terms of machine learning programs. Automatic morpho-syntactical text tagging is currently making more than 95% correct results.

The paradigms mentioned previously differ in the way of describing a language model: the symbolic paradigm is using explicitly given grammatical rules, morphological and other rules, while in the quantitative approach these rules are implicitly given by examples of correct grammatical and language constructions (e.g. with annotated corpus) and its meaning, or by numerical parameters of an appropriate statistical model. A lexicon participates in the quantitative paradigm also as a part of the model, e.g. established together with given N-gram frequencies (also, for statistic parsers see examples 12, 19, 20) TreeBank[18] corpora are very important and interest-

ing, where whole sentences are being tagged by syntax trees). The statistical approach is dating from early Shanon's foundations of information theory, with the shortest and most probable hypothesis (Shannon Claude E., Weaver Warren 1949), and the famous example of a Shannon game with the goal of recognizing the next word in the sentence (1951, where the actual goal is to minimize the entropy). Later in the 1960-ties Solomonoff (Solomonoff 1964), Kolmogorov, Chaitin, Gold (Gold 1967) and others established the foundations of the statistical inductive theory, stochastic context-free grammars and language identification. Noam Chomsky made a remark at that time about the concept of word probability, stating that there isn't any word probability notion that has a reasonable meaning. The further advancement of the statistical theory of machine learning ((Vapnik 1999), (Mitchell 1997)) has later on influenced the better defining of these notions and problems, which in turn influenced making better practical results. Given no input error, a natural language is abundant with problems for this kind of approach – above all, problems which come from multiple semantic values (polysemy, e.g. "I saw a girl with high temperature") and syntactic values of words - e.g. in the syntagm "from time to time" program might tag differently its part "time to time" also as complex form or attribute, or it might tag each single word as a preposition or noun, or the whole syntagm as one complex token.

In this paper some already prepared corpora are used with more details following. There are well known English annotated corpus examples like Brown's corpus (around 1000000 or 1M words in length, created in 1960-ties) and Susanne corpus which was created from a part of Brown's corpus, National British Corpus (BNC), WSJ and PTB (specially the Wall Street Journal articles used for making the Penn Tree Bank corpus). An annotated corpus can be created (by part) manually for a given tagset as it was done in the MULTEXT-East[14] project, where an additional CLOG utility was used for the lemmatization preparation by an automated learning for each tag (Erjavec et al. 2005). The corpus annotation can be also done on an automatically generated tagset (in the unsupervised learning manner). A few corpus examples according to the Wortschatz project[5] as given in table 1 describe some of the basic annotated corpora features (M = million):

Corpus	BNC	CLEF	Wortschatz
Language	English	Dutch	German
Length	100 M	70 M	755 M
Lexicon Size	25706	21863	74398
Tagset size	344	418	511

Table 1 – several corpus examples

3. Different taggers and machine learning methods

In programs which use annotated corpora, the language model is above all defined (during training phase) by a lexicon and tagset, and implicitly also by a training corpus and its learning model which depends on the specific algorithm and implementation. That could be represented by additional N-gram data as with the TnT tagger, rewriting rules (more precisely, rules of changing existing tags according to the annotated text as given context) as with the Brill's tagger, or other parameters. The language model understood in this way differs from the traditional set of formal grammatical and other rules which are also present in such model, but in a different, quantitative form. N-grams as subsequences of length *N* of (consequent) words or tags in a corpus with total occurrence of frequencies in the reference corpus contain implicit information about correct or at least statistically expected constructions and links between words, but making syntax structure dim on the other hand, which is not that important for tagging up to a certain limit.

As mentioned earlier also, first successful tagger implementations have been based on statistical models like the HMM. Some of the most popular and efficient taggers today like TnT are still using basically the same method or an improved form (more in section 4.1). Many improvements are concerned about the tagger's sensitivity towards unknown words (which are not present in the lexicon and training corpus) where a great decrease of efficiency lies (going up to 50%) if no modifications are used (a consequence of the shallow quantitive approach). Some taggers recognize parts of words or suffixes and prefixes (within given maximal number of characters) in the lexicon, or follow and use additional lexeme features, which can often be specific for the given language model or the language itself, or use special lexical rules, as in Brill's tagger case, by which the most probable tag is chosen for an unknown token. The tagging of tokens can be understood as a problem of machine learning ((Mitchell 1997), (Nillson 2005)) and classification, and so in time some non-statistical taggers emerged. The Brill's tagger is a reference example based on rewriting rules that increase performance of the tagger. It is achieved by devising rules that involve words beside tags, compared to the standard HMM which uses only tags (more in section 4.3). The best results so far[1] are achieved by taggers like SVMTool based on the SVM classification (more in section 4.5), Stanford PoS Tagger (Stanford NLP Group[12]) based on Bayes networks and a variant of the Maximum Likelihood Estimation (MLE) method, and LTAG[21] PoS Tagger which is based on a variant of artificial neural networks (ANN). All these taggers use the bidirectional learning instead of the standard learning which is based on the "left to right" direction.

There are many variations of statistical learning which are not based on the HMM and similar models. For instance, MXPOST is a tagger using the model of maximum entropy (MEP, see section 4.4). The statistical approach is somewhat also present in the Brill's tagger (for the initial state), although it is not the essential idea behind it, but rather it is the transformation based learning. Among methods which are not statistically oriented there are also distinguished methods like the learning based on decision trees which is used by Tree Tagger (more in section 4.2), or the memory based learning (used by the well known Memory-Based Tagger - MBT[10]) which adds up to instance based (lazy) learning (where no training is needed), among others.

New solutions and tagging implementations arise often, there are competitions and conferences[2] more or less related to them, where dozens of taggers (and other programs like parsers) get evaluated - currently an ongoing "war" for almost every tenth of performance percent is being fought every day. Apart from accuracy, tagger's speed, its training speed, and the ease of its use are sometimes far more important than few per mill performance advantages. Some of the most popular taggers and associated machine learning methods will be shortly described and evaluated further on in this paper.

3.1 The tagger performance

Different criteria can be important or interesting when measuring performance of a tagger, but the most important performance mark by far is the accuracy of tagging (in functional terms, this is according to the machine learning theory exactly the same performance measure, just as with any other machine learning problem). In this paper, compared to the text length, the error of tagging for the given text will be defined as the number of incorrectly tagged words, while accuracy is defined as the number of correctly tagged words. Beside this property of a tagger, for these programs might also be important the speed of tagging and training (determined by the algorithm complexity and implementation) as mentioned, but not considered with details here. Additional

options and capabilities, learning model options and the ease of use are also important.

The testing of performance is done by the 10-fold cross-validation testing, which is also used by some learning algorithms for the fine parameter adjustment or the optimal model parameter learning. This testing process starts with the dividing of a training corpus in 10 equal partitions, and then training starts on one 9/10 of the corpus, while the testing is done on the remaining 1/10. This process is repeated 10 times for every "9/10+1/10" combination of the training and testing, by which 10 results and error rates are generated in that way, and the average value and its variance is the final measure which corresponds to this method of evaluation. A more sophisticated analysis of the performance would involve statistics about the class of words known to the lexicon, and specially for words unknown to the tagger. In the latter case, the decrease of performance can be up to 30-50%. The overall State-Of-The-Art[1] results for English language are about 96-97%, in which unknown words are not treated separately, and in the former evaluation method there should be always at least about 10% of unknown words in each training corpus. All tagging programs here are tested equally by the same evaluation method, same learning corpora and same 10-fold testing partitions. Statistics gathered here serve principally as a performance indicator for comparing different programs, not as a measure of the distance from state-of-the-art results. The tagger behavior with unknown tokens is expressed in the percent ratios (compared to successfully tagged tokens, and additionally compared to all unknown testing tokens, where the result varies within the boundaries of expected results achieved on similar tests).

4. Chosen solutions and taggers

The evaluation of taggers in this paper is done on texts in Serbian. Taggers were selected as reference academic and non-commercial tagging solutions, and an additional reason of their choice is their availability on the web. The input annotated training corpus and the output result for all these taggers can be in the vertical format where each word and its associated tag are separated by a white space in their own separate line:

```
Bio         Vmps-sman-n---p
je          Va-p3s-an-y---p
vedar       Afpmsnn
i           C-s
hladan      Afpmsnn
aprilski    Aopmpn
dan         Ncmsn—n
.           SENT
```

Example 1a – vertical text format of annotated corpus

and where some taggers can also have lemma in that same line beside the tag for the given word. The text can also be given in the horizontal format, where each word is connected by a separator to its tag, each sentence is given in its own separate line, while words are separated by white space instead of separate lines (Brill's tagger is using "/" as separator, while MXPOST is using "_"):

```
Bio/Vmps-sman-n---p  je/
Va-p3s-an-y---p  vedar/
Afpmsnn
i/C-s  hladan/Afpmsnn
aprilski/Aopmpn  dan/
Ncmsn—n
./SENT
```

Example 1b – horizontal text format of annotated corpus

Each of these programs has an appropriate utility of its own that is used on a given training corpus for building a language model which is usually represented by a set of specific files with available learning options.

4.1 TnT - Trigrams'n'Tags

The TnT[7] tagger (Trigrams'N'Tags), whose author is Thorsten Brants, Saarland University in

Germany, Computational Linguistics and Phonetics department, has emerged in the period 1993-2000 (Thorsten Brants 1999, 2000). This tagger is using machine learning based on the HMM which is a case of Bayes learning and it represents an example of the class of machine learning algorithms that use probability estimation given by the acyclic directed graph of computation (Bayesian network) defined by the conditional dependence relation among random variables as nodes. The computing problem then comes down to the classification problem with such a probability computing network where one finds the most probable sequence of tags for the given sequence of words. The HMM is a Bayesian network whose node set is divided in two partitions: the set of hidden events or states, and the set of observations or symbols. Each state has different observations which depend only on that state in the sequence of states which is described by the symbol emission probabilities, while the hidden state can depend on all previous hidden states in the sequence. Order of the HMM model is defined by the number of previous states on which current hidden state depends (for instance, with second order HMM each hidden state depends only on two previous adjacent hidden states). In the tagging problem, the set of hidden states represents tags, while the set of observations represents words in the text. The HMM model is given by probability distributions of initial states and transitions from one state to another. Basic questions in this case are: how to determine the most probable sequence of hidden states for the given sequence of observations (the decoding problem), and how to learn model parameters in order to maximize the probability of sequence of observations in the given training set (the learning problem). TnT is an example of a successful statistically based (stochastic) tagger, based primarily on the Viterby's algorithm for decoding second order HMM, and on N-grams and interpolation (smoothing) for the problem of normalization for learning using the Baum-Welch algorithm, (Rabiner 1989), (Welch 2003).

TnT is truly an example of a tagger that is comfortable to use, learning fast and also tagging fast. This system is using several types of data files. For the learning it is sufficient to have an annotated input corpus in the vertical format whose file name having no extension is later used as the model name. Each sentence must be ended by a token with tag SENT, or by a tag specified with option "-st". This system uses several data files generated by the learning which can be modified afterwards. There is a lexicon as a file in the vertical format having at least 4 columns separated with white spaces: the first column contains the word (token), the second column contains its frequency in the training corpus, the third column contains the appropriate tag, and the fourth column contains the frequency of that tag. Many tags for the given word are allowed, and each tag is followed by its frequency, while the sum of such frequencies is equal to the word's frequency. Beside this file, there is also an N-gram file, a tagset list, and a special file for tag mapping which can be used to change tag names in the output file differently from the existing source model.

This program does not support lemmatization. Its availability and licensing is somewhat different from what is usual with the open source programs. One can obtain download access for the programs for learning and tagging (and maybe the source code) only by addressing the author directly via e-mail, while it is generally available for academic and non-commercial purposes for free. It is created and tested before all on Posix (Unix, Linux) platforms.

4.2 Tree Tagger

The author of Tree Tagger (TT) is Helmut Schmid[6] (Schmidt 1994), Institut fuer maschinelle Sprachverarbeitung, Stuttgart University in Germany. It was created during 1994-1996. as part of the TC project. Tree Tagger is not a true sto-

chastic tagger example compared to other known programs that use the HMM because it differs in the way it is learning, yet its tagging is done by Viterby's algorithm, too. The decision tree learning algorithm is used for the training (similar to ID3 and C4.5 algorithms), with the aid of trigram model. The decision tree is pruned after being built by the information gain rule, which is important for overcoming the learning problem of over-fitting and and for making the optimal algorithm performance.

The TT lexicon consists of three parts: the full-form lexicon, the suffix lexicon with suffices up to 5 characters in length, and the given default value used if word is not found in the former parts of lexicon. If a word is not found in the full-form lexicon during the tagging process even after changing its capitalization, the suffix lexicon is then searched by using suffix decision tree. The suffix decision tree is built during the training process just as the previously described tagging decision tree is built: for each word annotated with a tag from the open class tagset, character by character of the word suffix. Additionally, each parent node has a default child node marked with the probability which added to the sum of other children is equal to 1, and if it is the only branch left it is then pruned. Leaves are labeled with probability vectors of tags with the given suffix.

For tagging it is sufficient to provide the input file and the parameter file produced in the training. The list of tags in the open class is the list of tags available for the unknown words. The tagging can be done without lemmas in the lexicon, for instance by giving the same "dummy" value to all the words in the lexicon, e.g. "-". The input file is in the vertical format, and so is the output file. The learning and model generation is quite simple, though lexicon and open class data files have to be prepared additionally before that (eg. by additional script[22], refer to Section 5) beside the annotated training corpus. The model produced is stored in a binary parameter data file. Here is a part of a lexicon file given in the Example 1:

```
prisloni   Vmia3s-an-n---e
prisloniti
talenta    Ncmsg--n
talenat
gutljaju   Ncmsl--n
gutljaj
nagnut     Vmp--smpn-n---e
nagnuti
Vinstonovo Aspnsn
Vinstonov  Aspnsa
Vinstonov
```

Example 1 – a part of a lexicon in the Tree Tagger

Tree Tagger supports lemmatization opposed to other taggers described here. The licensing is similar to the TnT licensing - it is free for non-commercial and academic use, and it also is not an open source program (unless an agreement is made somehow directly with the author himself), while the program is freely available for the download. It is available both for Posix and for Windows operating system.

4.3 Brill – Rule Based Tagger (RBT)

Eric Brill[8] (Brill 1992) is one of the pioneers of the transformation learning (Transformation-Based Learning) and the creator of the RBT tagger (Rule Based Tagger) founded on this model of learning. It was developed in the period 1992-1994 and it has set many standards for taggers at the time. In some parts it is realized as Perl scripts because Brill promotes the idea that this is a perfect programming language and tool for linguistic researches due to its abundance and capabilities of sequence processing and regular expressions, but this is also the reason why his implementation of training is very slow. This system represents one of the basic tagger examples which are not mainly statistical, but instead, this tagger is based mainly on the incremental contex-

tual transformation rule learning. It is using the rewriting rules which achieve the performance improvement by choosing result with the highest estimation of accuracy, or in other words, with the lowest tagging error. In essence, the tagging is done in two stages: in the first stage, the starting state of the input corpus is initiated by using its lexicon, and intrinsic and lexical rules of the model. In order to overcome the HMM limitations of using only tag N-grams, Brill proposed and invented transformations which refer both to tags and words. Then, in the second stage, the tagging performance is increased by applying contextual rules of the trained model. These rules are similar to lexical rules, with a second tag added which replaces the source tag - all rules suppose one of the several types of conditions, and in essence change the tag after being triggered if the condition is true. The learning consists of the lexical rules learning phase, and the context rules learning phase driven by the least tagging error. In the latter phase the error is estimated iteratively for each candidate rule before and after the transformation, and then the best available rule by the mark is chosen and put into the rule set until there is no applicable rule left with the error under the default error rate threshold. This tagger has "built-in" lexical rules for unknown words and for words which begin with a capital letter: they are considered as proper nouns automatically (taken as nouns, otherwise), after which the unknown words subprogram is applied. In short, taking suffixes up to 4 characters in length and searching for the known sub-words in the lexicon, and learning new rules if found.

The model data generated by the learning is stored in text files. The input training annotated corpus is expected in the horizontal format, and so is the generated output. The learning process is not simple, and it is done in several stages: the tokenization (given in the vertical format by the Penn Tree Bank standard, for example: "The/det cat/noun sat/verb on/prep the/det mat/noun ./."), the dividing (splitting) of the annotated training corpus into the training corpus for lexical rules and into the partition for contextual rules, separately. One of the limitations of this program is the slow learning speed, but this can be controlled by appropriate setting of the error rate threshold. This tagger is intended to be used incrementally by increasing and changing the set of rules with new and smaller training corpora. The learning is stopped when the error reaches the threshold value hard-coded in the very Perl code, and the whole learning procedure is far too complicated and demanding compared to other taggers. The detailed description and instructions for this and other tools are available for download[22].

This tagger does not support lemmatization. RBT and its source code are available for free without limitations, but under a license which is of an open source type. Though there are implementations for different platforms, including Java as a platform, the learning in basic form is not supported on Windows operating system.

4.4 MXPOST

Adwait Ratnaparkhi, working currently in the Yahoo! company, is the author of the MXPOST[11] (MX shortly) tagger which he developed during 1996-1999. This tagger is using a statistical concept of learning and a method of Maximum Entropy Principle (MEP shortly, (Adwait Ratnaparkhi, 1996)), which is under some conditions dual to the previously mentioned MLE. This learning concept maximizes the conditional entropy of the learning model on a given training set instead of maximizing the likelihood. A bigger entropy of the learning model means that there is more information coded in the model parameters. This program (MX) is closer to the Brill's tagger and SVMTool by its slow learning speed, which is still slower for an order of magnitude in case of two latter taggers. The training of this tagger is always done in 100 iterations regardless of whether the learning performance is reached or not.

The data file with learning examples has similar horizontal structure to the one that Brill's RBT is using, only that it uses "_" sign as separator for token and its tag. The output file is in the same format. The learned model data is stored into text files in a separate model directory.

This program does not support lemmatization. The original implementation is realized as Java program and it is still available as a somewhat test version. MX is free for non-profit purposes, freely available for download and it is not open source.

4.5 SVMTool

Authors, Jesus Gimenez and Lluis Marquez, created SVMTool[9] during 2000-2004 under the TALP Research Center NLP group, Universitat Politècnica de Catalunya. This is one of the programs by which a "state-of-the-art" result was reached based on WSJ corpus, and a program with the richest range of options for the training and influence on the process of tagging. It supports, among other features: bidirectional learning and tagging, cross validation of results, special language exceptions and dictionary corrections, different learning models, and other. This leads also to one of the main deficiencies of this tagger, which is the inefficient learning. In short, the idea is to code different text features into a sequence of frames or windows of a given width measured by tokens, centered around the token that is currently analyzed, and with the default width 7. There are features of words (whether a word is capitalized or not, punctuation and special signs, numbers, etc), and also N-grams, tags, and other features. The vector of such features is then classified by the SVM algorithm, where each class represents an appropriate tag. Vapnik prepared theoretical ideas for the Support Vector Macihne classifier (SVM, (Vapnik 1999)) already in 1963. This classification method is a method of maximum margin hyper-plane classification, because the goal for these classifiers is to determine the hyper-plane which separates the classification space into two partitions (half-spaces) in that way that the nearest distance between this margin hyper-plane and a class instance is maximized. SVM additionally minimizes the classification error by using numerical solving of a quadratic programming (QP) problem. SVMTool uses one of the most simple and most efficient implementations, SMO (Sequential Minimal Optimization[9]), (Platt 2000). SVM methods can be easily transferred into non-linear methods by using non-linear kernels (Aizerman's „Kernel trick") and represent one of the most efficient known classifiers today.

The input training annotated corpus and the output result are in vertical format, with the plain space instead of white space. For the learning it is necessary to set configuration parameters in a text file, either with a short or with a complex template available. More than two hours for a 2500 word corpus were not enough for it to converge with the longer template, while with the shorter template it takes only slightly less than Brill's tagger. The more advanced usage option considers detecting SVM model parameters based on the fine tuning by validation on training set, but the main question is if the training lasting several times longer than usual just for a per mil or few of higher accuracy is really justified.

This tagger does not support lemmatization. Authors are offering the program in Perl under the open source LGPL and FSF license and it is not fairly tested on the Windows operating system. It is freely available for the download, but the program relies on an external tool SVMLight whose author is Thorsten Joachim, which is also open source, but author's permission is needed for any commercial purpose.

5. The evaluation of selected taggers

The process of evaluation of taggers selected for testing in this paper is done automatically with a script that enables this in following steps:

• input corpora data files are transformed into vertical text form with appropriate XSLT transformations giving the output result in out1.txt for whole corpus;

• initialization of stat_*TAG*.txt files with statistical counters, where *TAG* belongs to set {*TNT, TT, RBT, MX, SVM*} of chosen taggers;

• main loop, script is executed in 10 iterations, for variable *n* in range from 0 to 9

• by using additional script cross-tab10.sh the training corpus learn.txt and testing corpus test.txt is prepared as n-th "9/10+1/10" partition of whole corpus;

• for each tagger *TAG* gets transformed corpus and other needed files prepared using AWK scripts, training is started afterward and then tagging of unannotated corpus for testing;

• tagging result in vertical form of tagger *TAG* is in file test_*TAG0*.txt, and earlier prepared annotated test corpus is in file test_*TAG*.txt; vertical form enables easier comparison between the resulting annotation and the source;

• for each tagger *TAG* counters are updated for unknown words using additional scripts unknown.sh and unknown.awk (incorrectly tagged word are searched in the training lexicon, and if not found there they are treated as unknown);

• aggregated statistics are made using additional scripts report1.awk and report2.awk;

• file clean-up is done for the data and temporary files created during the training process;

This automated testing process and the tagger evaluation offer the opportunity of easier extending with additional taggers in later evaluations and their comparison with earlier testing results, making the evaluation process and statistics computation less prone to errors, and above all, it represents a convenient working environment for experimenting with taggers. The possibility of the integration of different taggers and their training parameters and models is also very interesting, even for different tagsets and corpora. For instance, as described in (Jakub Zavrel, Walter Daelemans 2000), with the Bootstrapping method the learning with new tagset is achieved with a much smaller number of examples, and also its tagging performance is not smaller or at least slightly better than the performance of each single tagger. A simple way of the tagger integration with some performance improvement is done by using the voting method – a tag is chosen as the result if it's selected or given the best marks by the largest number of taggers. The further approach might be the choice made by a second level machine learnt experience based on different taggers, their tagsets, parameters and input.

All these taggers were installed by their installation manuals without any bigger difficulties. The total script execution time for the training and tagging is based on the Intel(R) Core™2 Duo CPU/T7700@2.40GHz processor platform, which is described later in this paper. Linux Fedora FC8 is selected as the OS platform for testing because many programs (TnT, RBT) do not support learning on non-Posix platforms or have some other limitations. However, the main reason for this choice is the comfortable working with strings, regular expressions, command piping and numerous handy tools for the text processing like word and line counting, difference matching, powerful scripting languages and other. All scripts developed here are available for downloading[22], together with reference papers with detailed explanations, and a completely prepared tagging environment with scripts.

5.1 Corpora

The basic property of an annotated training corpus is its length, or the number of tokens in the sequence of annotated tokens which represents the corpus text. But, the size of the set of tokens which constitute a lexicon based on a corpus can be more statistically important because the word distribution is not changing much with

a big enough length according to Zipf's law. A very important property of a corpus is its annotation tagset size. The tagset can be complex and of big cardinality, which influences the most performance of a tagger. The tagset size depends on the tagging concept, meaning what is expected to be achieved by the tagging, and it also depends on the language itself. For instance, Serbian language is certainly more demanding in that manner than English if the language case is used in tagging, too. Beside that, different taggers are differently sensible to the input corpus information quality. The smaller but certain number of errors can significantly decrease the tagging performance (for instance, an incorrectly tagged token in the training corpus, or a punctuation missing at the end of a sentence in the training corpus can produce tagging errors later). Some minor errors can be detected and manually corrected if script shows an error during the test, others can be corrected by a simple lexical processing – but many of them remain "hidden". All taggers are trained with their default training parameters, without any special changes (and are equally treated in that way). Of course, some taggers might have been able to give a better performance, but only with the additional effort and specific customizations.

Two types of input data files were used throughout the process of evaluation for building training corpora. The first type has the XML structure by CES[15] standards, having in short:

• each word with an XML tag `mw` that has the attribute:

- id for a unique identifier,
- lex giving lexeme or token, word,
- lemma for lemma of the given token,
- tag for its tag.

• each sentence with an XML tag seg and the attribute id which identifies it uniquely

• each page marked by an XML tag p, and the division by an XML tag div

Example of such a data file and its structure is given in the Example 2 which is the part of the file `02HP-SR-Lemma.xml`, and it is transformed into a needed plain text format with an appropriate XSLT transformation by using data.xsl from the Example 3:

```
<Annotation type="morpho">
  <body>
    <div>
      <head>
        <mw id="mw__1 " lex="ZAKLJUCAK"
lemma="ZAKLJUCAK" tag="?"/>
      </head>
      <p>
      <seg id="n1">
        <mw id="mw_1_1 " lex="Na" lemma="na"
tag="PREP+p4"/>
        <mw id="mw_1_2 " lex="međunarodnom"
lemma="međunarodni" tag="A"/>
        <mw id="mw_1_3 " lex="planu" lemma="plan"
tag="N"/>
        <mw id="mw_1_4 " lex="poslednjih"
lemma="poslednji" tag="A"/>
        <mw id="mw_1_5 " lex="decenija"
lemma="decenija" tag="N"/>
        <mw id="mw_1_6 " lex="preduzeti"
lemma="preduzeti" tag="V+Perf+Tr"/>
        <mw id="mw_1_7 " lex="su" lemma="jesam"
tag="V+Imperf+It+Iref"/> ...
      </seg>
      <seg id="n2"> ...
```

Example 2 – the XML structure of corpus 1 and corpus 2

```
<?xml version="1.0" encoding="UTF8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" omit-xml-declaration=
                    "yes" indent="no"/>
  <xsl:template match="//seg">
    <xsl:for-each select="mw">
      <xsl:value-of select="@lex"/>~
      <xsl:value-of select="@tag"/>~
      <xsl:value-of select="@lemma"/>~
    </xsl:for-each>*SENT*
  </xsl:template>
</xsl:stylesheet>
```

Example 3 – data.xsl

The second type of data files used for the evaluation are prepared according to TEI[16] rec-

ommendations. It is using an XML notation, too, but with some differences in the structure: XML tags and attributes are not the same, and lexemes are not given as attributes but as values of the XML tag w. The TEI standard includes a mandatory header with the bibliographic data, information about code page, about the structure of the file and many other meta data, and also many additional structures (e.g. describing the semantic structure of the tagset).

```
<TEI.2 id="Osr" lang="sr">
  <teiHeader creator="CK" status="update" ...
id="Osr.teiHeader">
    <fileDesc>
       <titleStmt> ... </fileDesc>
    <encodingDesc>
       <projectDesc> ... </encodingDesc>
    <revisionDesc> ... </revisionDesc>
</teiHeader>
<text lang="sr" id="Osr.">
  <body>
    <div id="Osr.1" type="part" n="1">
    <div id="Osr.1.2" type="chapter" n="1">
      <p id="Osr.1.2.2">
        <s id="Osr.1.2.2.1">
          <w lemma="biti" ana="Vmps-sman-n---p">Bio
</w>
          <w lemma="jesam" ana="Va-p3s-an-y---p">je
</w>
          <w lemma="vedar" ana="Afpmsnn">vedar
</w>
          <w lemma="i" ana="C-s">i</w>
          <w lemma="hladan" ana="Afpmsnn">hladan
</w>
          <w lemma="aprilski" ana="Aopmpn">aprilski
</w>
          <w lemma="dan" ana="Ncmsn--n">dan</w>
          <c>;</c>
...  <!-- pb n=283 -->
      </p>
    </div>
  </body>
</text>    </TEI.2>
```

Example 4 – the XML structure of corpus 3 according to TEI

An example of this form is given in the Example 4 as part of the file `oana-sr.xml`. Appropriate XSLT transformation which is producing output similar to the previous data form is used in a similar manner. After that, other processing tasks follow in awk scripts which bring the final vertical form of the annotated training corpus.

• Three corpora are used for the evaluation which will be referred in this text as corpus 1, corpus 2 and corpus 3 (source files on which they were built are available for download[22]):

• corpus 1 is created from the file `01HP-SR-Lemma.xml` which represents part of the document "Helsinške sveske br. 15, nacionalne manjine i pravo"[0] in CES format which also have `02HP-SR-Lemma.xml` and `03HP-SR-Lemma.xml` files

• corpus 2 is created by the concatenation of files `01HP-SR-Lemma.xml`, `02HP-SR-Lemma.xml`, `03HP-SR-Lemma.xml`, and additionally `Radiodif-SR-lemma.xml` and `Radionica-SR-lemma.xml` which contain Serbian Radio diffusion Law[0] and materials from UNDP workshops; they are all in CES format and of size 1-2MB

• corpus 3 is created from the file **`oana-sr.xml`** in TEI form of Orwell's "1984", with the size around 4.5MB

The semantic structure of a tagset which is also known as MSD, the morpho-syntactic description, that is used in corpus 1 and corpus 2 is not the same as in the corpus 3. Namely, corpora 1 and 2 have only types of words coded and each token there has its lemma, while the corpus 3 is coded with more details[22], according to morpho-syntactical description developed in the MULTEXT-East project. The corpus 3 consists of the text of the Orwell's novel „1984" (Krstev Cvetana et al. 2004) which is developed in the course of the European MULTEXT-East project[13] as a parallel multi-lingual corpus. By the TEI standard, MSDs are given in the libraries of feature structures. For example, as part of the previously mentioned MULTEXT-East

project[14] a verb is described with 15 features, of which two with possible values are as given in the Example 5:

```
Verb (V)
**** **** **** **** **** **** **** ---- ---- ---- ---- ---- ---- ---- ---- ----
PoS Type VFrm Tens Pers Numb Gend Voic Neg Def Cltc Case Anim Clt2 Aspt
**** **** **** **** **** **** **** ---- ---- ---- ---- ---- ---- ---- ---- ----
= ========= ========= = EN RO SL CS BG ET HU HR SR
P ATT VAL C x x x x x x x x x
= ========= ========= =

1 Type main m x x x x x x x x x
auxiliary a x x x x x x x x x
modal o x x x x x x x x
copula c x x x x x x
base b x
- ---------- ---------- -
2 VForm indicative i x x x x x x x x x
subjunctive s x
imperative m x x x x x x x x
conditional c x x x x x x x
infinitive n x x x x x x x x
participle p x x x x x x x x
gerund g x x x x
supine u x x
transgressive t x
quotative q x
- ---------- ---------- -
...
```

Example 5– MSD structure

As a part of this project a special tagger ToTaLe[14] was developed (Erjavec 2005), and also TnT and MBT were used.

6. Results

Results generated with the previously described process for all three corpora are presented in Tables 2a, 2b and 3 in which mark * is used to determine results on known words, mark ** is used on unknown words, while mark *** is related to the training set. Length, size and lemmas refer to the appropriate corpus properties expressed as the number of distinct words, where K stands for 1000 words. Table 2a is a general review about test and corpora used for all taggers, while Table 2b shows the performance rate for each tagger independently from the word class together with rate of unknown words among unsuccessfully annotated ones (**, a smaller percent is better). The Table 3 is showing a better tagger performance concerning unknown words because it gives the proportion of incorrect unknown words and all unknown words in the test corpus (**), and for known words the proportion of correctly tagged words and the rest of the test corpus is given (*, a larger percent is better). This table shows a better perspective about tagging results for word classes compared to previous, but it also has less meaning about the overall tagging performance. Corpora 2 and 3 are not much different in the number of tokens, but these differ significantly in the number of tags in the tagset.

Tagger / Corpus		**Corpus 1**	**Corpus 2**	**Corpus 3**
	length	7.5K	75K	105K
	size	2.5K	11K	18K
	n/o lemmas	1.6K	5K	7.6K
	n/o tags	79	129	908
Overall test duration		22min.	9h : 50min.	5 days, 1h : 29min.
*** average	size	2290–2378 (2335)	9766–10952 (10368)	16550-17372 (16919)
	n/o tags	73 – 79 (77)	120 – 129 (126)	840 – 897 (884)

Table 2a – basic properties of the tests and corpora

Tagger/ Corpus		**Corpus 1**		**Corpus 2**		**Corpus 3**	
		%	**% ****	**%**	**% ****	**%**	**% ****
TT	average	85.44	64.93	**94.39**	33.30	79.65	35.05
	std. dev.	3.90	3.87	1.86	20.25	1.92	1.85
SVM	average	84.93	64.70	94.27	38.02	85.24	34.67
	std. dev.	3.60	5.51	1.72	22.61	1.87	2.27
TnT	average	**86.18**	67.65	94.11	37.42	**85.47**	32.26
	std. dev.	3.60	4.33	1.65	21.85	1.75	2.19
MX	average	82.69	54.01	92.78	29.43	82.07	28.62
	std. dev.	3.84	2.49	1.79	16.93	1.79	16.93
RBT	average	84.96	**82.15**	93.14	**47.24**	85.20	**37.96**
	std. dev.	4.34	4.32	3.21	26.29	1.95	1.97

Table 2b

The tagging speed is not measured here, but the measure of total time for the training and

evaluation is given (overall test duration). TnT is certainly the champion of both tagging and learning speed, and his performances have proved as best, and so did its simplicity of usage.

Tagger	Corpus 1		Corpus 2		Corpus 3	
	* %	** %	* %	** %	* %	** %
TT	98.37	56.71	97.53	**71.49**	91.78	36.79
SVM	98.29	55.18	97.69	67.17	93.98	54.60
TnT	**98.54**	**57.50**	97.57	67.17	93.86	**58.36**
MX	97.43	57.01	96.48	69.09	92.06	54.26
RBT	**99.10**	**43.96**	**97.97**	**48.17**	**94.24**	50.33

Table 3

The script which divides the whole corpus into partitions for learning and testing is reading the corpus in sequential order, by the equal number of sentences for each partition. This maybe isn't ideal at the first glance, and it could be improved by a random choice of sentences as it is done in the divide-in-two-rand.prl in RBT, or by using corpora of a bigger length. In the test procedure described here, some of the testing partitions in the second corpus were practically left with no unknown words, and that caused the unusually big standard deviation for unknown words – on the other hand, that made a more realistic test. Thorsten gives[7] results with a standard deviation of 0.13 for Penn Treebank (0.76 for Susanne Corpus in English, 0.29 for NEGRA corpus in German), which shows a standard deviation comparable with results presented in this paper. Of course, such a comparison with statistical data in other papers is not a complete way to prove that conclusions are correct here, but it describes well important tagger properties. Among the given references in this area there are more detailed proofs about nature of taggers and their performance.

7. Conclusions

Results obtained from (Erjavec et al. 2005) are given here in Ttable 4 for two taggers and are based on Orwell's "1984"[13] as a MULTEXT-East resource, too. These results are comparable with results presented in this paper for TnT tagger both for unknown words (having here even somewhat better result), and for known words. Similar results on the very same corpus 3 bring even more sense to a comparison of the tagging results in this way. Although the greater number of tags in the corpus 3 compared to corpus 2 had significant impact on its performance decay, while corpus 2 has achieved performance close to the state-of-art results (or almost the best results), still the test with corpus 3 is more realistic and therefore more usable.

	TnT	MBT
Known	93.55%	93.58%
Unknown	60.77%	44.45%

Table 4

Judging upon the Table 2, the TnT tagger has shown better performance for corpora 1 and 3, while the Tree Tagger has "won" in case of corpus 2, but by a per mil. If Table 3 is considered, one can find that RBT is making a better performance on corpora 1 and 2, and so is Tree Tagger better on corpus 3, concerning unknown words. It can be also concluded from that table that Brill's RBT has better results than other taggers in tagging known words, but this should not be treated as an important performance indicator because differences are too small. The final conclusion in general might be that Tree Tagger does somewhat better with smaller tagsets, but in all other cases TnT is obviously much better and is making an "easy victory". In the end, all these differences might be also considered as very small and SVM is also very close to all these good results.

Each of these programs can be tuned and additionally customized up to some point in order to achieve somewhat better results where SVM-Tool has by far more capabilities than all other tested taggers, but its training performance becomes very poor depending on the training corpus. BNC corpus is about 1000 times longer than corpus 3, while having a three times smaller tag-

set and a slightly smaller number of tokens than corpus 3 which is even more important – so, the tagset size is most important. Corpora such as the corpus 2, with a smaller tagset (and even a smaller length) are ideal for exploring several taggers and methods, and for comparative testing of their performance, but they are not good for a real exploit. The unexplored challenge is to reveal the maximally achievable performance reach of all these taggers on Serbian.

The optimal corpus length and size as a training set is one of the important achievements of statistical theory of machine learning – it is shown that it depends only on the desired error magnitude and the learning probability, and also on the size and structure of the hypothesis space (details available in (Vapnik 1999) and (Nillson 2005)). The corpus (like corpus 3) is certainly more important for finer investigations and sophisticated performance, while having in mind over-fitting in learning (a too big training set can degrade performance and generalization abilities) which is handled good by all tested taggers themselves implicitly with different mechanisms of their own.

(internet pages were accessed from November 2008-2010)

0 http://www.anem.rs/download/files/cms/attach?id=4 ISBN 86-7208-065-3 http://www.helsinki.org.rs/serbian/doc/sveske15.zip

1 State-of-the-Art results: http://aclweb.org/aclwiki/index.php?title=POS_Tagging_(State_of_the_art)

2 Tagger Competitions and Conferences, Resources: http://www.aclweb.org http://www.lrec-conf.org http://alias-i.com/lingpipe/web/competition.html http://ltrc.iiit.ac.in/nlpai_contest07/cgi-bin/index.cgi

3 http://www.no-free-lunch.org

4 http://www.lti.cs.cmu.edu/Research/Pangloss/

5 http://wortschatz.uni-leipzig.de/~cbiemann/pub/2007/BiemannGiulianoGliozzoRANLP07.pdf

6 TT http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html

7 TNT http://www.coli.uni-saarland.de/~thorsten/tnt/ http://coli.uni-sb.de/~thorsten/tnt/

8 Brill http://www.cst.dk/download/tagger/ http://www.tech.plym.ac.uk/soc/staff/guidbugm/software/RULE_BASED_TAGGER_V.1.14.tar.Z http://www.ling.gu.se/~lager/Home/brilltagger_ui.html

9 SVM: http://www.lsi.upc.edu/~nlp/SVMTool/ http://svmlight.joachims.org/

10 MBT: http://ilk.uvt.nl/mbt/

11 MXPOST: ftp://ftp.cis.upenn.edu/pub/adwait/jmx/ http://www.inf.ed.ac.uk/resources/nlp/local_doc/MXPOST.html

12 Stanford NLP Group – statistical Parser http://nlp.stanford.edu/software/lex-parser.shtml

13 Orwell http://nl.ijs.si/ME/bib/mte-nlprs01.pdf

14 MULTEXT-East: http://nl.ijs.si/ME/V3/msd/ http://nl.ijs.si/et/ http://nl.ijs.si/et/talks/SFB441/tue-slides/ http://langtech.jrc.it/Documents/LTC-2005_Multilingual-corpus-compilation_Erjavec-et-al.pdf

15 CES: http://www.cs.vassar.edu/CES

16 TEI: http://www.tei-c.org http://bcdlib.tc.ca/tools-standards.html

17 Stanford Tagger http://nlp.stanford.edu/software/tagger.shtml

18 treebanks: PTB http://www.cis.upenn.edu/~treebank/

ICE http://www.comp.leeds.ac.uk/amalgam/tagsets/ice.html

19 Michael Collins 1998. PhD http://people.csail.mit.edu/mcollins/

ftp://ftp.cis.upenn.edu/pub/mcollins/PARSER.tar.gz

20 Dan M. Bikel http://www.cis.upenn.edu/~dbikel/

21 LTAG POS Tagger http://www.cis.upenn.edu/~xtag/spinal/

22 download: http://users.hemo.net/shoom/taggers.tar.gz

http://users.hemo.net/shoom/tag.pdf

References

Brill Eric. 1992. A simple rule-based part of speech tagger. Speech and Natural Language: Proceedings of a WorkshopHeld at Harriman, New York.Morgan Kaufmann Publishers, Inc., San Francisco, Californi, pp. 112–116.

Dreyfus L. Hubert, Haugeland John. 1974. An Exchange On Artificial Intelligence. http://www.nybooks.com/articles/9452

Džeroski Sašo, Erjavec Tomaž, Zavrel Jakub. 2000. Morphosyntactic Tagging of Slovene: Evaluating Taggers and Tagsets. 2nd International Conference on Language Resources & Evaluation (LREC), pp. 1099-1104.

Erjavec Tomaž, Ignat Camelia, Pouliquen Bruno, Steinberger Ralf. 2005. Massive multi lingual corpus compilation: Acquis Communautaire and totale. Proc. of 2nd Language & Technology Conference, pp. 32-36.

Gold E. Mark. 1967. Language identification in the limit, Information and Control, 10:447—474. http://www.isrl.uiuc.edu/~amag/langev/paper/gold67limit.html

Jakub Zavrel, Walter Daelemans. 2000. Bootstraping a Tagged Corpus through Combination of Existing Heterogeneous Taggers. Proceedings of the second international conference on language resources and evaluation (LREC), pp. 17-20.

Krstev Cvetana, Vitas Duško, Erjavec Tomaz. 2004. Morpho-Syntactic Descriptions in MULTEXT-East -

the Case of Serbian. In Informatica, No. 28, The Slovene Society Informatika, Ljubljana. pp. 431-436.
http://www.matf.bg.ac.yu/~cvetana/biblio/mtesr-inform04.pdf

Leech G. 1987. Garside R. and Sampson G. The Computational Analysis of English: A Corpus-based Approach. London: Longman
http://www.comp.lancs.ac.uk/computing/research/ucrel/claws/

Mitchell M. Tom. 1997. Machine Learning. McGraw-Hill. ISBN 0-8493-1232-9

Nillson J. Nils. 2005. Introduction To Machine Learning. Stanford unpublished textbook draft
http://ai.stanford.edu/people/nilsson/mlbook.html

Platt C. John. 2000. Fast Training of Support Vector Machines using Sequential Minimal Optimization. MIT Press, ISBN:0-262-19416-3 pp. 185-208
http://research.microsoft.com/~jplatt/abstracts/SMO.html

Rabiner R. Lawrence 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proc. IEEE 77(2), pp. 257-286.

Ratnaparkhi Adwait. 1996. Maximum Entropy Model for Part-Of-Speech Tagging. Proceedings of the Empirical Methods in Natural Language Processing Conference, University of Pennsylvania, pp. 133-142.

Schaffer Cullen. 1994. A conservation law for generalization performance. International Conference on Machine Learning, pp. 295-265.

Schmidt Helmut. 1994. Probabilistic part-of-speech tagging using decision trees. In Proceedings of the International Conference on New Methods in Language Processing, Manchester, UK, pp. 44-49.

Shannon Claude E., Weaver Warren. 1949. The Mathematical Theory of Communication. The University of Illinois Press, Urbana, Illinois, ISBN 0-252-72548-4

Solomonoff J. Ray. 1964. A Formal Theory of Inductive Inference. Information and Control, Part I: Vol 7, No. 1, pp. 1–22.
http://world.std.com/~rjs/pubs.html

Thorsten Brants. 1999. Cascaded Markov Models. Proceedings of 9th Conference of the European Chapter of the ACL (EACL-99), Bergen, pp. 118-125.

Thorsten Brants, 2000. TnT - A Statistical Part-of-Speech Tagger. 6th Conference on Applied Natural Language Processing, Seattle, Washington. pp. 224-231.

Vapnik N. Vladimir. 1999. The Nature of Statistical Learning Theory, ISBN 978-0-387-98780-4

Welch R. Lloyd. 2003. HMM and the Baum-Welch Algorithm, IEEE IT Society Newsletter.
http://www.itsoc.org/publications/newsletters/past-newsletters/itNL1203.pdf/view