Research Article

How optimizing perplexity can affect the dimensionality reduction on word embeddings visualization?

Gustavo H. de Rosa[1] · José R. F. Brega[1] · João P. Papa[1]

Received: 5 September 2019 / Accepted: 13 November 2019 / Published online: 25 November 2019

Abstract
Traditional word embeddings approaches, such as bag-of-words models, tackles the problem of text data representation by linking words in a document to a binary vector, marking their occurrence or not. Additionally, a term frequency-inverse document frequency encoding provides a numerical statistic reflecting how important a particular word is in a document. Nevertheless, the major vulnerability of such models concerns with the loss of contextual meaning, which inhibits them from learning proper pieces of information. A new neural-based embedding approach, known as Word2Vec, tries to mitigate that issue by minimizing the loss of predicting a vector from a particular word considering its surrounding words. Furthermore, as these embedding-based methods produce low-dimensional data, it is impossible to visualize them accurately. With that in mind, dimensionality reduction techniques, such as t-SNE, presents a method to generate bi-dimensional data, allowing its visualization. One common problem of such reductions concerns with the setting of their hyperparameters, such as the perplexity parameter. Therefore, this paper addresses the problem of selecting a suitable perplexity through a meta-heuristic optimization process. Meta-heuristic-driven techniques, such as Artificial Bee Colony, Bat Algorithm, Genetic Programming, and Particle Swarm Optimization, are employed to find proper values for the perplexity parameter. The results revealed that optimizing t-SNE's perplexity is suitable for improving data visualization and thus, an exciting field to be fostered.

Keywords Word embeddings · Dimensionality reduction · Meta-heuristic optimization

1 Introduction

Humans are capable of mastering communication techniques and creating several tools to express themselves, such as language and speech. Nevertheless, these tools are non-trivial when considering computer-based individuals, leaving a huge blank on how to produce a "natural" perception of the real world. An arising subarea under artificial intelligence, known as natural language processing (NLP), tries to mitigate that issue by fostering machine learning research and creating meaningful knowledge regarding natural language understanding. The NLP area has been widely researched throughout the last years, establishing various hallmarks in an extensive range of applications, such as language modeling [24], word embeddings [7], sentiment classification [8], among others.

A common problem when working with natural language copes with the fact that text data are discrete structures, being infeasible to work with modern machine learning algorithms, e.g., recurrent neural networks. Traditional approaches, known as bag-of-words models, tackles this problem by encoding the text data into binary vectors (one-hot encoding), where each vector's position represents the existence or not of a particular word. Although this seems to be a fair representation, there is vital contextual information lost in the process. Moreover, newer

✉ Gustavo H. de Rosa, gustavo.rosa@unesp.br; José R. F. Brega, remo.brega@unesp.br; João P. Papa, joao.papa@unesp.br | [1]Department of Computing, São Paulo State University, Av. Eng. Luiz Edmundo Carrijo Coube, 14-01, Bauru, Brazil.

SN Applied Sciences (2019) 1:1673 | https://doi.org/10.1007/s42452-019-1689-4

bag-of-words models, such as count vectorizer and term frequency-inverse document frequency, encode statistical information of how relevant a particular word is in a document. Nonetheless, these models produce high-dimensional sparse feature vectors, being impracticable when collated with large vocabularies.

A contextual information-based text representation, known as word embeddings,[1] tackles the problem by learning a distributed representation for words. In other words, this approach is capable of encoding semantic relationships among words and producing a vector space model (VSM), where closer vectors have a higher degree of similarity than distant vectors. Essentially, one can divide word embeddings into three distinct categories:

- Topics models: one of the most influential was Latent Semantic Analysis (LSA) [4], which was fostered in the context of information retrieval and latter enhanced as Latent Dirichlet Allocation (LDA) [2];
- Neural language models: based on neural networks, such as Convolutional Neural Networks (CNN) [17], Recurrent Neural Networks (RNN) [5] and Autoencoders [19];
- Distributional semantic models: often based on word co-occurrences representations, e.g., Hyperspace Analogue to Language (HAL) [20], Random Indexing [12] and BEAGLE [11].

The main difference between these models lies in the type of contextual information they use. Topic models use documents as their contexts, while neural language and distributional semantic models use words as contexts. Additionally, document-based models capture semantic relatedness, while word-based models capture semantic similarity.

A recent neural-based successful word embedding model, known as Word2Vec [22], uses shallow neural networks to minimize the loss of predicting a vector from a particular word considering its surrounding words, producing low-dimensional data. Notwithstanding, high or low dimensional data prevent humans from visualizing them, inhibiting important decision or insights making. With that in mind, it is possible to employ dimensionality reduction techniques, such as Principal Component Analysis (PCA) [10] or t-Distributed Stochastic Neighbour Embedding (t-SNE) [21], to generate bi-dimensional spaces and allow their visualization. One common problem concerning the t-SNE dimensionality reduction technique is the setting of its hyperparameters, such as the perplexity. To the best of the authors' knowledge, there is only one work that tries to set this parameter automatically [3].

In this paper, the problem of fine-tuning the perplexity parameter in t-SNE is modeled as a meta-heuristic-driven optimization task, in which agents encode the values of the perplexity in a search problem guided by the Kullback–Leibler divergence over the datasets. As far as we are concerned, this is the first work that attempted to address the problem of fine-tuning the perplexity parameter in t-SNE by meta-heuristic techniques. In order to validate the proposed approach, we employed Artificial Bee Colony (ABC) [13], Bat Algorithm (BA) [27], and Particle Swarm Optimization (PSO) [14], as we opted to use only swarm-based algorithms. Nevertheless, this might be not interesting because other meta-heuristics' taxonomies may perform differently in our proposed approach. Thus, we opted also to employ an evolutionary-based algorithm known as the Genetic Programming (GP) [15]. Finally, the main contributions of this paper are twofold: (1) to introduce meta-heuristic techniques to the context of fine-tuning perplexity in t-SNE, and (2) to fill the lack of research regarding dimensionality reduction hyperparameters optimization.

The remainder of this paper is organized as follows. Sections 2, 3 and 4 present some theoretical background concerning word embeddings, dimensionality reduction, and meta-heuristic techniques, respectively, while Sect. 5 discusses the methodology employed in this work. Section 6 presents the experimental results and Sect. 7 states conclusions and future works.

2 Word embeddings

The necessity of interpreting text data into continuous representations fostered the research on word embeddings. Essentially, an embedding is a numerical encoding of a text, where words or phrases from a particular vocabulary are mapped to vectors of real numbers. In this section, we present an overview regarding a neural-based word embedding, known as Word2Vec.

2.1 Word2Vec

Word2Vec [22] is a neural-based word embedding, represented by a shallow two-layer neural network, which is trained to reconstruct linguistic concepts of input words. Mainly, Word2Vec takes a corpus of text and produces a vector space model, typically composed of several hundred dimensions, where each unique word in the corpus is assigned to a corresponding vector in this space. Additionally, semantically similar words are mapped to nearby points in the vector space.

[1] The computational linguistics area prefers this term as distributional semantic models.

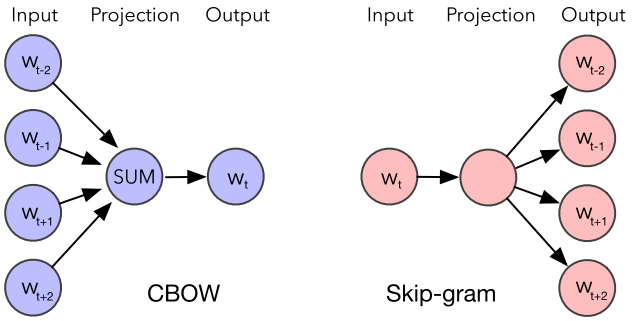

Fig. 1 Word2Vec architectures: CBOW and Skip-gram

Two distinct architectures are used to produce a continuous-based representation of words: continuous bag-of-words (CBOW) or continuous Skip-gram. Considering the CBOW architecture, the model predicts a current word based on its surrounding context words, while in the Skip-gram architecture, the model weights nearby context words more than distant context words. Figure 1 illustrates the CBOW and Skip-gram architectures.

3 Dimensionality reduction

In a machine learning problem, the factors that define the characteristics of the data are usually called features. Even with modern visualization techniques, it is still impossible for humans to visualize over three-dimensional spaces, i.e., four features or more. With that in mind, one can perceive that it gets harder to visualize data as the features' size grows. Additionally, some features correlate between themselves, being redundant.

The dimensionality reduction procedure arises in an attempt to mitigate the issue mentioned above. It is a process of reducing the number of features by obtaining new sets of principal features and is divided into two categories:

- Feature selection: tries to find a subset of the original set of features, composing a new and smaller set used to model the problem, e.g., filter-, wrapper-, and embedded-based approaches;
- Feature extraction: reduces the data from a high-dimensional to a low-dimensional space, e.g., principal component analysis, linear discriminant analysis (LDA), and t-distributed stochastic neighbour embedding.

3.1 t-Distributed stochastic neighbour embedding

The t-Distributed Stochastic Neighbour Embedding [21] is a nonlinear dimensionality reduction technique used to embed high-dimensional data into low-dimensional spaces, easing its visualization. It models each high-dimensional sample in an n-dimensional point where nearby points model similar samples and distant points model distinct samples.

Firstly, the t-SNE algorithm constructs a probability distribution over the high-dimensional samples, where similar samples have a high probability of being picked, while divergent samples have a low probability of being chosen. Secondly, it defines a similar probability distribution in a low-dimensional space and minimizes the Kullback–Leibler (KL) divergence between these two distributions. As the algorithm approximates these distributions, the KL divergence gets lower, and hence, makes the dimensionality reduction better.

Let $\boldsymbol{x} \in \mathfrak{R}^n$ be high-dimensional points, such that the similarity of x_j to x_i is the conditional probability $p(j|i)$, defined as follows:

$$p(j|i) = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_j||^2/2\sigma_i^2)}. \quad (1)$$

Additionally, let $\boldsymbol{y} \in \mathfrak{R}^n$ be low-dimensional points, such that the similarity of y_j to y_i is the conditional probability $q(j|i)$, defined as follows:

$$q(j|i) = \frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq i} exp(-||y_i - y_j||^2)}. \quad (2)$$

If the similarity between $\boldsymbol{x}$ and $\boldsymbol{y}$ is modeled correctly, the conditional probabilities $p(j|i)$ and $q(j|i)$ will be equal. Therefore, the Kullback–Leibler divergence is used to measure the faithfulness in which $q(j|i)$ models $p(j|i)$, such as follows:

$$KL(P_i|Q_i) = \sum_i \sum_j p(j|i) log\left(\frac{p(j|i)}{q(j|i)}\right), \quad (3)$$

where P_i represents the conditional probability distribution over all other samples given sample x_i, and Q_i represents the conditional probability distribution over all other samples given sample y_i.

Additionally, one of most important parts of the algorithm is the definition of σ_i, which defines the local scale

around x_i. Its value is not defined by hand, but rather found by a binary search, denominated Perplexity (γ). The perplexity of P_i is defined as:

$$\gamma(P_i) = 2^{H(P_i)}, \tag{4}$$

where

$$H(P_i) = -\sum_j p(j|i) log_2 p(j|i). \tag{5}$$

The perplexity is a smooth measure of the effective number of neighbors, being a valuable hyperparameter to be adjusted according to the problem itself.

4 Meta-heuristic optimization

Traditional optimization methods [1], such as the iterative methods, e.g., Newton method, Quasi-Newton method, Gradient Descent, Interpolation methods, use the evaluation of gradients and Hessians, being unfeasible to several applications due to their computational burden. Recently, an exciting proposition denoted as meta-heuristic has been employed to solve several optimization problems. A meta-heuristic technique consists of a high-level procedure, projected to generate or select a heuristic, which provides a sufficiently feasible solution to the optimization problem. Moreover, a meta-heuristic is a procedure that combines the concepts of *exploration*, used to perform searches throughout the search space, and *exploitation*, which is used to refine a promising solution based on its neighborhood.

Essentially, meta-heuristic techniques [26] are strategies that guide the process of searching for quasi-optimum solutions. They are mostly constituted of simple local searches and complex learning procedures, usually inspired by biological behaviors. Additionally, they are non-domain specific and have mechanisms to avoid being trapped in local optima points. Furthermore, one can classify them according to their taxonomy, as stated by Fister Jr. et al. [6], i.e., swarm intelligence, bio-inspired, physics- and chemistry-based, and evolutionary ones. In this work, we opted to use three swarm-based algorithms, as well as an evolutionary-based one.

4.1 Artificial bee colony

Artificial Bee Colony is a nature-inspired algorithm based on honey bee swarms, which is composed of three distinct groups of bees: employees, onlookers, and scouts. Each group has particular importance and function to the swarm, such as choosing a food source, going to the food source, and randomly searching food in new areas [13]. Additionally, the whole bee colony is split in half into employees[2] and onlookers bees. Moreover, when the employee bee exhausts its food source, it becomes a scout bee.

Let $\mathcal{S} = (\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_M)$ be a set of food sources such that $\boldsymbol{s}_i \in \Re^N$ stands for the position of food source *i*. Also, let $\mathcal{T} = (t_1, t_2, \ldots, t_M)$ be the number of cycles for each food source, known as the "food source trials", which is regulated by the `n_trials` parameter. After exploring a food source or discovering a newer one, bees share their discovered information about the nectar (food). Hence, an onlooker bee chooses a nectar source based on a probability associated with its achieved fitness, as formulated below:

$$p_i = \frac{F_i}{\sum_{k=1}^{M} F_k}, \tag{6}$$

where F_i is the fitness value of food source *i*.

Finally, one can use Eq. 7 to formulate a new food source position, as follows:

$$\boldsymbol{s}_i = \boldsymbol{s}_i + \phi(\boldsymbol{s}_i - \boldsymbol{s}_k), \tag{7}$$

where $i \neq k$ and $\phi \in [-1, 1]$ denotes a random value that controls the bee visualization of other food sources. Algorithm 1 depicts the pseudo-code of ABC.

[2] An employee bee is only responsible for a single food source.

Algorithm 1: Artificial Bee Colony pseudo-code.

Input: Number of food sources M, number of decision variables N, maximum iterations T and number of trials n_{trials}.
Output: $\mathbf{g}$ decision variables vector holding the most suitable values for the task's fitness function.

```
Fitness function f(s), s = (s_1, ..., s_N)^T;
Initialize food sources s_i, i = (1, 2, ..., M);
Initialize food sources trials t_i, i = (1, 2, ..., M);
t ← 1;
while (t < T) do
    for i from 1 to M do
        Choose a food source i to update its position according to Equation 6;
        if (t_i > n_trials) then
            Discover a new uniform random location for food source i;
        else
            Update food source i position according to Equation 7;
        if (f(s_i) < f(s*)) then
            Accept new solution s_i;
        Rank the food sources and find the current best s*;
    t ← t + 1;
g ← s*;
```

4.2 Bat algorithm

Bat Algorithm is a biological-inspired algorithm proposed by Yang et al. [27] primarily used for solving engineering optimization tasks. It takes into account the advanced capability of the bats' echolocation system, where they have a sonar-like mechanism that enables them to detect food, avoid obstacles, and communicate among themselves.

Mathematically speaking, let $\mathcal{B} = (b_1, b_2, \ldots, b_M)$ be a set of bats that compose the swarm, such that $b_i = (\boldsymbol{w}_i, \boldsymbol{z}_i)$, where $\boldsymbol{w}_i \in \Re^N$ and $\boldsymbol{z}_i \in \Re^N$ stand for the position and velocity of bat i, respectively. Additionally, each bat is associated with a frequency value of $f \in [f_{min}, f_{max}]$, a loudness value of A and a pulse rate of r. Each bat is initialized with random values for its position, velocity, and frequency. During each iteration, Eqs. 8, 9 and 10 are responsible for updating their frequency, velocity and position values, respectively:

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{8}$$

and

$$\boldsymbol{z}_i^{t+1} = \boldsymbol{z}_i^t + (\boldsymbol{w}_i^t - \hat{\boldsymbol{w}})f_i, \tag{9}$$

and

$$\boldsymbol{w}_i^{t+1} = \boldsymbol{w}_i^t + \boldsymbol{z}_i^{t+1}, \tag{10}$$

where β is a uniform random number between [0, 1], and $\hat{\boldsymbol{w}}$ denotes the current best global position (solution).

Furthermore, if a bat's pulse rate is smaller than a sampled probability p, e.g., $r_i < p$, a new solution is generated around the current best solution. Equation 11 formulates this procedure, as follows:

$$\boldsymbol{w}_i^{t+1} = \hat{\boldsymbol{w}} + \epsilon\bar{A}, \tag{11}$$

where ϵ is a random value between $[-1, 1]$, and $\bar{A}$ is the mean loudness of all bats in the swarm. Algorithm 2 presents the pseudo-code of BA.

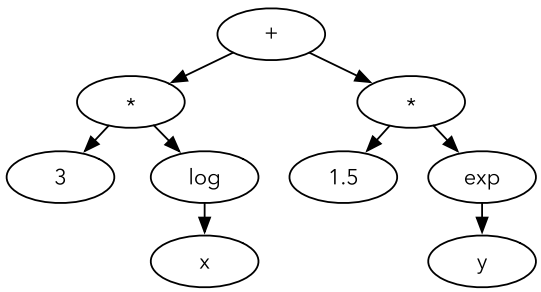

Fig. 2 A GP solution representing the expression $3log(x) + 5exp(y)$

Algorithm 2: Bat Algorithm pseudo-code.

Input: Number of bats M, number of decision variables N, maximum iterations T, minimum frequency f_{min}, maximum frequency f_{max}, loudness A and pulse rate r.

Output: $\mathbf{g}$ decision variables vector holding the most suitable values for the task's fitness function.

```
Fitness function f(w), w = (w_1, ..., w_N)^T;
Initialize bats population b_i, i = (1, 2, ..., M), and their positions w_i and velocities z_i;
Define the frequency f_i for each b_i;
t ← 1;
while (t < T) do
    for i from 1 to M do
        Generate new solution w_i from the frequency adjustment and velocity and position updates, according to Equations 8, 9 and 10;
        rand ← U(0, 1);
        if (rand > r) then
            Select the best solution w*;
            Generate a local solution w_i around the best selected solution using Equation 11;
        if (rand < A & f(w_i) < f(w*)) then
            Accept new solution w_i;
        Rank the bats and find the current best w*;
    t ← t + 1;
g ← w*;
```

4.3 Genetic programming

Genetic Programming [15] is an evolutionary-based metaheuristic that has arisen from the principles of Darwin's Theory of Evolution. Essentially, GP automatically creates expressions (solutions) that might solve a particular task and perform genetical operators in order to achieve more proper solutions.

Even though GP looks like the standard Genetic Algorithm (GA), there is a significant difference between them. Instead of using a binary "chromosome' holding the genes, GP uses a tree-based structure composed of terminal and function nodes, which is illustrated by Fig. 2. The terminal nodes represent constant values, while the function nodes are mathematical operators applied over the terminal nodes. During each generation, several operators are employed to improve the current population, such as (1) selection, (2) reproduction, (3) mutation, and (4) crossover.

Initially, the best individuals across the population are selected and reproduced to guarantee the best ones over the generations. Moreover, mutation and crossover procedures attempt to provide a variability factor in the population, i.e., mutation changes an individual gene arbitrarily, while the crossover switch branches between two trees. Furthermore, after the stopping criterion is satisfied, the best solution (individual) is recovered from the tree. Algorithm 3 presents the pseudo-code of GP.

Algorithm 3: Genetic Programming pseudo-code.

Input: Number of trees M, number of decision variables N, maximum iterations T, probability of reproduction p_r, probability of mutation p_m and probability of crossover p_c.

Output: $\mathbf{g}$ decision variables vector holding the most suitable values for the task's fitness function.

```
Fitness function f(x), x = (x_1, ..., x_N)^T;
Initialize trees population t_i, i = (1, 2, ..., M);
t ← 1;
while (t < T) do
    Choose p_r * M trees across the population and save them for next generation;
    Choose p_m * M trees and mutate their branches;
    Choose p_c * M pairs of trees and switch their branches;
    Evaluates the population by traversing the trees ;
    x* ← Best tree's solution;
    t ← t + 1;
g ← x*;
```

4.4 Particle swarm optimization

Particle Swarm Optimization is a swarm intelligence algorithm inspired by social behavior dynamics [14]. The idea behind employing social behavior learning is to allow each possible solution to move onto the search space, combining details from its previous and current locations with the ones provided by other swarm particles. One can understand this process as a simulation of the social interaction of birds looking for food or even humans trying to achieve a common objective.

Let $\mathcal{D} = (d_1, d_2, \ldots, d_M)$ be a set of particles that compose the swarm, such that $d_i = (\boldsymbol{\psi}_i, \boldsymbol{\rho}_i)$, where $\boldsymbol{\psi}_i \in \Re^N$ and $\boldsymbol{\rho}_i \in \Re^N$ stand for the position and velocity of particle i, respectively. Also, for each particle, we know its best local solution $\hat{\boldsymbol{\psi}}$, as well as the best solution (global) of the entire swarm $\boldsymbol{g}$. Each particle is initialized with random values for both velocity and position. Hence, each individual is evaluated with respect to a given fitness function, thus having its local minimum updated. At the end, the global minimum for each decision variable is updated with the value of the particle that achieved the best position in the swarm. This process is repeated until a convergence criterion is satisfied. Equations 12 and 13 present the update formulation concerning the velocity and position of particle i at time step t, respectively:

$$\boldsymbol{\rho}_i^{t+1} = \mu\boldsymbol{\rho}_i^t + c_1 r_1(\hat{\boldsymbol{\psi}}_i - \boldsymbol{\psi}_i^t) + c_2 r_2(\boldsymbol{g} - \boldsymbol{\psi}_i^t) \tag{12}$$

and

$$\boldsymbol{\psi}_i^{t+1} = \boldsymbol{\psi}_i^t + \boldsymbol{\rho}_i^{t+1}, \tag{13}$$

where μ is the inertia weight that controls the interaction among particles, and $r_1, r_2 \in [0, 1]$ are random variables that give a stochastic trait to PSO. Additionally, variables c_1 and c_2 are constants that conduct the swarm's members onto the search space. Algorithm 4 depicts the pseudo-code of PSO.

Algorithm 4: Particle Swarm Optimization pseudo-code.

Input: Number of particles M, number of decision variables N, maximum iterations T, constants c_1 and c_2, and inertia weight μ.
Output: $\mathbf{g}$ decision variables vector holding the most suitable values for the task's fitness function.

```
Fitness function f(ψ), ψ = (ψ_1, ..., ψ_N)^T;
Initialize the particles population d_i, i = (1, 2, ..., M);
t ← 1;
while (t < T) do
    for i from 1 to M do
        Update each particle i velocity according to Equation 12;
        Update each particle i position according to Equation 13;
    t ← t + 1;
    ψ* ← Best swarm's particle;
g ← ψ*;
```

Table 1 Word2Vec parameters configuration

Parameter	Value
Number of features	300
Window size	5
Minimum word-occurence	100
Algorithm	CBOW
η (learning rate)	0.1
t (number of iterations)	10

5 Methodology

In this section, we present the proposed approach to fine-tune the perplexity parameter concerning t-SNE dimensionality reduction, as well as describe the employed datasets and the experimental setup.

5.1 Modeling t-SNE optimization

We propose to model the problem of selecting a suitable perplexity parameter considering t-SNE in the task of word embeddings dimensionality reduction. As aforementioned in Sect. 3, the t-SNE learning step has four parameters: the number of components c, the perplexity γ, the learning rate η and the number of iterations t. As we are interested in fine-tuning the perplexity only, we fixed the 3-tuple (c, η, t) and played with parameter γ in order to minimize the Kullback–Leibler divergence (KL) of the t-SNE's dimensionality reduction.

Table 2 t-SNE parameters configuration

Parameter	Value
c (number of components)	2
η (learning rate)	200
t (number of iterations)	1000
γ (perplexity)	[1, 100]

5.2 Datasets

We considered three English-based text datasets in the experimental section, as follows:

- 20 Newsgroups[3] [16]: comprises around 18,000 newsgroups posts on 20 distinct topics;
- Movie Reviews[4] [23]: composed of 2000 movie reviews;
- Reuters-21578 (ApteMod)[5] [18]: it is a collection of 10,788 documents from the Reuters financial newswire service.

Table 3 Meta-heuristic algorithms parameters configuration

Algorithm	Parameters
ABC	`n_trials` = 10
BA	f = [0, 2] | A = 0.5 | r = 0.5
GP	`n_terminals` = 2 | `depth` = [2, 5] | `functions` = [SUM, SUB, MUL, DIV]
	`p_reproduction` = 0.25 | `p_mutation` = 0.1 | `p_crossover` = 0.2
PSO	μ = 0.7 | c_1 = 1.7 | c_2 = 1.7

[3] http://qwone.com/~jason/20Newsgroups.
[4] http://cs.cornell.edu/people/pabo/movie-review-data.
[5] https://martin-thoma.com/nlp-reuters.

5.3 Experimental setup

Before feeding the word embeddings into the t-SNE, we need to obtain them by training a Word2Vec algorithm.[6] Table 1 depicts the parameters employed to train the Word2Vec architecture. Such values are the same used for all datasets.

After obtaining the word embeddings, we fixed each t-SNE[7] architecture parameters according to Table 2. Concerning the perplexity rates, we set $\gamma \in [1, 100]$, which means we used such ranges to initialize the optimization techniques. Such a range of values was chosen by previous experimental analysis. Note that the baseline experiments also use the very same configurations. The main difference lies in the value of γ, where the 'default-perplexity' t-SNE

Table 4 Mean results and their respective standard deviation over 20 Newsgroups dataset

Algorithm	KL-Divergence	Perplexity
ABC	**1.6535 ± 0.0037**	99.93 ± 0.21
BA	**1.6512 ± 0.0032**	99.91 ± 0.28
GP	**1.6489 ± 0.0014**	100.00 ± 0.00
PSO	**1.6519 ± 0.0035**	99.98 ± 0.08
RS	1.9373 ± 0.1671	44.88 ± 27.65
T_l	1.6733 ± 0.0077	1.00 ± 0.00
T_d	2.0154 ± 0.0056	30.00 ± 0.00
T_h	**1.6570 ± 0.0056**	100.00 ± 0.00

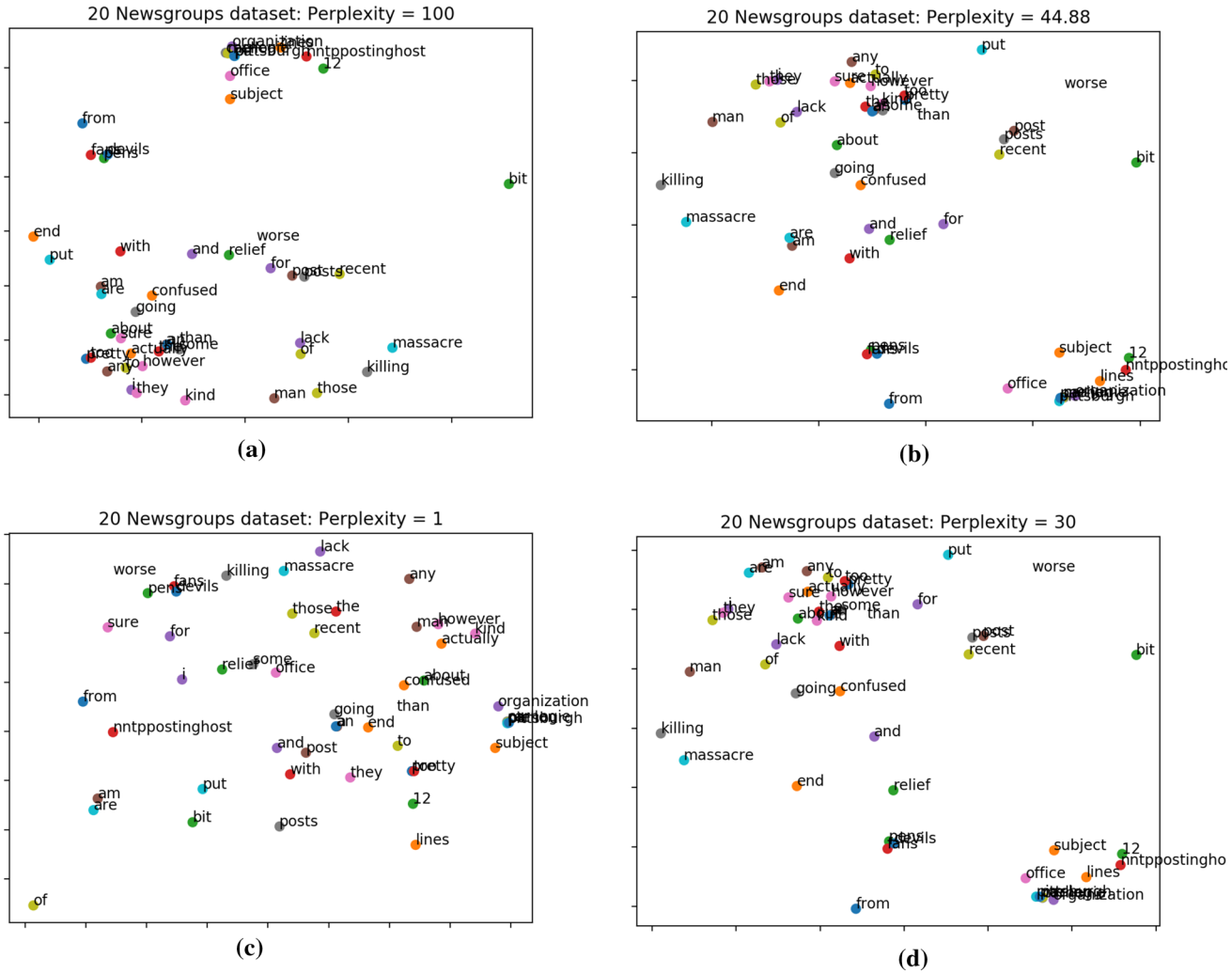

Fig. 3 t-SNE's dimensionality reduction over 20 Newsgroups dataset: **a** GP, **b** RS, **c** T_l and **d** T_d

[6] We opted to use the Word2Vec provided by the Gensim package.

[7] We opted to use the t-SNE implementation provided by the Scikit-Learn package.

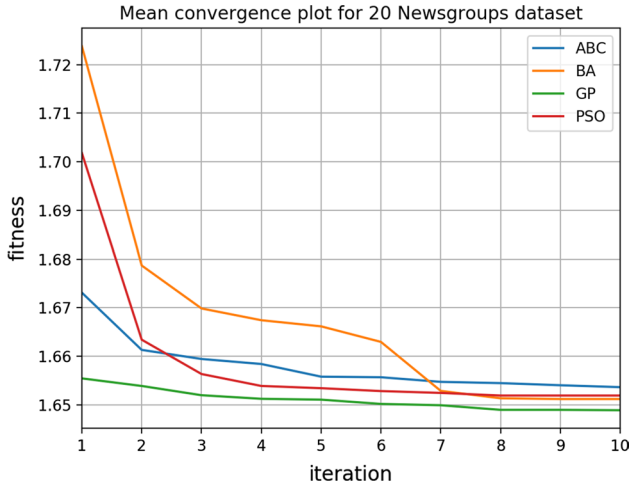

Fig. 4 Mean fitness convergence over 20 Newsgroups dataset optimization

Table 5 Mean results and their respective standard deviation over Movie Reviews dataset

Algorithm	KL-Divergence	Perplexity
ABC	0.9943 ± 0.0059	99.65 ± 0.57
BA	**0.9860 ± 0.0073**	99.77 ± 0.49
GP	**0.9850 ± 0.0041**	100.00 ± 0.00
PSO	**0.9869 ± 0.0044**	100.00 ± 0.00
RS	1.1366 ± 0.1324	74.46 ± 21.87
T_l	1.0306 ± 0.0160	1.00 ± 0.00
T_d	1.4135 ± 0.0260	30.00 ± 0.00
T_h	1.0152 ± 0.0228	100.00 ± 0.00

uses $\gamma = 30$ (T_d), the 'low-perplexity' t-SNE uses $\gamma = 1$ (T_l), and the 'high-perplexity' version uses $\gamma = 100$ (T_h).

For every dataset, each meta-heuristic[8] was evaluated under the whole dataset, as we are trying to find the dataset's most suitable word embeddings visualization.[9] As we are dealing with only a one-dimension optimization task, we performed a small previous grid-search experiment to select the number of agents, ranging from 1 to 10 and found out that the most suitable value was 5. Thus, for every meta-heuristic, 5 agents (particles) were used over 10 convergence iterations.[10] To provide a thorough and fair comparison among meta-heuristics in the context of t-SNE perplexity parameter fine-tuning, we have chosen different techniques, ranging from swarm-based to evolutionary-inspired ones: Artificial Bee Colony, Bat Algorithm, and Genetic Programming, Particle Swarm Optimization, as well as a Random Search (RS), i.e., a random initialization of the γ parameter. Table 3 exhibits the parameter configuration for every meta-heuristic technique.[11]

To perform a reasonable comparison among distinct meta-heuristic techniques, we must rely on mathematical methods that will sustain these observations. The first step is to decide whether to use a parametric or a non-parametric statistical test [9]. Unfortunately, we can not consider a normality state from our experiments due to insufficient data and sensitive outliers, restraining our analysis to non-parametric approaches.

Secondly, acknowledging that our experiments' results are independent (e.g., reconstruction error) and continuous over a particular dependent variable (e.g., number of observations), we can identify that the Wilcoxon signed-rank test [25] will satisfy our obligations. It is a non-parametric hypothesis test used to compare two or more related observations (in our case, repeated measurements over a certain meta-heuristic) to assess whether there are statistically significant differences between them.

6 Experimental results

This section aims at presenting the experimental results concerning t-SNE perplexity parameter fine-tuning.[12]

6.1 20 Newsgroups

Table 4 depicts the mean results concerning the t-SNE optimization over 20 Newsgroups dataset (vocabulary size = 4581), where the bolded values are the best ones according to Wilcoxon signed-rank test. One can perceive that the meta-heuristic techniques were almost trapped into the upper bound, mainly due to the direct correlation between the perplexity parameter and the KL divergence. As the perplexity increases, the KL divergence decreases, thus, making it unfeasible to explore the search space properly. Additionally, it is crucial to highlight that the default perplexity parameter (T_d) obtained the worst result among all techniques, creating a gap of parameter selection and allowing meta-heuristic techniques to fulfill it. Moreover, while the RS algorithm obtained a feasible perplexity parameter, it also produced a high standard deviation, being an extremely random search algorithm, as its name suggests.

[8] All meta-heuristics are available in the Opytimizer library: https://github.com/gugarosa/opytimizer.

[9] Each dataset was evaluated 10 times in the attempt to mitigate the meta-heuristics stochastic nature.

[10] The number of iterations is our stopping criteria.

[11] Note that these values were empirically chosen according to their author's definition.

[12] The experiments' source code can be found at https://gist.github.com/gugarosa/fbbc294da27b163caed924062ad69a7e.

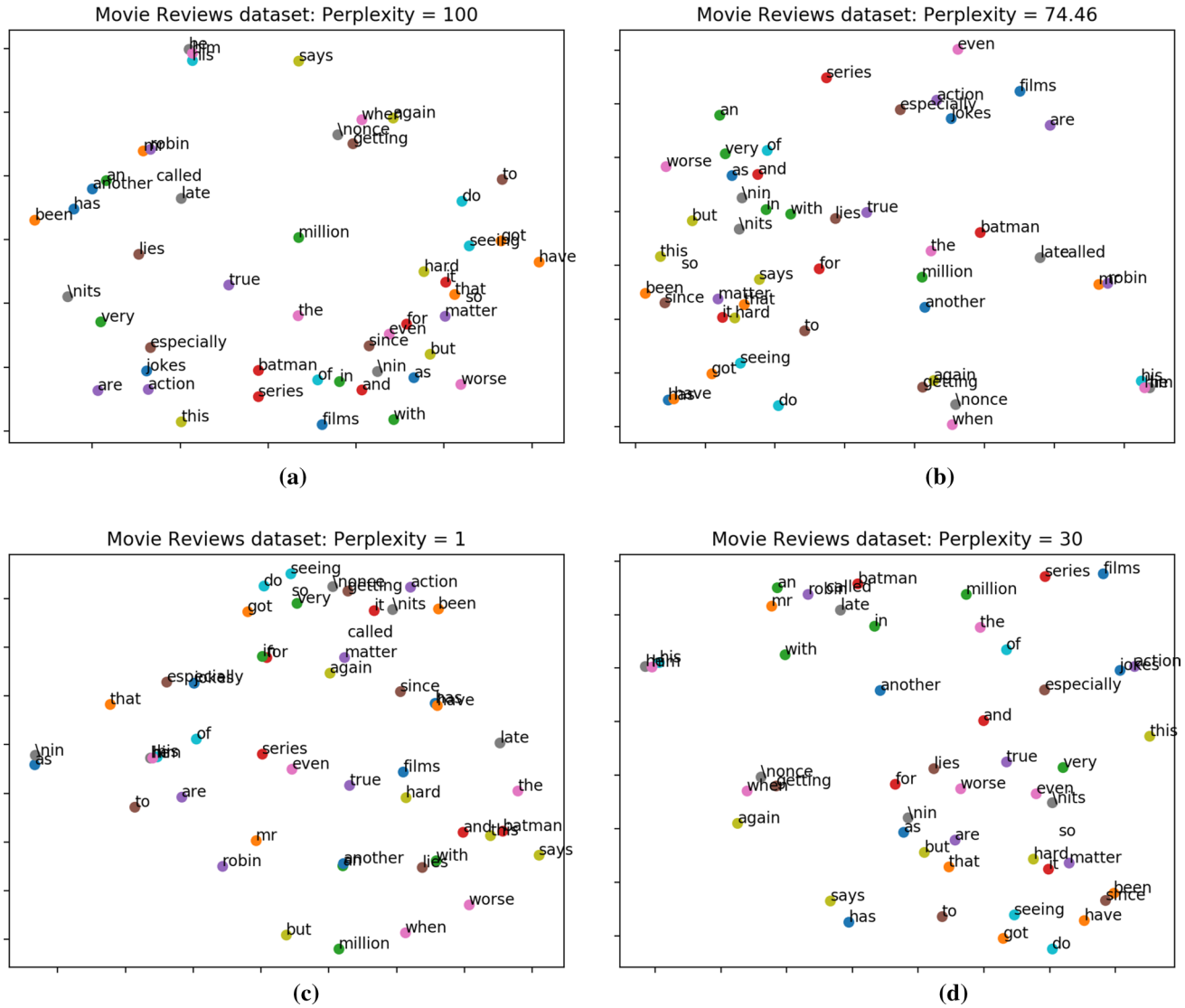

Fig. 5 t-SNE's dimensionality reduction over Movie Reviews dataset: **a** GP, **b** RS, **c** T_l and **d** T_d

Figure 3 illustrates the t-SNE's dimensionality reduction process over 20 Newsgroups dataset using four distinct perplexity values. One can observe that lower perplexity values spread the words over the space, while higher perplexity values aggregate them into clusters of similar meanings, e.g., 'post' and 'posts' being closer with $\gamma = 30$ and $\gamma = 100$.

Moreover, Fig. 4 displays the mean fitness convergence for all the employed meta-heuristic techniques. We can highlight that GP and BA were able to achieve the lowest fitness and corroborate the results exhibited in Table 4. Also, ABC was able to converge well and obtain a slightly worse value than BA, GP, and PSO.

6.2 Movie reviews

Table 5 depicts the mean results concerning the t-SNE optimization over Movie Reviews dataset (vocabulary size = 1392), where the most significant results according to Wilcoxon signed-rank test are in bold. One can observe that the meta-heuristic techniques were trapped into the upper bound (unless ABC and BA, which almost achieved the upper bound). Nevertheless, even trapped by the upper bound, they were able to achieve the lowest KL divergence among all techniques and perform substantially better than RS, which provides a plausible mean perplexity value but with the cost of a higher standard deviation. Moreover, as this dataset provides a smaller vocabulary size, the KL divergence is smaller than the previous dataset experiments.

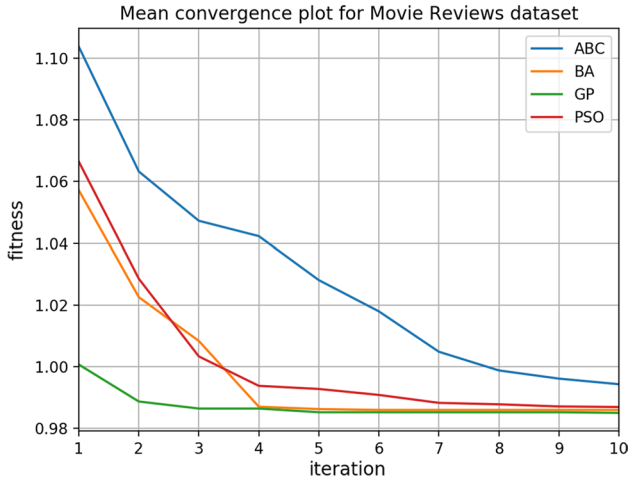

Fig. 6 Mean fitness convergence over Movie Reviews dataset optimization

Table 6 Mean results and their respective standard deviation over Reuters-21578 (ApteMod)

Algorithm	KL-Divergence	Perplexity
ABC	**1.0399 ± 0.0216**	40.60 ± 51.12
BA	**1.0529 ± 0.0065**	99.99 ± 0.04
GP	**1.0247 ± 0.0112**	1.00 ± 0.00
PSO	**1.0507 ± 0.0044**	100.00 ± 0.00
RS	1.3014 ± 0.1512	50.24 ± 28.93
T_l	**1.0400 ± 0.0109**	1.00 ± 0.00
T_d	1.4004 ± 0.0063	30.00 ± 0.00
T_h	1.0756 ± 0.0103	100.00 ± 0.00

Figure 5 illustrates the t-SNE's dimensionality reduction process over Movie Reviews dataset using four distinct perplexity values. Again, one can observe that lower perplexity values spread the words over the space, while higher perplexity values aggregate them into clusters of similar meanings, e.g., 'films' and 'batman' being closer with $\gamma = 100$.

Figure 6 displays the mean fitness convergence for all the employed meta-heuristic techniques. Considering the Movie Reviews dataset, BA, GP, and PSO accomplished the best mean convergence among all techniques and obtained the best results according to Wilcoxon's. Additionally, ABC did not achieve the same performance as BA, GP, and PSO, being significantly worse result than its counterparts.

6.3 Reuters-21578 (ApteMod)

Table 6 depicts the mean results concerning the t-SNE optimization over Reuters-21578 (ApteMod) dataset (vocabulary size = 1595), where the best results according to Wilcoxon signed-rank test are in bold. It is essential to highlight that ABC was trapped into the lower and upper bounds throughout the runnings, producing a mean perplexity value with a high standard deviation. Nevertheless, one interesting point to elucidate is that it was able to achieve the lowest KL divergence. Additionally, according to Wilcoxon's signed-rank test, ABC, BA, GP, PSO, and T_l achieved the best results among all techniques. Another vital point to highlight is that GP was trapped in the lower bound, and achieved the lowest KL divergence, corroborating with the values found by T_l.

Figure 7 illustrates the t-SNE's dimensionality reduction process over Reuters-21578 (ApteMod) dataset using three distinct perplexity values. In this particular dataset, higher values of perplexity aggregate the words almost into a single cluster, losing valuable information regarding their similarity. Lower values of perplexity, $\gamma = 1$, provides a clearer visualization, where 'corp', 'share', and 'pct' are close together. This is a fruitful insight as usually corporation shares are measured in percentage values. Also, it is possible to correlate 'oil' and 'gas', which makes sense as these two words share similar concepts. Moreover, one can see in Fig. 7 the outcome of t-SNE's dimensionality reduction corroborating with the values depicted in Table 6.

Figure 8 displays the mean fitness convergence for all the employed meta-heuristic techniques. Considering the Reuters-21578 (ApteMod) dataset, GP achieved the best mean convergence among all techniques. Nevertheless, its performance can be compared with ABC, BA, and PSO ones, as indicated by Wilcoxon's signed-rank test.

6.4 Similarity discussion

When dealing with Word2Vec models, it is possible to extract the similarity between two or more words through a cosine distance metric, which varies from −1 to 1. Let w_1 be the first word and w_2 the second word. Also, let Eq. 14 be the cosine distance metric, used to calculate their similarity, as follows:

$$similarity = cos(\theta) = \frac{w_1 \cdot w_2}{||w_1|| ||w_2||} \tag{14}$$

In order to provide an additional experiment regarding how the perplexity parameter influences the dimensionality reduction, we present a similarity discussion analysis to check whether if t-SNE is suitable or not to visualize the similarity between word embeddings. Note that we provide a visual similarity analysis and that the 'positive' term (blue dots) stands for the most similar words, while the 'negative' term (red dots) stands for the least similar words.

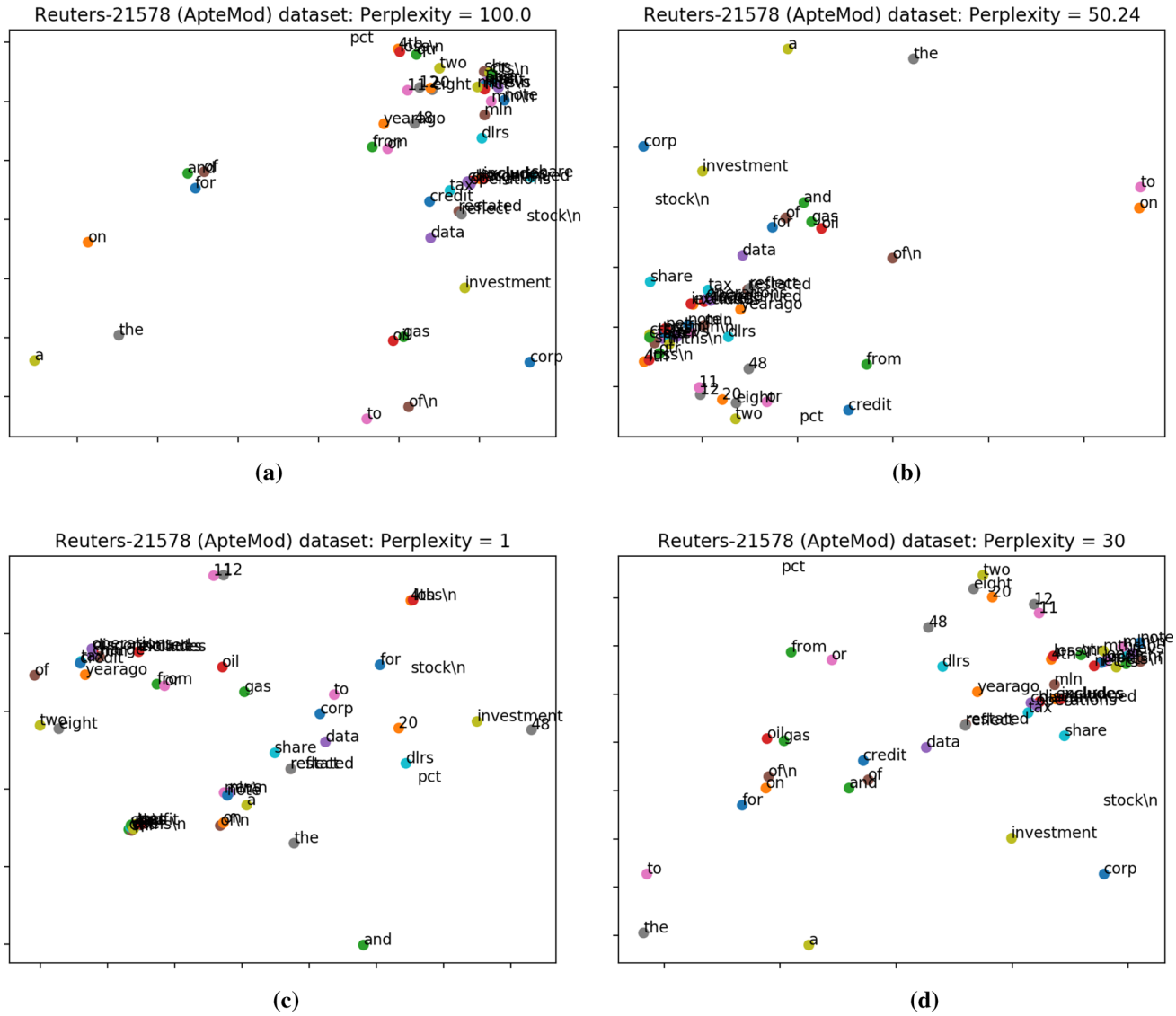

Fig. 7 t-SNE's dimensionality reduction over Reuters-21578 (ApteMod) dataset: **a** GP, **b** RS, **c** T_l and **d** T_d

Figure 9 depicts the top-10 positive and negative most similar words to 'posts' on the 20 Newsgroups dataset. One can perceive that increasing the perplexity parameters, 'positive' words start to become closer to the reference word (posts). Nevertheless, it is important to highlight that due to the loss of information in the dimensionality reduction process, there are no positive nor negative distinct clusters, undermining the similarity visualization.

Figure 10 depicts the top-10 positive and negative most similar words to 'posts' on the Movie Reviews dataset, while Fig. 11 depicts the top-10 positive and negative most similar words to 'gas' on the Reuters-21578 (ApteMod) dataset. As aforementioned, t-SNE impacts in the loss of crucial information regarding words' similarity. Notwithstanding, another interesting point to elucidate is the narrowing of the space when using higher perplexity values, indicated by the labels ticks. When using a higher perplexity parameter, words become closer to each other, as the KL-divergence is minimized throughout the dimensionality reduction procedure.

6.5 Computational load

Another crucial point to elucidate is the computational burden between the meta-heuristic techniques. One can perceive that it is vital to accomplish an optimization task within feasible solutions and a viable time. Figure 12 exhibits the computational load between all meta-heuristics concerning the 20 Newsgroups dataset. Some swarm-based techniques obtained the highest

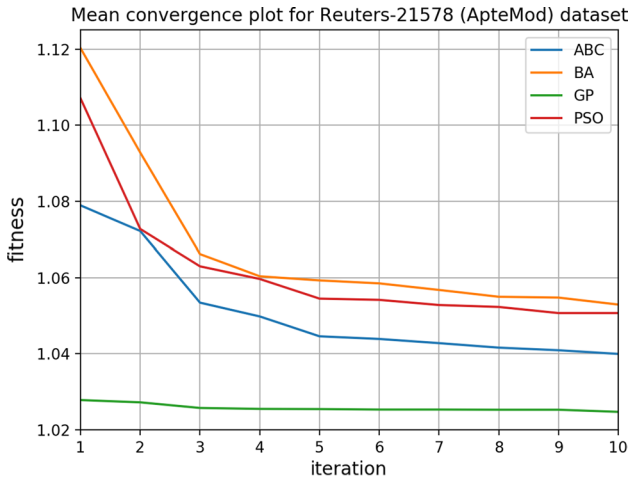

Fig. 8 Mean fitness convergence over Reuters-21578 (ApteMod) dataset optimization

computational loads, e.g., ABC and BA, as they have more complex positions update mechanisms throughout each iteration. Moreover, another swarm-based technique, e.g., PSO, almost reached the lowest computational time, as it is a straightforward technique that does not need to create additional particles nor evaluate specific ones throughout the iterations. On the other hand, GP achieved the lowest time throughout all runnings, as this evolutionary-based technique selects, mutates, and crosses a population into a new one, only evaluating it once per iteration.

Additionally, Figs. 13 and 14 depicts the computational load concerning the Movie Reviews and Reuters-21578 (ApteMod) datasets, respectively. One can notice the meta-heuristics behaved in the same way as the previous dataset, i.e., ABC and BA obtained the highest computational burden, followed by PSO and GP. Therefore, we can conclude that the most suitable technique to fulfill our optimization task was GP, as it found suitable solutions within the lowest computational load.

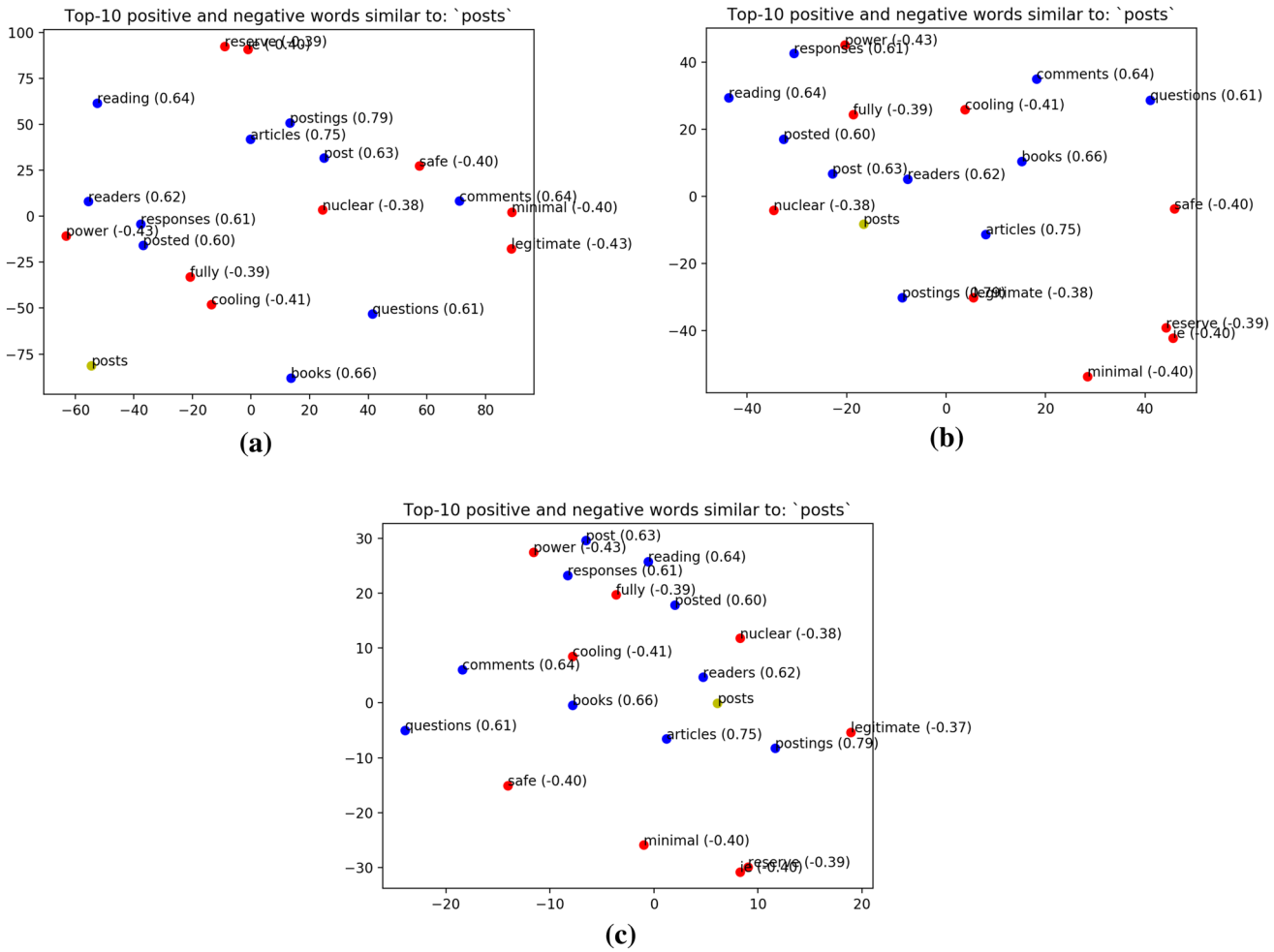

Fig. 9 Most similar words to 'posts' over 20 Newsgroups dataset: **a** T_l, **b** T_d and **c** T_h

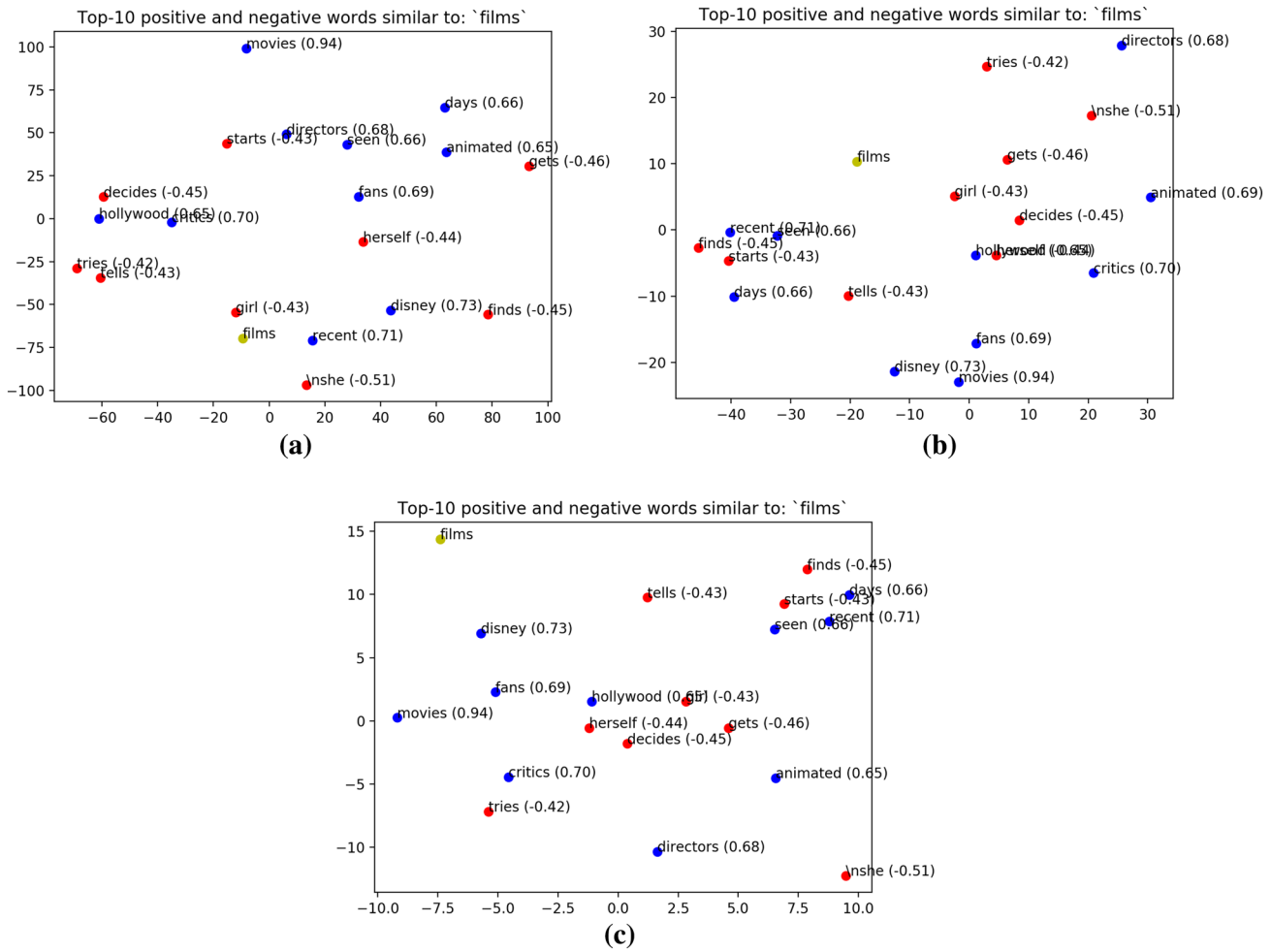

Fig. 10 Most similar words to 'films' over Movie Reviews dataset: **a** T_l, **b** T_d and **c** T_h

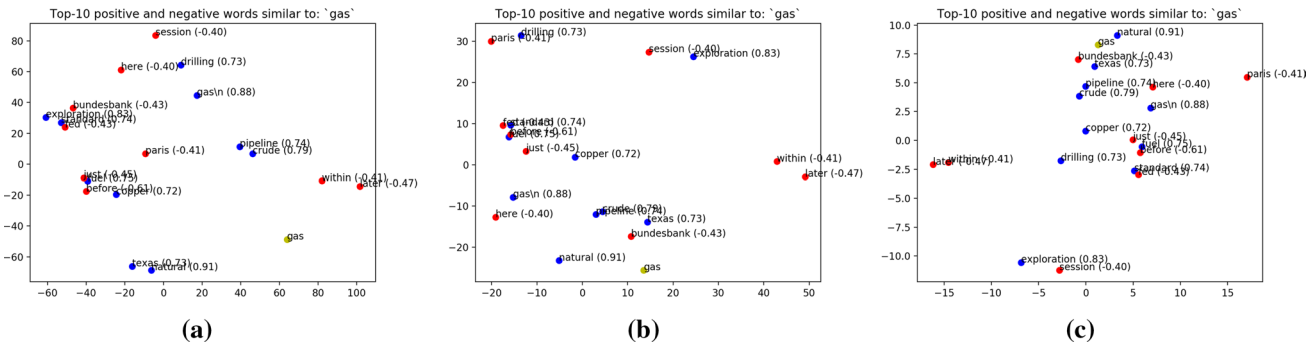

Fig. 11 Most similar words to 'gas' over Reuters-21578 (ApteMod) dataset: **a** T_l, **b** T_d and **c** T_h

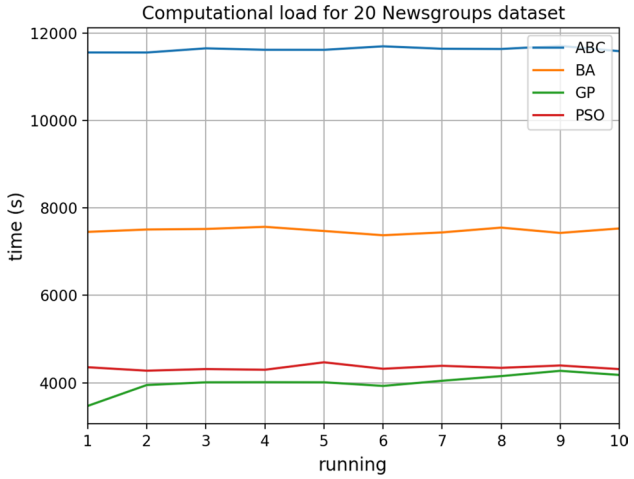

Fig. 12 Computational load over 20 Newsgroups dataset optimization

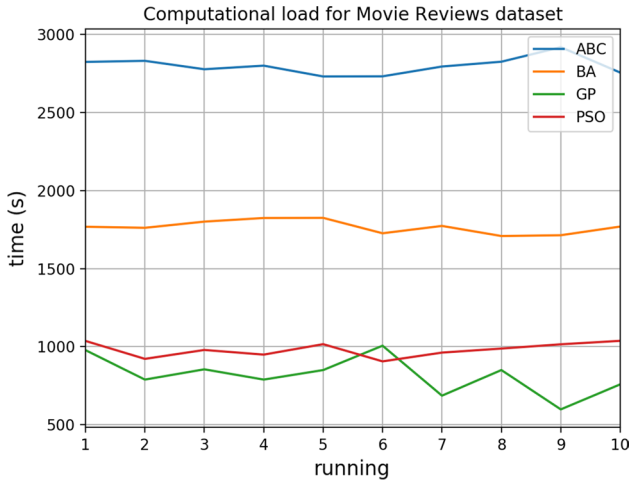

Fig. 13 Computational load over Movie Reviews dataset optimization

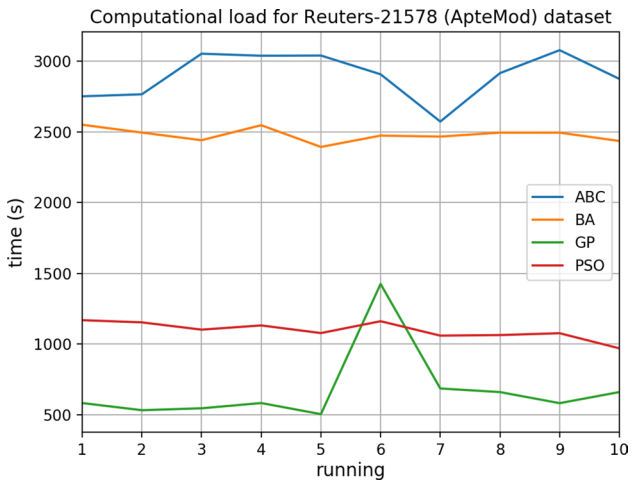

Fig. 14 Computational load over Reuters-21578 (ApteMod) dataset optimization

7 Conclusions

This paper discussed the problem of fine-tuning the perplexity parameter in t-SNE dimensionality reduction through meta-heuristic optimization algorithms, such as Artificial Bee Colony, Bat Algorithm, Genetic Programming, and Particle Swarm Optimization.

All four algorithms were able to fine-tune the perplexity parameter in all datasets, supported by the KL divergence decay. Moreover, it is possible to correlate the KL divergence directly to the perplexity's choice, i.e., as the perplexity value increases, the KL divergence decreases. Therefore, it might be interesting to consider additional variables over the search space in an attempt to modify the fitness function and remove the direct correlation between it and the employed decision variable.

Additionally, we provide a similarity analysis regarding three different terms from the three assessed datasets, where it was possible to visualize how t-SNE influences the similarity visualization. As the t-SNE stands for a dimensionality reduction technique, it plays a significant role in losing vital information, making not so visual the difference between positive and negative similar words, i.e., positive and negative words were mixed in the visualization plot. Moreover, we analyze the computational load between all meta-heuristics techniques. Considering all datasets, ABC obtained the highest computational time, followed by BA, PSO, and GP. Also, it is essential to highlight that GP obtained feasible solutions within the lowest optimization times for all datasets, being the most suitable meta-heuristic technique in this work.

For future works, we aim to explore the correlations between other t-SNE parameters, e.g., the learning rate and the number of iterations, as well as to consider such an approach in the context of additional word embeddings models, e.g., GloVe, ELMO, and BERT, alongside with 3-D visualization models.

Acknowledgements The authors are grateful to FAPESP Grant #2019/02205-5.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Bertsekas DP (1999) Nonlinear programming. Athena Scientific, Nashua
2. Blei DM, Ng AY, Jordan MI (2003) Latent Dirichlet allocation. J Mach Learn Res 3(Jan):993–1022

3. Cao Y, Wang L (2017) Automatic selection of t-SNE perplexity. arXiv preprint arXiv:1708.03229
4. Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R (1990) Indexing by latent semantic analysis. J Am Soc Inf Sci 41(6):391–407
5. Elman JL (1990) Finding structure in time. Cogn Sci 14(2):179–211
6. Fister Jr I, Yang XS, Fister I, Brest J, Fister D (2013) A brief review of nature-inspired algorithms for optimization. arXiv preprint arXiv:1307.4186
7. Ganguly D, Roy D, Mitra M, Jones GJ (2015) Word embedding based generalized language model for information retrieval. In: Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval, pp 795–798. ACM
8. Go A, Bhayani R, Huang L (2009) Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, vol 1(12), p 2009
9. Hollander M, Wolfe DA (1999) Nonparametric statistical methods. Wiley, Hoboken
10. Jolliffe I (2011) Principal component analysis. Springer, Berlin
11. Jones MN, Mewhort DJ (2007) Representing word meaning and order information in a composite holographic lexicon. Psychol Rev 114(1):1
12. Kanerva P, Kristoferson J, Holst A (2000) Random indexing of text samples for latent semantic analysis. In: Proceedings of the annual meeting of the Cognitive Science Society, vol 22
13. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Global Optim 39(3):459–471
14. Kennedy J, Eberhart R (2001) Swarm intelligence. M. Kaufman, Vienna
15. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
16. Lang K (1995) Newsweeder: learning to filter netnews. In: Proceedings of the twelfth international conference on machine learning, pp 331–339
17. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
18. Lewis DD (1991) Evaluating text categorization. In: Proceedings of speech and natural language workshop, pp 312–318. Defense Advanced Research Projects Agency, Morgan Kaufmann
19. Liou CY, Cheng WC, Liou JW, Liou DR (2014) Autoencoder for words. Neurocomputing 139:84–96
20. Lund K, Burgess C (1996) Producing high-dimensional semantic spaces from lexical co-occurrence. Behav Res Methods Instrum Comput 28(2):203–208
21. Maaten L, Hinton G (2008) Visualizing data using t-sne. J Mach Learn Res 9(Nov):2579–2605
22. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781
23. Pang B, Lee L, Vaithyanathan S (2002) Thumbs up?: sentiment classification using machine learning techniques. In: Proceedings of the ACL-02 conference on empirical methods in natural language processing, vol 10, pp 79–86. Association for Computational Linguistics
24. Sundermeyer M, Schlüter R, Ney H (2012) LSTM neural networks for language modeling. In: Thirteenth annual conference of the international speech communication association
25. Wilcoxon F (1945) Individual comparisons by ranking methods. Biom Bull 1(6):80–83
26. Yang XS (2011) Review of meta-heuristics and generalised evolutionary walk algorithm. Int J Bio-Inspired Comput 3(2):77–84
27. Yang XS, Gandomi AH (2012) Bat algorithm: a novel approach for global engineering optimization. Eng Comput 29(5):464–483