Looking under the Hood of Stochastic Machine Learning Algorithms for Parts of Speech Tagging

Jana Diesner Kathleen M. Carley

July 2008
CMU-ISR-07-131

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Center for the Computational Analysis of Social and Organizational Systems
CASOS technical report.

This work was supported in part by the National Science Foundation under grants: No. ITR/IM IIS-0081219, NSF 0201706 doctoral dissertation award, and NSF IGERT 9972762 in CASOS. Additional support was provided by CASOS and ISRI at Carnegie Mellon University. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation, or the U.S. government. We are grateful to Alex Rudnicky from CMU for providing the training data to us and to Yifen Huang, CMU, for discussing the project with us.

Keywords: Part of Speech Tagging, Hidden Markov Models, Viterbi Algorithm, AutoMap

Abstract

A variety of Natural Language Processing and Information Extraction tasks, such as question answering and named entity recognition, can benefit from precise knowledge about a words' syntactic category or Part of Speech (POS) (Stolz, Tannenbaum et al. 1965; Church 1988; Rabiner 1989). POS taggers are widely used to assign a single best POS to every word in text data, with stochastic approaches achieving accuracy rates of up to 96 to 97 percent (Jurafsky and Martin 2000). When building a POS tagger, human beings needs to make a set of decisions, some of which significantly impact the accuracy and other performance aspects of the resulting engine. In this paper we provide an overview of these decisions and empirically determine their impact on POS tagging accuracy. We envision the gained insights to be a valuable contribution for people who want to design, implement, modify, fine-tune, integrate, or simple reasonably use a POS tagger. Based on the results presented herein we built and integrated a POS tagger into AutoMap, a tool that facilitates Natural Language Processing and relational text analysis, as a stand-alone feature as well as an auxiliary for other tasks.

Table of Contents

1. Introduction

Part of Speech Tagging (POST) assigns a single best part of speech (POS), such as noun, preposition or personal pronoun, to every word in a text or text collection. What is the knowledge about words' lexical category useful for? First, a large variety of Natural Language Processing (NLP) and Information Extraction (IE) tasks benefit from accurate knowledge about words' lexical categories, such as:

- Stemming (conversion of terms into their morphemes) (Krovetz, 1995; Porter, 1980)
- Named Entity Extraction (identification of relevant types of information that are referred to by a name, such as people, organizations, and locations) (Bikel, Schwartz, & Weischedel, 1999)
- Anaphora resolution (conversion of personal pronouns into the actual entities that those pronouns refer to) (Lappin & Leass, 1994)
- Creation of positive (thesaurus) and negative (delete list) filters (Diesner & Carley, 2004)
- Ontological text coding (classification of relevant types of information according to an ontology or taxonomy) (Diesner & Carley, 2008)

Second, POS are often used as one feature for machine learning tasks that involve text data (Arguello & Rose, 2006; Bikel et al., 1999).

What is the challenge in POST? While many words can be unambiguously associated with one tag, e.g. *computer* with noun, other words match multiple tags, depending on the context that they appear in. *Wind*, for example, can be a noun in the context of weather, and can be a verb that refers to coiling something. DeRose (DeRose, 1988) for example reports that in the Brown corpus, which is part of the data set that we use in this study, over 40% of the words are syntactically ambiguous. This example illustrates the fact that ambiguity resolution is the key challenge in POST.

At the Center for Computational Analysis of Social and Organizational Systems (CASOS) at Carnegie Mellon University (CMU) we have developed AutoMap, a tool that facilitates the extraction of relational data from texts (Diesner & Carley, 2004; McConville, Diesner, & Carley, 2008). A variety of NLP and IE routines, such as those listed above, are part of that process. Therefore, a highly accurate POS Tagger is a crucial auxiliary for multiple routines in AutoMap. Furthermore, we envision a high-quality tagger to serve as a helpful stand-alone functionality in AutoMap.

What computational approach to use for building a POS tagger for AutoMap? Taggers can be divided into rule-based, stochastic and transformation-based systems (Manning & Schütze, 1999). For this project we focus on stochastic taggers, which exploit the power of probabilities and machine learning techniques in order to disambiguate and tag sequences of words (Bikel et al., 1999; Stolz, Tannenbaum, & Carstensen, 1965). One widely and successfully applied approach to statistical modeling of natural language data are Hidden Markov Models (HMM) (Baum, 1972). In the domain of speech recognition for instance, HMM has become the favored model (Rabiner, 1989). HMM are also used for POST, where the most accurate systems achieve errors rates of less than four percent (Jurafsky & Martin, 2000). Most of the existent HMM-based POS taggers are trained with labeled data (e.g. (DeRose, 1988; Weischedel, Meter, Schwartz, Ramshaw, & Palmucci, 1993)), while fewer ones use unlabeled data in order to train a model based on expectation maximization (EM) (e.g.(Kupiec, 1992). Given the performance rates that others have achieved with HMM-based stochastic POS taggers we decided to deploy this approach for building a POS tagger for AutoMap.

The next two sections explain some of the most crucial decisions that need to be made when implementing a POS Tagger. By looking under the hood of HMM-based stochastic POST we learnt that some of these decisions significantly impact the resulting accuracy rates. The main contribution of this report is to quantify and explain the change in tagging accuracy due to the design and implementation decisions that human beings needs to make when building a stochastic POS tagger. We envision the knowledge about the sensitivity of the resulting engine and its part to be valuable information for creators and users of who build or apply off-the-shelve or self-made taggers.

2. Method

Markov Models (MM) model the probabilities of non-independent events in a linear sequence (Rabiner, 1989). Applying this idea to natural language empowers us to model language as an interactive system in which words and their underlying features are not discrete events, but do impact each other. Applying MM to POST means aiming to find the most likely sequence of POS in a given sequence, typically a sentence, for all sequence (sentences) in a text or corpus (Baum, 1972; DeRose, 1988; Stolz et al., 1965).

MM are based on a set of assumptions: First, HMM assume a limited horizon into the past. This means that given a present element in a sequence, future elements are conditionally independent of past elements, which implies that present elements depend only on themselves and a few predecessors. The number of predecessors considered equals the order of the HMM. If one decides to look at only the most recent data point (word) from the past, then a first-order HMM is applied. The vast majority of HMM practical applications deploy first-order models. This seems counterintuitive if one follows the idea that higher-order HMM could lead to more accurate predictions than lower order models, because state sequences might depend not only on one (first-order HMM), but multiple predecessors (e.g. in *Department of Labor*). A time horizon of greater than one, however, results in less and sparser training data due to the lack of local histories for the beginning of sequences (Manning & Schütze, 1999). This constraint translates into a serious disadvantage if sentences in the training data are rather short, or if comas are used as delimiters instead of sentence marks. Because a shift from a first-order HMM to a second-order HMM reduces the amount of training data available and the numerical stability of the constructed model we decided to work with a first-order HMM. While the limited horizon assumption enables us to account for the fact that the words in a sentence may depend on each other, especially in the case of meaningful bigrams such as *human rights*, it excludes the consideration of long-range dependencies (Diesner & Carley, 2008). Long-range dependencies are not meaningful N-grams whose elements co-occur next to each other, but elements that interact without being collocated (such as personal pronouns that refer back to a social entity mentioned earlier in the text). This limitation has been shown to be a serious constrain if relevant data points are sparsely scattered across the data. Since in POST training data every word has a tag, this limitation does not apply to POST.

Second, the time invariance assumption means that probabilities are stationery (invariant over time). This assumption can be related to the idea of generalizability of models that are trained on a specific data set and are later on being applied to new and unseen data. The time invariance assumption is a theoretical one only. In reality, language is a dynamic system, in that rules (syntax) and elements (vocabulary) emerge and vanish over time, and across places and people.

Relating the made assumptions to POST enables us to combine and exploit every word's probability and fairly local context as given by a word's predecessor(s) - if available in the training data. HMM, a probabilistic function of MM, brings these two pieces of information together by computing the tag sequence $P(tag_{1\text{-}end})$ that maximizes the likelihood of the product of word probability $P(word_i|tag_j)$ and tag sequence probability $P(tag_j|\ previous\ n\ tags_{j\text{-}N})$.

In practical POST applications, the true sequence of POS that underlies an observed sequence (piece of text data) is unknown, thus forming the hidden states. A POS tagger aims to find the sequence of hidden states that most probably has generated the observed sequence. This task is referred to as decoding, which means that given a set of observations X (words in sentences) and a model μ (the result of supervised learning) we want to reveal the underlying Markov chain of tags that is probabilistically linked to the observed states. Model μ consists of three parameters:

1. Initial state probabilities π. This is a vector that quantifies the probability of the tag of first hidden state in a sentence. Why is that needed? When POST is performed on the sentence level (the classical approach), the first word in the sequence has no predecessor. In order to decode this token, it is typically assumed that the most frequently observed tag for this token across the data set is the most likely tag for this token.
2. State transition probabilities a_{ij}, stored in a transition matrix, quantify the likelihood of observing a certain hidden state given the previous hidden state.
3. State emission probabilities b_{ij}, stored in a confusion or emission matrix, specify the

probabilities of observing a particular state (word) while the HMM is in a certain hidden state.

When training a POS tagger in a supervised fashion, the parameters of model μ are computed from the training data. Therefore, the process of estimating parameters during model training is a visible Markov process, because the surface pattern (word sequence) and underlying MM (POS sequence) can be fully observed. In contrast to that, applying the trained model to tag new and unseen data truly represents a hidden MM, because the tag sequence is hidden underneath the surface pattern and will be revealed using previously gathered empiric evidence (model μ).

Different algorithms for implementing a HMM exist. A widely used one in the NLP domain is the Viterbi algorithm (Viterbi in the following) (Viterbi, 1967). The solution that a POS tagger will suggest is contained in the search space of the applied algorithm or technique. A search space describes and confines the room of possible solutions. For Viterbi, the search space can be represented as a trellis. A trellis is a field composed of a chain of words (the length of the chain depends on the number of tokens per sequence) and a related matrix of all hidden states that were empirically observed during model construction by the probabilistic connections (transitions) between the hidden states. The chain of observed states and the matrix of hidden state transitions are probabilistically connected via the empirically observed emission probabilities for a word by the full set of tags. Viterbi's basic idea and main advantage are the reduction of the complexity of examining every full path through a trellis (all possible combinations of tag transitions and word emissions in a sequence) by recursively finding partial probabilities δ for the most likely path from one state to the next throughout each sequence. Viterbi requires three steps for searching and identifying one complete and the most probable route through the trellis.

Viterbi Algorithm, Goal:

Finding the sequence of hidden states that generates the maximum partial probability $\delta_j(t)$ of possible state combination while moving through the trellis:

$$\delta_j(t) = \max_X P(X, O, X_t = j \mid \mu)$$

where

j… index of potential state

t…index in the sequence of observations

$X = X_1 \ldots X_{t-1} \ldots$ sequence of (hidden) states

$O = O_1 \ldots O_{t-1}$ …sequence of observations

The following steps will be executed in order to achieve the goal:

1. Initialization $\delta_j(1) = \pi_j, 1 \le j \le N$
2. Induction $\delta_j(t+1) = \max_{1\le i\le N} \delta_j(t)\, a_{ij}\, b_{ij o_t}, 1 \le j \le N$

   Store backtrace $\psi_j(t+1) = \arg\max_{1\le i\le N} \delta_j\, a_{ij}\, b_{ij o_t}, 1 \le j \le N$

   where $\psi_j(t)$ = storage of node of incoming arc to most probable path
3. Termination and path (most likely tag sequence) readout (by backtracking)

$$\hat{X}_{T+1} = \arg\max_{1\le i\le N} \delta_i(T+1)$$

$$\hat{X}_t = \psi_{\hat{x}_{T+1}}(t+1)$$

$$P(\hat{X}) = \max_{1\le i\le N} \delta_i(T+1)$$

In summary, the supervised, sequential, stochastic machine learning technique described herein (machine learners are systems that improve their performance (here, POS tagging accuracy) with experience) constructs a model μ that for each sequence of (*x*,*y*), where *x* are the words in a sentence and *y* the corresponding POS tags, predicts an entity sequence *y* = *μ(x)* for any sequence of x, including new and unseen text data. Since HMM estimate a joint probability (the one of words and tags) they are a member of the family of generative models (Dietterich, 2002). The tag sequence that results from applying model μ to new data may not necessarily be the correct one, but it will be the most likely one given the model and the new data. It is for this reason that careful and informed design and implementation of a tagger are key to success.

3. Data

The data set used for training and validation in this project is the tagged version of the Penn Treebank 3 (PTB) corpus (M. P. Mitchell, Marcinkiewicz, & Santorini, 1993; P. M. Mitchell, Santorini, Marcinkiewicz, & Taylor, 1999)**Error! Reference source not found.**. The PTB ollection contains 2,499 texts from differenet sources such as over three years of news coverage from the Wall Street Journal (1989-1992) and a tagged version of the Brown corpus (1961). Every word in the corpus is annotated with at least one out of 36 possible tags (see the Appendix for a list of tags and their meaning). The PTB figu is stored in 500 data files, which are organized in 15 folders.

In cases where the human coders who annotated the PTB texts with POS were uncertain about the best POS for a word, e.g. when a word was syntactically ambiguous, multiple tags were assigned in a non-standardized order (Klein & Manning, 2002). For example, *England-born/NNP/VBN* means that England-born might be a singular proper noun as well as past-participle verb. There is a total of 121 such cases of tag indeterminacy in PTB. We performed several qualitative checks (human reasoning about the best out of the offered tags) on randomly drawn instances of this issue from PTB, which convinced us of the random order of multiple tags per word.

4. Experiment

We conducted series of experiments was in order to identify the impact of several independent variables, which we explain in detaiin this section, on the dependent variable of interest: the accuracy of tagging new data by using the constructed model. What can the outcome of this exercise be useful for? First, we need such detailed knowledge in order to construct the best POST model for AutoMap (for machine learning problems, the best model is typically the most concise one that generalizes with highest accuracy to new data). Second, we envision creators and users of HMM implementations to use this kind of knowledge in order to build or reasonably apply such systems.

4.1 Dissembling Viterbi

In section 2 we described the different computational steps that are involved in the Viterbi algorithm. How much accuracy gain can be attributed to each of these steps? In order to answer this question we isolated each step and ran experiments in order to quantify the partial accuracy gain that the application of the following computational steps accounts for:

1. Probabilities of words in isolation
2. Emission and Transition Probabilities
3. Partial probabilities δ and back pointers ψ
4. Backtracing

Step 1 enables us to isolate and measure the accuracy achieved by using emission probabilities only. This procedure disregards the impact the POS of the proceeding word on the subsequent word's POS, thus not making use of a words' historical context (a "zero-order HMM"). As a result, the tag that has been observed most frequently for the word under consideration during training will be selected and returned. This step resembles the initialization stage as well as the computation of the initial state probabilities as described in section 2. In HMM and Viterbi, probabilities of words in isolation are used for tagging the first word in every sentence as well as for one-word sentences. We further on refer to this approach as the Unigram Model (UM in the following). We use the UM as our baseline performance measure.

Step 2 represents a regular HMM (HMM in the following). That is the product of emission (of a word by a tag) probabilities and transition (from POS to POS) probabilities as computed during the induction stage of Viterbi. The difference in accuracy rates between step 1 and 2 allows us to isolate and quantify the impact of transition probabilities on tagging accuracy. A HMM performs a local search. This means that the model decides upon the most likely tag for each token (by choosing the POS that maximized the product of the possible transition and emission probabilities between the current and preceding words and their POS) prior to moving on to the next word.

Step 3 is the heart of Viterbi. It combines partial probabilities as computed in step 2 with a forward search for the best (a complete and the most probable) path through the trellis. At each step while moving through the sequence for which the hidden states need to be determined and for each possible hidden state the algorithm computes partial probabilities. These partial probabilities are the product of the emission probability of the potential state, the highest transition probability from the previous possible states and the partial probability of the previous state that generated the highest transition probability. Hence this algorithm considers the emissions, transitions and the globally optimal sequences of hidden states that are determined while moving through each step in the trellis. In the following we refer to this step as VitF (Viterbi Forward). The difference in accuracy between steps 2 and 3 represents the difference between global forward search and local search.

Step 4 not only computes all possible forward paths through a trellis (as done in step 3) from start to end, but after completing the forward search it determines the final partial probability of the last state, which represents the globally optimal solution and backtraces the most probable path through the trellis from the last to the first token. In the following we refer to this step as VitB (Viterbi with backtracing).

In summary, the difference between points 1 and 2 versus points 3 and 4 represents the difference between a globally versus locally maximized solution. An actual implementation of the Viterbi algorithm requires all four points. Each of these points and in the order as they are outlined here includes the previous point(s) (if applicable), thus furthermore adding to Viterbi's time and space complexity. This increase in computational complexity is because each step, in the presented order, increases the amount of information or empiric evidence that is comprised in the process of making a decision about the best tag sequence. For this reason we hypothesize that the POS accuracy increases from each step to the next.

4.2 Handling Noise

Typically, text data includes various types of noise in varying quantity. What precisely qualifies as noise and how much of it will be normalized or eliminated depends on the goal, resources, and researcher. For this project, tagged tokens are not considered as noise if and only if they are composed of or tagged as an arbitrarily long sequence of any of the following:

- Characters from a or A to z or Z (regular words)
- Numbers from 0 to 9 (numbers)
- Sentence markers (digits and end of sentences)
- Ampersands (used e.g. in corporation names such as *John Wiley & Sons*)
- Dollar symbols (mainly used to denote monetary values)
- Hyphens (often used to denote genitive markers)

- Dashes (often used in compound words such as *long-term*)

All tagged tokens that are or comprise any symbol not listed above are considered as noise herein. The set of noise terms for this project contains for example tokens whose tag resemble the token (e.g. *:/:*) or most (99.84 percent) tokens that are tagged as symbol (SYM). Commas are part of the SYM set. Only 0.01 percent of the tokens tagged as list markers (LS) qualified as noise, while most list markers are actual words or numbers.

For other projects, the tokens and tags that we consider as noise terms might be valuable signals. For data that is stored as coma separated values, for instance, commas would serve as the sequence delimiter. The following example shows an excerpt from a POS-tagged PTB data file in that we printed the tagged elements that we consider as noise in red and bolt font. Any word-tag tuple in which one or both elements qualify as noise can be removed prior to learning and model evaluation or not. Table 1 shows the transition probabilities for the example given in Figure 1 with and without symbol performing removal. The transitions that both versions differ in are printed in bold and red font. This example shows that when noise is not removed, more and a higher variety of transitions will be learnt.

Why do we think that determining the impact of removing noise prior to learning on POS results could matter? For practical POST applications, people are typically not interested in predicting tags for symbols, but only for what is typically considered as content. From a computational as well as practical standpoint, decoding noise requires resources (space and time), which one might not want to spend. We hypothesize that POST results will be more accurate when learning and evaluating are based on clean instead of noisy data.

Figure 1: Excerpt from PTB Data

Publication/NN
and/CC
distribution/NN
:/:
Volume/NN 1/CD
(/((/(
A[fj]/SYM
)/) of/IN
the/DT seventh/JJ edition/NN

Table 1: Impact of Noise Definition on Transitions

Transitions before symbol removal	*Transitions after symbol removal*
NN - CC	NN - CC
CC - NN	CC – NN
NN - :	**NN – NN**
: - NN	NN – CD
NN – CD	**CD - IN**
CD – (	IN – DT
(- (	DT – JJ
(- SYM	JJ - NN
SYM – IN	
IN – DT	
DT – JJ	
JJ - NN	

4.3 Smoothing and Handling of Unknown Data Points

Any HMM implementation requires cautious handling of small numbers and zero probabilities at various points: first, multiplying and propagating partial probabilities in the induction stage can lead to number underflows. Since UM and HMM disregard partial probabilities, this issue only applies to Viterbi. This problem can be avoided by using the natural logs of the transition and emission probabilities, and translating the respective multiplications into summations.

Second, words and state sequences that have not been observed in the training data, but do occur in the evaluation data, will cause:

- Zero probability in the induction step of Viterbi. As a result, an entire vertical column in the trellis (all δ for step i) would have zero probabilities, so that the propagation of any path would break.
- Accuracy loss for UM, HMM, VitF, and VitB during model evaluation. This is because tokens that did not occur in the training data but are observed in the evaluation or any other new data will have a zero probability of being emitted by any tag as well as a zero probability of being involved in any tag transition. In such cases, the *unknown* tag is typically initially assigned to these words. Practically, *unknown* never matches the best tag for a word, and therefore contributes to an increase in the tagging error rate. Depending on the algorithm used, *unknowns* on average account for up to 28 percent of all tokens when a model trained on one portion of PTB data and is applied to another portion of PTB (detail on that in section 4.4). Accuracy loss due to not handling unknowns cannot be solved by increasing the amount of training data used: even models trained on humongous training sets are likely to encounter new words when being applied to unseen data. The issue represents the downside of the time-invariance assumption made for MM: language is a continuously changing system with words emerging and vanishing across time and places, e.g. in the cases of new names of people, places, or products.

We empirically test the impact of handling unknowns on POST accuracy. The following unknown handling strategy is used: Zero probabilities for emissions are prevented by adding tokens newly encountered during evaluation to the emission matrix, tagging them as ”UNKNOWN”, and assigned a minimum probability to them. This intervention prevents the multiplication by zero in the development of the trellis. Zero probabilities for transitions that involve the UNKNOWN tag (*P(t|t=UNKNOWN, P(t=UNKNOWN| t)))* and that have not been observed a priori are caught by assigning a minimum probability to them as well. Initially, we chose a minimum probability that equaled the smallest empirically observed probability in the learning data set. This solution resembles the Adding One strategy (Church, 1988), which in addition to linear interpolation is a frequently applied smoothing technique in tagging (Kupiec, 1992). Later on we realized that in some cases our minimum probability equaled empirically observed probabilities. In cases of ties between any tag and the unknown tag, our engine makes a random choice, which can give an empirically observed small probability (EP) the same weight as the artificially assigned minimum probability (AP). In order to weight EPa higher than APs we decided to first find the smallest EP in the data, dividing it by 100 (we ran multiple tests in order to find an appropriate value), and using this value as the AP. We found that for handling emission probabilities involving unknowns this strategy leads to major, positive changes in accuracy rates, especially for VitF and VitB. For taking care of transitions that comprise the unknown tag, this strategy does not lead to significant changes in POST accurate rates, but it does suppress the detection of unknowns to a degree where they become unlikely to ever be selected. However, in some cases we want to maintain the unknown tag in order to be able to send it to a post-processor, as explained in the text paragraph. It is for this reason that we choose the weight EP deterministically higher than APs only for emissions, but not for transitions.

After zero probabilities for emission and transition have been converted to minimum probabilities lower than empirical probabilities words tagged as *UNKNOWN* are passed to a post-processor that applies a set of rules in order to re-label unknown words with an actual POS. The best-performing unknown-word resolution techniques in tagging use information about the word's spelling (DeRose, 1988; Viterbi, 1967). We built upon this idea. The construction of the post-processor is described in section 5.3.. We hypothesize that post-processing of unknown words will further increase the POST accuracy results, because an actual tag is more likely to match the best tag for a word than the *unknown* tag.

4.4 Aggregating Hidden States

PTB uses a set of 48 unique tags. 36 of them are regular POS, and the other 12 are symbols (#,$,.,,,:,(,),",',",',"). The Appendix lists the regular POS them along with the total frequency of their occurrence in PTB. Section 4.2. explained how we handle the symbols. For many real-world applications, this categorization is too fine grained. When analyzing newspaper articles for instance, people might be interested in using POST to support the identification of terms that refer to the who, what, where, when, why and how of what is reported in text data. For identifying the *who* (one or multiple people) for instance, a category named *agents* might be more appropriate than classifying singular proper nouns and plural proper nouns separately. As a second example, for finding all words that indicate an action and therefore can be thought of representing the *what* category, one might want to collect all verbs in one category, regardless of whether they are a base form verb, a present participle or gerund verb, a present tense not 3rd person singular verb, a present tense 3rd person singular verb, a past participle verb, or a past tense verb (six categories). We aggregated the regular POS from the PTB tag set into twelve categories as shown in Table 2.

Table 2: Aggregation of PTB Categories

Aggregated Tag	Meaning	Number of Categories in PTB	Instances in PTB
IRR	Irrelevant term	16	409,103
NOUN	Noun	2	217,309
VERB	Verb	6	166,259
ADJ	Adjective	3	81,243
AGENT	Agent	1	62,020
ANA	Anaphora	1	47,303
SYM	Noise	8	36,232
NUM	Number	1	15,178
MODAL	Modal verb	1	14,115
POS	Genetive marker	1	5,247
ORG	Organization	1	1,958
FW	Foreign Word	1	803

The consolidated set consists of personal singular noun (AGENT), personal plural noun (ORG), verbs (VERB), modal verbs (MODAL), nouns (NOUN), adjectives (ADJ), personal pronouns (ANA), genitive markers (POS), non-content beating words (IRR), symbols (NOISE), numbers (NUM), and foreign words (FW). Seven of the aggregated categories map to only one PTB category, while the other categories are represented by up to 16 different categories in PTB. The rows in Table 2 are sorted by decreasing frequency of the cumulative occurrence of each category in PTB (last column in Table 2) in order to illustrate the fact that the number of words per tag category varies widely (for details see the Appendix).

Our aggregation is one possible solution. For other text sets, domains, or projects, other consolidations might be more appropriate. We hypothesize that aggregation will lead to more accurate POST results, because the classifier needs to pick one best POS out of a smaller pool of choices.

5. Results

The impact of each independent variable or routine described in the previous section on POST accuracy was empirically tested. For these tests, we performed ten ten-fold cross validations per variable. In order to perform ten-fold cross validations, the corpus was randomly split (500 files, about one million words) into ten partitions for every single run. Nine folds (450 files) were used for training and generating model μ. From the remaining tenth fold (50 files), all tags were removed, μ was applied to this fold in order to tag the data, and the assigned tags were compared to the original labeling of the tenth fold. Every deviation was recorded as an error. This procedure was repeated nine more times. The reported error rates result from averaging the errors rates of ten consecutive runs.

Typically, taggers are evaluated by running the Gold Standard test and/ or by comparing the results to a Unigram Baseline test (Jurafsky & Martin, 2000)**Error! Reference source not ound.**. In this paper, we use both tests: The Gold Standard measures performance by identifying the portion of tags that the tagger and a human-labeled validation set agree upon. We apply this test during model evaluation. The Unigram Baseline test is the same as the UM model already described in this paper.

5.1 Dissembling Viterbi

How much partial accuracy can be attributed to the different steps involved in Viterbi? Our findings as shown in Table 3 and 4 indicate that on average the baseline model (UM) tags 86.82 of the words in the evaluation set correctly. Upgrading from UM to HMM leads to a significant accuracy increase of 5.16 percent (significance statement based on a paired, two–sided t-test with a 95% confidence interval). This insight suggests that the consideration of transitions only among hidden, not among observed states in a linear chain improves predictive power substantially. Further enhancing the implementation to VitF results in a 0.3 percent accuracy gain. VitB, which out of the algorithms tested exploits the most empiric evidence, achieves another significant 1.06 increase in accuracy; confirming that backtracing does improve Viterbi. The standard deviations (0.49 percent at the most), which decrease by algorithm, suggest that the results are fairly robust across different portions of the data set.

Table 3: Impact of Algorithm on Accuracy

	UM	HMM	VitF	VitB
Average	**86.82%**	**91.99%**	**92.29%**	**93.35%**
Min	85.99%	91.49%	91.81%	92.89%
Max	87.36%	92.40%	92.65%	93.67%
Std Dev	0.49%	0.36%	0.33%	0.30%

Table 4: Difference between Algorithms

From	To	Difference	Significance
UM	HMM	5.16%	0.000**
HMM	VitF	0.30%	0.000**
VitF	VitB	1.06%	0.000**

Overall, our findings suggest that the baseline algorithm, which only considers emission probabilities, is a powerful prediction method. The efficiency of this simple approach has already been recognized by (Atwell, 1987). Moreover we can confirm our hypothesis that global search outperform local search. However, we showed that the difference between HMM and VitF is fairly small (0.3 percent) and smaller than all other differences between algorithms. One possible explanation for this observation is the following chain of thoughts: HMM weight transition and emission probabilities about equally strong, while Viterbi (both versions of it) weights transitions higher than emissions. VitF enables very small and occasionally meaningless transition probabilities – an effect that VitB partially corrects for, with the tradeoff being that VitB weights transitions even stronger than emissions. VitF outperforming HMM only slightly suggests that once transitions between hidden states that are probabilistically linked to a surface pattern are considered, one needs to go the extra mile of searching through a web of probabilistic connections among underlying states back *and* forth in order to achieve a substantial gain from global search over local search. Searching through the space of possible solutions not only into one direction, but into two, has a greater (in our case about more than three times greater) impact than considering connections amongst underlying patterns at all.

5.2 Handling Noise

Is it worthwhile cleaning the data from symbols that do not need to be predicted for practical POS applications? For generating the results shown on the previous page we did not remove any symbols. These numbers will now serve as our control case. Not considering any token-tag tuple that contains any element which is not a letter, number, ampersand, dollar symbol, hyphen, or dash for neither learning nor evaluating leads to the accuracy rates shown in Table 5.

Table 5: Impact of Handling Noise on Accuracy

Dataset	Accuracy Measure	UM	HMM	VitF	VitB
Clean	**Average**	**85.66%**	**91.35%**	**91.55%**	**92.59%**
	Min	85.15%	91.01%	91.17%	92.38%
	Max	86.29%	91.79%	91.99%	92.85%
	Std Dev	0.39%	0.24%	0.25%	0.16%
Noisy	**Average**	**86.82%**	**91.99%**	**92.29%**	**93.35%**
Noisy to Clean	**Difference in Average**	**-1.16%**	**-0.63%**	**-0.74%**	**-0.76%**
	Significance of Difference	0.000**	0.000**	0.000**	0.000**

The results show that keeping noise in the data consistently and significantly improves accuracy rates. This observation contradicts with our hypothesis that learning and evaluating with clean data would lead to more accurate results than working with noisy data. Why did we observe the opposite? Looking further into the data revealed that most of the noise signals are not ambiguous. A comma, for instance, is hardly ever assigned to anything other than comma. Due to the resulting strong and unambiguous emission probability for noise symbols, we predict noise with very high accuracy.

What does that imply for modeling? Accuracy rates significantly benefit from data that contains certain entities classes that occur frequently and that are easier to predict than other categories because they are less ambiguous (not much to anyone’s surprise). For boosting tagging accuracy, keeping noise in the data is beneficial. However, as for any other machine learning application as well, special attention needs to be paid to cross-validating a model on new data prior to making generalizations. In order to build POST models that do not overfit to the prediction of noise by overly adjusting themselves to this idiosyncrasy, noise needs to be removed from the data. One might argue that real data are likely to contain the sort of noise that was eliminated for this project. That is true. However, not removing noise prior to learning reduced the empiric evidence that can be gathered on transitions of tags other than noise while increasing the information learnt about transitions between noise and tags of interest, thus limiting the predictive capabilities for practical applications where correct tagging of content is favored over tagging noise.

5.3 Handling Unknowns

Applying a POS tagger to new data can result in two types of errors: misclassification of words that the model has prior knowledge about (algorithmic failure), and failure to find the right class label for a word that has not been observed by the model during training (failure in handling unknowns). Some of the newly encountered words will be correctly resolved by the algorithm by exploiting transition probabilities, while others will still be misclassified. In order to figure out if it is worthwhile to resolve unknowns after evaluating the model and prior to outputting the results we first need to understand the distribution of the two error types introduced in this section across the algorithms that we test herein.

Table 6: Error Types per Algorithm (clean data)

Error Type	UM	HMM	VitF	VitB
Unknowns	28.2%	6.0%	9.8%	1.8%
Algorithmic	71.8%	94.0%	90.2%	98.2%

Table 6 shows that for all algorithms, the vast majority of errors is due to algorithmic failures. The baseline model by far has the greatest potential for benefitting from unknown resolution. For HMM and VitF, an accuracy increase of up to 6 and 9.8 percent, respectively, due to handling unknowns correctly is theoretically possible. For VitB we cannot expect a major accuracy improvement from unknown handling – the algorithm accomplishes most of the unknown handling by itself; the remaining errors due to unknowns might be data-artifacts. This insight suggests that the more an automated solution exploits empiric evidence the less it can be further improved by man-made post-processing strategies. For machinery that makes decisions on its own by strongly relying on its computational power and by trying to resolve uncertainties rather than admitting them careful and well-informed engine construction is crucial since posteriori interventions cannot improve functionality. For UM, HMM and VitB, all of which exploit less empiric evidence than VitB does, a combination of an initial automated solution with hand-crafted heuristics has a potential for outperforming fully-automated approaches. Such algorithms that are less decisive and declare more uncertainties allow for posterior intervention. It is the engineer in the first place who determines how much uncertainly shall be disclosed by the engine (as for instance described in section 2 where we reason about the minimum probability for transitions and missions for the case of unknowns).

In order to develop post-processing rules for unknown handling in a data driven fashion we started by collecting the outputs from the ten cross-fold validations that we ran on clean data. From these data we parsed out all cases where any of the four algorithms assigned “unknown” to a word after trying to solve unknowns algorithmically and prior to making a decision for the best tag for this word. This resulted in 19,202 unique words that were mistagged as unknown. We found that no algorithm made any mistake on unknowns that the UM would not make. Therefore we further worked with the set of unknowns that the UM detected. Next we systematically examined the unknown words for regularities in their association with certain POS. Table 7 provides details on this process. The insights gained from this analyzed were formalized and implemented as the following orthographic rules for tagging unknown words:

1. Words containing a digit are tagged as numbers (CD).
2. Capitalized words ending with –s are tagged as proper plural nouns (NNPS).
3. Capitalized words are tagged as proper singular nouns (NNP).
4. Words ending with -ant, -able, -al, -ory, -ent, -ful, -ian, -ible, -ic, -ish, -less, -oid, -ory, or -ous are tagged as adjectives (JJ).
5. Words ending with –s are tagged as common plural nouns (NNS).
6. Words ending with -ing are tagged as present participle or gerund verbs (VBG).
7. Words ending with –ed are tagged as past participle verbs (VBN).
8. Words ending with -ly are tagged as adverbs (RB).
9. Words ending with –ize are tagged as verbs (VB).
10. Words ending with -est are tagged as adjective, superlative (JJS).
11. Words ending with -er are tagged as adjective, comparative (JJR).
12. All remaining unknowns are labeled as singular or mass noun (NN).

Next we tested these rules in the order presented above. After evaluating a rule (let’s call this rule A) we kept rule A applied for evaluating the subsequent rule (rule B for now) if and only if A caused more harm than damage. Testing the impact of unknown handling on clean data led to the results shown in Table 7 and 8.

Table 7: Rule Evaluation (on clean data)

Rules				Types of errors in detecting tag			Impact of applying rule(s)			
ID	If token is unknown	Then	Other rules applied	Total	Algorith-mic	Unknown	Tokens impacted by rule	Success	Failure False Positives	Failure False Negative
1	contains digit	CD		369	62.6%	37.4%	141	126	15	12
2	capitalized and ends with -s	NNPS	1	107	43.9%	56.1%	264	54	210	6
3	capitalized	NNP	1	2217	29.1%	70.9%	2016	1561	455	10
4	ends with any of *	JJ	1,3	1550	67.9%	32.1%	546	379	167	119
5	ends with -s	NNS	1,3,4	678	48.4%	51.6%	431	334	97	16
6	ends with -ing	VBG	1,3-5	280	67.1%	32.9%	119	92	27	0
7	ends with -ed	VBN	1,3-6	789	89.4%	10.6%	171	83	88	1
8	ends with -ly	RB	1,3-7	867	93.0%	7.0%	63	60	3	1
9	ends with -ize	VB	1,3-8	1247	96.5%	3.5%	5	5	0	39
10	ends with -est	JJS	1,3-9	51	84.3%	15.7%	9	8	1	0
11	ends with -er	JJR	1,3-10	133	94.7%	5.3%	53	7	46	0
12	remainder	NN	1,3-10	3207	86.8%	13.2%	599	422	177	0

* -ant, -able, -al, -ory, -ent, -ful, -ian, -ible, -ic, -ish, -less, -oid, -ory, -ous

** cases in which failure exceeds success are marked with gray background

Table 8: Rule Evaluation (on clean data)

Rules				Number of unknowns				Change in accuracy from previous rule(s)*			
ID	If token is unknown	Then	Other rules applied	UM	HMM	VitF	VitB	UM	HMM	VitF	VitB
0	and nothing else happens	error		4100	511	892	140	NA	NA	NA	NA
1	contains digit	CD		3959	508	889	140	0.117%	0.003%	0.003%	0.000%
2	capitalized and ends with -s	NNPS	1	3695	445	797	130	0.050%	0.001%	0.001%	0.000%
3	capitalized	NNP	1	1943	126	193	3	1.397%	0.154%	0.401%	0.071%
4	ends with any of *	JJ	1,3	1397	123	149	3	0.351%	0.003%	0.016%	0.000%
5	ends with -s	NNS	1,3,4	966	93	101	3	0.310%	0.003%	0.013%	0.000%
6	ends with -ing	VBG	1,3-5	847	76	84	3	0.085%	0.013%	0.013%	0.000%
7	ends with -ed	VBN	1,3-6	676	25	29	1	0.077%	0.009%	0.014%	0.000%
8	ends with -ly	RB	1,3-7	613	12	16	0	0.056%	0.011%	0.011%	0.001%
9	ends with -ize	VB	1,3-8	608	12	16	0	0.005%	0.000%	0.000%	0.000%
10	ends with -est	JJS	1,3-9	599	12	16	0	0.007%	0.000%	0.000%	0.000%
11	ends with -er	JJR	1,3-10	546	9	11	0	0.006%	0.000%	0.000%	0.000%
12	remainder	NN	1,3-10	0	0	0	0	0.385%	0.000%	0.003%	0.000%

* cases which resulted in no accuracy gain are marked with dark gray background, cases which resulted in accuracy gains greater than zero and smaller than 0.05% are marked with light gray background

We decided to keep only those rules that caused more success (correct resolution due to application of rule(s) as shown in 3rd column from the left in Table 7) than harm (false positives and negatives as shown in last two columns in Table 7) in our test scenario. This policy led to the following modification of our rule set:

- Dropping the rule that capitalized words ending with –s get tagged as proper plural nouns. 60 percent of the false negatives that resulted from applying this rule turned out to be proper singular nouns, which have a high probability for getting solved with the subsequent rule, and 30 percent were common plural nouns.

- Dropping the rule that words ending with -er are tagged as comparative adjectives. This rule correctly resolved all of the remaining seven comparative adjectives, but also converted 42 tags that belonged into other tag classes into comparative adjectives.
- Converting words ending with –ed caused slightly more misclassifications than correct resolutions. However, this rule tabbed into the past tense verb, and since we plan on aggregating all different verb classes into one general verb class later on we decided to keep this rule.

We assume the final set of rules to be not just corpus-specific, but of general applicability for POST. Our results (Table 9) show that the rules which we designed and implemented are capable of resolving between 5.6 and 10.6 percent of the unknowns. But only for algorithms that effectively allow for hybrid strategies (UM, HMM, VitF), unknown handling via post-processing rules leads to significant accuracy increases (0.3 to 3 percent), while for algorithms that maxes out on unknown handling algorithmically (VitB), only a small and insignificant increase was achieved. Even after UM, HMM and VitB significantly increased in accuracy, VitB still outperforms every single one of them.

Table 9: Impact of Unknown Handling on Tagging Accuracy

Data	Measure	UM	HMM	VitF	VitB
Handle Unknowns	**Average**	**88.65%**	**91.69%**	**92.14%**	**92.75%**
	Min	88.26%	91.21%	91.73%	92.37%
	Max	88.86%	92.24%	92.56%	93.18%
	Std Dev	0.18%	0.32%	0.28%	0.26%
	Percent of Unknowns correctly resolved	**10.61%**	**5.59%**	**6.03%**	**9.32%**
Clean to Handle Unknowns	**Difference in Average**	**2.99%**	**0.33%**	**0.59%**	**0.16%**
	Significance of Difference	0.000**	0.018**	0.000**	0.113

In summary, data-driven derivation of post-processing rules as well as rule testing is a time-consuming process that requires the allocation of trained human resources. In summary, our findings suggest that if one does not make such an investment, spending resources instead on building algorithms that handle uncertainties on their own without confessing to the end-user that they behave in this way will lead to the even better performance.

5.4 Aggregation of Hidden States

The tests on tag aggregation were run on clean data in which unknowns are handled as described in the previous section. Consolidating the tag classes as defined in PTB (total of 36) into fewer (12), user-defined classes that are tailored to the end-user's analytical needs led across all algorithms tested to the highest accuracy rates and the highest gain in accuracy across the independent variables tested herein (Table 10, Figures 3 and 4).

Table 10: Accuracy Rates per Algorithm and Independent Variable

Independent Variables applied	UM	HMM	VitF	VitB
Baseline	86.82%	91.99%	92.29%	93.35%
Clean Data (CD)	85.66%	91.35%	91.55%	92.59%
CD + Handle Unknowns (HU)	88.65%	91.69%	92.14%	92.75%
CD + HU + Aggregate Tag Classes	94.30%	93.07%	94.07%	94.25%

Table 11: Difference between Algorithms (clean data, unknowns handled, aggregation applied)

From	To	Difference	Significance
UM	HMM	1.06%	0.000**
UM	VitB	1.04%	0.330
HMM	VitF	0.61%	0.000**
VitF	VitB	0.18%	0.004**

When the POST model is learnt and evaluated on clean data, only aggregated tag classes are used, and unknown handling is applied during evaluation, the simple UM model outperforms all other algorithms. The difference between UM and VitB, however, is not statistically significant. Therefore, we decided to use the latter for AutoMap.

Figure 2: Impact of Independent Variable on POST Accuracy

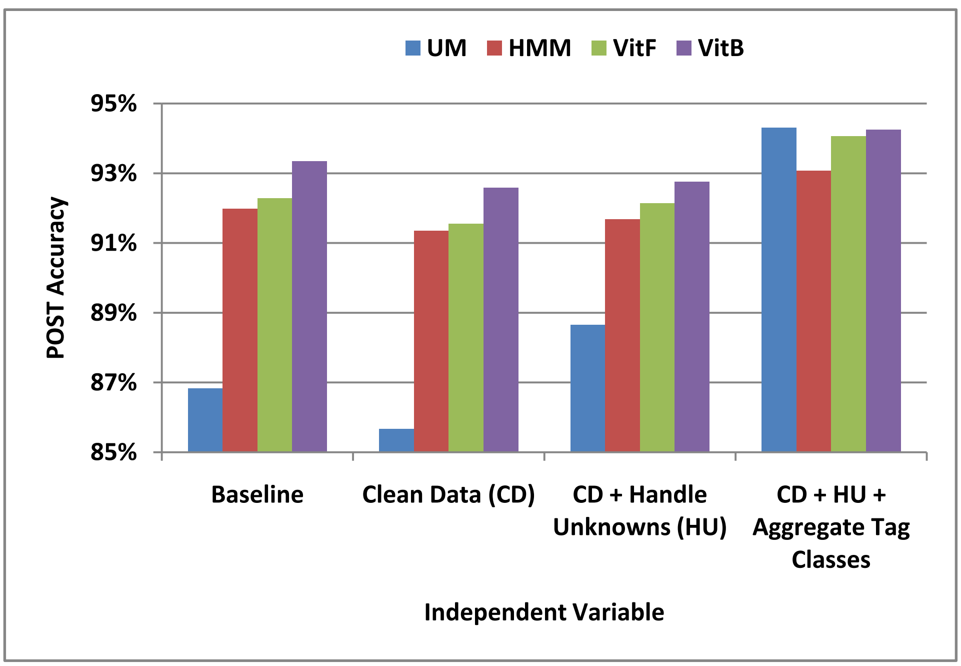
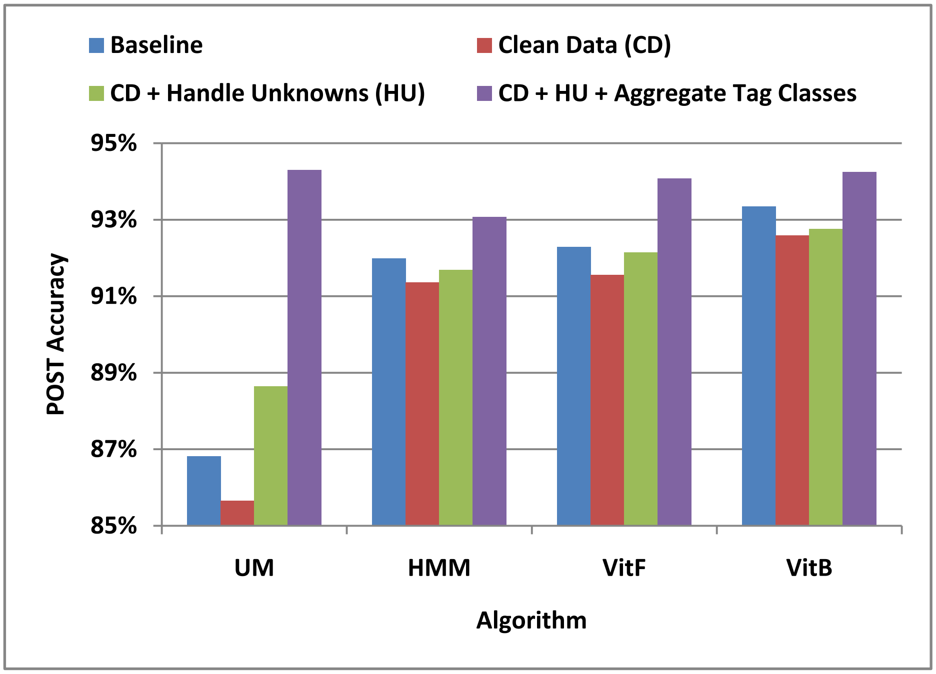


Figure 3: Impact Algorithm on POST Accuracy



In summary, our results on aggregation suggest that an informed, needs-driven and user-defined consolidation of available choices can lead to performance improvements that consistently across various algorithm of different complexity can have a greater positive impact than eliminating prominent error sources such as noise and unknown data. We therefore emphasize the design of analytical solutions that enable end-users to interact with tools or human beings on the developmental side of solutions in such a way that customer needs can be elicited and considered for the sake of performance improvements.

6. Integration of Parts of Speech Tagging into AutoMap

Based on the insights gained by testing the impact of various independent variables on the accuracy of four different POST algorithms (the difference between the algorithms themselves being one of the variables) we decided to build a POS tagger that:

Is based on th Viterbi algorithm with backtracing implementation.

Is enhanced it with unknown-handling heuristics.

Uses the aggregated tag set shown in Table 2.

We implemented and integrated this tagger into AutoMap as follows: First, we trained the POS model on the full PTB set (not only 90 percent of it), output the emission and transition matrices as data files, and added these data to AutoMap. When a user clicks the "Tag texts" button in the AutoMap GUI or activates the "Tag Texts" command in the batch mode version, first the untagged texts will be split into sentences by using a sentence splitter (Piao, n.d.). Given the data per sentence, the initialization vector will be constructed. Using the initialization vector as well as the states as represented in the emission and transition matrices a trellis will be built for every sentence in the data. These trellises are used to find a complete and the most likely sequence of POS per words per sentence. Users can use the POS tagger in the GUI or batch mode version of AutoMap in three ways:

Stand-alone feature: when the “Tag Texts” option is selected, AutoMap performs POST and displays each word along with the POS that the tagger predicted for it. The user can store the POS annotated corpus. For the sample text shown in Figure 4, AutoMap generated the POS annotated text shown in Figure 5.

Output a table (coma separated values format) that lists all words in a corpus in the first column and the respective POS that the model has identified for that word in the following column. If more than one POS was predicted for a word, the word-tag tuples will be placed in multiple rows. Tables 12 and 13 show that list for the sample text given in Figure 4.

Figure 4: Raw text loaded into AutoMap

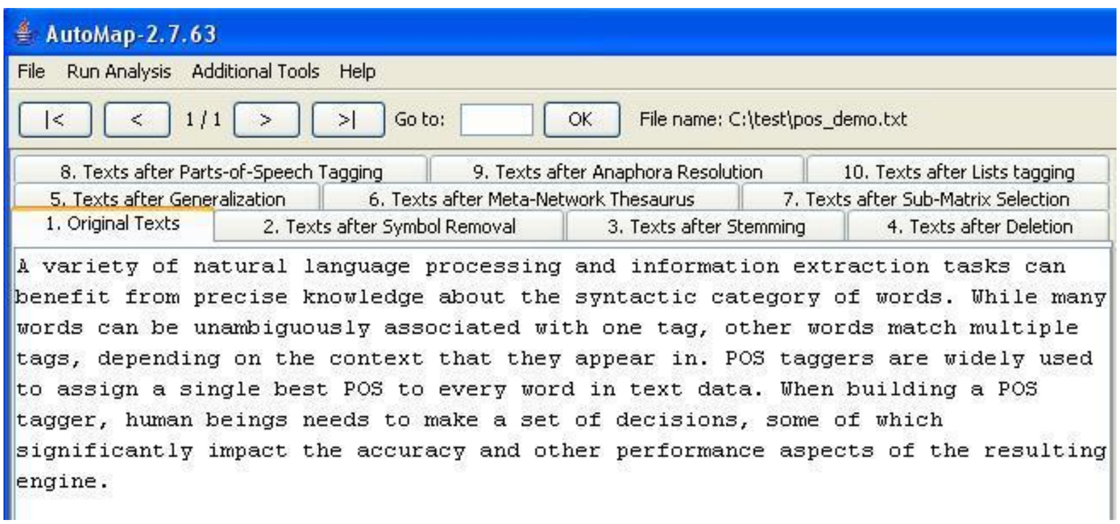


Figure 5: Integration of POS Tagger as stand-alone routine in AutoMap

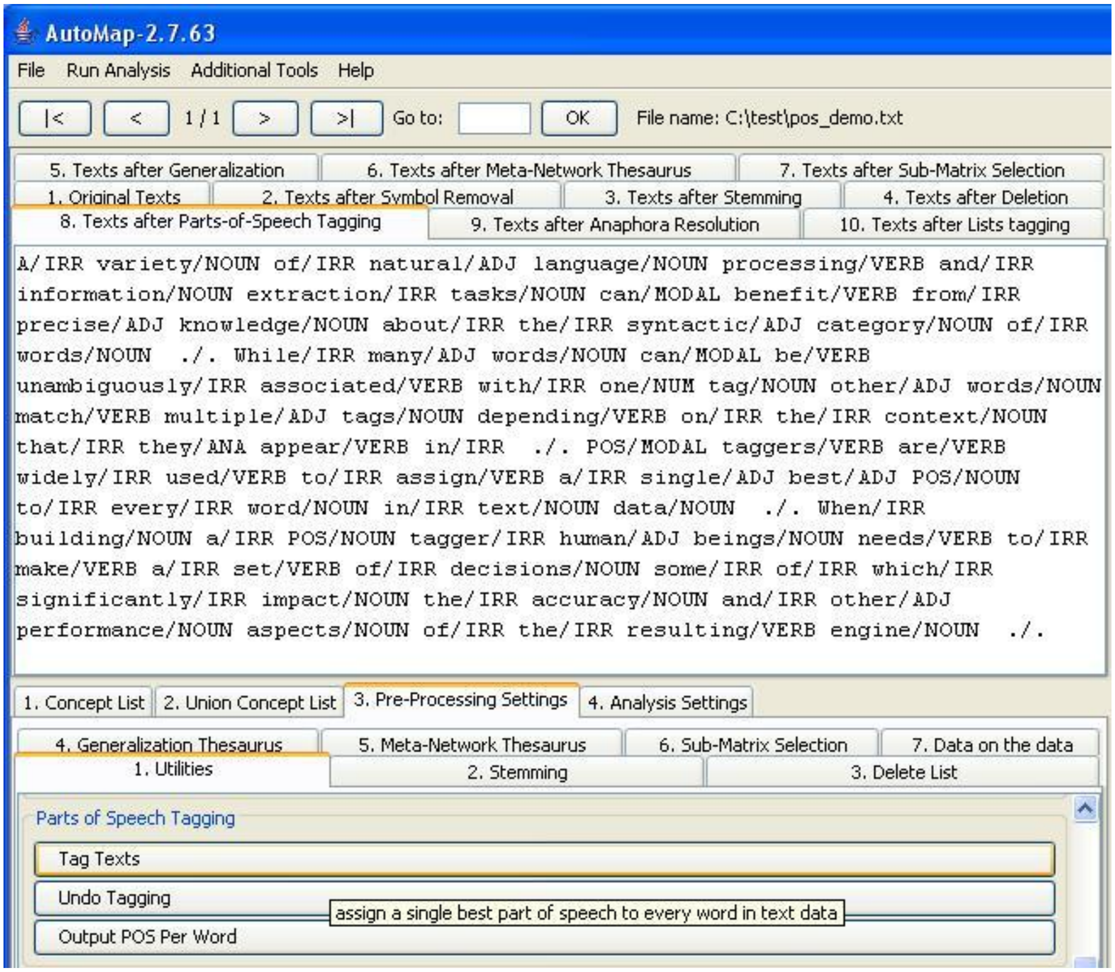

AutoMap-2.7.63
File
Run Analysis
Additional Tools
Help
|<
<
1 / 1
>
>|
Go to:
OK
File name: C:\test\pos_demo.txt
5. Texts after Generalization
6. Texts after Meta-Network Thesaurus
7. Texts after Sub-Matrix Selection
1. Original Texts
2. Texts after Symbol Removal
3. Texts after Stemming
4. Texts after Deletion
8. Texts after Parts-of-Speech Tagging
9. Texts after Anaphora Resolution
10. Texts after Lists tagging
A/IRR variety/NOUN of/IRR natural/ADJ language/NOUN processing/VERB and/IRR information/NOUN extraction/IRR tasks/NOUN can/MODAL benefit/VERB from/IRR precise/ADJ knowledge/NOUN about/IRR the/IRR syntactic/ADJ category/NOUN of/IRR words/NOUN ./. While/IRR many/ADJ words/NOUN can/MODAL be/VERB unambiguously/IRR associated/VERB with/IRR one/NUM tag/NOUN other/ADJ words/NOUN match/VERB multiple/ADJ tags/NOUN depending/VERB on/IRR the/IRR context/NOUN that/IRR they/ANA appear/VERB in/IRR ./. POS/MODAL taggers/VERB are/VERB widely/IRR used/VERB to/IRR assign/VERB a/IRR single/ADJ best/ADJ POS/NOUN to/IRR every/IRR word/NOUN in/IRR text/NOUN data/NOUN ./. When/IRR building/NOUN a/IRR POS/NOUN tagger/IRR human/ADJ beings/NOUN needs/VERB to/IRR make/VERB a/IRR set/VERB of/IRR decisions/NOUN some/IRR of/IRR which/IRR significantly/IRR impact/NOUN the/IRR accuracy/NOUN and/IRR other/ADJ performance/NOUN aspects/NOUN of/IRR the/IRR resulting/VERB engine/NOUN ./.
1. Concept List
2. Union Concept List
3. Pre-Processing Settings
4. Analysis Settings
4. Generalization Thesaurus
5. Meta-Network Thesaurus
6. Sub-Matrix Selection
7. Data on the data
1. Utilities
2. Stemming
3. Delete List
Parts of Speech Tagging
Tag Texts
Undo Tagging
assign a single best part of speech to every word in text data
Output POS Per Word

Table 12: POS per Word (part1)

Word	Tag	Frequency
.	.	4
a	irr	4
about	irr	1
accuracy	noun	1
and	irr	2
appear	verb	1
are	verb	1
aspects	noun	1
assign	verb	1
associated	verb	1
be	verb	1
beings	noun	1
benefit	verb	1
best	adj	1
building	noun	1
can	modal	2
category	noun	1
context	noun	1
data	noun	1
decisions	noun	1
depending	verb	1
engine	noun	1
every	irr	1
extraction	irr	1
from	irr	1
human	adj	1
impact	noun	1
in	irr	2
information	noun	1
knowledge	noun	1
language	noun	1
make	verb	1
many	adj	1
match	verb	1
multiple	adj	1
natural	adj	1

Table 13: POS per Word (part2)

Word	Tag	Frequency
needs	verb	1
of	irr	5
on	irr	1
one	num	1
other	adj	2
performance	noun	1
pos	modal	1
precise	adj	1
processing	verb	1
resulting	verb	1
set	verb	1
significantly	irr	1
single	adj	1
some	irr	1
syntactic	adj	1
tag	noun	1
tagger	irr	1
taggers	verb	1
tags	noun	1
tasks	noun	1
text	noun	1
that	irr	1
the	irr	4
they	ana	1
to	irr	3
unambiguously	irr	1
used	verb	1
variety	noun	1
when	irr	1
which	irr	1
while	irr	1
widely	irr	1
with	irr	1
word	noun	1
words	noun	3

Besides supporting a variety of NLP and IE routines, AutoMap's main purpose is to facilitate content analysis as well as the extraction of one- or multi-modal relational data from texts (Diesner & Carley, 2004, 2006; McConville et al., 2008). When relational data is extracted with AutoMap, outputs can be stored as DyNetML files (DyNetML is an XML derivate designed for graph representation (Carley, Diesner, Reminga, & Tsvetovat, 2007). DyNetML files represent one or multiple graphs that comprise vertices and edges. The nodes and edges can hold attributes. POS are one possible node attribute. ORA (Carley et al., 2007), a software for relational data analysis, can read DyNetML files and run several reports that consider POS in the computation of network analytic measures.

Internally, AutoMap uses POST as one out of multiple decision support features for:

1. Named Entities Extraction, which identifies relevant types of information that are referred to by a name, such as people, organizations, and locations.
2. Anaphora Resolution, which converts personal pronouns into the actual social entities that those pronouns refer to.

How can users exploit POST for text analysis projects? We envision a variety of potential usages:

1. Data reduction: Deleting non-content bearing words from texts: Though it ultimately depends on the user and application domain what the "non-content" set of words entails, such concepts often belong to one of categories that we aggregated in the NOISE class. Users can output the word-POS tuple table and add the words that are classified as IRR to a delete list. When applying a delete list, AutoMap searches the texts that are currently loaded for the words specified in the delete list and removes any matches by either dropping them completely or inserting a placeholder at the position where a word was removed (this choice is made by the user).
2. Data reduction: In order to remove noise that does not occur in word form, words being associated with the SYM can also be added to the delete list.
3. Named Entity Extraction: the AGENT class collects instances of individual agents from the user's data, and the ORG class comprises instances of organizations or mentions of multiple people. Retrieving these entities and performing network text analysis on them in AutoMap can help people to explore the social network(s) represented in their data. Since POST operates on a word-by-word basis, the identification of agents or organizations that occur as N-grams (e.g. *Henry Ford* or *Occupational Safety and Health Administration*) the user would need to parse the texts after POST for sequences in which the agent or organization tag are collocated.
4. Identification of social structure: One application of AutoMap is the approximation of the organizational structure that is represented in text data. Revealing and further analyzing such structure helps people in going beyond the identification of the social network configuration by also answering questions like: Who is located where, and what people or groups have access to what resources, tasks, and knowledge? Further analysis such information helps people to understand the benefits or risks that result for an organization from the revealed structure. For such projects, the words in the VERB class could serve as an initial list of events or tasks, the list of nouns can be screened for resources, and the MODAL group might serve as node attributes.

5. Identification of node attributes: AutoMap supports the extraction of two types of relational data: one-mode networks (in which all nodes are of the same type) and multi-mode networks (where nodes can be associated with different node-classes). By default, all nodes in a one-mode network belong to the node class *knowledge*, while in multi-mode networks, nodes can belong to one or multiple of the classes *agent*, *organization*, *task/event*, *resource*, *knowledge*, *location*, and *time*. One-mode network extraction has been used to reveal mental models of (groups of) people. Mental models are considered to represent the reality that people have in their minds and use to make sense of their surroundings, or the cognitive constructs that reflect people's knowledge and information about a certain topic. Multi-mode network extraction serves the exploration of network configuration as described in the previous point. People are not bound to those categories, but can use their own ontologies or taxonomies in AutoMap (Diesner & Carley, 2008). Whether using the default or self-defined node classification schemata, and whether extracting one- or multi-mode networks, people can also extract attributes on nodes. The ADJ class might be an appropriate candidate for providing suggestions for words that qualify as node attributes.

7. Limitations and Conclusions

We have shown how design decisions about computational solutions for common NLP tasks, here POST, can significantly impact the behavior of the resulting engine. The empirical comparison of four POS algorithms, which all are integral parts of the Viterbi algorithm, confirmed our assumption that an increase in the empirical evidence that an algorithm identifies and exploits causes increases in accuracy rates. Therefore, the upgrade from local search to global search leads to improvements in accuracy at the expense of higher computational complexity. This investment pays off most if the search space is traversed through for the best solution not only in a forward fashion, but with a bidirectional search.

Removing noise from the training data prior to learning a model leads to significant decreases in accuracy rates while the amount and numerical stability of the learnt probabilities for the tags of interest increase. We argue that the generalizability of the model benefits from the decision to remove noise.

Across all algorithms tested, the majority of errors was caused by algorithmic failures, while a lower portion was due to not labeling words newly encountered in the evaluation data. People who build POS taggers can impact the ratio of algorithmic errors to errors caused by seeing new words during evaluation only slightly, but can also develop post-processing rules for handling new words. The process of constructing and testing unknown handling rules is fairly cumbersome. The more an algorithm is designed to admit uncertainties rather than resolving them on its own the more hybrid strategies of initial algorithmic solutions plus manually constructed post-processing heuristics can improve accuracy.

Across all independent variables tested in this project we observed the strongest performance improvement when the tag set was aggregated into less categories that are tailored towards our needs. We therefore advocate the development of models and tools that allow end-users to specify or participate in the consolidation of categories out of a predefined pool of choices according to their needs.

We conclude that error rates reported on POS taggers and obtained by users who work with such tools highly depend on the decisions made during the design and implementation stage of a tagger. Therefore, the variables that significantly impact a tagger's performance need to be identified and their effect on the tagger needs to be measured and reported so that everyone – developers and users – can learn about the sensitivity of the engine.

Several limitations apply. First, model training and evaluation were performed by using separate portions of one data set. Even though this corpus contains more than a million data points, it still reflects a certain time period, style (journalistic writing) and range of domains (news paper articles). Applying the model to data that differs in these respects is likely to result in accuracy rates lower than the ones achieved herein. Second, we did not test MM of a higher order. For data sets with lengthy sentences, e.g. academic writing, or for data in that N-grams of size larger than size two are crucial and occur often, using a MM of a higher order might further improve tagging accuracy while also increasing computational complexity. Finally, all algorithms tested are stochastic taggers, so that a comparison to accuracy rates achieved with rule- or transformation-based systems could be valuable.

The POST algorithms that we implemented and tested for this project performed reasonably well on tagging texts unseen during model training. What does reasonably well mean? If for instance our tagger (trained on clean data, performing unknown handling, not using aggregated tag set) was used to tag a 20 word sentence, it would mislabeled two to three (precisely 2.3 words) words when using UM, and one to two words when using HMM (1.7), VitF (1.6) or VitB (1.5). If beyond the given configuration above the model was also trained on and using the aggregated tag set, it would mislabel about one (1.1 for UM, 1.4 for HMM, 1.2 for VitF and VitB) word in a 20-word sentence. Using the same tagger configuration, the probability that all words in 20 word sentence would get tagged correctly is 31% for UM and VitB, 24% for HMM, and 29% for VitF. Overall, our results do not match, but closely approximate the best accuracy rates reported for HMM- and Viterbi-based taggers (96% to 97%), as well as for Baseline taggers (90% to 91%) (Jurafsky & Martin, 2000).

References

Arguello, J., & Rose, C. P. (2006). Museli: A Multi-source Evidence Integration Approach to Topic Segmentation of Spontaneous Dialogue. *Proceedings of the North American Chapter of the Association for Computational Linguistics (short paper)*.

Atwell, E. (1987). Constituent-likelihood grammar. In R. Garside, G. Sampson & G. Leech (Eds.), *The computational analysis of English: a corpus-based approach*. London: Longman.

Baum, L. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities, 3*, 1-8.

Bikel, D., M. , Schwartz, R., & Weischedel, R., M. . (1999). An Algorithm that Learns What's in a Name, *Machine Learning* (Vol. 34, pp. 211-231): Kluwer Academic Publishers.

Carley, K. M., Diesner, J., Reminga, J., & Tsvetovat, M. (2007). Toward an interoperable dynamic network analysis toolkit. *Decision Support Systems. Special Issue on Cyberinfrastructure for Homeland Security, 43*(4), 1324-1347.

Church, K. (1988). *A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text.* Paper presented at the 2nd Conference on Applied Natural Language Processing, Austin, TX.

DeRose, S. (1988). Grammatical category disambiguation by statistical optimization. *Computational Linguistics, 14*, 31-39.

Diesner, J., & Carley, K. M. (2004). *AutoMap1.2 - Extract, analyze, represent, and compare mental models from texts*: Carnegie Mellon University, School of Computer Science, Institute for Software Research International, Technical Report CMU-ISRI-04-100o. Document Number)

Diesner, J., & Carley, K. M. (2006). Revealing Social Structure from Texts: Meta-Matrix Text Analysis as a novel method for Network Text Analysis. In V. K. Narayanan & D. J. Armstrong (Eds.), *Causal Mapping for Information Systems and Technology Research: Approaches, Advances, and Illustrations* (pp. 81-108). Harrisburg, PA: Idea Group Publishing.

Diesner, J., & Carley, K. M. (2008). Conditional Random Fields for Entity Extraction and Ontological Text Coding. *Journal of Computational and Mathematical Organization Theory*.

Dietterich, T. G. (2002). *Machine Learning for Sequential Data: A Review.* Paper presented at the Joint IAPR International Workshops SSPR 2002 and SPR 2002, August 6-9, 2002,, Windsor, Ontario, Canada.

Jurafsky, D., & Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*: Prentice Hall PTR.

Klein, D., & Manning, C. D. (2002). *Conditional structure versus conditional Estimation in NLP models.* Paper presented at the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP-02), Philadelphia, USA.

Krovetz, B. (1995). *Word sense disambiguation for large text databases.* University of Massachusetts, Amherst.

Kupiec, J. (1992). Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language, 6*, 225-242.

Lappin, S., & Leass, H. J. (1994). An algorithm for pronominal anaphora resolution. *Comput. Linguist., 20*(4), 535-561.

Manning, C., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

McConville, E., Diesner, J., & Carley, K. M. (2008). *Software demonstration of AutoMap*. Paper presented at the XXVIII Sunbelt Social Network Conference.

Mitchell, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a Large Annotated Corpus of English: The Penn Treebank *Computational Linguistics, 19*(2), 313-330.

Mitchell, P. M., Santorini, B., Marcinkiewicz, M. A., & Taylor, A. (1999). *Advances Treebank 3*: Linguistic Data Consortium, Philadelphiao. Document Number)

Piao, S. S. (n.d.). English sentence breaker. *http://text0.mib.man.ac.uk:8080/scottpiao/sent_detector*.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program, 14*(3), 130−137.

Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc. IEEE 77, 2*, 257-285.

Stolz, W. S., Tannenbaum, P. H., & Carstensen, F. V. (1965). Stochastic Approach to the Grammatical Coding of English. *Communications of the ACM, 8*, 399–405.

Viterbi, A. J. (1967). Error bounds for convolutional codes and asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory, 13*, 260-269.

Weischedel, R., Meter, M., Schwartz, R., Ramshaw, L., & Palmucci, J. (1993). Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, *19*(2), 359-382.

Appendix: PTB Tagset

PTB Tag	Meaning	Aggregated Tag	Instances in PTB
NN	noun, common, singular or mass	NOUN	161397
IN	preposition or conjunction, subordinating	IRR	136714
DT	determiner	IRR	116454
JJ	adjective or numeral, ordinal	ADJ	76586
NNP	noun, proper, singular	AGENT	62020
NNS	noun, common, plural	NOUN	55912
RB	adverb	IRR	52037
PRP	pronoun, personal	ANA	47303
VBD	verb, past tense	VERB	46684
CC	conjunction, coordinating	IRR	38097
VB	verb, base form	VERB	36887
VBN	verb, past participle	VERB	29435
TO	to as preposition or infinitive marker	IRR	26135
VBZ	verb, present tense, 3rd person singular	VERB	21627
VBG	verb, present participle or gerund	VERB	17255
PRP$	pronoun, possessive	IRR	16918
CD	numeral, cardinal	NUM	15178
VBP	verb, present tense, not 3rd person singular	VERB	14371
MD	modal auxiliary	MODAL	14115
:	:	SYM	10917
''	''	SYM	9201
``	``	SYM	8838
POS	genitive marker	POS	5247
WDT	WH-determiner	IRR	4990
WP	WH-pronoun	IRR	4732
WRB	Wh-adverb	IRR	4625
JJR	adjective, comparative	ADJ	2914
)	)	SYM	2506
(	(	SYM	2477
EX	existential there	IRR	2224
NNPS	noun, proper, plural	ORG	1958
RBR	adverb, comparative	IRR	1901
JJS	adjective, superlative	ADJ	1743
RP	particle	IRR	1630
SYM	symbol	SYM	1268
UH	interjection	IRR	883
FW	foreign word	FW	803
RBS	adverb, superlative	IRR	784
PDT	pre-determiner	IRR	728
$	$	SYM	579
LS	list item marker	SYM	446
WP$	WH-pronoun, possessive	IRR	251