DOI: 10.2478/ijasitels-2020-0005

Part of Speech Tagging Using Hidden Markov Models

Adrian BĂRBULESCU[1], Daniel I. MORARIU[1]

[1]Computer Science and Electrical and Electronics Engineering Department, Faculty of Engineering, "Lucian Blaga" University of Sibiu, Romania

{adrian.barbulescu, daniel.morariu}@ulbsibiu.ro

Abstract

In this paper, we present a wide range of models based on less adaptive and adaptive approaches for a PoS tagging system. These parameters for the adaptive approach are based on the n-gram of the Hidden Markov Model, evaluated for bigram and trigram, and based on three different types of decoding method, in this case forward, backward, and bidirectional. We used the Brown Corpus for the training and the testing phase. The bidirectional trigram model almost reaches state of the art accuracy but is disadvantaged by the decoding speed time while the backward trigram reaches almost the same results with a way better decoding speed time. By these results, we can conclude that the decoding procedure it's way better when it evaluates the sentence from the last word to the first word and although the backward trigram model is very good, we still recommend the bidirectional trigram model when we want good precision on real data.

Keywords: Part of Speech, Hidden Markov Model, rule-based tagger, word structure analysis

1 Introduction.

Part of speech tagging is an important step in the domain of natural language processing, nowadays many companies use speech and language processing algorithms to develop different applications for their clients. Some examples of those applications used in the real world are: intelligent chatbots, various virtual assistance technologies such as Alexa or Siri, social network websites, most well-known search engines such as Google, Bing, DuckDuckGo, many smartphone mobiles, etc.

In this paper, we present an automated system that analyzes an English text and tries to correctly identify the parts of speech for each word using machine learning algorithms. The system uses a benchmark text corpus for the learning and for the evaluation phase, which is presented in section 2 and continues in section 3 with the necessary steps to prepare and preprocess the input data for the learning algorithm. In section 4 we present the theoretical aspects for the learning algorithms and how it was modified and adapted to work in the current context. In section 5, we present methods used to evaluate the model and the obtained results are presented in section 6. The conclusions and possible further developments are presented in section 7.

2 The dataset

The used dataset is a collection of texts documents that were created and annotated specifically to have the possibility to evaluate the quality of the supervised learning algorithms used for the tagging process. For this paper, we have used the *Brown Corpus*, a collection of sentences and phrases written in English language, collected, and organized by Winthrop Nelson Francis & Henry Kucera from the language department at Brown University [1]. The corpus contains over 1 million words in total and exactly 500 documents. The documents from the corpus are divided in 2 main categories: informative prose containing 9 subcategories with a total of 374 documents and imaginative prose containing 6 subcategories with a total of 126 documents.

The dataset was divided using both the 70-30 approach and *k*-fold cross-validation approach. In 70-30 approach the documents from each subcategory were divided randomly: 70% for the training set and the rest (30%) is for the testing set. In the other approach *k*-fold cross-validation, divides each category into several distinct groups called folds and at each turn, a single fold will form the testing set and the other folds will form the training set.

3 Preprocessing

The dataset must be prepared in order to be used by the learning algorithm, so a series of actions are being applied and the text from the dataset is converted into a vector representation (bag-of-word type). This representation also contains annotated text with the appropriate part of speech (PoS) for each word. All the words in the dataset and the corresponding PoS are extracted based on separation rules. For Brown Corpus, space, tab and newline are very good separation elements.

This dataset contains many type of PoS tags (approx. 100 tags) and the analysis made by the learning algorithm would take a long time if the algorithm verifies each candidate tag at every step (the prediction would be slow and the results may suffer). For this reason, we have grouped all the existing part of speech tags into 10 basic categories, those being the basic speaking parts in English language. In table 1 in the first column there are listed the names for the basic categories and in the second column there are presented 2 or 3 examples of the original tags that were included in that category.

Before extracting the relevant words (also called tokens) from the dataset, first we remove the special characters (or stop characters), those are characters like round brackets '()', square brackets '[]' and braces '{}' which are not important for the tagging process. Another process is eliminating tokens that contain only numbers, if the token contains numbers and letters, we will remove the numbers and keep the token as a word only if it will pass a certain remaining letters threshold.

In the preprocessing step in the training phase, we will also keep a list of words that start with a capital letter (capitalized words) and separately another list in which all words are converted to lowercase. The words in the test set will go through the same filter except for the last step, words that start with a capital letter will not be converted to lowercase.
In the test set, we will also remove repetitions for tokens that have a "End of sentence" tag, this is done because the algorithm doesn't evaluate these tags and repetitions like "?!?!" may issue errors later in the decoding phase.

Base part of speech	Brown Corpus tags	No. of words	Percentage
Noun (NN)	nn, nns, nps$, …	273608	23.56 %
Verb (VB)	vb, bem, hvd, …	176081	15.16 %
Article/Determiner (AT/DT)	at, ap, dt, …	142123	12.24 %
Preposition (PP)	in, to	137735	11.86 %
Others (OT)	cd, nil, *, …	108766	9.37 %
Adjective (JJ)	Jj, jjs, jjt, …	72125	6.21 %
Pronoun (PN)	pn, pp$, wps, …	71421	6.15 %
End of sentence (<s>)	"."	61254	5.28 %
Conjunction (CC)	cc, cs	60551	5.21 %
Adverb	rb, rp, qlp, …	57528	4.95 %

Table 1: Base part of speech and the frequency of occurrence for each one of them

4 Part of Speech - tagging

We used two approaches for the automatic part of speech tagging process. The first approach is the less adaptive one, those ones may either use the most frequent word tag as the model or using a default-tag to return all the tags or a combination between these two. The second approach is based on the Hidden Markov Models, these try to predict the PoS based on the transitioning probability from one hidden state to another state and based on the emission probability which returns the association probability of a tag with a certain word.

4.1 Less adaptive approaches

One simple approach to tag a text corpus is by using a default tagger, this one returns only a specific tag for every word in the text. Usually, the most used tag for this model is the "noun" tag, which is almost a quarter percent of Brown Corpus. Another less adaptive approach is using the most frequent class baseline model. This uses a training set to form a dictionary with tag frequencies for each word, when choosing the tag for a certain word the model will search the dictionary for that word and it will pick the tag with the highest frequency, if the word doesn't have a dictionary (word not existing in the training set) then the model will return the "not found" tag. Instead of returning the "*not found*" tag, we can combine this model with the *default-tagger = noun*, this will return the "noun" tag when an unknown word is found in the test set. This process of combining models will be able to obtain a decent accuracy for unknown words and it will also increase the overall accuracy of the most frequent class baseline model. We called less adaptive approaches (not un-adaptive approaches) because the parameters for those methods depend on the training dataset, are not general valid parameters for all contexts (only "Default tag" can be considered general valid).

4.2 Adaptive approaches

4.2.1 Hidden Markov Model

4.2.1.1 n-Gram model

The Hidden Markov Model (HMM) can link the connection between observable states (in our case these are the words in the text) and hidden states (these are part of speech tags). A Hidden Markov Model has 2 important components, a matrix *A* that contains the transition probabilities and a matrix *B* that contains the emission probabilities [2], [3].
The *A* matrix contains the probability that a tag will appear after another tag appeared at the previous step. For example, knowing that the article "The" appeared on the previous step, it will be more likely that at the current step, the selected tag will be a noun "The car…". The probabilities are evaluated based on the test set and are calculated by counting every tag sequence that appears in the training set.
To be able to form a matrix with the transition probabilities for a "*n*-gram", the Markov model uses a training set from which these occurrence frequencies are extracted. To collect the occurrence frequencies, each sequence in the training set will be added or incremented in the transition matrix *A*.
After creating the Hidden Markov Model in the training phase, we only calculate the probabilities for unigram (1-gram), bigram (2-gram) and trigram (3-gram) [4] from the testing set using the following formulas.
Unigram:

$$P(t_i) = \frac{c(t_i)}{N} \quad (1)$$

where $c(t_i)$ represents the occurrence frequency for the tag t_i in the training set and N is the total numbers of tokens in the training set.
Bigram:

$$P(t_i|t_{i-1}) = \frac{c(t_{i-1},t_i)}{c(t_{i-1})} \quad (2)$$

where $c(t_{i-1}, t_i)$ represents the occurrence frequency for the tag t_{i-1} followed by the tag t_i in the training set and $c(t_{i-1})$ represents the occurrence frequency for the previous tag in the training set.
Trigram:

$$P(t_i|t_{i-1}, t_{i-2}) = \frac{c(t_{i-2}, t_{i-1}, t_i)}{c(t_{i-2}, t_{i-1})} \quad (3)$$

where $c(t_{i-2}, t_{i-1}, t_i)$ represents the occurrence frequency for the tag t_{i-2} followed by the tag t_{i-1} which is also followed by the tag t_i in the training set. $c(t_{i-2}, t_{i-1})$ represents the previous bigram frequency.
The *B* matrix, with the emission probabilities, represents the probability that a certain tag is associated with a certain word in the training set. The formula, used by the system, to describe the maximum probability estimate is as follows:

$$P(w_i|t_i) = \frac{c(t_i, w_i)}{c(t_i)} \quad (4)$$

where $c(t_i, w_i)$ represents the occurrence frequency for the tag t_i associated with the word w_i in the training set and $c(t_i)$ represents the occurrence frequency for the tag t_i in the training set.

4.2.1.2 Smoothing techniques

These techniques are used when certain sequences of probabilities are missing from the A matrix and the higher the rank of the selected *n*-gram is, the chance that some sequences of probabilities may be missing, is higher. Various data smoothing techniques are used in the literature to solve this problem, so that when a *n*-gram sequence is not found, its value is estimated by a smoothing function. In this article we used 2 estimated methods for the missing probabilities sequences.

One method of estimation is via linear interpolation [5]. This method involves calculating a new probability composed of the sum of the transition probabilities (unigram, bigram, trigram) multiplied by a predetermined weight:

$$P_{LI}(t_3|t_1, t_2) = \lambda_1 P(t_3) + \lambda_2 P(t_3|t_2) + \lambda_3 P(t_3|t_1, t_2) \quad (5)$$

The values of the coefficients $\lambda_1, \lambda_2, \lambda_3$ are estimated by the linear interpolation function [2], [5] and the probabilities are computed using formulas (2), (3) and (4).
Another estimation method used is the additive smoothing (α-estimate) [6]. This involves adding some constant values to the numerator and denominator in the probability function (usually correlated with the length of the dataset).

$$\theta_i = \frac{x_i + \alpha}{N + \alpha d} \quad (6)$$

Where $\frac{x_i}{N}$ represents the non-smoothed probability, α is the smoothing constant and d is the size of the data ($i = \overline{1, d}$). For $\alpha = 0$, then the above formula (6) does not use any smoothing and for $\alpha = 1$, then the new formula will be called *Laplace's Rule of Succession* or Laplace's smoothing technique.

4.2.1.3 Unknown words model

The Hidden Markov Model for unigram, bigram and trigram, that uses only this model on the tagging system, can get a decent accuracy but it can not reach a very good accuracy. These occur because there are many words that are not in the training set and the system presented so far does not predict these words at all. If the tagging system would be used as an application working with real data where the number of existing words is much higher than the words learned, the performance would decrease considerably. Because of this, we also expanded our system to be able to predict these unknown words (words that are not in the training set). There are several ways to tag unfamiliar words such as: using a rule-based system, unsupervised learning algorithms, word structure analysis algorithms, etc.

In this paper we have developed a system that uses two methods to tag unknown words. One is based on the analysis of the word structure and the other one uses manually added rules that were created based an analysis of the training dataset. The final function will combine these 2 methods and it will return a probability of associating the unknown word with a specific tag.
The method that is based on the analysis of the word structure tries to identify suffixes / prefixes that can appear in the specified word. In order to be able to choose the best suffixes & prefixes, those were not deduced and estimated from the training set (high computational time and mediocre results), but were chosen as the most representative, using a list of suffixes & prefixes provided by [7]. Following this analysis, the list of prefixes and suffixes chosen for this system are:

List of prefixes: `"inter", "intra", "mis", "mid", "mini", "dis", "di", "re", "anti", "in", "en", "em", "auto", "il", "im", "ir", "ig", "non", "ob", "op", "octo", "oc", "pre", "pro", "under", "epi", "off", "on", "circum", "multi", "bio", "bi", "mono", "demo", "de", "super", "supra", "cyber", "fore", "for", "para", "extra", "extro", "ex", "hyper", "hypo", "hy", "sub","com", "counter", "con", "co", "semi", "vice", "poly", "trans", "out", "step", "ben", "with", "an", "el", "ep", "geo", "iso", "meta", "ab", "ad", "ac", "as", "ante", "pan", "ped", "peri", "socio", "sur", "syn", "sy", "tri", "uni", "un", "eu", "ecto", "mal", "macro", "micro", "sus", "ultra", "omni", "prim", "sept", "se", "nano", "tera", "giga", "kilo", "cent", "penta", "tech".`

List of suffixes: `"able", "ible", "ble", "ade", "cian", "ance", "ite", "genic", "phile", "ian", "ery", "ory", "ary", "ate", "man", "an", "ency", "eon", "ex", "ix","acy", "escent", "tial", "cial", "al", "ee", "en","ence", "ancy", "eer", "ier", "er", "or", "ar", "ium", "ous", "est", "ment", "ese", "ness", "ess", "ship", "ed", "ant", "ow", "land", "ure", "ity", "esis", "osis", "et", "ette", "ful", "ify", "ine", "sion", "fication", "tion", "ion", "ish", "ism", "ist", "ty", "ly", "em", "fic", "olve", "ope","ent", "ise", "ling", "ing", "ive", "ic", "ways", "in", "ology", "hood", "logy", "ice", "oid", "id", "ide", "age", "worthy", "ae", "es".`

In order for the model to be able to use these affixes, it is first necessary to identify with which tags the words that have these affixes are associated in the training set and to calculate the probability of association with the tag encountered. If the affix is not found in the training set, then the additive smoothing will take place. To calculate this, we use the following formula:

$$P_{sp}(x_i|t_i) = \frac{c(t_i,x_i) + \alpha}{\sum_{k=1}^{T_n^{x_i}} k + \alpha d} \quad (7)$$

where $c(t_i, x_i)$ represents the occurrence frequency for the tag t_i associated with the prefix/suffix x_i, $\sum_{k=1}^{T_n^{x_i}} k$ represents the sum of the tags frequencies associated with the prefix/suffix x_i.
The second component of the tag identification function for unknown words is the manually based rule component. The following rules were used in this article: "words that start with a capital letter" are more likely to be nouns, "words that contain an apostrophe and end with the *s* character" are very likely to be nouns, "words that contain hyphen ('-') or slash ('/')" have a higher probability to be compound words of type OT (others) or JJ (adjective), "words that contain an apostrophe and end with *the t character*" are very likely to be verbs and "words that contain an apostrophe and end with the *'ve'* or *'ll'* characters sequence" are very likely to be pronouns.
In order to be able to combine these 2 methods presented above, the probability of the unknown word with the current tag is calculated according to the suffixes and prefixes associated with it (note P_{sp}) and the rule-based probability is calculated according to the passed conditions for the weights of rules (note P_r). These are combined in the following final probability:

$$P(w_k|t_i) = P_{sp}(x_i|t_i) + P_r(w_k|t_i) \quad (8)$$

Following this addition, the result may exceed the probability interval (0,1]. For this, a threshold function will be executed that will round the value to 1.0. Exceeding the limit only suggests that there is a probability of 100% (maximum confidence) that the current tag being tested is also the correct one, usually this value is obtained for the noun tag which in most cases is also the correct tag.

4.2.2 Decoder

In the previous section we presented how models are created for known words (emission and transition probabilities) and for unknown words (rule-based and word structure analysis). Next, we present the decoder part of the system, without it the tagger cannot determine the hidden variables sequence (the tag sequence) associated with the observations sequence (the words of a sentence) [2].
The algorithm used to decode an HMM, based on the dynamic programming, is the Viterbi algorithm [2]. The Viterbi algorithm can process the states of the trellis from left to right or opposite. The general formula to calculate any node at each step in the trellis is as follows (at time step t, where t ≠ 0):

$$v_t(j) = max_{i=1}^{N} v_{t-1}(i) P_{LI}(t_3|t_1,t_2) P(w_i|t_i) \quad (9)$$

where $v_t(j)$ represents the current Viterbi node processed for tag j and v_{t-1} represents the Viterbi node processed at previous time step.
To achieve better performances for the HMM, we came up with three decoding methods and their results are presented in this article. We will call these methods *forward* (goes from the first word of the sentence to the end of the sentence, then a backtrack is made to return the final tags), *backward* (goes

from the end of the sentence to the beginning of it, then the backtrack is made) and *bidirectional* (will execute both forward and backward method and then will backtrack either by the forward or backward methods, depending on the maximum value of the final node).

5 Model evaluation

The dataset used for this paper is a pre-labeled corpus, in this case the test set is also pre-labeled with the correct tags. These tags are not used in the prediction process, they are only used to evaluate the performance of the learning algorithms (evaluating models phase). To evaluate the model, we used 2 approaches, the first approach calculates a simple prediction accuracy, and the second approach calculates the metrics based on the confusion matrix. The first assessment involves calculating the percentage of correctly predicted tags (by comparing the predictive tag with the actual one from the test set) from the total number of predicted tags. We further note this evaluation metric as "Accuracy_1". For this approach, we could also calculate the accuracy for the known words and separately for the unknown words to get a better understanding of each individual model.
The second approach involves calculating the confusion matrix for the entire testing set and extracting the evaluation parameters from that matrix as: Accuracy_2, Precision, Recall, Specificity and F1-score [8].

6 Obtained results

Table 2 presents the results obtained by the bidirectional trigram model for each part of speech separately as well as an average for each evaluated metric (*TOTAL* line). We chose this model because it obtained the best results and integrated both forward and backward methods. Those results were obtained using the 70-30 approach to split the dataset and the confusion matrix was computed for each tag in the testing phase.

TAG	ACCURACY_2	PRECISION	RECALL	SPECIFICITY	F1-SCORE
NN	97.86%	95.70%	95.75%	98.57%	95.72%
OT	99.85%	99.19%	99.19%	99.92%	99.19%
CC	99.37%	91.88%	97.50%	99.48%	94.61%
JJ	98.83%	89.72%	93.16%	99.23%	91.41%
PP	99.41%	97.74%	97.59%	99.67%	97.67%
AT/DT	99.37%	98.19%	96.94%	99.73%	97.56%
VB	98.64%	96.67%	94.77%	99.38%	95.71%
PN	99.85%	98.96%	98.72%	99.93%	98.84%
RB	98.87%	90.19%	88.47%	99.46%	89.32%
TOTAL	99.12%	95.36%	95.79%	99.49%	95.56%

Table 2: Results obtained by the bidirectional trigram model

The most difficult tags to predict, which have the lowest score, were the adverb and the adjective. These 2 tags are the most context dependent tags and the context dependent tags do not have a high probability for only a tag and are, therefore, very difficult to predict in some contexts. The predicted tag with the best score is the Others tag, this tag contains interjections, numbers (cardinal numbers), compound words, etc. which most of the time have a unique morphological form and are not context dependent (for example, the word "one" will always have the cardinal number tag). These very good results can be explained by the fact that the training set and the test set are from the same dataset and have the same distribution. Even if there are other documents used for testing than those used for training, they come from the same sources (same authors) that tend to use the same words in the same contexts and automatically with the same part of speech.

For the forward bigram model, the average training time of the model is approximatively 1,41 minutes and the average decoding time for all sequences is 1,44 minutes. For the bidirectional trigram model, the average training time of the model is 1,38 minutes (similar to the average training time for the forward bigram) and the average decoding time for all sequences is 3,56 minutes (twice as long as the forward bigram model). The training time between these models does not differ too much because the training function uses parallel threads to use the most of the processor's capabilities. The decoding time is much longer for the bidirectional trigram model because it has to calculate the trigram transition probability (also the transition probabilities for bigram and unigram) and evaluate the model both forward and backward, after which it needs to decode the best sequence for each sentence. Processing times were estimated on a desktop system with Windows 10, a Quad-Core processor with a frequency of 3.60 GHz and 16 GB Ram.

Model parameters options	**Unknown words percentage from test dataset (%)**	**Unknown words accuracy (%)**	**Known words accuracy (%)**	**Accuracy_1 (%)**
Default-tag: Noun	-	-	-	24.96
Most frequent class baseline	13.76	0.00	95.85	82.66
Most frequent class baseline + Default-tag: Noun	13.76	52.35	95.85	89.87
Forward bigram	3.84	77.37	96.45	95.72
Backward bigram	3.83	82.20	96.51	95.96
Bidirectional bigram	3.83	82.23	96.50	95.95
Forward trigram	3.84	78.28	96.60	95.90
Backward trigram	3.80	81.46	96.63	96.05
Bidirectional trigram	3.83	81.54	96.63	96.05

Table 3: Comparatively results obtained by all approaches (adaptive and less-adaptive)

In table 3 we compare the results between the less-adaptive approach and the adaptive approach. The results are presented in terms of simple accuracy (Accuracy_1) for the known and unknown words but also for all the words all together. The known words are words in the test set that also appear in the training set and the unknown words are words that do not appear in the training set. The accuracy on unknown words verifies the tagging system performance that must adapt to unknown situations when it encounters new cases. Those results were obtained using the *k*-fold cross validation approach to split the dataset (in this case $k = 4$).

For less adaptive models (specifically most frequent class baseline model from table 3), we do not keep a separated list for the words that start with a capital letter. The default-tagger doesn't have any concept of known words or unknown words because it only predicts one tag, the most frequent class baseline and this model compound together with the default-tagger keep a list of unmodified words without converting the capitalized words to lowercase in the testing set. Without this conversion and without keeping a separated list with capitalized words only, the percent of unknown words for less adaptive models is way higher than the percent of unknown words for the adaptive models. As we can see, less adaptive approach can get good results (only on familiar words) but on a completely new dataset, the performance could decrease quite a bit.

From these experiments we noticed that a backward trigram model is almost as good as the bidirectional trigram. Analyzing in more detail, we noticed that in the case of bidirectional trigram, in 85.52% of cases the backward branch is chosen when using the bidirectional method. Figure 1 shows the percentage in which a bidirectional trigram model chooses the backward method as the optimal back trace path, being compared to the forward method. In 0.03% of cases, both models return the same result because the final nodes (final states) for these two methods have the same value.
Another interesting observation is that the bidirectional bigram model has the best accuracy for unknown words (82.23%) but does not have as good an accuracy as the bidirectional trigram for known words (96.63%). The forward bigram has a poor performance (comparing it with the others that use a Markov model) and the forward trigram has a better performance compared to the forward bigram. The problem of tagging of PoS is complicated because analyzing the dataset, 67% of tokens are ambiguous [2] (have more than one possible part of speech in different contexts), if we use a less adaptive approach like the most frequent class baseline + default tag: noun, we can almost achieve a 89.87% accuracy of correctly predicted tags. In the preprocessing section we showed that the noun tag represents approximately 23.56% of the dataset, eliminating the end of sentence tag and using only the first model with the default-tag = noun, an accuracy of ~ 25% need to be obtained, which implies that almost a quarter of the test set was "predicted" correctly. Because of the large dataset, the difference between these models' results are small, same applies for the results presented in table 2. The problem becomes

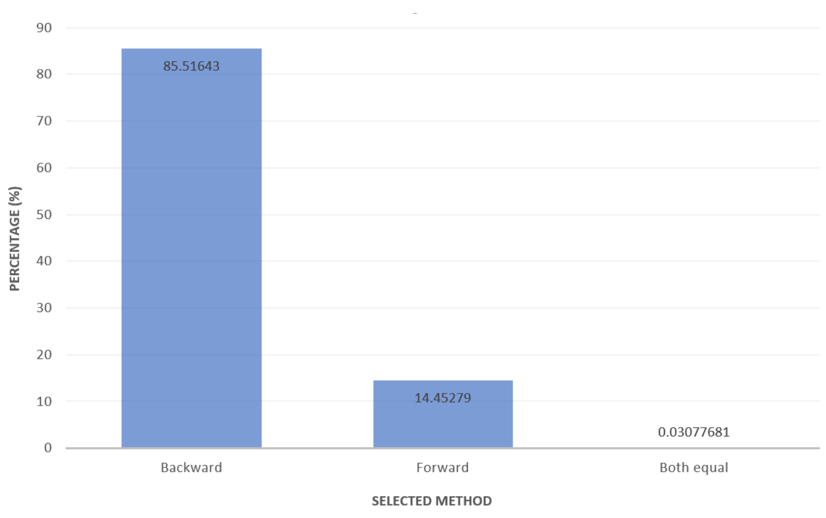

Figure 1:Percentage for each final node value chosen by the bidirectional method

complicated when the testing set have a different distribution than the training set (for example sentences from real world). In table 3, in the first column "Unknown words percentage from test dataset" the percent differs (is between 3.80% to 3.84%) because we use *k*-fold cross-validation for each test and the documents were randomly grouped in each fold.

7 Conclusions

The purpose of this paper is to analyze the performance of the Hidden Markov Model in the PoS tagging system. For this we implemented and tested 2 different models of HMM (bigram and trigram) and 3 different methods to compute the maximum probability values for a sentence (forward, backward and bidirectional). The trigram model obtains better results combined with the bidirectional or the backward method. In the bidirectional method, in almost all cases (85.51%) the best choice was the backward method. Also, we compare the obtained results by the HMM approaches with less-adaptive approaches as "predicting the same tag all the time" or predicting the "most frequent class baseline" to see the improvement from the accuracy standpoint, if we add the learning process in the PoS tagging process. The increase was from 89.87% (when we use a combination between all less-adaptive approaches) to 96.05% (when we use the learning approach).

In conclusion, the automatic speech part tagging system presented in this paper uses well-established algorithms in the field of machine learning and natural language processing to achieve state of the art performances. These performances were achieved on a test set extracted from the same dataset that the training set also uses.

The system presented in this paper was also tested with sentences entered manually by the user and the response was evaluated only by the user. Until now the system works only for the English language and it can be extended easily to other languages.

References

[1] W. Nelson Francis and Henry Kučera at Department of Linguistics, Brown University Standard Corpus of Present-Day American English (Brown Corpus), Brown University Providence, Rhode Island, USA, korpus.uib.no/icame/manuals/BROWN/INDEX.HTM
[2] Dan Jurafsky, James H. Martin, Speech and Language Processing, third edition online version, 2019
[3] Lawrence R. Rabiner, A tutorial on HMM and selected applications in Speech Recognition, Proceedings of the IEEE, vol 77, no. 2, 1989
[4] Adam Meyers, Computational Linguistics, New York University, 2012
[5] Thorsten Brants, TnT - A statistical Part-of-speech Tagger (2000), Proceedings of the Sixth Applied Natural Language Processing Conference ANLP-2000, 2000
[6] C.D. Manning, P. Raghavan and M. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008
[7] Lois L. Earl, Part-of-Speech Implications of Affixes, Mechanical Translation and Computational Linguistics, vol. 9, no. 2, June, 1966
[8] Daniel Morariu, Radu Crețulescu, Text mining - document classification and clustering techniques, Published by Editura Albastra, 2012