Comparison of Unigram, HMM, CRF and Brill's Part-of-Speech Taggers Available in NLTK Library

Miroslav Potočár, Michal Kvet
University of Žilina, Žilina, Slovakia
Miroslav.Potocar, Michal.Kvet@fri.uniza.sk

***Abstract*—Part-of-speech tagging is for many NLP researchers the first task they encounter in the field of natural language processing. This task is undoubtedly related to part-of-speech taggers. We focus on a detailed description of the functioning of the unigram, hidden Markov model, conditional random fields and Brill taggers, followed by a comparison of these models. We use implementations available in the natural language toolkit library, without addressing the selection of the best parameters. We focus on finding out which tagger produces the best results using default settings or in other words, which one works best in "take it as it is" mode. To determine this, we make an experiment in which we track various metrics such as prediction time, accuracy on unknown words, number of correctly labeled sentences and others. From the results of the experiment, we find out that the CRF tagger achieves the highest accuracy among all participants in the experiment. It is also able to tag previously unseen words with the highest accuracy among all taggers compared.**

I. Introduction

Upon the arrival of chatbots such as ChatGPT [1], [2] or LaMDA [3], an increased interest in the area of natural language processing (NLP) can be expected. Many researchers dive into the uncharted waters of this field and encounter new concepts, models, and algorithms. However, sooner or later, everyone faces the task of part-of-speech tagging, which is the foundation for many other tasks within NLP. The need may arise to create one's own tagset or annotate one's own text. At that point, many will look for available libraries and come across the open-source natural language toolkit (NLTK) library, available for Python. However, time is limited, and we cannot waste it by selecting models and tuning parameters. From the part-of-speech taggers available in NLTK, we want to find the one that achieves the best results with the least effort.

Since part-of-speech tagging is a fundamental task in NLP, we decided to look at some of the taggers available in the NLTK library. We chose the unigram, hidden Markov model (HMM), conditional random fields (CRF) and Brill taggers. We collected information about each of these models across available sources, and based on that, we composed a comprehensive description of their operation. We then exposed each model to an experiment, compared the results with each other and we also provided possible reasons why some models had values that differed from the others.

Part-of-speech tagging, which is a fundamental task in NLP, is a well-explored area. There are many articles and books that deal with this issue. Some are focused on specific types of taggers [4]–[14], while others are focused on developing or comparing taggers in other languages different from English [15]–[22]. Despite of many existing publications on this topic, we felt the need to gather information about the unigram, HMM, CRF, and Brill taggers and compare the performance of these models in annotating English texts. We chose these primarily because they represent the basis for many advanced taggers. Our personal observation suggests that they provide an easier understanding of part-of-speech tagging task and thus, serve as a good stepping stone into the world of NLP.

In our research, we focused solely on selected implementations of part-of-speech taggers available in the NLTK library. We used these with default settings, meaning we did not try to find the most suitable parameter values for model training. We wanted to find out how the individual models perform in "take it as it is" mode. The experiments in this study ran only on the Penn Treebank corpus and using the Penn Treebank tagset. For more objective results, it would be appropriate in the future to compare performance on different corpora and using multiple tagsets. When evaluating individual taggers, we used only a limited set of metrics, which could be expanded in future research.

In section II, we briefly describe the task of part-of-speech tagging. In the next section III, we describe what a part-of-speech tagger is and what properties are required of it. Section IV deals with a detailed description of the way how each of tagger involved in the comparison works. In section VI, we describe the methodology used in conducting the experiment focused on comparing taggers. Section VII contains the actual results of the experiment. The next section VIII includes an evaluation of the experimental results, along with considerations on the reasons for the significantly different observed values. In the last section IX, we state that, in terms of accuracy and given the conditions of our experiment, the CRF tagger appears to be the best choice. In addition, we mention the limitations of our study and make recommendations for further research.

II. Part-of-speech tagging

Part-of-speech tagging represents one of the most important steps in NLP. Information about part-of-speech tags allows for high-level analysis, such as recognition of noun phrases and other patterns in text. Therefore, it forms the basis for other NLP tasks, such as named entity recognition, semantic analysis etc.

Part-of-speech tagging (POS tagging) is the process of assigning a part-of-speech (POS) tag to each word in a text. The input is a sequence of words $x_1, x_2, ..., x_n$ together with a set of tags (tagset). The output of the process is a sequence of labels $y_1, y_2, ..., y_n$, where each output y_i corresponds to exactly one input x_i. Thus, the aim is to find the correct tag for each given situation. [23]

This is a disambiguation task. Words are often ambiguous in their parts-of-speech. The English word *"store"* can be understood as a noun, a finite verb or an infinitive. In speech, this ambiguity is typically resolved by the context in which the word occurs. [24]

Jurafsky and Martin [23] examined the Brown and WSJ corpora and found that the majority of words (85-86%) are unambiguous. This means that around 14-15% of the vocabulary in these corpora consists of ambiguous words. However, these words are very common: 55-67% of the tokens in a text are ambiguous. Common words such as *"that"*, *"back"*, *"down"* and *"put"* are among the ambiguous words that occur very frequently in texts [23].

Part-of-speech tagging for English is well-studied, and many taggers that have been developed achieve accuracy exceeding 98% [15].

III. Part-of-speech tagger

Part-of-speech tagger is a system that uses context in order to assign parts-of-speech to corresponding words. The tagger assigns a (unique or ambiguous) part-of-speech tag to each token input and returns this output for further processing [4].

According to [8], a tagger, in order to function as a practical component in a language processing system, must meet the following properties:

- **Robustness** - Corpora often contain words that the tagger has not seen before. It is necessary for the tagger to be able to handle such situations as best as possible.
- **Efficiency** - If the tagger is to be used for processing very large corpora, it must be efficient. Performance should be linear in time with respect to the number of words. If training is required, it should also be fast, allowing for a quick transition to a new corpus or text genre.
- **Accuracy** - The tagger should attempt to assign the correct part-of-speech tag to every word it encounters.
- **Tunability** - The tagger should be able to benefit from linguistic knowledge. It should be possible to correct systematic errors by providing appropriate guidance. It should also be possible to provide different hints for different corpora.

According to [25], the architectures of taggers are quite similar and consist of these parts:

- **Tokenization** - The input text is divided into tokens suitable for further analysis.
- **Searching for ambiguity** - Includes the use of a lexicon and a guesser, which is used on tokens that are not found in the lexicon.
  - The simplest case is when a lexicon is a list of word forms and their possible parts-of-speech. More efficient solutions are based on finite-state models.
  - The guesser analyzes the remaining tokens, which are the tokens that were not found in the lexicon (e.g., unknown words that did not occur in the training corpus). The design of the guesser is often based on what is known about the lexicon. For example, we know that the lexicon contains all closed-class words (pronouns, articles, etc.), so we can design the guesser to only handle open-class analysis (nouns, verbs).
  - Using a compiler, lexicon and guesser, a lexical analyzer is formed, which provides meaningful analyses and alternatives for each of the tokens.
- **Distinguishing ambiguity and ambiguity resolution** - Disambiguation is performed based on two information sources:
  1) Information about the word itself - for example, that the word *"tables"* is more commonly used as a noun than as a verb.
  2) Contextual information about word/tag sequence. For example, the model may prefer to perform noun analysis before verb analysis if the previous word was a preposition or article.

Researchers can choose from a wide range of available part-of-speech taggers. The decision for a particular tool is influenced primarily by tagging accuracy, but also by other practical issues such as ease of use, applicability to the target language or domain, availability for a certain hardware platform or other factors that may affect the choice [21].

IV. Characteristics of comparison study participants

In this section, we present the models that will participate in our experiment. For each of them, we provide a comprehensive description of way they work.

A. Unigram tagger

The unigram tagger is a simple statistical tagging model. It assigns each token the tag that is most likely for that token. Before this tagger can be used to tag data, it must be trained on a training corpus. This corpus is used to determine which tags are most common for a given token. This tagger assigns a default tag of "None" to any token that it did not encounter during training [20].

B. Hidden Markov model

Hidden Markov model is a modeling technique for linear problems such as sequences or time series and is widely used in applications for speech recognition. It is a generative model, meaning that it focuses on how the entire sequence was generated. It is based on the extension of a Markov chain [9]–[11], [14], [23], [26].

Markov chain is a mathematical model used to describe the probabilities of sequences, states, where each state can

take values from some set. These sets can be words, tags or symbols representing anything. It is a type of Markov model where the future state of the system depends only on the current state and not on the entire sequence of preceding states. The Markov model represents a broader concept that refers to any probabilistic model which assumes that the future is conditionally independent of the past given the present.

Markov chain makes a strong assumption, also called the **Markov assumption**. This assumption states that the future state of the system depends only on the current state and not on the previous states. In other words, the future is conditionally independent of the past given the present. This assumption simplifies the modeling of complex systems by reducing the number of variables that need to be taken into account when predicting the future state. This allows for efficient calculation of the **transition probabilities** and prediction of future outputs based on previous states. The assumption is formulated as follows [23]:

$$P(q_i = a|q_1...q_{i-1}) = P(q_i = a|q_{i-1}) \quad (1)$$

Where q_i in equation 1 represents the state at time *i* and $q_1...q_{i-1}$ represents the sequence of states preceding the state q_i.

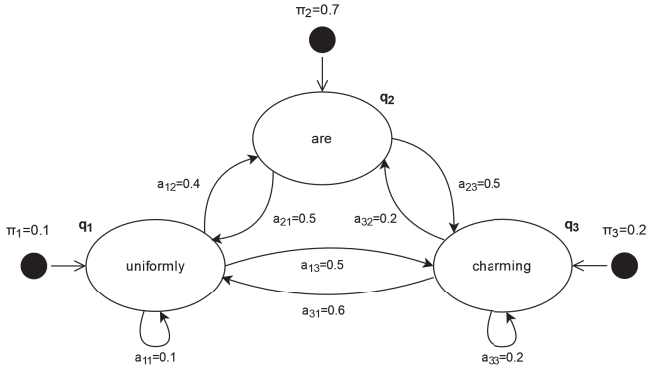

Fig. 1. Markov chain for words. Adapted from [23].

The notation of individual components of the Markov model varies across publications [23], [26]. In Fig. 1, we used the notation from [23], where each component represents:

- $Q = q_1, q_2, ..., q_N$ - The set of states in which the system can appear. It is a set of size *N*. In our case, this set contains *Q*={*uniformly, are, charming*} and *N=3*.
- $A = a_{11}, a_{12}, ..., a_{N1}, ..., a_{NN}$ - Transition probability matrix A, where element a_{ij} represents the probability of transitioning from state *i* to state *j*. Since this is a probability, the following conditions must be met:

$$\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$$
$$a_{ij} \geq 0 \quad \forall\, i, j$$

- $\pi = \pi_1, \pi_2, ..., \pi_N$ - Initial probability distribution across states. State π_i represents the probability that a Markov chain will begin in state *i*. Some states *i* may have a probability of $\pi_i = 0$ which means that they cannot be initial states. As this is a probability, conditions must be satisfied in this case as well:

$$\sum_{i=1}^{n} \pi_i = 1$$
$$\pi_i \geq 0 \quad \forall i$$

The Markov chain is useful when we want to calculate the probability for a sequence of observable states. However, there are many cases where the events of interest are hidden, meaning that we cannot observe them directly [23]. An example is the determination of part-of-speech tags in a sentence. We can only directly observe the sequence of words, but the part-of-speech tags of the words themselves remain hidden. Tags are hidden because they cannot be easily determined from the input. To infer them, we use the input observable sequence.

In a Markov chain, each state corresponds to a deterministically observable event (i.e., the output symbol in a given state is not random). The natural extension of a Markov chain introduces a non-deterministic process that generates observable output symbols in any state. This extended Markov chain is known as a hidden Markov model. An HMM is simply a Markov chain where the output observed symbol is a random variable *X* generated according to the output probability function associated with each state [26].

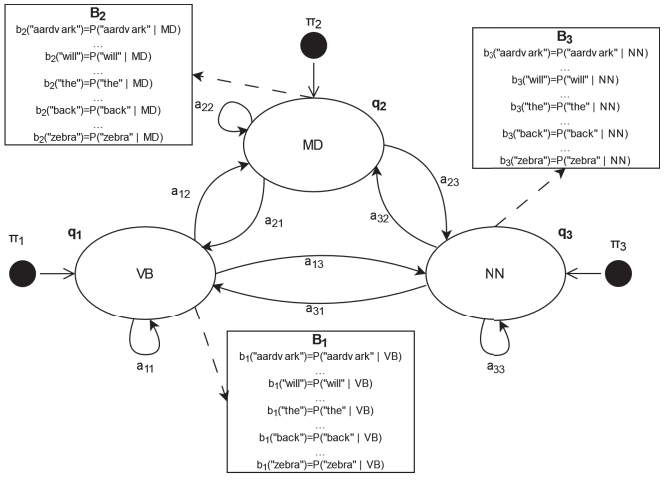

Fig. 2. Illustration of HMM for part-of-speech tagging.Adapted from [23].

As with Markov models, the notation for the individual components of an HMM varies across publications [14], [23], [26]. In Fig. 2, we use the notation used in [23], where the individual components represent:

- $Q = q_1, q_2, ..., q_N$ - The set of states in which the system can appear. It is a set of size *N*. In our case, *N=3*.
- $A = a_{11}, a_{12}, ..., a_{N1}, ...a_{NN}$ - Transition probability matrix A, where the element a_{ij} represents the probability of transitioning from state *i* to state *j*. In our case, it represents the transition between part-of-speech tag *i* and

a tag *j*. As it is a probability, the following conditions must be satisfied:

$$\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$$
$$a_{ij} \geq 0 \quad \forall\, i, j$$

- $O = o_1, o_2, ..., o_T$ - A sequence of observations with length *T*. Each observation comes from a vocabulary $V = v_1, v_2, ..., v_V$. In our case, an observation represents a single word.
- $B = b_i(o_t)$ - Sequence of observation probabilities. It is also called the emission probability, and each one expresses how likely the observation o_t was generated from state q_i. In our case, it will be the probability of observing a word given that the word has a specific part-of-speech tag. Since this is a probability, the following condition must be met:

$$\sum_{t=1}^{T} b_i(o_t) = 1$$
$$b_i(o_t) \geq 0 \quad \forall i, t$$

- $\pi = \pi_1, \pi_2, ..., \pi_N$ - Initial probability distribution across states. State π_i represents the probability that the Markov chain starts in state *i*. Some states *i* can have probability $\pi_i = 0$ which in other words means they cannot be the initial state. As it is a probability, the following conditions must be satisfied:

$$\sum_{i=1}^{N} \pi_i = 1$$
$$\pi_i \geq 0 \quad \forall i$$

The complete specification of an HMM includes two constant parameters *N* and *T*, which represent the total number of states and the size of the observed vocabulary, the observed vocabulary *O* and three probability matrices A, B, π. The following notation is typically used to indicate the entire set of parameters [26]:

$$\Phi = (A, B, \pi)$$

Sometimes, the symbol Φ is also used to represent the hidden Markov model itself.

In HMM, we also encounter the term "order of HMM." Order refers to the number of previous states taken into account when determining the probability of the current state. First-order and second-order HMM are commonly used. In a first-order HMM, only the current state is considered when determining the next state. In a second-order HMM, both the current and previous states are taken into account. In theory, HMMs of higher orders than two exist, but these models are rarely used in practice due to their computational and data complexity.

For the part-of-speech tagging task, a first-order HMM is used. This simple and computationally efficient assumption that the next state depends only on the current state is often sufficient to capture the dependencies relevant to part-of-speech tagging.

In a first-order HMM, two assumptions are made [23], [26]:

1) Markov assumption for a Markov chain described in equation 1.
2) Output independence assumption:

$$P(o_i|q_1..., q_i, ..., q_T, o_1, ..., o_i, ..., o_T) = P(o_i|q_i)$$

which states that the probability of the output observation o_i depends only on the state q_i that produced the observation and not the previous states or observations.

In general, when using HMM, we want to solve one of the following problems [9], [14], [26]:

1) **The Evaluation Problem** - Given a model Φ and a sequence of observations $O = (o_1, o_2, ..., o_t)$, we aim to find out with what probability $P(O, \Phi)$ the given model generated this sequence of observations.
2) **The Decoding Problem** - Given a model Φ and a sequence of observations $O = (o_1, o_2, ..., o_t)$, we want to determine the most probable sequence of states $Q = (q_0, q_1, q_2, ..., q_t)$ in the model that produced this sequence.
3) **The Learning Problem** - Given a model Φ and a set of observations, we try to adjust the parameters $\hat{\Phi}$ to maximize the joint probability $\prod_O P(O|\Phi)$.

For part-of-speech tagging, we are solving the decoding problem. We are trying to find the most probable sequence of tags $t_1, ..., t_n$ (states sequence) given the sequence of observations of *n* words $w_1, ..., w_n$ (observation sequence) [23]:

$$\hat{t}_{1:n} = \underset{t_1...t_n}{argmax} P(t_1...t_n|w_1...w_n)$$
$$\approx \underset{t_1...t_n}{argmax} \prod_{i=1}^{n} \overbrace{P(w_i|t_i)}^{emission} \overbrace{P(t_i|t_{i-1})}^{transition}$$

The **Viterbi algorithm** is used for efficient decoding.

C. Conditional Random Fields

HMM represents a powerful and useful model, but it has been shown that achieving high accuracies requires a lot of augmentation. In the task of PoS tagging, we often encounter unknown words, such as proper nouns, abbreviations and even verbs that enter the language at a surprising pace. It would be nice to have a way to add arbitrary features that could address this problem with unknown words [23]. Such a feature could be the initial letter case of a word, as proper nouns are generally capitalized or information about affixes, as the suffix *-ed* is usually present in past tense verbs. Information about the preceding or succeeding word can also be a useful feature (e.g., if the preceding word was an article, the succeeding part-of-speech tag will not be a verb).

We can attempt to make changes to HMM and find a way to include some of these features. However generally, it is challenging for generative models such as HMM to directly add such features to the model in a clear way. Log-linear models, on the other hand, can combine such features in a principled way. An example of such a model is **logistic regression**, but it is not able to process a sequential data. However, there is a **discriminative** sequential model based on log-linear models, **conditional random fields** [23].

CRF is a framework for creating probabilistic models for sequence labeling and segmentation [12], [27]. Simply put, it is a conditional distribution $p(y|x)$ associated with a graphical structure [28]. It takes the form of an undirected graphical model that defines a single log-linear distribution over sequence of labels given a particular observed sequence. The main advantage of CRF over HMM is their conditional nature, which allows relaxing the strong independence assumption that HMM requires to ensure inferable conclusions.

CRFs are a special case of Markov random fields [7], which are an undirected graphical model that satisfies the Markov property. In the case of CRF, we can view it as an undirected graph, globally conditioned on **X**, which is a random variable representing observed sequences. In the case of part-of-speech tagging, **X** ranges over natural language sentences [12]. We define an undirected graph $G = (V, E)$, where each vertex $v \in V$ corresponds to each random variable, representing an element Y_v of **Y** [27]. It is assumed that all components Y_v of **Y** range over a finite set of labels $\mathcal{Y}$ [12]. **Y** in the case of part-of-speech tagging ranges over part-of-speech taggings of sentences, and $\mathcal{Y}$ is the set of all possible part-of-speech tags [12]. If each random variable Y_v obeys the Markov property with respect to *G*, then (**Y**, **X**) is a CRF [27]. The structure of the graph *G* can be theoretically arbitrary, but it must provide a representation of the conditional independencies in the modeled tagging sequences. The absence of an edge between two vertices in *G* implies that the random variables represented by these edges are conditionally independent with respect to all other random variables in the model [27].

The graphical structure of a CRF allows the joint distribution to be factorized over the elements Y_v of **Y** into a normalized product of **feature functions**. A feature function operates on a subset of random variables represented by vertices in *G* [27]. Feature functions must ensure that the joint probability can be factorized such that the conditional independent random variables do not appear in the same feature function. The way to achieve this requirement is to require each feature function to operate on a set of random variables whose corresponding vertices form a maximum clique in the graph *G* [27].

Individual feature functions have no probabilistic interpretation. They represent constraints placed on sets of random variables on which the function is defined [27]. However, this affects the probability of the global set, where a higher probability belongs to the set where more of these constraints are satisfied.

When modeling sequences, the simplest graph encountered is usually a linear chain, where *G* is a simple chain in which the vertices corresponding to elements of **Y** form a linear chain [12], [27]: $G = (V = \{1, 2, ..., n\}, E = \{(i, i+1)\})$. This graph is shown in Fig. 3. The limitations of this structure, known as a **linear chain CRF**, allow for versions of the efficient **Viterbi algorithm** and **Forward-Backward** algorithm from HMM to be used. In contrast, general CRFs allow for connections to exist between any two vertices, which is necessary for tasks where the decision depends on distant vertices, such as Y_{i-4}.

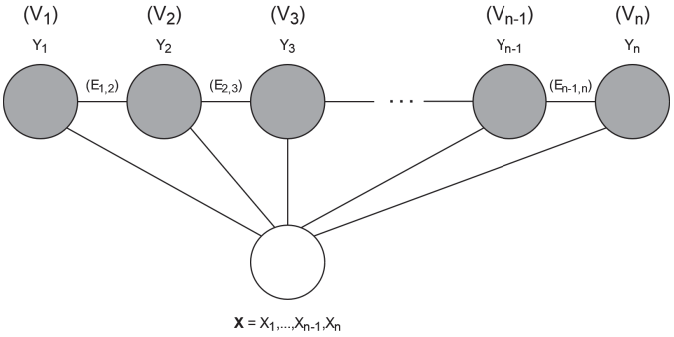

Fig. 3. Structure of first-order chain CRFs. Adapted from [27].

In the case of CRF with chain structures, the entire observed sequence **x**, the position *i* within the sequence, and the labels at positions *i* and *i-1* are input into the feature function. In some publications [12], [27], these feature functions are divided into:

- **Transition feature functions**, which take the entire observed sequence and the labels at positions *i* and *i-1* in the label sequence as input. This function has the form $t_k(y_{i-1}, y_i, \boldsymbol{x})$. An example in the field of part-of-speech tagging could be:

$$t_k(y_{i-1}, y_i, \boldsymbol{x}) = \begin{cases} 1 & \text{if } y_{i-1} = \text{DET and } y_i = \text{NOUN} \\ 0 & \text{otherwise} \end{cases}$$

- **State feature function**, which takes the position *i* and the entire observation sequence **x** as input. It has the form $s_k(y_i, \boldsymbol{x}, i)$. An example in the field of part-of-speech tagging could be:

$$s_k(y_i, \boldsymbol{x}, i) = \begin{cases} 1 & \text{if } y_i = \text{DET and the observation on} \\ & \text{position } i \text{ is word } x_i = \text{"the"} \\ 0 & \text{otherwise} \end{cases}$$

For simplicity, it is not necessary to distinguish between these two types of feature functions, and therefore they can be written uniformly as $f_k(y_{i-1}, y_i, \boldsymbol{x}, i)$.

Using the feature functions, it is possible to calculate the probability of the label sequence **y** given the observation sequence **x**:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{exp\left(\sum_{k=1}^{K} w_k F_k(\boldsymbol{x}, \boldsymbol{y})\right)}{\sum_{\boldsymbol{y}' \in Y} exp\left(\sum_{k=1}^{K} w_k F_k(\boldsymbol{x}, \boldsymbol{y}')\right)} \quad (2)$$

Where these K functions $F_k(\boldsymbol{x}, \boldsymbol{y})$ are called **global features**, because each represents a property of the entire input sequence **x** and output sequence **y**. We obtain them by summing the **local features** for each position *i* in the sequence **y**.

$$F_k(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} f_k(y_{i-1}, y_i, \boldsymbol{x}, i)$$

Each of these global features is associated with a weight w_k, which is learned during training phase.

The denominator in equation 2 goes through all possible output sequences **y**. It is called the normalization factor and ensures that the result is in the range 0 to 1. This denominator is usually pulled out into a function [12], [23], [27], [28]:

$$Z(\boldsymbol{x}) = \sum_{\boldsymbol{y}' \in Y} exp\left(\sum_{k=1}^{K} w_k F_k(\boldsymbol{x}, \boldsymbol{y}')\right)$$

D. Brill's tagger

Stochastic taggers such as HMM, CRF have many advantages, but perhaps the most obvious one is that they do not require demanding manual construction of rules to capture useful information. However, they also have their disadvantages, including memory requirements, since linguistic information is captured indirectly in large tables of statistics. Another problem is the demanding search and implementation of improvements to these models. Another cons is poorer portability from one tagset or genre corpus to another. Many of these downsides can be eliminated by Brill's tagger [5], which combines rule-based and transformation-based approach. This tagger surpasses common rule-based NLP approaches because it is robust and the rules are automatically acquired.

As mentioned, this is a combination of rule-based and transformation-based taggers. Rule-based because an initial-state annotator is part of this tagger, which assigns a part-of-speech tag to each word based on certain rules. The tagger then applies a series of transformations to the initially annotated text, which, based on the context of words in the sentence, attempt to correct the initial tag assignments, leading to increased accuracy of the tagger. This is why it is also a transformation-based tagger.

To train and evaluate the tagger, we need three corpora [15]: [15]:

- A large high-quality tagged training corpus, used for training,
- a smaller tagged corpus called a patch corpus, used for creating patches,
- a test corpus, used to evaluate the tagger.

During training, a list of patches is created based on a patch corpus, which is subsequently applied to the output of the initial-state annotator. In the original version of the tagger [5], transformations were added that were created based on 8 prespecified patch patterns, which had the following form:

- if a word has tag **a** and is in context **C**, then change the tag to **b**, or
- if a word has tag **a** and has lexical property **P**, then change the tag to **b**, or
- if a word has tag **a** and a word in region **R** has lexical property **P**, then change the tag to **b**.

In a later version of the tagger [6], the original transformations were extended to include:

- **contextual transformations**, which could refer to both words and part-of-speech tags,
- **unknown-word transformations**, which contained patch templates used for words not seen in the training corpus.

The process of training the tagger takes place in several steps:

1) In the first step, the initial-state annotator is trained. In the original version of the tagger [5], the initial-state annotator determined the most probable tag for each word and used this information to initially assign states to the input text. Additionally, the tagger contained two procedures to improve performance, both of which did not use any contextual information. One procedure provided information that if a word not seen in the training corpus had a capital letter, it is likely a proper noun, and attempted to correct errors based on this. The second procedure try to assign tags to words not in the training corpus by assigning them the most common tag for words that end with the same three letters. In later versions of the tagger, a different initial-state annotator could be used. The complexity of the initial-state annotator ranged from random output assignment to a sophisticated manually crafted rules for tag assignment [6].
2) In the next step, the patch corpus is annotated based on the trained model of the initial-state annotator. Transformations from a pre-specified list of templates are applied to the initially annotated data in sequence. The annotated text of the patch corpus is compared to the true labels, generating a list of tagging errors. The list consists of triples $< tag_a, tag_b, number >$ that indicate how many times the tagger incorrectly labeled the word with tag_a when it should have been labeled as tag_b based on the true labels.
3) Subsequently, for each error triple, the patch template is determined that led to the highest score after its application. In the original version of the tagger [5], this score was represented by error reduction, which was calculated as the difference between the number of corrected words after patch application (the incorrect tag of the word was changed from tag_a to the correct tag tag_b) and the number of new errors that appeared after patch application (the correct tag of the word tag_a was changed to the incorrect tag tag_b). This was limiting, so in later versions, the user could choose a scoring function for comparison and transformation selection
4) The transformation with the highest score from all combinations of $tag_a, tag_b, number$ triples and transformations is selected. If its application results in an

error reduction above a predetermined threshold, the transformation is added to the ordered list, and the training process on the patch corpus is repeated.

The entire training process is depicted in a flow-chart diagram in Fig. 4.

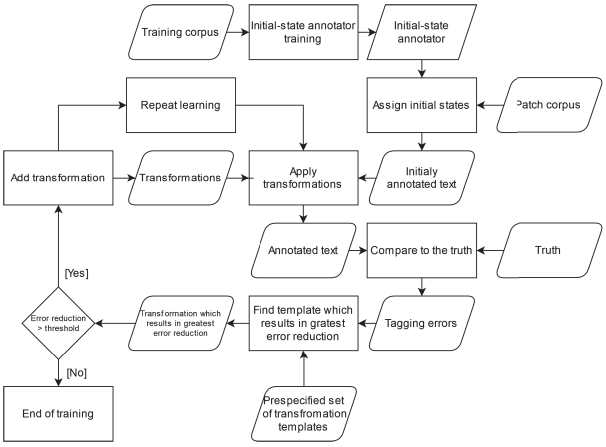

Fig. 4. Flow-chart diagram of the Brill tagger training process. Adapted from [6].

Once training is complete and a ordered list of transformations is generated, new texts can be annotated. The text is first labeled using the initial-state annotator and then all learned transformations are applied sequentially one by one.

V. Natural Language Toolkit library

NLTK is a popular open-source Python library for natural language processing that provides a comprehensive set of tools and resources for building NLP applications. It offers a wide range of functions and modules for working with text data, including tokenization, part-of-speech tagging, chunking, named entity recognition and more. It also provides an interface for working with popular corpora such as the Brown Corpus and Penn Treebank, and allows access to various pre-trained datasets and models.

VI. Methodology

In this section, we will describe the methodology of our experiment. We will introduce the data on which the experiment was run, the manner in which the experiment was performed, and the metrics that we monitored for each tagger.

A. Data

In the experiment, we used a subset of the Penn Treebank corpus, which is a widely used benchmark dataset for part-of-speech tagging. It contains approximately 4.5 million words of American English, including texts from various genres such as news articles, books and conversation transcripts [29]. The texts in this corpus were manually annotated with detailed linguistic information, including part-of-speech tags, syntactic structures and named entities.

We used a portion of this corpus that is directly available through the NLTK library. The subset contains 5% of the Penn Treebank corpus, corresponding to 3914 tagged sentences.

The tags used come from the Penn Treebank tagset, which is a widely used tagset for part-of-speech tagging. It consists of 36 tags that are used to label each word in a sentence with its part-of-speech tag, providing information on the grammatical function of the word in the sentence.

B. Model training

The implementation of taggers available in the NLTK library was used. For the unigram, HMM, and CRF models, predefined settings and parameters were used for their creation and training.

In the case of the Brill's tagger, an initial-state annotator was required, as well as a list of templates used to create transformations. The selection of the set of templates depends on the specifics of the task and the characteristics of the data used. There is no universal set of templates that would be optimal for every case.

In our case, the UnigramTagger was chosen as the initial tagger, and the set of templates used was "fntbl37". It contains a wide range of rule types, such as unigrams, bigrams, trigrams as well as contextual rules that take into account surrounding words in the sentence. There are several reasons why "fntbl37" may be a good choice for part-of-speech tagging. These include efficiency, generality, and the fact that "fntbl37" is part of the NLTK library, making it very easy to use.

C. Experimental setup

In order to obtain accurate results, the experiment was run 1000 times. In each run, the tagged sentences from the Penn Treebank corpus were divided into training and testing sets. The training set represented 80% and the testing set 20% of the original set. For each such division, we store the number of sentences in the training set, the number of tokens in the training set, the number of sentences in the testing set, the number of tokens in the testing set and the number of unknown words. The average values across the 1000 runs are shown in Table I.

TABLE I. Average values of data set descriptive characteristics

No. of train sentences	No. of train tokens	No. of test sentences	No. of test tokens	No. of unknown tokens
3131	80550.449	783	20125.551	1501.13

Subsequently, each of the models was trained on the training set and tested on the previously unseen testing set.

D. Evaluation

The testing set was used to evaluate the models. Tagging was performed sequentially. For each model, sentences from the testing set were iterated through in a cycle, and the corresponding tagger tagged the sentence. The labeled sequence was then added to the list of predicted sentences. After tagging

all samples in the testing set, the resulting list of sentences tagged by the taggers was used to evaluate the performance of the model.

The performance of each model was evaluated based on several metrics. We observed:

- **Average training time** - The average time it took to train the model.
- **Average prediction time** - The average time it took for the model to label all tokens in the test set.
- **Average number of correct tokens** - The average number of tokens that the model correctly labeled.
- **Average number of correct unknown tokens** - The average number of unknown tokens that the model correctly labeled.
- **Accuracy** - The proportion of correctly predicted tokens to all tokens.
- **Unknown tokens accuracy** - The proportion of correctly predicted unknown tokens to all unknown tokens.
- **Correctly tagged sequences** - The number of sentences that were correctly tagged.
- **Correctly tagged sequences accuracy** - The proportion of sentences that were correctly tagged to the total number of sentences.

By correctly tagged sequences, we mean the case when the tagger was able to correctly assign a part-of-speech tag to each token in the sentence, thus correctly identifying the entire sequence of tags. We consider it appropriate to include this metric since many taggers achieve more than 97% accuracy in tagging tokens, but the accuracy of labeling entire sequences is around 56% [30]. High accuracy in determining tags for individual tokens is also due to the fact that punctuation marks are included, which artificially increases accuracy.

VII. Results

The results of the experiments revealed that unigram, HMM, CRF and Brill's taggers performed differently in the part-of-speech tagging task. The performance of each tagger was evaluated based on several metrics, including accuracy, accuracy on unknown tokens, the average number of correctly tagged sequences and others. All metrics along with their descriptions can be found in subsection VI-D. Overall, CRF tagger performed the best in terms of accuracy. It had the best results in assigning tags and was also able to correctly tag a significant portion of tokens that did not appear in the training set.

VIII. Discussion

Regarding prediction speed, the unigram tagger has the fastest average prediction time. However, the accuracy of this tagger is the lowest among the other tested taggers. This suggests that when selecting a part-of-speech tagger, there may be a trade-off between accuracy and tagging speed. Another interesting observation about this tagger is its inability to predict tokens that did not appear in the training set. This is understandable given the functioning principle of this tagger, which assigns a tag to words based on the tag they most frequently occur with in the training set. Unknown words are assigned the tag *None*. This limitation could be improved to some extent by assigning a tag to unknown words based on the most frequently occurring tag across the training set. Of course, more sophisticated methods could also be used to solve the problem of assigning tags to previously unseen words.

TABLE II. Average values of metrics for each tagger

	Taggers			
Metric	**Unigram**	**HMM**	**CRF**	**Brill**
Training time [s]	0.136352	0.124735	33.542999	7.408718
Predicting time [s]	0.013648	4.154508	0.136719	0.195736
Correct tokens	17685.687	18350.972	19089.906	18078.205
Correct unknown tokens	0	559.167	1231.530	0
Correct sequences	72.201	146.737	254.298	112.552
Accuracy [%]	87.8773	91.1824	94.8538	89.8277
Unknown tokens accuracy [%]	0	37.2440	82.0370	0
Correct sequence accuracy[%]	9.2211	18.7404	32.4774	14.3745

An interesting value is found in the column for the HMM tagger. Specifically, the average prediction time is several times higher than that of the other taggers. One reason for this result may be that although HMM typically has a simpler model structure compared to CRF, it may require more computations during the inference phase to calculate probabilities for each possible sequence of tags. HMM uses the Viterbi algorithm to compute the most probable sequence of tags for a given input, which requires computing a table of probabilities for each possible tag at each position in the input sequence. This can be computationally demanding for long input sequences or large tagsets. In our case, we used the Penn Treebank tagset, which contained 36 tags, and the test sentence consisted of an average of approximately 25.7 tokens. Whether this is the reason for the high prediction time is left to the reader's consideration. Another reason may be the specific implementation of HMM in the NLTK library. It is possible that the implementation of HMM is suboptimal in terms of efficiency.

The experiment results suggest that the CRF tagger with default parameters available in the NLTK library outperformed other part-of-speech taggers, also available in the NLTK library and with default parameters, in terms of accuracy. It achieved a high overall accuracy (94.85%), but also significantly better accuracy in tagging unknown tokens (82.04%) compared to the other taggers in the experiment. As a result, it was able to correctly tag an average of 32.48% of sentences. Its disadvantage is a considerably longer training time. However, the longer training time compared to other taggers is understandable, as CRF is a more complex and computationally

demanding tagger than Unigram, HMM or Brill's taggers. CRF is a discriminative model that takes into account all combinations of feature values and output labels when making predictions, which makes it more computationally demanding. The remaining taggers are generative models that consider only the probabilities of input features. Although the training time takes more than in the case of other taggers, the accuracy achieved by this tagger is better than that of other taggers.

In the case of the Brill's tagger, we achieved only slightly better accuracy than with unigram tagger. Such results are understandable, since Brill's tagger used unigram tagger as an initial annotator, which it then tried to improve by applying transformations learned during the training phase. The results table also showed that Brill's tagger using the "fntbl37" template set is not able to tag unknown words. This fact could have led to a poor overall accuracy of the tagger. The reason why the tagger was not be able to tag unknown words may be related to the used template set, which is designed to capture common patterns in the data and does not have rules for dealing with unknown words. In our experiment, the tagger assigns a value of *None* to unknown words, just like the unigram tagger. This may also be related to the fact that we used unigram tagger without extensions to deal with unknown words as an initial annotator. This limitation could be addressed by choosing a more sophisticated model as an initial annotator that can to some extent deal with unseen words. Adding additional transformation rules that apply to unknown words could also help.

IX. Conclusion

In our study, we focused on providing a comprehensive introduction to several part-of-speech taggers and their subsequent comparison. We looked in detail at how unigram, HMM, CRF and Brill taggers work. We wanted to find out how well the implementation of these taggers available in the NLTK library perform. When comparing, we used taggers with default settings. We performed 1000 runs of the experiment, in which for each tagger we observed metrics such as overall accuracy, accuracy on unknown tokens, training time, prediction time, accuracy of correctly predicted sequences and others.

In terms of accuracy, the CRF tagger appears to be the best choice, with an overall accuracy of approximately 94.85%. It was also able to handle unseen words, achieving an accuracy of approximately 82.04%, which was more than twice that of the second-best performing HMM tagger. Regarding prediction speed, the unigram tagger clearly leads, being multiple times faster than the other taggers in the comparison. An interesting finding we find out was the relatively high prediction time of the HMM tagger compared to the other taggers. The results also show that the unigram and Brill taggers were unable to handle unknown words with default settings. In the case of the Brill tagger, part of the blame may lie with the incorrectly selected template set "fntbl37", which may not have contained the patterns necessary for tagging unknown words.

The study had several limitations. We only tested a portion of the taggers available in the NLTK library. We used only default settings and parameter values directly available in the NLTK library for each tagger. This could have caused, for example, the Brill tagger to be unable to handle unknown words. Another limitation was the used corpus. In the study, we only used a portion of the Penn Treebank corpus available through the NLTK library. We also used only a single tagset, specifically the Penn Treebank tagset.

In future studies, it would be appropriate to also look at taggers available within the NLTK library. For taggers participating in the comparison, we could use different settings and parameter values estimated by a more sophisticated method. The performance of individual taggers should also be tested on different corpora and tagsets to obtain more objective results.

If you do not want to deal with tuning and finding the optimal parameters of a part-of-speech tagger, we recommend using the implementation of the CRF part-of-speech tagger available in the NLTK library. This model provides relatively fast and accurate results considering the amount of work and time required for its creation.

Acknowledgment

This publication was realized with support of Operational Program Integrated Infrastructure 2014 - 2020 of the project: Intelligent operating and processing systems for UAVs, code ITMS 313011V422, co-financed by the European Regional Development Fund.

It was partially supported by the Erasmus+ project: Project number: 022-1-SK01-KA220-HED-000089149, Project title: Including EVERyone in GREEN Data Analysis (EVERGREEN).

References

[1] H. H. Thorp, "Chatgpt is fun, but not an author," pp. 313–313, 2023.
[2] E. A. van Dis, J. Bollen, W. Zuidema, R. van Rooij, and C. L. Bockting, "Chatgpt: five priorities for research," *Nature*, vol. 614, no. 7947, pp. 224–226, 2023.
[3] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du *et al.*, "Lamda: Language models for dialog applications," *arXiv preprint arXiv:2201.08239*, 2022.
[4] T. Brants, "Tnt - A statistical part-of-speech tagger," *CoRR*, vol. cs.CL/0003055, 2000. [Online]. Available: https://arxiv.org/abs/cs/0003055
[5] E. Brill, "A simple rule-based part of speech tagger," Pennsylvania Univ Philadelphia Dept of Computer and Information Science, Tech. Rep., 1992.
[6] ——, "Some advances in transformation-based part of speech tagging," *arXiv preprint cmp-lg/9406010*, 1994.

[7] P. Clifford, "Markov random fields in statistics," *Disorder in physical systems: A volume in honour of John M. Hammersley*, pp. 19–32, 1990.
[8] D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun, "A practical part-of-speech tagger," in *Third conference on applied natural language processing*, 1992, pp. 133–140.
[9] S. R. Eddy, "Hidden markov models," *Current opinion in structural biology*, vol. 6, no. 3, pp. 361–365, 1996.
[10] S. Fine, Y. Singer, and N. Tishby, "The hierarchical hidden markov model: Analysis and applications," *Machine learning*, vol. 32, pp. 41–62, 1998.
[11] J. Kupiec, "Robust part-of-speech tagging using a hidden markov model," *Computer speech & language*, vol. 6, no. 3, pp. 225–242, 1992.
[12] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.
[13] B. Pham, "Parts of speech tagging: Rule-based," 2020.
[14] L. Rabiner and B. Juang, "An introduction to hidden markov models," *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, 1986.
[15] S. Acedański, "A morphosyntactic brill tagger for inflectional languages," in *Advances in Natural Language Processing*, H. Loftsson, E. Rögnvaldsson, and S. Helgadóttir, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 3–14.
[16] A. Allauzen and H. Bonneau-Maynard, "Training and evaluation of pos taggers on the french multitag corpus." in *LREC*, 2008.
[17] S. Besharati, H. Veisi, A. Darzi, and S. H. H. Saravani, "A hybrid statistical and deep learning based technique for persian part of speech tagging," *Iran Journal of Computer Science*, vol. 4, pp. 35–43, 2021.
[18] W. Demilie, "Analysis of implemented part of speech tagger approaches: the case of ethiopian languages," *Indian J Sci Technol*, vol. 13, no. 48, pp. 4661–71, 2020.
[19] E. Giesbrecht and S. Evert, "Is part-of-speech tagging a solved task? an evaluation of pos taggers for the german web as corpus," in *Proceedings of the fifth Web as Corpus workshop*. Citeseer, 2009, pp. 27–35.
[20] F. M. Hasan, N. UzZaman, and M. Khan, "Comparison of different pos tagging techniques (n-gram, hmm and brill's tagger) for bangla," in *Advances and innovations in systems, computing sciences and software engineering*. Springer, 2007, pp. 121–126.
[21] T. Horsmann, N. Erbs, and T. Zesch, "Fast or accurate?-a comparative evaluation of pos tagging models." in *GSCL*, 2015, pp. 22–30.
[22] S. Sayami and S. Shakya, "Nepali pos tagging using deep learning approaches," *NU. International Journal of Science*, vol. 17, no. 2, pp. 69–84, 2020.
[23] D. Jurafsky and J. H. Martin, "Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition (3rd draft ed.), 2023," 2022.
[24] H. Schmid, "Part-of-speech tagging with neural networks," *CoRR*, vol. abs/cmp-lg/9410018, 1994. [Online]. Available: http://arxiv.org/abs/cmp-lg/9410018
[25] A. Voutilainen, "Part-of-speech tagging," *The Oxford handbook of computational linguistics*, pp. 219–232, 2003.
[26] X. Huang, A. Acero, H.-W. Hon, and R. Reddy, *Spoken language processing: A guide to theory, algorithm, and system development*. Prentice hall PTR, 2001.
[27] H. M. Wallach, "Conditional random fields: An introduction," *Technical Reports (CIS)*, p. 22, 2004.
[28] C. Sutton, A. McCallum *et al.*, "An introduction to conditional random fields," *Foundations and Trends® in Machine Learning*, vol. 4, no. 4, pp. 267–373, 2012.
[29] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," 1993.
[30] C. D. Manning, "Part-of-speech tagging from 97% to 100%: is it time for some linguistics?" in *Computational Linguistics and Intelligent Text Processing: 12th International Conference, CICLing 2011, Tokyo, Japan, February 20-26, 2011. Proceedings, Part I 12*. Springer, 2011, pp. 171–189.