Lighweight Text Classification

Sarah Marek
Institute of Databases and Information Systems
Ulm, Germany
sarah.marek@uni-ulm.de

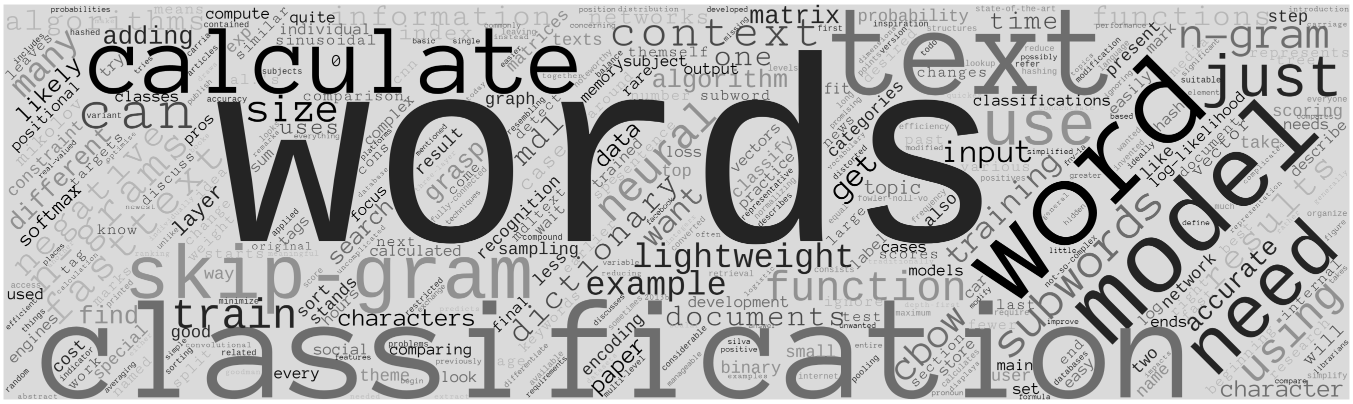

Figure 1: Tag cloud of a part of this paper

ABSTRACT

This paper discusses three different algorithms for lightweight text classification: skip-gram, fastText, and MDLText. It compares them and discusses the strengths and weaknesses to find the most suitable use case for each classification algorithm.

Lightweight text classifications are fast and uncomplicated algorithms to organize a text to one previously trained class or tag. Instead of a complex deep neural network, they use a few levels of a simple neural network.

The comparison criteria of the algorithms includes: What the algorithm does and how it differs from its predecessor if it has one. The document type and language of the dataset the algorithm is using. And lastly, the accuracy and stability of the final result concerning the listed points mentioned.

Possible results might be for example, that the accuracy of the algorithm might be better in multiple languages than in English, or the document type could influence the accuracy. Discussing the results with the different results can help us understand how the criteria influence the results.

CCS CONCEPTS

• **Information systems** → Clustering and classification.

KEYWORDS

classification, text

ACM Reference Format:
Sarah Marek. 2022. Lighweight Text Classification. In *Ulm '22: Lighweight Text Classification, February 27, 2022, Ulm, DE.* ACM, New York, NY, USA, 8 pages.

1 INTRODUCTION

Text classification was manageable in the past, with just traditionally printed media available. In the past, librarians could organize and categorize documents into categories and sort unimportant things out.

Today everyone can publish and access everything using the internet. Through that, everyone can easily exchange remarks or information through text. That means there is plenty of data from social network platforms, news articles, books, and many other places.

To extract the desired information from the unwanted there is a need to differentiate between the two. For that, the algorithms need to label texts with subject tags. These are called classes in this paper. For detecting a keyword, it is essential to train a neural network that promises good results.

Skip-gram, fastText, and MDLText will be discussed in this paper and the results of the datasets that come out. The main evaluation criteria are minimum, maximum, average accuracy, and the stability of the results, which heavily depend on the language and the text document type. After analyzing these algorithms even more in section 5, some sensible areas of applications are discussed.

2 RELATED WORK

Non-lightweight algorithms to classify texts need large datasets and considerable time to train and test them. Sometimes there is no time to train a neural network for hours, or a user does not want to wait too long for the answer. That is why uncomplicated quicker text classifications are needed.

They use a simplified model of the complicated multi-level neural networks, and with the right features, they archive state-of-the-art accuracy. We look into three algorithms: skip-gram, fastText, and MDLText.

For skip-gram, the algorithm from Mikolov et al. [2] will be explained and expanded by the research by another work from Mikolov et al. [9]. As datasets, the ones mentioned in the papers, Svoboda et al. [12], Körper et al. [7] and Berardi et al. [1] are used.

In fastText, Mikolov et al. [4] use a further development of Cbow. For Cbow, the referred paper from Mikolov et al. from 2013 [8] will be used. The summary of the differences between skip-gram and Cbow written by Ria Kulshrestha on the platform "Towards Data Science" will be used to explain it further. [5] Also, the softMax function from Goodman is referred to in this paper. [3]

MDLText was researched by Silva et al. [11] with the MDL principle from Ken Lang. [6] The datasets are not well explained in the paper of Silva et al., because of that the cited work of Zhang et al. [13] and Rossi et al. [10] is used.

3 ALGORITHMS

As we present the different classification algorithms in this section, we also discuss their development.

We will begin with skip-gram, an algorithm that uses character n-grams to get subwords out of the text. With these subwords, we can grasp the most likely theme of the text. Mikolov et al. [2] researched this algorithm that we present. His older publication [9] served as the basis of this.

FastText [4] is an algorithm invented by Facebook AI Research to get fast and accurate results using context word n-grams. It draws inspiration from the Cbow model. The Cbow model is built like a reversed skip-gram algorithm. Through that, the subject can be calculated through the context of the words.

MDLText tries to balance between a not-so-complex model and promising results by comparing the frequency of the words with the trained documents. We present Silva et al.'s research on this topic. [11]

3.1 skip-gram

The skip-gram model we describe here is a modified version of the skip-gram model by Mikolov et al..[9] The modification here for skip-gram is negative sampling with subword recognition.

3.1.1 original model. The model from Mikolov et al. uses a vocabulary size W to train a vector w with elements $\{1..., W\}$. The idea is that, in the end, every dictionary word's probability is summed up in a vector. The dictionary represents every class that is trained in the training stage.

To calculate the maximum of the log-likelihood 1 of a word w_t the context words $\{w_{t-c}, ..., w_{t-1}, w_{t+1}, ..., w_{t+c}\}$ of this word will be used. The quantity of the training context that are before w_t is c, the quantity of words after w_t is also c. With that, we can calculate likely context words of w_t:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{1}$$

$p(w_c|w_t)$ 2 is using the softmax function, where w_t is the input word and w_c is the output word.

$$p(w_c|w_t) = \frac{exp\ (s(w_t, w_c))}{\sum_{j=1}^{W} exp\ (s(w_t, j))} \tag{2}$$

The scoring function s 3 is a scalar function with the input and output vectors u_{w_t} and v_{w_c} that are equal to words w_t and w_c.

$$s(a, b) = v_a \cdot u_b'^{T} \tag{3}$$

Mikolov et al. describe if we use more context words, we get more accurate results, but we need more training time because of the size of the training dataset.

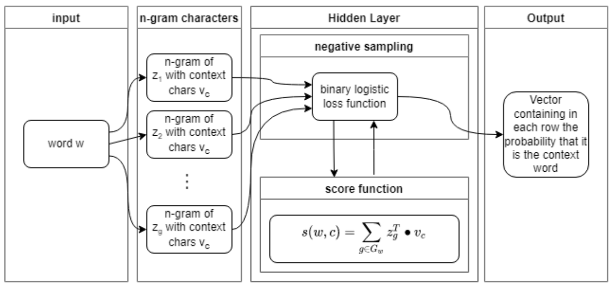
input
n-gram characters
Hidden Layer
Output
word w
n-gram of z_1 with context chars v_c
n-gram of z_2 with context chars v_c
n-gram of z_g with context chars v_c
negative sampling
binary logistic loss function
score function
$s(w, c) = \sum_{g\in G_w} z_g^T \bullet v_c$
Vector containing in each row the probability that it is the context word

Figure 2: Simplified representation of the skip-gram algorithm with negative sampling, with the output parameters being context words.

3.1.2 Model with negative sampling. The original model predicts just one context word w_c and needs too much time to figure out the results. We want to grasp the class of the text and not just a context word of the sentence. That is why we need to modify the algorithm, to calculate the class from 2 types of words: random negative examples $\mathcal{N}_{t,c}$ that are taken from the dictionary and context words.

For a word with index t, we need a position c from the context words that we calculate with the binary logistic loss function $\log(1+exp(-s(w_t, w_c)) + \sum_{n\in\mathcal{N}_{t,c}} \log(1+exp(s(w_t, n)))$. For the negative examples, we use $N_{t,c}$ to refer to them and simplify the exp function with $l : x \longrightarrow \log(1+e^{-x})$.

$$\sum_{t=1}^{T}\left[\sum_{c\in C_t} l(s(w_t, w_c)) + \sum_{n\in N_{t,c}} l(-s(w_t, n))\right] \tag{4}$$

This is the last step in skip-gram in figure 2.

3.1.3 Subword recognition. A word can be split into character n-grams to ignore the internal structures of the original word. N-grams are groups with a size n.

We would have too many positive cases because n-grams with sizes less than $n = 3$ have too little information. For example, words with the n-gram "he" calculate that they are similar to a pronoun.

A size greater than 6 has too many characters. We would have no to few positive cases. For example, words with the n-gram "carria" from "carriage" calculate that they are similar to no other word. So in practice, we use n-grams of the size between 3 to 6 to get good results.

To know where the n-gram starts and ends, we use the special characters < and >. The word "carriage" with an n-gram of 3 is

<ca, car, arr, rri, ria, iag, age, ge>

Noteworthy is that the subwords "car" and "age" are not quite resembling themself because of the missing special characters at the beginning and end, but they are subwords of themself. This is the first step in the figure 2.

To use n-grams in the skip-gram algorithm, we also change the original scoring function s. We have a dictionary of n-grams with the size of G. For a word w, we can name the vector for the n-grams in $w \quad G_w \subset \{1..., G\}$. For an n-gram g, there will be a vector representation z_g to calculate the most likely word the n-gram describes.

With all these changes to the algorithm, we can calculate for each n-gram the most likely word it represents. The scoring function is a function in the hidden layer of this algorithm as we see in figure 2. By adding the n-gram results together, we get the most likely representation of the entire word:

$$s(w, c) = \sum_{g \in G_w} z_g^T \cdot v_c \tag{5}$$

For this calculation, the dictionary of the n-grams needs more memory than a dictionary with just the words. Because memory is often restricted, we need to store fewer words or reduce the size of the dictionary in the memory. If we take fewer words in the n-gram dictionary, we could get distorted results because we ignore some subwords to compare. For reducing the size of the dictionary in the memory, we can use a hash function. Mikolov et al. use the FNV-1a variant of the Fowler-Noll-Vo hash function with integers from 1 to K with $K = 2.10^6$.

3.1.4 *Classification.*

By adding the probabilities of the context words and then adding the calculated results together, we get the vector with the chances of each trained class being the main class. This is shown as the last step in figure 2.

3.2 fastText

FastText uses a similar model named CBOW from Mikolov et al.[8][5].

3.2.1 *Cbow.*

Cbow is similar to skip-gram. It takes the sentence as n-grams and calculates a score with them.

In Cbow, unlike skip-gram, each word of an n-gram is converted into an input vector. The next step is to mark the main word, leaving only the context words. Each of these words is then calculated by using the hidden layer matrix. The "hidden layer matrix" is just a weight matrix trained in the training stage. The average of these vectors will then be applied to the softmax layer.

3.2.2 *Further development.*

Like Cbow, fastText marks the main word with a label y_n from the n-gram x_n. The variable n stands here for n-th document of N documents. The weight matrices are

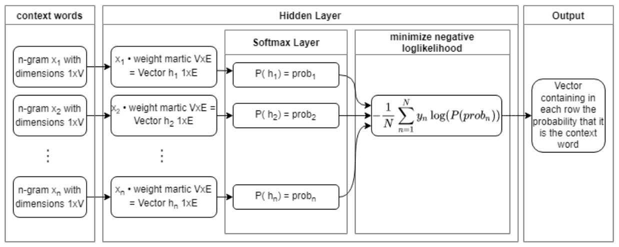
context words
Hidden Layer
Output
Softmax Layer
minimize negative loglikelihood
n-gram x_1 with dimensions 1xV
n-gram x_2 with dimensions 1xV
n-gram x_n with dimensions 1xV
x_1 • weight martic VxE = Vector h_1 1xE
x_2 • weight martic VxE = Vector h_2 1xE
x_n • weight martic VxE = Vector h_n 1xE
P(h_1) = $prob_1$
P(h_2) = $prob_2$
P(h_n) = $prob_n$
$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(P(prob_n))$
Vector containing in each row the probability that it is the context word

Figure 3: Simplified representation of the fastText algorithm, with the input parameters being context words.

A and B, where A is a lookup matrix over the trained words. Figure 3 displays the matrices just as a weight matrix. To minimize the negative log-likelihood over the classes we use:

$$-\frac{1}{N} \cdot \sum_{n=1}^{N} y_n \log\left(P(B \cdot A \cdot x_n)\right) \tag{6}$$

In formula 6 P refers to the hierarchical softmax function from Goodman [3] to calculate the distribution probability over the trained words, with:

$$P(w_1 ... w_n) = \prod_{i=1}^{n} P(w_i | w_1 ... w_{i-1}) \tag{7}$$

$$P(w | w_1 ... w_{i-1}) = \frac{exp\left(\sum_j \lambda_j f_j(w, w_1 ... w_{i-1})\right)}{Z_\lambda(w_1 ... w_{i-1})} \tag{8}$$

These two steps of calculating the probability and minimizing the negative log-likelihood are the last two steps of skip-gram as we see in figure 2.

To optimize the training data, we need to train the real-valued constraint λ_j in the training stage. $Z_\lambda(w_1..., w_{i-1})$ is the normalizing constraint so that the sum of all probabilities is 1. The function f_j is a large set of probability indicator functions. These functions have either a result of 0 or 1. This causes us to build a tree in the classification phase, in which we just calculate the most likely paths.

3.2.3 *Classification.*

All in all, fastText can compute the subject of a word through the n-grams of the words around it. By adding the scores of all the individual context words, we can get the most likely class of the word. By adding the individual scores that we calculated in the last step, we can then grasp the class of the text.

3.3 MDLText

The MDL principle is the foundation of MDLText. MDL selects the model with the slightest description length out of two or more models. Considering the number of models, we do not want too complex models. MDL is often used to solve machine learning problems.

3.3.1 *MDL principle.*

The methods behind the term "MDL" have different approaches: Some approaches may involve data compression for all types of inductive inferences. Others find a model with the minimum description length as data. Some other papers use the term MDL in another way.

Algorithm 1: Classification stage for MDLText

```
1  FUNCTION mdl_classify(d, C, dict, n, $\hat{n}$,$\phi$, $|\hat{D}|$)
2      INPUT: d(unlabeled document), C(set of all
               possible classes), dict (list of tokens
               obtained in the training stage), n(sum of
               the weights of each token in dict for each
               class in C), $\hat{n}$(sum of the weights in n for
               each class in C), $\phi$(frequency of each token
                in dict for each class in C), and$|\hat{D}|$(
               number of documents for each class in C)
3      OUTPUT: c(d)(the class predicted for document d)
4
5    // variables that will be used later on
6      FOR EACH token $t_i$ in d DO
7          IF $t_i$ is in dict THEN
8              $F(t_i) \leftarrow token\_score(t_i, \phi_{\forall c, t_i})$
9          ELSE
10             $F(t_i) \leftarrow 0$
11         END IF
12         $\Omega \leftarrow$ preserve a portion of the description
               length
13         $K(t_i) \leftarrow \frac{1}{(1+10^{-3})-F(t_i)}$
14     $TF \leftarrow$ local term frequency of $t_i$
15     $DF_{t_i} \leftarrow$ number of training-set documents where $t_i$
           appears
16     $w(t_i, d) = \log(1 + TF(t_i, d)) \times \log(\frac{|D|+1}{DF_{t_i}+1})$
17     $\hat{w}(t_i, d) = \frac{w(t_i,d)}{||w(:,d)||_2}$
18     END FOR
19
20   //calculate $L(d|c_j)$ without penalty and $\bar{c}$
21     FOR EACH class $c_j$ in C DO
22         $L(d|c_j) \leftarrow 0$
23         FOR EACH token $t_i$ in d DO
24             IF $t_i$ is not in dict THEN
25                 $n_{c_j,t_i} \leftarrow 0$
26                 $\hat{n}_{c_j} \leftarrow 0$
27             END IF
28             $\beta(t_i|c_j) = \frac{n_{c_j,t_i} + \frac{1}{|\Omega|}}{\hat{n}_{c_j}+1}$
29             $L(t_i|c_j) = \lceil -\log_2 \beta(t_i|c_j) \rceil$
30             $L(d|c_j) = L(d|c_j) + (L(t_i, c_j) \times K(t_i))$
31             $\bar{c}_j(t_i) = \frac{n_{c_j,t_i}}{|\hat{D}c_j|}$
32         END FOR
33     END FOR
34
35   // calculate penalty to L
36     FOR EACH class $c_j$ in C DO
37         $S(d, \bar{c}_j) = \frac{\sum_{i=1}^{|d|} \hat{w}(t_i,d|c_j) \times \bar{c}_j(t_i)}{||\hat{w}(:,d)||_2 \times ||\bar{c}_j||_2}$
38         $\hat{S}(d, \bar{c}_j) = -\log_2 \left(\frac{1}{2} \times S(d, \bar{c}_j)\right)$
39         $L(d|c_j) = \hat{S}(d, c_j) \times L(d|c_j)$
40     END FOR
41
42   // get the index of the calculated class of the
         text
43     $c(d) = arg\ min_{\forall c} L(d|c_j)$
44     RETURN $c(d)$
45 END FUNCTION
```

We will be using MDL to find the minimum description length of the classes. The MDL principle uses two or more potential models $M_1, ..., M_{|M|}$. To calculate the best result, we use the description length of the model $L(M)$ and the description length of the data d encoded by the model $L(d|M)$.

$$M_{mdl} = \arg\min_{\forall M}[L(M) + L(d|M)] \tag{9}$$

3.3.2 MDLText implementation. In contrast to skip-gram and fastText, MDLText uses the whole text of a document 'd' to calculate

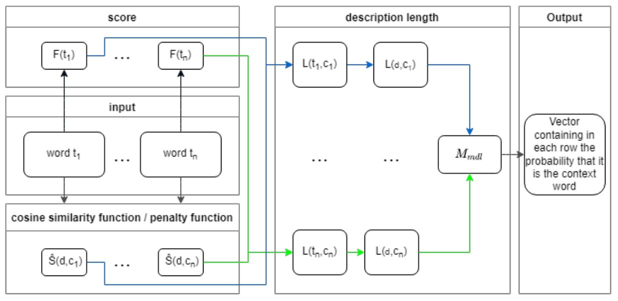

Figure 4: Simplified representation of the used MDLText algorithm

the class. The text is split into tokens 't_i' (Algorithm 1,line 6). We call a sub-method *token_score*, which calculates the score $F(t_i)$, with the help of the token and its frequency ϕ if we trained MDL to that token. If the token was not in the training stage, we set the score to 0. With the score, we can calculate the penalty function $K(t_i)$ (Algorithm 1,line 13), wich is then used in the overall penalty function $\hat{S}(d, c_j)$. The goal of this function is to highlight the difference between the different classes and help to differentiate later.

$$\hat{S}(d, c_j) = -\log_2(\frac{1}{2} \times S(d, \bar{c})) \tag{10}$$

The cosine similarity function 10 is the penalty function for the classes. This is one of the split first step in MDLText as we see in figure 4. The term penalty means that the difference between the classes is getting larger. $\bar{c}$ is here an average vector of the tokens (Algorithm 1,line 31).

$$L(d|c_j) = \hat{S}(d, c_j) \times \sum_{i=1}^{|d|} L(t_i|c_j) \times K(t_i) \tag{11}$$

The implementation of this formula 11 can be found in lines 19 and 30. It calculates the description length of all the tokens with the help of the trained classes 'c'. For $L(t_i|c_j)$, we compute the trained TF-IDF weights with our token by the function β (Algorithm 1,line 28, formula 12) with a negative log. This function is the last MDLText uses as we see in figure 4.

$$\beta(t_i|c_j) = \frac{n_{c_j,t_i} + \frac{1}{\Omega}}{\hat{n}_{c_j} + 1} \tag{12}$$

$$n_{c_j,t_i} = \sum_{\forall d} \hat{w}(t_i, d|c_j) \tag{13}$$

Here, $\hat{n}_{c_j}$ represents the total number of tokens in the training document that belong to c_j, and Ω is used to keep the description length of tokens that never appear in the training documents.

$$w(t_i, d) = log(1 + TF(t_i, d)) \times \log(\frac{|D| + 1}{DF_{t_i} + 1}) \tag{14}$$

$$\hat{w}(t_i, d) = \frac{w(t_i, d}{||w(:, d)||_2} \tag{15}$$

$w(t_i, d)$ is the TF-IDF calculation function, with a normalized function $\hat{w}(t_i, d)$. TF stands for the local term frequency of the token t_i in document d and DF_{t_i} stands for the number of documents in which t_i appears.

With all this calculation until now, we now have all the scores of the classes that we need. With the MDL principle, we can get the most suitable classification index by using:

$$c(d) = arg\ min_{\forall c} L(d|c_j) \tag{16}$$

With the index, we can look up the class of document d.

3.3.3 Scoring function. We can use every scoring function with a codomain of $[0, 1] \subset \mathcal{R}$. Silva et al.[11] uses the confidence factors to calculate $F(t_i)$:

$$F(t_i) = \frac{1}{|C| - 1} \sum_{\forall j | j \neq v} \frac{\left(\frac{(MH)^2 + PH - \frac{\lambda_1}{SH}}{(SH)^2} \right)^{\lambda_2}}{1 + \frac{\lambda_3}{SH}} \tag{17}$$

where $SH = \phi_{c_v,t_i} + \phi_{c_j,t_i}$, $PH = \phi_{c_v,t_i} \times \phi_{c_j,t_i}$ and $MH = \phi_{c_v,t_i} - \phi_{c_j,t_i}$ with v being the index of the most frequent class and ϕ_{c_j,t_i} being the number of documents in which the token t_i is in the class c_j. Algorithm 2 shows us an implementation of this function from Silva et al.[11]. As we can see in figure 4, the scoring function is the second part of the split first step of MDLText.

Algorithm 2: Scoring function

```
 FUNCTION token_score($t_i$, $\phi$)
     INPUT: C(set of all possible classes), $\phi$(
             frequency of each token in dict for each
             class in C), and $t_i$ is a token of a document
     OUTPUT: $F(t_i)$ (token score)

     $v \leftarrow$ index of the most frequent class
     // decay speed proposed by Assis et. al \cite{
             Assis2006}
     $\lambda_1 = 0.25$
     $\lambda_2 = 10$
     $\lambda_3 = 8$
     FOR EACH class $c_j$ in C do
         IF $j \neq v$ THEN
             $MH = \phi_{c_v,t_i} - \phi_{c_j,t_i}$
             $PH = \phi_{c_v,t_i} \times \phi_{c_j,t_i}$
             $SH = \phi_{c_v,t_i} + \phi_{c_j,t_i}$
             $F(t_i) = F(t_i) + \frac{\left( \frac{(MH)^2 + PH - \frac{\lambda_1}{SH}}{(SH)^2} \right)^{\lambda_2}}{1 + \frac{\lambda_3}{SH}}$
         END IF
     END FOR
     $F(t_i) = \frac{1}{|C|-1} * F(t_i)$
     RETURN F(t_i)
END FUNCTION
```

3.3.4 Classification. With MDL, we look up how often different words occur. If we have untrained expressions, we ignore those. Using this information and the cosine similarity function 10, we can calculate what class our text belongs to using the function L.

4 RESULTS

We evaluate these models with the minimum, maximum, average accuracy, and the stability of the accuracy of the results compared with the language and document types of the predefined datasets mentioned in the papers.

4.1 Datasets

For skip-gram, we use all datasets cited in Mikolov et al. [9]. For MDLText and FastText, we use just the English and multilanguage datasets from Mikolov et al.[4] and Silva et al.[11].

The datasets from MDLText are explained in detail in the paper of Rossi et al. [10] and Zhang et al. [13]. FastText uses the datasets from Mikolov et al., which are detailed explained in Zhang et al. [14]. Skipgram uses the datasets from four different papers: Mikolov et al. [8], Svoboda and Brychcin [12], Köper et al. [7] and Berardi et al. [1].

4.2 Validation of accuracy

To validate the accuracy of the output of the algorithms, we use different techniques that we will be discussing in this section.

4.2.1 skip-gram. For skip-gram, we use Spearman's rank correlation coefficient, with *d* being the difference and *n* being the number of observations:

$$p = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{18}$$

Spearman calculates the strength of association between two ranked variables. Using that, we can compare how accurate the class calculated is versus the one defined by the dataset.

4.2.2 FastText. For FastText, we use 10 hidden units five times, with a learning rate of {0.05, 0.1, 0.25, 0.5} from a validation set for each database. By using 2-grams to validate the correct output, we boost the accuracy by 1-4%.

4.2.3 MDLText. For MDLText, Silve et al. use the macro-average F-measure function mentioned:

$$F - measure_{macro} = 2 \times \frac{Precision_{macro} \times Recall_{macro}}{Precision_{macro} + Recall_{macro}} \tag{19}$$

$$Recall_{macro} = \frac{1}{m} \sum_{j=1}^{m} \frac{TP_j}{TP_j + FN_j} \tag{20}$$

$$Precision_{macro} = \frac{1}{m} \sum_{j=1}^{m} \frac{TP_j}{TP_j + FP_j} \tag{21}$$

It calculates a more accurate result with true positives (TP), false positives (FP), false negatives (FN), and the number of classes (m). 5-fold cross-validation is performed on each database ten times by Silva et al.. Table 1 shows the results that will be discussed in detail in section 5.

4.2.4 Evaluation. We see that all three algorithms use different techniques to calculate the accuracy. The only difference between the algorithms is that some are more accurate, for example, by using negative samples. To compare the three, we look at the average, minimum, maximum, and stability of each algorithm's accuracy with the evaluation criteria, where stability means the difference between the minimum and maximum.

4.3 Evaluation criteria

To compare different areas of application, we need specific evaluation criteria. From the databases mentioned earlier, we can group the criterias in the language and the type of document it represents.

The different evaluation tables are uploaded on the GitHub page of this paper. [1]

4.4 Language results

We evaluate the results of each algorithm with English and Multilingual databases. We compare their accuracy to determine if any algorithm performs better in certain conditions.

4.4.1 English. We see that MDLText has the best average accuracy with 86,62% in table 3. We also see that it performs best with English web pages. The difference between the maximal and minimal accuracy is $|98,5\% - 67,2\%| = 31,3\%$.

fastText is not far behind, with an average accuracy of 79, 87%. The stability is the worst with $|95,7\% - 60,2\%| = 35,5\%$.

Skip-gram has an average accuracy of 57, 5% and has a difference of maximal and minimal accuracy of $|72\% - 43\%| = 29\%$.

4.4.2 Multilingual. We see that MDLText has an accuracy of 77,69%. The difference between the best and the worst result is $|88,3\% - 68,7\%| = 19,6\%$. Compared to the stability in English texts of 31, 3% this is not as bad.

For fastText, we have just one result with an accuracy of 98,6%. Because this is just one test case, it cannot stand for the entirety of fastText. But important to note is that the result is still better than all results of MDLText.

The third algorithm has no multilingual test cases, so it will be skipped in this criteria.

4.5 Document type results

In addition to the language, the type of documents can also affect accuracy. So we compare two types of documents: news articles (NEWS) and web pages(WEB).

Because skip-gram uses word pairs with a regular vocabulary, the algorithm gets a type "word-pairs" (WP) to compare it in the end to the general accuracy and will be ignored when comparing other document types.

4.5.1 News articles. The best result has fastText with an average accuracy of 94,65% with two databases. The difference between them is $|96,8\% - 92,5\%| = 4,3\%$.

MDL has an average accuracy of 94,65% given sixteen datasets. The difference between the maximum of 92% and the minimum accuracy of 67,2% is $|92\% - 67,2\%| = 24,8\%$.

4.5.2 web pages. MDLText processed nine datasets to get an average accuracy of 83,6%. Its stability is $|98,5\% - 68,7\%| = 29,8\%$.

The accuracy of the only available dataset of FastText was better than MDLText's maximum. But because it is just one test case, it does not give an accurate representation.

4.5.3 Word pairs. As we said before, skip-gram is the only dataset that uses word pairs. With that in mind, we can see that it has an average of 57, 4%. The maximum it archives is here 78% and the minimum is 35%, hence the difference between the two is $|78\% - 35\%| = 43\%$. As can be seen, this difference is the largest of all types.

Table 1: Accuracy of the MDL database

language	type	Database	MDLText
EN	NEWS	20newsgroups	90,20%
EN	WEB	7Sectors	91,30%
EN	SCIENCE	ACM	82,20%
EN	SCIENCE	CSTR	89,20%
MULTI	WEB	Dmoz-Business	68,70%
MULTI	WEB	Dmoz-Computers	70,40%
MULTI	WEB	Dmoz-Health	82,60%
MULTI	WEB	Dmoz-Science	73,70%
MULTI	WEB	Dmoz-Sports	88,30%
EN	Mail	Enron	71,60%
MULTI	NEWS	Fbis	79,00%
MULTI	WEB	Industry-Sector	81,10%
EN	NEWS	Irish economic	67,20%
EN	NEWS	La1S	87,70%
EN	NEWS	La2S	88,30%
EN	NEWS	Latimes	84,70%
EN	NEWS	New3S	85,80%
EN	SCIENCE	Oh0	92,50%
EN	SCIENCE	Oh10	81,90%
EN	SCIENCE	Oh15	85,90%
EN	SCIENCE	Oh5	90,70%
EN	SCIENCE	Ohscal	78,00%
EN	Q+A	Opinosis	67,40%
EN	SCIENCE	Pubmed-Cancer	92,00%
EN	SCIENCE	Pubmed-Cancer-2000	84,80%
EN	NEWS	RCV1 Top-4	92,00%
IT	NEWS	Rcv2-Italian	79,40%
PT	NEWS	Rcv2-Portuguese	79,10%
SP	NEWS	Rcv2-Spanish	84,10%
EN	NEWS	Re0	81,50%
EN	NEWS	Re1	81,20%
EN	NEWS	Re8	91,40%
EN	NEWS	Reuters	91,20%
EN	NEWS	Q+As	90,30%
EN	WEB	Techtc300-1092-135724	97,80%
EN	WEB	Techtc300-1092-789236	98,50%
EN	TRC	Tr11	82,70%
EN	TRC	Tr12	87,90%
EN	TRC	Tr21	83,60%
EN	TRC	Tr23	92,10%
EN	TRC	Tr31	82,60%
EN	TRC	Tr41	92,40%
EN	TRC	Tr45	90,30%
EN	MAIL	Trec7-3000	98,10%
		average	84,80%

[1]evaluation tables:
github.com/Koraiko/Seminar-Lighweight_Text_Classification/tree/main/tables

Table 2: Accuracy of the skip-gram and FastText databases

language	type	Database	skip-gram	FastText, h=10 2-gram
EN	NEWS	AG’s News		92,5%
ZN	NEWS	Sogou News		96,8%
MULTI	WEB	DBPedia		98,6%
EN	Q+A	Yelp Rev. Pol.		95,7%
EN	Q+A	Yelp Rev. Full		63,9%
EN	Q+A	Yahoo!Answers		72,3%
EN	Q+A	Amzon Rev. full		60,2%
EN	Q+A	Amzon Rev. Pol.		94,6%
AR	WP	WS353	51%	
DE	WP	GUR350	61%	
DE	WP	GUR65	78%	
DE	WP	ZG222	35%	
EN	WP	RW	43%	
EN	WP	WS353	72%	
ES	WP	WS353	57%	
FR	WP	RG65	70%	
RO	WP	WS353	48%	
RU	WP	HJ	59%	
		average	57,4%	84,33%

Table 3: Summary of the accuracy of the 3 algorithms

evaluation criteria		skip-gram	FastText h=10 2-gram	MDLText,
general	average	57,4%	84,33%	84,8%
	min	35%	60,2%	67,2%
	max	78%	98,6%	98,5%
	stability	43%	38,4%	31,3%
NEWS	average	-	94,65%	84,57%
	min	-	92,5%	67,2%
	max	-	96,8%	92%
	stability	-	4,3%	24,8%
WEB	average	-	98,6%	83,6%
	min	-	98,6%	68,7%
	max	-	98,6%	98,5%
	stability	-	0%	29,8%
word pairs	average	57,4%	-	-
	min	35%	-	-
	max	78%	-	-
	stability	43%	-	-
EN	average	57,5%	79,87%	86,62%
	min	43%	60,2%	67,2%
	max	72%	95,7%	98,5%
	stability	29%	35,5%	31,3%
MULTI	average	-	98,6%	77,69%
	min	-	98,6%	68,7%
	max	-	98,6%	88,3%
	stability	-	0%	19,6%

5 COMPARISON

We compare the results we got, with the criteria of who got the best accuracy, related to the most accurate, the most steady algorithm, and the influence of language. With the conclusion of each point, we discuss a practical field of application for each algorithm.

5.1 skip-gram

We see skip-gram has the lowest results in all of the three algorithms. With that in mind, we can easily say that skip-gram is impracticable to use with an average accuracy of 57,4%. Furthermore, the fact that some results are below 50% is alarming.

5.1.1 Problems. Part of the problem is that we have just one word as input and try to guess the context words from the subwords. So the context of the used word is not trained. In some cases, we cannot detect the subwords and did not train these words in the training stage. Therefore, the result shifts, and we falsify the scores.

Skip-gram is also not a very steady algorithm with a difference of 43% from the maximum and minimum.

5.1.2 Possible solution. This problem exists because of the broad range of the trained word-pairs: The datasets have ordinary words and do not have a type like the databases of the other two algorithms. This makes them broader in usage but also not as accurate. We see that some databases like GUR65 have the highest accuracy with 78%. So it is at least possible to get an accuracy of 75% or higher with skip-gram.

The problem is that we need to train it in a specific topic with handpicked word pairs. The word pairs should be common subwords of that type of text. With that, we get the subwords out and get higher accuracy.

5.1.3 Application area. The accuracy issue makes it less practical than the other two, but skip-gram is still usable in certain conditions:

With the problem of accuracy and one-word input, we can train the algorithm with elemental words, also known as subwords, to avoid training too long or uncommon word pairs. With this, we can deduct the class of a text more easily.

The problem with uncommon words makes skip-gram unsuitable for untrained environments. Skip-gram should therefore train with word pairs carefully selected from that environment. For example, a cooking platform would take different recipe types and pair them with the names of the product.

The difference between skip-gram and the other algorithms is that skip-gram uses just one word. That detail could be useful in a search engine, where people write the most critical words and want a result for these.

5.2 fastText

fastText has the second-best accuracy of our algorithms. It is not far behind MDLText with an average accuracy of $84,33\%$. The difference to MDLText is just 0,48% for the average accuracy. If we round the results to an integer, the two algorithms are equal in accuracy.

The minor difference of $0,47$ between FastText and MDLText, compared to the difference of $26,93\%$ to skip-gram, could be explained by the algorithm getting better information from context words than just one word. Through that, it has more information in the training phase about the context of the used word. The negative sampling function at the end improves the accuracy of fastText by using negative words from the dictionary. By using 2-grams it improves even more by 1-4% [4].

5.2.1 Problems. By looking at the evaluation criteria, we see that it gets over 79% average accuracy. But we must look at it with caution because there are fewer than two datasets in some of these evaluation criterias.

FastText's accuracy, for example, is lower than MDLText's when it comes to the English evaluation criteria. So the average accuracy of each evaluation criteria except English could change with more test cases.

5.2.2 Application area. Through the points mentioned before, we see that fastText has a high accuracy if we look at the evaluation criterias, but if we look at all datasets together, we see that the average is less than the results of the individual criteria, but still over 75%. So we do not have a problem of too low accuracy in the usage. When comparing it to MDLText, we see fastText is not far behind MDLText in accuracy.

Because fastText uses context words to calculate the class of the text, we can easily say that it is not practical in search engines like skip-gram is, where people search after important keywords and ignore grammatical structures. A review portal where people write short paragraphs of their opinions is also not a good application area, due to the short and inconsistent texts.

So a practical usage is document classification like web pages and news articles. These documents communicate the class straightforward to the user. Like skip-gram fastText benefits from a specific training dataset to be more accurate for the usage.

5.3 MDLText

MDLText has the best results across almost every evaluation category. It does not have the problems of fastText of not enough datasets for each evaluation criteria. With 7-44 datasets in each category, it is representative for MDLText. Since it has the highest stability it is more stable than the other algorithms.

5.3.1 Problems. The only problem is the complexity of MDLText. It is more complex than skip-gram or fastText. As a result, it leans more toward the non-lightweight algorithm of classification. Although MDLText is more complex than the other algorithms mentioned here, the training and classification times are shorter than training and classifying with non-lightweight algorithms.

5.3.2 Application area. Looking at MDLText in table 1, we can easily see that evaluation criterias still have highly accurate values.

However, science papers, particularly medical papers, have the highest accuracy of all datasets, as seen in table 1. Hence we recommend its application for science papers, especially medical papers. The reason is that the datasets in Silva et al. use medical datasets to test the algorithm and the highest accuracy was in these cases. Because of the stability of the accuracy, using it in different fields has no disadvantages.

6 CONCLUSION

In this paper, we looked at three different approaches to text classifications. We discussed the problems of the algorithms, presented a solution if it was available, and tried to find a good application of the algorithm.

skip-gram is promising for easy word tag searches in a specially trained environment. For fastText and MDLText, it is advantageous to use them in environments with sentences, with MDLText being more precise but more complex than fastText.

Our discussion showed that even when the accuracy is low, it is still practical in specific fields, and more complex algorithms are not always better.

ACKNOWLEDGMENTS

I would like to thank my colleagues for proofreading and giving insights as a third party.

REFERENCES

[1] Giacomo Berardi, Andrea Esuli, and Diego Marcheggiani. 2015. Word Embeddings Go to Italy: A Comparison of Models and Training Datasets. In *Proceedings of the 6th Italian Information Retrieval Workshop, Cagliari, Italy, May 25-26, 2015 (CEUR Workshop Proceedings, Vol. 1404)*, Paolo Boldi, Raffaele Perego, and Fabrizio Sebastiani (Eds.). CEUR-WS.org. http://ceur-ws.org/Vol-1404/paper_11.pdf

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. 2016. Enriching Word Vectors with Subword Information. *CoRR* abs/1607.04606 (2016). arXiv:1607.04606 http://arxiv.org/abs/1607.04606

[3] Joshua Goodman. 2001. Classes for Fast Maximum Entropy Training. *CoRR* cs.CL/0108006 (2001). https://arxiv.org/abs/cs/0108006

[4] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomás Mikolov. 2016. Bag of Tricks for Efficient Text Classification. *CoRR* abs/1607.01759 (2016). arXiv:1607.01759 http://arxiv.org/abs/1607.01759

[5] Ria Kulshrestha. 2019. NLP 101: Word2Vec — Skip-gram and CBOW. https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314

[6] Ken Lang. 1995. NewsWeeder: Learning to Filter Netnews. In *in Proceedings of the 12th International Machine Learning Conference (ML95.* 331–339. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.6286

[7] Sabine Schulte im Walde Maximilian Koper, Christian Scheible. 2015. Multilingual Reliability and "Semantic" Structure of Continuous Word Spaces. https://aclanthology.org/W15-0105.pdf

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 [cs.CL]

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.), Vol. 26. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

[10] Rafael Geraldeli Rossi, Alneu de Andrade Lopes, and Solange Oliveira Rezende. 2016. Optimization and label propagation in bipartite heterogeneous networks to improve transductive classification of texts. *Information Processing Management* 52, 2 (2016), 217–257. https://doi.org/10.1016/j.ipm.2015.07.004

[11] Renato M. Silva, Tiago A. Almeida, and Akebo Yamakami. 2017. MDLText: An efficient and lightweight text classifier. *Knowledge-Based Systems* 118 (2017), 152–164. https://doi.org/10.1016/j.knosys.2016.11.018

[12] Lukás Svoboda and Tomás Brychcín. 2016. New word analogy corpus for exploring embeddings of Czech words. *CoRR* abs/1608.00789 (2016). arXiv:1608.00789 http://arxiv.org/abs/1608.00789

[13] Lungan Zhang, Liangxiao Jiang, Chaoqun Li, and Ganggang Kong. 2016. Two feature weighting approaches for naive Bayes text classifiers. *Knowledge-Based Systems* 100 (2016), 137–144. https://doi.org/10.1016/j.knosys.2016.02.017

[14] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level Convolutional Networks for Text Classification. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf