


ÖNDER ÇOBAN 

## ASSESSMENT OF NATURE-INSPIRED ALGORITHMS FOR TEXT FEATURE SELECTION

**Abstract** *This paper provides a comprehensive assessment of basic feature selection (FS) methods that have originated from nature-inspired (NI) meta-heuristics; two well-known filter-based FS methods are also included for comparison. The performances of the considered methods are compared on four balanced high-dimensional and real-world text data sets regarding the accuracy, the number of selected features, and computation time. This study differs from existing studies in terms of the extent of experimental analyses that were performed under different circumstances where the classifier, feature model, and term-weighting scheme were different. The results of the extensive experiments indicated that basic NI algorithms produce slightly different results than filter-based methods for the text FS problem. However, filter-based methods often provide better results by using lower numbers of features and computation times.*

**Keywords** nature-inspired algorithms, feature selection, text categorization

**Citation** Computer Science 23(2) 2022: 179–204

**Copyright** © 2022 Author(s). This is an open access publication, which can be used, distributed and reproduced in any medium according to the Creative Commons CC-BY 4.0 License.

## 1. Introduction

Text categorization (TC) is one of the most effective ways [7,22] of automatically categorizing documents into predefined sets of classes or categories. Many real-world applications use TC methods, including sentiment analysis, cyberbully detection, spam filtering, and authorship classification (to name a few). In classical TC, the data at hand has a very large number of features – especially when the features are extracted with the bag-of-words (BoW) model. Therefore, a very high amount of data volume makes the most of the text-processing applications to deal with high dimensionality in today’s world [1, 2, 6, 32]. High dimensionality and the presence of noise significantly degrade the performance of machine-learning (ML) tasks. Due to the strong interdependence among individual features or the behaviors of combined features, identifying and extracting patterns or rules in such a situation is extremely difficult [9]. Moreover, an ML model that is designed by a high-dimensional data set is difficult to understand and employ [32]. As such, there is a need to work on a subset of features rather than a complete data set.

Feature selection (FS) is an important step in TC for overcoming the aforementioned challenges by discovering a subset of features that can effectively represent an original data set [2, 24]. In other words, FS is a task that involves the extraction of meaningful information from mountains of data by using a small collection of attributes [32]. Hereby, FS also contributes to enhancing the classification accuracy by creating an accurate predictive model by reducing over-fitting, computation time, and storage [24]. Various techniques have been developed in the literature to measure the quality of discovered feature subsets. Considering the evolution criteria, these techniques are categorized into three types; namely, filters, wrappers, and embedded methods [5, 22, 24]. In filter methods, various distance and information measures are usually used as evolution criteria. Then, the features are ranked based on the employed importance measure (chi-square, mutual information, and so on) [9]. In wrapper methods, the performance of a specific learning algorithm is used to measure the quality of a feature subset. On the other hand, embedded methods are internally involved during the learning processes [24]. As an instance of wrapper methods, nature-inspired (NI) algorithms are becoming popular choices for the engineering of optimization problems [33]. These techniques work without derivatives; thus, they are suitable for problems with high-dimensional spaces. Besides the optimization problems, meta-heuristic algorithms have also been used specifically for creating an optimal set of features.

The use of NI algorithms for FS has, therefore, received considerable attention in various domains (including healthcare, finance, social and behavioral science, TC, and so on) [5, 32, 33]. Examples of NI methods are the genetic algorithm (GA) [18], particle swarm optimization (PSO) [14], the firefly search algorithm (FS) [36], the artificial bee colony algorithm (ABC) [16], and so on. From the TC perspective, many studies have tried to employ [1, 2, 21, 31] or improve [7, 19] NI algorithms when dealing with

text FS. In addition, some studies have proposed new [22] or hybrid methods of NI algorithms [4, 6, 7, 30] for text FS.

This paper presents an up-to-date comprehensive assessment of NI methods for FS on text data. For this purpose, two well-known filter methods (namely, chi-square and mutual information) are compared to major representatives of NI algorithms. The performances of the considered FS methods are compared by using four high-dimensional text data sets. To validate their efficacies, selected feature subsets are used to build classification models under different circumstances. Notice that there are a few previous studies [1, 2, 12, 30] that have focused on comparing NI algorithms for text FS; however, these studies were narrower in their scopes and did not consider the effect of the term-weighting process. As such, this study differs from the existing studies with respect to the extent of the considered algorithms and extensive experiments that were obtained under different circumstances where the classifiers, feature models, and weighting schemes were different.

The novelty of the results of this study lies in the following considerations: (i) employing basic NI methods without any extensions or improvements paves the way for them to fall behind classical filter-based methods in terms of their accuracy; and (ii) it is a better choice to apply both NI and filter-based methods on  $tf*idf$ -weighted text data sets to achieve a better result.

## 2. Related work

Hundreds of studies exist that focus on evolutionary optimization algorithms. Among these algorithms, the use of NI methods specifically for FS has attracted the attention of the research community. This section only focuses on related studies that employ NI algorithms for text FS.

The published works on this topic are as follows. An improved sine cosine algorithm (SCA) was proposed in [5], where the authors showed that their method achieved high performance and was useful for the TC task. In [22], a novel multi-objective algorithm for text FS was proposed that includes two objective functions. The first objective function measures the relevance between the features and a target category, while the second one evaluates the inter-correlation of the features. The authors compared the proposed method against well-known filter methods and concluded that it provided a better performance on benchmark data sets. A new text-FS method based on the bat algorithm (BAT) was proposed in [7]. This method simply employs the bat algorithm for FS over feature subsets that are selected with the help of filter methods. The proposed method improves the classification accuracy as compared to traditional FS methods. In [23], the authors compared four different entropy-based measures for TC and proposed a new measure that combined the entropy of a word with its distribution among documents in a collection. A narrow scope comparison of ant colony optimization (ACO) against information gain (IG) and chi-square (CHI) for text FS was performed in [2] and [1]. The authors concluded that ACO outperformed the other two filter methods on a benchmark data set.

An enhanced binary grey wolf optimizer (GWO) was proposed in [6] for text FS in Arabic TC tasks. The results and analysis of the authors showed that the proposed method enhanced the efficacy of the TC task in the Arabic language. A hybrid algorithm of GA and ACO was proposed in [4] for text FS. The authors concluded that the hybrid method outperformed the ACO, CHI, and IG methods on a benchmark data set. Another narrow-scope comparison of GA, harmony search (HS), PSO, and simulating annealing for FS was performed with different classifiers for spam classification. According to the results, FS improved the accuracy – especially when a support vector machine was used as a classifier. A hybrid GWO and grasshopper optimization algorithm (GOA) was used to select text features in [30]. The experiments showed that the hybrid algorithm provided better results than the GOA and GWO when it is employed with different classification algorithms.

A fine-tuned GOA algorithm with classifiers were used for TC on different multi-class data sets in [19]. According to the results, it was shown that the used approach outperformed its state-of-the-art peers. A comparison of traditional and swarm intelligent-based search methods (i.e., ACO and ABC) was performed for correlation-based text FS in [21]. The authors concluded that any features that were selected by using a swarm intelligence search were more robust for classification results. Another narrow scope comparison was performed in [12] to detect the best feature selector among the GA, BAT, and GWO methods for sentiment analysis. The experiments and analysis showed that GA was the most effective method concerning the feature-reduction percentage, while GWO provided a result that was inferior to that of the hybrid GA/bat algorithm.

Based on our review of the literature, it can be understood that there are not any previous studies that have comprehensively assessed major NI algorithms for the TC task – especially for the effect of the term-weighting phase. As such, this paper addresses this purpose by performing experiments under different cases where the feature-representation, evaluation, and FS methods are different.

## 3. Methods

### 3.1. Preprocessing

In TC, texts need to be passed through a preprocessing step in order to transform them into a classification-ready structure. In this study, therefore, simple preprocessing has been applied to the data sets. First, specific steps were applied to keep the feature characteristics of the data sets (see Section 4 for further details). For the CB data set, emoticons were encoded to keep them within the content and used as features [27]. For TTC, on the other hand, it was provided to prevent the underscore character, as this data set includes domain-specific multi-terms joined with the underscore character [43].

A vectorizer was employed with a minimum document frequency of 2 and 3 for CB and TTC, respectively. On the other hand, the benchmark data sets (i.e., R8 and N8)

were preprocessed with a minimum term length of 2 and a minimum document frequency of 3. Additionally, stopwords removal and stemming steps (with the help of Porter Stemmer [29]) were employed with the help of the `nlTK`<sup>1</sup> package for only the bow feature model. Finally, all of the four data sets passed through the common steps that include removing the numerical values and punctuation as well as lower-casing content.

### 3.2. Feature extraction and term weighting

This step converts the text data into a term-document matrix where the rows and columns represent documents and extracted features, respectively. In classic TC, texts (or documents) are converted into the term-document matrix by extracting word-level and character-level features. In this study, bow features were used as word-level features, while tri-grams were used as character-level features. The bow model took each word as a separate feature, whereas the character-level model considered each subsequent string with a length of  $n$  as a new feature [8]. Upon completion of the feature extraction, an important intermediate step (namely, term weighting) was used to assign a weight to each of the extracted features. In the TC field, this process was used to determine the importance of a feature for classifying the texts into two or more predefined categories. In this study, well-known and unsupervised weighting schemes (namely, term frequency [ $tf$ ], binary, and term frequency-inverse document frequency [ $tf \cdot idf$ ]) were used to assign weights to the features. These methods were formulated as follows [8]:

$$W_{tf}(c, d) = tf(c, d) \quad (1)$$

$$W_{binary}(c, d) = \begin{cases} 1 & \text{if } tf(c, d) \geq 1 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

$$W_{tf \cdot idf}(c, d) = tf(c, d) \cdot \log \frac{N}{df(c)} \quad (3)$$

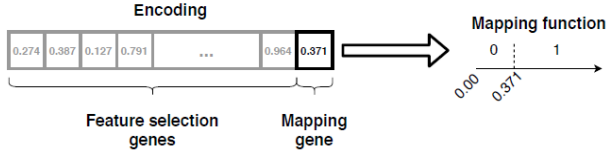
where  $c$  represents a term observed in document  $d$ , while  $N$  is the total number of documents and  $df(c)$  is the document frequency of term  $c$ .

The main reason for using different weighting schemes is to observe the behaviors of the FS algorithms (especially the NI ones) according to the selected-weighting method. This is because mapping the genes to the absence of features affects the performances of the NI algorithms. Additionally, term weighting has a direct effect on both the inner and outer evaluation phases for wrapper-based FS methods. Therefore, this investigation is one of the main contributions of this study, as there has not been any previous work that has focused on this issue. In this study, the `evopreprocess` library was used to employ the NI algorithms for text FS. Notice that this library uses a self-adaptation mechanism where only one gene is used to encode the genes for FS.

As can be seen in Figure 1, the mapping gene is obtained from the genotype and adapted during the evolution process (depending on the data set at hand).

---

<sup>1</sup><https://www.nltk.org/>



**Figure 1.** Example self-adaptation with mapping gene in *evopreprocess* [17] for encoding phase of NI feature selectors

### 3.3. Feature-selection algorithms

A feature (i.e., an attribute) is an important characteristics of a data set. FS is a process of selecting a meaningful subset of features with respect to the optimization problem. Formally, the objective of FS is to extract a minimal subset of  $m$  features  $(t_1, t_2, t_3, \dots, t_m)$  from an original data set with  $n$  features  $(t_1, t_2, t_3, \dots, t_n)$ , where  $m < n$  [32]. This section presents a brief description of the filter-based and wrapper-based FS algorithms that were used in this study.

#### 3.3.1. Filter methods

Filter-based FS methods eliminate redundant features based on several criteria such as document frequency, term frequency, gini index, and so on. Among the filter-based methods, the most widely used ones are chi-square and mutual information [7, 35]. As such, these two filter-based methods were used in this study to make comparisons with the NI methods. The following two sub-headings present brief descriptions of the aforementioned methods.

**Chi-square:** From the FS perspective, chi-square (CHI) is used to measure how a term is independent from a category [35, 42]. Using a two-way contingency table of a term  $t$  and a category  $c$ , this is formulated as follows [42]:

$$\chi^2(t, c) = \frac{N \times (AD - CB)^2}{(A + C) \times (B + D) \times (A + B) \times (C + D)} \quad (4)$$

where  $A$  and  $C$  represent the numbers of instances that belong to category  $c$  and with/without term  $t$ , respectively. Similarly,  $B$  and  $D$  represent the number of instances that do not belong to category  $c$  and with/without term  $t$ , respectively. Finally,  $N$  is the total number of documents.

**Mutual information:** Mutual information (MI) takes its maximum value if there is a strong dependence between a term and a class. In TC, the MI value of a term  $t$  with respect to a class  $c$  is computed as follows [35, 42]:

$$MI(t, c) = \log \frac{A \times N}{(A + C) \times (A + B)} \quad (5)$$

where  $A$  is the number of times that  $t$  and  $c$  co-occur,  $B$  is the number of times that  $t$  occurs without  $c$ ,  $C$  is the number of times that  $c$  occurs without  $t$ , and  $N$  is the total number of documents [42].

### 3.3.2. NI meta-heuristics

There are numerous NI algorithms – more than a hundred different algorithms and variants are estimated to exist [38]. As it is not possible to include even a good fraction of these algorithms, we used algorithms in this study that were particularly considered to be major representatives – those that are based on swarm intelligence. This subsection presents the major characteristics of these algorithms instead of their fully detailed descriptions and mathematical backgrounds.

**The bat algorithm** (BAT) [41] uses frequency-tuning and the echo-location behavior of bats. It also uses pulse-emission rate  $r$  and loudness  $A$  to control the positions and velocities of the bats at each time step  $t$  as follows [33, 37, 41]:

$$v_i^t = v_i^t + (x^* - x_i^{t-1}) \overbrace{(f_{min} + (f_{max} - f_{min})\beta)}^{f_i} \text{ and } x_i^t = x_i^{t-1} + v_i^t \quad (6)$$

where  $\beta \in [0, 1]$  is a random number that is drawn from uniform distribution so that frequency  $f_i$  can vary between  $f_{min}$  and  $f_{max}$ .  $x_*$  is the position of the current global best bat. For each bat, alpha ( $0 < \alpha < 1$ ) and gamma ( $\gamma > 0$ ) parameters are used to update the loudness and pulse emissions as follows [33, 37]:

$$A_i = \alpha A_i^{t-1} \text{ and } r_i^t = r_i^0 (1 - e^{\gamma(t-1)}) \quad (7)$$

After updating the global and local fitness, the position of each bat is also updated as [33]  $x_{new} = x_{old} + \epsilon A'_t$ , where the  $\epsilon$  parameter takes random values between  $-1$  and  $+1$ . The reader is advised to check [41] for more details about the mathematical background of the bat algorithm.

**The grey wolf algorithm** (GWO) [26] was developed based on the social intelligence of grey wolf packs in leadership and hunting. In the social hierarchy of grey wolves, the most powerful is the alpha wolf ( $\alpha$ ), which guides the whole group, while the second-strongest is the beta wolf ( $\beta$ ), which takes the lead of the group when the alpha is ill, dead, or is not among the pack [3, 10]. The delta ( $\delta$ ) and omega ( $\omega$ ) wolves have less power and domination than the alpha and beta wolves. The alpha, beta, and delta wolves in GWO represent the first-, second-, and third-best solutions, respectively, while the omega wolf represents the rest of the candidate solutions [26]. Formally, the mathematical model of GWO is comprised of three main components: namely, encircling, hunting, and attacking their prey. The following equation is used in the encircling-the-prey component [26]:

$$\vec{X}(t+1) = \vec{X}_p(t) - \vec{A} \cdot |\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)| \quad (8)$$

where  $\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a}$  and  $\vec{C} = 2 \cdot \vec{r}_2$  are the coefficient vectors, while  $\vec{X}_p$  and  $\vec{X}$  are the position vectors of the prey and grey wolves, respectively. The  $\vec{r}_1$  and  $\vec{r}_2$  components are random vectors that are within a range of  $[0, 1]$ , whereas  $\vec{a}$  is a vector whose values are decreased from 2 to 0 during the course of the iterations. In the hunting

component, the three best solutions ( $\alpha$ ,  $\beta$ , and  $\delta$ ) are retained, and the other candidate solutions ( $\omega$ ) are forced to update their positions accordingly as follows [3, 10, 26]:

$$\vec{X}(t+1) = (\vec{X}_1 + \vec{X}_2 + \vec{X}_3)/3 \quad (9)$$

where  $\vec{X}_1 = \vec{X}_\alpha(t) - \vec{A}_1 \cdot |\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}|$ ,  $\vec{X}_2 = \vec{X}_\beta(t) - \vec{A}_2 \cdot |\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}|$ , and  $\vec{X}_3 = \vec{X}_\delta(t) - \vec{A}_3 \cdot |\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}|$ . Attacking the prey component is expressed based on  $\vec{a}$  and is linearly reduced as  $\vec{a} = 2 - t(2/T)$ , where  $t$  and  $T$  are the current iteration and the maximum number of iterations, respectively [3, 10]. The reader is advised to check [26] for more details about the GWO algorithm.

**The sine cosine algorithm** (SCA) [25] is based on sine and cosine functions – it updates its solutions at each iteration as follows [5, 25]:

$$X(i, j)_{t+1} = \begin{cases} X(i, j)_t + r_1 \cdot \sin(r_2) \cdot |r_3 P(j)_t - X(i, j)_t| & \text{if } r_4 < 0.5 \\ X(i, j)_t + r_1 \cdot \cos(r_2) \cdot |r_3 P(j)_t - X(i, j)_t| & \text{if } r_4 \geq 0.5 \end{cases} \quad (10)$$

where  $X(i, j)_t$  and  $P(j)_t$  are the position of the current solution and the position of the best individual  $i$  in the  $j^{\text{th}}$  dimension at the  $t^{\text{th}}$  iteration, respectively [5].  $r_1$ ,  $r_2$ ,  $r_3$ , and  $r_4$  are random variables from which  $r_1$  is a control variable that is used to guide the next position's region as  $r_1 = a - t * a/T$ , where  $t$  is the current iteration,  $T$  is the maximum number of iterations, and  $a$  is a constant. The reader is advised to check [25] for further details and the mathematical background of the SCA algorithm.

**The artificial bee colony algorithm** (ABC) [16] was developed by inspiring the intelligent foraging behavior of honey bees. This algorithm consists of three groups of bees: employed bees, onlookers, and scouts. In the ABC algorithm, the position and nectar content of a food source represent a possible solution and its fitness, respectively. Each food source is exploited by one employed bee; therefore, the number of employed bees is equal to the number of food sources [11]. A scout is a bee whose food source has been abandoned. An onlooker bee selects a food source using its fitness as follows [11, 16]:

$$p_i = \frac{fit_i}{\sum_{i=1}^{SN} fit_i} \quad (11)$$

where  $fit_i$  and  $SN$  are the fitness value of solution  $i$  (food source) and the number of solutions (food sources), respectively. Then, candidate solution  $V_i = [v_{i1}, v_{i2}, \dots, v_{iD}]$  is created from the old one  $X_i = [x_{i1}, x_{i2}, \dots, x_{iD}]$  as follows [11, 16]:

$$v_{i,j} = x_{i,j} + \phi_{i,j}(x_{i,j} - x_{k,j}) \quad (12)$$

where  $k \in \{1, 2, \dots, SN\}$  is a randomly selected variable (different from  $i$ ),  $D$  is the number of variables of the objective function, and  $\phi$  is a random variable that is within a range of  $[-1, 1]$ . If a new candidate solution is of higher quality than an old one, it replaces the old one; otherwise, the old one is retained. If a solution



cannot be improved with a predefined number of iterations (cycles), then this solution is assumed to be abandoned. In this case, a scout bee discovers a new solution to replace the abandoned solution as follows [11, 16]:

$$x_{i,j} = x_{min,j} + rand(0, 1)(x_{max,j} - x_{min,j}) \quad (13)$$

The reader is advised to check [16] for further details of the ABC algorithm.

**The cuckoo search algorithm** (CS) [40] was developed based on the reproduction technique of cuckoo birds. These birds lay their eggs in other host birds' nests instead of building their own nests. Cuckoo eggs mimic the physical characteristics of a host bird's eggs with respect to their spot patterns and colors [33]. Their strategy ends up unsuccessfully if the host bird discovers the cuckoo's eggs. CS uses the following equation to generate new solutions by performing a Lévy flight [40]:

$$x_i^{t+1} = x_i^t + \alpha \oplus \text{Lévy}(\lambda) \quad (14)$$

where  $\alpha > 0$  is the step size, and  $\oplus$  represents entry-wise multiplications. The step length is drawn from the Lévy distribution as follows [33, 39, 40]:

$$\text{Lévy} \sim u = t^{-\lambda}, (1 < \lambda \leq 3) \quad (15)$$

which has an infinite variance with an infinite mean. The reader is advised to check [39, 40] for more details about the mathematical background of the CS algorithm.

**The differential evolution algorithm** (DE) [34] generates new solution vectors by adding the weighted difference of two different solution vectors ( $x_p$  and  $x_q$ ). A numerical parameter  $F \in [0, 2]$  is used to scale the mutation operator, which is then used to perturb an existing solution  $x_r$  to generate a new solution ( $x_k = x_r + F(x_p - x_q)$ ) [37].  $p$ ,  $q$ , and  $r$  are randomly selected integers (i.e., they stand for different solution vectors) and are different from running index  $k$ . The diversity of the mutated vector is increased by employing a crossover, which is used to form a trial vector. Afterward, a selection mechanism is used to save any new solution  $u^t$  that is better than previous solutions with respect to fitness [37]:

$$x_i^t = \begin{cases} u^t & \text{if } f(u^t) \leq f(x_i^{t-1}) \\ x_i^{t-1} & \text{otherwise} \end{cases} \quad (16)$$

There are different variants of DE in terms of the mutation operator. The reader is advised to check [34] for more details about, the mathematical background of, and the variants of the DE algorithm.

**The harmony search algorithm** [13] was inspired by the music-improvisation process where musicians improvise their instruments' pitches by searching for a better state of harmony [15]. The procedure of HS includes the following main steps: (i) initialize the problem and algorithm parameters; (ii) initialize the harmony memory

(HM); (iii) improvise a new harmony; (iv) update HM; (v) check the stopping criteria [15, 35]. The reader is advised to check [13] for more details of the HS algorithm.

**The genetic algorithm** (GA) [14] was inspired by the evolution of biological systems. This algorithm has three main genetic operators: crossover, mutation, and selection [37]. These three operators are used to generate new solutions – each of which is encoded with binary or real strings to create a “chromosome.” The crossover (or recombination) operator creates a new solution by combining the genetic information of two existing parent solutions. The mutation flips between 0 and 1 at random positions of the existing solution for maintaining the genetic diversity. The quality of a solution is determined by its fitness; this is used to select the best solutions. GA does not usually have any explicit equations in terms of generating new solutions [37].

**Particle swarm optimization** (PSO) [18] simulates the swarming behavior of birds and fish using the following equations to update the particles’ positions (i.e., solutions) and velocities [37]:

$$v_i^{t+1} = v_i^t + \alpha\epsilon_1[g^* - x_i^t] + \beta\epsilon_2[x_i^* - x_i^t] \text{ and } x_i^{t+1} = x_i^t + v_i^{t+1} \quad (17)$$

where  $\epsilon_1$  and  $\epsilon_2$  are uniformly distributed random numbers that re within a range of  $[0, 1]$ , while the  $\alpha$  and  $\beta$  parameters are within a range of  $[0, 2]$  [37].  $g^*$  represents the best solution that has been discovered so far for all of the particles in a population, while  $x_i^*$  represents the best solution that has been found by the  $i^{th}$  particle individually. The reader is advised to check [18] for more details of the PSO algorithm.

**The firefly algorithm** (FA) [36] was developed based on the swarming and light-flashing behavior of tropical fireflies [37]. This algorithm can be used to solve multi-model and non-linear problems by using the exponential decay of light absorption and the inverse-square of light variation over distance. The main process that determines the movement of a firefly is as follows [36, 37]:

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2} (x_j^t - x_i^t) + \alpha\epsilon_i^t \quad (18)$$

where  $\alpha$  and  $\gamma$  control the parameters for the step sizes of the random walk and the visibility of the fireflies, respectively. On the other hand,  $\beta_0$  is the attractiveness constant when the distance between two fireflies is zero (i.e.,  $r_{ij} = 0$ ). The reader is advised to check [36] for more details about and the mathematical background of the FA algorithm.

### 3.4. Classification

In the classification stage, well-known traditional ML classifiers from the `scikit-learn` library [28] were used to build the learning models; these were the logistic regression (LR), support vector machine (SVM), naive Bayes (NB), multinomial naive Bayes (MNB), decision trees (DT), and random forest (RF) classifiers.

### 3.5. Performance evaluation

To evaluate the performances of the classifiers, we employed the accuracy (Acc) metric that was computed based on a confusion matrix that stores the number of actual and predicted instances with respect to the categories. Among these values, TN and TP represented the number of correctly categorized negative and positive instances, respectively, while FP and FN represented the number of incorrectly classified negative and positive instances, respectively [8]. Using these values, Acc was calculated as  $\text{Acc} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN})$ .

## 4. Data sets

In this study, four different and balanced data sets were used to evaluate the performances of the classifiers with respect to the FS algorithms. Two of these data sets were of the non-benchmark variety and included texts that were written in the Turkish language. On the other hand, the other two data sets were well-known benchmark data sets that included texts that were written in the English language.

The first non-benchmark data set [27] was comprised of 900 messages that were written in Turkish. These messages were collected manually from Twitter and Instagram. Each message in this data set was labeled “yes” or “no” to show whether the message had cyberbullying content or not. This process yielded a balanced data set in which half of the messages were labeled “yes” and the remaining labeled “no”. On the other hand, the second non-benchmark data set [43] included Turkish news texts from seven categories; namely, world, economy, culture-art, health, politics, sports, and technology. This data set contained 4900 documents, and each category included 700 documents. A subset of this data set was used in this study; due to the size of the data set, it was too extensive for the scope of this paper. This subset was created by taking 50 documents from each of the 7 categories; thus, it contained 350 documents in total. Notice that the first non-benchmark data set is named “CB”, while the second non-benchmark sub-data set is called “TTC” in this study.

The benchmark data sets were created by taking subsets of the well-known Reuters-21578 and 20 Newsgroups data sets, respectively. Notice that this was because sub-setting a data set is a well-known practice in the literature – especially when employed with an algorithm that requires a very high computational complexity. In this phase, R8 was created by taking the eight most observed categories in the Reuters corpus. The R8 data set only included documents that were assigned to just one category. Similarly, documents from 8 categories of the 20 Newsgroups corpus were taken (N8). Please note that those categories that had an equal or similar number of documents were selected to reduce class skewness. Due to the popularity of these data sets, the existing ML libraries make it easy to access, preprocess, and use them in any TC task. As such, the R8 data set was obtained in this study thanks to the `nlTK`<sup>2</sup> package, while the `scikit-learn`<sup>3</sup> package was used to acquire and create

---

<sup>2</sup><https://www.nltk.org/book/ch02.html>

<sup>3</sup>[https://scikit-learn.org/0.19/datasets/twenty\\_newsgroups.html](https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html)

the N8 data set. Then, subsets of these data sets were created by taking the first  $n$  documents from each of the categories within the train/test splits of the R8 and N8 data sets, respectively. We set the values of  $n$  to be 80 and 100 while taking the subsets of the R8 and N8, respectively.

We would like to note that the R8 and N8 data sets were in the form of train/test splits, while the CB and TTC data sets were not published in the form of a train/test split. These data sets were preferred due to their balanced structures; their subsets were used since the size of the original data set was outside the scope of this study.

Table 1 presents the categories and the number of documents within the train/test splits for both of the created R8 and N8 data sets, respectively. On the other hand, Table 2 presents a quantitative description of all of the data sets. As can be seen in Table 2, we intentionally left the data sets to be balanced to prevent our results from being biased.

**Table 1**

Quantitative description of our subsets created from R8 and N8 data sets

R8			N8		
Category	# of instances		Category	# of instances	
	train	test		train	test
acq	80	80	comp.os.ms-windows.misc	100	100
crude	80	80	comp.sys.ibm.pc.hardware	100	100
earn	80	80	comp.windows.x	100	100
grain	80	80	rec.autos	100	100
interest	80	80	sci.crypt	100	100
money-fx	80	80	sci.med	100	100
ship	80	80	sci.electronics	100	100
trade	80	80	sci.space	100	100

**Table 2**

Summarized quantitative description of four utilized data sets

Data set	# of ...			balanced
	classes	instances per class	instances	
CB	2	450	900	✓
TTC	7	50	350	✓
R8	8	80	1280	✓
N8	8	100	1600	✓

## 5. Experimental results

In this study, the feature-extraction, term-weighting, filter-based FS, and classification steps were performed with the help of `scikit-learn` [28], which is one of the most prominent ML libraries in Python. To employ the NI methods, the `evopreprocess`

package was used; this enables practitioners to apply various meta-heuristics for different problems (including data sampling, FS, and data weighting) [17]. For the non-benchmark data sets (i.e., CB and TTC), all of the results were obtained with 5-fold stratified cross-validation (cv) [20]. On the other hand, all of the results for the benchmark data sets (i.e., R8 and N8) were obtained on the train/test splits with a hold-out validation strategy.

During the classification stage, the training set was vectorized at first, and the test set was vectorized by fitting the vectorizer that contains the weights of the extracted features in the training set. Then FS process was applied, and the classification task was performed over reduced training and test sets. With the help of the `EvoFeatureSelection` wrapper in the `evopreprocess` [17], the classifiers (i.e., LR, SVM, DT, and RF), MI algorithm, and NI algorithms were employed with a random seed of 123 to ensure that the different replicates of the algorithms produced the same results. All of the other settings of the algorithms that were employed in the feature extraction, FS, and classification steps were left at their default values (which are summarized in Table 3), where NP and max\_evals represent the population size and the maximum number of the function evaluations, respectively.

**Table 3**  
Summary of default parameter settings for used algorithms (alg.)  
along with package names that they have built-in (pkg.)

Pkg.	Alg.	Parameters
scikit-learn	SVM	$C=1.0$ , kernel=linear, degree=3, cache_size=200, random_state=123
	LR	penalty=l2, $C=1.0$ , solver=lbfqs, max_iter=1000, multi_class=auto
	NB	priors=None, var_smoothing=1e-09
	MNB	alpha=1.0, fit_prior=True, class_prior=None
	DT	criterion=gini, splitter=best, max_depth=None, random_state=123
	RF	n_estimators=100, criterion=gini, max_depth=None, random_state=123
	CHI	NA
	MI	discrete_features=auto, n_neighbors=3, random_state=123
evopreprocess	GA	NP=10, max_evals=1000, tournament_size=5, mutation_rate=0.25, crossover_rate=0.25, random_state=123
	FA	NP=10, max_evals=1000, alpha=1, beta_min=1, gamma=2, random_state=123
	PSO	NP=10, max_evals=1000, c1=2.0, c2=2.0, w=0.7, min_velocity=-1.5, max_velocity=1.5, random_state=123
	HS	NP=10, max_evals=1000, r_accept=0.7, r_pa=0.35, b_range=1.42, random_state=123
	DE	NP=10, max_evals=1000, differential_weight=1, crossover_probability=0.8, random_state=123
	CS	NP=10, max_evals=1000, pa=0.2, alpha=0.5, random_state=123
	BAT	NP=10, max_evals=1000, loudness=0.5, pulse_rate=0.5, min_frequency=0.0, max_frequency=2.0, random_state=123
	ABC	NP=10, max_evals=1000, limit=100, random_state=123
	SCA	NP=10, max_evals=1000, a=3, r_min=0, r_max=2, random_state=123
	GWO	NP=10, max_evals=1000, random_state=123

Notice that all of the experiments were conducted on a computer that featured an Intel(R) Core(TM) i7-4700HQ 2.4GHz CPU with 16GB of RAM running on Windows 10 (64-bit). The code<sup>4</sup> has also been made available on GitHub to ensure the reproducibility of the analysis. First of all, the best feature set, weighting scheme, and classifier were detected in order to make a comprehensive assessment. Using the best classifier and feature set, the results were then obtained for the filter and NI methods, respectively, as compared to the different weighting schemes. All of the experiments were conducted on two non-benchmark and two benchmark data sets – this was intended to exploit the behaviors of the NI algorithms on different data sets for FS.

### 5.1. Choosing best feature set and classifier

In the first steps of the experiments, different feature-extraction, term-weighting, and classification methods were employed on the four data sets. In the feature-extraction phase, bow and tri-gram features were extracted from the preprocessed data sets. The numbers of extracted features (with respect to the data sets and feature models) are given in Table 4. As can be seen in Table 4, the TTC data set had a greater number of features – even though it had fewer instances than the CB data set did (see Table 2). The tri-gram model also produced more features than the bow model did. After the feature extraction, the different weighting methods (i.e., tf, tf\*idf, and binary) were applied, and each data set was converted into a classification-ready structure. Finally, the classification experiments were performed using different classifiers. The main aim of this step was to detect the feature model, term-weighting scheme, and classifier in the respective scenarios that produced the greatest accuracies for the data sets.

**Table 4**

The number of extracted features considering feature model and data set

Data set	Feature model	
	BOW	Tri-gram (3G)
CB	1859	4222
TTC	4076	5827
R8	2274	3767
N8	3253	5549

As can be seen in Table 5, the best accuracies were often obtained with tf\*idf weighting for the CB and TTC data sets. The probability-based classifiers provided better results on the binary and tf-weighted data, while LR and SVM often worked better on the tf\*idf-weighted data. On the CB data set, the first- and second-best accuracies were obtained by the MNB and LR classifiers (**0.731** and 0.728, respectively), while respective accuracies of **0.825** and 0.820 were achieved with the help of the LR and MNB classifiers, respectively, on the TTC data set. As expected, MNB

<sup>4</sup><https://github.com/ocbn/NIFS>

worked well for data that can easily be converted into frequency values (such as word counts in text). Other than MNB, the results showed that the best-fitting classifier often turned out to be **LR** for both data sets. In terms of the feature model, the tri-gram features often provided better results than the bow features. As such, the **LR** classifier was used in the rest of the experiments on the **tf\*idf**-weighted CB and TTC data sets that were transformed into classification-ready structures by using the **tri-gram** features.

**Table 5**  
Accuracies of four data sets considering different feature models (FM), weighting schemes (W), and classifiers

FM <sub>W</sub>	CB						TTC					
	LR	SVM	NB	MNB	DT	RF	LR	SVM	NB	MNB	DT	RF
BOW <sub>tf</sub>	0.693	0.685	0.614	0.693	0.636	0.668	0.697	0.640	0.668	0.782	0.491	0.734
BOW <sub>tf*idf</sub>	0.695	0.701	0.598	0.701	0.621	0.657	0.802	0.814	0.657	0.780	0.468	0.694
BOW <sub>binary</sub>	0.703	0.687	0.624	0.722	0.658	0.671	0.740	0.688	0.660	0.777	0.482	0.720
3G <sub>tf</sub>	0.712	0.713	0.579	<b>0.731</b>	0.617	0.687	0.780	0.700	0.600	<b>0.820</b>	0.497	0.722
3G <sub>tf*idf</sub>	<b>0.728</b>	0.727	0.611	0.723	0.610	0.705	<b>0.825</b>	0.814	0.585	0.665	0.534	0.757
3G <sub>binary</sub>	0.718	0.700	0.585	0.726	0.602	0.698	0.811	0.774	0.628	0.802	0.445	0.711
FM <sub>W</sub>	R8						N8					
BOW <sub>tf</sub>	0.801	0.772	0.558	0.822	0.681	0.805	0.577	0.401	0.526	0.582	0.419	0.601
BOW <sub>tf*idf</sub>	0.828	<b>0.846</b>	0.548	0.823	0.678	0.795	<b>0.654</b>	0.632	0.498	0.648	0.411	0.587
BOW <sub>binary</sub>	0.804	0.780	0.560	0.780	0.664	0.786	0.573	0.402	0.494	0.642	0.434	0.587
3G <sub>tf</sub>	0.787	0.710	0.460	0.815	0.639	0.785	0.463	0.322	0.413	0.530	0.380	0.511
3G <sub>tf*idf</sub>	0.812	<b>0.834</b>	0.464	0.750	0.569	0.782	<b>0.603</b>	0.600	0.353	0.426	0.364	0.521
3G <sub>binary</sub>	0.791	0.739	0.436	0.744	0.572	0.755	0.511	0.421	0.355	0.561	0.383	0.516

Similarly, the best accuracies were often obtained with the **tf\*idf** weighting for both the N8 and R8 data sets (as can be seen in Table 5). On the R8 data set, the best accuracies were obtained with the help of SVM (**0.846** and 0.834). On the N8 data set, the best accuracies were obtained by the LR classifier (**0.654** and 0.603). For both of the data sets, the highest accuracies were obtained by using the bow features.

Table 6 presents a summary of these results; this shows that the best feature sets of the benchmark and non-benchmark data sets were bow and trigram, respectively. Other than MNB, the best-fitting classifier for both of the non-benchmark data sets was the LR. However, the respective best classifiers on the benchmark data sets were different and they were SVM and LR for R8 and N8 data sets respectively.

Based on these findings, our experiments were performed throughout the rest of this study with 5-fold stratified cross-validation on the **tf\*idf**-weighted trigram features and the LR classifier for the non-benchmark data sets. On the contrary, the experiments were performed with hold-out validation on the **tf\*idf**-weighted bow features on the benchmark data sets. SVM and LR were used for the R8 and N8 data sets, respectively.

**Table 6**

Summary of experimental setup and results obtained without FS in first step with respect to four data sets

Data set	Evaluation strategy	Best			
		classifier	feature model	weighting scheme	acc
CB	5-fold	MNB	tri-gram	tf	0.731
TTC	5-fold	LR	tri-gram	tf*idf	0.825
R8	hold-out	SVM	bow	tf*idf	0.846
N8	hold-out	LR	bow	tf*idf	0.654

On the CB and TTC data sets, the NI algorithms were configured to run for FS with outer stratified 5-fold cv and inner 2-fold cv, where four independent runs of the NI algorithm at hand were used at each fold. On the R8 and N8 data sets, the NI algorithms were configured to run with hold-out as an outer evaluation strategy and 2-fold cv as an inner evaluation strategy. Finally, the respective best classifier for each data set was used as both the inner and outer classifiers for evaluating the solutions.

## 5.2. Results of filter-based methods

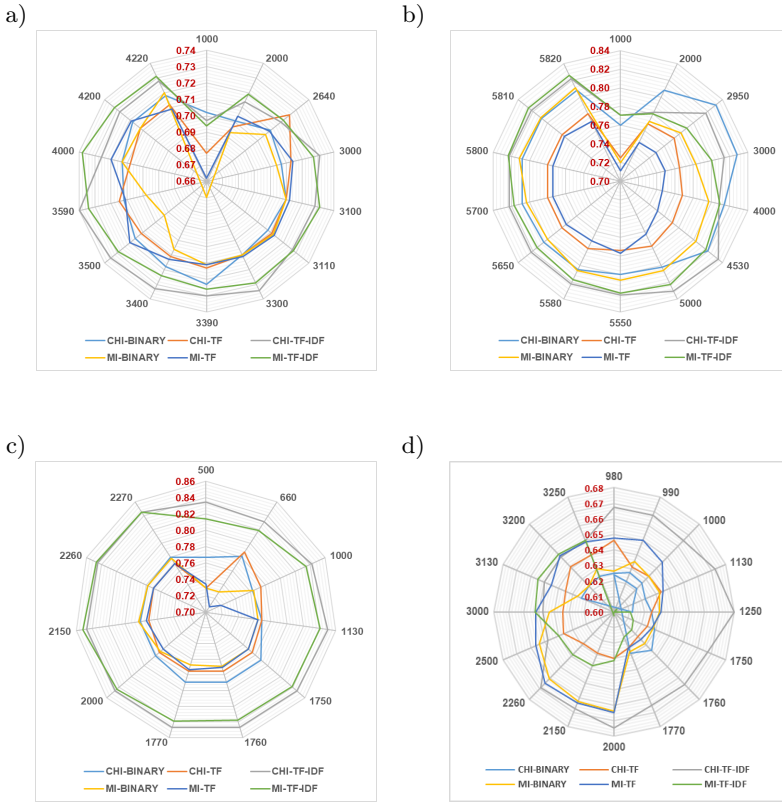
In this step, experiments were conducted to explore the effects of filter-based FS on the performance of the respective best classifier for each data set that was detected in the previous step. For this purpose, the CHI and MI methods were employed on the CB, TTC, R8, and N8 data sets, and their behaviors were compared with the weighting scheme as well as the  $k$  parameter that specified the number of features to be selected. The value of  $k$  increased by 10 at each step; the obtained results for the four data sets are depicted in Figure 2. Notice that not all of the results for parameter  $k$ 's different values are provided; this is meant to reduce the complexity and make the charts easy to follow.

As can be seen in Figure 2, CHI and MI featured similar behavior for the benchmark and non-benchmark data sets. Applying the CHI and MI methods on the binary and tf-weighted data often produced equal or lower results as compared to the results that were obtained without FS (see Table 6). However, these two methods improved the classification accuracy on the tf\*idf-weighted data at the end of the day. As can be seen in Figure 2a, the CHI and MI methods achieved their best accuracies on the CB data set (**0.740** and **0.737**, respectively) by selecting the most informative **3590** and **4000** tf\*idf-weighted features, respectively. As can be seen in Figure 2b, these methods achieved their best results for the TTC data set (**0.834** and **0.825**) when using the **4530** and **5810** tf\*idf-weighted features, respectively.

On the other hand, Figure 2c shows that the CHI and MI methods obtained their best results (**0.850** and **0.851**, respectively) by selecting the most informative **1130** and **2150** features on the tf\*idf-weighted R8 data set. Similarly, Figure 2d shows that the CHI and MI methods produced the best accuracies (**0.680** and **0.655**,



respectively) by selecting the most informative **1250** and **3130** features on the  $tf^*idf$ -weighted N8 data set.



**Figure 2.** Accuracies of respective best classifiers for reduced CB (a), TTC (b), R8 (c), and N8 (d) data sets considering different filter-based methods, numbers of selected features ( $k$ ), and weighting schemes

These results make it clear that the results of the MI and CHI methods were slightly different over the same data set and that these two methods often improved the classification accuracy or provided the same accuracy by using a lower number of features when compared to the classification experiments without FS. The CHI method often outperformed MI in terms of accuracy by selecting a lower number of features.

A summary of the results that had been obtained thus far without FS and with filter-based FS is provided in Table 7, which obviously shows that applying filter-based FS improved the classification accuracy – especially with the help of CHI over the  $tf^*idf$ -weighted data sets. For instance, the CHI method improved the best accuracies on the CB and TTC data sets (from **0.731** to **0.740**, and from **0.825** to **0.834**, respectively).

**Table 7**  
Comparative summary of experimental setup and results  
obtained with/without filter-based FS

Data set	Weighting scheme	Classifier	Feature model	Evaluation strategy	Best acc		
					Without FS (✗) (see Table 6)	Filter-based FS (✓)	
						MI	CHI
CB	tf*idf	LR	trigram	5-fold cv	0.731	0.737	0.740
TTC	tf*idf	LR	trigram	5-fold cv	0.825	0.825	0.834
R8	tf*idf	SVM	bow	hold-out	0.846	0.851	0.850
N8	tf*idf	LR	bow	hold-out	0.654	0.655	0.680

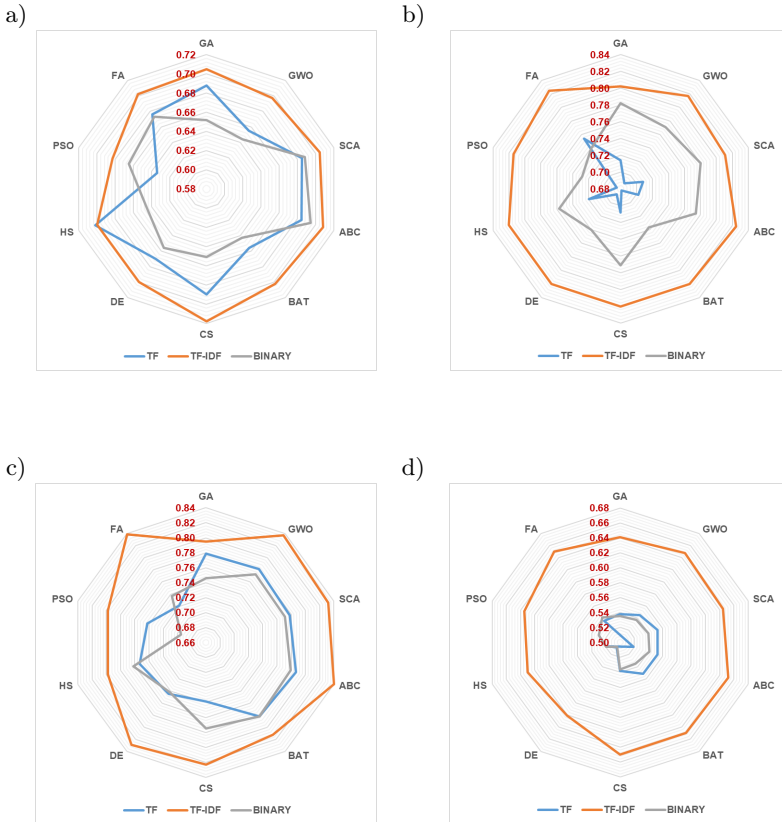
### 5.3. Results of NI methods

In this step, the respective best classifier for each of the data sets was again used to perform classification experiments on the reduced data sets. Different from the previous step, ten different NI algorithms (briefly introduced in 3.3.1) were employed on data sets that were weighted using the tf, tf\*idf, and binary schemes in order to reduce the feature space. First of all, the accuracies were obtained with respect to the NI methods on the four data sets. Then, the number of selected features by the NI methods were extracted, and their computation times were established. The following sub-headings provided the respective results of these experiments.

**Obtained accuracies:** Figure 3 depicts the accuracies that were obtained with the respective best classifiers for each of the data sets after being reduced by the different NI methods and weighted by using the tf-, tf\*idf-, and binary-weighting schemes. As can be seen in Figure 3, the NI methods produced better results with tf\*idf weighting for all of the data sets (which was similar to the filter-based methods). On the other hand, these methods often worked better with tf weighting as compared to binary weighting.

Considering the tf\*idf weighting on the CB data set (see Figure 3a), the three best accuracies were obtained (**0.718**, **0.708**, and **0.702**) with the help of the **CS**, **ABC**, and **FA** methods, respectively. The other methods often had similar behaviors and produced slightly different results; the worst accuracy was obtained by PSO (0.683). On the TTC data set (see Figure 3b), the best accuracy was obtained by both the **ABC** and **FA** algorithms (**0.825**), while BAT, **CS**, DE, and HS obtained an equal accuracy (0.820). On the other hand, the worst accuracy was obtained by the GA algorithm (0.802). On the R8 data set (see Figure 3c), the three best accuracies were obtained (**0.840**, **0.839**, and **0.837**) with the help of the **ABC**, **FA**, and **GWO** algorithms, respectively. The rest of the algorithms produced quite similar results; the worst accuracy was produced by the GA algorithm (0.795). Finally, the best accuracy was obtained on the N8 data set (**0.652**) by using the **ABC** algorithm (see Figure 3d), while BAT, **CS**, and **FA** have produced the same accuracy of 0.650 which

is the second-best result. The worst accuracy on this data set was obtained by the DE algorithm (0.621).



**Figure 3.** Accuracies of respective best classifiers on reduced CB (a), TTC (b), R8 (c), and N8 (d) data sets with respect to weighting schemes and NI methods

Considering the overall results, it seems that performances of the NI methods varied depending on the data set at hand; however, their behaviors did not tend to show major changes. Oftentimes, the most successful algorithms in terms of accuracy were ABC, FA, and CS; however, it was clear that applying the NI methods on FS did not improve the classification accuracy for all of the data sets. On the CB data set, the best accuracy decreased from **0.731** to **0.718**, while on the TTC data set, the best accuracy of **0.825** did not change. On the R8 data set, the best accuracy similarly decreased (from **0.846** to **0.840**), while on the N8 data set, the best accuracy decreased from **0.654** to **0.652**. Hence, the best accuracy values of the NI methods fell behind those of the best results of the filter-based methods.

**The average number of features and computation times:** The other two parameters for evaluating the NI methods were the number of selected features and the computation times. It was important to evaluate the numbers of selected features, as this had a direct effect on their costs and performances. On the other hand, the NI methods decided those features that were to be selected by using their inner evolution strategy. As such, it was not possible to compare their computation times by using a predefined constant number of features (as was the case in the filter-based methods). Even though the number of selected features provided insight into their costs for the outer classification task, it was required to measure the inner costs of the NI methods for the computation time.

In this step, therefore, the number of selected features was extracted for each of the NI methods that were employed on the four data sets (the average value was taken for the 5-fold evaluation strategy). In addition, the time that was required to complete both the classification and FS tasks was also measured for all of the employed NI methods. As in previous steps of the experiments, the  $tf*idf$ -weighted data sets were taken into consideration, as both the filter-based and NI methods produced better results on the  $tf*idf$ -weighted data sets. The number of selected features, as well as the computation times of the NI methods on the four data sets, are depicted in Figure 4.

In Figure 4, the charts on the left show the number of selected features, while the charts on the right side depict the computation times with respect to the NI methods for CB, TTC, R8, and N8, respectively. Figures 4a and 4c depict the obtained average numbers of features that were selected by the different NI methods during the process of the 5-fold cross-validation on the CB and TTC data sets, respectively. As can be seen in these figures, the methods had similar behaviors for the number of selected features on both of the data sets. However, SCA, CS, ABC, and HS often selected a higher number of features on average when compared to the others. Similarly, Figures 4e and 4g show the numbers of selected features in the process of the hold-out classification on the R8 and N8 data sets, respectively. As can be seen in these figures, the methods again exhibited similar behaviors, and ABC, SCA, and CS often selected a higher number of features. In the previous step of the experiments, it was found that the top three methods in terms of accuracy on the  $tf*idf$ -weighted data were FA, ABC, and CS. However, the most interesting finding was that FA selected a lower number of features on average when compared to the ABC and CS methods. This shows that FA was a bit more cost-effective (in terms of the number of selected features) for classification tasks (unless it did not have much more inner complexity).

Considering the computation times (see Figures 4b, 4d, 4f, and 4h), the NI methods again showed similar behaviors on the four data sets. CS, ABC, and SCA generally required less time to complete their computations as compared to the others. On the other hand, the PSO, DE, GWO, and HS methods often had more inner complexity, whereas FA featured a moderate computation time. Considering the results of the previous step of the experiments, the results showed that ABC, FA, and CS often

produced better results. However, ABC and CS selected a higher number of features with lower inner complexities, whereas FA selected a moderate number of features with greater inner complexity. As classification requires less time than the inner evaluation of the NI methods does, it seems that it is a better choice to select ABC and CS for text FS.

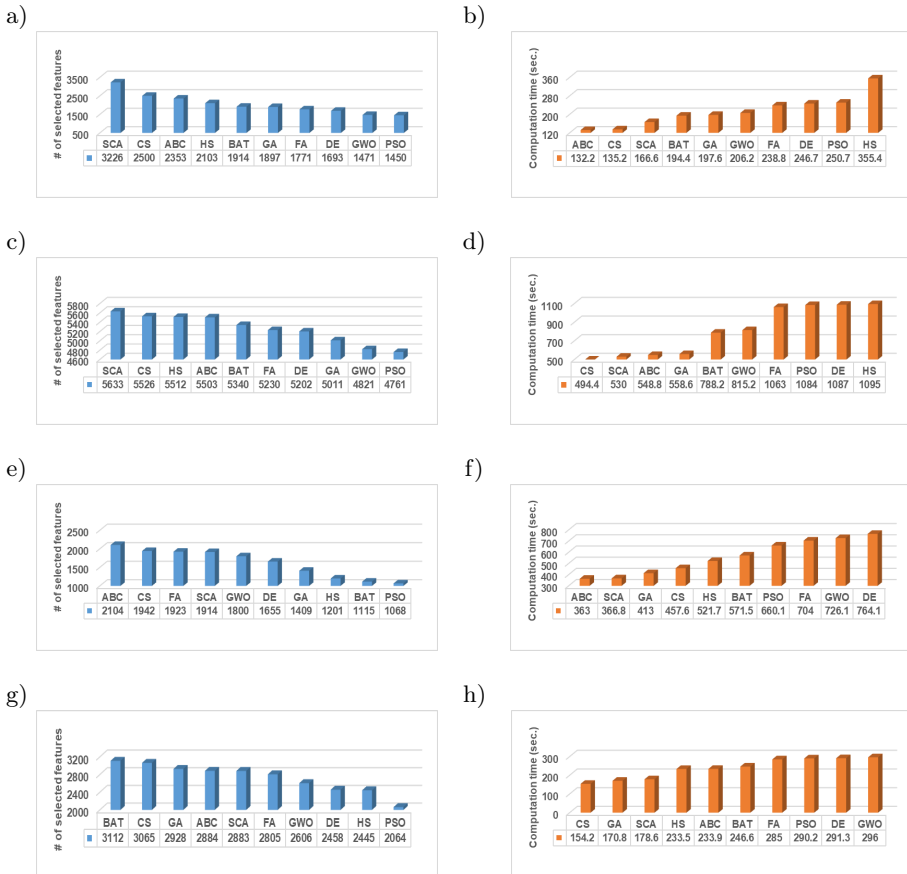


Figure 4. Numbers of selected features (left) and computation times (right) for NI methods on CB (a, b), TTC (c, d), R8 (e, f), and N8 (g, h) data sets

### 5.4. Overall comparison

During this step, an overall comparison was performed in order to provide clearer insight into the NI methods that were employed for text FS. For this purpose, the results that were obtained in the previous steps have been put together in Table 8 considering different parameters such as the accuracy, inner computation time (FST – i.e., feature selection), outer computation time (CT – i.e., classification), and the number

of selected features. As can be seen in Table 8, the filter-based methods provided better results than the NI methods did for FS. Specifically, the CHI method had a very low inner complexity and was also superior to its peer MI with respect to all of the parameters. Unfortunately, the NI methods had a very high inner complexity and fell below that of the filter-based methods considering the accuracy.

**Table 8**

Comparison of filter-based methods with three best-performing NI algorithms with respect to accuracy (Acc), # of features, feature-selection time (FST), and outer-classification time (CT)

5-fold average	CB						TTC					
	FS (✗)	FS (✓)					FS (✗)	FS (✓)				
		CHI	MI	ABC	CS	FA		CHI	MI	ABC	CS	FA
Acc	0.728	0.740	0.737	0.708	0.718	0.702	0.825	0.834	0.825	0.825	0.820	0.825
# of features	4222	3590	4000	2353	2500	1771	5827	4530	5810	5503	5526	5230
FST [s]	NA	0.036	42.75	132.2	135.2	238.8	NA	0.021	36.46	548.8	494.4	1062.6
CT [s]	0.155	0.098	0.131	0.065	0.074	0.040	0.328	0.249	0.326	0.226	0.217	0.207
Hold-out value	R8						N8					
Acc	0.846	0.850	0.851	0.840	0.823	0.839	0.654	0.680	0.655	0.652	0.650	0.650
# of features	2274	1130	2150	2104	1942	1923	3253	1250	3130	2884	3065	2805
FST [s]	NA	0.062	65.23	363.0	457.6	704.0	NA	0.109	151.3	233.8	154.1	285.0
CT [s]	4.484	1.999	4.390	2.232	1.995	1.947	0.890	0.484	1.061	0.306	0.396	0.264

## 6. Discussion and conclusion

This paper provides a comprehensive assessment of basic NI algorithms for text FS on both benchmark and non-benchmark data sets. Unique among its peers in the literature, the novelty of this study lies in the consideration of ten major representatives of basic NI algorithms along with the exploration of the effect of the term-weighting process on their performances.

Based on the results, it can be concluded first and foremost that filter-based methods provide better results and have very low inner complexity when compared to basic NI ones. On the other hand, NI methods provide slightly different results than filter-based methods and often select a lower number of features – especially when the filter-based method is MI. Even though the performances of the NI methods vary depending on the data set at hand, they provide slightly different results. It seems that FA, CS, and ABC are often better choices as compared to other NI ones.

Applying both filter-based and NI methods on  $tf^*idf$ -weighted data provided better results in all cases. This was because weighting has a direct effect on both the inner and outer evaluation phases for the NI methods. It is also important to keep in mind that this study used major and basic NI methods (without any extension, modification, or parameter tuning) along with their default parameter settings. How-

ever, changing their algorithm-specific parameters along with other ones (such as the number of folds for inner evaluation, population size, and the maximum number of function evaluations) can take the performances of NI methods one step further.

As such, it has also been concluded that basic NI methods can be employed for text FS by hybridizing or improving the existing ones. However, their high inner complexity (stemming from the higher number of function evaluations) is the most important challenge for employing them – especially on high-dimensional text data sets.

## Acknowledgements

*The author would like to thank reviewers and editors for their valuable suggestions, which contributed greatly to the improvement of this paper.*

## References

- [1] Aghdam M.H., Ghasem-Aghaee N., Basiri M.E.: Text feature selection using ant colony optimization, *Expert systems with applications*, vol. 36(3), pp. 6843–6853, 2009.
- [2] Aghdam M.H., Ghasem-Aghaee N., Basiri M.E.: Application of ant colony optimization for feature selection in text categorization. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 2867–2873, IEEE, 2008.
- [3] Al-Tashi Q., Rais H.M., Abdulkadir S.J., Mirjalili S., Alhussian H.: A review of grey wolf optimizer-based feature selection methods for classification, *Evolutionary Machine Learning Techniques*, pp. 273–286, 2020.
- [4] Basiri M.E., Nemati S.: A novel hybrid ACO-GA algorithm for text feature selection. In: *2009 IEEE Congress on Evolutionary Computation*, pp. 2561–2568, IEEE, 2009.
- [5] Belazzoug M., Touahria M., Nouioua F., Brahimi M.: An improved sine cosine algorithm to select features for text categorization, *Journal of King Saud University – Computer and Information Sciences*, vol. 32(4), pp. 454–464, 2020.
- [6] Chantar H., Mafarja M., Alsawalqah H., Heidari A.A., Aljarah I., Faris H.: Feature selection using binary grey wolf optimizer with elite-based crossover for Arabic text classification, *Neural Computing and Applications*, vol. 32(16), pp. 12201–12220, 2020.
- [7] Chen H., Hou Q., Han L., Hu Z., Ye Z., Zeng J., Yuan J.: Distributed Text Feature Selection Based On Bat Algorithm Optimization. In: *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, vol. 1, pp. 75–80, IEEE, 2019.
- [8] Çoban Ö., Özyer B., Özyer G.T.: Sentiment analysis for Turkish Twitter feeds. In: *2015 23rd Signal Processing and Communications Applications Conference (SIU)*, pp. 2388–2391, IEEE, 2015.

- [9] Diao R., Shen Q.: Nature inspired feature selection meta-heuristics, *Artificial Intelligence Review*, vol. 44(3), pp. 311–340, 2015.
- [10] Faris H., Aljarah I., Al-Betar M.A., Mirjalili S.: Grey wolf optimizer: a review of recent variants and applications, *Neural Computing and Applications*, vol. 30(2), pp. 413–435, 2018.
- [11] Gao W., Liu S.: Improved artificial bee colony algorithm for global optimization, *Information Processing Letters*, vol. 111(17), pp. 871–882, 2011.
- [12] Garg S., Verma S.: A Comparative Study of Evolutionary Methods for Feature Selection in Sentiment Analysis. In: *IJCCI*, pp. 131–138, 2019.
- [13] Geem Z.W., Kim J.H., Loganathan G.V.: A new heuristic optimization algorithm: harmony search, *Simulation*, vol. 76(2), pp. 60–68, 2001.
- [14] Holland J.H.: *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, MI, 1975.
- [15] Inbarani H.H., Bagyamathi M., Azar A.T.: A novel hybrid feature selection method based on rough set and improved harmony search, *Neural Computing and Applications*, vol. 26(8), pp. 1859–1880, 2015.
- [16] Karaboga D., Basturk B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, *Journal of Global Optimization*, vol. 39(3), pp. 459–471, 2007.
- [17] Karakatič S.: EvoPreprocess-Data Preprocessing Framework with Nature-Inspired Optimization Algorithms, *Mathematics*, vol. 8(6), pp. 1–29, 2020.
- [18] Kennedy J., Eberhart R.: Particle swarm optimization. In: *Proceedings of ICNN'95 – International Conference on Neural Networks*, vol. 4, pp. 1942–1948, IEEE, 1995.
- [19] Khurana A., Verma O.P.: A Fine Tuned Model of Grasshopper Optimization Algorithm with Classifiers for Optimal Text Classification. In: *2020 IEEE 17th India Council International Conference (INDICON)*, pp. 1–7, IEEE, 2020.
- [20] Kohavi R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: *International Joint Conference on AI*, pp. 1137–1145, Montreal, Canada, 1995.
- [21] Kyaw K.S., Limsiroratana S.: Traditional and Swarm Intelligent Based Text Feature Selection for Document Classification. In: *2019 19th International Symposium on Communications and Information Technologies (ISCIT)*, pp. 226–231, IEEE, 2019.
- [22] Labani M., Moradi P., Jalili M.: A multi-objective genetic algorithm for text feature selection using the relative discriminative criterion, *Expert Systems with Applications*, vol. 149, pp. 1–21, 2020.
- [23] Langeron C., Moulin C., Géry M.: Entropy based feature selection for text categorization. In: *Proceedings of the 2011 ACM symposium on applied computing*, pp. 924–928, 2011.



- [24] Mafarja M., Qasem A., Heidari A.A., Aljarah I., Faris H., Mirjalili S.: Efficient hybrid nature-inspired binary optimizers for feature selection, *Cognitive Computation*, vol. 12(1), pp. 150–175, 2020.
- [25] Mirjalili S.: SCA: a sine cosine algorithm for solving optimization problems, *Knowledge-Based Systems*, vol. 96, pp. 120–133, 2016.
- [26] Mirjalili S., Mirjalili S.M., Lewis A.: Grey wolf optimizer, *Advances in Engineering Software*, vol. 69, pp. 46–61, 2014.
- [27] Özel S.A., Saraç E., Akdemir S., Aksu H.: Detection of cyberbullying on social media messages in Turkish. In: *2017 International Conference on Computer Science and Engineering (UBMK)*, pp. 366–370, IEEE, 2017.
- [28] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M., Duchesnay E.: Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [29] Porter M.F.: An algorithm for suffix stripping, *Program: Electronic Library and Information Systems*, vol. 14(3), pp. 130–137, 1980.
- [30] Purushothaman R., Rajagopalan S., Dhandapani G.: Hybridizing Gray Wolf Optimization (GWO) with Grasshopper Optimization Algorithm (GOA) for text feature selection and clustering, *Applied Soft Computing*, vol. 96, pp. 1–14, 2020.
- [31] Rawashdeh G., Mamat R., Bakar Z.B.A., Abd Rahim N.H.: Comparative between optimization feature selection by using classifiers algorithms on spam email, *International Journal of Electrical and Computer Engineering*, vol. 9(6), pp. 5479–5485, 2019.
- [32] Sharma M., Kaur P.: A Comprehensive Analysis of Nature-Inspired Meta-Heuristic Techniques for Feature Selection Problem, *Archives of Computational Methods in Engineering*, pp. 1103–1127, 2021.
- [33] Shrivastava P., Shukla A., Vepakomma P., Bhansali N., Verma K.: A survey of nature-inspired algorithms for feature selection to identify Parkinson’s disease, *Computer Methods and Programs in Biomedicine*, vol. 139, pp. 171–179, 2017.
- [34] Storn R., Price K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization*, vol. 11(4), pp. 341–359, 1997.
- [35] Wang Y., Liu Y., Feng L., Zhu X.: Novel feature selection method based on harmony search for email classification, *Knowledge-Based Systems*, vol. 73, pp. 311–323, 2015.
- [36] Yang X.S.: Firefly algorithm, stochastic test functions and design optimisation, *International Journal of Bio-Inspired Computation*, vol. 2(2), pp. 78–84, 2010.
- [37] Yang X.S.: *Nature-inspired algorithms and applied optimization*, vol. 744, Springer, 2017.
- [38] Yang X.S.: *Nature-inspired optimization algorithms*, Academic Press, 2020.
- [39] Yang X.S., Deb S.: Cuckoo search via Lévy flights. In: *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pp. 210–214, Ieee, 2009.

- [40] Yang X.S., Deb S.: Engineering optimisation by cuckoo search, *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 1(4), pp. 330–343, 2010.
- [41] Yang X.S., Gandomi A.H.: Bat algorithm: a novel approach for global engineering optimization, *Engineering Computations*, 2012.
- [42] Yang Y., Pedersen J.O.: A Comparative Study on Feature Selection in Text Categorization. In: *ICML'97: Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 412–420, Nashville, TN, USA, 1997.
- [43] Yildirim S., Yildiz T.: A Comparison of Different Approaches to Document Representation in Turkish Language, *Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, vol. 22(2), pp. 569–576, 2018.

## Affiliations

Önder Çoban 

Adiyaman University, Department of Computer Engineering, Adiyaman, Turkey;  
ocoban@adiyaman.edu.tr, ORCID ID: <https://orcid.org/0000-0001-9404-2583>

**Received:** 14.04.2021

**Revised:** 09.11.2021

**Accepted:** 09.11.2021