Unsupervised and Semi-Supervised Learning
Series Editor: M. Emre Celebi

Michael W. Berry
Azlinah Mohamed
Bee Wah Yap *Editors*

Supervised and Unsupervised Learning for Data Science

Springer

Unsupervised and Semi-Supervised Learning

Series Editor

M. Emre Celebi, Computer Science Department, Conway, Arkansas, USA

Springer's Unsupervised and Semi-Supervised Learning book series covers the latest theoretical and practical developments in unsupervised and semi-supervised learning. Titles – including monographs, contributed works, professional books, and textbooks – tackle various issues surrounding the proliferation of massive amounts of unlabeled data in many application domains and how unsupervised learning algorithms can automatically discover interesting and useful patterns in such data. The books discuss how these algorithms have found numerous applications including pattern recognition, market basket analysis, web mining, social network analysis, information retrieval, recommender systems, market research, intrusion detection, and fraud detection. Books also discuss semi-supervised algorithms, which can make use of both labeled and unlabeled data and can be useful in application domains where unlabeled data is abundant, yet it is possible to obtain a small amount of labeled data.

Topics of interest in include:

- Unsupervised/Semi-Supervised Discretization
- Unsupervised/Semi-Supervised Feature Extraction
- Unsupervised/Semi-Supervised Feature Selection
- Association Rule Learning
- Semi-Supervised Classification
- Semi-Supervised Regression
- Unsupervised/Semi-Supervised Clustering
- Unsupervised/Semi-Supervised Anomaly/Novelty/Outlier Detection
- Evaluation of Unsupervised/Semi-Supervised Learning Algorithms
- Applications of Unsupervised/Semi-Supervised Learning

While the series focuses on unsupervised and semi-supervised learning, outstanding contributions in the field of supervised learning will also be considered. The intended audience includes students, researchers, and practitioners.

More information about this series at http://www.springer.com/series/15892

Michael W. Berry • Azlinah Mohamed
Bee Wah Yap
Editors

Supervised and Unsupervised Learning for Data Science

Springer

Editors
Michael W. Berry
Department of Electrical Engineering
and Computer Science
University of Tennessee at Knoxville
Knoxville, TN, USA

Azlinah Mohamed
Faculty of Computer & Mathematical
Sciences
Universiti Teknologi MARA
Shah Alam, Selangor, Malaysia

Bee Wah Yap
Advanced Analytics Engineering Centre,
Faculty of Computer
and Mathematical Sciences
Universiti Teknologi MARA
Shah Alam, Selangor, Malaysia

ISSN 2522-848X ISSN 2522-8498 (electronic)
Unsupervised and Semi-Supervised Learning
ISBN 978-3-030-22474-5 ISBN 978-3-030-22475-2 (eBook)
https://doi.org/10.1007/978-3-030-22475-2

This Springer imprint is published by the registered company Springer Nature Switzerland AG.
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Supervised and unsupervised learning algorithms have shown a great potential in knowledge acquisition from large data sets. Supervised learning reflects the ability of an algorithm to generalize knowledge from available data with target or labeled cases so that the algorithm can be used to predict new (unlabeled) cases. Unsupervised learning refers to the process of grouping data into clusters using automated methods or algorithms on data that has not been classified or categorized. In this situation, algorithms must "learn" the underlying relationships or features from the available data and group cases with similar features or characteristics. When small amounts of labeled data are available, the learning is specified as "semi-supervised." This volume provides both foundational knowledge for novice or beginning researchers in machine learning and new techniques for improving both the accuracy and computational complexity of supervised and unsupervised learning in the context of relevant and practical applications.

Part I of this volume is dedicated to the discussion of state-of-the-art algorithms used in supervised and unsupervised learning paradigms. In Chap. 1, Alloghani et al. provide a systematic literature review of scholarly articles published between 2015 and 2018 that address or implement supervised and unsupervised machine learning techniques in different problem-solving paradigms. In Chap. 2, C. Lursinsap addresses recent approaches to overcome commonly observed problems in big data analytics, such as data overflow, uncontrollable learning epochs, arbitrary class drift, and dynamic imbalanced class ratios. In Chap. 3, T. Panitanarak discusses recent improvements in the performance of graph-based shortest path algorithms that are commonly used in machine learning. Finally, in Chap. 4, R. Lowe and M. Berry illustrate the use of tensor-based algorithms for the unsupervised learning of influence in text-based media.

Part II of this volume highlights the various applications of learning algorithms including cancer diagnosis, social media and text mining, and prediction of stress-strain parameters in civil engineering. In Chap. 5, Prasetyo et al. demonstrate the use of support vector machines (SVMs) in cancer survival data analysis. In Chap. 6, D. Martin et al. discuss the use of latent semantic analysis (LSA) for unsupervised word sense disambiguation in textual documents. In Chap. 7, Pornwattanavichai

et al. explain how hybrid recommendation systems with latent Dirichlet analysis (LDA) can improve the unsupervised topic modeling of tweets. Finally, in Chap. 8, Jebura et al. demonstrate the use of artificial neural networks (ANNs) to predict nonlinear hyperbolic soil stress-strain relationship parameters in civil engineering models.

Some of the research described in this volume was supported by the US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Robinson Pino, program manager, under contract number DE-AC05-00OR22725.

Knoxville, TN — Michael W. Berry
Shah Alam, Selangor, Malaysia — Bee Wah Yap
Shah Alam, Selangor, Malaysia — Azlinah Mohamed

Contents

Part I
Algorithms

Chapter 1
A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science

Mohamed Alloghani, Dhiya Al-Jumeily, Jamila Mustafina, Abir Hussain, and Ahmed J. Aljaaf

1.1 Introduction

The demand for advanced data analytics leading to the use of machine learning and other emerging techniques can be attributed to the advent and subsequent development of technologies such as Big Data, Business Intelligence, and the applications that require automation. As Sandhu [1] explains, machine learning is a subset of artificial intelligence, which uses computerized techniques to solve problems based on historical data and information without unnecessarily requiring modification in the core process. Essentially, artificial intelligence involves creation of algorithms and other computation techniques that promote smartness of machines. It encompasses algorithms that think, act, and implement tasks using protocols that are otherwise beyond human's reach.

M. Alloghani (✉)
Applied Computing Research Group, Liverpool John Moores University, Liverpool, UK

Abu Dhabi Health Services Company (SEHA), Abu Dhabi, UAE
e-mail: M.AlLawghani@2014.ljmu.ac.uk; mloghani@seha.ae

D. Al-Jumeily · A. Hussain
Applied Computing Research Group, Liverpool John Moores University, Liverpool, UK

J. Mustafina
Kazan Federal University, Kazan, Russia
e-mail: dnmustafina@kpfu.ru

A. J. Aljaaf
Applied Computing Research Group, Liverpool John Moores University, Liverpool, UK

Centre of Computer, University of Anbar, Anbar, Iraq
e-mail: A.J.Kaky@ljmu.ac.uk; a.j.aljaaf@uoanbar.edu.iq

M. W. Berry et al. (eds.), *Supervised and Unsupervised Learning for Data Science*,
Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-030-22475-2_1

Machine learning is a component of artificial intelligence although it endeavors to solve problems based on historical or previous examples [2]. Unlike artificial intelligence applications, machine learning involves learning of hidden patterns within the data (data mining) and subsequently using the patterns to classify or predict an event related to the problem [3]. Simply, intelligent machines depend on knowledge to sustain their functionalities and machine learning offers such a knowledge. In essence, machine learning algorithms are embedded into machines and data streams provided so that knowledge and information are extracted and fed into the system for faster and efficient management of processes. It suffices to mention that all machine learning algorithms are also artificial intelligence techniques although not all artificial intelligence methods qualify as machine learning algorithms.

Machine learning algorithms can either be supervised or unsupervised although some authors also classify other algorithms as reinforcement, because such techniques learn data and identify pattern for the purposes of reacting to an environment. However, most articles recognize supervised and unsupervised machine learning algorithms. The difference between these two main classes is the existence of labels in the training data subset. According to Kotsiantis [4], supervised machine learning involves predetermined output attribute besides the use of input attributes. The algorithms attempt to predict and classify the predetermined attribute, and their accuracies and misclassification alongside other performance measures is dependent on the counts of the predetermined attribute correctly predicted or classified or otherwise. It is also important to note the learning process stops when the algorithm achieves an acceptable level of performance [5]. According to Libbrecht and Noble [2], technically, supervised algorithms perform analytical tasks first using the training data and subsequently construct contingent functions for mapping new instance of the attribute. As stated previously, the algorithms require prespecifications of maximum settings for the desired outcome and performance levels [2, 5]. Given the approach used in machine learning, it has been observed that training subset of about 66% is rationale and helps in achieving the desired result without demanding for more computational time [6]. The supervised learning algorithms are further classified into classification and regression algorithms [3, 4].

Conversely, unsupervised data learning involves pattern recognition without the involvement of a target attribute. That is, all the variables used in the analysis are used as inputs and because of the approach, the techniques are suitable for clustering and association mining techniques. According to Hofmann [7], unsupervised learning algorithms are suitable for creating the labels in the data that are subsequently used to implement supervised learning tasks. That is, unsupervised clustering algorithms identify inherent groupings within the unlabeled data and subsequently assign label to each data value [8, 9]. On the other hand, unsupervised association mining algorithms tend to identify rules that accurately represent relationships between attributes.

1.1.1 Motivation and Scope

Even though both supervised and unsupervised algorithms are widely used to accomplish different data mining tasks, the discussion of the algorithms has been mostly done singly or grouped depending on the need of learning tasks. More importantly, literature reviews that have been conducted to account for supervised and unsupervised algorithms either handle supervised techniques or unsupervised ones with limited focus on both approaches in the same. For instance, Sandhu [1] wrote a review article on machine learning and natural language processing but focused on supervised machine learning. The author did not conduct a systematic review and, as such, the article does not focus on any specific period or target any given database. Baharudin et al. [10] also conducted a literature review on machine learning techniques though in the context of text data mining and did not implement any known systematic review methodology. Praveena [11] also conducted a review of papers that had implemented supervised learning algorithms and, as such, did implement any of the known systematic review approaches. However, Qazi et al. [12] conducted a systematic review although with a focus on the challenges that different authors encountered while implementing different classification techniques in sentimental analysis. The authors reviewed 24 papers that were published between 2002 and 2014 and concluded that most review articles published during the period focused on eight standard machine learning classification techniques for sentimental analysis along with other concept learning algorithms. Unlike these reviews, the systematic review here conducted focused on all major stand-alone machine learning algorithms, both supervised and unsupervised published during the 2015–2018 period.

1.1.2 Novelty and Review Approach

The systematic review relied on Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) tool to review studies that have used different supervised and unsupervised learning algorithms to address different issues [13]. The approach used in the search was such that different papers published between 2013 and 2018 dealing with the use of machine learning algorithms as methods of data analysis were included. The identification and subsequent inclusion and exclusion of the articles reviewed was based on whether the paper is peer-reviewed, scholarly, full-text, and year of publication that ranges between 2015 and 2018 [13–15]. The search was conducted on EBSCO and ProQuest Central Databases. The search queries used are as follows, and they were implemented in the two databases. In conventional PRISMA review, it is a requirement to check and identify the search criteria in the title and the structure of the abstract alongside introduction (rationale and objectives) and methods including information sources, data items, summary measures, and synthesis results [16]. However, such an approach was adopted, and

Table 1.1 Summary of the queries used to search ProQuest Central and EBSCO databases

Query
(“Machine learning”) AND (Supervised Learning AND Unsupervised Learning)
(“Data mining”) AND (Supervised machine learning algorithms)
(“Supervised Machine Learning”) AND (“Unsupervised Machine Learning”)

applied to published articles instead of being implemented on review articles. Table 1.1 summarizes the search queries that were run in the two databases.

The inclusion criteria deferred for both databases with EBSCO relying on date of publication and full-text to narrow the search, while ProQuest Central search filters included Abstract (AB), Document Text (FT), Document Title (TI), and Publication Title (PUB). An instance of search implemented in ProQuest Central with some of the above criteria is as shown below.

```
ft(Supervised machine learning) AND ft(Unsupervised machine
learning) OR ti(Supervised machine learning) AND ti(Unsupervised
machine learning) OR pub(Supervised machine learning) AND
pub(Unsupervised machine learning)
```

1.2 Search Results

The search and screening results based on PRISMA and elements of meta-analysis are presented in the following section. The major steps used to arrive at the final articles and subsequent analysis included screening (rapid title screening), full test screening, data extraction including extraction of the characteristics of the study, and meta-analysis based on specific check lists and aspects of the machine learning algorithm used.

1.2.1 EBSCO and ProQuest Central Database Results

The search results obtained from the two databases before the commencement of the review process were as follows. The EBSCO search identified 144 articles that were published between 2015 and 2018. Of the 144 documents, 74 had complete information including name of authors, date of publication, name of journal, and structured abstracts. However, only 9 of the 74 articles had full-text and, as such, selected for inclusion in the review process. As for the search results from ProQuest Central, the initial search yielded over 19,898 results, but application of the filters reduced 3301 articles, of which 42 were reviews and 682 covered classification techniques, while 643 covered or had information related to algorithms in general. However, the subject alignment of the research papers was not considered because of the wide spectrum of application of the algorithms such that both supervised

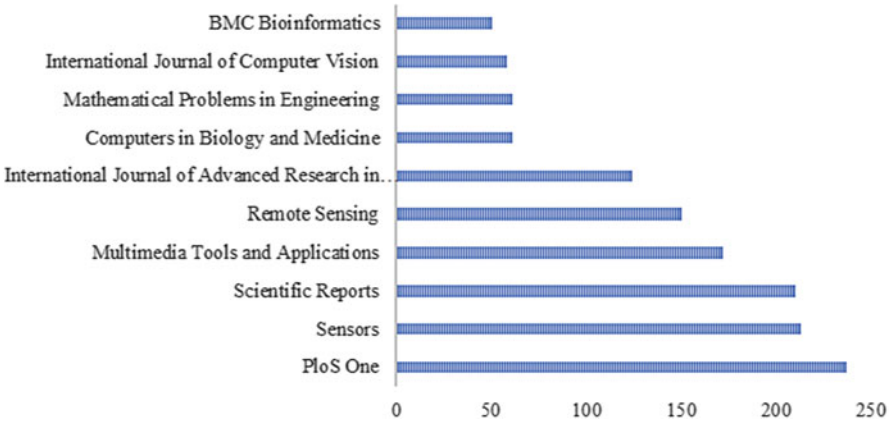


Fig. 1.1 The distribution of ProQuest Central Search Results as per the top ten publication titles (journals)

and unsupervised methods were also applied in other subjects. The distribution the search result based on top ten journals is as shown in Fig. 1.1.

Figure 1.1 shows that PloS One had the highest number of articles published on supervised and unsupervised machine learning. Sensors and Scientific Reports (Nature Publisher Group) had 213 and 210 articles. Multimedia Tools and Applications (172), Remote Sensing (150), and International Journal of Computer Vision (124) had over 100 articles. Even though Mathematics Problems in Engineering and Internal Computer Vision had 61 and 58 articles, the two publications were better placed at exploring the mathematical and algorithmic aspects of supervised and unsupervised machine learning algorithms. The inclusion and exclusion criteria focused on the algorithms as well as their mathematical discourse and application in different fields.

Based on the PRISMA checklist, a total of 84 articles were included in the study and their content analyzed for the implementation of supervised and unsupervised machine learning techniques.

The final number of articles used in the review is 84, although 20 of them underwent meta-analysis when each study was vetted for clarity of the objectives and study questions. Regarding study questions and the effectiveness of the approached used to implement the chosen machine learning algorithms resulted in exclusion of 1290 articles (Fig. 1.2). The rest (1985) met the required study question criteria but also screened for the comprehensiveness of the literature search, data abstraction, evaluation of the results, and the applicability of results [17–19]. It is imperative to note that publication bias and disclosure of funding sources were not considered as part of the screen process. The 84 articles met these meta-analysis requirements and were subsequently included in the analysis (Fig. 1.2).

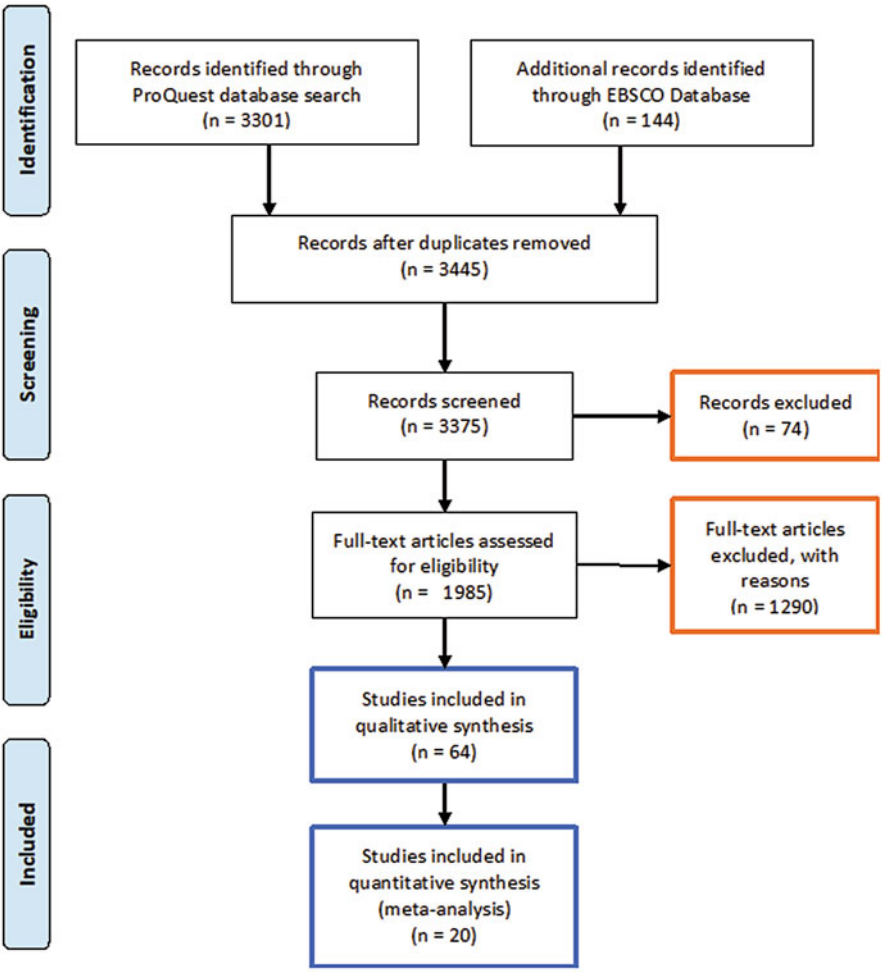


Fig. 1.2 The PRISMA flow diagram for the search conducted on ProQuest Central and EBSCO and the final number of studies included the analysis

It is crucial to note that of the 84 articles that were included in the study, 3 were published in 2013 and 3 were published in 2014 but were not filtered out by the data of publication restriction.

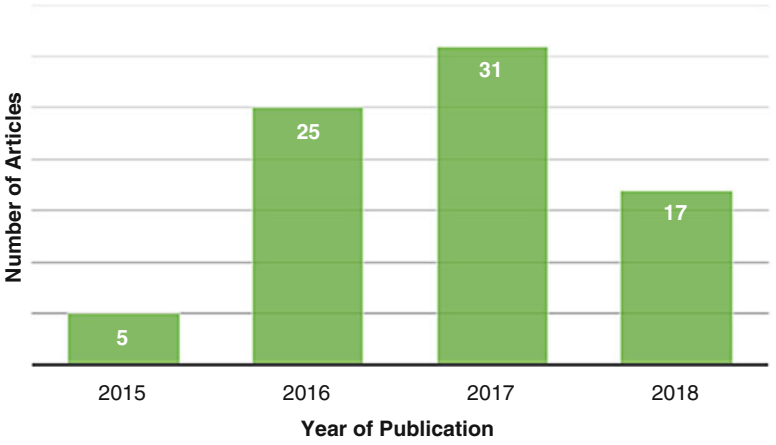


Fig. 1.3 Distribution of articles based on year of publication

1.2.2 Distribution of Included Articles

The articles used in the study consisted of Feature, Journal Articles, General Information, Periodical, and Review types with a distribution represented in the following chart.

From Fig. 1.3, 78 articles were published between 2015 and 2018, while the missing articles were published in 2013 [20–22] and 2014 [23–25] and their inclusion can be associated to publication biasness, which is also observed in the type of documents or study. According to the search, inclusion, and inclusion criteria, the final results ought to have only journal articles, but others were features, general information, periodicals, and reviews. The six papers that were published between 2013 and 2014 were included, because they met all the criteria required for meta-analysis and the indexed meta-data showed that the papers were published in 2015. Regarding the misinformation, we can deduce that the publications had an inaccuracy of about 7.2%.

1.3 Discussion

The 84 articles discussed different supervised and unsupervised machine learning techniques without necessarily making the distinction. According to Praveena [11], supervised learning requires an assistance born out of experience or acquired patterns within the data and, in most cases, involves a defined output variable [26–30]. The input dataset is segregated into train and test subsets, and several papers address the concept of training datasets based on the desired outcome [31–34]. All the algorithms that use supervised learning approach acquire patterns within the

training dataset and subsequently apply them to the test subset with the object of either predicting or classifying an attribute [35–37]. Most of the authors described the workflow of a supervised machine learning and, as it also emerged from the review, decision tree, Naïve Bayes, and Support Vector Machines are the most commonly used algorithms [8, 38–42].

1.3.1 Decision Tree

It is important to recall that supervised learning can either be based on a classification or regression algorithm, and decision tree algorithm can be used as both although it is mainly used for classification as noted in these articles [20, 43–45]. The algorithm emulates a tree, and it sorts attributes through groupings based on data values [46]. Just like a conventional tree, the algorithm has branches and nodes with nodes representing variable group for classification and branches, assuming the values that the attribute can take as part of the class [47, 48]. The pseudocode illustrating the decision tree algorithm is as shown below. In the algorithm, D is the dataset, while x and y are the input and target variables, respectively [49, 50].

Algorithm 1.1: Decision Tree

Protocol *DT Inducer* (D, x, y)

1. $\boldsymbol{T = Tree\ Growing\ (D, x, y)}$
2. Return Tree Pruning (D, T)

Method Tree Growing (D, x, y)

1. Create a tree T
2. **if** at least one of the Stopping Criteria is satisfied **then**;
3. label the root node as a leaf with the most frequent value of y in D as the correct class.
4. **else**;
5. Establish a discrete function f(x) of the input variable so that splitting D according to the functions outcomes produces the best splitting metric
6. **if** the best metric is greater or equal to the threshold **then**;
7. Mark the root node in T as f(x)
8. **for** each outcome of f(x) at the node **do**;
9. $\boldsymbol{Subtree = Tree\ Growing\ \left(\delta_{f(x)=t_1}, D, x, y\right)}$
10. Connect the root of T to Subtree and label the edge $\boldsymbol{t_1}$
11. **end for**
12. **else**
13. Label the root node T for a leaf with the frequent value of y in D as the assigned class
14. **end if**

15. **end if**
16. Return T

Protocol *Tree Pruning* (D, T, y)

1. **repeat**
2. Select a node t in T to maximally improve pruning evaluation procedure
3. **if** $t \neq 0$ **then**;
4. $\boldsymbol{T} = \boldsymbol{pruned}\ (\boldsymbol{T}, \boldsymbol{t})$
5. **end if**
6. **until** $t = 0$
7. Return T

As illustrated in the pseudocode, Decision Tree achieves classification in three distinct steps. Firstly, the algorithm induces both tree growing and tree pruning functionalities [51]. Secondly, it grows the tree by assigning each data value to a class based on the value of the target variable that is the most common one at the instance of iteration [52, 53]. The final step deals with pruning the grown tree to optimize the performance of the resultant model [19, 53, 54]. Most of the reviewed studies involved application of decision trees for different applications, although most involved classification cancer and lung cancer studies, clinical medicine especially diagnosis of conditions based on historical data as well as some rare forms of artificial intelligence applications [40, 52, 55–57]. Most of the studies have also recognized decision tree algorithms to be more accurate when dealing with data generated using the same collection procedures [43, 44, 52].

1.3.2 Naïve Bayes

The Naïve Bayes algorithm has gained its fame because of its background on Bayesian probability theorem. In most texts, it is considered a semisupervised method, because it can be used either in clustering or classification tasks [58, 59]. When implemented as a technique for creating clusters, Naïve Bayes does not require specification of an outcome and it uses conditional probability to assign data values to classes and, as such, is a form of unsupervised learning [47, 60–62]. However, when used to classify data, Naïve Bayes requires both input and target variables and, as such, is a supervised learning technique [55, 63, 64]. As a classifier, the algorithm creates Bayesian networks, which are tree generated based on the condition probability of an occurrence of an outcome based on probabilities imposed on it by the input variables [65, 66]. The pseudocode for the Naïve Bayes algorithm is presented below [49, 67, 68].

Algorithm 1.2: Naïve Bayes Learner

Input: training set $\boldsymbol{T_s}$, Hold-out set $\boldsymbol{H_s}$, initial components, $\boldsymbol{I_c}$, and convergence thresholds $\boldsymbol{\rho_{EM}}$ and $\boldsymbol{\rho_{add}}$

Initial M using one component
$\boldsymbol{I} \leftarrow \boldsymbol{I_c}$.
repeat
Add I components to M thereby initializing M using random components drawn from the training set $\boldsymbol{T_s}$
Remove the I initialization instances from $\boldsymbol{T_s}$
repeat
E-step: Proportionally assign examples in $\boldsymbol{T_s}$ to resultant mixture component using M
M-Step: Calculate maximum likelihood parameters using the input data.
if ***log P*** $(\boldsymbol{H_s/M})$ is the best maximum probability, then save M in $\boldsymbol{M_{best}}$
every 5 cycles of the two steps, prune low-weight components of M
until P $(\boldsymbol{H_s/M})$ fails to increase by the ratio $\boldsymbol{\rho_{EM}}$
M$\leftarrow\boldsymbol{M_{best}}$
Prune low weight components of M
$\boldsymbol{I} \leftarrow \boldsymbol{2I}$.
until P $(\boldsymbol{H_s/M})$ fails to increase by the ratio $\boldsymbol{\rho_{add}}$
Execute both E: step and M: step twice on $\boldsymbol{M_{best}}$ using examples from $\boldsymbol{H_s}$ and $\boldsymbol{T_s}$
Return M$\leftarrow\boldsymbol{M_{best}}$

As the pseudocode illustrates, Naïve Bayes algorithm relies on Bayes' theorem represented mathematical below to assign independent variables to classes based on probability [31, 58].

$$P\,(H|D) = \frac{P(H)P\,(D|H)}{P(D)} \tag{1.1}$$

In Eq. (1.1), the probability of H when the probability of D is known is defined in terms of the product probability of H, probability of D given the probability of H divided by the probability of D. The H and D are events with defined outcome and they can represent Heads and Tails in coil tossing experiments [12, 45, 69, 70]. The extension of the theorem in supervised learning is of the form represented in Eq. (1.2).

$$P\,(H|D) = P\,(x_i, \ldots, x_n|H) = \prod_i P\,(x_i|H) \tag{1.2}$$

In the above equation, $x_i, \ldots, x_n$ represents the input attribute, for which conditional probabilities are computed based on the known probabilities of the target variables in the training dataset [71–73]. The algorithm has been discussed in different contexts and its application is mainly attributed to the creation of data labels for subsequent unsupervised learning verifications [16, 74, 75].

1.3.3 Support Vector Machine

The support vector machines (SVMs) algorithm was also common among the search results articles. The articles that explored the applications of SVM did so with the objective of evaluating its performance in different scenarios [30, 58, 73, 76]. All the applications of SVM are included toward classification and the tenet of the algorithm is computation of margins [53, 77, 78]. Simply, SVM draws margins as boundary between the classes in the provided dataset. Its principle is to create the margins such that the distance between each class and the nearest margin is maximized and in effect leading to the minimum possible classification error [68, 78–80]. The margins are defined as the distance between two supporting vectors separated by a hyperplane. The pseudocode for the SVM algorithm is as demonstrated below. The algorithm assumes that the data are linearly separable so that the weight associated with support vectors can be drawn easily and the margin computed [62, 70]. The assumption makes regularization possible [49, 81].

Algorithm 1.3: Support Vector Machine

Input: S, λ, T, k
Initialize: Choose $\boldsymbol{w_1}$ such that $\|\boldsymbol{w_1}\| \leq \sqrt{\boldsymbol{\lambda}}$
FOR $\boldsymbol{t = 1,2 \ldots ,T}$
Select $\boldsymbol{A_t \subseteq S}$, *in which* $\boldsymbol{|A_t| = k}$
Set $\boldsymbol{A_t^+ = \{(x, y) \in A_t : y\,(w_t, x) < 1\}}$
Set $\boldsymbol{\delta_t = \frac{1}{\lambda t}}$
Set $\boldsymbol{w_{t+0.5} = (1 - \delta_t \lambda)\, w_t + \frac{\delta_t}{k} \sum_{(x,y) \in A_t^+} yx}$
Set $\boldsymbol{w_{t+1} = \left\{1, \frac{1/\sqrt{\lambda}}{\|w_{t+0.5}\|}\right\} w_{t+0.5}}$
Output: $\boldsymbol{w_{T+1}}$

The implementation of the algorithm and its accuracy is dependent on its ability to margin violations and subsequent misclassification of classes on either side of the vectors. The margin is based on the following set of equations:

$$\begin{aligned} W^{\mathrm{T}}x + b &= 1 \\ W^{\mathrm{T}}x + b &= 0 \\ W^{\mathrm{T}}x + b &= -1 \end{aligned} \tag{1.3}$$

In Eq. (1.3), the three sets of equation describe the hyperplane separating two linear support vectors $W^{T}x + b = 1$ and $W^{T}x + b = -1$, and all the classes within the two support vectors are classified accurately, while those outside the support vectors violate the margin [25, 81, 82]. Consequently, the larger the distance between the support vectors, the higher the chances that points are correctly classified.

As for unsupervised learning algorithms, most of the studies either discussed, cited, or implemented k-means, hierarchical clustering, and principal component analysis, among others [20, 55, 73, 83, 84]. Unlike supervised learning, unsupervised learning extract limited features from the data, and it relies on previously learned patterns to recognize likely classes within the dataset [85, 86]. As a result, unsupervised learning is suitable for feature reduction in case of large dataset and clustering tasks that lead to the creation of new classes in unlabeled data [80, 87, 88]. It entails selection and importation of data into appropriate framework followed by selection of an appropriate algorithm, specification of thresholds, review of the model, and subsequent optimization to produce desired outcome [89, 90]. Of the many unsupervised learners, k-means was widely discussed among the authors and as such was also previewed in the review.

1.3.4 k-Means Algorithms

The algorithm has been used in different studies to create groups or classes in unlabeled datasets based on the mean distance between classes [91, 92]. The technique initiates and originates the classes or labels that are subsequently used in other prospective analysis [69]. A pseudocode for the k-means algorithm is as shown in the illustration below [15, 61].

Algorithm 1.4: k-Means Learner

Function k-means ()

Initialize k prototypes $(\boldsymbol{w_1} \ldots, \boldsymbol{w_k})$ so that the weighted distance between the clusters becomes $\boldsymbol{w_j} = \boldsymbol{i_l}, \boldsymbol{j} \in \{\mathbf{1}, \ldots, \boldsymbol{k}\}, \boldsymbol{l} \in \{\mathbf{1}, \ldots, \boldsymbol{n}\}$

Associate each cluster $\boldsymbol{C_j}$ with the prototype weight $\boldsymbol{w_j}$

Repeat

for each input vector $\boldsymbol{i_l}, \boldsymbol{l} \in \{\mathbf{1}, \ldots, \boldsymbol{n}\}$

do

Assign $\boldsymbol{i_l}$ to cluster $\boldsymbol{C_{j_*}}$ with the nearest $\boldsymbol{w_{j_*}}$

for each cluster $\boldsymbol{C_{j_*}}: \boldsymbol{j} \in \{\mathbf{1}, \ldots, \boldsymbol{k}\}$, **do**;

Update the prototype $\boldsymbol{w_j}$ to be centroid of the sample observations in the current $\boldsymbol{C_{j_*}}$; $\boldsymbol{w_j} = \sum_{\boldsymbol{i_l} \in \boldsymbol{c_j}} \boldsymbol{i_l} / \left| \boldsymbol{C_j} \right|$

Calculate the error function

$$E = \sum_{j=1}^{k} \sum_{i_l \in C_j} \left| i_l - w_j \right|^2$$

until E becomes constant or does not change significantly.

The pseudocode demonstrates the process of assigning data values to classes based on their proximity to the nearest mean with the least error function [93–96]. The error function is computed as the difference between the mean and the assigned cluster mean [97, 98].

1.3.5 Semisupervised and Other Learners

Even though the search was focused and narrowed down to supervised and unsupervised learning techniques, it emerged that research preferred using different methods for the purposes of comparing the results and verification of the classification and prediction accuracy of the machine learning models [75, 99, 100]. Some of the studies used supervised and unsupervised machine learning approaches alongside reinforcement learning techniques such as generative models, self-training algorithms, and transudative SVM [101–103]. Other studies focused on ensemble learning algorithms such as boosting and bagging, while other studies defined different perceptions related to neural networks. [59, 66, 104–107]. Finally, some of the studies addressed algorithms such as k-Nearest Neighbor as an instance-based learning but could not categorize it as either supervised or unsupervised machine learning algorithm because of the limitations of the applications [41, 108–110].

1.4 Conclusion and Future Work

Even though the search results yielded over 3300 qualified papers, the filtering processes based on title screening, abstract screening, full text screening, and data extraction coupled with meta-analysis reduced the number of articles to 84. Despite the narrowing the search results to supervised and unsupervised machine learning as key search words, the results contained articles that addressed reinforced learners and ensembled learners among other techniques that review did not focus. The trend is understandable, because machine learning and data science is evolving and most of the algorithms are undergoing improvements, hence the emergence of categories such as reinforced and ensembled learner. Hence, future systematic review prospect should focus on these emerging aggregations of learners and assess through research progress based on authorship, regions, and applications to identify the major driving forces behind the growth.

References

1. Sandhu, T. H. (2018). Machine learning and natural language processing—A review. *International Journal of Advanced Research in Computer Science, 9*(2), 582–584.
2. Libbrecht, M. W., & Noble, W. S. (2015). Machine learning applications in genetics and genomics. *Nature Reviews Genetics, 16*(6), 321–332.
3. Alpaydın, E. (2014). *Introduction to machine learning*. Cambridge, MA: MIT Press.
4. Kotsiantis, S. B. (2007). Supervised machine learning: A review of classification techniques. *Informatica, 31*, 249–268.
5. MathWorks. (2016). *Applying supervised learning*. Machine Learning with MATLAB.
6. Ng, A. (2012). 1. Supervised learning. *Machine Learning*, 1–30.
7. Hofmann, T. (2001). Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning, 42*, 177–196.
8. Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *Machine Learning Proceedings*.
9. Marshland, S. (2015). *Machine learning: An algorithm perspective*. Boca Raton, FL: CRC Press.
10. Baharudin, B., Lee, L. H., & Khan, K. (2010). A review of machine learning algorithms for text-documents classification. *Journal on Advance in Information Technology, 1*(1), 4–20.
11. Praveena, M. (2017). A literature review on supervised machine learning algorithms and boosting process. *International Journal of Computer Applications, 169*(8), 975–8887.
12. Qazi, A., Raj, R. G., Hardaker, G., & Standing, C. (2017). A systematic literature review on opinion types and sentiment analysis techniques: Tasks and challenges. *Internet Research, 27*(3), 608–630.
13. Hutton, B., et al. (2015). The PRISMA extension statement for reporting of systematic reviews incorporating network meta-analyses of health care interventions: Checklist and explanations. *Annals of Internal Medicine, 163*(7), 566–567.
14. Zorzela, L., Loke, Y. K., Ioannidis, J. P., Golder, S., Santaguida, P., Altman, D. G., et al. (2016). PRISMA harms checklist: Improving harms reporting in systematic reviews. *BMJ (Online), 352*, i157.
15. Shamseer, L., et al. (2015). Preferred reporting items for systematic review and meta-analysis protocols (prisma-p) 2015: Elaboration and explanation. *BMJ (Online), 349*, g7647.
16. Moher, D., et al. (2015). Preferred reporting items for systematic review and meta-analysis protocols (PRISMA-P) 2015 statement. *Systematic Reviews, 4*, 1.
17. Stroup, D. F., et al. (2000). Meta-analysis of observational studies in epidemiology: A proposal for reporting. Meta-analysis Of Observational Studies in Epidemiology (MOOSE) group. *JAMA, 283*(15), 2008–2012.
18. Bloch, M. H., Landeros-Weisenberger, A., Rosario, M. C., Pittenger, C., & Leckman, J. F. (2008). Meta-analysis of the symptom structure of obsessive-compulsive disorder. *The American Journal of Psychiatry, 165*(12), 1532–1542.
19. Fujimoto, M. S., Suvorov, A., Jensen, N. O., Clement, M. J., & Bybee, S. M. (2016). Detecting false positive sequence homology: A machine learning approach. *BMC Bioinformatics, 17*, 101.
20. Mani, S., et al. (2013). Machine learning for predicting the response of breast cancer to neoadjuvant chemotherapy. *Journal of the American Medical Informatics Association, 20*(4), 688–695.
21. Kovačević, A., Dehghan, A., Filannino, M., Keane, J. A., & Nenadic, G. (2013). Combining rules and machine learning for extraction of temporal expressions and events from clinical narratives. *Journal of the American Medical Informatics Association, 20*(5), 859–866.
22. Klann, J. G., Anand, V., & Downs, S. M. (2013). Patient-tailored prioritization for a pediatric care decision support system through machine learning. *Journal of the American Medical Informatics Association, 20*(e2), e267–e274.

23. Gultepe, E., Green, J. P., Nguyen, H., Adams, J., Albertson, T., & Tagkopoulos, I. (2014). From vital signs to clinical outcomes for patients with sepsis: A machine learning basis for a clinical decision support system. *Journal of the American Medical Informatics Association, 21*(2), 315–325.
24. Mani, S., et al. (2014). Medical decision support using machine learning for early detection of late-onset neonatal sepsis. *Journal of the American Medical Informatics Association, 21*(2), 326–336.
25. Nguyen, D. H. M., & Patrick, J. D. (2014). Supervised machine learning and active learning in classification of radiology reports. *Journal of the American Medical Informatics Association, 21*(5), 893–901.
26. Deo, R. C. (2015). Machine learning in medicine HHS public access. *Circulation, 132*(20), 1920–1930.
27. Mullainathan, S., & Spiess, J. (2017). Machine learning: An applied econometric approach. *The Journal of Economic Perspectives, 31*(2), 87–106.
28. Wu, M.-J., et al. (2017). Identification and individualized prediction of clinical phenotypes in bipolar disorders using neurocognitive data, neuroimaging scans and machine learning. *NeuroImage, 145*, 254–264.
29. Oudah, M., & Henschel, A. (2018). Taxonomy-aware feature engineering for microbiome classification. *BMC Bioinformatics, 19*, 227.
30. Palma, S. I. C. J., Traguedo, A. P., Porteira, A. R., Frias, M. J., Gamboa, H., & Roque, A. C. A. (2018). Machine learning for the meta-analyses of microbial pathogens' volatile signatures. *Scientific Reports, 8*, 1–15.
31. Jaspers, S., De Troyer, E., & Aerts, M. (2018). Machine learning techniques for the automation of literature reviews and systematic reviews in EFSA. *EFSA Supporting Publications, 15*(6), 1427E.
32. Crawford, M., Khoshgoftaar, T. M., Prusa, J. D., Richter, A. N., & Al Najada, H. (2015). Survey of review spam detection using machine learning techniques. *Journal of Big Data, 2*(1), 1–24.
33. Dinov, I. D. (2016). Methodological challenges and analytic opportunities for modeling and interpreting Big Healthcare Data. *Gigascience, 5*, 12.
34. Dimou, A., Vahdati, S., Di Iorio, A., Lange, C., Verborgh, R., & Mannens, E. (2017). Challenges as enablers for high quality Linked Data: Insights from the Semantic Publishing Challenge. *PeerJ Computer Science, 3*, e105.
35. Trilling, D., & Boumans, J. (2018). Automatische inhoudsanalyse van Nederlandstalige data. *Tijdschrift voor Communicatiewetenschap, 46*(1), 5–24.
36. Van Nieuwenburg, E. P. L., Liu, Y., & Huber, S. D. (2017). Learning phase transitions by confusion. *Nature Physics, 13*(5), 435–439.
37. Hoyt, R., Linnville, S., Thaler, S., & Moore, J. (2016). Digital family history data mining with neural networks: A pilot study. *Perspectives in Health Information Management, 13*, 1c.
38. Dobson, J. E. (2015). Can an algorithm be disturbed? Machine learning, intrinsic criticism, and the digital humanities. *College Literature, 42*(4), 543–564.
39. Downing, N. S., et al. (2017). Describing the performance of U.S. hospitals by applying big data analytics. *PLoS One, 12*(6), e0179603.
40. Hoang, X. D., & Nguyen, Q. C. (2018). Botnet detection based on machine learning techniques using DNS query data. *Future Internet, 10*(5), 43.
41. Kothari, U. C., & Momayez, M. (2018). Machine learning: A novel approach to predicting slope instabilities. *International Journal of Geophysics, 2018*, 9.
42. Thompson, J. A., Tan, J., & Greene, C. S. (2016). Cross-platform normalization of microarray and RNA-seq data for machine learning applications. *PeerJ, 4*, e1621.
43. Ahmed, M. U., & Mahmood, A. (2018). An empirical study of machine learning algorithms to predict students' grades. *Pakistan Journal of Science, 70*(1), 91–96.
44. Carifio, J., Halverson, J., Krioukov, D., & Nelson, B. D. (2017). Machine learning in the string landscape. *Journal of High Energy Physics, 2017*(9), 1–36.

45. Choudhari, P., & Dhari, S. V. (2017). Sentiment analysis and machine learning based sentiment classification: A review. *International Journal of Advanced Research in Computer Science, 8*(3).
46. Lloyd, S., Garnerone, S., & Zanardi, P. (2016). Quantum algorithms for topological and geometric analysis of data. *Nature Communications, 7*, 10138.
47. Pavithra, D., & Jayanthi, A. N. (2018). A study on machine learning algorithm in medical diagnosis. *International Journal of Advanced Research in Computer Science, 9*(4), 42–46.
48. Krittanawong, C., Zhang, H., Wang, Z., Aydar, M., & Kitai, T. (2017). Artificial intelligence in precision cardiovascular medicine. *Journal of the American College of Cardiology, 69*(21), 2657–2664.
49. Kaytan, M., & Aydilek, I. B. (2017). A review on machine learning tools. *2017 International Artificial Intelligence and Data Processing Symposium, 8*(3), 1–4.
50. Lynch, C. M., van Berkel, V. H., & Frieboes, H. B. (2017). Application of unsupervised analysis techniques to lung cancer patient data. *PLoS One, 12*(9), e0184370.
51. Beck, D., Pfaendtner, J., Carothers, J., & Subramanian, V. (2017). Data science for chemical engineers. *Chemical Engineering Progress, 113*(2), 21–26.
52. Heylman, C., Datta, R., Sobrino, A., George, S., & Gratton, E. (2015). Supervised machine learning for classification of the electrophysiological effects of chronotropic drugs on human induced pluripotent stem cell-derived cardiomyocytes. *PLoS One, 10*(12), e0144572.
53. Torkzaban, B., et al. (2015). Machine learning based classification of microsatellite variation: An effective approach for Phylogeographic characterization of olive populations. *PLoS One, 10*(11), e0143465.
54. Guo, Z., Shao, X., Xu, Y., Miyazaki, H., Ohira, W., & Shibasaki, R. (2016). Identification of village building via Google earth images and supervised machine learning methods. *Remote Sensing, 8*(4), 271.
55. Xia, C., Fu, L., Liu, Z., Liu, H., Chen, L., & Liu, Y. (2018). Aquatic toxic analysis by monitoring fish behavior using computer vision: A recent progress. *Journal of Toxicology, 2018*, 11.
56. Fuller, D., Buote, R., & Stanley, K. (2017). A glossary for big data in population and public health: Discussion and commentary on terminology and research methods. *Journal of Epidemiology and Community Health, 71*(11), 1113.
57. Gibson, D., & de Freitas, S. (2016). Exploratory analysis in learning analytics. *Technology, Knowledge and Learning, 21*(1), 5–19.
58. Cuperlovic-Culf, M. (2018). Machine learning methods for analysis of metabolic data and metabolic pathway modeling. *Metabolites, 8*(1), 4.
59. Tan, M. S., Chang, S.-W., Cheah, P. L., & Yap, H. J. (2018). Integrative machine learning analysis of multiple gene expression profiles in cervical cancer. *PeerJ, 6*, e5285.
60. Meenakshi, K., Safa, M., Karthick, T., & Sivaranjani, N. (2017). A novel study of machine learning algorithms for classifying health care data. *Research Journal of Pharmacy and Technology, 10*(5), 1429–1432.
61. Dey, A. (2016). Machine learning algorithms: A review. *International Journal of Computer Science and Information Technology, 7*(3), 1174–1179.
62. Zhao, C., Wang, S., & Li, D. (2016). Determining fuzzy membership for sentiment classification: A three-layer sentiment propagation model. *PLoS One, 11*(11), e0165560.
63. Mossotto, E., Ashton, J. J., Coelho, T., Beattie, R. M., MacArthur, B. D., & Ennis, S. (2017). Classification of paediatric inflammatory bowel disease using machine learning. *Scientific Reports, 7*, 1–10.
64. Lau, O., & Yohai, I. (2016). Using quantitative methods in industry. *Political Science and Politics, 49*(3), 524–526.
65. Qiu, J., Wu, Q., Ding, G., Xu, Y., & Feng, S. (2016). A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing, 2016*, 1–16.
66. Parreco, J. P., Hidalgo, A. E., Badilla, A. D., Ilyas, O., & Rattan, R. (2018). Predicting central line-associated bloodstream infections and mortality using supervised machine learning. *Journal of Critical Care, 45*, 156–162.

67. Wuest, T., Irgens, C., & Thoben, K.-D. (2016). Changing states of multistage process chains. *Journal of Engineering, 2016*, 1.
68. Tarwani, N. (2017). Survey of cyberbulling detection on social media big-data. *International Journal of Advanced Research in Computer Science, 8*(5).
69. Martinelli, E., Mencattini, A., Daprati, E., & Di Natale, C. (2016). Strength is in numbers: Can concordant artificial listeners improve prediction of emotion from speech? *PLoS One, 11*(8), e0161752.
70. Liu, N., & Zhao, J. (2016). Semi-supervised online multiple kernel learning algorithm for big data. *TELKOMNIKA, 14*(2), 638–646.
71. Goh, K. L., & Singh, A. K. (2015). Comprehensive literature review on machine learning structures for Web spam classification. *Procedia Computer Science, 70*, 434–441.
72. Mishra, C., & Gupta, D. L. (2017). Deep machine learning and neural networks: An overview. *IAES International Journal of Artificial Intelligence, 6*(2), 66–73.
73. Yan, X., Bai, Y., Fang, S., & Luo, J. (2016). A kernel-free quadratic surface support vector machine for semi-supervised learning. *The Journal of the Operational Research Society, 67*(7), 1001–1011.
74. Yared, R., & Abdulrazak, B. (2016). Ambient technology to assist elderly people in indoor risks. *Computers, 5*(4), 22.
75. Osborne, J. D., et al. (2016). Efficient identification of nationally mandated reportable cancer cases using natural language processing and machine learning. *Journal of the American Medical Informatics Association, 83*(5), 605–623.
76. Kolog, E. A., Montero, C. S., & Tukiainen, M. (2018). Development and evaluation of an automated e-counselling system for emotion and sentiment analysis. *Electronic Journal of Information Systems Evaluation, 21*(1), 1–19.
77. Rafiei, M. H., Khushefati, W. H., Demirboga, R., & Adeli, H. (2017). Supervised deep restricted Boltzmann machine for estimation of concrete. *ACI Materials Journal, 114*(2), 237–244.
78. Almasre, M. A., & Al-Nuaim, H. (2017). Comparison of four SVM classifiers used with depth sensors to recognize Arabic sign language words. *Computers, 6*(2), 20.
79. Hashem, K. (2018). The rise and fall of machine learning methods in biomedical research. *F1000Research, 6*, 2012.
80. Torshin, I. Y., & Rudakov, K. V. (2015). On the theoretical basis of metric analysis of poorly formalized problems of recognition and classification. *Pattern Recognition and Image Analysis, 25*(4), 577–587.
81. Petrelli, M., & Perugini, D. (2016). Solving petrological problems through machine learning: The study case of tectonic discrimination using geochemical and isotopic data. *Contributions to Mineralogy and Petrology, 171*(10), 1–15.
82. Min-Joo, K., & Kang, J.-W. (2016). Intrusion detection system using deep neural network for in-vehicle network security. *PLoS One, 11*(6). https://doi.org/10.1371/journal.pone.0155781
83. Alicante, A., Corazza, A., Isgrò, F., & Silvestri, S. (2016). Unsupervised entity and relation extraction from clinical records in Italian. *Computers in Biology and Medicine, 72*, 263–275.
84. Shanmugasundaram, G., & Sankarikaarguzhali, G. (2017). An investigation on IoT healthcare analytics. *International Journal of Information Engineering and Electronic Business, 9*(2), 11.
85. Huang, G., Song, S., Gupta, J. N. D., & Wu, C. (2014). Semi-supervised and unsupervised extreme learning machines. *IEEE Transactions on Cybernetics, 44*(12), 2405–2417.
86. Rastogi, R., & Saigal, P. (2017). Tree-based localized fuzzy twin support vector clustering with square loss function. *Applied Intelligence, 47*(1), 96–113.
87. Muscoloni, A., Thomas, J. M., Ciucci, S., Bianconi, G., & Cannistraci, C. V. (2017). Machine learning meets complex networks via coalescent embedding in the hyperbolic space. *Nature Communications, 8*, 1–19.
88. Saeys, Y., Van Gassen, S., & Lambrecht, B. N. (2016). Computational flow cytometry: Helping to make sense of high-dimensional immunology data. *Nature Reviews. Immunology, 16*(7), 449–462.

89. Gonzalez, A., Pierre, & Forsberg, F. (2017). *Unsupervised machine learning: An investigation of clustering algorithms on a small dataset* (pp. 1–39).
90. Necula, S.-C. (2017). Deep learning for distribution channels' management. *Informatica Economică, 21*(4), 73–85.
91. Munther, A., Razif, R., AbuAlhaj, M., Anbar, M., & Nizam, S. (2016). A preliminary performance evaluation of K-means, KNN and em unsupervised machine learning methods for network flow classification. *International Journal of Electrical and Computer Engineering, 6*(2), 778–784.
92. Alalousi, A., Razif, R., Abualhaj, M., Anbar, M., & Nizam, S. (2016). A preliminary performance evaluation of K-means, KNN and EM unsupervised machine learning methods for network flow classification. *International Journal of Electrical and Computer Engineering, 6*(2), 778–784.
93. Alanazi, H. O., Abdullah, A. H., & Qureshi, K. N. (2017). A critical review for developing accurate and dynamic predictive models using machine learning methods in medicine and health care. *Journal of Medical Systems, 41*(4), 1–10.
94. Almatarneh, S., & Gamallo, P. (2018). A lexicon based method to search for extreme opinions. *PLoS One, 13*(5), e0197816.
95. Assem, H., Xu, L., Buda, T. S., & O'sullivan, D. (2016). Machine learning as a service for enabling Internet of things and people. *Personal and Ubiquitous Computing, 20*(6), 899–914.
96. Azim, M. A., & Bhuiyan, M. H. (2018). Text to emotion extraction using supervised machine learning techniques. *TELKOMNIKA, 16*(3), 1394–1401.
97. Sirbu, A. (2016). Dynamic machine learning for supervised and unsupervised classification ES. *Machine Learning*.
98. Wahyudin, I., Djatna, T., & Kusuma, W. A. (2016). Cluster analysis for SME risk analysis documents based on pillar K-means. *TELKOMNIKA, 14*(2), 674.
99. Davis, S. E., Lasko, T. A., Chen, G., Siew, E. D., & Matheny, M. E. (2018). Calibration drift in regression and machine learning models for acute kidney injury. *Journal of the American Medical Informatics Association, 24*, 1052–1061.
100. Wallace, B. C., Noel-Storr, A., Marshall, I. J., Cohen, A. M., Smalheiser, N. R., & Thomas, J. (2017). Identifying reports of randomized controlled trials (RCTs) via a hybrid machine learning and crowdsourcing approach. *Journal of the American Medical Informatics Association, 24*(6), 1165–1168.
101. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., & Lloyd, S. (2017). Quantum machine learning. *Nature, 549*(7671), 195–202.
102. Bisaso, K. R., Anguzu, G. T., Karungi, S. A., Kiragga, A., & Castelnuovo, B. (2017). A survey of machine learning applications in HIV clinical research and care. *Computers in Biology and Medicine, 91*, 366–371.
103. Bauder, R., Khoshgoftaar, T. M., & Seliya, N. (2017). A survey on the state of healthcare upcoding fraud analysis and detection. *Health Services and Outcomes Research Methodology, 17*(1), 31–55.
104. Bashiri, A., Ghazisaeedi, M., Safdari, R., Shahmoradi, L., & Ehtesham, H. (2017). Improving the prediction of survival in cancer patients by using machine learning techniques: Experience of gene expression data: A narrative review. *Iranian Journal of Public Health, 46*(2), 165–172.
105. Breckels, L. M., Mulvey, C. M., Lilley, K. S., & Gatto, L. (2018). A bioconductor workflow for processing and analysing spatial proteomics data. *F1000Research, 5*, 2926.
106. Saad, S. M., et al. (2017). Pollutant recognition based on supervised machine learning for indoor air quality monitoring systems. *Applied Sciences, 7*(8), 823.
107. Fiorini, L., Cavallo, F., Dario, P., Eavis, A., & Caleb-Solly, P. (2017). Unsupervised machine learning for developing personalised behaviour models using activity data. *Sensors, 17*(5), 1034.

108. Bunn, J. K., Hu, J., & Hattrick-Simpers, J. R. (2016). Semi-supervised approach to phase identification from combinatorial sample diffraction patterns. *JOM, 68*(8), 2116–2125.
109. Cárdenas-López, F. A., Lamata, L., Retamal, J. C., & Solano, E. (2018). Multiqubit and multilevel quantum reinforcement learning with quantum technologies. *PLoS One, 13*(7), e0200455.
110. Chen, R., Niu, W., Zhang, X., Zhuo, Z., & Lv, F. (2017). An effective conversation-based botnet detection method. *Mathematical Problems in Engineering, 2017*, 4934082.

Chapter 2
Overview of One-Pass and Discard-After-Learn Concepts for Classification and Clustering in Streaming Environment with Constraints

Chidchanok Lursinsap

2.1 Introduction

Tremendous amount of data have been rapidly generated since the advancement of internet technology. Achieving the faster learning speed of these streaming data than the speed of data generation with limited memory size is a very challenging problem. The term *learning* in our context refers to both classification and clustering. In addition to this learning speed race, the streaming environment induced the problem of dynamic data imbalance ratio and class drift in the classification domain [3, 10]. But for clustering, there are no problem of dynamic imbalance ratio. Only the special class drift in forms of expired data is concerned. The size of learning data chunk at any time is a priori unknown to the learning process. There have been several attempts proposed to handle this classification and clustering challenges.

Most proposed methods for solving streaming data learning are based on the concept of incremental learning where a neuron is gradually added to the network to improve the accuracy [1, 7, 9, 15, 18, 25, 27, 28]. This incremental learning is simple but it encounters the problem of data overflow memory. New incoming data must be mixed with the previously learned data. Consequently, the learning speed significantly drops when the amount of data temporally increases. Other constraints such as arbitrary class drift, expired data in clusters, and dynamic imbalance ratio have not been fully included as a part learning algorithm. These constraints actually occur in various applications [21].

C. Lursinsap (✉)
Advanced Virtual and Intelligent Computing Center, Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University, Bangkok, Thailand

The Royal Institute of Thailand, Bangkok, Thailand

M. W. Berry et al. (eds.), *Supervised and Unsupervised Learning for Data Science*,
Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-030-22475-2_2

In this paper, we provided the overview of *one-pass learning* and *discard-after-learn* concept introduced in our previous series of studies [11, 12, 14, 20, 21, 23, 24]. These two concepts were designed to solve the problems of classification and clustering of streaming data with the constraints on data overflow, arbitrary class drift, dynamic imbalance ratio, data expiration. Moreover, a set of new mathematical shapes and new recursive functions for updating the parameters of these shapes were proposed.

The rest of this paper is organized into nine sections. Section 2.2 defines the constraints and conditions of the learning environment. Section 2.3 summarizes the concept of one-pass and discard-after-learn for classification and clustering. Section 2.4 discusses the structure of malleable hyper-ellipsoid function. Section 2.5 explains how to update the parameters of malleable hyper-ellipsoid. Section 2.6 analyzes the time and space complexities for computing the proposed functions. Section 2.7 introduces a new network structure called *dynamic stratum* structure to solve arbitrary concept drift problem. Section 2.8 proposed a new hyper-cylindrical structure to solve data expiration in dynamic clustering. Section 2.9 discusses the advantages and disadvantages of the proposed concept and network structure. Section 2.10 concludes the paper.

2.2 Constraints and Conditions

This study concerns classification and clustering learning in streaming data environment. Both types of learning involve the following similar and different sets of constraints and conditions.

Classification Generally, there are two different environments of training data. The first environment regards the stationary characteristics of training data. These characteristics involve the fixed amount of training data and the fixed target for each datum throughout a learning process. The second environment is opposite to the first environment. The training data enter a learning process as a stream of single datum or a stream of data chunks with different sizes. Each datum may reenter the learning process with a new target. Notice that the first environment is no longer interesting because of the tremendous amount of data produced in the internet per unit time. The non-stationary behavior of the second environment induces the following several challenging constraints:

1. *Data overflow*. Since the data enter the learning process as a stream, it is possible that the data can overflow the memory and make the learning process impossible. Some data may be lost if the learning time is slower than the incoming speed of data stream.
2. *Class drift*. This is a situation when a datum temporally changes its class due to various reasons such as changing types of deposit account, career promotion. The features of datum are fixed but its target is changed. No assumption of class-

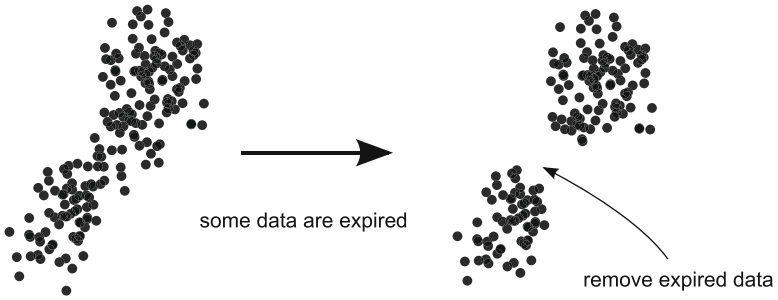


Fig. 2.1 An example of removing expired data and splitting the cluster into two clusters

drift probability is concerned. By this situation, the state of being *alive* or *dead* of datum is a special case of class drift.

3. *Class imbalanced ratio*. The amount of data of all classes entering the learning process is not equal. Furthermore, different sets of classes may occur in an incoming data chunk at any learning time.

Clustering The constraints for clustering are similar to those of classification in the aspects of data overflow and class drift. The situation of class imbalance ratio never exists in clustering problem. Data overflow in clustering constraint is exactly the same as that of classification. But for class drift, the only difference is that a datum is in only two states which are either *alive* or *dead*. A datum is labeled as *dead* if it is no longer in the learning process. For example, if a bank customer closed his deposit account, then his personal data are considered as *dead* data. The state of either *alive* or *dead* of any datum is assumed to be provided by the trainer prior to the learning process. Any dead datum must be disappeared and removed from its present cluster. Obviously, removing any dead data from its present cluster can change the topology of data distribution in the cluster. Figure 2.1 shows an example of removing some dead data. The present cluster is topologically split into two clusters.

2.3 Concept of One-Pass and Discard-After-Learn for Classification and Clustering

Data overflow is the first main problem to be solved. The conditions of data overflow in this study are defined as follows. Let M and C_{t_i} be the available size of memory used for learning process and the incoming data chunk at time t_i, respectively. The size of each C_{t_i} is denoted by $|C_{t_i}|$. Memory is partitioned into three main portions for storing the learning algorithm, buffering the incoming data chunk, and using as miscellaneous and temporary storage during the learning process. Data overflow implies that $M < \sum_{t_i \in \mathbf{N}} C_{t_i}$, where N is the total time used to input all data chunks

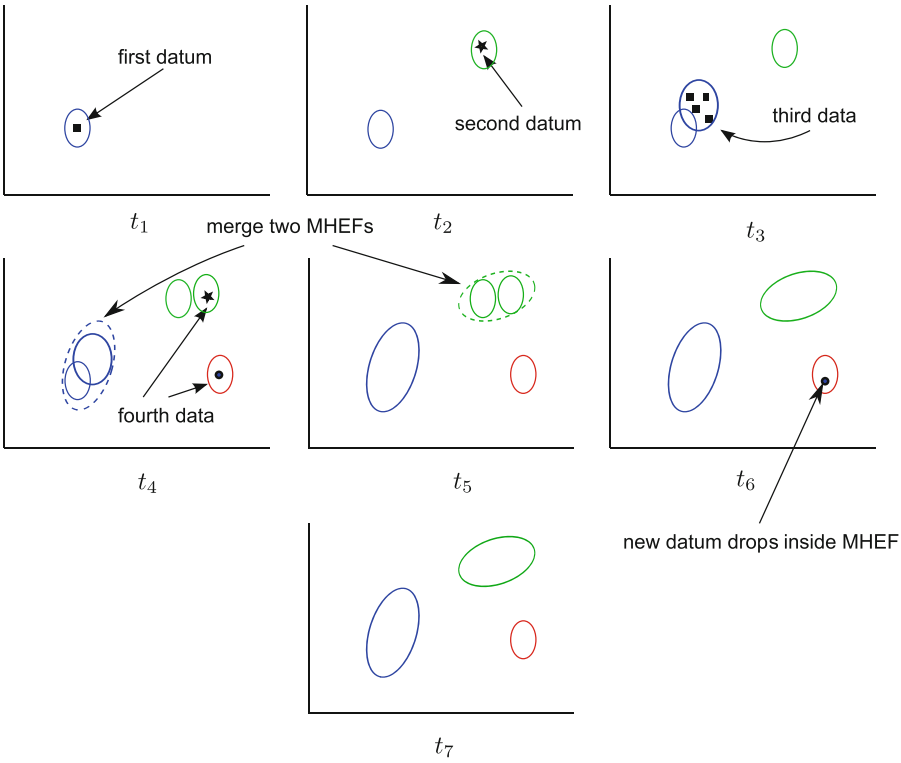


Fig. 2.2 An example of how the concept of *discard-after-learn* works for classification

by one chunk at a time. In this review, it is assumed that, for any t_i, $C_{t_i} < M$ and time period $[t_{i+1} - t_i]$ is less than or equal to the learning time of C_{t_i}.

For classification of streaming data in d dimensions, each chunk C_{t_i} is captured by a *malleable hyper-ellipsoid function* (MHEF) in the form of covariance matrix. This matrix obviously represents the direction of data distribution and the distribution width in each dimension. In the scenario of streaming data, if all incoming data are stored in the memory, then it is possible that the situation of data overflow, i.e. $M < \sum_{t_i \in \mathbf{N}} C_{t_i}$, can occur. To alleviate this consequence, once C_{t_i} is captured, all data in C_{t_i} are discarded forever. Only the covariance matrix of size d^2 is kept. Figure 2.2 illustrates this concept. There are three classes denoted by colors red, blue, and green MHEFs.

At time t_1, one datum of class *square* enters the learning process. This datum is captured by a MHEF and discarded afterwards. A datum of class *star* enters the process at time t_2. Since it is not in the same as the previous datum, a new MHEF is introduced to capture this datum. At time t_3, one datum of class *square* enters the process. It is captured by a new MHEF first whose location is close to another MHEF of the same class. Thus, these two MHEFs are merged into one larger MHEF

and the previous MHEFs are discarded as shown at time t_4 by the dashed MHEF. A MHEF with dashed boundary denotes the just happening result of merging two MHEFs. The merging process also occurs at time t_5. At time a new datum of class *circle* enters the process but its location is within the MHEF of the same class. In this case, there is no need to introduce a new MHEF for this datum. The datum is just discarded from the MHEF as shown at time t_7. To make the concept of *discard-after-learn* possible, the structure of MHEF must be rotatable, expandable, and transposable without disturbing the captured data.

For clustering of streaming data, the concept of *discard-after-learn* with MHEF structure can be adapted. In clustering problem, two touching MHEFs with similar degree of information measure and similar direction of data distribution are merged into a larger MHEF. Even though they touch each other but their directions of data distribution may not be congruent. In this case, they are not labeled as the same cluster. Figure 2.3 shows an example of this concept. There are three clusters denoted by red, blue, and green colors. The first data chunk enters the clustering process at time t_1. It is captured by a blue MHEF and all data are discarded afterwards. At time t_2, there are two small data chunks entering the learning process

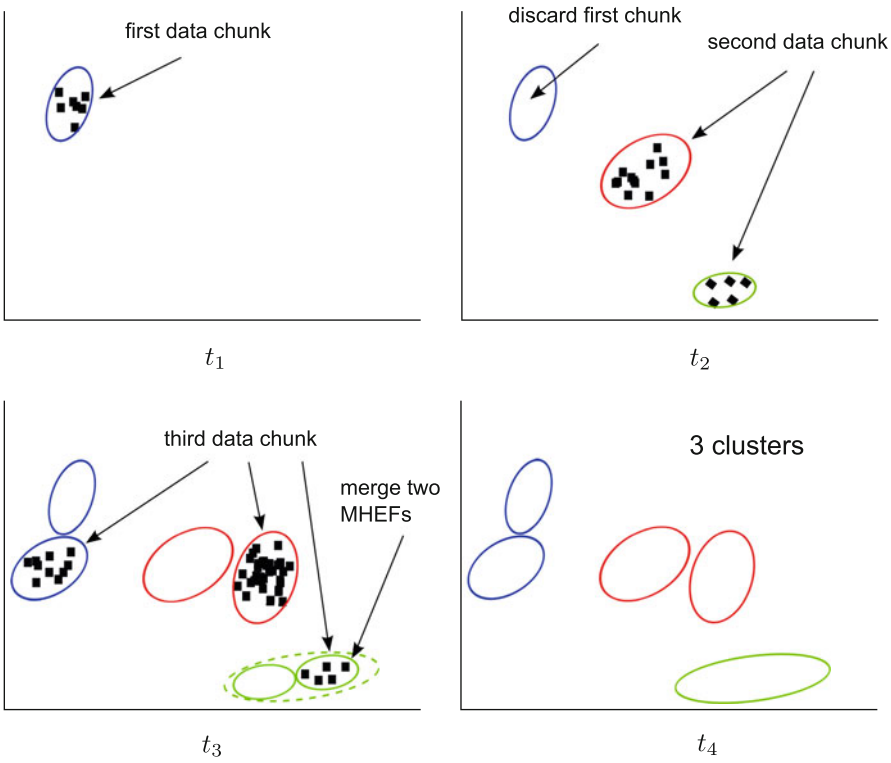


Fig. 2.3 An example of how the concept of *discard-after-learn* is adapted for clustering

at the same time. These two chunks are far apart. So each of them is captured by one MHEF (red and green MHEFs). At this moment, it is assumed that each of them is from different cluster. At time t_3, three small data chunks enter the process. Each chunk is then captured by one MHEF. Notice that although two blue MHEFs are close to each other but the directions of data distribution of both MHEFs are not congruent. So these two MHEFs are not merged into a larger one because the region of new MHEF may cover the much space which may be the location of a new cluster. The red clusters also possess this limitation. Only the green clusters are merged into one larger MHEF.

2.4 Structure of Malleable Hyper-ellipsoid Function

Let $\mathbf{c} \in R^d$ be the center and λ_j be the width in dimension j of a malleable hyper-ellipsoid. The malleable hyper-ellipsoid function (MHEF) capturing a datum $\mathbf{x}_k$ is defined as follows:

$$H(\mathbf{x}_k) = \sum_{j=1}^{d} \frac{((\mathbf{x}_k - \mathbf{c})^T \mathbf{v}_j)^2}{\lambda_j^2} - 1 \tag{2.1}$$

$\mathbf{v}_j$ is the jth eigenvector and λ_j is the jth eigenvalue computed from the following covariance matrix $\mathbf{M}_i$. Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be the set of data captured by the MHEF.

$$\mathbf{M} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{c})(\mathbf{x}_i - \mathbf{c})^T \tag{2.2}$$

$$\mathbf{M} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T - \mathbf{c}\mathbf{c}^T \tag{2.3}$$

Notice that vector $\mathbf{v}_j$ is used to rotate the MHEF according to the direction of data distribution and $\mathbf{c}$ is used to transpose MHEF from the origin to the center of $\mathbf{X}$. Datum $\mathbf{x}_k$ is inside $(\mathbf{x}_k - \mathbf{c}_i)$ if $(\mathbf{x}_k - \mathbf{c}_i) < 0$ and outside if $(\mathbf{x}_k - \mathbf{c}_i) > 0$. But if $(\mathbf{x}_k - \mathbf{c}_i) = 0$, then the class of $\mathbf{x}_k$ is indeterminate. The shape of MHEF can be expanded by computing the new eigenvalues from the new covariance matrix based on new incoming chunk of data. Function $H(\mathbf{x}_k)$ can be written in the following generic structure where r is a constant.

$$H(\mathbf{x}_k) = \sum_{j=1}^{d} \frac{((\mathbf{x}_k - \mathbf{c})^T \mathbf{v}_j)^2}{\lambda_j^2} - r \tag{2.4}$$

This structure can be easily transformed into a hyper-spherical function by setting $\lambda_j = 1$ and retaining the constant r as its radius. Hence, using this MHEF is more practical due to its generic mathematical structure. The technique of updating the present covariance matrix and center by using only new incoming data and the present covariance matrix as well as center will be discussed next.

2.5 Updating Malleable Hyper-ellipsoid Function

There are two parameters of MHEF which are the center and covariance matrix to be updated during the learning process. Since all captured data of any MHEF are completely discarded, updating the center and covariance matrix by using only incoming data must deploy the following structure of recursive function.

2.5.1 Recursively Updating Center

Let $\mathbf{c}^{(new)}$ and $\mathbf{c}^{(old)}$ be the newly updated center and the old center, respectively. The size of incoming data chunk may have only one datum or more than one datum. In case of one incoming datum, let $\mathbf{x}_{n+1}$ be a new incoming datum and $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be the set of present data. The updated $\mathbf{c}^{(new)}$ is computed by the following equation:

$$\mathbf{c}^{(old)} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i \tag{2.5}$$

$$\mathbf{c}^{(new)} = \frac{n}{n+1}\mathbf{c}^{(old)} + \frac{\mathbf{x}_{n+1}}{n+1} \tag{2.6}$$

Equation (2.6) can be modified to update the center in case of multiple data as follows. Let μ be the center of incoming data chunk $\{\mathbf{x}_{n+1}, \ldots, \mathbf{x}_{n+m}\}$ of size m.

$$\mu = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_{n+i} \tag{2.7}$$

$$\mathbf{c}^{(new)} = \frac{1}{n+m}(n \cdot \mathbf{c}^{(old)} + m \cdot \mu) \tag{2.8}$$

2.5.2 Recursively Updating Covariance Matrix

The directions of data distribution and the variance in each direction can be directly computed from the covariance matrix. Let $\mathbf{M}^{(new)}$ and $\mathbf{M}^{(old)}$ be the newly updated

covariance matrix and the old covariance matrix, respectively. Suppose there is only one new incoming datum $\mathbf{x}_{n+1}$.

$$\mathbf{M}^{(new)} = \frac{n}{n+1}\mathbf{M}^{(old)}\frac{\mathbf{x}_{n+1}\mathbf{x}_{n+1}^T}{n+1} - \mathbf{c}^{(new)}(\mathbf{c}^{(new)})^T + \mathbf{c}^{(old)}(\mathbf{c}^{(old)})^T - \frac{\mathbf{c}^{(old)}(\mathbf{c}^{(old)})^T}{n+1} \tag{2.9}$$

In case of an incoming data chunk $\{\mathbf{x}_{n+1}, \ldots, \mathbf{x}_{n+m}\}$, the newly updated covariance matrix can be computed as follows. Let μ be the center of the incoming data chunk.

$$\mu = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_{n+i} \tag{2.10}$$

$$\mathbf{M}^{(new)} = \frac{n}{m}(\mathbf{M}^{(old)} + \mathbf{c}^{(old)}(\mathbf{c}^{(old)})^T) + \frac{1}{n+m}\sum_{i=1}^{m}\mathbf{x}_{n+i}\mathbf{x}_{n+i}^T - \mu\mu^T \tag{2.11}$$

2.5.3 Merging Two Covariance Matrices

Two MHEFs must be merged to reduce the redundant MHEFs if they lie in the same direction. After merging them, center and the covariance matrix of a new MHEF must be computed from the centers and covariance matrices of two merged MHEFs as follows. Let

1. $\mathbf{c}$ be the center of new MHEF after merging two MHEFs.
2. $\mathbf{c}_1$ and $\mathbf{c}_2$ be the centers of the first and second MHEFs, respectively.
3. $\mathbf{M}$ be the new covariance matrix after merging MHEFs.
4. $\mathbf{M}_1$ and $\mathbf{M}_2$ be the covariance matrices of the first and second MHEFs, respectively.
5. n_1 and n_2 be the number of data captured by the first and second MHEFs, respectively.

$$\mathbf{c} = \frac{n_1\mathbf{c}_1 + n_2\mathbf{c}_2}{n_1+n_2} \tag{2.12}$$

$$\mathbf{M} = \frac{n_1\mathbf{M}_1}{n_1+n_2} + \frac{n_2\mathbf{M}_2}{n_1+n_2} + \frac{n_1 n_2}{(n_1+n_2)^2}(\mathbf{c}_1-\mathbf{c}_2)(\mathbf{c}_1-\mathbf{c}_2)^T \tag{2.13}$$

2.6 Analysis of Time and Space Complexities of Updating Computation

The time and space complexities for computing and updating the center and covariance matrix are separately analyzed as follows:

1. *Computing and updating center*: For only one incoming datum, the space complexity to compute the center is obviously equal to $O(n)$ where n is the size of the first training data set. The time complexity is $O(n)+m \cdot O(1) = O(n+m)$ where m is the number of incoming data after the first training data set. But for incoming data chunk, the upper bound of memory size is assumed to be $O(n)$ and the each next incoming data chunk is $m \leq n$. Thus, the space complexity to compute the center is equal to $O(n)$ because the center μ of incoming data chunk must be computed first.
2. *Computing and updating covariance matrix*: For only one incoming datum, the time to compute the covariance matrix of the first training data set of size n is $O(nd^2)$ with the space complexity of $O(nd)$. After the first covariance matrix, the covariance matrix is updated by Eq. (2.11) with time complexity of $O(d^3)$ for each incoming datum. But in case of following incoming data chunk, as given in Eq. (2.11), the covariance matrix of the incoming chunk must be computed first with time complexity of $O(md^2)$. Then, the old covariance matrix is updated with time complexity of $O(md^2) + O(d^2)$. The space complexity is $O(nd)$.
3. *Computing new center and covariance matrix after merging two MHEFs*: The size of each covariance matrix is of $O(d^2)$, where d is the number of dimensions. The time complexity to compute the new center is $O(d)$. For the new covariance matrix in Eq. (2.13), adding two matrices takes $O(d^2)$ and computing $(\mathbf{c}_1 - \mathbf{c}_2)(\mathbf{c}_1 - \mathbf{c}_2)^T$ takes $O(d^3)$. The space complexity is $O(d^2)$.

2.7 Applying Discard-After-Learn to Arbitrary Class Drift

Arbitrary class drift is a scenario when a datum can be assigned to any classes at any time with no assumption of the probability of class change. This means that the probability is unknown in advance. A datum maintains its features throughout the learning process of streaming environment. Only its class is arbitrarily and temporally changed due to some reasons. When a datum has a class drift, a new MHEF of the corresponding class is introduced to capture the datum and the datum is discarded afterwards. This implies that if this datum keeps changing its class, then there must be a stack of MHEF capturing this datum. The difficulty of this class drift is how to keep track of the recent class of any datum. To solve this problem, a new structure called *dynamic stratum* was introduced. The structure has two strata, lower and upper. Each stratum contains a set of MHEFs of several classes. When a datum enters the learning process, it is assigned to the lower stratum. But when its class is changed, it is moved to the upper stratum. To determine the recent class of a queried

Table 2.1 An example of class drift and the stratum assignment of class-drift datum

(a) Incoming data and their classes at different time sequence

Time	Incoming data	
	(x_1, x_2)	class
t_1	(3,4)	1
	(7,8)	2
	(1,2)	1
	(6,9)	3
	(4,6)	2
t_2	**(7,8)**	**3**
	(2,3)	4
	(1,2)	**3**
t_3	**(3,4)**	**2**
	(8,9)	4

(b) The stratum assignment of data

Time	Data	Class	Stratum
t_1	(3,4)	1	Lower
	(7,8)	2	Lower
	(1,2)	1	Lower
	(6,9)	3	Lower
	(4,6)	2	Lower
t_2	(3,4)	1	Lower
	(7,8)	**3**	**Upper**
	(1,2)	**3**	**Upper**
	(6,9)	3	Lower
	(4,6)	2	Lower
	(2,3)	4	Lower
t_3	**(3,4)**	**2**	**Upper**
	(7,8)	3	Upper
	(1,2)	3	Upper
	(6,9)	3	Lower
	(4,6)	2	Lower
	(2,3)	4	Lower
	(8,9)	4	Lower

datum, the upper stratum is searched first. If there exists a MHEF covering the location of feature vector of the queried datum, then the class of MHEF is indicated as the class of queried datum. But if there is no such MHEF, the lower stratum is searched by the same concept. Table 2.1 illustrates an example of class drift when all data are in a 2-dimensional space. Table 2.1a is the incoming data chunk at time t_1. At time t_2, data (7,8) and (1,2) have class drifts and a new datum (2,3) enters the process. At time t_3, datum (3,4) changes its class from 1 to 2 and one new datum (8,9) appears. The assignment of MHEFs of all data based on their class drift to different stratum at each time step is clarified in Table 2.1b.

The sequence of capturing each datum according to Table 2.1 is illustrated in Fig. 2.4. After the class of a datum is changed, the center and covariance matrix capturing this datum prior to the class drift must be updated. This is the process of removing a datum from MHEF. Let $\mathbf{x}_k$ be a datum having a class drift. The new center and covariance matrix are updated as follows:

$$\mathbf{M}^{(new)} = \frac{(-\mathbf{c}^{(new)} + \mathbf{x}_k)(\mathbf{c}^{(new)} - \mathbf{x}_k)^T}{n} + \frac{n}{n-1}\mathbf{M}^{(old)} \tag{2.14}$$

$$\mathbf{c}^{(new)} = \frac{n}{n-1}\mathbf{c}^{(old)} - \frac{\mathbf{x}_k}{n-1} \tag{2.15}$$

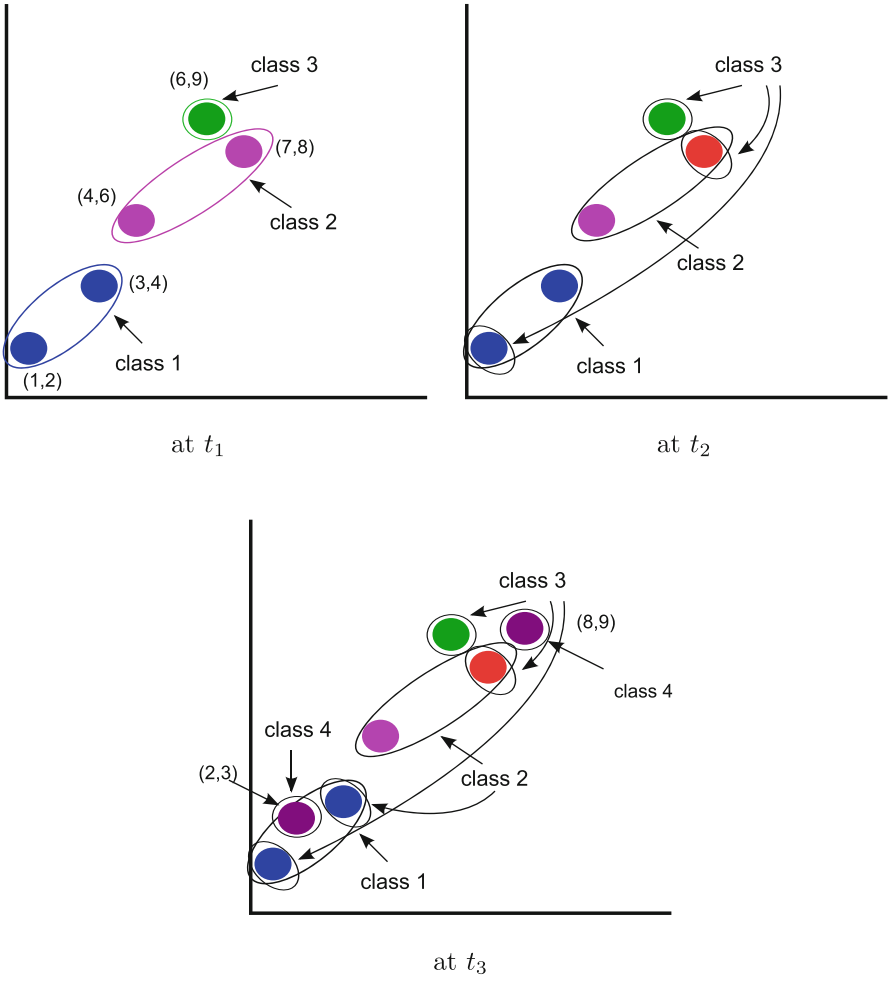


Fig. 2.4 The sequence of capturing data by MHEFs according to the class drift in Table 2.1

In case of removing a data chunk having multiple class drifts, the center and covariance matrix can be updated as follows. Let $\mathbf{c}^{(out)}$ and $\mathbf{M}^{(out)}$ be the center and covariance matrix of data chunk to be removed after class drift, respectively. The size of removed chunk is m.

$$\mathbf{M}^{(new)} = \frac{n}{n-m}\mathbf{M}^{(old)} - \frac{m}{n-m}\mathbf{M}^{(out)} - \frac{nm}{(n-m)^2}(\mathbf{c}^{(old)} - \mathbf{c}^{(out)})(\mathbf{c}^{(old)} - \mathbf{c}^{(out)})^T \tag{2.16}$$

$$\mathbf{c}^{(new)} = \frac{n}{n-m}(n\mathbf{c}^{(old)} - m\mathbf{c}^{(out)}) \tag{2.17}$$

The time complexity of updating the covariance matrix is still $O(d^3)$ for each removed chunk. Furthermore, the space complexity is $O(d^2) + O(\max(n, m))$.

2.8 Applying Discard-After-Learn to Expired Data in Clustering

The problem of class drift does not exactly occur in clustering. The only possibility is that the lives of some data in some existing clusters may expire due to some reasons. Similar to the class drift, the probability of expiration is unknown. Once a datum is expired, it must be removed from the cluster. It is assumed that the information of any expired datum is provided to the learning process in terms of feature vector and its status of being existent or expired. If there are many expired data, the corresponding MHEF must be split into two smaller MHEFs. But in *discard-after-learn* concept, all clusters data are completely discarded. This makes splitting a MHEF rather complex because of its hyper-elliptical shape. Some information may be lost after splitting. To alleviate this problem, a hyper-cylindrical shape was introduced. This shape is not a mathematical function but rather a set with some attributes. The definition of a hyper-cylindrical shape is the following. Let **c** be the center of hyper-cylindrical shape whose radius is r and length is L.

Definition 1 Hyper-cylindrical shape (HCS) $C = \{\mathbf{x}_i \mid \mathbf{x}_i \in R^d; 1 \leq i \leq n\}$ such that

1. $||(\mathbf{x}_i - \mathbf{c}) \cdot \mathbf{v}|| \leq L/2$.
2. $||\mathbf{x}_i - \mathbf{c}|| - ||(\mathbf{x}_i - \mathbf{c}) \cdot \mathbf{v}|| \leq r$.

v is the eigenvector computed from the covariance matrix of data set captured by this HCS whose eigenvalue is maximum. Figure 2.5 illustrates the structure of hyper-cylindrical shape in a 2-dimensional space. This shape is suitable to cope

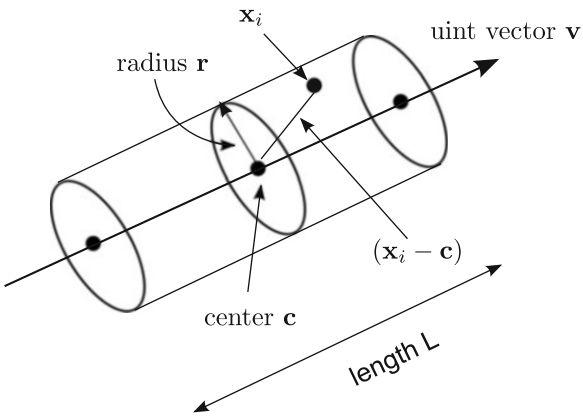


Fig. 2.5 Hyper-cylindrical shape and its center **c**, radius r, and length L in a 2-dimensional space. $\mathbf{x}_i$ is a datum captured by the hyper-cylinder

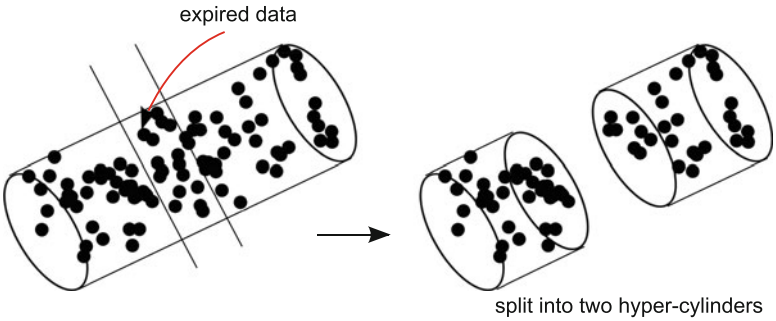


Fig. 2.6 An example of splitting a hyper-cylinder into two smaller hyper-cylinders because of expired data

with expired data because its shape can be easily split into two smaller hyper-cylindrical shapes by re-computing the length of each split hyper-cylinder as shown in the example of Fig. 2.6. The information regarding expired data is occasionally provided during the clustering process.

2.9 Discussion

The concept of one-pass learning and discard-after-learn is very practical and efficient in terms of time and space complexities. According to the experimental results reported in [11, 12, 20, 21, 23, 24], the number of neurons deployed in all experiments is significantly less than the other methods [2, 4–6, 8, 13, 16–18, 22, 26, 27]. It is noticeable that the learning process based on this concept does not involve any cost function to adjust the synaptic weights. This implies that there is no effect of large-error domination from any class. Hence, it is possible to adapt this concept to cope with dynamic imbalance ratio in a streaming environment. Our experimental results reported in [20] confirm that capability.

Although the proposed concept is rather versatile, the initial width of MHEF is still not efficient enough. Furthermore, the accuracy may be sensitive to the incoming order of data from different classes. These disadvantages require further study.

2.10 Conclusion

A new concept of one-pass learning with discard-after-learn and the relevant mathematical functions for learning streaming data previously introduced in [11, 12, 20, 21, 23, 24] are summarized. This approach achieved the lower bounds of

time and space complexities of neural learning. The accuracy of this approach is also significantly higher than the accuracy of other approaches. Furthermore, the modified version of this concept is introduced to construct a new network structure called *dynamic stratum* for handling arbitrary class drift problem.

Acknowledgement This work is supported by Thailand Research Fund under grant number RTA6080013.

References

1. Abdulsalam, H., Skillicorn, D. B., & Martin, P. (2011, January). Classification using streaming random forests. *IEEE Transactions on Knowledge and Data Engineering, 23*(1), pp. 22–36.
2. Aggarwal, C. C., Han, J., Wang, J., & Yu, P. S. (2003). A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases* (pp. 81–92).
3. Brzezinski, D., & Stefanowski, J. (2014, January). Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems, 25*(1), 81–94.
4. Cao, F., Ester, M., Qian, W., & Zhou, A. (2006). Density-based clustering over an evolving data stream with noise. In *SIAM International Conference on Data Mining* (pp. 328–339).
5. Ditzler, G., Rosen, G., & Polikar, R. (2014, July). Domain adaptation bounds for multiple expert systems under concept drift. In *Proceedings of the International Joint Conference on Neural Networks* (pp. 595–601).
6. Elwell, R., & Polikar, R. (2011, October). Incremental learning of concept drift in non-stationary environments. *IEEE Transactions on Neural Networks, 22*(10), 1517–1531.
7. Furao, S., & Hasegawa, O. (2008, December). A fast nearest neighbor classifier based on self-organizing incremental neural network. *Neural Networks, 21*(10), 1537–1547.
8. Hahsler, M., & Dunham, M. H. (2010). rEMM: Extensible Markov model for data stream clustering in r. *Journal of Statistical Software, 35*(5).
9. He, H., Chen, S., Li, K., & Xu, X. (2011, December). Incremental learning from stream data. *IEEE Transactions on Neural Networks, 22*(12), 1901–1914.
10. Hoens, T. R., Polikar, R., & Chawla, N. V. (2012, April). Learning from streaming data with concept drift and imbalance: An overview. *Progress in Artificial Intelligence, 1*(1), 89–101.
11. Jaiyen, S., Lursinsap, C., Phimoltares, S. (2010, March). A very fast neural learning for classification using only new incoming datum. *IEEE Transactions on Neural Networks, 21*(3), 381–392.
12. Junsawang, P., Phimoltares, S., & Lursinsap, C. (2016). A fast learning method for streaming and randomly ordered multi-class data chunks by using one-pass-throw-away class-wise learning concept. *Expert Systems with Applications, 63*, 249–266.
13. Kranen, P., Assent, I., Baldauf, C., & Seidl, T. (2011). The ClusTree: Indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems, 29*(2), 249–272.
14. Laohakiat, S., Phimoltares, S., & Lursinsap, C. (2016). Hyper-cylindrical micro-clustering for streaming data with unscheduled data removals. *Knowledge-Based Systems, 99*, 183–200.
15. Ozava, S., Pang, S., & Kasabov, N. (2008, June). Incremental learning of chunk data for online pattern classification systems. *IEEE Transactions on Neural Networks, 19*(6), 1061–1074.
16. Pang, S., Ban, T., Kadobayashi, Y., & Kasabov, N. K. (2012). LDA merging and splitting with applications to multi-agent cooperative learning and system alteration. *IEEE Transactions on Systems, Man, and Cybernetics. Part B, Cybernetics, 42*(2), 552–564.

17. Pang, S., Ozawa, S., & Kasabov, N. (2005). Incremental learning discriminant analysis classification of data streams. *IEEE Transactions on Systems, Man, and Cybernetics-part B: Cybernetics, 35*(5), 905–914.
18. Shen, F., & Hasegawa, O. (2008). A fast nearest neighbor classifier on self-organizing incremental neural network. *Neural Networks, 21*, 1537–1547.
19. Singla, P., Subbarao, K., & Junkins, J. L. (2007, January). Direction-dependent learning approach for radial basis function networks. *IEEE Transaction on Neural Networks, 18*(1), 203–222.
20. Thakong, M., Phimoltares, S., Jaiyen, S., & Lursinsap, C. (2017). Fast learning and testing for imbalanced multi-class changes in streaming data by dynamic multi-stratum network. *IEEE Access, 5*, 10633–10648.
21. Thakong, M., Phimoltares, S., Jaiyen, S., & Lursinsap, C. (2018). One-pass-throw-away learning for cybersecurity in streaming non-stationary environments by dynamic stratum networks. *PLoS One, 13*(9), e0202937.
22. Tu, L., Chen, Y. (2009). Stream data clustering based on grid density and attraction. *ACM Transactions on Knowledge Discovery from Data, 3*(3), 12:1–12:27.
23. Wattanakitrungroj, N., Maneeroj, S., & Lursinsap, C. (2017). Versatile hyper-elliptic clustering approach for streaming data based on one-pass-thrown-away learning. *Journal of Classification, 34*, 108–147.
24. Wattanakitrungroj, N., Maneeroj, S., & Lursinsap, C. (2018). BEstream batch capturing with elliptic function for one-pass data stream clustering. *Data & Knowledge Engineering, 117*, 53–70.
25. Wu, X., Li, P., & Hu, X. (2012, September). Learning from concept drifting data streams with unlabeled data. *Neurocomputing, 92*, 145–155.
26. Xu, Y., Shen, F., & Zhao, J. (2012). An incremental learning vector quantization algorithm for pattern classification. *Neural Computing and Applications, 21*(6), 1205–1215.
27. Zheng, J., Shen, F., Fan, H., & Zhao, J. (2013, April). An online incremental learning support vector machine for large-scale data. *Neural Computing and Applications, 22*(5), 1023–1035.
28. Žliobaitė, I., Bifet, A., Read, J., Pfahringer, B., & Holmes, G. (2015, March). Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Machine Learning, 98*(3), 455–482.

Chapter 3
Distributed Single-Source Shortest Path Algorithms with Two-Dimensional Graph Layout

Thap Panitanarak

3.1 Introduction

With the advance of online social networks, World Wide Web, e-commerce, and electronic communication in the last several years, data relating to these areas has become exponentially larger day by day. This data is usually analyzed in a form of graphs modeling relations among data entities. However, processing these graphs is challenging not only from a tremendous size of the graphs that is usually in terms of billions of edges, but also from real-world graph characteristics such as sparsity, irregularity, and scale-free degree distributions that are difficult to manage. For example, we can construct a graph from Twitter users who retweet a popular message. For users (or vertices) who are well-known (e.g., celebrities), the numbers of retweets (edges) from these users is very large compared to most regular users. Thus, the graph will have a few vertices with very high degree, while most vertices have very low degree in general.

Large-scale graphs are commonly stored and processed across multiple machines or in distributed environments due to a limited capability of a single machine. However, current graphs analyzing tools, which have been optimized and used on sequential systems cannot directly be used on these distributed systems without scalability issues. Thus, novel graph processing and analysis are required, and parallel graph computations are mandatory to be able to handle these large-scale graphs efficiently.

Single-source shortest path (SSSPs) is a well-known graph computation that has been studied for more than half a century. It is one of the most common graph

T. Panitanarak (✉)
Department of Mathematics and Computer Science, Chulalongkorn University, Bangkok, Thailand
e-mail: thap.p@chula.ac.th

M. W. Berry et al. (eds.), *Supervised and Unsupervised Learning for Data Science*,
Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-030-22475-2_3

analytical analyses for many graph applications such as networks, communication, transportation, electronics, and so on. There are many SSSP algorithms that have been proposed such as well-known Dijkstra's algorithm [1] and Bellman-Ford algorithm [2, 3]. However, these algorithms are designed for serial machines, and do not efficiently work on parallel environments. As a result, many researchers have studied and proposed parallel SSSP algorithms or implemented SSSP as parts of their parallel graph frameworks. Some well-known graph frameworks include the Parallel Boost Graph Library [4], GraphLab [5], PowerGraph [6], Galois [7], and ScaleGraph [8]. More recent frameworks have been proposed based on Hadoop systems [9] such as Cyclops [10], GraphX [11], and Mizan [12]. For standalone implementations of SSSP, most recent implementations usually are for GPU parallel systems such as [13–15]. However, high-performance GPU architectures are still not widely available and they also require fast CPUs to speed up the overall performance. Some SSSP implementations on shared memory systems include [16–18].

In this chapter, we focus on designing and implementing efficient SSSP algorithms for distributed memory systems. While the architectures are not relatively new, there are few efficient SSSP implementations for this type of architectures. We are well aware of the recent SSSP study of Chakaravarthy et al. [19] that is proposed for massively parallel systems, IBM Blue Gene/Q (Mira). Their SSSP implementations have applied various optimizations and techniques to achieve very good performance such as direction optimization (or a push-pull approach), pruning, vertex cut, and hybridization. However, most techniques are specifically for SSSP algorithms and can only be applied to a limited variety of graph algorithms. In our case of SSSP implementations, most of our techniques are more flexible and can be extended to many graph algorithms, while still achieving good performance. Our main contributions include

- Novel SSSP algorithms that combine advantages of various well-known SSSP algorithms.
- Utilization of a two-dimensional graph layout to reduce communication overhead and improve load balancing of SSSP algorithms.
- Distributed cache-like optimization that filters out unnecessary SSSP updates and communication to further increase the overall performance of the algorithms.
- Detailed evaluation of the SSSP algorithms on various large-scale graphs.

3.2 Overviews

3.2.1 Single-Source Shortest Path Algorithms

Let $G = (V, E, w)$ be a weighted, undirected graph with $n = |V|$ vertices, $m = |E|$ edges, and integer weights $w(e) > 0$ for all $e \in E$. Define $s \in V$ called a source vertex, and $d(v)$ to be a tentative distance from s to $v \in V$ (initially set to ∞). The single

source shortest path (SSSP) problem is to find $\delta(v) \leq d(v)$ for all $v \in V$. Thus, $\delta(v)$ is the shortest path from s to v. Define $d(s) = 0$, and $d(v) = \infty$ for all v that are not reachable from s.

Relaxation is an operation to update $d(v)$ using many well-known SSSP algorithms such as Dijkstra's algorithm and Bellman-Ford. The operation updates $d(v)$ use a previously updated $d(u)$ for each $(u, v) \in E$. An edge relaxation of (u, v) is defined as $d(v) = \min \{d(v), d(u) + w(u, v)\}$. A vertex relaxation of u is a set of edge relaxations of all edges of u. Thus, a variation of SSSP algorithms is generally based on the way the relaxation has taken place.

The classical Dijkstra's algorithm relaxes vertices in an order starting from a vertex with the lowest tentative distance first (starting with s). After all edges of that vertex are relaxed, the vertex is marked as settled; that is, the distance to such vertex is the shortest possible. To keep track of a relaxing order of all active vertices v (or vertices that have been updated and are to be relaxed), the algorithm uses a priority queue that orders active vertices based on their $d(v)$. A vertex is added to the queue only if it is visited for the first time. The algorithm terminates when the queue is empty. Another variant of Dijkstra's algorithm for integer weight graphs that is suited for parallel implementation is called Dial's algorithm [20]. It uses a bucket data structure instead of a priority queue to avoid the overhead from maintaining the queue while still giving the same work performance as Dijkstra's algorithm. Each bucket has a unit size, and holds all active vertices that have the same tentative distance as a bucket number. The algorithm works on buckets in order, starting from the lowest to the highest bucket numbers. Any vertex in each bucket has an equal priority and can be processed simultaneously. Thus, the high concurrency of the algorithm can be obtained by the presence of these buckets.

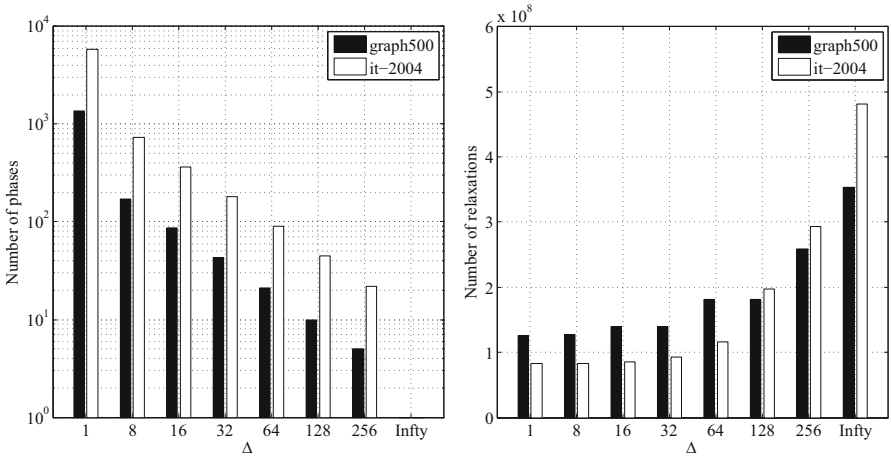
Another well-known SSSP algorithm, Bellman–Ford, allows vertices to be relaxed in any order. Thus, there is no guarantee if a vertex is settled after it has been once relaxed. Generally, the algorithm uses a first-in-first-out (FIFO) queue to maintain the vertex relaxation order, since there is no actual priority of vertices. A vertex is added to the queue when its tentative distance is updated, and is removed from the queue after it is relaxed. Thus, any vertex can be added to the queue multiple times whenever its tentative distance is updated. The algorithm terminates when the queue is empty. Since the order of relaxation does not affect the correctness of the Bellman–Ford algorithm, it allows the algorithm to provide high concurrency from simultaneous relaxation.

While Dijkstra's algorithm yields the best work efficiency since each vertex is relaxed only once, it has very low algorithm concurrency. Only vertices that have the smallest distance can be relaxed at a time to preserve the algorithm correctness. In contrast, Bellman–Ford requires more works from (possibly) multiple relaxations of each vertex. However, it provides the best algorithm concurrency, since any vertex in the queue can be relaxed at the same time. Thus, the algorithm allows simultaneously relaxations, while the algorithm's correctness is still preserved.

The Δ-stepping algorithm [21] compromises between these two extremes by introducing an integer parameter $\Delta \geq 1$ to control the trade-off between work efficiency and concurrency. At any iteration $k \geq 0$, the Δ-stepping algorithm relaxes

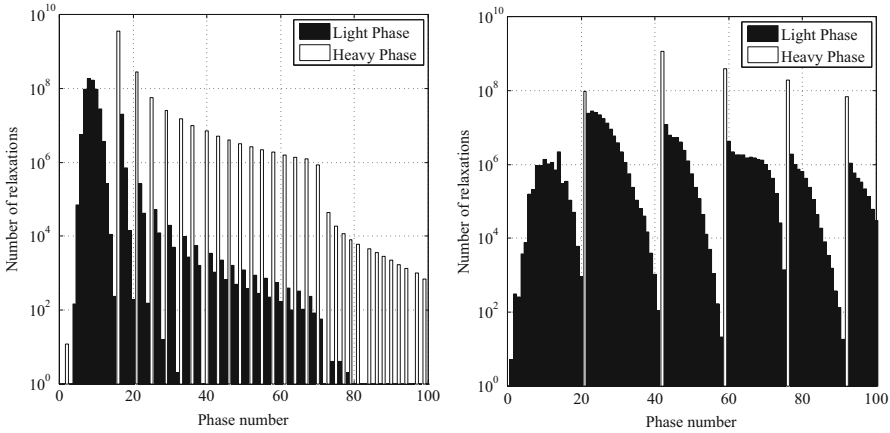
the active vertices that have tentative distances in $[k\Delta, (k+1)\Delta - 1]$. With $1 < \Delta < \infty$, the algorithm yields better concurrency than the Dijkstra's algorithm and lower work redundancy than the Bellman–Ford algorithm. To keep track of active vertices to be relaxed in each iteration, the algorithm uses a bucket data structure that puts vertices with the same distant ranges in the same bucket. The bucket k contains all vertices that have the tentative distance in the range $[k\Delta, (k+1)\Delta - 1]$. To make the algorithm more efficient, two processing phases are introduced in each iteration. When an edge is relaxed, it is possible that the updated distance of an adjacency vertex may fall into the current bucket, and it can cause cascading reupdates as in Bellman–Ford. To minimize these reupdates, edges of vertices in the current bucket with weights less than Δ (also called light edges) are relaxed first. This forces any reinsertion to the current bucket to happen earlier, and, thus, decreasing the number of reupdates. This phase is called a light phase, and it can iterate multiple times until there is no more reinsertion, or the current bucket is empty. After that, all edges of vertices, which are previously relaxed in the light phases with weights greater than Δ (also called heavy edges), are then relaxed. This phase is called a heavy phase. It only occurs once at the end of each iteration, since, with edge weights greater than Δ, the adjacency vertices from updating tentative distances are guaranteed not to fall into the current bucket. The Δ-stepping algorithm can be viewed as a general case of SSSP algorithms with the relaxation approach. The algorithm with $\Delta = 1$ is equivalent to Dijkstra's algorithm, while the algorithm with $\Delta = \infty$ yields Bellman–Ford.

Figure 3.1a, b shows the number of phases and edge relaxations, respectively, with different Δ values on two graphs, graph500 with scale 27 (2^{27} vertices) and it-2004. The Dijkstra's algorithm and Bellman–Ford are shown with Δ equal to



(a) Number of phases (b) Number of edge relaxations

Fig. 3.1 The effect of Δ in the Δ-stepping algorithm on the numbers of phases (**a**) and edge relaxations (**b**)



(a) graph500 (b) it-2004

Fig. 3.2 The number of relaxations on the first 100 phases of the Δ-stepping algorithm with $\Delta = 32$ on graph500 (**a**) and it-2004 (**b**)

1 and ∞, respectively, as they are equivalent algorithms. Increasing Δ results in decreasing the number of phases as each phase with larger Δ can relax more vertices. Thus, the larger the Δ, the more the concurrency of the algorithm. However, increasing Δ also yields higher number of edge relaxations as it increases a chance of each vertex to be relaxed multiple times. Thus, the larger the Δ, the less the work efficiency. Note that it-2004 has higher numbers of phases because of its large graph diameter compared to graph500.

Figure 3.2 shows the numbers of relaxations on the first 100 phases of Δ-stepping algorithm with $\Delta = 32$ on graph500 and it-2004. Note that graph500 and it-2004 require 123 and 954 phases to complete the SSSP execution, respectively. The results of both graphs show that the first half of the algorithm is dominated by light phase relaxations, while the latter half is dominated by heavy phase relaxations. During early phases, most of vertices are still unsettled, and result in more work in light phases. At some points where a large portion of vertices is settled, more work is shifted to heavy phases as the number of reinsertion of unsettled vertices decreases.

3.2.2 Two-Dimensional Graph Layout

Most distributed graph computations utilize distributed adjacency lists of vertices, which are usually presented by a compressed sparse row format to make an efficient use of memory. Because of its simplicity and flexibility to extend to distributed graph computations, it is widely used as an underlying graph data structure. For example, to distribute a graph with n vertices among p processors, nonoverlapping

n/p vertices along with their corresponding outgoing edges can be assigned to each processor. This approach can be viewed as a one-dimensional (1D) graph layout, since it is generally a partition of the graph adjacency matrix by row into p submatrices. Each submatrix contains nonoverlapping n/p rows. While this approach works well in general, there are two major flaws that can degrade the overall performance. First, it has high communication overhead, because data relating to vertex adjacencies is distributed among all processors. Any update to the vertices will affect all other processors. Thus, all processors are required to participate in the update usually in the form of an all-to-all collective communication. Secondly, the 1D graph layout only considers an equal distribution of vertices of a graph. While each processor gets approximately the same number of vertices, there is no guarantee that the number of edges in each partition is equally distributed. This can lead to load imbalance issues for graph algorithms that require edge traversal. The problem is more pronounced, specifically, in power-law real-world graphs, since these graphs contain very few high-degree vertices, while most of vertices have very low degrees. Even though there are some techniques that have been used in many graph frameworks and algorithms to handle the load-balancing issue such as dynamic load balancing and vertex cut [11], they also introduce more complexity and computation from additional data structures to maintain the originality of the problem.

A two-dimensional (2D) graph layout had been previously studied in [22] for parallel breadth-first search. This approach partitions an adjacency matrix of graph vertices into grid blocks instead of a traditional row partition, 1D graph layout. The 2D layout reduces communication space and also provides better edge distributions of a distributed graph than the 1D layout as any dense row of the high degree vertices can now be distributed across multiple processors instead of only one processor as in the 1D layout. The adjustment has also taken place on the underling graph data structures. Thus, it can be extended to other distributed graph algorithms efficiently and effectively. To illustrate the advantage of the 2D layout, consider partitioning a sparse adjacency matrix of a graph into grid blocks of r rows and c columns. Each interprocessor communication still occurs only on one dimension either along the row or column. Thus, the interprocessor communication space is now reduced. If an $n \times n$ sparse adjacency matrix of a graph is partitioned into $p = r \times c$ partitions, with the traditional 1D layout, each set of n/p consecutive rows of the matrix is assigned to one partition (see Fig. 3.3a). Alternatively, we can partition the adjacency matrix into grid blocks, and assign each block to one processor (see Fig. 3.3b). With $p = r \times c$ processors, the communication space can be reduced from $r \times c$ to only r for any row communication such as an all-to-all communication. Furthermore, this approach provides better load balancing of both vertices and edges of a graph as any dense row of any high degree vertex can now be distributed across multiple processors instead of residing on one processor as in the 1D layout.

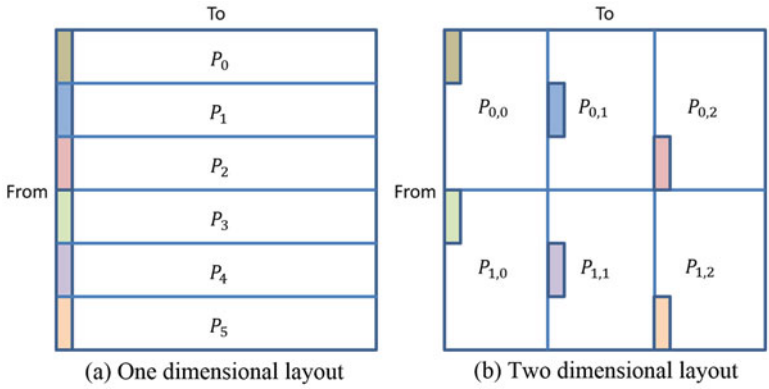


Fig. 3.3 The comparison of the (**a**) one- and (**b**) two-dimensional graph layouts

3.3 Novel Parallel SSSP Implementations

3.3.1 General Parallel SSSP for Distributed Memory Systems

We consider SSSP implementations with 1D layout in [23], which are based on a bulk-synchronous Δ-stepping algorithm for distributed memory systems. The algorithm is composed of three main steps, a local discovery, an all-to-all exchange, and a local update for both light and heavy phases. In the local discovery step, each processor looks up to all adjacencies v of its local vertices u in the current bucket, and generates corresponding tentative distances $dtv = d(u) + w(u, v)$ of those adjacencies. Note that, in the light phase, only adjacencies with light edges are considered, while, in the heavy phase, only adjacencies with heavy edges are processed. For each (u, v), a pair (v, dtv) is generated, and stored in a queue called QRequest. The all-to-all exchange step distributes these pairs in QRequest to make them local to processors so that each processor can use this information to update a local tentative distance list in the local update step. An edge relaxation is part of the local update step that invokes updating vertex tentative distances, and adding/removing vertices to/from buckets based on their current distances.

3.3.2 Parallel SSSP with 2D Graph Layout

To apply the 2D graph layout for the Δ-stepping algorithm, each of the three steps needs to be modified according to the changes in the vertex and edge distributions. While the vertices are distributed in similar manner as in the 1D graph layout, edges are now distributed differently. Previously in the 1D layout, all edges of local vertices are assigned to one processor. However, with the 2D layout, these edges are

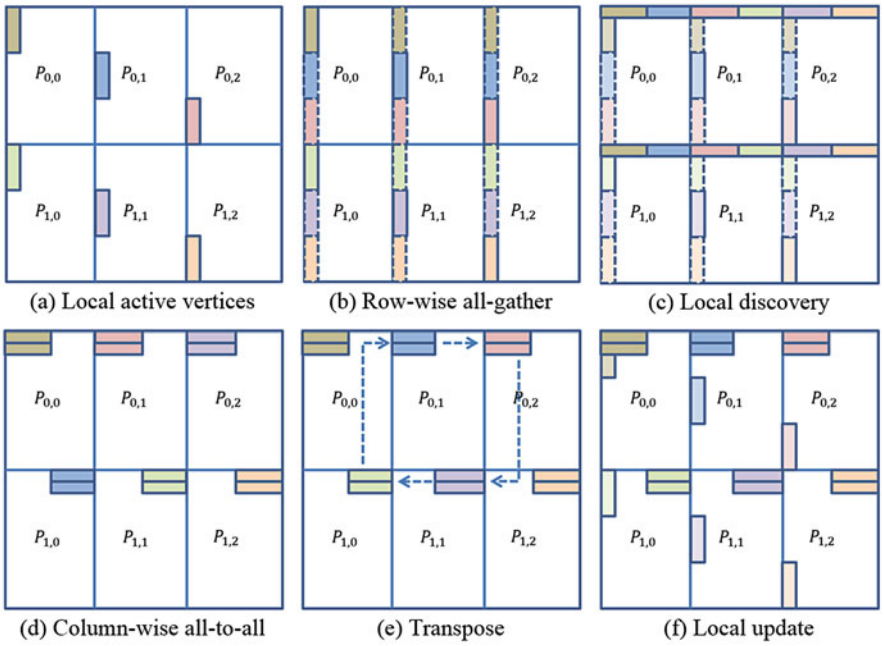


Fig. 3.4 The main SSSP operations with the 2D layout. (**a**) Each color bar shows the vertex information for active vertices owned to each processor $P_{i,j}$. (**b**) The row-wise all-gather communication gathers all information of actives vertices among the same processor rows to all processors in the same row. (**c**) Each processor uses the information to update the vertex adjacencies. (**d**, **e**) The column-wise all-to-all and transpose communications group the information of the updated vertices owned by the same processors and send this information to the owner processors. (**f**) Each processor uses the received information to update its local vertex information

now distributed among row processors that have the same row number. Figure 3.4a illustrates the partitioning of vertices and edges for the 2D layout.

In the local discovery step, there is no need to modify the actual routine. The only work that needs to be done is merging all current buckets along the processor rows by using a row-wise all-gather communication. The reason is that the edge information (such as edge weights and adjacencies) of local vertices owned by each processor is now distributed among the processor rows. Thus, each processor with the same row number is required to know all the active vertices in the current bucket of their neighbor processor rows before the discovery step can take place. After the current buckets are merged (see Fig. 3.4b), each processor can now simultaneously work on generating pairs (v, dtv) of its local active vertices (see Fig. 3.4c).

In the all-to-all exchange step, the purpose of this step is to distribute the generated pairs (v, dtv) to the processors that are responsible for maintaining the information relating to vertices v. In our implementation, we use two subcommunications: a column-wise all-to-all exchange and a send-receive transposition. The column-wise all-to-all communication puts all information pairs of vertices owned

by the same owner onto one processor. Figure 3.4d shows a result of this all-to-all exchange. After that, each processor sends and receives these pair lists to the actual owner processors. The latter communication can be viewed as a matrix transposition as shown in Fig. 3.4e.

In the local update step, there is no change within the step itself, but only in the data structure of the buckets. Instead of only storing vertices in buckets, the algorithm needs to store both vertices and their current tentative distances so that each processor knows the distance information without initiating any other communication. Figure 3.4f illustrates the local update step. Since all pairs (d, dtv) are local, each processor can update the tentative distances of their local vertices simultaneously.

The complete SSSP algorithm with the 2D graph layout is shown in Algorithm 3.1. The algorithm initialization shows in the first 10 lines. The algorithm checks for the termination in line 11. The light and heavy phases are shown in lines 12–25 and lines 26–35, respectively. The termination checking for the light phases of a current bucket is in line 12. The local discovery, all-to-all exchange, and local update steps of each light phase are shown in lines 13–19, 20, and 22, respectively. Similarly for each heavy phase, its local discovery, all-to-all exchange, and local update steps are shown in lines 26–31, 32, and 34, respectively. Algorithm 3.2 shows the relaxation procedure used in Algorithm 3.1.

Algorithm 3.1: Distributed SSSP with 2D Graph Layout

```
1:  for each u do
2:      d[u] ← ∞
3:  end for
4:    current ← 0
5:  if onwer(s) = rank then
6:      d[s] ← 0
7:  end if
8:  if onwerRow(s) = rankRow then
9:      Bucket[current] ← Bucket[current] ∪ (s, 0)
10: end if
11: while Bucket ≠ ∅ do                    //Globally check
12:   while Bucket[current] ≠ ∅ do //Globally check
13:     for each (u, du) ∈ Bucket[current] do
14:       for each (u, v) ∈ LightEdge do
15:         dtv ← du + w(u, v)
16:         QRequest ← QRequest ∪ (v, dtv)
17:       end for
18:       QHeavy ← QHeavy ∪ (u, du)
19:     end for
20:     Alltoallv(QRequest, row); Transpose(QRequest)
21:     for each (v, dtv) ∈ QRequest do
22:       Relax(v, dtv)
```

```
23:     end for
24:     Allgatherv(Bucket[current], col)
25:   end while
26:   for each (u, du) ∈ QHeavy do
27:     for each (u, v) ∈ HeavyEdge do
28:       dtv ← du + w(u, v)
29:       QRequest ← QRequest ∪ (v, dtv)
30:     end for
31:   end for
32:   Alltoallv(QRequest, row); Transpose(QRequest)
33:   for each (v, dtv) ∈ QRequest do
34:     Relax(v, dtv)
35:   end for
36:   current ← current + 1        //Move to next bucket
37:   Allgatherv(Bucket[current], col)
38: end while
```

Algorithm 3.2: *Relax*(*v*, *dtv*)

```
1: if d[v] > dtv then
2:   old ← d[v]/Δ; new ← dtv/Δ
3:   Bucket[old] ← Bucket[old] − (v, d[v])
4:   Bucket[new] ← Bucket[new] ∪ (v, dtv)
5:   d[v] ← dtv
6: end if
```

3.3.3 Other Optimizations

To further improve the algorithm performance, we apply other three optimizations, a cache-like optimization, a heuristic Δ increment, and a direction optimization. The detailed explanation is as follows:

Cache-like optimization: We maintain a tentative distance list of every unique adjacency of the local vertices as a local cache. This list holds the recent values of tentative distances of all adjacencies of local vertices. Every time a new tentative distance is generated (during the discovery step), this newly generated distance is compared to the local copy in the list. If the new distance is shorter, it will be processed in the regular manner by adding the generated pair to the QRequest, and the local copy in the list is updated to this value. However, if the new distance is longer, it will be discarded, since the remote processors will eventually discard this request during the relaxation anyway. Thus, with a small trade-off of

additional data structures and computations, this approach can significantly avoid unnecessary work that involves both communication and computation in the later steps.

Heuristic Δ increment: The idea of this optimization is from the observation of the Δ-stepping algorithm that the algorithm provides a good performance in early iterations when Δ is small, since it can avoid most of the redundant work in the light phases. Meanwhile, with a large Δ, the algorithm provides a good performance in later iterations, since most of vertices are settled so that the portion of the redundant work is low. Thus, the benefit of the algorithm concurrency outweighs the redundancy. The algorithm with Δ that can be adjusted when needed can provide better performance. From this observation, instead of using a fix Δ value, we implement algorithms that starts with a small Δ until some thresholds are met, and then, the Δ is increased (usually to ∞) to speed up the later iterations.

Direction-optimization: This optimization is a heuristic approach first introduced in [24] for breadth-first search (BFS). Conventional BFS usually proceeds in a top-down approach such that, in every iteration, the algorithm checks all adjacencies of each vertex in a frontier whether they are not yet visited, adds them to the frontier, and then marks them as visited. The algorithm terminates whenever there is no vertex in the frontier. We can see that the algorithm performance is highly based on processing vertices in this frontier. The more vertices in the frontier, the more work that needs to be done. From this observation, the bottom-up approach can come to play for efficiently processing of the frontier. The idea is that instead of proceeding BFS only using the top-down approach, it can be done in a reverse direction if the current frontier has more work than the work using the bottom-up approach. With a heuristic determination, the algorithm can alternately switch between the top-down and bottom-up approaches to achieve an optimal performance. Since the discovery step in SSSP is done in a similar manner as BFS, Chakaravarthy et al. [19] adapt a similar technique called a push–pull heuristic to their SSSP algorithms. The algorithms proceed with a push (similar to the top-down approach) by default during heavy phases. If a forward communication volume of the current bucket is greater than a request communication volume of aggregating of later buckets, the algorithms switch to a pull. This push–pull heuristic considerably improves an overall performance of the algorithm. The main reason of the improvement is because of the lower of the communication volume; thus, the consequent computation also decreases.

3.3.4 Summary of Implementations

In summary, we implement four SSSP algorithms:

1. *SP1a*: The SSSP algorithm based on Δ-stepping with the cache-like optimization.
2. *SP1b*: The SP1a algorithm with the direction optimization.

3. *SP2a*: The SP1a algorithm with the 2D graph layout.
4. *SP2b*: The SP2a algorithm with the Δ increment heuristic.

The main differences of each algorithm are the level of optimizations that additionally increases from SP#a to SP#b that is the SP#b algorithms are the SP#a algorithms with more optimizations, and from SP1x to SP2x that is the SP1x algorithms use the 1D layout while the SP2x algorithms use the 2D layout.

3.4 Performance Results and Analysis

3.4.1 Experimental Setup

Our experiments are run on a virtual cluster using StarCluster [25] with the MPICH2 complier version 1.4.1 on top of Amazon Web Service (AWS) Elastic Compute Cloud (EC2) [26]. We use 32 instances of AWS EC2 m3.2xlarge. Each instance consists of eight cores of high-frequency Intel Xeon E5-2670 v2 (Ivy Bridge) processors with 30 GB of memory. The graphs that we use in our experiments are listed in Table 3.1. The graph500 is a synthetic graph generated from the Graph500 reference implementation [27]. The graph generator is based on the RMAT random graph model with the parameters similar to those used in the default Graph500 benchmark. In this experiment, we use the graph scale of 27 with edge factor of 16; that is, the graphs are generated with 2^{27} vertices with an average of 16 degrees for each vertex. The other six graphs are real-world graphs that are obtained from Stanford Large Network Dataset Collection (SNAP) [29], and the University of Florida Sparse Matrix Collection [28]. The edge weights of all graphs are randomly, uniformly generated between 1 and 512.

We fix the value of Δ to 32 for all algorithms. Please note that this value might not be the optimal value in all test cases, but, in our initial experiments on the systems, it gives good performance in most cases. To get the optimal performance in all cases is not practical, since Δ needs to be changed according to the systems such as CPU, network bandwidth and latency, and numbers of graph partitions. For more discussion about the Δ value, please see [23].

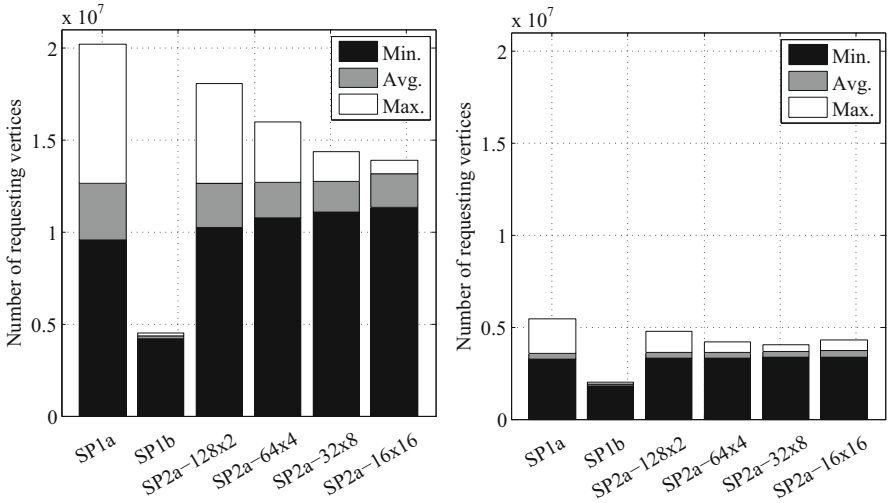
Table 3.1 The list of graphs used in the experiments

Graph	Number of vertices (millions)	Number of edges (billions)	Reference
graph500	134	2.1	[27]
it-2004	41	1.1	[28]
sk-2005	50	1.9	[28]
friendster	65	1.8	[29]
orkut	3	0.12	[29]
livejournal	4	0.07	[29]

3.4.2 Algorithm and Communication Cost Analysis

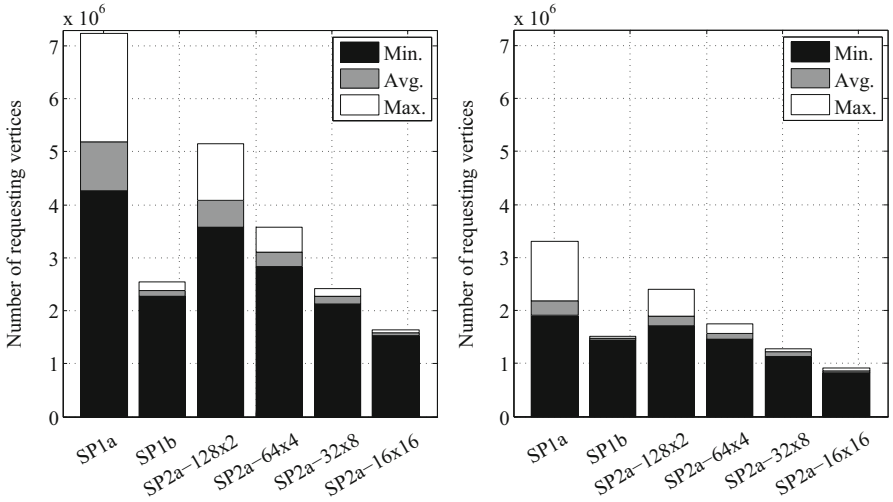
For SSSP algorithms with the 2D layout, when the number of columns increases, the all-to-all communication overhead also decreases, and the edge distribution is more balanced. Consider processing a graph with n vertices and m edges on $p = r \times c$ processors. The all-to-all and all-gather communication spaces are usually proportional to r and c, respectively. In other words, the maximum number of messages for each all-to-all communication is proportional to m/c, while the maximum number of messages for each all-gather communication is proportional to n/r. In each communication phase, processor $P_{i,j}$ needs to interact with processors $P_{k,j}$ for the all-to-all communication where $0 \leq k < r$, and with processors $P_{i,l}$ for the all-gather communication where $0 \leq l < c$. For instance, by setting $r = 1$ and $c = p$, the algorithms do not need any all-to-all communication, but the all-gather communication now requires all processors to participate.

During the SSSP process on scale-free graphs, there are usually a few phases of the algorithms that consume most of the computation and communication times due to the presence of few vertices with high degrees. Figure 3.5a, b and c, d shows the average, minimum, and maximum vertices to be requested and sent, respectively, for relaxations during the phase that consumes the most time of the algorithms SP1a, SP1b, and SP2a on graph500 and it-2004 with 256 MPI tasks. Note that we use the abbreviation SP2a-$R \times C$ for the SP2a algorithm with R and C processor rows and columns, respectively. For example, SP2a-64 $\times$ 4 is the SP2a algorithm with 64 row and 4 column processors (which are 256 processors in total). The improvement of load balancing of the requested vertices for relaxations can easily be seen in Fig. 3.5a, b as the minimum and maximum number of the vertices decreases on both graphs from SP1a to SP1b and SP1a to SP2a. The improvement from SP1a to SP1b is significant as the optimization is specifically implemented for reducing the computation and communication overheads during the high-requested phases. On the other hand, SP2a still processes on the same number of vertices, but with lower communication space and better load balancing. Not only the load balancing of the communication improves, but the number of (average) messages among interprocessors also reduces, as we can see in Fig. 3.5c, d. However, there are some limitations of both SP1b and SP2a. For SP1b, the push–pull heuristic may not trigger in some phases as the costs of push and pull approaches are slightly different. In contrast, for SP2a, although increasing the number of columns improves load balancing and decreases the all-to-all communication in every phase, it also increases the all-gather communication proportionally. There is no specific number of columns that gives the best performance of the algorithms, since it depends on various factors such as the number of processors, the size of the graph, and other system specifications.



(a) The number of requested vertices: graph500

(b) The number of requested vertices: it-2004



(d) The number of sent vertices: graph500

(e) The number of sent vertices: it-2004

Fig. 3.5 The numbers of (**a**, **b**) requested and (**c**, **d**) sent vertices during the highest relaxation phase of the SP2a algorithm on graph500 and it-2004 using different combinations of processor rows and columns on 256 MPI tasks

3.4.3 *Benefits of 2D SSSP Algorithms*

Figure 3.6 shows the algorithm performance in terms of traversed edges per second (TEPS) on Amazon EC2 up to 256 MPI tasks. Although SP1b can significantly reduce computation and communication during the high-requested phases, its overall performance is similar to SP2a. The SP2b algorithm gives the best performance in all cases, and it also gives the best scaling when the number of processors increases. The peak performance of SP2b-32 $\times$ 8 is approximately 0.45 GTEPS that can be observed on graph500 with 256 MPI tasks, which is approximately 2$\times$ faster than the performance of SP1a on the same setup. The SP2b algorithm also shows good scaling on large graphs such as graph500, it-2004, sk-2005, and friendster.

3.4.4 *Communication Cost Analysis*

Figure 3.7 shows the breakdown execution time of total computation and communication of each algorithm. More than half of the time for all algorithms is spent on communication as the networks of Amazon EC2 are not optimized for high performance computation. The improvement of SP1b over SP1a is from the reduction of computation overhead as the number of processing vertices in some phases is reduced. On the other hand, SP2a provides lower communication overhead over SP1a as the communication space is decreased from the use of the 2D layout. The SP2b algorithm further improves the overall performance by introducing more concurrency in the later phases, resulting in lower both communication and communication overhead during the SSSP runs. Figure 3.8 shows the breakdown communication time of all algorithms. We can see that when the number of processor rows increases, it decreases the all-to-all communication, and slightly increases the all-gather and transpose communications. In all cases, SP2b shows the least communication overhead with up to 10$\times$ faster for the all-to-all communication and up to 5$\times$ faster for the total communication.

3.5 Conclusion and Future Work

We propose scalable SSSP algorithms based on the Δ-stepping algorithm. Our algorithms reduce both communication and computation overhead from the utilization of the 2D graph layout, the cache-like optimization, and the Δ increment heuristic. The 2D layout improves the algorithm performance by decreasing the communication space, thus reducing overall communication overhead. Furthermore, the layout also improves the distributed graph load balancing, especially, on scale-free graphs. The cached-like optimization avoids unnecessary workloads for both communication and communication by filtering out all updated requests that are known to be

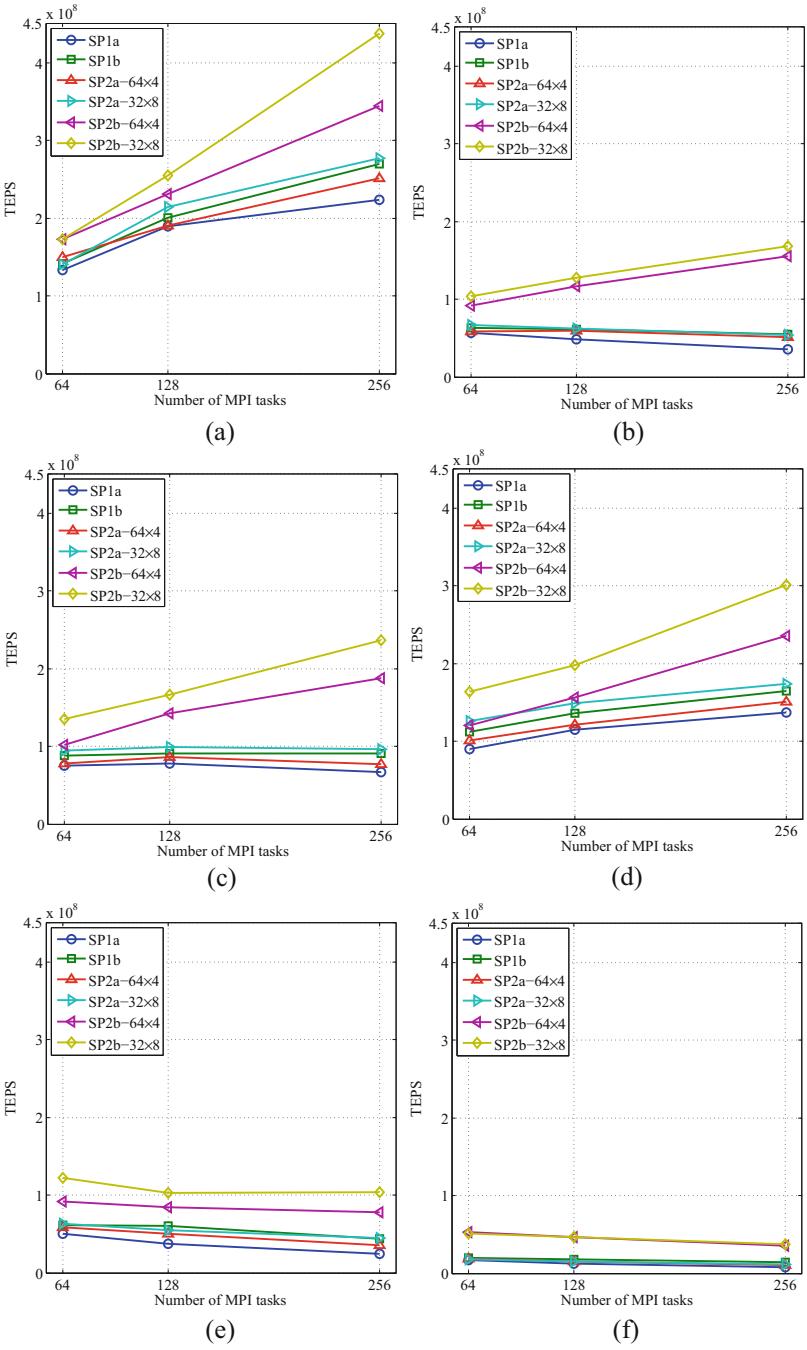


Fig. 3.6 The performance (in TEPS) of SSSP algorithms up to 256 MPI tasks. (**a**) graph500, (**b**) it-2004, (**c**) sk-2005, (**d**) friendster, (**e**) orkut, (**f**) livejournal

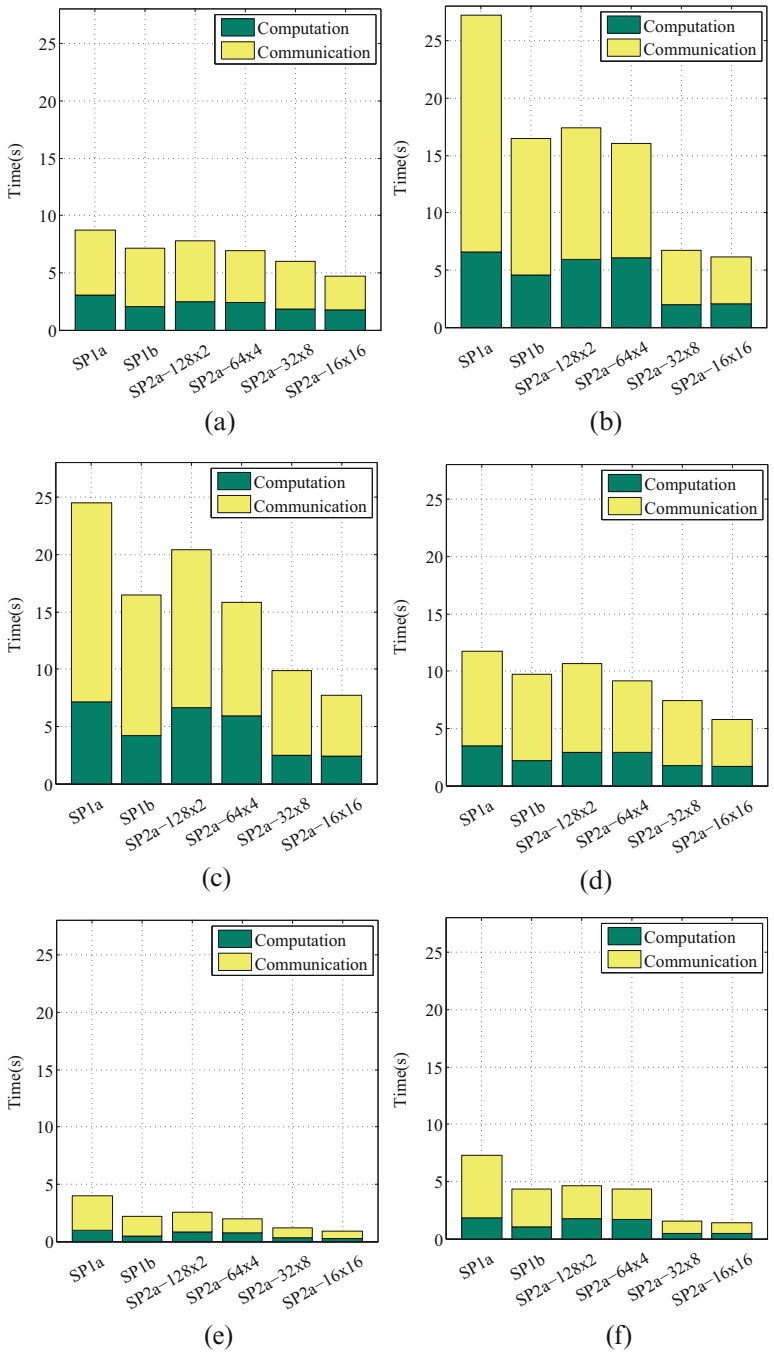


Fig. 3.7 The communication and computation times of SSSP algorithms on 256 MPI tasks. (**a**) graph500, (**b**) it-2004, (**c**) sk-2005, (**d**) friendster, (**e**) orkut, (**f**) livejournal

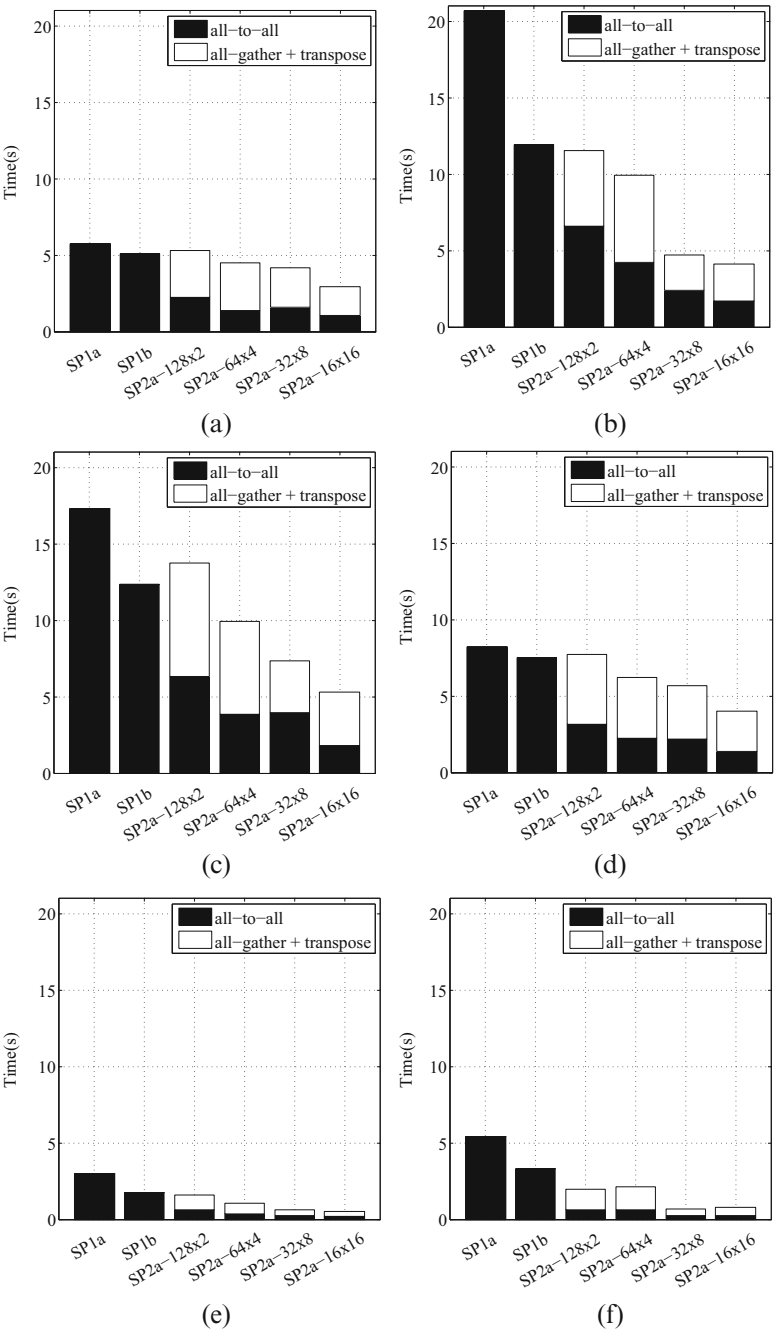


Fig. 3.8 Communication breakdown of SSSP algorithms on 256 MPI tasks. (**a**) graph500, (**b**) it-2004, (**c**) sk-2005, (**d**) friendster, (**e**) orkut, (**f**) livejournal

discarded. Finally, by increasing the Δ values during the algorithms progress, we can improve the concurrency of the algorithms in the later iterations.

Currently, our algorithm is based on the bulk-synchronous processing for distributed memory systems. We plan to extend our algorithms to also utilize the shared memory parallel processing that can further reduce the interprocessing communication of the algorithms.

Acknowledgments The author would like to thank Dr. Kamesh Madduri, an associate professor at Pennsylvania State University, USA, for the inspiration and kind support.

References

1. Dijkstra, E. W. (1959). A note on two problems in connection with graphs. *Numerische Mathematik, 1*(1), 269–271.
2. Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics, 16*, 87–90.
3. Ford, L. A. (1956). *Network flow theory*. Tech. Rep. Report P-923. Santa Monica, CA: The Rand Corporation.
4. Gregor, D., & Lumsdaine, A. (2005). The Parallel BGL: A generic library for distributed graph computations. *Parallel Object-Oriented Scientific Computing, 2*, 1–18.
5. Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., & Hellerstein, J. M. (2012). Distributed GraphLab: A framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment, 5*(8), 716–727.
6. Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., & Guestrin, C. (2012). PowerGraph: Distributed graph-parallel computation on natural graphs. In: *OSDI* (Vol. 12, p. 2).
7. Galois. Retrieved July 15, 2018, from http://iss.ices.utexas.edu/?p=projects/galois.
8. Dayarathna, M., Houngkaew, C., & Suzumura, T. (2012). Introducing ScaleGraph: An X10 library for billion scale graph analytics. In *Proceedings of the 2012 ACM SIGPLAN X10 Workshop* (p. 6). New York: ACM.
9. White, T. (2012). *Hadoop: The definitive guide*. Newton, MA: O'Reilly Media.
10. Chen, R., Ding, X., Wang, P., Chen, H., Zang, B., & Guan, H. (2014). Computation and communication efficient graph processing with distributed immutable view. In *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing* (pp. 215–226). New York: ACM.
11. Xin, R. S., Gonzalez, J. E., Franklin, M. J., & Stoica, I. (2013). Graphx: A resilient distributed graph system on Spark. In *First International Workshop on Graph Data Management Experiences and Systems* (p. 2). New York: ACM.
12. Khayyat, Z., Awara, K., Alonazi, A., Jamjoom, H., Williams, D., & Kalnis, P. (2013). Mizan: A system for dynamic load balancing in large-scale graph processing. In *Proceedings of the 8th ACM European Conference on Computer Systems* (pp. 169–182). New York: ACM.
13. Davidson, A. A., Baxter, S., Garland, M., & Owens, J. D. (2014). Work-efficient parallel GPU methods for single-source shortest paths. In *International Parallel and Distributed Processing Symposium* (Vol. 28).
14. Wang, Y., Davidson, A., Pan, Y., Wu, Y., Riffel, A., & Owens, J. D. (2015). Gunrock: A high-performance graph processing library on the GPU. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (pp. 265–266.. PPoPP 2015).
15. Zhong, J., & He, B. (2014). Medusa: Simplified graph processing on GPUs. *IEEE Transactions on Parallel and Distributed Systems, 25*(6), 1543–1552.

16. Madduri, K., Bader, D. A., Berry, J. W., & Crobak, J. R. (2007). An experimental study of a parallel shortest path algorithm for solving large-scale graph instances. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX)* (pp. 23–35). Society for Industrial and Applied Mathematics.
17. Prabhakaran, V., Wu, M., Weng, X., McSherry, F., Zhou, L., & Haridasan, M. (2012). Managing large graphs on multi-cores with graph awareness. In *Proceedings of USENIX Annual Technical Conference (ATC)*.
18. Shun, J., & Blelloch, G. E. (2013). Ligra: A lightweight graph processing framework for shared memory. In: *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (pp. 135–146). PPoPP'13.
19. Chakaravarthy, V. T., Checconi, F., Petrini, F., & Sabharwal, Y. (2014). Scalable single source shortest path algorithms for massively parallel systems. In *Proceedings of IEEE 28th International Parallel and Distributed Processing Symposium* (pp. 889–901).
20. Dial, R. B. (1969). Algorithm 360: Shortest-path forest with topological ordering. *Communications of the ACM, 12*(11), 632–633.
21. Meyer, U., & Sanders, P. (2003). Δ-stepping: A parallelizable shortest path algorithm. *Journal of Algorithms, 49*(1), 114–152.
22. Buluç, A., & Madduri, K. (2011). Parallel breadth-first search on distributed memory systems. In*: Proceedings of High Performance Computing, Networking, Storage and Analysis (SC)*.
23. Panitanarak, T., & Madduri, K. (2014). Performance analysis of single-source shortest path algorithms on distributed-memory systems. In *SIAM Workshop on Combinatorial Scientific Computing (CSC)* (p. 60). Citeseer.
24. Beamer, S., Asanović, K., & Patterson, D. (2013). Direction-optimizing breadth-first search. *Scientific Programming, 21*(3–4), 137–148.
25. StarCluster. Retrieved July 15, 2018, from http://star.mit.edu/cluster/.
26. Amazon Web Services. *Amazon elastic compute cloud*. Retrieved July 15, 2018, from http://aws.amazon.com/ec2/.
27. The Graph 500. Retrieved July 15, 2018, from http://www.graph500.org.
28. The University of Florida Sparse Matrix Collection. Retrieved July 15, 2018, from https://www.cise.ufl.edu/research/sparse/matrices/.
29. SNAP: Stanford Network Analysis Project. Retrieved July 15, 2018, from https://snap.stanford.edu/data/.

Chapter 4
Using Non-negative Tensor Decomposition for Unsupervised Textual Influence Modeling

Robert E. Lowe and Michael W. Berry

4.1 Introduction

> Nam cum pictor praecogitat quae facturus est, habet quidem in intellectu sed nondum intelligit esse quod nondum fecit. – Anselm of Canterbury [7]

In the eleventh century, Anselm of Canterbury wrote what has since come to be known as the ontological argument for the existence of God [7]. Anselm's argument was based on the assumption that all ideas, or more specifically, all thoughts originate either from perceptions of the outside world or from images formed within the imagination. From this he provides an argument for the existence of a divine being. The research presented here follows this same epistemological assumption to a much less trivial end. Instead of proving divine influence, the present work shall attempt to measure the influence present in the written works of less divine beings.

The basic assumption made about text documents is the same assumption that Anselm made about the origin of thoughts. Every word, phrase, sentence, paragraph, and theme in a document must come from one of two sources. Either the author created the thought from within their own mind, and as such this counts as a literary contribution, or the author adopted ideas from some outside source. These sources can take on many forms. In the case of academic writing, the author is likely to

R. E. Lowe
Division of Mathematics and Computer Science, Maryville College, Maryville, TN, USA
e-mail: robert.lowe@maryvillecollege.edu

M. W. Berry (✉)
Department of Electrical Engineering and Computer Science, University of Tennessee at Knoxville, Knoxville, TN, USA
e-mail: mberry@utk.edu

M. W. Berry et al. (eds.), *Supervised and Unsupervised Learning for Data Science*,
Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-030-22475-2_4

have been influenced primarily by the various books and papers that they have read over the course of their research. Another form of influence is a coauthor (though in the case of academic literature, coauthors are almost always explicitly stated.) Of course, the influence over the text in a paper is not constrained merely to the literature that the author has cited, but is ultimately a reflection of an author's entire life experience and background. In the case of literary writing, such as a novel or play, a reasonable assumption is that an author is influenced by other works within their genre as well as by the society in which they live.

Given that every written document is influenced by at least a small set of outside documents, the present work attempts to model and quantify this influence by separating documents into a set of factors and then searching for common factors among the documents. The desired result has two parts. First, a weight is assigned to each factor indicating its importance in the target work. Second, the factors themselves should carry enough semantic meaning to identify the ideas and elements of style which have been transferred from a source document to a target document. In this chapter, we highlight the work originally discussed in [22] on the identification of influencing factors and the quantification of the influence they exert on a target document.

The usefulness of such a measurement should be readily apparent to anyone working in any academic field. In modern research, the performance of participants is rooted in an attempt to measure that person's influence over their chosen field. Traditional approaches to this problem involve counting citations over a specific window of time [1] while more modern approaches tend to involve some document semantics [11, 16]. Measuring influence in a written document can also be applied in situations where authorship is in question. Given a corpus of works of confirmed provenance, and a disputed document, influence modeling can identify the possible influence of each author. Thus textual influence modeling can be used to answer the question of authorship where it is disputed, or could potentially be used to identify plagiarized passages [22].

4.2 Modeling Influence

At a high level, an influence model identifies elements that appear to have been incorporated into a target document from a source document. These elements are numerous. For example, they could include elements of style, topics, phrases, or ideas. A human reader seems to be able to identify these elements on an intuitive level, as can be seen readily whenever a reader says one author *sounds like* another. This operation is also in effect when tracing ideas through written academic literature. In either case, the text of a target document along with its corpus of cited documents seems to provide sufficient evidence to identify potential sources of influence in the target document.

The chief problem with an intuitive model such as the one outlined in the previous paragraph is that it is highly subjective. Every conclusion reached by human scholars in such a system must appeal to intuition and logic, and so determining the strength of any perceived relationships present in the corpus presents a difficult challenge. In recent years, the emerging field of computational stylistics has offered several techniques for quantifying these elements of style which can serve as markers of influence [3, 6, 10]. In so much as it can, computational stylistics has the principal goal of using textual evidence to answer the question of authorship. The current state-of-the-art techniques for addressing these questions rely upon statistical analysis of word frequencies within documents [10]. The typical approach is to use a set of *marker words* to determine the likelihood of an author's contribution to a target document. Those words that are more likely to occur in the works of one author are ascribed to them if the word has sufficient classifying power, which is usually determined by the statistical significance of the word's frequency. This current approach offers only a coarse level of determination. Computational stylistic analysts can identify words that are more likely to come from one author's work, and they in turn identify whether that author appears to have contributed to a target document. Thus the current techniques only inform the probability of an author's contribution as a dichotomy. Each potential author was either a contributor or they were not. The objective of the model outlined in this work is to extend this model to include more detail. As opposed to determining whether an author has contributed to a target work directly, this model assumes that influence is present in multiple forms; the present model seeks to identify the strength of influence, as well as to identify the specific nature of those influences.

4.2.1 Tensors and Decompositions

In order to analyze a document, it must first be quantified in some way that allows for analysis. The model we consider represents documents using tensors. The term "tensor" has been broadly applied across multiple fields to describe several different types of related objects. For the purposes of factor analysis, a tensor is simply an extension of matrices into a higher number of modes. In tensor terminology a "mode" is a dimension along which the tensor can be indexed. A scalar is a mode zero tensor, a vector is a mode one tensor, and a matrix is a mode two tensor. When the number of modes exceeds two, it is customary to refer to the array simply as a tensor. A detailed account of the tensor operations performed by this model is given in the next section. For a complete treatment of tensors as they pertain to factor analysis, see Tamara Kolda's tutorial [19].

The underlying principle of tensor analysis is polyadic decomposition, which was first described by Frank Hitchcock in 1927 [15]. When a tensor is expressed in polyadic form, it is expressed as the sum of rank 1 tensors, which is usually written as the outer product of vectors. (This is also referred to as the tensor product of vectors, which is in line with the geometric interpretation of tensors as the outer

product of vector spaces.) Each polyadic factor in a tensor of m modes is the tensor product of m vectors. For example, given a 3-mode tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$, its polyadic decomposition into r factors is a set of factors which satisfies Eq. 4.1. For the sake of convenience, the remainder of this discussion will assume a three-mode tensor, however everything discussed here can readily be extended to any number of modes.

$$\mathcal{T} \approx \sum_{i=1}^{r} a_i \otimes b_i \otimes c_i \tag{4.1}$$

The tensor, or outer, product $a \otimes b$ used in Eq. 4.1 results in a tensor where the modes are the concatenation of the modes of a and b. For instance, if a and b are vectors of size i and j respectively, $a \otimes b$ results in a 2-mode tensor with dimensions $i \times j$. In the case of building a 3-mode tensor, three vectors are needed, and the elements of the product result are computed as in Eq. 4.2. A graphical representation of this product is shown in Fig. 4.1. Of note is how each vector serves as scaling values for a mode.

$$\mathcal{T}_{ijk} = a_i b_j c_k \tag{4.2}$$

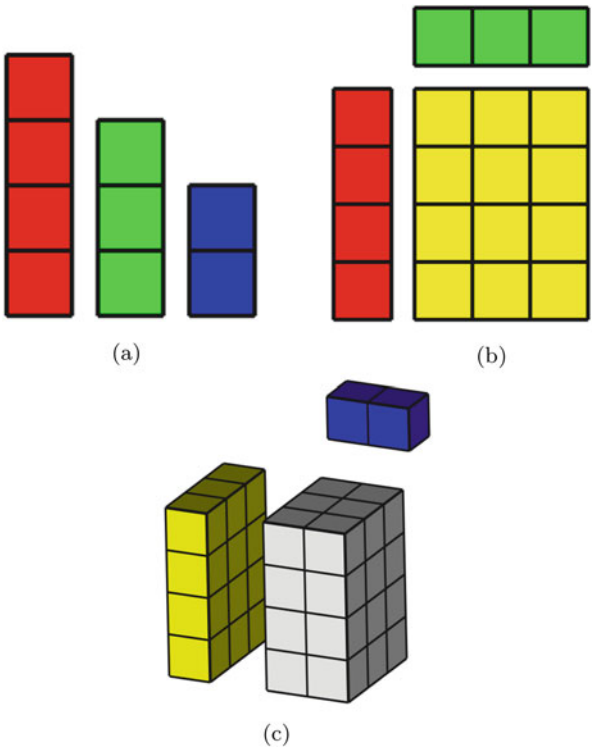


Fig. 4.1 Tensor product representations [22]. (**a**) Mode vectors. (**b**) $a \otimes b$. (**c**) $a \otimes b \otimes c$

The notion of the tensor product also gives rise to the notion of tensor rank. Tensor rank is not the same as matrix rank, and is in fact much more difficult to compute. Perhaps the easiest way to understand the notion of tensor rank is to think recursively. A rank 1 tensor is a tensor which can be constructed completely from the tensor product of 1-mode tensors (vectors). The tensor described in Eq. 4.2 and Fig. 4.1 is therefore a rank-1 tensor, as are all the factors in the polyadic decomposition. A tensor of general rank r is the result of the summation of r rank-1 tensors. The rank of a tensor $\mathcal{T}$ is therefore defined as the minimum number of rank-1 tensors needed to sum to $\mathcal{T}$.

Hitchcock's paper mainly presents the polyadic decomposition from a purely mathematical perspective, with applications to studying tensor invariants and tensor rank. In fact, as later papers show, the problem of determining the rank of a tensor is NP-Complete [14]. Polyadic decomposition began to see other uses when it was rediscovered in 1970 by Richard Harshman [12], Carroll and Chang [8]. Harshman coins the term "PARAFAC," a portmanteau of "Parallel Factors" while Carroll and Chang refer to the model as "CANDECOMP" in place of "Canonical Decomposition." Both papers present the model as a means of studying psychological data by treating the tensor factors as explanatory variables for the variance in the tensor data. In recent years, tensor analysis has begun to take root in other fields such as chemometrics [4] and text mining [3]. In several modern treatments, the polyadic decomposition is referred to as "CPD" or "Canonical Polyadic Decomposition." For the purposes of factor analysis, factors are often normalized, without loss of generality [3, 4], yielding the decomposition shown in Eq. 4.3.

$$\mathcal{T} \approx \sum_{i=1}^{r} \lambda_i a_i' \otimes b_i' \otimes c_i' \tag{4.3}$$

Here λ_i is a scalar where $\lambda_i = \text{norm}(a_i \otimes b_i \otimes c_i)$. This is desirable because the factors in this form are proportional profiles [12] with all of the magnitude of the factor contained in λ_i. As can be clearly seen from Eq. 4.3, λ_i is also an expression of the influence that factor i exerts over the tensor $\mathcal{T}$. For this reason, factors are typically expressed in order from largest λ_i to the smallest λ_i. Thus these factor norms serve a similar purpose as eigenvalues in principal component analysis (PCA), or as singular values in singular value decomposition (SVD).

In fact, the similarities between PCA, SVD, and CPD do not end with the inclusion of weights! Nor is it true that CPD is the only tensor decomposition. The closest competing decomposition is the Tucker decomposition, first proposed in 1963 and fully formed in 1966 [19]. Given tensor $\mathcal{T}$, the Tucker decomposition yields the factor matrices $\mathbf{A} \in \mathbb{R}^{I \times P}$, $\mathbf{B} \in \mathbb{R}^{J \times Q}$, and $\mathbf{C} \in \mathbb{R}^{K \times R}$. The model also contains a so-called core tensor $\mathcal{G} \in \mathbb{R}^{P \times Q \times R}$. These factors are fit to satisfy Eq. 4.4, where $\mathcal{G} \times_n \mathbf{M}$ is the n-mode product of tensor $\mathcal{G}$ and matrix $\mathbf{M}$.

$$\mathcal{T} \approx \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \tag{4.4}$$

An element-wise version of the tucker decomposition is shown in Eq. 4.5. For a complete treatment of the n-mode tensor matrix product, see the Kolda tutorial [19].

$$t_{ijk} \approx \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_{ip} b_{jq} c_{kr} \tag{4.5}$$

As was shown by Henk Kiers, the relationship between PCA, Tucker decomposition, and CPD is hierarchical [18]. While Kiers's paper focuses on 3-way analysis, his results extend to any number of modes. Because a tensor can be unfolded along any dimension to form a two-dimensional matrix, it is always possible to use PCA to find explanatory factors for tensor data. In fact, Tucker-3 is a constrained version of PCA. The exact nature of these constraints is beyond the scope of the present discussion, however they are a direct result of the presence of a core tensor. A simplified summary is that the core tensor's dimensions predetermines the number of factors to be discovered. CPD, in turn, is a constrained variant of the Tucker decomposition. While the CPD is usually written without a core tensor, it can be thought of as having an identity tensor as its core. The identity tensor is simply a tensor with ones along its super-diagonal and zeros everywhere else. (Or stated more formally, an identity tensor is a tensor containing ones where $i_1 = i_2 = \ldots = i_n$ for all n modes and zeros in all other positions.) Also, in CPD, $P = Q = R$ (and so on if there are more than three modes). Tucker decomposition allows for each mode to have a different number of factors, while CPD does not. As can be expected, the more constraints placed upon the explanatory model of the tensor come at the expense of quality of fit. Hence, PCA will always provide the best fit, Tucker will either be as good or worse than PCA, and CPD will always be as good or worse than a Tucker model [4, 18].

So then if the tensor decomposition models provide a worse fit, the question becomes why are they important? The answer lies in several properties of the tensor decompositions. First, tensor decompositions retain the structure of the original data [12, 19]. Unfolding a tensor into a matrix loses semantic information about the variables being analyzed, and extracting intuitive semantics from the resultant PCA model is difficult and usually impossible [4]. Also, in the case of CPD, the factors are unique under rotation so long as the number of factors extracted is greater than or equal to the rank of the tensor [13]. Another desirable property of the CPD is that it does not partition space by hyper-surfaces. Instead, it creates a sort of implicit set of axes for factor separation by providing a proportional profile along the tensor's basis [13]. Thus if several tensors of like dimensions are decomposed, their factors can be logically thought of as existing within the same space. This allows for comparison among the factors to be carried out, unlike under PCA where the factor space of each matrix is a projection into a new space, making comparison of factors from disparate matrices difficult to perform in a meaningful way.

In some instances of tensor analysis, it can be convenient to apply additional constraints to the model. The most common constraint applied to CPD is a non-negativity constraint [4, 19, 21]. This is done for a variety of reasons, most notably

as a form of dimension reduction and when analyzing data which are naturally predisposed to be non-negative. We consider both outcomes necessary for the model under consideration. First, the tensors used in this model are extremely sparse, and so introducing negative factors makes the search space for factors so large that fitting the model becomes intractable. Second, the tensors used in this model represent frequency data, which means that negative values in the factors would have no valid semantic meaning.

4.2.2 Representing Documents as Tensors

The text documents to be analyzed are represented as tensors by dividing them into phrases of length n. These phrases, commonly referred to as n-grams, are counted and their frequencies are entered into a tensor. Each word in the corpus vocabulary is assigned an index, and the tensors n modes refer to these indexes. For example, suppose $n = 3$. The document tensor $\mathcal{D}$ would have 3 modes. The entry d_{ijk} refers to the frequency of the n-gram consisting of words i, j, and k from the corpus vocabulary.

The tensors produced by this encoding will be cubic. Given a vocabulary consisting of v words, the resultant tensor will have v indexes in each mode. The tensor can represent the frequency of all possible n-grams, and as such will be extremely sparse as very few of these n-grams are likely to appear in a document. When decomposed into polyadic form, these tensors will yield sets of related words as well as related n-grams that appear in the same factors.

4.2.3 Modeling Influence

The basic model applied to the document begins with the decomposition of a document into its individual factor tensors.

$$\mathcal{D} = \sum \mathcal{F}_i \tag{4.6}$$

where $\mathcal{F}_i \in \mathrm{F}$ is a factor of the tensor $\mathcal{D}$, ($\mathcal{F}_i = a_i \otimes b_i \otimes c_i$). As has been previously noted, the model becomes more expressive by separating out a normalizing value λ_i from each f_i. Thus, the decomposed document becomes

$$\mathcal{D} = \sum \lambda_i \mathcal{F}'_i \,, \tag{4.7}$$

where $\mathcal{F}'_i = \frac{1}{|\mathcal{F}_i|}\mathcal{F}_i$ and $\lambda_i = |\mathcal{F}_i|$. This is desirable for two reasons. Given that all tensors within the corpus have the same dimensions, and that all are decomposed using Non-Negative CPD, the factors occupy the same type of space as the factors of

other documents. By normalizing them into a proportional model of the document, the factors become directly comparable irrespective of the magnitude of influence they exert in their source document.

Let C be a corpus of documents, encoded as tensors $\mathcal{D}_j \in \mathrm{C}$. Let $\mathcal{D}_t$ be the target document to be studied, and all other documents in $\mathrm{S} = \mathrm{C} - \mathcal{D}_t$ are treated as source documents for $\mathcal{D}_t$. The goal of the influence model is to ascribe the factors of $\mathcal{D}_t$ to a factor from each source document $\mathcal{D}_s \in \mathrm{S}$ and assign weights to each of the source document influences. Each document in C is decomposed as per Eq. 4.7. $\mathcal{D}_t$ is also decomposed into its components. This produces sets of factors F'_s and Λ_s for each source document as well as F'_t and Λ_t for the target document. By measuring the similarity of each factor $f'_t \in \mathrm{F}'_t$ and source factors $f'_s \in \mathrm{F}'_s$ in every F'_s, each factor can be ascribed to originating in a source document $\mathcal{D}_s$ or as being original to $\mathcal{D}_t$. Using the similarity measurements between the f'_t and f'_s factors, each corresponding f_t factor of $\mathcal{D}_t$ is categorized as either belonging to one of several sets: F_t^s for all factors of $\mathcal{D}_t$ ascribed to some factor of $\mathcal{D}_s$ and F_t^n for all factors with no matching source.

For each of these factor sets, tensors can be formed by summing over the set. For every F_t^s, the tensor $\mathcal{F}_t^s$ is the sum of all components of document t ascribed to document s.

Hence, the model of the target document can be expressed as

$$\mathcal{D}_t \approx \sum_{s=1}^{|\mathrm{S}|} \mathcal{F}_t^s + \mathcal{F}_t^n \,. \tag{4.8}$$

Normalizing as in the previous equations, the target document's model becomes

$$\mathcal{D}_t \approx \sum_{s=1}^{|\mathrm{S}|} \lambda_t^s \mathcal{F}_t'^s + \lambda_t^n \mathcal{F}_t'^n \,. \tag{4.9}$$

These new factor tensors, which are no longer necessarily rank 1 tensors, contain the proportions of related n-grams, separated into components according to their attributed source. This comprises the sought-after semantic model of the document.

Given these new factors, the influence of each document is extracted by

$$\Lambda_t = (\lambda_1, \lambda_2, \ldots \lambda_{|S|}, \lambda_t) \,. \tag{4.10}$$

Weights for each document are then extracted as their proportion of importance to the target document.

$$\mathrm{W} = \frac{1}{\sum \Lambda_t} \Lambda_t \tag{4.11}$$

Note that the weights from the source documents are not used. Essentially, the only purpose the source document factors serve is to classify the factors of the target document. Having accomplished the classification step, constructed the model in Eq. 4.9, and extracted the weights in Eq. 4.11, the target document has been decomposed into a set of tensors which identify both the semantic shape of each contribution and its corresponding weight.

4.2.4 Summary of Influence Modeling Procedure

Generating the influence model can be subdivided into the following steps:

1. Encode each document in the corpus as a tensor.
2. Decompose each document using non-negative CPD.
3. Classify each factor of the target document $\mathcal{D}_t$ as either belonging to a source document or as an original contribution of the author.
4. Extract weights from each subset of factors to determine the influence of each class of factors.

An overview of each of these steps is provided in Sect. 4.4 and the reader is referred to [22] for a more detailed discussion of the modeling procedure.

4.3 Related Work

Much of the inspiration for frequency based analysis for authorship detection comes from the work of John Burrows and Hugh Craig [5, 6, 10]. In their papers, Burrows and Craig utilize a variety of numerical techniques to explore marker words, and they use frequencies of marker words coupled with T-distribution sampling to provide an argument for attributing authorship of disputed works. They explore a variety of literary works, ranging from poetry to Shakespeare's plays. (Most of their focus is on Elizabethan and Victorian era works.) In these studies, the frequencies explored are based on single marker words, and the words are extracted based on how unique they are to the authors in question.

For n-gram classification, Noriaki's Kawamae's paper has shown that n-grams are capable of building a generative topic model of a corpus of documents [17]. Kawamae's work shows that a combination of n-gram and word frequencies reveals information about a corpus's structure, especially hierarchical information pertaining to topics within the corpus. Kawamae's model builds a tree with probabilistic relationships which are then used to infer information about the structure of a corpus, and shows that n-grams provide a sufficient basis for modeling the transfer of ideas through a corpus.

Another related n-gram study was performed by Antonia et al. [2]. In this paper, Antonia et al. attempt to reproduce marker word studies using n-gram frequencies in

place of word frequencies. They were able to show that n-gram frequencies are able to identify stylistic signatures of contributors to a text document. When $n = 1$, their model is equivalent to marker words, and as they increase n, they retest to determine how expressive the model is. They noted that there is no one length that seems to give the best performance in all cases when analyzing English language documents. Their results show that 1-, 2-, and 3-gram analysis tends to work well, but when exploring longer phrases the power of the model drops off. Even in instances where 1-grams or 2-grams are best, 3-grams are still a reasonable choice and are their recommendation for testing. The analysis performed in Antonia et al.'s work was conducted using delta and zeta tests as was established in the standard marker word approach. As such, only the most frequent n-grams of each author were explored, and they were only used as an evidentiary marker of an author's participation.

4.4 Influence Model

The influence model is governed by a document list and a set of parameters. The inputs to the model are described in Table 4.1. The document list contains a list of all of the documents in the corpus and is comprised of potential source documents and one target document. The target document is placed at the end of the document list by convention.

The generated model's output consists of the set of factors which have been found to influence the target document, the weights of each document's influence on the target document, and the set of factors found from the decomposition of the document tensors. The output variables of the generated model are described in Table 4.2.

Table 4.1 Model input

Parameter	Explanation
$docs$	A list of documents in the corpus. The target document is the final entry in the list
n	The number of modes to use in tensor construction
$nfactors$	The number of factors for tensor decomposition
$threshold$	The threshold value for factor matching

Table 4.2 Model output

Parameter	Explanation
W	Set of weights of each factor of the target document W_i is the weight of target document factor i
S	The set of source indexes for each factor S_i is the index of the source factor, 0 if the factor is unique to the target document
F	The set of all document factor tensors

input : $docs, n, nfactors, threshold$
output: W, S, F

`prepare(`$docs$`);`
V ← `build_vocabulary(`$docs$`);`
C ← ∅;
foreach *d in docs* **do**
  $\mathcal{D}$ ← `build_tensor` $(d, n,$ V);
  C ← C ∪ $\{\mathcal{D}\}$;
end
Λ,F ← `extract_factors(`C, $nfactors$`);`
M ← `build_distance_matrix(`F`);`
λ ← the entries in Λ corresponding to the target document.;
W, S ← `extract_influence(`$|docs|$, M,F,λ, $threshold$`);`
return W, S, F;

Algorithm 1: Influence model construction

4.4.1 Approach Overview and Document Preparation

The overall algorithm is described in Algorithm 1. The principal activities in the model building process are document preparation, tensor construction, and influence extraction.

The first step is to prepare the document corpus for processing. Documents are left mostly intact with the only filtering being to remove punctuation, numbers, and convert all letters to lowercase. The document strings are then treated as a list of lowercase words and will be treated as such for the rest of this discussion. Note that none of the words, including stop words, of the document are removed during filtering, and no stemming is performed. The reason for this is that these elements go to the style of the author. In fact, both stop words and un-stemmed words have been shown to be powerful markers for authorship and style [2, 6, 24].

In order to build tensors, a vocabulary is first extracted from the corpus. The vocabulary is simply the set of all words within the corpus. The set V contains a single entry for each word. The index of each word is used in the next step to create tensor representation of each document.

4.4.2 Tensor Construction

Following the preparation of the corpus and vocabulary extraction, the next step is key to the construction of the tensor model. In this step, each document is represented as a tensor. The tensor is constructed with n modes where each mode contains |V| dimensions. For example, 3-mode tensor over a 30-word vocabulary would result in a $30 \times 30 \times 30$ tensor. This is used to count the frequency of n-grams within each document. Thus, entry $\mathcal{D}_{ijk}$ counts the number of occurrences of the phrase $\mathrm{V}_i \mathrm{V}_j \mathrm{V}_k$ within the document.

input : d, n, V, n
output: $\mathcal{D}$

$\mathcal{D} \leftarrow$ Tensor with dimension $|\mathrm{V}| \times |\mathrm{V}| \ldots \times_n |\mathrm{V}|$;
Fill $\mathcal{D}$ with 0;
$len \leftarrow$ number of words in d;
for $i \leftarrow 1$ *to* $len - n$ **do**
  /* Compute Tensor Element Index */
  $index \leftarrow$ list of n integers;
  for $j \leftarrow 1$ *to* n **do**
    $index[j] \leftarrow$ index of word $d[i]$ in V;
  end
  /* Update Frequency of This n-gram */
  $\mathcal{D}[index] \leftarrow \mathcal{D}[index] + 1$;
end
return $\mathcal{D}$

Algorithm 2: Build tensor

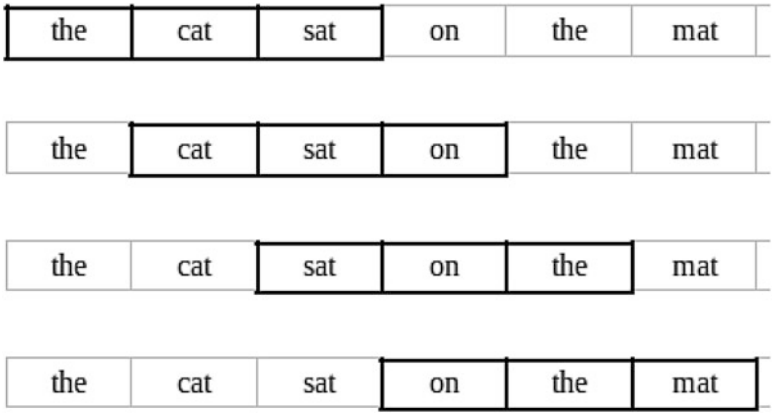


Fig. 4.2 Sliding window

The construction of this tensor is detailed in Algorithm 2. The tensor is constructed using a sliding window, beginning with the first word of the document and proceeding until n words from the end of the document. This process is illustrated in Fig. 4.2.

The resultant tensor is typically very sparse as it counts the frequency of all possible n-grams over the vocabulary V, of course few (if any) documents would use every possible n length combination of its vocabulary as most of these phrases would be nonsensical. Thus the set of tensors C produced by Algorithm 2 is a set of sparse tensors representing the document corpus. The last tensor in the set is the representation of the target document as it has been placed at the end of the document list, as mentioned previously.

4.4.3 Tensor Decomposition

The next step in the process is to decompose each document tensor into rank-1 components. As noted in Sect. 4.1, several decompositions exist which can accomplish this task. Because the data the present model consumes is frequency counting, the tensors comprising the corpus are all strictly non-negative. This arrangement naturally lends itself to non-negative factorization. Moreover, as these tensors are expected to be very sparse, any factorization that admits negative numbers would likely take an intractably long time to converge as the standard methods of decomposition would explore many combinations of factors which sum to zero. In fact, the magnitude of these alternating negative and positive factors would necessarily dominate the model and would make comparison of factors very difficult while providing no useful information about the underlying document. Therefore, not only is non-negative factorization a logical choice, it is also a necessary choice to ensure the expressiveness of the resultant model.

The method of non-negative factorization employed in this model is the Column-wise Coordinate Descent (CCD) method described in Ji Liu et al.'s paper [21]. The CCD method decomposes a tensor $\mathcal{A}$ into a core tensor $\mathcal{C}$ and a set of factor matrices $U_{1\ldots m}$ where m is the number of modes. The result of this decomposition is shown in Eq. 4.12. The objective of the model is to minimize the error tensor $\mathcal{E}$, and CCD accomplishes this by iteratively solving for U_i by holding the other factor matrices constant. The optimal solution for each entry of U_i is determined by a differential equation which is solved iteratively as it has no closed form solution. The key advantage to the CCD method is that rows in each column are independent, and so entire columns can be solved in parallel. CCD also allows for L_1 sparsity constraints to be applied, though this is not used in the present model. (The L_1 penalty is set to 0 for this model.)

$$\mathcal{A} = (\mathcal{C} \times_1 U_1 \times_2 \ldots \times_m U_m) - \mathcal{E} \tag{4.12}$$

Note that the CCD model is a non-negative version of the Tucker decomposition. By constraining the CCD model to use a square identity tensor (of dimension $n \times n \times \ldots n$) for $\mathcal{C}$, the model becomes equivalent to the non-negative canonical polyadic decomposition. Each U_i matrix will contain n columns, and when this product is carried out, it can be rewritten as the sum of the tensor product of the columns of U. That is, the equivalent CP model can be expressed using Eq. 4.13.

$$\mathcal{A} = \sum_{i=1}^{n} U^1_{:,i} \otimes U^2_{:,i} \otimes \ldots \otimes U^m_{:,i} - \mathcal{E} \tag{4.13}$$

Having extracted the rank-1 tensors which approximate $\mathcal{A}$, the last remaining step is to normalize these factors. The norm used in this model is the L_1 norm.

Separating these out, the final approximation of each document tensor is shown in Eq. 4.14.

$$\mathcal{A} \approx \sum_{i=1}^{r} \lambda_i \mathcal{F}_i \tag{4.14}$$

The L_1 norm is used here, and in the distance calculation in a later step, because it produces a sparse solution. The tensors being factored in this model are already sparse, and the non-negative decomposition produces factors which are also sparse. Using the L_2 norm would tend to allow large differences between the factors to dominate the model's selection of related factors when the differences between factors are computed. Because the lower sensitivity of the L_1 norm is desired in the distance calculation, it is also used here. Doing so sets the range of distances between factors to the interval [0, 2].

The entire process of the construction of these factors is shown in Algorithm 3. The result is an L_1-normalized set of rank-1 tensors.

As noted in Sect. 4.1, the number of factors determines the uniqueness of the decomposition. In the case of canonical polyadic decomposition, the solution is unique if the number of factors exceeds the tensor rank. However, computing the tensor rank is intractable, and so it must be approximated through trial and error. One rule of thumb for a 3-mode tensor (used in this case study) is that its expected minimal rank is given in Eq. 4.15 [9].

```
input  : C, nfactors
output: Λ, F
F ← ∅;
Λ ← ∅;
nmodes ← number of modes in C[1];
foreach D in C do
    U ← ccd_ntfd(D, nfactors);
    for i = 1 to nfactors do
        /* Build the Factor                                    */
        T ← U[1][:, i];
        for m = 2 to nmodes do
            T ← T ⊗ U[m][:, i];
        end
        /* Compute the norm and normalize the factor           */
        λ ←L1_norm(T);
        T ← T/λ;
        /* Insert the factor and norm into the list            */
        F ← F ∪ {T};
        Λ ← Λ ∪ {λ};
    end
end
return Λ, F
```

Algorithm 3: Extract factors

$$R = \left\lceil \frac{IJK}{I + J + K - 2} \right\rceil \tag{4.15}$$

However, this estimate assumes a generic tensor and not a sparse tensor! In fact, in every instance of the document tensors used here, this minimal rank would far exceed the number of non-zero elements. This leaves trial and error as the only choice for determining the number of factors in this model given the current state of knowledge of sparse tensor rank. The approach used to find this number begins with assuming that the non-zero elements, nnz, of the document tensor were packed into a dense tensor with dimensions $\sqrt[3]{nnz} \times \sqrt[3]{nnz} \times \sqrt[3]{nnz}$. This starting point is then computed using Eq. 4.16. From this point, decompositions are attempted with increasing rank until the fit begins to become worse, or until the error ratio drops below 20% (Eq. 4.17). (Of course a different threshold could be used if desired.)

$$R_0 = \left\lceil \frac{nnz}{3\sqrt[3]{nnz} - 2} \right\rceil \tag{4.16}$$

$$\frac{|\mathcal{D} - \hat{\mathcal{D}}|}{|\mathcal{D}|} \tag{4.17}$$

Extending the starting point from Eq. 4.16 for tensors of arbitrary, n, modes yields Eq. 4.18. However, the rank of sparse tensors is very much an open question. There is no real theoretical basis for these equations other than sensible conjectures. The process of finding the number of modes is, for now, confined to a process of trial and error. The objective is always to find a model that fits reasonably well, and until the problem of sparse tensor rank is solved this is all that can be achieved without an exhaustive search of all possible ranks. These starting points do seem to yield good results in practice, and as 3-grams have been shown [22] to work best for author classification [2], 3-mode tensors (and Eq. 4.16) are used to test our model.

$$R_0 = \left\lceil \frac{nnz}{n\sqrt[n]{nnz} - 2} \right\rceil \tag{4.18}$$

4.4.4 Factor Classification

Having extracted factors from the document corpus, the next step is to classify each of the target document's factors as either belonging to the set F_t^s (target factors with sources) or F_t^n (target factors without sources). In order to do this, the similarity of each factor pair must be measured. Because each factor has the same dimensions, and each factor's modes represent indexes over the same vocabulary, they can be compared by distance from each other within the factor space. Algorithm 4

```
input  : F
output: M
M ← Matrix with dimension |F| × |F|;
for i = 1 to |F| do
    for j = 1 to |F| do
        M[i, j] ← L1_norm(F[i] − F[j]);
    end
end
return M
```

Algorithm 4: Build distance matrix

accomplishes this task by finding the L_1 distance between each pair of factors. The result is a matrix M where M_{ij} is the L_1 distance between F_i and F_j. Of course, this matrix will have zeroes on the diagonal. Because each factor is non-negative and already L_1 normalized, $0 \leq M_{ij} \leq 2$, where 0 is a perfect match and 2 indicates maximum distance.

In actuality, only the entries corresponding to the target document factors are necessary. That is, $M[i, j]$ is only needed where i is the index of a target factor and j is the index of a potential source factor. The entire distance matrix is useful for studying the distribution of factor distances which is useful in finding model thresholds as well as quantifying the uniqueness of each factor.

The final task to be performed in constructing the model is to identify which source factors are closest to each target factor and compute the corresponding weights of those factors. This task is carried out by Algorithm 5. The basic strategy is for each factor in the target document to be assigned the source factor with the minimum distance. The only issue with this approach is it would always assign a source to a target factor, even though some target factors are expected to have no relatable source. For this reason, two steps are needed. First, the minimum is found, second it is compared against a threshold. If the minimum value is below this threshold, the factor is assigned a source. If, on the other hand, the minimum distance is above the threshold, it is not assigned a source.

The threshold value is a heuristic parameter which controls the matching of factors. Recall that the factors are L_1-normalized, and the distance computed between the factors is the L_1-distance. If two factors are a perfect match, this results in a distance of 0. If they are completely disparate, the result will be a maximum distance of 2. This latter arrangement implies that no non-zero entries in the factor tensors were found in the same position and would therefore signify completely unrelated factors. A sensible default setting for this threshold is 0.2 as this requires a 90% agreement of the entries. Another approach to selecting a threshold value is to examine the distribution of distances within the distance matrix and use that information to select the threshold. A threshold value of 0.2 was recommended in [22].

input : $ndocs$, M, F, λ, $threshold$
output: W, S

/* Compute Weights */
$sum \leftarrow \sum \lambda$;
W $\leftarrow \lambda / sum$;
S $\leftarrow$ list of integers of size $|\lambda|$;
/* Classify Factors */
$nfactors \leftarrow |\lambda|$;
for $i = 1$ *to* $nfactors$ **do**
  $min \leftarrow M[row, 1]$;
  $minIndex \leftarrow 1$;
  $row \leftarrow i + nfactors * (ndocs - 1)$;
  for $j = 1$ *to* $nfactors * ndocs$ **do**
    if M *[row,j]*$< min$ **then**
      $min \leftarrow M[row, j]$;
      $minIndex \leftarrow j$;
    end
  end
  if $min \leq threshold$ **then**
    S[i] $\leftarrow minIndex$;
  else
    S[i] $\leftarrow 0$;
  end
end
return W, S;

Algorithm 5: Extract influence

input : $ndocs$, S, W
output: I, $author$

I $\leftarrow$ List of 0 repeated $ndocs - 1$ times;
for i = *1 to ndocs* **do**
  if S[i] $= 0$ **then**
    $author = author +$ W[i];
  else
    $j \leftarrow$ Document number corresponding with S[i];
    I[j] $\leftarrow$ I[j] + W[i];
  end
end

Algorithm 6: Final summation

The result of the classification operation is the set W which is simply the normalized set of λ values for the target document, and the set S where entry S_i is the index of the factor which is the source for target factor i. If target factor i has no assignable source, then a value of 0 is written to position S_i.

After the factors have been matched, the final output of the model can be summarized by summing the influence of each source document and author contribution factor using Algorithm 6.

4.5 Implementation

Implementing the model described in the previous section comes with several challenges. The biggest challenge is the size of the tensors, as well as a lack of good support for sparse tensors in available software. Several packages were tried, but ultimately a custom tensor library was needed to support these tensors. The attempted software packages were Tensor Flow (Python), Tensor Toolbox (Matlab), SciPy/NumPy (Python). While all three packages provide support for sparse tensors, their operations are not well optimized for sparse tensor usage. Also, in some cases, tensors were converted into dense tensors before operations were performed. Unconstrained vocabularies can often have tens of thousands of words, which leads to a tensor which far exceeds the capacity of any machines available for this project. Before these libraries were abandoned, constrained vocabularies were attempted.

4.5.1 Constraining Vocabularies

As already stated, the tensors used in this model are very sparse. Essentially, every word in a document is the beginning of a new phrase, and so every document tensor will contain the same number of non-zero entries as there are words in the document. (With the exception being the last n-gram as each word following the first in the n-gram cannot be the start of a new n-gram.) Storing the frequency counts of these documents is trivial, but the model needs them in their positions within the tensor in order to decompose the documents into factors. Most of the complexity therefore lies with the dimensionality of the tensor which is driven by the size of the vocabulary.

Looking at the vocabularies in several documents during preliminary experiments showed that each document only had about five hundred to one thousand frequent words. By sorting the vocabulary in descending order by frequency, the vocabulary can be shortened with minimal disturbance to the structure of the document and the makeup of most of the n-grams. Constraining the vocabulary to 600 words when building a 3-gram tensor results in a tensor which has 216,000,000 potential entries. Even in dense format, this can be comfortably accommodated by the memory of even a modest modern desktop machine. However, a problem still remains with this approach. Decompositions of a tensor of this size, typically into 100–200 factors, requires too much time. When using Matlab on an 8-core 3.4 GHz Intel Linux machine with 16 GB of RAM, non-negative factorization tended to require about 2 h of elapsed wall-clock time, and so a faster method was still needed even in the constrained case. Searching for the optimal number of factors by multiple factorings proved even more challenging. This is especially challenging when considering that this model requires the decomposition of multiple documents within a corpus.

To address the problems of time and memory constraints, a new C-based software library was created [22]. This library, called *sptensor*, was written in ANSI C since it provides enough control for optimizing memory usage and it is well situated to have libraries for other languages bound to it. The sptensor library is designed to only represent sparse tensors, and so it has the opposite problem the common tensor libraries have. Dense tensors in sptensor are fairly inefficient both in time and space complexity. However, for our target application the library performs quite well. Where Matlab factorization of tensors over constrained vocabularies typically requires 1–2 h to complete on a modest desktop computer (3.4GHz), sptensor can accomplish the task in 5–10 min. This speedup allowed for the construction and testing of the model to proceed.

4.6 A Conference Paper Case Study

For a real-world case study, a conference paper was pulled from the ACM Digital library. This was the first paper listed in the first conference listed in their regional conference proceedings. This paper cites four other papers and two websites. The two websites were used to pull data, and so they are not included in the corpus. In addition to the four cited papers, two unrelated papers are included to test if the model will select factors from these unrelated papers. The complete corpus, listed with the target paper last, is shown in Table 4.3. Documents 1–4 are the papers cited by the target paper, documents 5–6 are unrelated papers, and document 7 is the target paper.

Table 4.3 Conference paper corpus

Num	Document information
1	Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In Proc. DMKD 2003, pages 2–11. ACM Press, 2003
2	Andreas Schlapbach and Horst Bunke. Using hmm based recognizers for writer identification and verification. In Proc. FHR 2004, pages 167–172. IEEE, 2004
3	Yusuke Manabe and Basabi Chakraborty. Identity detection from on-line handwriting time series. In Proc. SMCia 2008, pages 365–370. IEEE, 2008
4	Sami Gazzah and Najoua Ben Amara. Arabic handwriting texture analysis for writer identification using the DWT-lifting scheme. In Proc. ICDAR 2007, pages 1133–1137. IEEE, 2007
5	Kolda, Tamara Gibson. Multilinear operators for higher-order decompositions. 2006
6	Blei, David M and Ng, Andrew Y and Jordan, Michael I. Latent Dirichlet allocation. 2007
7	Serfas, Doug. Dynamic Biometric Recognition of Handwritten Digits Using Symbolic Aggregate Approximation. Proceedings of the ACM Southeast Conference 2017

Table 4.4 Conference model parameters

n	$nfactors$	$threshold$
3	150	0.2

Table 4.5 Conference classification results

Document	Influence	Number of matched factors
1	0.21	10
2	0.09	9
3	0.06	3
4	0.06	1
5	0.00	0
6	0.00	0
Author	0.57	127

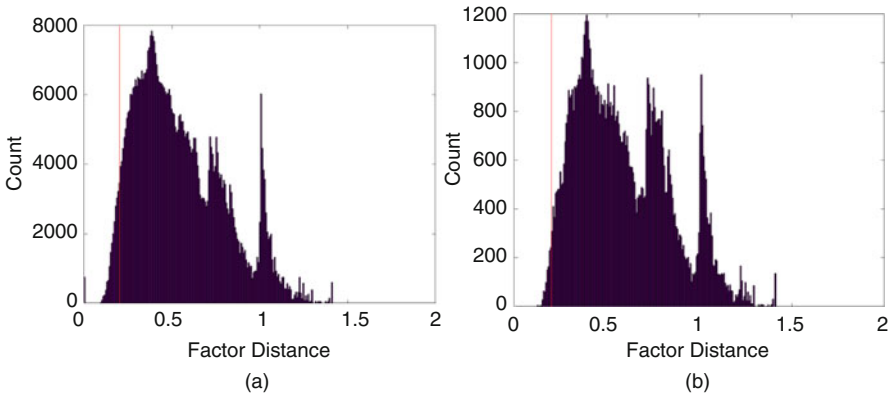


Fig. 4.3 Factor distance distribution. (**a**) All factor distances. (**b**) Target factor distances

The entire corpus consists of 45,152 words with the vocabulary truncated to 600 words (see Sect. 4.5.1). The 600 words were the most frequent words across the corpus. The other parameters are shown in Table 4.4.

Again, 0.2 is used as the threshold as it is a good default setting. The decomposition of the 7 documents into 150 factors was carried out on a machine with a 3.9 GHz 8-core Intel processor and 15 GB of RAM. The decomposition and construction of normalized tensors took approximately 2.5 h to complete. Calculation of the distance matrix and classifying the factors required another hour and a half. The results are shown in Table 4.5.

The distribution of the factor distances is shown in Fig. 4.3. The vertical red line on the graphs shows the threshold for factor matching. Figure 4.3a shows the distribution of factor distances for the entire factor matrix while Fig. 4.3b shows the distribution of the distances from the target factors. Note that both distributions are tri-modal. The two spikes on the far right correspond to factors from the unrelated documents, which shows that they are well separated from the target document's factors and from each other.

Given the separation of the factors, and that the model excluded two obviously unrelated papers, the model appears to have produced the desired results. For final

verification, the text of the documents in question will now be summarized. The target paper details an algorithm which is used to identify handwritten characters [23]. The algorithm presented is an extension of another algorithm, Symbolic Aggregate Approximation (SAX), which was first presented in Lin et al.'s paper [20]. Because the target paper is an extension of this one, a reasonable assumption would be that it would be heavily influenced by this source. This is precisely what the model stated in that this paper was assigned a weight of 0.21, and had a total of ten matching factors. The remaining citations that were found to influence the document present competing algorithms, and are primarily mentioned in the target paper's related works section. A summary of one of the most influential of the 10 matched factors is shown in Table 4.6. This is the factor 56 (out of 150) in the decomposition of the

Table 4.6 First 30 non-zero entries of factor 56

Word 1	Word 2	Word 3	Proportion
is	the	sax	0.000887
into	the	data	0.000886
symbols	the	sax	0.000874
digits	the	timeseries	0.000865
digits	the	data	0.000857
with	the	sax	0.000856
from	the	square	0.000852
however	the	sax	0.000844
up	the	sax	0.000844
characters	the	sax	0.000844
becomes	the	sax	0.000844
note	the	sax	0.000843
using	the	sax	0.000841
for	the	square	0.000838
into	the	sax	0.000838
from	the	svc	0.000833
from	the	paa	0.000832
author	the	square	0.000828
on	the	author	0.000824
for	the	svc	0.000819
for	the	paa	0.000818
on	the	accuracy	0.000814
on	the	array	0.000814
digits	the	sax	0.00081
author	the	svc	0.000809
author	the	paa	0.000808
on	the	distance	0.000806
on	the	x	0.000806
from	the	timeseries	0.000804
of	the	author	0.000802

target document and represents a weight of 0.04. The factor was sorted in decreasing order by proportion and only the first 30 n-grams are shown. (The actual factor contains 11,661 n-grams, most of which have very small proportional entries.) Note most of the n-grams are discussing the SAX algorithm, and various properties of it. In fact, this is what is found in the other nine factors, they all discuss different elements of SAX.

Another property of the target paper that bears examination is the makeup of the text itself. The paper is 4 pages long, the first page being devoted to frontmatter and the related works. The second page contains the conclusion of the related works section, which occupies approximately 25% of the page. The rest of the second page, and the entirety of the third page have the author's contributions and the conclusion. Half of the fourth page has the final conclusion paragraph, and then the bibliography. By rough estimate, therefore, the paper contains 1.75 pages of what is essentially the summary of existing work. This leaves 2.25 pages of original material, which means that a cursory analysis of the paper would imply that the author has contributed 56% of the text of the paper. The model's output weight for the author's contributions of 57% is in line with this rough estimate.

As these results show, the model makes a set of reasonable matches, and it does not select unrelated documents. The actual makeup of the factors is much more complex, however ideas can be traced through them. Unfortunately, these factors are much too large to be included verbatim in this chapter. However, the factors that were matched from paper one all deal with a technique which generates a symbolic representation of a time series. This technique serves as the basis for the invention in the target paper, and is mentioned several times with many of the same explanations used in the first paper. Thus the model has not only avoided unrelated information, it has given a greater weight to the paper which had the greatest semantic influence on the work being studied.

4.7 Conclusions and Future Work

Experimentation has shown that the factors discovered by non-negative tensor decomposition do in fact contain related n-grams. Moreover, the factors are unique enough that a match, or a near match within some heuristic bound, provides evidence of a relationship between factors. The related factors also make sense on an intuitive level, each having a clear semantic relationship to both target and source material.

The quantifications of the influencing factors also perform as expected. While no ground truth is available for a weighted mixture of source documents to the target documents, inspecting the documents in the corpus shows that the model's output reasonably matches the expectations of a human reader.

The immediate future plans for this research involve the further development of the *sptensor* library. Further optimization through the use of parallelization is planned. Following that, library bindings to higher level languages will be created.

Another application of the model is to replicate the studies conducted by Craig and Burrows [6, 10]. Addressing the problem of Shakespearean authorship using this model poses several unique challenges, not least of which is due to the inconsistencies in Elizabethan spelling (which necessitates the decomposition over full vocabularies). Additional applications will also be explored, including establishing chronologies of documents via topological sorting and modeling the influence flow through a hierarchical network of documents.

The effects of constrained vocabularies are another area which needs to be addressed. Following the authorship studies, another future effort will be to address the effect of the vocabulary size on the output of the model. Other aspects warranting further study are the effects of the various parameters of the model.

Finally, another possible extension of the model would be to the task of plagiarism detection. In this paradigm, the model could examine not only plagiarism from one source document, but would be able to take into account many potential documents of origin.

References

1. Adler, R., Ewing, J., & Taylor, P. (2009). Citation statistics. *Statistical Science, 24*(1), 1–14.
2. Antonia, A., Craig, H., & Elliott, J. (2014). Language chunking, data sparseness, and the value of a long marker list: Explorations with word n-grams and authorial attribution. *Literary and Linguistic Computing, 29*(2), 147–163.
3. Bader, B., Berry, M. W., & Browne, M. (2007). Discussion tracking in Enron email using PARAFAC. In M. W. Berry & M. Castellanos (Eds.), *Survey of Text Mining*, chapter 8 (pp. 147–163). Berlin: Springer.
4. Bro, R. (1997). Parafac. Tutorial and applications. *Chemometrics and Intelligent Laboratory Systems, 38*(2), 149–171.
5. Burrows, J. (2006). All the way through: Testing for authorship in different frequency strata. *Literary and Linguistic Computing, 22*(1), 27–47.
6. Burrows, J., & Craig, H. (2017). The joker in the pack?: Marlowe, Kyd, and the co-authorship of Henry VI, part 3. In G. Taylor & G. Egan (Eds.), *The New Oxford Shakespeare Authorship Companion*, chapter 11 (pp. 194–217). Oxford: Oxford University Press.
7. Cantuariensis, A. (c. 1078). *Proslogion*. Ordo Sancti Benedicti.
8. Carroll, J. D., & Chang, J.-J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika, 35*(3), 283–319.
9. Comon, P., Luciani, X., & De Almeida, A. L. (2009). Tensor decompositions, alternating least squares and other tales. *Journal of Chemometrics, 23*(7–8), 393–405.
10. Craig, H., & Kinney, A. F. (2009). *Sheakespeare, Computers, and the Mystery of Authorship*. Cambridge: Cambridge University Press.
11. Dietz, L., Bickel, S., & Scheffer, T. (2007). Unsupervised prediction of citation influences. In *Proceedings of the 24th International Conference on Machine Learning* (pp. 233–240). New York: ACM.
12. Harshman, R. A. (1970). Foundations of the parafac procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16.
13. Harshman, R. A., & Lundy, M. E. (1994). Parafac: Parallel factor analysis. *Computational Statistics & Data Analysis, 18*(1):39–72.
14. Håstad, J. (1990). Tensor rank is NP-complete. *Journal of Algorithms, 11*(4), 644–654.

15. Hitchcock, F. L. (1927). The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics, 6*(1–4), 164–189.
16. Jiang, Z., Liu, X., & Gao, L. (2014). Dynamic topic/citation influence modeling for chronological citation recommendation. In *Proceedings of the 5th International Workshop on Web-scale Knowledge Representation Retrieval & Reasoning* (pp. 15–18). New York: ACM.
17. Kawamae, N. (2016). N-gram over context. In *Proceedings of the 25th International Conference on World Wide Web* (pp. 1045–1055). New York: International World Wide Web Conferences Steering Committee.
18. Kiers, H. A. (1991). Hierarchical relations among three-way methods. *Psychometrika, 56*(3), 449–470.
19. Kolda, T. G., & Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review, 51*(3), 455–500.
20. Lin, J., Keogh, E., Lonardi, S., & Chiu, B. (2003). A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research Issues in Data Mining and Knowledge Discovery* (pp. 2–11). New York: ACM.
21. Liu, J., Liu, J., Wonka, P., & Ye, J. (2012). Sparse non-negative tensor factorization using columnwise coordinate descent. *Pattern Recognition, 45*(1), 649–656.
22. Lowe, R. E. (2018). *Textual Influence Modeling Through Non-Negative Tensor Decomposition*. PhD thesis, University of Tennessee, Knoxville.
23. Serfass, D. (2017). Dynamic biometric recognition of handwritten digits using symbolic aggregate approximation. In *Proceedings of the SouthEast Conference* (pp. 1–4). New York: ACM.
24. Stamatatos, E. (2011). Plagiarism detection based on structural information. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management* (pp. 1221–1230). New York: ACM.

Part II
Applications

Chapter 5
Survival Support Vector Machines: A Simulation Study and Its Health-Related Application

Dedy Dwi Prastyo, Halwa Annisa Khoiri, Santi Wulan Purnami, Suhartono, Soo-Fen Fam, and Novri Suhermi

5.1 Introduction

The support vector machine (SVM) is applied in many fields and has become a state-of-the-art classification method. In its development, the SVM is extended to support vector regression (SVR), which is employed for a prediction of cross-section independent observations. Recently, the SVR has been applied to forecast time-series data that are typically auto-correlated. The feature selection in SVR, for both cross-section and time-series data, has become a new issue that is attracting much attention from researchers and practitioners.

One of the main field applications of SVM or SVR is health. The SVM and its extension are employed to discriminate patients who potentially suffer from specific diseases. One of the most popular applications in health is survival data analysis. Many statistical methods have been developed to analyze survival data. These methods can be categorized into parametric, semi-parametric, and non-parametric approaches.

The parametric approach to survival analysis requires knowledge about the distribution of survival time. This requirement sometimes cannot be met in real applications because the goodness-of-fit test fails to find the distribution under the null hypothesis. This requirement can be considered a drawback [1–3]. Therefore, semi-parametric approaches are introduced. One of the most popular semi-parametric models is the Cox proportional hazards model (Cox PHM). However,

D. D. Prastyo (✉) · H. A. Khoiri · S. W. Purnami · Suhartono · N. Suhermi
Department of Statistics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia
e-mail: dedy-dp@statistika.its.ac.id

S.-F. Fam
Faculty of Technology Management and Technopreneurship, Universiti Teknikal Malaysia Melaka, Durian Tunggal, Melaka, Malaysia

M. W. Berry et al. (eds.), *Supervised and Unsupervised Learning for Data Science*, Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-030-22475-2_5

the Cox PHM has a proportional hazard (PH) assumption that needs to be satisfied. This assumption sometimes fails to be met in real data. In such a case, the non-parametric approach can play a role.

The SVM is one of the popular non-parametric approaches in classification that has a global optimal solution [4]. The SVM is then extended for regression [5]. Previous studies [2, 3, 6, 7] employed and extended SVR for survival data analysis. Later, the least square-SVM (LS-SVM) is applied to survival data, the so-called survival least square support vector machine (SURLS-SVM), because it has equality constraints [8] instead of inequality constraints as in the SVM. The SURLS-SVM employs a prognostic index instead of a hazard function.

Many existing articles do not explain the effect of each predictor on the performance of the SURLS-SVM. Feature selection can be employed to explain the contribution of each predictor. The feature selections commonly used in SVM are the filter and wrapper approaches. Goli et al. [9] applied it to breast cancer data. In this work, the SURLS-SVM is applied to cervical cancer (CC) data with Cox PHM as a benchmark model as in Khotimah et al. [10, 11] with a more detailed explanation. The empirical results show that the SURLS-SVM outperforms the Cox PHM before and after feature selection.

5.2 SURLS-SVM for Survival Analysis

One of the main functions calculated in survival data analysis is the survival function defined as the probability of failure time greater than time t, denoted as $S(t) = P(T > t) = 1 - F(t)$, with T denoting failure time and $F(t)$ its cumulative distribution function. Another important function is the hazard function $h(t)$, which shows the failure rate instantaneously after objects survive until time t. The relationship between hazard function and survival function is $h(t) = f(t)/S(t)$, with $f(t)$ being the first derivative of $F(t)$, also known as the probability density function (pdf). Given that there are features $\mathbf{x}$ that affect the hazard function, one of the most popular models for building a relationship between hazard function and features as input is the Cox PHM as in (5.1) below:

$$h\left(t, \mathbf{x}\right) = h_0(t) \exp\left(\boldsymbol{\beta}'\mathbf{x}\right), \tag{5.1}$$

with $h_0(t)$ being the baseline hazard and $\boldsymbol{\beta}' = (\beta_1, \beta_2, \ \ldots \ , \beta_d)$ a vector of coefficients.

The SVM for classification is obtained by solving the following optimization problem [4, 5, 12]:

$$\min_{w,b,\xi} \frac{1}{2} \| \mathbf{w} \|^2 + \gamma \sum_{i=1}^{n} \xi_i \tag{5.2}$$

subject to $y_i(\mathbf{x}_i' \mathbf{w} + b) \geq 1 - \xi_i; \xi_i \geq 0$. The optimization in (5.1) can be extended to find a non-linear classifier by transforming optimization lying in data space ($\mathbf{x}_i \in R^d$) to feature space (higher dimension, $\mathscr{H}$) using the kernel trick [12], i.e. $\varphi(\mathbf{x}_i): R^d \rightarrow \mathscr{H}$. Owing to the drawback in SVM optimization, the least square (LS) idea is then applied to the LS-SVM [8], which is more efficient. The LS-SVM can be solved using linear programming with the objective function formulated in (5.3) as follows:

$$\min_{w,\xi} \frac{1}{2} \| \mathbf{w} \|^2 + \frac{1}{2}\gamma \sum_{i=1}^{n} \xi_i^2 \tag{5.3}$$

subject to $y_i[\varphi(\mathbf{x}_i)'\mathbf{w} + b] = 1 - \xi_i; i = 1, 2, \ldots, n$.

The rapid development in survival data analysis attracts the attention of researchers to extend the SVM such that it complies with survival data analysis. The survival SVM (SUR-SVM) extends Eq. (5.1) into Eq. (5.4) as follows [6]:

$$\min_{w,\xi} \frac{1}{2}\mathbf{w}'\mathbf{w} + \frac{\gamma}{2} \sum_{i} \sum_{i<j} v_{ij}\xi_{ij}; \gamma \geq 0 \tag{5.4}$$

subject to $\mathbf{w}'\varphi(\mathbf{x}_j) - \mathbf{w}'\varphi(\mathbf{x}_i) \geq 1 - \xi_{ij}; \forall\, i<j; \xi_{ij} \geq 0; \forall\, i<j$. The indicator variable v_{ij} plays a role as an indication of whether or not two subjects are comparable. It is formulated in Eq. (5.5) as follows [6, 7]:

$$v_{ij} = \begin{cases} 1; \left(\delta_i = 1, t_i < t_j\right) \text{ or } \left(\delta_j = 1, t_j < t_i\right) \\ 0; \text{otherwise} \end{cases}, \tag{5.5}$$

where δ is censored status with value $\delta = 1$ when an event occurs and $\delta = 0$ for censoring.

The least-square version of SUR-SVM is so-called survival least square SVM (SURLS-SVM). It has equality constraints that make it simpler. Rather than modeling the hazard function as in Cox PHM, the SURLS-SVM uses the prognostic index (also known as the health index in the literature) [2, 6] to predict the rank of observed survival time given the known features. The prognostic function is defined as follows:

$$u(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x}), \tag{5.6}$$

with $u: R^d \rightarrow R$, the w is weight vector, and $\varphi(x)$ is feature mapping. The prognostic function increases when the failure time increases. Let two samples (i, j) and the event deaths have a prognostic index $(u(x_i), \ u(x_j))$ and survival time $(t_i, \ t_j)$. If $t_i < t_j$, then $u(x_i) < u(x_j)$ is expected.

In survival data analysis, the prediction of survival time is difficult. Therefore, the study can be carried out by predicting the rank of the prognostic index corresponding

to the observed survival time [13]. The Eq. (5.7) formulates the optimization of SURLS-SVM [6]:

$$\min_{w,\xi} \frac{1}{2}\mathbf{w}'\mathbf{w} + \frac{\gamma}{2}\sum_{i}\sum_{i<j} v_{ij}\xi_{ij}{}^2; \gamma \geq 0 \tag{5.7}$$

subject to $\mathbf{w}'\varphi(\mathbf{x}_j) - \mathbf{w}'\varphi(\mathbf{x}_i) = 1 - \xi_{ij}; \quad \forall\, i < j$, with γ being a regularization parameter and v_{ij} is an indicator variable as defined in eq. (5.5). The ranking is obtained if $\mathbf{w}'\varphi(\mathbf{x}_j) - \mathbf{w}'\varphi(\mathbf{x}_i) > 0$ is satisfied. The next procedure can be read in Prastyo et al. [14].

5.3 Data Description and Methodology

The procedure to generate survival time follows the algorithm proposed by Bender et al. [15]. Prastyo et al. [14] performed a simulation by generating 17 features, survival time, and censored status. The censored state (1 for death and 0 for alive) is generated on an increasing percentage from 10% to 90% in 10% increments. The survival time is generated based on the Cox model with Weibull distribution. The coefficients corresponding to each feature are set. The features are the combination of a categorical and a continuous scale. To impose non-linearity on a features pattern, the two interactions between predictors are considered. The performance of the model is measured using the concordance index (C-index).

The performance of the proposed approach is measured using the C-index produced from the prognostic index [2, 3, 6, 14]. The higher the C-index delivered the better the performance of the method. The performance can be improved by applying feature selection to gain essential predictors. There are several feature selection methods such as filter methods, wrapper methods, and embedded methods [13]. In this study, the feature selection is done with the wrapper method using backward selection. The backward elimination is selected because this method can identify suppressor variables, while forward and stepwise elimination cannot. The suppressor features have a significant effect when all of them are included in the model [16].

The backward elimination procedure is started by choosing all features as input. Next, removing each feature and calculating the C-index resulted from each all-minus-one-features input. For example, if there are four features (x_1, x_2, x_3, x_4), then the all-minus-one-features are the set of (x_2, x_3, x_4), (x_1, x_3, x_4), (x_1, x_2, x_4) and (x_1, x_2, x_3). Based on the difference between the C-index calculated from all features and all-minus-one-features, remove the least significant feature, i.e., a missing feature that causes the C-index to increase the most. The procedure is repeated for removal of the next feature. The algorithm terminates when the C-index cannot be improved further. The performance of the proposed approach is validated by the simulation study involving feature selection with backward elimination. The simulation is repeated 100 times.

This study extends the previous work [14] in terms of the simulation setting. In addition to the percentage censorship, the empirical results of the effect of features dimension and sample size on the performance of SURLS-SVM are provided. Furthermore, the tuning parameters in SURLS-SVM are optimized using a grid search with C-index as an evaluation criterion.

The application of SURLS-SVM to health data refers to the work of Prastyo et al. [14], who applied the proposed method to a cervical cancer (CC) case study with 412 records. Recall the information in Prastyo et al. [14]; the inclusion criteria in the data are: (1) female patients, (2) the event of interest is death, and (3) patients' complete medical records are used as a predictor. The data are right-censored. Twenty-seven patients died (7%), and 385 (93%) patients survived. All features are described as follows:

- P_1: Age
- P_2: Complication status
- P_3: Anemia status
- P_4: Type of treatment {chemotherapy, transfusion, chemotherapy and transfusion, others}
- P_5: Stadium level
- P_6: Age at marriage as the proxy for first sexual intercourse
- P_7: Age at first menstrual period
- P_8: Menstruation cycle
- P_9: Duration of menstruation
- P_{10}: Parity
- P_{11}: Family planning status
- P_{12}: Education level (elementary school, junior high school, senior high school or higher).

5.4 Empirical Results

5.4.1 Effect of Features Dimension and Sample Size

By using the same simulation setup as in Prastyo et al. [14], data with a linear pattern are analyzed using Cox PHM and SURLS-SVM. Figure 5.1 displays its performance measures. The data sets have 0.5 censoring rates, using the kernel parameter and regularization parameters 0.5 and 0.1 respectively for SURLS-SVM. The plots on the left side exhibit performance measures of Cox PHM and SURLS-SVM when using a sample size of 100 whereas the plots on the right side are for a sample size of 1000. The SURLS-SVM has better performance than Cox for all numbers of features (for sample sizes of both 100 and 1000). The hazard ratio of Cox PHM has higher value than the one of SURLS-SVM for all different number of features. When using a sample size of 100, the hazard ratio of Cox PHM is almost the same as that of SURLS-SVM for data with 15 and 17 features. The hazard ratio of Cox PHM decreases sharply for six and eight features, whereas that of SURLS-

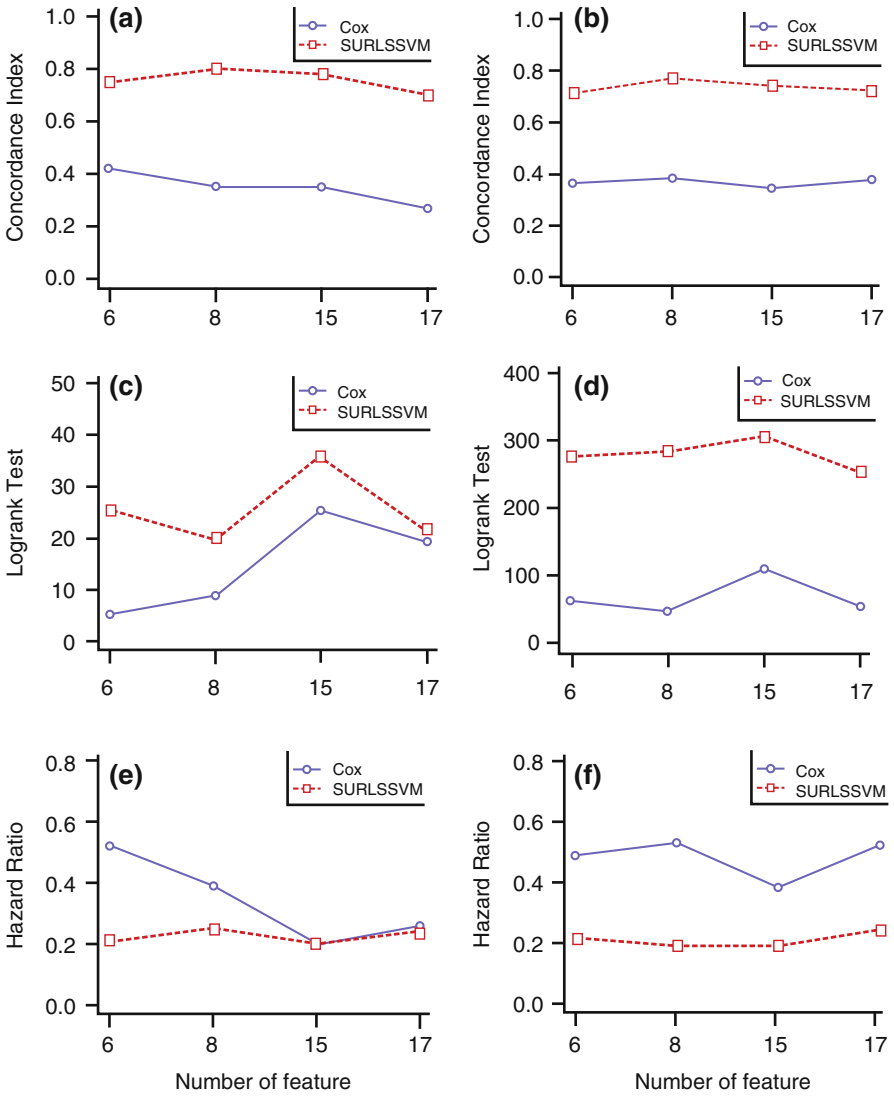


Fig. 5.1 Performance of Cox proportional hazards model (PHM) and survival least square support vector machine (SURLS-SVM) on data with a linear pattern using 6, 8, 15, and 17 features for a sample size of 100 (**a**, **c**, and **e**) and for a sample size of 1000 (**b**, **d**, and **f**)

SVM increases slightly as the number of features increases. In the sample size of 1000, the hazard ratio of Cox is higher than that of SURLS-SVM.

Figure 5.2 visualizes the performance measures of data with a non-linear pattern generated using various numbers of features (6, 8, 15, and 17) with a censoring percentage of 0.5. The left plots exhibit performance measures of a sample size of 100. The C-index and hazard ratio of SURLS-SVM outperform those of Cox PHM.

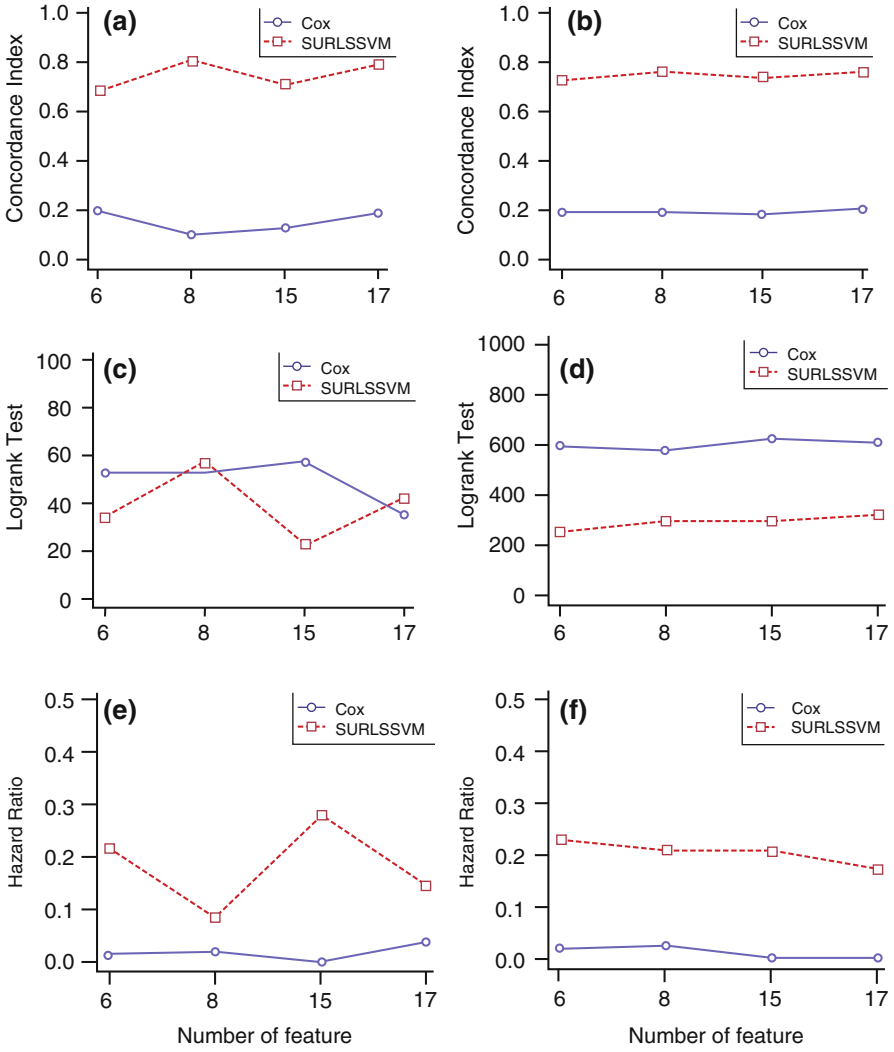


Fig. 5.2 Performance of Cox PHM and SURLS-SVM on data with a non-linear pattern using 6, 8, 15, and 17 features for a sample size of 1000 (**a**, **c**, and **e**) and for a sample size of 1000 (**b**, **d**, and **f**)

The values of the C-index and hazard ratio do not depend on the features dimension. The log-rank of SURLS-SVM is not always higher than that of Cox PHM. When using 8 and 17 features, the SURLS-SVM yields a slightly better result. However, it does not happen in data with 6 and 15 features (where Cox PHM is much better than SURLS-SVM). The right side of Fig. 5.2 displays performance measures when using a 1000 sample size. It shows the same information obtained from 100 samples on the left side, except for the log-rank test.

5.4.2 *Effect of Censoring Percentage*

Figure 5.3 displays performance measures of data with a linear pattern using six features with a different censoring percentage. The left plots produce the performance measures of a sample size of 100. The C-index of SURLS-SVM always yields a better result than that of Cox PHM at all censoring percentage levels. The hazard ratio of Cox PHM outperforms SURLS-SVM in a high censoring

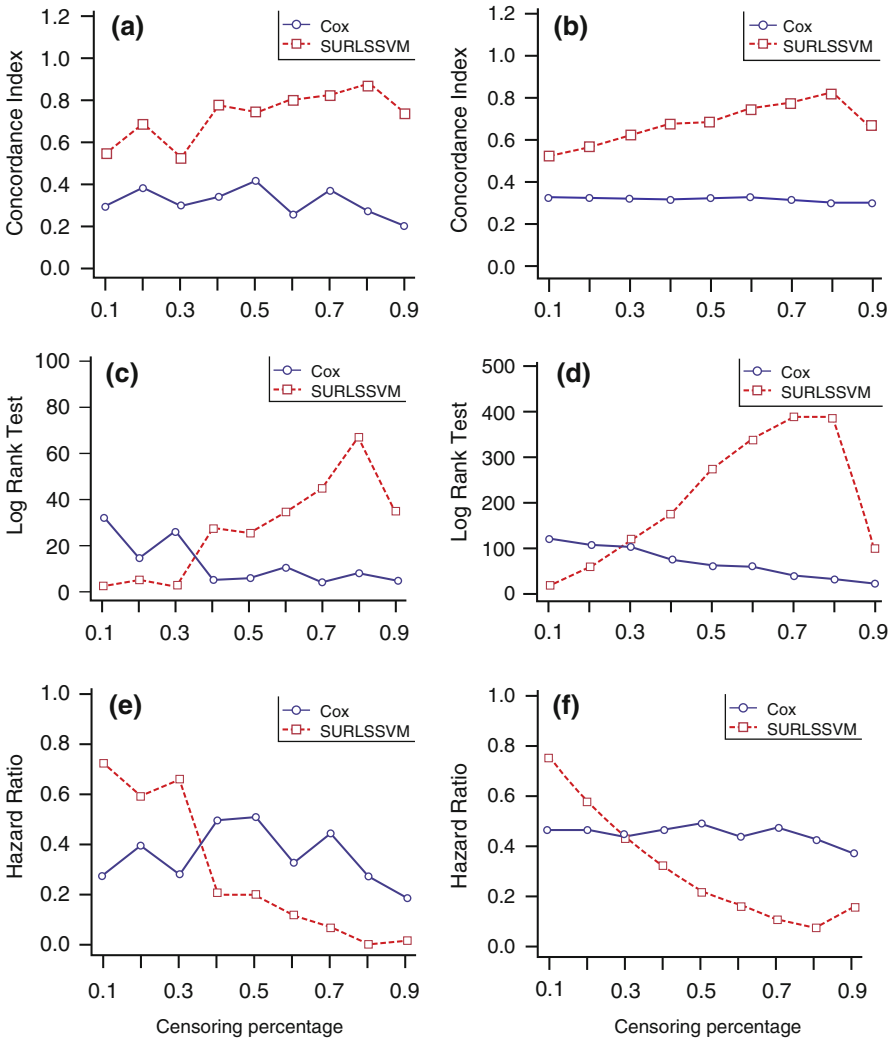


Fig. 5.3 Performance of Cox PHM and SURLS-SVM on data with a linear pattern using different censoring percentages for a sample size of 100 (**a**, **c**, **e**) and for a sample size of 1000 (**b**, **d**, **f**)

percentage (greater than 0.3). The SURLS-SVM indicates a better result when it is applied to data with a low censoring percentage (on the plot it is less than 0.3). The hazard ratio of SURLS-SVM decreases as a censoring percentage increase, except when the censoring percentage is equal to 0.9, but the hazard ratio of Cox PHM is relatively constant. The log-rank of Cox PHM shows a better result than SURLS-SVM when the censoring percentage is less than 0.3, and it decreases as the censoring percentage increases. The SURLS-SVM has a better result than Cox PHM in data with a censoring level greater than 0.3. For data with a high censoring rate, SURLS-SVM is more appropriate than Cox PHM. The right side of Fig. 5.3 displays the performance measures of data with a linear pattern using a sample size of 1000. It presents similar information as depicted by the plot for 100 samples on the left side.

Figure 5.4 displays the performance index of data with the non-linear pattern on a different censoring percentage using six predictor variables. On the left side (sample size of 100), the C-index of SURLS-SVM is much higher than that of Cox PHM for all censoring levels. It increases as the censoring percentage increases and only slightly decreases when the censoring percentage is equal to 0.5. The log-rank test of Cox PHM is much better when it is applied to a low censoring portion, and SURLS-SVM outperforms Cox PHM at a high censoring level. When the censoring levels are 0.6 and 0.7, the log-rank of Cox PHM and SURLS-SVM is slightly different.

The log-rank of Cox PHM decreases as the censoring percentage increases, in contrast to SURLS-SVM, except that it decreases at a censoring level of 0.9. The hazard ratio of SURLS-SVM is higher than that of Cox PHM at 0.1 up to a censoring level of 0.8 and slightly different from Cox PHM when the censoring percentages are 0.8 and 0.9. The hazard ratio of SURLS-SVM decreases as the censoring level increases, whereas the hazard ratio of Cox PHM is relatively constant. The right side of Fig. 5.4 (a sample size of 1000, a non-linear pattern, six features) displays similar information to that reported from the 100 samples visualized on the left side.

5.4.3 Effect of Sample Size

The effects of the features dimension and sample size, in addition to the censoring percentage, are analyzed in subsections 5.4.1 and 5.4.2 respectively. This part focuses only on the effect of sample size on the performance of the SURLS-SVM. The simulation study was conducted based on three different sample sizes.

Performance measures based on sample size are displayed in Fig. 5.5. The generated data use a censored percentage of 0.5 with 17 features. The C-index of SURLS-SVM outperforms Cox PHM on data with the linear and non-linear pattern. The difference in C-index between the two models is significant. The sample size increases, but the C-index is almost constant, except for Cox PHM on data with the linear approach, when the sampling size increases as the C-index increases. From the plot in Fig. 5.5b, when Cox PHM is applied to data with a linear pattern, the hazard ratio of Cox PHM outperforms that of SURLS-SVM, but when Cox PHM is

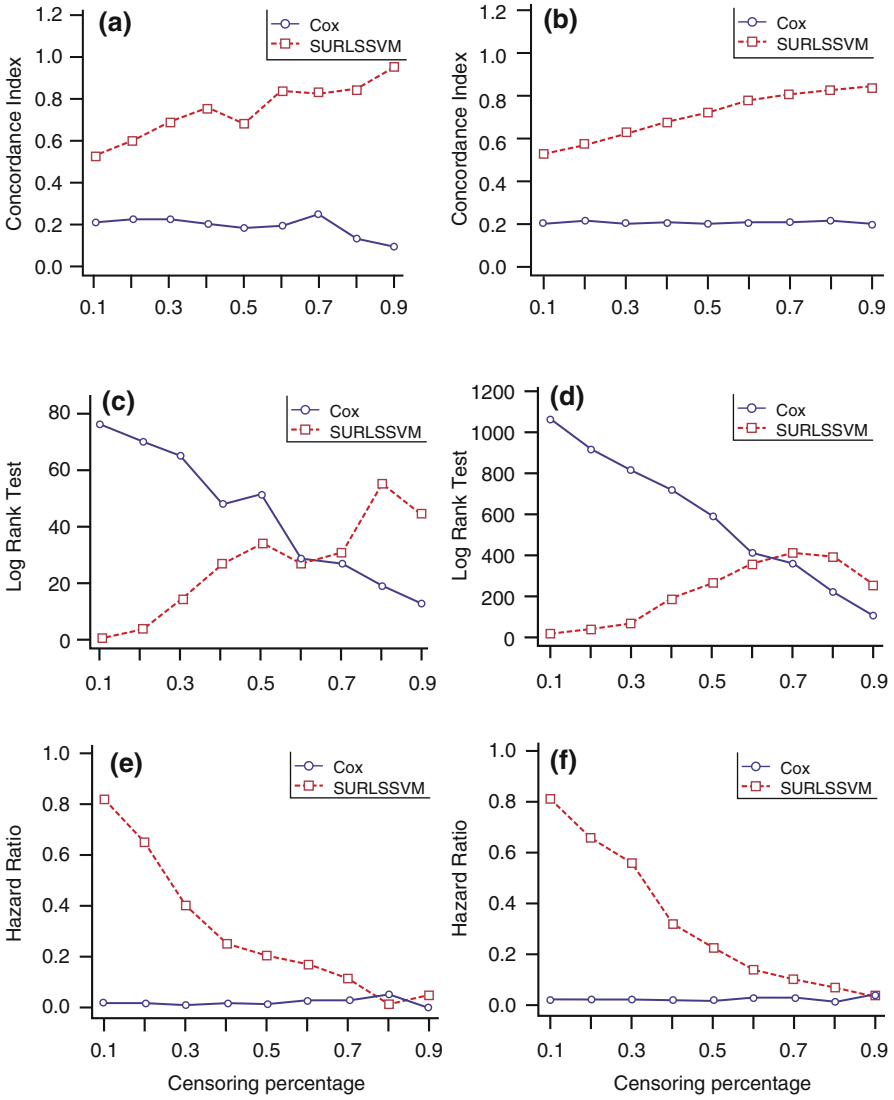


Fig. 5.4 Performance of Cox PHM and SURLS-SVM on data with a non-linear pattern using different censoring percentages for a sample size of 100 (**a**, **c**, **e**) and for a sample size of 1000 (**b**, **d**, **f**)

used on data with the non-linear pattern, the hazard ratio of Cox PHM is less than that of SURLS-SVM.

The hazard ratio of SURLS-SVM is not much different when applied to small data with the linear and non-linear patterns. In Fig. 5.5c, a log-rank test of SURLS-SVM in the two data patterns are slightly different. The Cox PHM outperforms SURLS-SVM when it is applied to data with the non-linear pattern, but for data

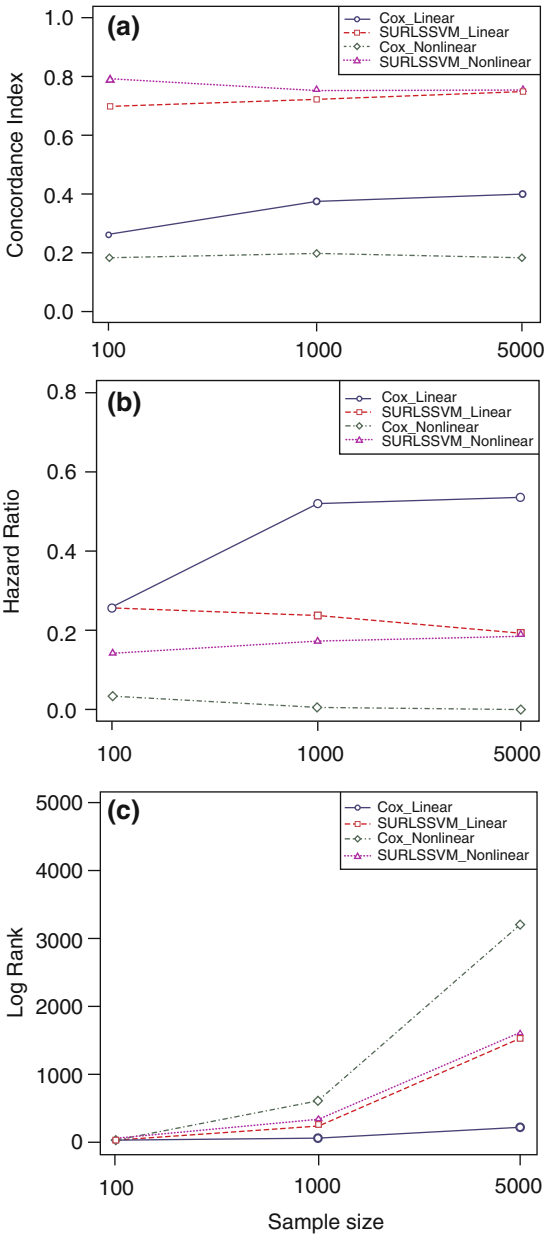


Fig. 5.5 (**a**–**c**) Performance Performance of Cox PHM and SURLS-SVM on data with a linear and non-linear pattern using different sample sizes

with a linear pattern, the value of the log-rank test of Cox PHM is less than that of SURLS-SVM.

In this simulation study, the performance of the two methods is analyzed based on a different number of features or predictors, various censoring percentages, and

different sample sizes. Each performance measure yields a different result on a different number of features. The hazard ratio and log-rank test of SURLS-SVM are not always better than those of Cox PHM. The SURLS-SVM indicates a better log-rank test when it is applied to data with a non-linear pattern, whereas Cox PHM yields a better hazard ratio from data with a linear pattern. The SURLS-SVM is always better than Cox PHM with regard to the C-index when it is applied to data with both the linear and the non-linear pattern.

There are a few papers that explain comparing the performance measures Cox PHM and SURLS-SVM. Smola and Scholk$\ddot{o}$pf and Van Belle et al. [5, 7] compared Cox PHM and some survival methods, one of them SURLS-SVM. They used breast cancer data as an experimental dataset, and it obtained that SURLS-SVM does not always display better results than Cox PHM. In the current study, based on a percentage of censoring level, the SURLS-SVM appears much a better result only with regard to the C-index.

The difference in the C-index between Cox PHM and SURLS-SVM on data with a linear pattern is less than on data with a non-linear pattern (Fig. 5.5b). This fact indicates that if the goal is to know the ranking of the prognostic index on data with a non-linear pattern, then SURLS-SVM produces a more accurate ranking than Cox PHM.

The better log-rank test of SURLS-SVM is obtained from high censoring percentage data with the linear and non-linear pattern. Although the better hazard ratio of SURLS-SVM is obtained from a low censoring percentage on data with a linear pattern, SURLS-SVM is applied to data with a non-linear pattern and displays a better hazard ratio for all different censoring rates, except for 0.9. Sample size does not significantly influence the performance of SURLS-SVM.

In the current study, the SURLS-SVM and Cox PHM yield different results depending on censoring percentage based on the value of the log-rank test and hazard ratio. It indicates that they are not proper performance measures. From Figs. 5.1 through 5.4, although SURLS-SVM outperforms Cox PHM or vice versa, both models have a significant log-rank test value. Thus, SURLS-SVM and Cox PHM can divide patients into high- and low-risk. SURLS-SVM is not always better than Cox PHM because it divides groups into high and low risk based only on the mean or median of the prognostic index. If the prognostic index of the object is larger than the mean, the object is classed as high risk and labeled 1, otherwise it is designated low-risk and marked 0. Labeling 0 and 1 do not take into account the size of the difference between the prognostic index of the object using the mean or median; thus, a small or large difference between the prognostic index and the mean or median is assumed to be the same value.

5.4.4 Discussion of the Results of the Simulation

Mahjub et al. [2] employed machine learning with a regression approach to analyze survival data from five experiments and artificial data. They reported that in some

experimental data sets, standard SVR (SSVR) has a slightly better performance than Cox PHM in all performance measures, but SSVR does not always have better results than Cox PHM in different censoring percentages. This study finds that the C-index of SURLS-SVM is still better than that of Cox PHM in all different censoring percentages; thus, the C-index is the best performance measure for comparing SURLS-SVM and Cox PHM because the results are consistent. The SURLS-SVM use a ranking approach and C-index can explain precisely the relation between survival time (or failure time) and prognostic index prediction instead of survival time prediction. The C-index is consistent because the prognostic index is not labeled by mean or median, but the labeling is based on concordance between survival time and prognostic index by considering censored status. Based on Figs. 5.3a, b and 5.4a, b, the difference in C-index between SURLS-SVM and Cox PHM is high in data with a non-linear pattern. However, when the SURLS-SVM and Cox PHM are applied to data with a linear pattern, the difference in C-index decreases. The difference in C-index increases as the censoring percentage increases. In Figs. 5.1a, b and 5.2a, b, the difference in C-index does not decrease significantly. Therefore, some predictors do not significantly affect the C-index.

This study finds that a better model cannot be explained only from log-rank and hazard ratio, but it can be described precisely by C-index. The sample sizes do not matter, but when applied to the data with a sample size of 1000, all performance measures are more stable. Various censoring percentages of data with sample sizes of 100 and 1000 produce the same pattern plots. Based on sample size, SURLS-SVM always outperforms Cox PHM based on C-index, but Cox PHM obtains better values of hazard ratio and log-rank. The results of SURLS-SVM obtained from data with the linear and non-linear patterns are slightly different, but Cox PHM produces many different results when it is applied to data with the linear and non-linear trends. Therefore, SURLS-SVM yields more consistent performance than Cox PHM.

Khotimah et al. [10] applied additive SURLS-SVM to simulated data along with the real application to predict cervical cancer status. They also applied feature selection using a backward elimination technique based on C-index increments. The simulation study showed that additive SURLS-SVM performs better when the censoring percentage is small. The recent feature selection techniques, such as Lasso and Elastic-net as applied by Haerdle et al. [12], in addition to SCAD and Elastic-SCAD, can be a potential technique to be employed to improve the work of Shieh [16]. Future research work can also consider time-changing features of the object observed until the event of interest happened—the work of Prastyo et al. [17] employed Bayesian multiple-period logit to analyze survival data with time-varying features. As logit and SVM are two main methods in classification, it is possible to extend the survival SVM to multiple-period survival SVM.

In Prastyo et al. [14] it was shown that the C-index of SURLS-SVM is much higher than that of Cox PHM; hence, the SURLS-SVM is better than Cox PHM at all censoring percentages. The lowest C-index is obtained when the data contain 10% of censored data. High censoring percentage data mean that there are many events happening; hence, the probability of an increase in mis-ranking. The C-index is produced by the prognostic index; hence, the prognostic index of SURLS-SVM

is obtained by parameter optimization, whereas the Cox PHM only uses an estimate of the parameter, which certainly contains an error when it is predicted.

Backward elimination is applied to the same data to obtain features that give a significant effect of increasing the C-index. The increase in the C-index is achieved when the feature selection is employed. To see how the SURLS-SVM and backward elimination can detect this condition, replication was simulated. It was replicated 100 times on a censoring percentage of 10% because this has the lowest C-index when all predictors are included in the model. The measure that was used to consider the eliminating predictor is the C-index.

5.4.5 Application to Health Data

The full description of the application to health data (CC) can be seen in Prastyo et al. [14]. The two predictors of CC, i.e., stage and level of education, are merged because they do not have sufficient data based on cross-tabulation for each predictor. The stage does not satisfy the PH assumption. This condition indicates that Cox PHM is not appropriate for analyzing these data; hence, the analysis requires another model, i.e., SURLS-SVM. Furthermore, the results of estimation parameters and testing significantly for each predictor yield three significant predictors, i.e., type of treatments (chemotherapy and transfusion), stage, and level of education (junior high school).

The C-index increases after backward elimination is applied. The first predictor to be eliminated is anemia status (P_3), followed by family planning status (P_{11}), and the age at first menstrual period (P_7). The C-index for each predictor after backward elimination is 97.09%, 97.14%, and 97.17% respectively.

The predictor that gives the highest effect is age (P_1) because it has the highest difference. Therefore, the order of predictors based on effected C-index are age (P_1), the age at marriage (P_6), stage (P_5), menstrual cycle (P_8), type of treatment (P_4), parity (P_{10}), and the last is features that do not have a decreasing C-index, i.e., complication status (P_2), length of menstruation (P_9), and level of education (P_{12}).

This study uses C-index instead of the significant level; hence, the predictors that give the smallest increase in C-index are eliminated. On the other hand, the feature selection can delete redundant predictors from the final model, and then the C-index of the final model can be increased. To validate how feature selection works on SURLS-SVM, this paper use replication (on simulated data) with the same scenario; furthermore, the result of replication shows that the irrelevant predictors (predicting which has a zero coefficient) are often included in the model. This is caused by main-confounder and sub-main confounder features, where they have an interaction that generates survival time, but they are not included in the analysis.

This work can be expanded by considering interaction of features in the analysis and by working on a more advanced method of feature selection, for example, feature selection with a regularization approach [18] as part of an embedded approach, which has a more straightforward step. Model-based feature selection [19],

i.e. feature selection based on the information from parametric model, may also be considered to give more intuitive reasoning. Even the proposed method in this paper belongs to the non-parametric approach, the information from parametric approach typically can improve the performance of the proposed method.

5.5 Conclusion

The simulation study shows evidence that SURLS-SVM outperforms Cox PHM based on the C-index criterion in any number of features. This result does not always happen when other performance measures, log-rank or hazard ratio, are used. However, the SURLS-SVM is better than Cox PHM in most of the scenario combinations, in particular, in the censoring percentage, the SURLS-SVM outperforms Cox PHM at all censored level data based on the C-index criterion. The feature selection of SURLS-SVM contributes to performance improvement. Also, the replication procedure informs that the irrelevant predictors can be selected as relevant features in SURLS-SVM because of the confounding effect. In application to the cervical cancer dataset, the significant features in Cox PHM are also the features that improve the C-index of SURLS-SVM once the backward elimination has been applied.

Acknowledgement Authors thank the reviewers for their advice. This research (2017–2019) is supported by PDUPT research scheme financed by DRPM DIKTI (through ITS), Indonesian Ministry of Research, Technology and Higher Education. The second author is grateful for LPDP scholarship.

References

1. Kleinbaum, D. G., & Klein, M. (2012). *Survival analysis: A self-learning text* (3rd ed.). London: Springer.
2. Mahjub, H., Faradmal, J., Goli, S., & Soltanian, A. R. (2016). Performance evaluation of support vector regression models for survival analysis: A simulation study. *IJACSA, 7*(6), 381–389.
3. Van Belle, V., Pelckmans, K., Suykens, J. A., & Van Huffel, S. (2007). Support vector machines for survival analysis. In *Proceedings of the third international conference on computational intelligence in medicine and healthcare (CIMED)*, Plymouth.
4. Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on computational learning theory* (pp. 144–152). Pittsburgh: ACM.
5. Smola, A. J., & Scholköpf, B. (2004). A tutorial on support vector regression, statistics, and computing. *Statistics and Computing, 14*(3), 192–222.
6. Van Belle, V., Pelckmans, K., Suykens, J. A., & Van Huffel, S. (2010). Additive survival least-squares support vector machines. *Statistics in Medicine, 29*(2), 296–308.
7. Van Belle, V., Pelckmans, K., Suykens, J. A., & Van Huffel, S. (2011). Support vector methods for survival analysis: A comparison between ranking and regression approaches. *Artificial Intelligence in Medicine, 53*(2), 107–118.

8. Suykens, J. A., & Vandewalle, J. (1999). Least squares support vector machines classifiers. *Neural Processing Letters, 9*(3), 293–300.
9. Goli, S., Mahjub, H., & Faradmal, J. (2016). Survival prediction and feature selection in patients with breast cancer using support vector regression. *Computational and Mathematical Methods in Medicine, 2016*, 1–12.
10. Khotimah, C., Purnami, S. W., Prastyo, D. D., Chosuvivatwong, V., & Spriplung, H. (2017). Additive survival least square support vector machines: A simulation study and its application to cervical cancer prediction. In *Proceedings of the 13th IMT-GT international conference on mathematics, statistics and their applications (ICMSA), AIP conference proceedings 1905 (050024)*, Kedah.
11. Khotimah, C., Purnami, S. W., & Prastyo, D. D. (2018). Additive survival least square support vector machines and feature selection on health data in Indonesia. In *Proceedings of the international conference on information and communications technology (ICOIACT)*. IEEE Xplore.
12. Haerdle, W. K., Prastyo, D. D., & Hafner, C. M. (2014). Support vector machines with evolutionary model selection for default prediction. In J. Racine, L. Su, & A. Ullah (Eds.), *The Oxford handbook of applied nonparametric and semiparametric econometrics and statistics* (pp. 346–373). New York: Oxford University Press.
13. Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers and Electrical Engineering, 4*(1), 16–28.
14. Prastyo, D. D., Khoiri, H. A., Purnami, S. W., Suhartono, & Fam, S. F. (2019). Simulation study of feature selection on survival least square support vector machines with application to health data. In B. W. Yap et al. (Eds.), *Soft computing and data science 2018 (SCDS2018)* (Communications in computer and information science) (Vol. 937, pp. 1–12). Singapore: Springer.
15. Bender, R., Augustin, T., & Blettner, M. (2005). Generating survival times to simulate Cox proportional hazards models. *Statistics in Medicine, 24*(11), 1713–1723.
16. Shieh, G. (2006). Suppression situations in multiple linear regression. *Educational and Psychological Measurement, 66*(3), 435–447.
17. Prastyo, D. D., Miranti, T., & Iriawan, N. (2017). Survival analysis of companies' delisting time in Indonesian stock exchange using Bayesian multiple-period logit approach. *Malaysian Journal of Fundamental and Applied Sciences, 13*(4-1), 425–429.
18. Haerdle, W. K., & Prastyo, D. D. (2014). Embedded predictor selection for default risk calculation: A southeast Asian industry study. In D. L. K. Chuen & G. N. Gregoriou (Eds.), *Handbook of Asian finance: Financial market and sovereign wealth fund* (Vol. 1, pp. 131–148). San Diego: Academic Press.
19. Suhartono, Saputri, P. D., Amalia, F. F., Prastyo, D. D., & Ulama, B. S. S. (2017). Model selection in feedforward neural networks for forecasting inflow and outflow in Indonesia. In A. Mohamed, M. Berry, & B. Yap (Eds.), *Soft computing and data science 2017 (SCDS2017)* (Communications in computer and information science) (Vol. 788, pp. 95–105). Singapore: Springer.

Chapter 6
Semantic Unsupervised Learning for Word Sense Disambiguation

Dian I. Martin, Michael W. Berry, and John C. Martin

6.1 Introduction

A fundamental characteristic of language is lexical ambiguity. Words with more than one meaning are used commonly throughout text. The 121 most frequently used words in the English language have an average of 7.8 different meanings, or senses, associated with each of them, and these words appear approximately 20% of the time in text [1]. When these words are read in context, there is little ambiguity to a human reader. Determining the senses of a word being used in a given passage of text, a process known as word sense disambiguation, is rather simple task for a human reader, but a difficult one to accomplish using automated, algorithmic systems [1–4].

6.1.1 Word Sense Disambiguation

For any given word, there exists one or more senses, where a word sense is the particular meaning associated with that word in a given context of usage. The complete definition of a word encompasses all the possible meanings or senses of a word, but generally, only one sense is intended when a word is used. Word

D. I. Martin (✉) · J. C. Martin
Small Bear Technologies, Inc., Thorn Hill, TN, USA
e-mail: Dian.Martin@SmallBearTechnologies.com; John.Martin@SmallBearTechnologies.com

M. W. Berry
Department of Electrical Engineering and Computer Science, University of Tennessee at Knoxville, Knoxville, TN, USA
e-mail: mberry@utk.edu

M. W. Berry et al. (eds.), *Supervised and Unsupervised Learning for Data Science*, Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-030-22475-2_6

senses can be considered either coarse-grain or fine-grain, depending on the level of detailed distinction that is required to divide them. Coarse-grain senses exhibit major, unrelated differences in meaning, where fine-grain senses have more subtle distinctions between them, and they are frequently interrelated. Words that possess only coarse-grain senses are called homographs, while words that have mainly fine-grained senses are polysemous words [5].

Often words possess both coarse- and fine-grain senses. The word *bank* is a classic example of a word having both types of senses. Examining course-grain senses, the word *bank* may be viewed as having two clear distinct senses as a noun with one meaning being "a slope beside a body of water," while another sense being "a financial institution." However, the word *bank* when taken in the sense of "a financial institution" can be divided into more subtle, fine-grained, distinctions such as "the physical building where financial transactions are performed," "the financial institution that accepts deposits and participates in different lending activities," "a reserve of money," or "a container for keeping money" [1]. Consider the textual phrase "*I am going to put my money in the bank*." There are several possible interpretations for the sense of the word *bank:* Is the money being put in a piggy bank? Is the money going to be deposited at a financial institution? Or is *bank* referencing a fund that is being put aside for future use or emergencies. Determining the level of granularity for the word sense depends on the application and context of the word being used.

Interestingly, human readers seem to be able to make sense distinctions innately and accurately, easily differentiating between multiple senses of an ambiguous word when given sufficient context. Of course, individual humans also have differences in their ability to perform this disambiguation task based on their cognitive model, exposure to spoken and written language, and domain-specific vocabulary. Even for human experts, lexicographers, determining the number of senses of a word and giving those senses a definition is a challenging and subjective task [6].

6.1.2 History and Approaches

Formal WSD research began with the work in computational linguistics and artificial intelligence (AI) in the 1940s and early 1950s. In his 1950 paper "Computing Machinery and Intelligence," Alan Turing described the primary indicator of the existence of intelligence as possessing the ability to understand language [7]. Language understanding requires the ability to disambiguate the meaning of words [8]. Research addressing the task of WSD originated in the field of Natural Language Processing (NLP) [5], computational linguistics [9], and in machine translation. Often, there is one word in a given language that has the potential to translate into multiple words in another language [5].

The WSD problem involves two major tasks: defining the possible sense definitions for a word (*sense discovery*) and then identifying which sense of a word

is being used for a given context (*sense identification*). Original work in WSD was performed manually by human lexicographers. Initial attempts to automate WSD began to appear in the 1970s. Over time, four major categories of WSD approaches have developed: dictionary- or knowledge-based methods, supervised methods, minimally supervised methods, and unsupervised methods also called word sense discrimination [2, 4, 6, 9–12]. Dictionary- or knowledge-based methods rely on existing dictionaries or some a priori lexical knowledge base, such as WordNet [13], that has been built up manually to inform their processing decisions for the identification task. Supervised methods for WSD use sense-annotated corpora produced by human lexicographers for training an automated system, which is then used for the identification task. Semisupervised or minimally supervised methods involve building a disambiguation model based on small amount of human annotated text or word-aligned bilingual corpora and then bootstrapping from this seed data to build additional sense indicators, which are then used to identify a sense used in a given context. Unsupervised methods for WSD use no external information and work directly with raw nonannotated corpora to induce the senses for words.

Currently, the best results for automated WSD are achieved using supervised methods [4, 12, 14], but these methods require extensive sense-tagging training data to be available beforehand. This sense-tagging data requirement poses a problem both in time and expense, along with the lack of flexibility in dealing with ever-changing language. Knowledge-based, supervised, and semisupervised methods all require significant a priori knowledge, and the systems based upon these methods must be custom-built by humans for a specific target language. Also, these systems are limited in the number of terms for which they can distinguish senses, which poses problems when using larger amounts of text, new domains, or new languages, and yet, recent findings suggest that WSD algorithms work best on large volumes of data [2]. Unfortunately, unsupervised systems for WSD still lag in their accuracy when compared to other types of systems [15–17].

6.2 Latent Semantic Analysis

One approach to WSD is to utilize Latent Semantic Analysis (LSA) to construct an underlying learning system or AI system that is then employed for the sense discovery and sense identification tasks. The learning system provides a basis for clustering words based on contexts given in a corpora or textual document collection and provides an automated platform for inducing and distinguishing word sense in a given text corpus. The process also provides additional insight into the learning system being used that can be leveraged to fine-tune or fit the learning system for other uses.

The LSA-based learning system, as illustrated in Fig. 6.1, processes input text, which is used to train the system to learn language using a mathematical technique, producing the cognitive model represented by the LSA semantic space. Use of

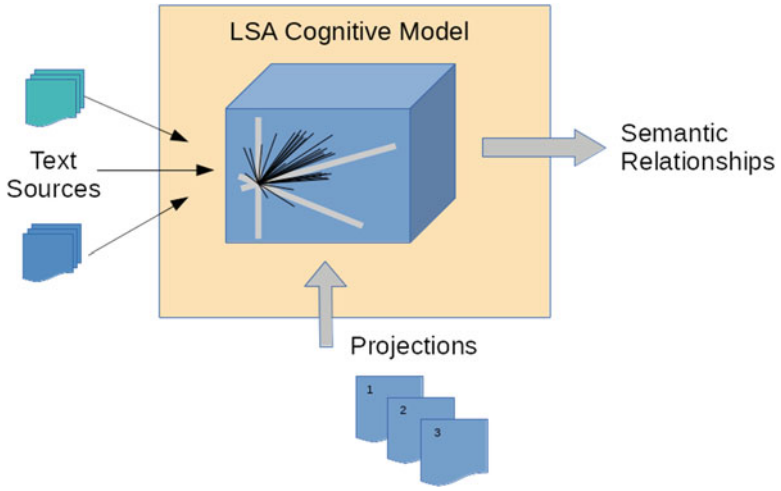


Fig. 6.1 Visual representation of the LSA-based learning system [3]

the system is facilitated by functions to interrogate or probe the cognitive model. This cognitive model represents the underlying semantic relationships between words and is an analog to the human understanding of language [18–20]. Many applications have shown that the performance of LSA-based learning systems on certain cognitive tasks has simulated human knowledge as well as the understanding of words and meanings of text [20–26]. The cognitive model is produced as a high-dimensional vector space representing the semantic relationships between items of the input text. The LSA cognitive model exploits patterns of word usage across a corpus to learn the meaning of words relative to each other within that body of text [20, 24, 25, 27]. Research in the field of LSA has reported that, given a context in which an ambiguous word is used, LSA can determine its sense for information comparison purposes [23, 28, 29]. For more detailed explanation of the mathematical derivation of the LSA-based learning system and the cognitive model, see Martin et al. [20], Martin and Berry [18], and Martin [3].

6.3 LSA-WSD Approach

The use of an unsupervised LSA-based learning system for WSD (LSA-WSD) has some promising advantages in that the supporting technology for this approach is fully automated, usable for any word, applicable to other languages, and may be updated easily to account for changes in language usage. Also, the LSA-WSD process can be used to characterize the knowledge represented in the LSA cognitive model being employed. The LSA-WSD system makes use of the cognitive model

from the LSA-based learning system along with a technique called Semantic Mean Clustering (SMC) for inducing word senses and a Context Comparison (CC) method for disambiguating individual words in context.

6.3.1 Sense Discovery

Given an LSA cognitive model, also known as the LSA semantic space, the initial task in LSA-WSD approach is sense discovery: determining the senses associated with a given word within the model. This is necessary as not all possible word senses will be represented in a specific model. The ideal input corpus would provide a general linguistic framework that supplied all the typical word senses that might be encountered, but this corpus could be augmented or targeted for a particular domain or may underrepresent the word knowledge of a typical user of that language. Sense discovery results are only as good as the learning system upon which the induction is obtained, because the ability to distinguish senses depends on the knowledge contained within the learning system.

The LSA-WSD system uses a modified clustering technique (SMC) that exploits the properties of the LSA cognitive model for relating semantically similar items to perform sense discovery. It is based on the concept that each term vector in an LSA semantic space carries all the possible meanings or senses for a term [23]. The challenge for sense discovery is then to separate these senses into individually identified senses. Examining the top s synonym words related to a given target word, the SMC technique is used to produce synonym clusters (synclusters), which represent possible senses for the target word. This sense discovery method can be applied to any word, and the method can also be extended to different domains and languages while dynamically adapting to the content being analyzed.

6.3.2 Sense Identification

After senses have been induced, the second task of the LSA-WSD approach is to distinguish the sense in which a word is being used within a particular input context. This is done using the Context Comparison method. This method processes an input context sentence, removes the target word, and maps the result into the LSA semantic space to identify the closest syncluster, which represents one of the previously discovered senses for the word. The closest cluster would serve to identify the sense of the target word as used in this context, disambiguating the sense of the word in question.

6.3.3 Semantic Mean Clustering

For the LSA-WSD approach, a clustering model was developed that integrates aspects of connectivity-based clustering and centroid-based clustering called Semantic Mean Clustering (SMC) [3]. The SMC technique leverages the fact that cosine measurements in the LSA cognitive model represent a similarity of meaning and have certain thresholds and bounds that can be interpreted with some useful implications. A centroid model for representing the clusters was needed to generalize the meaning mapping of the terms. For this synclustering application, it is not necessarily important that the individual items have a close relationship with each other, but that they have a relationship to certain meanings as manifested in the centroid. Additionally, the actual meanings of the terms that are found should drive the number of clusters identified. This requires that cluster membership for specific items has the ability to change as the cluster definition encompassed in the centroid evolves for each term that is processed. SMC is similar to k-means clustering in that it is based on centroids, but different in that it is not initialized with a random grouping of item clusters. The clustering outcome is reproducible, and there is no predefined value for k. The number of clusters is determined by the associations captured in the cognitive model. Like hierarchical clustering, SMC produces reproducible results and allows the data to drive the decision of how many clusters there are, but it differs in that items are assigned to a cluster by their similarity to cluster centroids and not their measurement to other items.

The SMC algorithm takes one input parameter, the cluster inclusion threshold, which defines the minimum similarity measurement required when comparing an item to the cluster centroid. Due to the constant recalculation of the cluster centroid, the cluster assignment for an item is not permanent until the entire process has been completed. During processing, as new items get added and the centroid for the cluster is recomputed, any item can fall outside of the similarity threshold for its cluster; therefore, multiple passes are required to reprocess any items that fall out of their initial cluster assignments. This reprocessing of fallout items is performed until all items have been assigned membership in some cluster. The complexity of the clustering used by the SMC algorithm is in the worst-case scenario not greater than $O(kn)$, where $k < n$, but in practice, only one fallout reprocessing pass has been observed with a small number of fallout items to be occasionally necessary. In comparison, the complexity for hierarchical clustering is generally $O(n^2)$, with the worst case being $O(n^3)$, and the complexity for k-means clustering is $O(n^2)$.

The SMC algorithm using the LSA cognitive model for synclustering is defined as follows [3]:

For each synonym in the top k terms closest to the given target word:

1. Project the term in the LSA semantic space.
2. Compare the term mapping to all identified cluster trajectories.

   (a) If this is the first item, it will be recorded as the first cluster trajectory (see Fig. 6.2, left).

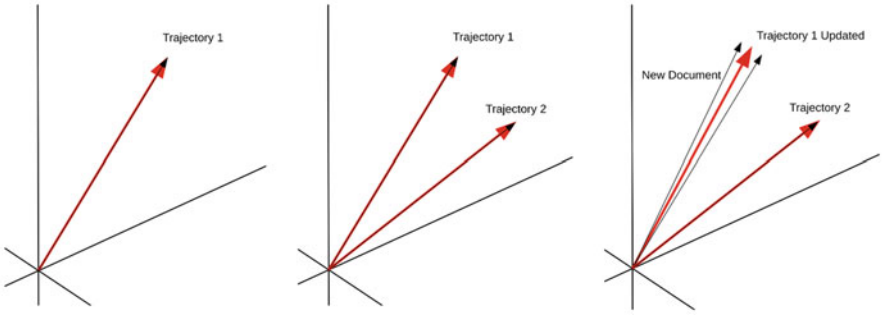


Fig. 6.2 Steps in SMC cluster construction [3]

(b) If the item mapping is not close to, outside of the similarity threshold of, any previously identified cluster trajectories, then the item mapping is recorded as a new cluster trajectory (see Fig. 6.2, middle).
(c) If the item mapping is close, within a certain similarity threshold, to an existing cluster trajectory, then it is grouped with items associated with the nearest cluster trajectory and the cluster trajectory (centroid) is updated by averaging all the member item mappings (see Fig. 6.2, right).

After the full set of items is processed, each item is then examined to determine if any of them fall outside of the similarity threshold.

3. If an item falls outside the similarity threshold for the cluster trajectory of which it is a member:

(a) The item is removed from the cluster group membership and the cluster trajectory is recalculated.
(b) The item is cataloged for reprocessing.

4. If an item is within the similarity threshold for the cluster trajectory of which it is a member, no action is necessary.

Repeat steps 1 and 2 with the items cataloged for reprocessing and then repeat steps 3 and 4 until no fallout items are identified.

The SMC process automatically determines the number and formation of the clusters where each cluster is a group of items, in this case the top terms closest to the given target word within the cognitive model, falling within a set similarity threshold of the centroid and representing a consolidated semantic trajectory within the LSA cognitive model as depicted in Fig. 6.3. Each cluster represents a sense identified within the LSA-based learning system for the target word.

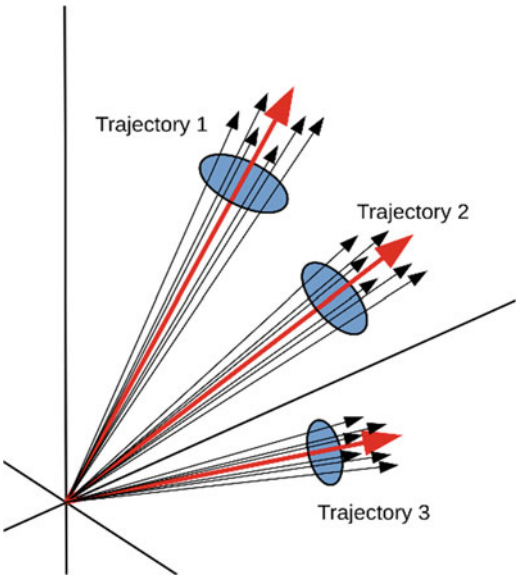


Fig. 6.3 Clustered trajectories identified by SMC [3]

6.4 Sense Discovery Using Synclustering

Synclustering is predicated on the idea of attempting to separate the senses for a target word embedded within its term vector by clustering the synonyms of that word based upon their similarity measurement with each other. In the cognitive model, synonyms are words that are close in their mapping, indicating a semantic relationship. These words may or may not be linguistic synonyms that are mutually replaceable. These *synonyms* indicate what associations of meaning exist for the target word within the cognitive model.

To perform synclustering, all the words present in the LSA-based learning system are examined to find the words closest to the target word using the cosine measurement for similarity. These top k synonym words are then clustered using the SMC algorithm to produce candidate word sense clusters (WSCs) for the target word. Once the candidate WSCs are induced, the closest word to the centroid of each of these clusters is taken to be the identifier, or descriptor, for the sense the cluster represents.

6.4.1 Experimentation Parameters

For these experiments, two document collections were used as input into the learning system, creating two different cognitive models. One document collection consisted of 3.5 million paragraph-sized text, derived from educational materials

Table 6.1 Target words used in experiments for word sense discovery and identification [3]

Words 1–6	Words 7–12	Words 13–18
Bank	Interest	Pretty
Batch	Keep	Raise
Build	Line	Sentence
Capital	Masterpiece	Serve
Enjoy	Monkey	Turkey
Hard	Palm	Work

(books, periodicals, etc.). Each text was previously noted with an associated Lexile level, which is an estimate of the reading difficulty level for the text ranging from elementary to adult [30, 31]. They are representative of texts encountered by a typical learner of language over time and consist of reading material often used in American schools [30, 32]. The other collection used was the Reuters Text Retrieval Corpus (RTRC) RCV1 collection [33] consisting of over 800,000 English language news articles, published by Reuters during the 1996–1997 timeframe.

A LSA semantic space, or cognitive model, was constructed from an approximately 200,000 document set for each of these corpora. The subset of documents from the grade level reading corpus included items ranging from elementary to adult difficulty level with each level being equally represented or chosen. The subset of documents from the news collection corpus contained varied content deemed to be typical for adult reading level. These document sets were selected to serve as a basis for linguistic meaning and thus allowed for an adequate base LSA semantic space, a sufficiently large representation of a typical adult cognitive model [20, 34].

Once the two cognitive models were created, additional parameters for synclustering were set. The number of synonyms related to the target word to use in synclustering was varied to test the clustering with 100 and 200 words. Synclustering was performed on the 18 target words, shown in Table 6.1. This set consists of words with coarse- and fine-grained senses and includes mostly nouns and verbs with a few adjectives and adverbs. Two cluster inclusion thresholds were tested, 0.3 and 0.4. These thresholds yielded good results, and testing of other threshold values was left for future work.

6.4.2 Observations and Results

Using the grade-level learning system, the top 100 closest words, or synonyms, to the target word *bank*, are shown in Table 6.2 prior to clustering in the order of their proximity to *bank*. Casing is not considered so that all words are processed in lowercase, including proper nouns.

Simply inspecting the words manually, it is interesting to note that the top 36 words are associated in some way with a riverbank or a body of water. The first term that is not clearly in that sense category is the word *deposit*. This word is ambiguous, because it can refer to a deposit on the bank of a river or to a money

Table 6.2 The top 100 closest words to the word *bank* in the grade-level learning system [3]

Terms 1–20	Terms 21–40	Terms 41–60	Terms 61–80	Terms 81–100
Bank	Levee	Riverbed	Riffles	Waterfall
Banks	Gorge	Barges	Snags	Waded
Downstream	Flatboat	Paddled	Money	Overhanging
Riverbank	Bend	Tributaries	Shallows	Crossing
Upstream	Boatmen	Thames	Creek	Sandbars
River	Canoe	Midstream	Conononka	Portage
Rapids	Steamboat	Canal	Savings	Bills
Downriver	Footbridge	Countercurrents	Flowing	Swift
Dam	Flood	Monongahela	Bottomland	Sawmills
Upriver	Ferrymen	Paddle	Creeks	Paddling
Bridge	Dammed	Reeds	Watercourse	Mississippi
Flowed	Bottomlands	Cash	Poled	Damming
Current	Sandbar	Ferryman	Wading	Meander
Raft	Flatboats	Boatman	Riverside	Murky
Tributary	Robb	Riverbanks	Narmada	Platte
Barge	Stream	Dams	Rhadamnanthus	Riverboat
Steamboats	Deposit	Rafts	Cocytus	Uminpeachable
Muddy	Loan	Headwaters	Radarscope	Potomac
Eddies	Willows	Silt	Insecttortured	Marshy
Bluffs	Nashua	Poling	Shallow	Spanned

deposit. Some of the term associations, reported in the list, are not immediately obvious; however, the words such as *Nashua*, *Thames*, *Monongahela*, *Conononka*, etc. are actually proper names of rivers. Those words should be associated with the river sense of *bank*. The only other terms in the list that would not be associated in some way with a riverbank or river are *loan* (ranked 38th), *cash* (ranked 52nd), *money* (ranked 63rd), *savings* (ranked 67th), and *bills* (ranked 87th). These terms suggest an association of money or money-related items with the word *bank*. From manual inspection of the top 100 synonyms to bank, there are two definite senses for the word *bank* that emerge, suggesting that synclustering might be expected to induce two senses from this list.

Using the grade-level learning system, the synclustering of these 100 words did yield two distinct clusters for the word bank (see Table 6.3). The two senses induced were a cluster for the "riverbank" sense of *bank* and one for the "money" sense. With cosine similarities of 0.78 and 0.51, the two clusters have centroids that are relatively close to the target word *bank*. A cosine of 1.0 would indicate that the mapping of the associated cluster is identical to the mapping for the target word. The cosine values associated with these candidate WSCs indicate that the cluster for the river sense is more closely associated with *bank* than the cluster for the money sense in this grade-level learning system.

In contrast, using the news learning system, synclustering produced somewhat different results for the word *bank*. First, the synonyms for *bank* in the news learning

Table 6.3 The WSCs discovered using synclusters on the top 100 synonyms for the word *bank* in the grade-level learning system [3]

Word sense cluster	Number in WSC	WSC descriptor	Next closest words	Cosine between *bank* and WSC centroid
WSC 1	93	Downstream	River Rapids Upstream Riverbank	0.78
WSC 2	6	Money	Bills Cash Savings Loan	0.51

Table 6.4 The WSCs using synclusters when clustering the top 100 synonyms for the word *bank* in the news learning system [3]

Word sense cluster	Number in WSC	WSC descriptor	Next closest words	Cosine between *bank* and WSC centroid
WSC 1	88	Banks	Banking Deposits Bankers Lending	0.78
WSC 2	9	Rates	Interest Reserve Mortgage Discount	0.36
WSC 3	1	Finance		0.21
WSC 4	1	Manages		0.21

system were different than the ones identified in the grade-level system, and using the same parameters with the WSI method, the top 100 synonyms to *bank* and a cluster inclusion threshold of 0.3, there were four candidate WSCs induced for the news learning system as described in Table 6.4. All four of them relate to the financial institution sense for the word *bank*. Only one, WSC 1, shows a strong association with the target word *bank*, having a cosine of 0.78 between the word *bank* and the centroid. The other three clusters, with cosine similarities of 0.36, 0.21, and 0.21, are more distant, suggesting that they may not represent usable senses for *bank*. The descriptors for each WSC suggests that there is really the only one primary sense for the *bank* that is understood by this learning system, though the second cluster could be taken to indicate a sense related to mortgages, which would be a fine-grained sense of the word.

Another target word that was examined was the word *palm*. Using the top 100 synonyms and a cluster inclusion threshold of 0.3, there were 12 candidate WSCs induced for *palm* using the grade-level learning system, while only 2 candidate WSCs were induced for the news learning space. Using the grade-level learning system, centroids for three of the candidate WSCs had a cosine similarity measure

Table 6.5 The WSC results for using synclusters on the word *palm* [3]

Grade-level learning system			News learning system		
WSC label	# in cluster	Cosine to centroid	WSC label	# in cluster	Cosine to centroid
Hand	29	0.65	Beach	3	0.55
Trees	40	0.50	Cigarette	96	0.47
Gripping	5	0.43			

with the target word *palm* that was greater than 0.4, while the remaining ones had cosine similarities of less than 0.31, suggesting that there were three senses associated with *palm* by the synclustering process, as shown in Table 6.5. Upon inspection, the induced senses correspond to the three coarse-grain senses of the dictionary definition of *palm*: (1) An unbranched, evergreen tree, (2) Inner surface of a hand between the wrist and the fingers, and (3) To hide or hold something in one's hand [35]. Using the news learning systems, the candidate WSCs induced two different senses for *palm*. One cluster appears to correspond to *Palm Beach* in Florida, which is not surprising as the cognitive model was constructed from news articles. News articles mention locations often. The other cluster has an interesting label of *cigarette*, indicating something to do with a hand; however, examining other words in the cluster are *tobacco*, *smokers*, *smoking*, *Lorillard* (name of a Tobacco company), and other names of people, suggesting an association of the meaning of *palm* with smoking.

For the target word *sentence*, synclustering produced only one candidate WSC for both learning systems. The grade-level learning system produced a WSC candidate with the descriptor of "spelling," while the news learning system produced a candidate WSC with the descriptor of "prison." The centroid cosine similarity to the target word *sentence* was high, greater than 0.96, for both cases. When the number of synonyms used for clustering was also expanded to the top 200, there was no change in the candidate WSC for either learning system. The grade-level learning system learned the definition of *sentence* as a group of words written together expressing a complete thought, and the news learning system learned the definition of *sentence* as a punishment assigned to a person in court.

Candidate WSCs induced for the word *line* produced 12 clusters using the grade-level learning system and 24 clusters with the news learning system. The news learning system did not find any clusters with a cosine similarity between the centroid and the target word of more than a 0.33, and after reviewing the top candidate, WSCs indicated no clear sense for *line* in the cognitive model constructed from the news input text. In contrast, the grade-level learning system induced some strong senses for *line*. Of the 12 candidate WSCs that were generated, 5 of them had a cosine similarity between their centroid and the target word that exceeded 0.43, as shown in Table 6.6. A manually produced description of the sense is shown that was derived by considering the other words associated with each WSC.

For all 18 of the target words examined, synclustering produced reasonable results, which reflected the word knowledge contained in the cognitive models

Table 6.6 The WSC results using synclusters on the top 200 words for the word *line* using the grade-level learning system [3]

WSC label	# in cluster	Cosine to centroid	Manual description of the sense
Zone	137	0.66	A line marked on a field or court that relates to the rules of a game or sport like a goal line or zone line
Assonance	6	0.63	A line of poetry
Bait	21	0.53	A line on a fishing rod
Horizontal	18	0.49	A mathematical term for a line or lines in particular directions
Ahead	7	0.44	A line marking the starting or finishing point in a race

that were used in the sense induction process. While not all the words resulted in clearly identifiable word senses, several such as *bank*, *palm*, *sentence*, and *line* showed promising results with clearly usable-induced senses for the subsequent task of word sense disambiguation. Empirical evidence suggests that candidate WSCs should have a cosine similarity between the centroid and the target word that exceeds 0.35 to be considered as a usable sense cluster for the word. As would-be expected coarse-grained senses appear to be more easily identified, however, some fine-grained senses were induced such as with the word *line* using the grade-level learning system. It was apparent in all cases that the two cognitive models were not equal in the represented word knowledge. While they showed agreement for several words in the senses induced, there were also some striking differences as can be observed with the words *sentence* and *line*. The grade-level learning system produced more broad-based and consistent results and was thus used in the subsequent sense identification task.

Testing different threshold parameters and selection quantities for the input synonyms led to the identification of a 0.3 clustering threshold and a selection of 100 synonyms as the best performing configuration. Further research is ongoing to refine these input criteria to optimize results. Synclustering for automatic word sense induction provides a means to induce senses and a way to examine and analyze the underlying LSA-based learning system.

6.5 Sense Identification Using the Context Comparison Method

The next task for the LSA-WSD approach is the identification of the sense of a target word in a context, also known as disambiguation. Sense identification amounts to the determination of the syncluster best representing the sense of the target word in the given sentence. The synclusters induced in the sense discovery task are each described by a centroid vector that maps the average of all the component term

vectors in the cluster and by a representative word that corresponds with the closest component vector to the cluster centroid.

Given a target word, the Context Comparison (CC) method takes the given context sentence minus the target word, projects this modified context sentence into the cognitive model, and then identifies the closest syncluster centroid using the cosine similarity measure. The reason the modified context is used is that the context words surrounding the target word in the sentence carry the information that suggests the sense of the target word within that usage. The algorithm for the CC method is as follows [3]:

1. Copy the input sentence *sentA* into *sentB*.
2. Remove the target word from *sentB*.
3. Project *sentB* into the LSA semantic space.
4. For each syncluster:
   (a) Compute the cosine similarity measure between the projection for *sentB* and the centroid vector for the syncluster.
   (b) Record the cosine similarity.
5. Determine the highest cosine similarity recorded and take the corresponding syncluster as the identified word sense.

6.5.1 Experimentation Parameters

The experiments for the sense identification task were conducted on the grade-level learning system. The sense discovery target words, and their corresponding word senses as determined by synclustering, were used in the disambiguation experiments to test the efficacy of the Context Comparison method. Results for the following target words were collected: *bank*, *palm*, *line*, *raise*, and *serve*. To evaluate WSD performance, test sentences for each of the target words were randomly selected from different sources including the grade-level corpus, online dictionaries, and WordNet [13]. Ten test sentences were then hand-annotated to identify the correct word sense used in each sentence. These ten were specifically selected to ensure coverage of all the different senses that had been previously induced by synclustering for each target word. Additional sentences containing different senses for the target word than those represented by the synclusters for each word, as well as ambiguous sentences, were added to the final test-set for each word. This sentence selection resulted in a test-set of 12 or more sentences for each of the target words with a human-generated correct word sense identification. Example test sentences for the words *line* are shown in Table 6.7.

Table 6.7 Test sentences used in the WSD task for the word *line* and their annotated sense determined by a human rater [3]

Annotated WSC label	Sentences using *line* in this sense
zone	Jackie stepped to the line and dropped in both foul shots. Jim plowed forward to stop the quarterback from reaching the goal line.
assonance	The pattern of stressed and unstressed syllables discernible in a line of poetry has been analyzed in order to determine whether the line follows an iambic or a dactylic or an anapestic metrical arrangement. Each stanza has eight lines.
bait	He reeled in the line and bent the pole. He cast out his line.
horizontal	The curved line represents the variation of voltage in the signal. Draw a horizontal line above the vertical line.
ahead	Matthew dashed across the finish line. I crossed the finish line, jogged to a stop, and kneeled on the cinders, breathing deeply.
Different Sense	The workers would build them on a moving assembly line.
Ambiguous Sense	Hold the line a minute, Diane.

6.5.2 *Observations and Results*

For each target word, each of the ten test sentences matching an induced sense was processed to identify a sense for the target word using the CC method. The results were compared to the human-generated sense identification to determine correctness. The results for five of the target words can be seen in Fig. 6.4.

For the target word *bank*, there were two induced senses resulting from syn-clustering (see Table 6.3). The CC method only incorrectly identified the sense for *bank* in one case by identifying the *downstream* sense for the sentence "It was a bank robbery in progress.." The correct sense of *bank* in the sentence was *money*. The computed cosine similarity was 0.14 for the *downstream* sense and 0.13 for the *money* sense. Both of these nearly equivalent cosines are small, suggesting a low degree of confidence in the result for the target word in this sentence. The additional test sentence contains a different sense (not corresponding to one of the induced senses): "Chuck was listening to the whistles and trills and adjusting dials on a bank of electronic equipment." This sentence makes use of the target word *bank* in the sense of a group or set of similar things. The CC method selected *downstream* as the sense, with a corresponding cosine similarity measurement of -0.02. With a cosine value near zero (indicating two vectors are unrelated), it can be interpreted that the CC method did not recognize the sense of the word *bank* in this sentence. This would be expected given that the operative sense for *bank* in this context had not been learned by the underlying learning system. The same sort of observation was noted for the ambiguous sentences, "That bank's not safe." and "Rosa had waved to her at the bank.." The calculated cosine values for each of the ambiguous sentences were

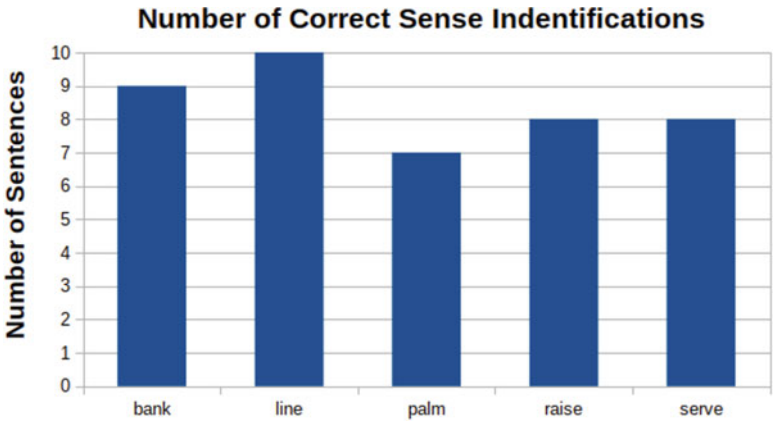


Fig. 6.4 The number of correctly identified word senses for the target words used in ten different context sentences [3]

0.05 and 0.04, respectively, suggesting that no sense could be strongly identified, indicating ambiguity that would require more contexts.

For the target word *line*, there were five senses induced by synclustering (see Table 6.6). The CC method was able to identify the correct sense for the ten test sentences containing *line* that had a matching-induced sense. For the additional sentences with an unlearned sense or ambiguous usage, the CC method produced cosine measurements near zero for the sentences, again suggesting low confidence in the sense identification produced for those contexts. However, interestingly, given the available learned senses to choose from, with the ambiguous sentence "Hold the line a minute, Diane.," the CC method identified the sense *bait* (fishing line), which from a human perspective is the most reasonable out of the given five induced senses: goal line, fishing line, finish line, poetry line, or mathematical line.

Examining the target word *palm* and its three induced senses, *hand*, *gripping*, and *trees*, the CC method identified 70% of the senses correctly, missing the correct identification for the sense of *palm* in the following three sentences:

1. Yellow sap oozed onto my palm.
2. She stroked my golden curls with a hand so large it seemed to palm my whole head.
3. I suspected that he had palmed a playing card.

The first sentence uses the word *palm* in the *hand* sense, but the CC method identified the sense as *trees* with a cosine value of 0.23 between the *trees* syncluster centroid and the original context sentence with the target word removed. While the word *sap* in the sentence suggests the *tree* sense, human readers recognize that the sentence uses *palm* in the context of a hand. For the other two sentences (2 and 3), it is clear that the *gripping* sense for the word *palm* was used. The CC method identified the sense for *palm* in sentence 2 as the *hand* sense with a cosine similarity

value of 0.51 and a cosine value of 0.23 for the sense of *gripping*, indicating the identification of the incorrect *hand* sense. For sentence 3, the CC method produced cosine similarities near zero for all senses (0.03, −0.02, and −0.04), indicating that no sense could be discerned with a high degree of confidence. For sentences with unlearned or ambiguous usage of the word *palm*, the results were similar to those observed for *bank* and *line*. The cosine similarity values were near zero in each case, indicating the sense was not identifiable with any degree of confidence.

The CC method identified the correct senses of 80% of the test sentences for the target words *raise*. For the word *raise*, the sense was misidentified as the *money* sense for the following two sentences:

1. With the new job also came a big raise in pay.
2. The federal reserve board is expected to raise interest rates.

Both sentences were identified as using *raise* in the *support* sense, which is the sense related to *support of or interest in something*. Intuitively, it would seem that the CC method should identify these sentences as using the word *raise* in the *money* sense. Further investigation into the cluster reveals that there are no words suggesting *increase* in the *money* sense cluster. As a matter of fact, the word *increase* did not appear in the top 100 synonyms for *raise*, but out of those top 100 synonyms, the word closest in meaning to the word *increase* is the word *improve* ranked 92nd. The word *improve* is included in the *support* syncluster, and that is the sense that was identified for both sentences. This gives evidence that the learning system did not induce a strong sense of *money*, like a pay raise, for the word *raise*.

In the results for the target word *serve*, the CC method identified the correct sense in 80% of the test cases. Of the two cases where the sense identification was incorrect, one was the sentence "The woman will serve on a jury for a murder trial." In this case, the correct sense was the *public* sense. The sense was incorrectly identified, but the correct *public* sense was ranked a very close second. The other sentence, "Two additional spheres attached to the bottom of the station would serve as observation points for studying the undersea environment.," again incorrectly identified the sense. The separation between the identified sense and the correct sense is a fine-grained distinction, and in this case, the error could be attributed to the human rater. For sentences with unlearned or ambiguous usage for the word *serve*, the CC method again, as evidenced by the near-zero cosine values, indicated that the sense of the target word was not identifiable.

For the sense identification task, the CC method did well in correctly identifying the sense of each of the target words in various contexts. Additionally, the cosine similarity measure produced by the CC method appears to provide information about the confidence of the sense identification and to be an indication of when a sense cannot be clearly determined. More experimentation with this method is needed to further develop these promising results. The confidence in the sense identification is a measure of the learning embodied in the LSA semantic space being used for the WSD tasks, and these observations can be used to further refine the characterizations of the cognitive model that is being employed.

6.6 Conclusion and Future Research

The experiments using the LSA-WSD system for measuring word sense discovery and word sense identification produced promising results. The synclustering approach leverages the word knowledge contained in the cognitive model to induce word senses for a given learning system and target word. The results suggest that candidate WSCs should have a cosine similarity between the centroid and the target word that exceeds 0.35 to be considered as a sense cluster for the word. While coarse-grained senses were more easily identified, a few fine-grained senses were discovered as well. The more general grade-level learning system produced more broad-based and consistent results for WSI.

It is worth noting that the LSA-WSD approach can be used as an unsupervised system, or as a semisupervised system, for WSI. The software produced for this research allows the user to judge the WSCs derived using synclustering. The user has the ability to refine the candidate WSCs by choosing a specific cosine cluster inclusion threshold as well as the number of top terms to use for clustering to capture the best senses for the target word derived from the learning system. Also, the user can indicate through input parameters which candidate WSCs to keep as induced senses to be used for the sense identification task.

For the sense identification task, the CC method performed well, identifying the correct sense for a given target word within the context sentences 84% of the time, and the associated cosine similarity measure provided information suggesting a degree of confidence for the sense identification. The cosine similarity measure also served as an indicator of when a sense could not be clearly determined.

The development of the LSA-WSD system has successfully produced a viable unsupervised learning system for automating both the sense discovery and sense identification tasks of WSD. The flexibility and adaptability of the system allows for the LSA-WSD system to be used for different applications and purposes. The system can also help to define the body of knowledge and use of language captured in the underlying learning system and to guide the creation of these systems for general application.

References

1. Agirre, E., & Edmonds, P. (2007a). Introduction. In E. Agirre & P. Edmonds (Eds.), *Word sense disambiguation: Algorithms and applications* (pp. 1–22). New York: Springer Science & Business Media.
2. Bhala, R. V., & Abirami, S. (2014). Trends in word sense disambiguation. *Artificial Intelligence Review, 42*(2), 159–171. https://doi.org/10.1007/s10462-012-9331-5
3. Martin, D. I. (2018). *A semantic unsupervised learning approach to word sense disambiguation* (Doctoral dissertation). Retrieved from http://trace.tennessee.edu/utk_graddiss/4950.
4. Pal, A. R., & Saha, D. (2015). Word sense disambiguation: A survey. *arXiv Preprint arXiv:1508.01346*. https://doi.org/10.5121/ijctcm.2015.5301.

5. Yarowsky, D. (2000). Word-sense disambiguation. In R. Dale, H. Somers, & H. Moisl (Eds.), *Handbook of natural language processing* (pp. 629–654). Boca Raton, FL: CRC Press.
6. Pedersen, T. (2007). Unsupervised corpus-based methods for WSD. In E. Agirre & P. Edmonds (Eds.), *Word sense disambiguation: Algorithms and applications* (pp. 133–162). New York: Springer Science & Business Media.
7. Turing, A. (1950). Computing machinery and intelligence. *Mind, LIX*(236), 433–460.
8. Hirst, G. (2007). Foreword. In E. Agirre & P. Edmonds (Eds.), *Word sense disambiguation: Algorithms and applications* (pp. xvii–xxix). New York: Springer Science & Business Media.
9. Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition* (2nd ed.). Englewood Cliffs, NJ: Prentice Hall.
10. Agirre, E., & Edmonds, P. (Eds.). (2007b). Word sense disambiguation: Algorithms and applications. Springer Science & Business Media, New York.
11. Schütze, H. (1998). Automatic word sense discrimination. *Computational Linguistics, 24*(1), 97–123.
12. Navigli, R. (2009). Word sense disambiguation: A survey. *ACM Computing Surveys (CSUR), 41*(2), 10. https://doi.org/10.1145/1459352.1459355
13. Fellbaum, C. (2012). WordNet. In *The encyclopedia of applied linguistics*. Chichester: John Wiley & Sons.
14. Zhou, X., & Han, H. (2005). Survey of word sense disambiguation approaches. In *FLAIRS Conference* (pp. 307–313).
15. Navigli, R., & Lapata, M. (2010). An experimental study of graph connectivity for unsupervised word sense disambiguation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 32*(4), 678–692. https://doi.org/10.1109/TPAMI.2009.36
16. Pilehvar, M. T., & Navigli, R. (2014). A large-scale pseudoword-based evaluation framework for state-of-the-art word sense disambiguation. *Computational Linguistics, 40*(4), 837–881.
17. Tomar, G. S., Singh, M., Rai, S., Kumar, A., Sanyal, R., & Sanyal, S. (2013). Probabilistic LSA for unsupervised word sense disambiguation. *International Journal of Computer Science Issues, 10*(5(2)), 127–133.
18. Martin, D. I., & Berry, M. W. (2007). Mathematical foundations behind latent semantic analysis. In T. K. Landauer, D. S. McNamara, S. Dennis, & W. Kintsch (Eds.), *Handbook of latent semantic analysis* (pp. 35–56). Mahwah, NJ: Lawrence Erlbaum Associates.
19. Martin, D. I., & Berry, M. W. (2010). Latent semantic indexing. In M. J. Bates & M. N. Maack (Eds.), *Encyclopedia of library and information sciences (ELIS)* (3rd ed., pp. 2195–3204). Oxford: Taylor & Francis.
20. Martin, D. I., Martin, J. C., & Berry, M. W. (2016). The application of LSA to the evaluation of questionnaire responses. In M. E. Celebi & K. Aydin (Eds.), *Unsupervised learning algorithms* (pp. 449–484). Cham, Switzerland: Springer International Publishing.
21. Landauer, T. K. (1998). Learning and representing verbal meaning: The latent semantic analysis theory. *Current Directions in Psychological Science, 7*(5), 161–164.
22. Landauer, T. K. (2002). On the computational basis of learning and cognition: Arguments from LSA. In *Psychology of learning and motivation* (Vol. 41, pp. 43–84). Amsterdam: Elsevier.
23. Landauer, T. K. (2007). LSA as a theory of meaning. In T. K. Landauer, D. S. McNamara, S. Dennis, & W. Kintsch (Eds.), *Handbook of latent semantic analysis* (pp. 3–34). Mahwah, NJ: Lawrence Erlbaum Associates.
24. Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review, 104*(2), 211–240.
25. Landauer, T. K., Foltz, P. W., & Laham, D. (1998a). An introduction to latent semantic analysis. *Discourse Processes, 25*(2–3), 259–284.
26. Landauer, T. K., Laham, D., & Foltz, P. W. (1998b). Learning human-like knowledge by singular value decomposition: A progress report. In M. I. Jordan, M. J. Kearns, & S. A. Solla (Eds.), *Advances in neural information processing systems* (pp. 45–51). Cambridge: MIT Press.

27. Deerwester, S., Dumais, S. T., Furnas, G., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science, 41*(6), 391–407.
28. Kintsch, W. (2007). Meaning in context. In T. K. Landauer, D. S. McNamara, S. Dennis, & W. Kintsch (Eds.), *Handbook of latent semantic analysis* (pp. 89–105). Mahwah, NJ: Lawrence Erlbaum Associates.
29. Landauer, T. K., McNamara, D. S., Dennis, S., & Kintsch, W. (Eds.). (2007). *Handbook of latent semantic analysis*. Mahwah, NJ: Lawrence Erlbaum Associates.
30. Landauer, T. K., Kireyev, K., & Panaccione, C. (2011). Word maturity: A new metric for word knowledge. *Scientific Studies of Reading, 15*(1), 92–108. https://doi.org/10.1080/10888438.2011.536130
31. Landauer, T. K., & Way, D. (2012). *Improving text complexity measurement through the Reading Maturity Metric*. Presented at the National Council on Measurement in Education, Vancouver, Canada.
32. Biemiller, A., Rosenstein, M., Sparks, R., Landauer, T. K., & Foltz, P. W. (2014). Models of vocabulary acquisition: Direct tests and text-derived simulations of vocabulary growth. *Scientific Studies of Reading, 18*(2), 130–154. https://doi.org/10.1080/10888438.2013.821992
33. Lewis, D., Yang, Y., Rose, T., & Li, F. (2004). RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research, 5*, 361–397.
34. Martin, J. C. (2016). Quantitative metrics for comparison of hyper-dimensional LSA spaces for semantic differences (Doctoral dissertation). Retrieved from http://trace.tennessee.edu/utk_graddiss/3942.
35. English Oxford Living Dictionaries. (n.d.). Retrieved from https://en.oxforddictionaries.com/.

Chapter 7
Enhanced Tweet Hybrid Recommender System Using Unsupervised Topic Modeling and Matrix Factorization-Based Neural Network

Arisara Pornwattanavichai, Prawpan Brahmasakha na sakolnagara, Pongsakorn Jirachanchaisiri, Janekhwan Kitsupapaisan, and Saranya Maneeroj

7.1 Introduction

In recent years, digital information has tended to be bigger and more complicated. Thus, it is harder for a human to analyze this information manually. From these problems, recommender systems (RS) were introduced to handle the large, complicated information and to suggest interesting information for people based on personal preference. RS constitute one application of machine learning that can be both supervised learning and unsupervised learning.

In everyday life, item recommendations are classified into two types: non-personalized and personalized. The non-personalized recommendation is a simple recommendation where items are recommended to all users without considering any historical user profile. This recommendation can be elicited from manual selections based on the popularity of items or from the new products. As the recommendations in this system are easy to implement and do not require a user profile to recommend, it has provided low efficiency and a lack of personalization, and the recommended items may not be suitable for everyone. Personalized recommendations analyze an individual user's behavior to make a suggestion to each user. Recommended items are items that have been frequently viewed, rated, or purchased by the user. This helps to increase sales, provided that the high accuracy of the recommendation matches the individual user's interest. Although this recommendation requires a

A. Pornwattanavichai · P. B. n. sakolnagara · P. Jirachanchaisiri · J. Kitsupapaisan
S. Maneeroj (✉)
Advanced Virtual and Intelligent Computing Center (AVIC), Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University, Pathumwan, Bangkok, Thailand
e-mail: arisara.p@math.sc.chula.ac.th; prawpan.b@math.sc.chula.ac.th; pongsakorn.j@math.sc.chula.ac.th; janekhwan.k@math.sc.chula.ac.th; saranya.m@chula.ac.th

M. W. Berry et al. (eds.), *Supervised and Unsupervised Learning for Data Science*, Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-030-22475-2_7

user profile before making a recommendation, many industries and researchers still mainly focus on personalized recommendations. Nowadays, this research field often calls personalized recommendations.

Recommendation techniques in RS can be mainly divided into content-based filtering (CBF) and collaborative filtering (CF). To recommend an item to a target user, CBF suggests an item similar to items that the target user liked in the past. On the other hand, CF suggests items of other users in the system who are similar to the target user; they are called neighbors.

The recommendation system is applied in various domains. One of the most popular domains in RS is a social network. The social network is the huge sources network and fast access that influences human life. Twitter is a social network in which the messages by which people communicate on Twitter are compact, simple, and fast. A user in the system can write up to 280 characters for posting and this action is called "tweeting." Twitter allows the user to receive messages from users that follow them, and these messages are also called "Tweets". Every interaction between users appears in the timeline of each user. Moreover, Twitter allows tweet to be repeated by the other user, and this action is called "retweeting." Nowadays, there are many Twitter messages on a user's timeline. Sometimes, Tweets do not match user preference. Thus, there is much research on RS that focus on the Twitter domain.

Twitter recommendations have a similar algorithm to documents including review recommendations. To recommend this type of content, LDA is a probabilistic generative model that is used to extract the latent topic by using observed content, which can be used to group similar content. Thus, many researchers aim to apply LDA to CBF or CF in the Twitter domain. Each researcher tries to embed the user and item to document and word with different recommendation assumptions. After that, model parameters from LDA are applied to make recommendations.

Nonetheless, both techniques have limitations. CBF recommends overspecialized items and these items are recommended to a target user repeatedly. However, the limitation of CBF is that items that recommended by CBF are repetitious. On the other hand, CF requires large quantities of data to find appropriate neighbors for making effective recommendations. Thus, if the quantity of data is not sufficient, the sparsity problem [1] and the cold-start problem [2] may occur. Therefore, using only CF or CBF is not enough to make effective recommendations.

From the limitations of CF and CBF, we propose an improved Twitter recommendation method by applying LDA on a hybrid recommendation, a technique that combines both CBF and CF to make effective recommendations. First, the input of this work consists of information from Twitter, which is collected as a user–tweet matrix and user–user matrix. Second, we apply the CBF process by estimating a preference of a target user with regard to each tweet to fulfill the user–tweet preference matrix using LDA, which is unsupervised topic modeling. Next, in the CF part, generalized matrix factorization (GMF), a supervised matrix factorization (MF)-based neural network (NN), is applied to the user–user matrix to find the similarity between the target user and others in the system, and is formed as a user–user similarity matrix. The completed user–tweet preference matrix and

the user–user similarity matrix from CBF and CF are used to predict the preference of the target user of each tweet. To evaluate the proposed model, we compared our proposed method with other research on the same dataset that uses LDA with either CF or CBF, which are the improved collaborative filtering algorithm using the topic model [3] and the user interest prediction in microblog using the recommendation method [4] respectively. The results show that the proposed method outperforms in terms of accuracy and coverage.

7.2 Related Works

In this section, we present related works including RS, Twitter and Tweets RS, latent Dirichlet allocation (LDA), RS based on LDA, and generalized matrix factorization (GMF).

7.2.1 Recommender System

An RS is an engine that is created to suggest items that may interest a target user. The suggestions of the RS are based on information about the target user that the user gave to the system in the past. The information contains many ratings that the target user gave within a specific domain, such as movies, music, and products. In RS, this information can be used as training data for either supervised learning or unsupervised learning such as classification or clustering problems. Recently, RS has been a popular engine implemented on many e-commerce websites to learn the preference of their customers and deliver suitable products to each person.

There are two main techniques of RS: CBF and CF [5]. CBF suggests an item similar to items that the target user liked in the past attempting to extract a preference of the target user and information on the items, which are called user profile and item profile respectively. For example, the inputs of CBF are an item-attribute matrix and user–item matrix as shown in Table 7.1.

Table 7.1 (top) shows the explicit information in the system that is attributed to each item and Table 7.1 (bottom) shows items that the user reached. From this table,

Table 7.1 Item-attribute matrix (top) and user–item matrix (bottom)

Item/attribute	Set of actors	Director	Genre
i_1	John, Scott	Emma	Romance
i_2	Kim, John	Edcar	Action
i_3	Jane, Jim	Emma	Action
i_4	Jim, Scott, Luiz	Jodan	Fantasy

User/item	$\boldsymbol{i_1}$	$\boldsymbol{i_2}$	$\boldsymbol{i_3}$	$\boldsymbol{i_4}$
u_1	0	1	0	0

Table 7.2 User–item rating matrix

User/item	i_1	i_2	i_3	i_4
u_1	2	3	–	3
u_2	3	–	2	–
u_3	–	3	5	4
u_4	5	1	4	1

the user profile and item profile can be extracted. After that, the distance between the item profile and user profile is measured using cosine distance or Euclidean distance. The item with the shortest distance is then recommended to the target user.

There are some advantages to CBF. First, CBF does not need much information. Furthermore, CBF starts recommending at an early stage and can recommend unique taste items for the target user. However, CBF is also problematic, because CBF only recommends the item that is of a similar type to the items that the target user has rated in the past. Therefore, an item that does not look similar to them is not recommended, and items that are recommended to the target user are not varied. This problem is called the serendipitous problem.

The other main technique is CF, which solves the serendipitous problem by recommending items of other users in the system who are similar to the target user. The input of CF is a user–item rating matrix, which is shown in Table 7.2. It shows the explicit information in the system, which consists of ratings that users made in the past. The first step is to find neighbors who have a similar preference to the target user. From this scenario, it assumes that the target user is user u, v is another user in the system. The similarity between user u and user v can be measured by various equations. Cosine similarity is one of the simplest equations that can calculate the similarity between users.

To calculate the similarity between users, only co-rated items can be used. Cosine similarity is shown as Eq. (7.1).

$$\text{cosine}\,(u, v) = \frac{\sum_i r_{ui} r_{vi}}{\sqrt{\sum_i r_{ui}^2}\sqrt{\sum_i r_{vi}^2}} \tag{7.1}$$

From Eq. (7.1), i denotes a co-rated item between user u and user v, r_{ui} denotes a rating that user u gave to item i, and r_{vi} denotes the rating that user v gave to item i.

After calculating the similarity between user u and others, top-k users that are most similar to user u are used as the neighbors of the target user to calculate predicted ratings in the next step. To predict the target item rating, let $N_i(u)$ denote neighbors as the set of users u and w_{uv} denotes a weight of user u and user v, which is a similiarity between them. The predicted rating $\hat{r}_{ui}$ denotes a predicted rating that user u may rate on item i, $\hat{r}_{ui}$ can be calculated as Eq. (7.2).

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i(u)} w_{uv} r_{vi}}{\sum_{v \in N_i(u)} \mid w_{uv} \mid} \tag{7.2}$$

The advantage of CF is that CF provides a variety of items. CF can recommend the items from many types depending on other users. However, CF has some limitations of sparsity. Neighbors cannot be detected if most of the users have not rated the same items. Although neighbors can be detected if they have not rated the target item, the rating of the target item cannot be predicted. This problem is called the cold-start problem. To solve the CF problem and keep the advantages of CBF, a hybrid recommendation that combines CBF and CF has been introduced [6].

In recent years, RS has become very popular and is applied in many fields such as the film industry [7–9], the music industry [10, 26], and social networking [11–13]. One of the social networks that RS has an influence on is Twitter.

7.2.2 Twitter

Twitter is a social networking service where users can post messages known as "Tweets" [14]. Users can post Tweets and read other users' Tweets. The relation type used on Twitter is many-to-many; thus, the relationship between two users may not be the same. Besides, each user can subscribe to many users to receive their Tweets, this is known as "following," and each user can have more than one subscriber, known as "followers." Furthermore, each Tweet can be re-posted by other users to their timeline, known as a "retweeting.". When you choose to follow another user, that user's Tweets appear on your Twitter page sequentially. The Tweets that appear on the main Twitter page are the mixture of Tweets from users who have been followed. Nowadays, Twitter is becoming a social network with much information, with millions of Tweets. Thus, many researchers are interested in analyzing and experimenting on many Twitter accounts. One of way of doing this is with an RS.

The recommendation technique of Twitter is similar to the way it recommends documents, reviews, and texts. The most common ways of recommending Tweets are by using CBF or CF. Some of the research that is relevant to Twitter recommendations is presented in this section.

User Interest Prediction in Microblog Using the Recommendation Method

In their work Jiantao and Ning [4] use LDA and MF to find the preference that the user assigns to each Tweet. First, the researchers use LDA to extract the user–topic distribution and the topic–Tweet distribution.

From the need to improve accuracy, they use MF on user–topic distribution to find the new user–topic distribution, which is more accurate than the normal user–topic distribution received from LDA. The equation of MF is shown in Eq. (7.3) where P is a Top-1 matrix, Q is the user–topic distribution, and R is the new user–topic distribution.

$$R \approx P \times Q^T \tag{7.3}$$

Then, they compare the new user–topic distribution with the topic–tweet distribution that is derived from LDA to find the user–tweet preference matrix, which told how much preference value the user had assigned to each Tweet using cosine similarity. For cosine similarity, Eq. (7.4) shows where A is the new user–topic distribution and B is the tweet–topic distribution.

$$\text{sim}\,(A, B) = \|A\|\,\|B\| \cos\theta \tag{7.4}$$

In their work, they only use data from the user–topic matrix to compare with the topic–tweet matrix to find the user–tweet preference matrix. Therefore, it can be concluded that they apply CBF to recommend Tweets.

Collaborative Personalized Tweet Recommendation

In their work Chen et al. [15] use collaborative ranking, latent factor, and explicit features to recommend Tweets to the target user. Retweets that users prefer are collected and computed to get the preference Tweet and make the recommendation. The collaborative ranking is a modified latent factor model with a ranking criterion. Since they want to increase the recommendation accuracy, the latent factor is used as other parameters in their model.

The latent factor is the feature that shows the possibility of retweeting. There are two types of factors, which are the word latent factor and the Tweet owner's latent factor. The word latent factor is probabilistic of a word that is about any topic. This factor figures out the user's preferred topics. The Tweet owner's latent factor explains the possibility that a publisher of the Tweet prefers any topic. This factor illustrates the similarity of the topics' preference between the user and the Tweet's owner.

In their work, they recommend tweets by using the latent factor, which is the similarity between the Tweet's owner and the user. Therefore, we can conclude that they apply the CF method to recommend Tweets.

7.2.3 Latent Dirichlet Allocation

Probabilistic topic models are algorithms that are aimed at extracting a hidden structure from a collection of documents. It uses probability to explain a hidden structure inside the documents. LDA is one of the probabilistic topic models that assumes that the document exhibits multiple hidden structures, called latent topics. For LDA, documents and words are the observed variables, and latent topics are inferred by computing their distribution, which is conditioned on the document [16].

Latent Dirichlet allocation is unsupervised learning because LDA takes input only a word document co-occurrence matrix and learns a document and word

distribution over latent topics. Over the past 10 years, LDA has also been extended to a supervised learning algorithm called supervised LDA (sLDA) [17].

From a mathematical viewpoint, a generative process of LDA can be described in three ways: generative processes, probabilistic graphical models, and joint probability distribution. Before explaining the details of each of these, we denote each notation as follows: N, M, and K are the number of words, document, and latent topic; W and D are a set of words N vocabulary and M document; β and θ are topic–word and document–topic multinomial distribution; z_n is a topic assignment of the nth word; η and α are Dirichlet prior distributions of β and θ.

The first way to describe LDA is as a generative process. Assume that the latent topics are pre-defined before any data are generated. For each document, each word can be generated without considering grammar and its order by the process shown in Table 7.3. Also, it can be represented as a probabilistic graphical model as in Fig. 7.1 and as joint probability distribution as in Eq. (7.5).

$$\begin{aligned} P(Z, W, \theta, \beta \mid \alpha, \eta) &= P(\beta \mid \eta)\, P(\theta \mid \alpha)\, P(Z \mid \theta)\, P(W \mid Z, \beta) \\ &= \prod_{k=1}^{K} P(\beta_k \mid \eta) \left[\prod_{d=1}^{M} P(\theta_d \mid \alpha) \left(\prod_{w=1}^{N} P\left(Z_{d,w} \mid \theta_d\right) \right. \right. \\ &\qquad \left. \left. P\left(W_{d,w} \mid \beta_{1:K}, Z_{d,w}\right) \right) \right] \end{aligned} \tag{7.5}$$

Table 7.3 Generative process and mathematical term of latent Dirichlet allocation

Generative process	Mathematical term
1. For each topic, randomly choose a topic distribution over words	1. For $k = 1 \ldots K$, $\beta_k \sim DIR(\eta)$
2. For each document, randomly choose a topic distribution over documents	2. For $d = 1 \ldots D$, $\theta_d \sim DIR(\alpha)$
3. For each word index,	3. For each $w_d \in d$:
3.1. Randomly choose a topic from document–topic distribution. After that, assign a topic to each word index	3.1. $z_{w_d} \sim Multi\,(\theta_d)$
3.2. Randomly choose a word from the drawn topic in 3.1 from a topic–word distribution	3.2. $w_d \sim Multi\left(\beta_{z_{w_d}}\right)$

DIR Dirichlet distribution, *Multi* multinomial distribution

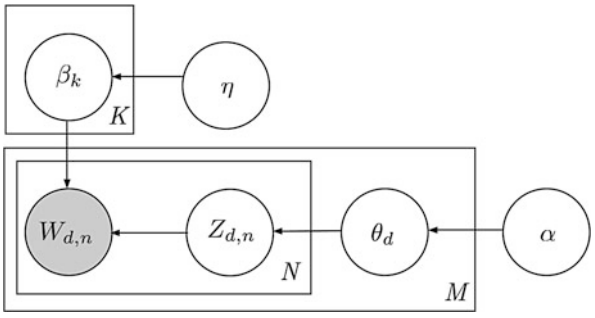


Fig. 7.1 Graphical model of latent Dirichlet allocation

For LDA learning, the generative process is reversed and the posterior distribution of the latent variables are learned given the observed data. The posterior distribution of LDA can be shown in Eq. (7.6).

$$P(Z, \beta, \theta) = \frac{P(Z, W, \beta, \theta \mid \alpha, \eta)}{P(W|\alpha, \eta)} \tag{7.6}$$

However, computing Eq. (7.6) is intractable because we cannot sum the joint distribution over all possible topics. Thus, there are approximation algorithms for LDA, including the sampling-based algorithm such as Gibbs sampling [18] and a variational algorithm.

In RS, LDA is adapted in many recommendation tasks. For example, LDA is used to extract features from sparse rating data. Moreover, LDA is also used with item reviews or comments to find topics of documents and make a recommendation based on these topics. In the next section, we present research that applies LDA to the recommendation task.

7.2.4 Recommender System with LDA

The disadvantages of CBF and CF recommendation are described above. Many researchers applied LDA to the RS to minimize those disadvantages and to improve accuracy [19, 20]. Research relevant to Tweet recommendations and applying LDA to either CF or CBF is introduced in this section.

Content-Based Filtering with LDA

Recommendations Based on LDA Topic Model in Android Applications

In their work, Pan et al. [21] recommend an application by measuring the similarity between the two applications. Thus, they measure the similarity between applications by the probability of the application's topics using Kullback–Leibler (KL) divergence. Therefore, they use the LDA process to find the probability distribution of an application's topic. First, they apply an application description as the input of the first LDA process to get application–topic distribution based on the application's content. After that, they apply application–topic distribution from the first LDA process as prior probability and users' comments as the input of the second LDA process. Finally, they obtain the probability of the application's topics.

After that, they compare the probability of the application's topics to find the distribution similarity using KL divergence. Finally, they choose the applications that have a higher similarity value with an application that the target user has ever used in the past to recommend to the target user. As they compare the similarity

value between two applications like comparing similarity values between the user profile and item profile, their work uses CBF with LDA.

Online Topic Model for Twitter Considering Dynamics of User Interests and Topic Trends

In their work, Sasaki et al. [22] improve the LDA model based on the topic tracking model (TTM) so that it should be able to estimate the dynamics of user interests and topic trends.

The TTM is a probabilistic consumer purchase behavior model based on Twitter LDA (the modified LDA model that assumes that a single Tweet consists of a single topic, and that Tweets consist of a topic and common words) for tracking the interests of each user and the trends in each topic. TTM assumes that the mean of user interests at the current time is the same as that at the previous time unless new data are observed.

To improve the Twitter LDA model with TTM, they change the user–topic parameter (θ_k) and topic–Tweet parameter (β_k) to $\theta_{t,u,k}$ and $\beta_{t,v,k}$. The $\theta_{t,u,k}$ represents the probability that user u is interested in topic k at time t, whereas $\beta_{t,v,k}$ represents the probability that word v is chosen in topic k at time t (t is a discrete variable and can be arbitrarily set as the unit time interval, e.g., at 1 day or 1 week). Moreover, the researchers say that π in the Twitter LDA model is common for all users, meaning that the rate between common words and topic words is the same for each user. However, this assumption could be incorrect, and the rate could differ with each user. Thus, they change π to π_u to infer the generative process of Tweets more efficiently.

For the recommendation process, they first collect all Tweets that user u posts in t time (they set t as 1 day). Next, they use Gibbs sampling of latent values and maximum joint likelihood to estimate parameters. After that, they bring $\theta_{t,u,k}$ to predict the topic trends that users are interested in at time t. Then, repeat this process in the next t time (next day).

In their work, they try to estimate the dynamics of user interests and topic trends by using the user–topic parameter and a topic–Tweet parameter that change from time to time. Therefore, we can conclude that they use LDA in the way of CBF.

Collaborative Filtering with LDA

Improved Collaborative Filtering Algorithm Using the Topic Model

The work of Na et al. [3] is concerned with a collaborative filtering recommender system (CF-RS), which has been introduced to recommend top-N items to the target user. Items act as documents and users act as words. Items are represented as a random mixture of latent topics; each topic is characterized by a distribution of users.

First, they apply the item–user matrix as the input of LDA. Thus, the outputs are the distribution of topic–user and the distribution of item–topic. Next, they apply these two distributions to find the user–topic distribution. After that, they find similarities between users by measuring the user–topic distribution of a pair of users in KL divergence.

Therefore, they obtain the user's neighbor according to similarity by considering the nearest neighbor. The *K*-users that are most similar to the target user are treated as the user's neighbors. The rating of the target user on the target item is predicted using a weighted average on the ratings of neighbors on the target item where the weight is a similarity between the target user and neighbor.

Their work uses LDA to find distributions of user–topic and use it to find similarity between users (to find a user's neighbors) and then recommend items of the user's neighbor to the user. Thus, it is a combination of LDA and CF.

Collaborative Topic Modeling for Recommending Scientific Articles

In their work, Wang and Blei [23] apply collaborative topic regression (CTR) to recommend scientific articles to the user by combining traditional CF with topic modeling (LDA), which emphasizes the dataset such that some users have never rated any items and some items have never been rated by any user.

They generate the CTR method, which combines LDA with probabilistic model of MF (PMF). CTR combines PMF with topic modeling by using topic proportions from LDA instead of latent item vectors from MF.

However, it still has some limitations of recommendation such that two articles have a similar topic proportion, but they influence a different target user group. Therefore, CTR adds a latent variable that offsets the topic proportion to modeling the ratings of the user that can be detected this situation. Then, they apply the EM algorithm to decompose the documents and learn the topics. If some users have never rated any items, the predicted rating is considered from a latent vector of the target user and another user in the system. Otherwise, it is considered from the latent vector of the target user and topic proportion of the item if some users have never rated any items. They apply another user's preference to predict the user–article rating. Therefore, their work uses CF.

Many previous researchers have applied LDA to either CF or CBF. However, both CF and CBF have disadvantages. The disadvantages of CBF are overspecialized items and that items from the recommendation system are proposed repeatedly to a target user. Thus, CBF cannot recommend a variety of items. On the other hand, the disadvantages of CF require many data to find appropriate neighbors for making effective recommendations. Moreover, sparsity and the cold-start problem may appear if the system has inadequate data. Therefore, using only CF or CBF is not sufficient for making effective recommendations. From the limitations of CF and CBF, we propose improving the Tweet recommendation method by applying LDA to both CF and CBF to make more effective recommendations.

7.2.5 Generalized Matrix Factorization

Generalized matrix factorization (GMF) is a part of neural MF [24], which predicts data based on MF and an NN. MF has a limitation caused by estimating complex user–item interactions in the low-dimensional latent space with a simple dot product. However, using a large number of latent factors may solve this problem, but it will cause overfitting and sparse settings. Thus, this limitation can be solved by combining this task by learning the interaction function using an NN with non-linearity. Therefore, GMF was introduced to predict unknown data from observed data, which is supervised learning. As the GMF process is based on MF and an NN, the basic concept of MF and NN are illustrated in Sections "Matrix Factorization" and "Neural Network".

Matrix Factorization

Matrix factorization is often used to find the latent feature from the interaction between user and item. The concept of MF is decomposing a matrix into two lower dimension matrices and then multiplying them back to get the original matrix.

Normally, MF input comes with a zero value. MF is used to predict zero values in the matrix, but the value that is predicted from the process has to be consistent with the other non-zero value. Thus, behind the MF process, there is an intuition, namely a latent feature that can help us to solve this problem.

The mathematical description of the MF process is shown in Eq. (7.7), where U is a set of users, D is a set of items R is a user–item rating matrix whose size is $|U| \times |D|$. There are U users and D items. R is a user–item rating matrix. First, they define a number of latent features K. Next, they find matrix $P(|U| \times K)$ and $Q(|D| \times K)$, which P and Q^T are able to compose to be matrix R as in Eq. (7.7).

$$R \approx P \times Q^T = \hat{R} \tag{7.7}$$

Each row of matrix P (p_i) represents the vector of the relations between a user and latent features. Each column of the matrix Q^T (q_j) represents the vector of the relations between an item and latent features. Next, they predict the rating ($\hat{r}_{ij}$) that user u_i gives to items d_j, which is calculated by the dot product of these two vectors as in Eq. (7.8).

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj} \tag{7.8}$$

The next part is the description of the MF process. First, they find matrix P and Q by randomly initializing values into these two matrices. Then, they find the dot product of these two matrices. They get matrix $\hat{R}$, which is a predicted matrix. Second, they find an error between R and $\hat{R}$ in each position by Eq. (7.9).

$$e_{ij} = \left(r_{ij} - \hat{r}_{ij}\right)^2 = \left(r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj}\right)^2 \tag{7.9}$$

After that, they update each p_{ik} and q_{kj} by Eq. (7.10) in every position of the matrix, where p'_{ik}, q'_{kj} are values of each element in a new round and α is a constant that is the rate of approaching the minimum.

$$\begin{aligned} p'_{ik} &= p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + \alpha \left(2 e_{ij} q_{kj} - \beta p_{ik}\right) \\ q'_{kj} &= q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + \alpha \left(2 e_{ij} p_{ik} - \beta q_{kj}\right) \end{aligned} \tag{7.10}$$

Neural Network

An NN is a kind of machine learning technique that simulates the functioning of the human nervous system. An NN can be divided into three layers, including an input layer, a hidden layer, and an output layer. The main process of the NN is to learn optimized parameters, which are weight and bias from the dataset. After that, these parameters are used to process test dataset to obtain the final output.

To explain more about the learning process, first, the parameters of the NN are randomly initialized and used to compute with the input layer to obtain a predicted value in the output layer. After that, the NN uses the error, which is the difference between the actual value and the predicted value to adjust the parameters in the next iteration. This process is repeated until the minimum error is reached. Finally, the parameters are used to carry out any tasks of the NN, including classification.

Nowadays, a deep neural network (DNN), or deep learning, which is layered stacks of an NN, plays a crucial role in the success in many fields including RS. Thus, an NN can be simply categorized into four types based on network topology. First, the perceptron, which is only the input and output layer. Second, a multi-layer perceptron (MLP), or feedforward NN, which contains one or more hidden layer from perceptron. The MLP can be extended to deep feedforward NN, depending on the number layers used. Third, a convolutional neural network that employs a convolution operation to process grid-like data. Fourth, a recurrent NN, which has special architecture for processing sequential data [25].

Generalized matrix factorization is an MLP that represents the MF process by taking input as a one-hot vector from implicit feedback. It tries to minimize the difference between actual user–item feedback and predicted feedback to obtain the weight of the NN for prediction.

7.3 The Proposed Method

From the ability of LDA to extract latent data, LDA is widely used in Twitter recommendations. However, the previous common research used LDA with either CF or CBF. On the other hand, if we bring the advantage of CBF, which does not require much data to reduce the cold-start problem, and the advantage of CF, which can recommend a variety of items, this may improve the efficiency of the recommendation. For this reason, we begin this improvement recommendation research that can recommend interesting Tweets to each user by using LDA with a hybrid recommendation (CBF and CF) on Twitter.

In this section, we first explain how we prepare data in data preparation. Then, we describe our CBF process and CF process. Finally, we show the prediction process of the user's preference for the target Tweet. The overall process of our proposed method is illustrated in Fig. 7.2.

7.3.1 Data Preparation

To prepare our data, we collected 11,685 Tweet messages by 3436 Twitter users. The average number of ratings of each user is 16. We call these data original data. Then, we created a user–tweet matrix and a user–user matrix. The user–tweet matrix represents the relationship between all users and all Tweets in binary form. If user u never tweets or retweets Tweet t, the value in position (u, t) in the user–tweet matrix is 0. Otherwise, the value is 1. The user–tweet matrix is shown in Fig. 7.3 (left).

The user–user matrix represents the relationship between any pair of users in binary form. If user u does not follow user v, the value in position (u, v) in the user–user matrix is 0. Otherwise, the value is 1. This matrix is shown in Fig. 7.3 (right).

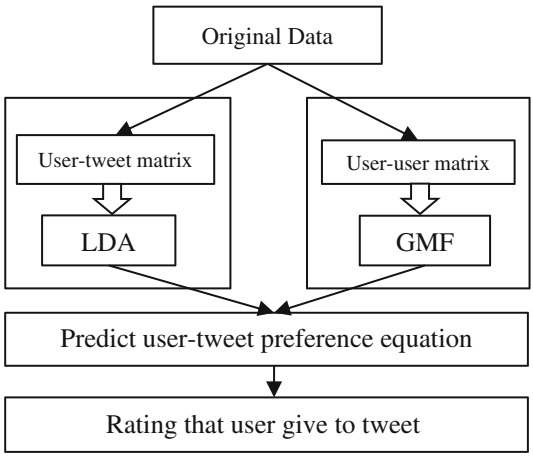


Fig. 7.2 Overall process of our proposed method

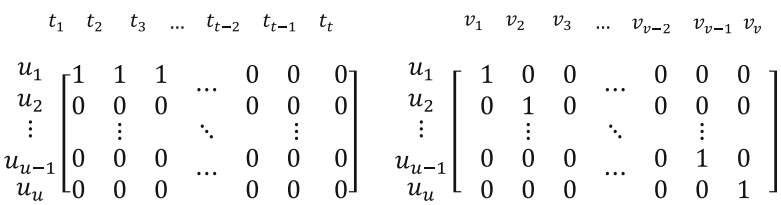


Fig. 7.3 The user–tweet matrix (left) and user–user matrix (right)

7.3.2 Content-Based Filtering Part

The CBF part is where we want to estimate the preference value of user toward item especially in the zero slot of the original user–tweet matrix (Fig. 7.3) by creating pseudo rating to fulfill this matrix by using LDA. First, we apply the original user–tweet matrix as input to find the user–topic distribution matrix (θ). Each row and each column of θ represents a user and topic respectively. Moreover, we find the topic–Tweet distribution matrix (β), where each row and each column represents each topic and each Tweet respectively, as in Fig. 7.4.

Next, we apply θ and β to Eq. (7.11), which is a dot product of θ and β. Each position in the matrix is called $u - t\ preference_{i,j}$ which estimates the preference value of user i to Tweet j. The process of CBF is illustrated as shown in Fig. 7.4.

$$u - t\ preference_{i,j} = \theta_{i1}\beta_{1j} + \theta_{i2}\beta_{2j} + \cdots + \theta_{it}\beta_{tj} \tag{7.11}$$

The procedures in this section relate only to the user's Tweet that each user has tweeted or retweeted before. Therefore, this process is CBF.

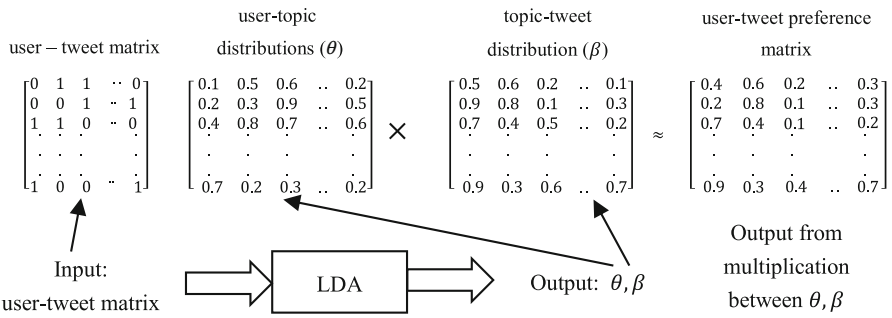


Fig. 7.4 Our content-based filtering process. *LDA* latent Dirichlet allocation

7.3.3 *Collaborative Filtering Part*

In this step, we consider generating values in the original user–user matrix as the probability of a relationship between a pair of users that users will follow another user in the system. Moreover, from the concept, if any user follows another user this means that both users tend to have a similar preference, or they are similar. Therefore, we consider the probability of a following relation between users as a similarity value between them. However, the original user–user matrix has a large amount of sparse data. To avoid this problem, we apply GMF [24] to reduce sparse data in the original user–user matrix (Fig. 7.3). In this step, we apply the original user–user matrix to the GMF process (Fig. 7.5) to obtain the predicted user–user matrix, which fulfills the sparse data in the original user–user matrix by predicting the following relation between a pair of users as their similarity. The test set was extracted from both the original user–user matrix and the user–item matrix, which considers only Twitter users from user number 0 to 1027. We mainly focus on the test set as the output data in our work. Thus, the test set was used in both the CF and CBF parts. In the CBF part, we apply the user–tweet matrix to be the input of LDA. In the CF part, we apply the user–user matrix as a test set of GMF. The training set includes only a user–user matrix, where Twitter user numbers 1028–3436 are selected. The user–user matrix from the training set is only used in GMF for learning optimal parameters such as weight and bias of GMF. After the learning phase of GMF, we use the test set for prediction and obtain results as the relationship among all users in the system.

At the beginning of GMF, we initialize NN parameters as follows: the learning rate is equal to 0.001, and the optimizer is ADAM. GMF takes inputs as a one-hot vector of two users, and then performs vector embedding of user u and user v as latent vectors p_u and q_v, and performs an element-wise product between them. Next, the product is passed to network with the sigmoid activation function to obtain predicted user–user relations. Then, it calculates error to adjust the NN weight. This step is repeated until we obtain suitable parameters for predicting user–user relations.

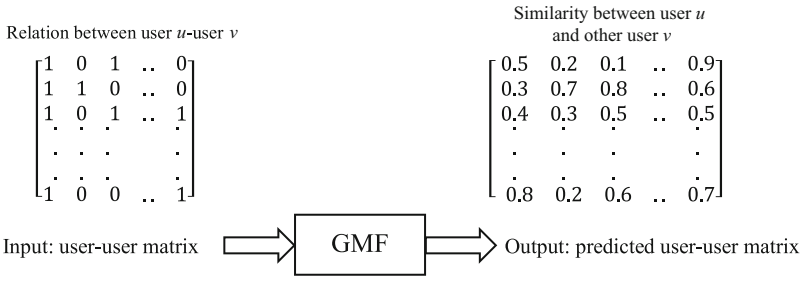


Fig. 7.5 Our collaborative filtering process. *GMF* generalized matrix factorization

In more detail, we apply vectors p_u and q_v to the mapping function of the first layer by element-wise product as in Eq. (7.12). After we obtain a vector $p_u \bigodot q_v$ from Eq. (7.12), we project this vector to the output layer in Eq. (7.13). Finally, we obtain the output as the similarity value between user u and user v

$$\varnothing_1 \ (p_u, q_v) = p_u \bigodot q_v \tag{7.12}$$

$$\hat{y}_{u,v} = a_{\text{out}} \left(h^T \left(p_u \bigodot q_v \right) \right) \tag{7.13}$$

Where $\hat{y}_{(u,v)}$ denotes the similarity that user u gives to user v, h^T and a_{out} denote the activation function and output layer weights respectively, and $\bigodot$ is the element-wise product. In this work, we use a sigmoid activation function as in Eq. (7.14).

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} \tag{7.14}$$

After we finish GMF, we obtain a predicted user–user matrix without unknown data where the values in the matrix are the probability of showing how similar users are to each other, as shown in Fig. 7.5. Then, we rank top k maximum similarities between the target user and the other users to get the neighbor list. Which is used in the next part.

In this step, we consider the process of estimating the similarity between any pair of users to find the user's neighbors. Thus, this step is the CF procedure.

7.3.4 Prediction Step

In this step, we focus on predicting Tweets that the target user may like most by applying a user–tweet preference matrix generated from the CBF part and the user–user similarity matrix, which is the output of GMF from the CF part, together. First, we pick Tweet t and find the pseudo-preference value that target user u gives to Tweet t using the user–tweet preference matrix generated from the CBF part, denoted as $rating_{u,t}$. From the CF part, we obtain top-k neighbors of the target user u from both GMF and other users in the system who have ever tweeted or retweeted the target Tweet t. As some neighbors from GMF did not interact with all target Tweets, the prediction of the target Tweet cannot be calculated. Therefore, we mainly consider other users who have ever interacted with the target Tweet to be more user's neighbors.

After that, we can obtain the similarity value between target user u and neighbor v_i, denoted as sim_{u,v_i} from the user–user matrix and obtain the pseudo-preference value that user u's neighbor (v_i) gives to Tweet t by the user–tweet matrix from LDA from the CBF part, denoted as $rating_{v_i,t}$. When we have obtained sim_{u,v_i} and $rating_{v_i,t}$, we multiply them to obtain $sim_{u,v_i} \times rating_{v_i,t}$ and sum this value from

all neighbors N of user u. Then, we divide it by the summation of sim_{u,v_i} from all neighbors of user u (N). Thus, we have already got the value of the CF part. Next, we multiply the value of CF part by 0.4 and sum it with $rating_{u,t}$, which is the value from the CBF part that was multiplied by 0.6. After we obtained two values from both the CF and CBF parts, they are calculated as P_{ut} in Eq. (7.15).

$$P_{ut} = 0.4\left(\frac{\sum_{i=1}^{N} sim_{u,v_i} \times rating_{v_i,t}}{\sum_{i=1}^{N} sim_{u,v_i}}\right) + 0.6\left(rating_{u,t}\right) \tag{7.15}$$

where sim_{u,v_i} denotes the preference value the target user u has toward the neighbor v_i, and $rating_{u,t}$ denotes the pseudo-preference value that a user u gives to Tweet t. Then, P_{ut} of all Tweets from each user u would be ranked into a top-N Tweet, which the system will recommend to the target user u.

In conclusion, we provide Eq. (7.15) as the hybrid recommendation equation for predicting user–item rating, which combines both the CBF and CF parts. We give each part a different weight. Because we think the way to estimate the target user item rating is based on him/herself using his/her rating history is not from estimating like the target user neighbors which were predicted from user–user matrix by GMF. Thus, we mainly focus on the CBF part. Therefore, we weight the CBF part by 0.6, which is greater than the CF part, which is weighted by 0.4.

In this step, our process finds the top 10 Tweets to recommend to the target user by considering the user–tweet preference matrix and predicted user–user matrix generated from the CBF and CF parts respectively. Thus, this process is a hybrid procedure.

7.4 Experimental Results

To evaluate the effectiveness of the proposed method, it is presented by comparing our method with those of other researchers who use LDA with either CBF or CF.

7.4.1 Dataset

The Dataset in our research consists of 11,685 Tweets provided by 3436 users. We separate 1028 users and another 2408 users to use as a test set and a training set respectively. More details of the dataset are shown in Table 7.4.

Table 7.4 Dataset characteristics

Dataset characteristics	Number
Number of users	1028
Number of tweets	11,685
Number of ratings	16,312
Minimum and maximum tweets per user	1 and 341
Average number of tweets per user	16

7.4.2 Evaluation Metrics

To compare the efficiency of our research, we choose the mean absolute error (MAE) and coverage as our metrics. The MAE is the mean of the errors between the predicted data and the actual data. If the research has a low MAE value, it means a high prediction accuracy. The equation of MAE is shown as Eq. (7.16), where $r_{u,\,i}$ is the real rating that the user u gives to item i, $\hat{r}_{u,i}$ is the predicted rating from the model that the user u would give to item i, and n is the number of the data in the test set.

$$\text{MAE}\,(u_i) = \sum_{i=1}^{n} \frac{\left|r_{u,i} - \hat{r}_{u,i}\right|}{n} \tag{7.16}$$

Coverage is used to measure the percentage of items in the test set that the system can recommend. The equation of coverage is shown as Eq. (7.17), where I is the set of items in the test set, Ip is the set of items for which a prediction model can be made.

$$\text{Prediction coverage} = \frac{\left|I_{\text{p}}\right|}{|I|} \times 100 \tag{7.17}$$

7.4.3 Experimental Results

The results of the proposed method, which is the result of the LDA combined with the hybrid recommendation, are compared with research that uses LDA with either the CBF or the CF technique. Research involving only the CBF is the user interest prediction in microblog using the recommendation method Jiantao and Ning [4] and research that includes only the CF is the improved collaborative filtering algorithm using the topic Model Na et al. [3]. Both forms of research are described in Sections "User Interest Prediction in Microblog Using the Recommendation Method" and "Collaborative Filtering with LDA", respectively. Along with the results of the proposed method, those of the two forms of research are implemented on the same dataset to avoid bias.

Table 7.5 The mean absolute error (MAE) and coverage results

Research titles	MAE	Coverage
The proposed method (hybrid with LDA)	0.018	100%
User interest prediction in microblog using the recommendation method (CBF with LDA)	0.260	100%
Improved collaborative filtering algorithm using the topic model (CF with LDA)	0.995	11.27%

LDA latent Dirichlet allocation, *CBF* content-based filtering, *CF* collaborative filtering

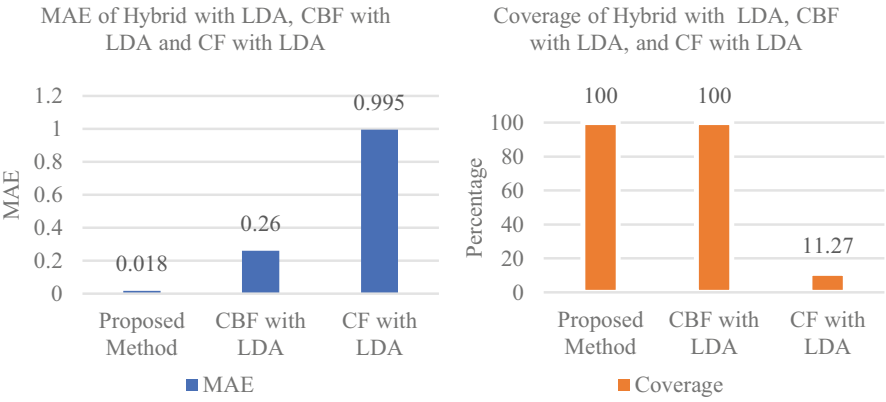


Fig. 7.6 Mean absolute error (MAE; left) and coverage (right) comparison

According to Table 7.5, the proposed method obtained the lowest MAE. Therefore, the proposed method has higher accuracy than the other two forms of research. However, coverage of the proposed method obtained 100%, whereas the CF with LDA [3] predicted only 11.27%. We summarize the experimental results in Fig. 7.6.

7.5 Discussion

The experimental results from Sect. 7.4 show that the proposed method, which is hybrid RS with LDA, achieves greater effectiveness than the two forms of research in terms of MAE and coverage. In this section, we discuss the proposed method's strength over the previous research in terms of applying LDA with either CBF or CF.

7.5.1 Comparison Between the Proposed Method and User Interest Prediction in Microblog Using the Recommendation Method (CBF with LDA)

The experimental results indicate that the proposed method has lower MAE than the work of Jiantao and Ning [4]. This improvement comes from the proposed method's consideration of targeting the user's neighbors.

In the recommendation model of Jiantao and Ning [4] CBF is performed in three steps. First, they use LDA on Twitter to find the user–topic and topic–Tweet distribution. Then, they use singular value decomposition (SVD) to reconstruct user–topic distribution. Finally, they make a CBF recommendation from the predicted user–topic distribution. However, using only information from a target user without information from neighbors leads to poor recommendations because the recommended Tweets will be historical Tweets in which the target user has expressed their preferences. Moreover, learning user–topic distribution in LDA causes errors, but applying SVD to the estimated user–topic distribution from LDA, which again contains errors, will lead to more errors in the reconstruction from SVD.

In contrast, the proposed method makes a CBF recommendation by using results from LDA directly on the user–tweet preference matrix. It helps to reduce errors from user–topic distribution reconstruction from SVD. Furthermore, the proposed method is enhanced by using neighbors derived from the predicted user–user matrix from GMF to perform CF. To determine the importance of the predicted rating from CBF and CF, we trust that using the target user's information to estimate their preference (CBF) will be more precise than the rating from the neighbors (CF). Thus, the proposed method has assigned weight to the rating from CBF more than that from CF. Moreover, using both pieces of information brings the advantages of CBF and CF to the proposed method.

Recommendations from the proposed method are balanced between the target's user preference and the neighbors' preference. Some Tweets that the target user may prefer but are not related to the target user's content are recommended by the neighbors. In other words, using information from the neighbor allows the target user to explore new items that have more variety and that are maybe unexpectedly preferred by the target user. These would help to increase the accuracy of recommendations.

In part of the coverage, Jiantao and Ning [4] reaches 100% because they perform CBF by applying user–topic distribution from LDA. In the same way, the proposed method performs CBF by applying the predicted user–tweet preference matrix. These distributions, which are applied in both the work of Jiantao and Ning [4] and the proposed method, can be estimated by a sampling algorithm. Thus, applying LDA to the user–tweet interaction matrix and performing CBF on the LDA results can achieve 100% recommendation coverage.

7.5.2 *Comparison Between the Proposed Method and the Improved Collaborative Filtering Algorithm Using the Topic Model (CF with LDA)*

The experimental results also indicate that the proposed method has a lower MAE than the work of Na et al. [3]. This improvement comes from the better neighbor detection of the proposed method.

The proposed method and the work of Na et al. [3] have different techniques for neighbor detection. The work of Na et al. [3] detects the target user's neighbors by comparing the user–topic distribution of a pair of users. If the target user and another user have a high degree of similarity of user–topic distribution, this user will be a neighbor of the target user. However, this interaction matrix shows the relation between the user and the Tweet, not the user following relation, which is the main characteristic and information of Twitter. Thus, by using the user–tweet relation to estimate the user–user relation may not be good enough to find precise user similarity.

On the contrary, the proposed method finds neighbors by comparing similarity from the dense user–user matrix from GMF. The input of GMF is observed from the user–user matrix, which shows the user–user following relation. For an example of the relation, if user A on Twitter prefers another user, namely user B, user A will follow user B because user A wants to follow Tweets from user B. Thus, by using similarity from the predicted user–user matrix, which comes from the user–user following relation, leads to a better user relation than similarity from a user–topic distribution that comes from the relation between user and item only.

For coverage, the proposed method is higher than that of Na et al. [3]. Their work is applied to Tweets by using binary ratings in which 0 indicates that the user does not tweet a tweet whereas 1 indicates that the user does tweet a Tweet. By applying binary data, the rating from the recommendation equation of that work can often be zero. This leads to a sparse matrix and causes two problems. If the neighbors cannot be found, the system cannot recommend the item to the target user. Although the system can find a few neighbors, these neighbors are low-quality neighbors and lead to an inaccurate recommendation. These effects cause low coverage in the results.

In contrast, the proposed method finds similarity by using the predicted user–user matrix from GMF. Therefore, the proposed method can obtain neighbors from this matrix with certainty. Moreover, we apply pseudo-ratings of the target user to perform CBF, whose rating can be computed from LDA certainty. Thus, applying GMF to find the predicted user–user matrix and using pseudo-ratings achieve 100% coverage.

7.6 Conclusion

We propose a new Tweet recommendation method, namely the enhanced Tweet hybrid recommender system using unsupervised topic modeling and matrix factorization-based neural network. The proposed method recommends Tweets for the target user by using unsupervised topic modeling, namely LDA and matrix factorization-based NN, namely GMF. LDA is applied in the CBF part to estimate user's preference in Tweets to fulfill the user–tweet preference matrix. In the CF part, GMF, a supervised MF-based NN, is used to predict similarity between users. The top-k users who are most similar to the target user will be the target user's neighbors. The fulfilled user–tweet preference matrix from CBF and the similarity between the target user and their neighbors from CF are combined to predict the rating that the target user will give to Tweet t. The top-N Tweets that have the highest rating will be recommended to the target user. From the evaluation, we found that the proposed method has a higher accuracy rate and greater coverage than the current recommender systems that apply LDA with either CBF or CF.

References

1. Guo, G. (2012). Resolving data sparsity and cold start in recommender systems. Paper presented at the proceedings of the 20th international conference on user modeling, adaptation, and personalization, Montreal.
2. Ahn, H. J. (2008). A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences, 178*, 37–51.
3. Na, L., Ying, L., Xiao-Jun, T., Hai-Wen, W., Peng, X., & Ming-Xia, L. (2016). Improved collaborative filtering algorithm using topic model. In *17th international conference on parallel and distributed computing, applications and technologies (PDCAT)*, 16–18 Dec 2016 (pp. 342–345).
4. Jiantao, Z., & Ning, S. (2014). User interest prediction in microblog using recommendation method. In *IEEE 7th joint international information technology and artificial intelligence conference* (pp. 367–370).
5. Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering, 17*, 734–749.
6. Ricci, F., Rokach, L., Shapira, B., & Kantor, P. B. (2010). *Recommender systems handbook*. New York: Springer-Verlag.
7. Agrawal, S., & Jain, P. (2017). An improved approach for movie recommendation system. Paper presented at the international conference on I-SMAC (IoT in social, mobile, analytics and cloud), 10–11 Feb 2017.
8. Arora, G., Kumar, A., Devre, G. S., & Ghumare, A. (2014). Movie recommendation system based on users' similarity. *International Journal of Computer Science and Mobile Computing, 3*, 765–770.
9. Ilhami, M., & Suharjito. (2014). Film recommendation systems using matrix factorization and collaborative filtering. In *International conference on information technology systems and innovation (ICITSI)* (pp. 1–6).

10. Shakirova, E. (2017). Collaborative filtering for music recommender system. In *IEEE conference of Russian young researchers in electrical and electronic engineering (EIConRus)*, 1–3 Feb 2017 (pp. 548–550).
11. Anandhan, A., Shuib, L., Ismail, M. A., & Mujtaba, G. (2018). Social media recommender systems: Review and open research issues. *IEEE Access, 6*, 15608–15628.
12. Jonnalagedda, N., & Gauch, S. (2013). Personalized news recommendation using Twitter. In *2013 IEEE/WIC/ACM international joint conferences on web intelligence (WI) and intelligent agent technologies (IAT)* (pp. 21–25).
13. Nguyen, D. L., & Le, T. M. (2016). Recommendation system for Facebook public events based on probabilistic classification and re-ranking. In *8th international conference on knowledge and systems engineering (KSE)*, 6–8 Oct 2016 (pp. 133–138).
14. Younghoon, K., & Kyuseok, S. (2014). TWILITE: A recommendation system for Twitter using a probabilistic model based on latent Dirichlet allocation. *Information Systems, 42*, 59–77.
15. Chen, K., Chen, T., Zheng, G., Jin, O., Yao, E., & Yu, Y. (2012). Collaborative personalized tweet recommendation. Paper presented at the proceedings of the 35th international ACM SIGIR conference on Research and Development in information retrieval, Oregon
16. Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research, 3*, 993–1022.
17. Blei, D. M., & McAuliffe, J. D. (2007). Supervised topic models. Paper presented at the proceedings of the 20th international conference on neural information processing systems, Vancouver.
18. Steyvers, M., & Griffiths, T. (2007). Latent semantic analysis: A road to meaning. In T. Landauer, S. D. McNamara, & W. Kintsch (Eds.), *Probabilistic topic models*. Mahwah, NJ: Laurence Erlbaum.
19. Chang, T.-M., & Hsiao, W.-F. (2013). LDA-based personalized document recommendation. In *Pacific Asia conference on information systems*. Korea: Association for Information Systems.
20. Godin, F., Slavkovikj, V., Neve, W. D., Schrauwen, B., & Van der Walle, R. (2013). Using topic models for twitter hashtag recommendation. Paper presented at the proceedings of the 22nd international conference on world wide web, Rio de Janeiro.
21. Pan, T., Zhang, W., Wang, Z., & Xu, L. (2016). Recommendations based on LDA topic model in android applications. In *IEEE international conference on software quality, reliability and security companion (QRS-C)*, 1–3 Aug 2016 (pp. 151–158).
22. Sasaki, K., Yoshikawa, T., & Furuhashi, T. (2014). Online topic model for twitter considering dynamics of user interests and topic trends. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)* (pp. 1977–1985). Association for Computational Linguistics.
23. Wang, C., & Blei, D. M. (2011). Collaborative topic modeling for recommending scientific articles. Paper presented at the proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining, San Diego.
24. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. Paper presented at the proceedings of the 26th international conference on world wide web, Perth.
25. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning. Adaptive computation and machine learning series*. Cambridge, MA: MIT Press.
26. Kodama, Y., Gayama, S., Suzuki, Y., Odagawa, S., Shioda, T., Matsushita, F., et al. (2005). Music recommendation system. In *International conference on consumer electronics (ICCE)*, 8–12 Jan 2005 (pp. 219–220).

Chapter 8
New Applications of a Supervised Computational Intelligence (CI) Approach: Case Study in Civil Engineering

Ameer A. Jebur, Dhiya Al-Jumeily, Khalid R. Aljanabi, Rafid M. Al Khaddar, William Atherton, Zeinab I. Alattar, Adel H. Majeed, and Jamila Mustafina

8.1 Introduction

Artificial neural networks (ANNs) can be classified as computational systems based on a simplified analogy to mimic the processing information of the human brain. An ANN network comprises of three interconnected layers, following the order: input layer, hidden layer(s), and output layer(s). Information in each layer is processed by the neurons that communicate with the neurons in the next layer throughout the network connections weights (w_{ij}) and biases [1, 2]. Specified a dataset of independent input and output parameters, the candidate ANN model can

A. A. Jebur (✉)
Department of Civil Engineering, Liverpool John Moores University, Liverpool, UK

Department of Civil Engineering, Al-Mustansiriya University, Baghdad, Iraq
e-mail: A.A.Jebur@2015.ljmu.ac.uk

D. Al-Jumeily
Department of Computer Science, Liverpool John Moores University, Liverpool, UK

K. R. Aljanabi
Department of Civil Engineering, University of Anbar, Ramadi, Iraq

R. M. Al Khaddar · W. Atherton
Department of Civil Engineering, Liverpool John Moores University, Liverpool, UK

Z. I. Alattar · A. H. Majeed
Department of Civil Engineering, Al-Mustansiriya University, Baghdad, Iraq
e-mail: A.A.Jebur@2015.ljmu.ac.uk

J. Mustafina
Kazan Federal University, Kazan, Russia

M. W. Berry et al. (eds.), *Supervised and Unsupervised Learning for Data Science*,
Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-030-22475-2_8

be trained: the network connection weight of the various neurons is updated during the training process in order the for the trained ANN model to model out in which the difference between the measured and the predicted values is minimal. During the data preprocessing, it is a common practice to divide the dataset being studied into three groups: training, testing, and cross-validation subset [3].

The training aim is to determine the network internal parameters by adjusting the connection weight (w_{ij}) and network bias (b_j) at each epoch during the training process. The testing set is placed to evaluate the reproducibility of the model being developed. It should be noted that the testing dataset was not involved during the training and it is identified to assess the generalization ability of the developed ANN model [4]. The cross-validation set is piloted to test the model performance, and to halt the process of learning once the network error starts to increase as a consequence of overfitting [5, 6].

In this chapter, artificial neural networks (ANNs) are used to predict the nonlinear hyperbolic soil stress–strain parameters (k and R_f). A database of actual laboratory measurements of these parameters is used to develop and verify the candidate ANN models. Supervised Feed-forward multilayer perceptions (MLPs) robust model, trained with the back-propagation algorithm technique, is employed in this process, as it has a high capability of data mapping and proved to be an accurate data-driven tool to solve a wide range of nonlinear engineering problems as they deliver a reliable model, which could be continuously updated (the network connections weights and biases) as a new dataset becomes available [7, 8]. MLPs trained with back-propagation algorithm have been applied successfully to reliably solve many geotechnical engineering problems [5]. Hence, they have been used in this work.

The objectives of this chapter are to

1. Examine the feasibility of three supervised ANN models for predicting the nonlinear soil stress–strain parameters (log k and R_f).
2. Study the influence of ANN geometry and internal parameters on the performance of ANN models.
3. Conduct comprehensive statistical analyses study to identify the main effective independent input parameters and to highlight the contribution of each input parameter on the ANN model outputs (k and R_f).
4. Develop a novel empirical equations that the geotechnical engineer for prediction of the hyperbolic nonlinear stress–strain parameters for soils tested under unconsolidated undrained conditions, this offer high efficiency solution to be used in the soil–structure interaction design stage instead of conducting expensive, tedious, and time-consuming trail axial tests type unconsolidated undrained (UU) tests in order to determine the main soil stress–strain parameters (k and R_f).

8.2 Prediction of Hyperbolic Nonlinear Soil Stress–Strain Parameters (log k and R_f) by a Supervised Artificial Neural Network (ANN)

8.2.1 Development of ANN Models

The data used to calibrate and validate the neural network models are obtained from the literature, they include laboratory measurements to determine the nonlinear soil stress–strain parameters that are vital in the soil–structure design process, and these are k, n, and R_f. The data cover a range of soil types. The database comprises a total of (83) cases recorded, and can be found in the literature. Full details of the database are given in Appendix 1. The steps for developing ANN models, as outlined in the following algorithm, are used as a guide in this work. These include the determination of model inputs and outputs, preprocessing and division of the available data, scaling of data, the determination of appropriate network architecture, and optimization of the connection weights. A PC-based commercial software system called Neuframe Version 4.0 (Neusciences 2014) is used, in which optimal network architecture is determine by trial and error.

A variety of ANN topologies have been developed to solve problems in many applications. One of the most popular and robust ANN configurations is the error back-propagation algorithm [9–12]. The error feed-forward back-propagation technique has been proven to be a highly efficient tool in modeling nonlinear relationships [13, 14]. The algorithm updates the ANN weights in such a way that the error of the network output is decreased as set in the original goal. An ANN is constructed in such a way that each processing element in a specific layer is fully connected to the next layer. Alternatively stated, every single node in the input layer will send its output to every neuron in the middle layer; consequently, every neuron in the input layer will then send its output to every neuron in the model output layer [15].

The multilayer back-propagation ANN learning process comprises of two processing elements passing through the different network layers: a forward pass and backward pass, by computing the gradient for each connection weight and bias utilizing the chain rule, as seen in Fig. 2.8. During the forward step, the synaptic network weights are all assumed to be fixed, whereas at the backward pass, the synaptic network weights are adjusted according to an updated error. This iteration procedure is repeated during the training process, which propagates the term of error needed for weight adjustment until the trained network can provide a set of connection weights, which has the input/output mapping that contains the minimum error value. Among the enhancements to the error back-propagation algorithm that have been produced, one method involves the use of the learning rate. The learning rate parameter (η) can be categorized as the factor that initiates the step size that the ANN takes in negative through the weight spaces in an attempt to minimize the training error magnitude. The momentum term (α) is another factor that needs to be considered in any ANN training process [16]. The objective of the α value is to

increase the step size when the weight space direction is the same as the previous step direction and vice versa. The Algorithm of Error in Back-propagation technique can be summarized in the following procedure:

1. Initialize the network connections weight between the measured and the predicted values.
2. Repeat the following steps until some criteria.
3. Sum up weighted inputs and apply the transfer function to compute the output of hidden layer:

$$h_i = f \sum_{i=1}^{n} \left(x_i \; w_{ij}\right) + \theta_j \tag{8.1}$$

h_i = hidden neuron output
x_i = input signal
w_{ij} = connection weight between input neuron i and hidden neuron j
f = the activation function
θ_j = bias on hidden neuron

4. Sum weighted output of hidden layer and apply the activation function to compute the output layer:

$$y_k = f \sum_{i=1}^{n} \left(h_j \; w_{jk}\right) + \theta_k \tag{8.2}$$

y_k = independent output
w_{ik} = connection weight between hidden nodes j and k

5. Back-propagation combinations

$$\delta_k = (d_k - y_k)\, \overline{f} \left(\sum j h_j w_{jk} + \theta_k \right) \tag{8.3}$$

$\overline{f}$ = the derivation of the activation function
d_k = the desired output of the neuron

6. Calculate of the weight correction term

$$\Delta w_{jk}(n) = \eta \delta_k h_j + \alpha \Delta w_{jk}\,(n-1) \tag{8.4}$$

$\Delta w_{jk}(n)$ = adjustment on connection weight between nodes j and k
η = learning rate
α = momentum term
h_j = actual output of hidden neuron
δ_k = back-propagation error
$\Delta w_{jk}(n-1)$ = previous weight correction

7. Sum delta input for each hidden unit and calculate error term

$$\delta_j = \sum k\delta_k w_{jk}\ \overline{f}\left(\sum i x_i w_{ij}\right) \tag{8.5}$$

8. Calculate weight correction term

$$\Delta w_{ij}(n) = \eta\delta_j x_i + \alpha\Delta w_{ij}\,(n-1) \tag{8.6}$$

9. Update weights

$$w_{jk}\,(\text{new}) = w_{jk}\,(\text{old}) + \Delta w_{jk} \tag{8.7}$$

$$w_{ij}\,(\text{new}) = w_{ij}\,(\text{old}) + \Delta w_{ij} \tag{8.8}$$

10. Compute the sum square error

$$\text{SSE} = \frac{1}{2}\sum_K (d_k - y_k)^2 \tag{8.9}$$

K = number of output neurons

11. End

8.2.2 Model Inputs and Outputs

It is generally accepted that four parameters have the most significant impact on the hyperbolic parameters, and are thus used as the ANN model inputs. These include the following:

1. Plasticity index, (PI) %
2. Unit weight (γ_{dry}) kN/m^3
3. Confining stress (CS) kN/m^2
4. Water content (wc) %

The outputs of the model are logarithm of modulus number, modulus exponent, and failure ratio (log k, n, and R_f). As one output for each network, a code is used in this chapter to identify the names of the different models developed. The code consists of two parts separated by hyphen. The first part represents an abbreviation of the current output (i.e., logarithm of modulus number, modulus exponent, and failure ratio (log k and R_f)). The second part denotes the model number; for example, "log k − 1" represents logarithm of modulus number, model number (1). The available data extracted from the database are given in Appendix 2.

8.2.3 Preprocessing and Data Division

Data processing is very important in using neural nets successfully. It determines what information is presented to create the model during the training phase. It can be in the form of data scaling, normalization, and transformation. Transforming the input data into some known forms (e.g., log., exponential, etc.) may be helpful to improve ANN performance. Thus, the modulus number is transformed to (log k). The next step in the development of ANN models is dividing the available data into their subsets (training, testing, and validation). Cross-validation is used as the stopping criteria in this study. Consequently, the data are randomly divided into three sets: training, testing, and validation, as is the standard practice in the development of ANN models in geotechnical engineering. In total, 80% of the data are used for training and 20% are used for validation. The training data are further divided into 70% for the training set and 30% for the testing set. These subsets are also divided in such a way that they are statistically consistent and thus represent the same statistical population. In order to achieve this, several random combinations of the training, testing, and validation sets are tried until three statistically consistent data sets are nearly obtained. The statistics of the training, testing, and validation sets for the ANN models are shown in Tables 8.1 and 8.2 for the nonlinear stress–strain parameters (log k and R_f).

Table 8.1 Input and output statistic for ANN model of stress–strain parameters for soil tested under unconsolidated undrained conditions (UU-test) for (log k)

		Input variables				Output
Data set	Statistical parameters	Plasticity index (%)	Unit weight (kN/m^3)	Water content (%)	Confining stress (kN/m^3)	log k
Training set	Max.	45.000	18.380	28.600	946.739	3.949
	Min.	1.000	10.630	8.710	193.651	1.000
	Mean	18.533	16.493	17.131	381.310	2.225
	S.D.*	8.142	1.605	4.340	192.484	0.621
	Range	44.000	7.750	19.890	753.088	2.949
Testing set	Max.	36.000	19.000	27.400	860.672	3.949
	Min.	1.000	14.450	8.710	207.500	1.556
	Mean	18.810	16.548	17.334	409.004	2.406
	S.D.*	9.862	1.298	5.371	181.228	0.656
	Range	35.000	4.550	18.690	653.172	2.393
Validation set	Max.	38.000	20.170	31.100	1452.380	3.699
	Min.	1.000	10.060	8.300	107.584	1.204
	Mean	19.529	16.506	17.394	412.029	2.290
	S.D.*	9.805	2.318	6.733	337.503	0.731
	Range	37.000	10.110	22.800	1344.796	2.495

*SD = Standered Deviation

Table 8.2 Null hypothesis test for ANN input variable and outputs for soil tested under unconsolidated undrained conditions (UU-test) for log k

Variable and data set	T-value	Lower critical value	Upper critical value	t-test	F-value	Lower critical value	Upper critical value	F-test
Plasticity index (PI) (%)								
Testing	−0.25	−2.00	2.00	*	0.68	0.49	2.27	*
Validation	−0.83	−2.00	2.00	**	0.69	0.47	2.49	*
Dry unit weight (kN/m^3)								
Testing	−0.29	−2.00	2.00	*	1.53	0.49	2.27	*
Validation	−0.05	−2.00	2.00	*	0.48	0.47	2.49	*
Water content ω_o (%)								
Testing	−0.34	−2.00	2.00	*	0.65	0.49	2.27	*
Validation	−0.37	−2.00	2.00	*	0.42	0.47	2.49	**
Confining stress (kN/m^2)								
Testing	−1.15	−2.00	2.00	*	1.13	0.49	2.27	*
Validation	−0.92	−2.00	2.00	*	0.33	0.47	2.49	**
log k								
Testing	−2.00	−2.00	2.00	*	0.90	0.49	2.27	*
Validation	−0.72	−2.00	2.00	*	0.72	0.47	2.49	*

Key: * = Accept, ** = Reject

Table 8.3 Input and output statistic for ANN model of stress–strain parameters for soil tested under unconsolidated undrained conditions (UU-test), for (R_f)

Data set	Statistical parameters	Input variables				Output
		Plasticity index (%)	Unit weight (kN/m^3)	Water content (%)	Confining stress (kN/m^3)	R_f
Training set	Max.	45.000	18.540	28.800	946.739	1.000
	Min.	4.000	10.060	8.710	193.651	0.570
	Mean	18.489	16.391	17.414	404.518	0.844
	S.D.*	8.209	1.789	4.442	214.763	0.100
	Range	41.000	8.480	20.090	753.088	0.430
Testing set	Max.	36.000	20.170	28.600	860.672	0.95
	Min.	1.000	14.520	8.300	107.584	0.520
	Mean	19.190	16.654	17.043	372.336	0.841
	S.D.*	10.038	1.356	5.771	179.117	0.115
	Range	35.000	5.650	20.300	753.088	0.430
Validation set	Max.	38.000	19.720	31.100	1452.380	0.970
	Min.	4.000	11.590	9.600	193.651	0.550
	Mean	19.176	16.644	17.006	395.893	0.866
	S.D.*	9.429	1.840	6.096	300.119	0.110
	Range	34.000	8.130	21.500	1258.729	0.420

The statistical parameters considered include the maximum, minimum, mean, standard deviation, and range. To examine how representative the training, testing, and validation sets are with respect to each other, *t*-test and *F*-test are carried out. The *t*-test examines the null hypothesis of no difference in the means of two data sets and the *F*-test examines the null hypothesis of no difference in the variances of the two sets. For a given level of significance, test statistics can be calculated to test the null hypotheses for the *t*-test and *F*-test, respectively. Traditionally, a level of significance equal to 0.05 was selected. Consequently, this level of significance is used in this study. This means that the confidence level (CI) is 95%. This is in agreement with previous study conducted by Jebur et al. [17]. The results of the *t*-test and *F*-tests are given in Tables 8.3 and 8.4 for the logarithm of modulus number (log k), and the failure ratio (R_f), respectively. Results indicate that all data, training, testing, and validation sets are generally representative of a single population.

8.2.4 Scaling of Data

Once the available data have been divided into their subsets, the input and output variables are preprocessed by scaling them to eliminate their dimension and to ensure that all variables receive equal attention during training. Scaling has to be commensurate with the limits of the transfer functions used in the hidden and output layers (i.e., −1.0 to 1.0 for tanh transfer function and 0.0 to 1.0 for sigmoid

Table 8.4 Null hypothesis test for ANN input variable and outputs for soil tested under unconsolidated undrained conditions (UU-test) for R_f

Variable and data set	T-value	Lower critical value	Upper critical value	t-test	F-value	Lower critical value	Upper critical value	F-test
Plasticity index (PI) (%)								
Testing	−0.62	−2.00	2.00	*	0.67	0.49	2.27	*
Validation	−0.58	−2.00	2.00	*	0.76	0.47	2.49	*
Dry unit weight (kN/m^3)								
Testing	−1.24	−2.00	2.00	*	1.74	0.49	2.27	*
Validation	−1.00	−2.00	2.00	*	0.95	0.47	2.49	*
Water content ω_o (%)								
Testing	0.59	−2.00	2.00	*	0.59	0.49	2.27	*
Validation	0.59	−2.00	2.00	*	0.53	0.47	2.49	*
Confining stress (kN/m^2)								
Testing	1.24	−2.00	2.00	*	1.44	0.49	2.27	*
Validation	0.26	−2.00	2.00	*	0.51	0.47	2.49	*
R_f								
Testing	0.51	−2.00	2.00	*	0.76	0.49	2.27	*
Validation	−1.59	−2.00	2.00	*	0.84	0.47	2.49	*

Key: * = Accept

transfer function). The simple linear mapping of the variables, extremes to the neural network's practical extremes, is adopted for scaling, as it is the most commonly used method. As part of this method, for each variable x with minimum and maximum values of $x_{(\max)}$ and $x_{(\min)}$, respectively, the scaled value x_i^{norm} is calculated as follows:

$$x_i^{\text{norm}} = \frac{x_{i\,(\text{actual})} - x_{i\,(\min)}}{x_{(\max)} - x_{(\min)}} \tag{8.10}$$

$$x_{i\,(\text{actual})} = x_i^{\text{norm}}\ (x_{\max} - x_{\min}) + x_{i\,(\min)} \tag{8.11}$$

8.2.5 *Model Architecture, Optimization, and Stopping Criteria*

One of the most important and difficult tasks in the development of ANN models is determining the model architecture (i.e., the number and connectivity of the hidden layer nodes). A network with one hidden layer can approximate any continuous function, provided that sufficient connection weights are used [18]. Consequently, one hidden layer is used in this study. The general strategy adopted for finding the optimal network architecture and internal parameters that control the training process is as follows: a number of trials are carried out using the default parameters of the software used with one hidden layer and 1, 2, 3, ..., 9 hidden layer nodes. It should be noted that 9 is the upper limit for the number of hidden layer nodes needed to map any continuous function for a network with (4) input parameters [19] and, consequently, is used in this work.

The network that performs best with respect to the testing set is retrained with different combinations of momentum terms, learning rates, and transfer functions in an attempt to improve model performance. Since the back-propagation algorithm uses a first-order gradient descent technique to adjust the connection weights, it may get trapped in a local minimum if the initial starting point in weight space is unfavorable. Consequently, the model that has the optimum momentum term, learning rate, and transfer function is retrained a number of times with different initial weights until no further improvement occurs. Using the default parameters of the software, a number of networks with different numbers of hidden layer nodes are developed and the results are shown graphically in Figs. 8.1 and 8.2 and are summarized in Tables 8.5 and 8.6 for ANN models named ($\log k - 1$ to $\log k - 9$), and ($R_f - 1$ to $R_f - 9$).

For model named ($\log k$), Fig. 8.1 shows that the lowest RMSE value is combined with a hidden layer network with (7) nodes. However, it is believed that, the network with (1), hidden layer nodes is considered optimal, to prediction error is not far from the network with (7) hidden layer nodes, the error difference being only (0.04), coupled with smaller number of connection weights. However, it is believed that the network with (1), hidden layer nodes is considered optimal, to predict error is

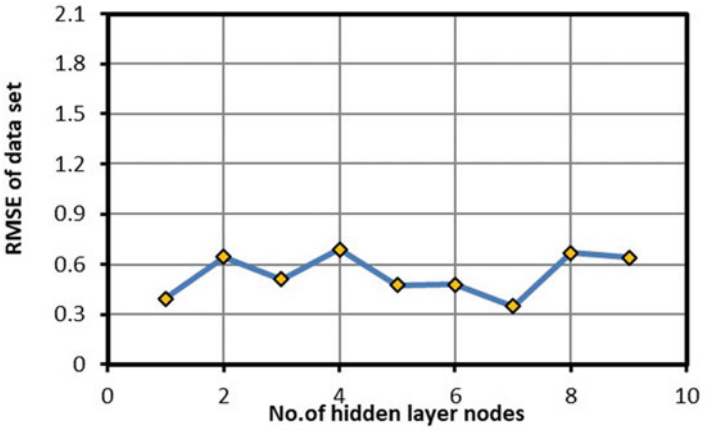


Fig. 8.1 Performance of the ANN models with different hidden layer nodes for testing (Learning rate = 0.2, Momentum term = 0.8), for log k

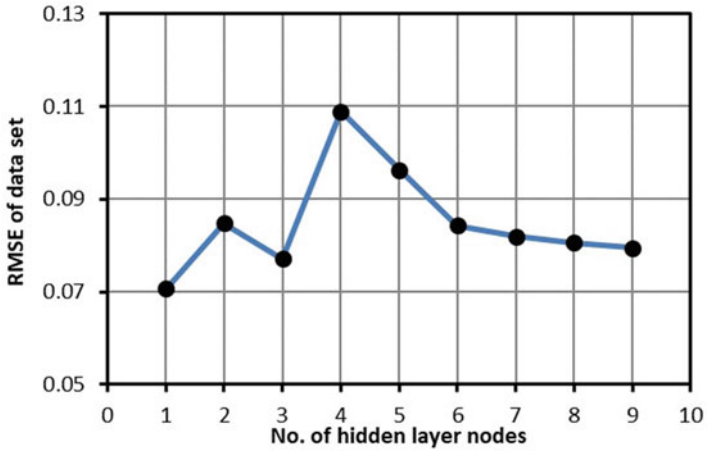


Fig. 8.2 Performance of the ANN models with different hidden layer nodes for testing (Learning rate = 0.2, Momentum term = 0.8) for R_f

not far from the network with (6) hidden layer nodes, the error difference being only (0.07), coupled with smaller number of connection weights. For models named (R_f), Fig. 8.2 shows that the network with (1), hidden layer node is considered optimal, which has the lowest RMSE value for testing data.

The effect of the internal parameters controlling the back-propagation (i.e. momentum term and learning rate) on model performance is investigated for the model with one hidden layer nodes for (log $k - 1$), and ($R_f - 1$), resulting in models log $k - 10$ to log $k - 18$ (Table 8.5), and $R_f - 10$ to $R_f - 18$ (Table 8.6). The effect of the momentum term on model performance is shown graphically in Figs. 8.3 and 8.4.

Table 8.5 Structure and performance of ANN models developed for modulus number (k)

Parameter effect	Model no.	No. of hidden nodes	Learning rate	Momentum term	Transfer function on hidden layer	Transfer function on output layer	Correlation coefficient			Performance measure					
										RMSE			MAE		
							T	S	V	T	S	V	T	S	V
	log $k-1$	1	0.2	0.8	Tanh	Sigmoid	0.3446	0.801	0.5425	0.59194	0.39457	0.60734	0.44512	0.31866	1.2673
	log $k-2$	2	0.2	0.8	Tanh	Sigmoid	0.43125	0.5485	0.6199	0.55797	0.64634	0.55044	0.43112	0.47981	0.4552
	log $k-3$	3	0.2	0.8	Tanh	Sigmoid	0.44756	0.5774	−0.3958	0.566	0.50925	0.73152	0.44971	0.39787	0.6001
Default parameter	log $k-4$	4	0.2	0.8	Tanh	Sigmoid	−0.3956	−0.728	−0.7466	0.62181	0.69023	0.7364	0.49274	0.52112	0.608
	log $k-5$	5	0.2	0.8	Tanh	Sigmoid	0.5392	0.6683	0.3397	0.51434	0.47428	0.70203	0.41745	0.39223	0.5794
	log $k-6$	6	0.2	0.8	Tanh	Sigmoid	0.44351	0.6536	0.5573	0.55754	0.4804	0.60921	0.4328	0.39683	0.4701
	log $k-7$	7	0.2	0.8	Tanh	Sigmoid	0.68774	0.8401	0.6598	0.43788	0.34885	0.51519	0.35207	0.28134	0.4267
	log $k-8$	8	0.2	0.8	Tanh	Sigmoid	−0.604	−0.7668	−0.3476	0.66121	0.67014	0.73866	0.55325	0.5207	0.6094
	log $k-9$	9	0.2	0.8	Tanh	Sigmoid	0.5206	0.5048	0.7157	0.53365	0.64088	0.64446	0.43515	0.47421	0.5265
	log $k-10$	1	0.2	0.01	Tanh	Sigmoid	0.53264	0.5217	0.5296	0.51656	0.66815	0.60238	0.41593	0.50233	1.2673
	log $k-11$	1	0.2	0.05	Tanh	Sigmoid	0.35576	0.6901	0.4498	0.60395	0.48728	0.63263	0.4746	0.41052	1.2673
Momentum term	log $k-12$	1	0.2	0.1	Tanh	Sigmoid	0.51879	0.5217	0.5755	0.52215	0.66815	0.57977	0.39553	0.50233	1.2673
	log $k-13$	1	0.2	0.15	Tanh	Sigmoid	0.55503	0.5778	0.687	0.51081	0.55307	0.50199	0.39933	0.42369	1.2673
	log $k-14$	1	0.2	0.2	Tanh	Sigmoid	0.51095	0.6901	0.6885	0.54779	0.48728	0.49528	0.42932	0.41052	1.2673
	log $k-15$	1	0.2	0.4	Tanh	Sigmoid	0.67805	0.8387	0.6497	0.44853	0.4142	0.5341	0.3801	0.3475	1.2673
	log $k-16$	1	0.2	0.6	Tanh	Sigmoid	0.66976	0.829	0.6515	0.45692	0.34098	0.52477	0.36215	0.26825	1.2673
	log $k-17$	1	0.2	0.9	Tanh	Sigmoid	0.67776	0.8382	0.6525	0.44539	0.39937	0.52972	0.37514	0.33591	1.2673
	log $k-18$	1	0.2	0.95	Tanh	Sigmoid	0.67723	0.8373	0.6587	0.44388	0.38889	0.52212	0.3704	0.32482	1.2673
	log $k-19$	1	0.05	0.8	Tanh	Sigmoid	0.37412	0.5176	0.4513	0.56778	0.65655	0.63105	0.43095	0.49219	1.2673

Learning rate	$\log k - 20$	1	0.02	0.8	Tanh	Sigmoid	0.55311	0.5194	0.5645	0.52085	0.65983	0.61488	0.42598	0.4953	1.2673
	$\log k - 21$	1	0.1	0.8	Tanh	Sigmoid	0.40605	0.5253	−0.3382	0.56498	0.66657	0.71783	0.43619	0.50108	1.2673
	$\log k - 22$	1	0.15	0.8	Tanh	Sigmoid	0.51127	0.5356	−0.3279	0.52951	0.66603	0.71809	0.41831	0.50066	1.2673
	$\log k - 23$	1	0.4	0.8	Tanh	Sigmoid	0.52122	0.5608	0.5936	0.5233	0.65078	0.55264	0.41132	0.48657	1.2673
	$\log k - 24$	1	0.6	0.8	Tanh	Sigmoid	0.72806	0.8714	0.7085	0.72806	0.27164	0.49731	0.33472	0.23979	1.2673
	$\log k - 25$	1	0.8	0.8	Tanh	Sigmoid	0.72975	0.875	0.6906	0.42168	0.35735	0.49427	0.35536	0.29934	1.2673
	$\log k - 26$	1	0.9	0.8	Tanh	Sigmoid	0.72931	0.8731	0.7029	0.41453	0.33479	0.48628	0.34947	0.28273	1.2673
	$\log k - 27$	1	0.95	0.8	Tanh	Sigmoid	0.72556	0.8668	0.7172	0.46237	0.27829	0.52904	0.35548	0.23211	1.2673
	$\log k - 28$	1	0.99	0.8	Tanh	Sigmoid	0.72985	0.8757	0.6774	0.43137	0.38057	0.50442	0.36329	0.31646	1.2673
Transfer function	$\log k - 29$	1	0.2	0.8	Sigmoid	Sigmoid	0.3564	0.5152	−0.355	0.57145	0.67461	0.71653	0.43975	0.5072	1.2673
	$\log k - 30$	1	0.2	0.8	Tanh	Tanh	0.44282	−0.4429	0.4684	0.58082	0.7606	0.65083	0.46713	0.55902	1.2673
	$\log k - 31$	1	0.2	0.8	Sigmoid	Tanh	0.16627	0.5196	−0.3744	0.62293	0.55318	1.26729	0.51422	0.43696	1.2673

Table 8.6 Structure and performance of ANN models developed for failure ratio (R_f)

Parameter effect	Model no.	No. of hidden nodes	Learning rate	Momentum term	Transfer function on hidden layer	Transfer function on output layer	Correlation coefficient			Performance measure					
										RMSE			MAE		
							T	*S*	*V*	*T*	*S*	*V*	*T*	*S*	*V*
	$R_f - 1$	1	0.2	0.8	Tanh	Sigmoid	0.74465	0.75976	0.75062	0.06851	0.07064	0.07831	0.05589	0.05638	0.46824
	$R_f - 2$	2	0.2	0.8	Tanh	Sigmoid	0.53652	0.64544	0.71995	0.08619	0.08478	0.0874	0.06873	0.06841	0.07305
	$R_f - 3$	3	0.2	0.8	Tanh	Sigmoid	0.40025	0.75654	0.67752	0.09285	0.07718	0.09098	0.07371	0.06511	0.07619
Default parameter	$R_f - 4$	4	0.2	0.8	Tanh	Sigmoid	0.49753	0.63797	0.66517	0.1001	0.10896	0.11114	0.08414	0.09561	0.10177
	$R_f - 5$	5	0.2	0.8	Tanh	Sigmoid	0.53582	0.63887	0.7054	0.10067	0.09626	0.10647	0.0841	0.0833	0.09755
	$R_f - 6$	6	0.2	0.8	Tanh	Sigmoid	0.52878	0.66448	0.69244	0.09011	0.08435	0.09653	0.07273	0.06805	0.08391
	$R_f - 7$	7	0.2	0.8	Tanh	Sigmoid	0.53018	0.74186	0.67044	0.09633	0.08202	0.10457	0.07974	0.07572	0.09436
	$R_f - 8$	8	0.2	0.8	Tanh	Sigmoid	0.54393	0.64941	0.67035	0.08315	0.08061	0.08265	0.06219	0.06339	0.06234
	$R_f - 9$	9	0.2	0.8	Tanh	Sigmoid	0.52098	0.73345	0.72027	0.08838	0.07948	0.08992	0.07137	0.06595	0.07806
	$R_f - 10$	1	0.2	0.01	Tanh	Sigmoid	0.52891	0.63856	0.75061	0.0849	0.08178	0.07933	0.06536	0.0644	0.46824
	$R_f - 11$	1	0.2	0.05	Tanh	Sigmoid	0.46291	0.71268	0.75299	0.08785	0.07575	0.07413	0.06623	0.06406	0.05674
Momentum term	$R_f - 12$	1	0.2	0.1	Tanh	Sigmoid	0.5875	0.66171	0.75306	0.08104	0.07971	0.07384	0.06387	0.06129	0.05631
	$R_f - 13$	1	0.2	0.15	Tanh	Sigmoid	0.54072	0.74058	0.75084	0.08374	0.07328	0.07832	0.06646	0.0609	0.06297
	$R_f - 14$	1	0.2	0.2	Tanh	Sigmoid	0.54189	0.71112	0.74911	0.08387	0.07548	0.07788	0.06357	0.06396	0.06306
	$R_f - 15$	1	0.2	0.4	Tanh	Sigmoid	0.45625	0.6482	0.75037	0.08878	0.08283	0.0793	0.06838	0.0654	0.06463
	$R_f - 16$	1	0.2	0.6	Tanh	Sigmoid	0.56723	0.74209	0.75115	0.0818	0.07246	0.07725	0.06376	0.05951	0.06151
	$R_f - 17$	1	0.2	0.9	Tanh	Sigmoid	0.5665	0.62056	0.74852	0.08251	0.08346	0.06697	0.06186	0.06437	0.0483
	$R_f - 18$	1	0.2	0.95	Tanh	Sigmoid	0.00338	0.74737	0.74805	0.09318	0.06903	0.06542	0.06621	0.054	0.04612
	$R_f - 19$	1	0.005	0.8	Tanh	Sigmoid	0.56648	0.6185	0.74966	0.08124	0.08315	0.07431	0.06263	0.06761	0.46824
	$R_f - 20$	1	0.02	0.8	Tanh	Sigmoid	0.59172	0.61374	0.74979	0.08027	0.08448	0.07522	0.06309	0.07036	0.05845

	$R_f - 21$	1	0.1	0.8	Tanh	Sigmoid	0.56682	0.61803	0.75007	0.0819	0.08365	0.07612	0.0638	0.0682	0.05981
	$R_f - 22$	1	0.15	0.8	Tanh	Sigmoid	0.56105	0.61661	0.74679	0.08395	0.08626	0.08004	0.06691	0.07048	0.06428
Learning rate	$R_f - 23$	1	0.4	0.8	Tanh	Sigmoid	0.43437	0.62317	0.75002	0.09045	0.08449	0.07157	0.0689	0.06961	0.05347
	$R_f - 24$	1	0.6	0.8	Tanh	Sigmoid	0.5632	0.61772	0.7539	0.09098	0.09261	0.09477	0.07521	0.0775	0.08439
	$R_f - 25$	1	0.8	0.8	Tanh	Sigmoid	0.51593	0.73278	0.74668	0.09327	0.08793	0.09332	0.07613	0.07377	0.08281
	$R_f - 26$	1	0.9	0.8	Tanh	Sigmoid	0.52182	0.61688	0.75197	0.08641	0.08933	0.07299	0.06239	0.06848	0.05383
	$R_f - 27$	1	0.95	0.8	Tanh	Sigmoid	0.51539	0.65225	0.74036	0.08746	0.08029	0.07483	0.06741	0.06303	0.05714
	$R_f - 28$	1	0.99	0.8	Tanh	Sigmoid	0.51485	0.75316	0.72628	0.09636	0.07513	0.0774	0.0692	0.05535	0.05228
Transfer function	$R_f - 29$	1	0.2	0.8	Sigmoid	Sigmoid	0.40547	0.62952	0.71492	0.09077	0.08308	0.08118	0.06968	0.06693	0.46824
	$R_f - 30$	1	0.2	0.8	Tanh	Tanh	0.53842	0.58136	0.70835	0.08617	0.09055	0.08161	0.06107	0.06971	0.46824
	$R_f - 31$	1	0.2	0.8	Sigmoid	Tanh	0.40585	0.7419	0.715	0.10478	0.07521	0.07158	0.07556	0.05509	0.46824

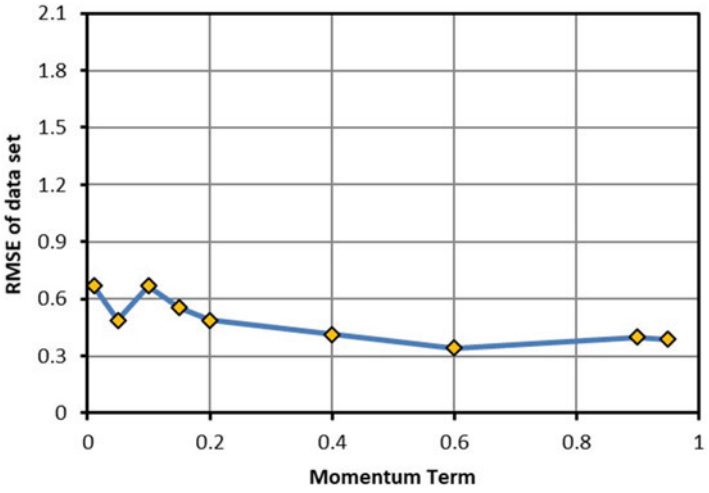


Fig. 8.3 Effect of various momentum terms on ANN performance for testing (Hidden nodes = 1 and learning rate = 0.2), for log k

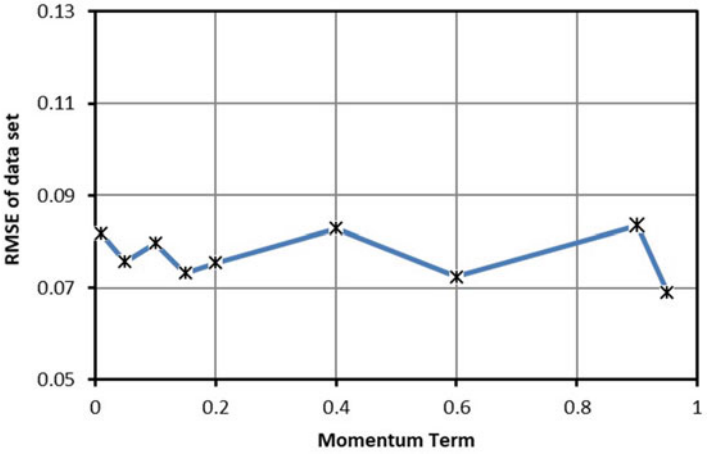


Fig. 8.4 Effect of various momentum terms on ANN performance for testing (Hidden nodes = 1 and learning rate = 0.2), for R_f

It can be seen that the performance of the ANN model is relatively insensitive to momentum terms, particularly in the range (0.6–0.98) for log k. Figures 8.5 and 8.6 show that the effect of different learning rates on the models, as shown below, is relatively sensitive to learning rates in the range (0.4–0.99) for log k, and (0.1–0.96) for R_f, while the performance will be insensitive to learning rates, for log k in the range (0.0–0.4). Thus, the optimum values for momentum term and learning rate used are 0.80 and 0.20, respectively. The effect of using different transfer functions

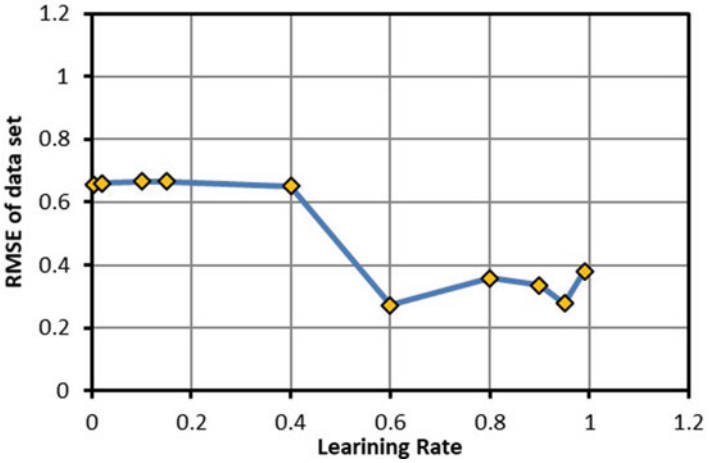


Fig. 8.5 Effect of various learning rates on ANN performance for testing (Hidden nodes = 1 and Momentum term = 0.8), for log k

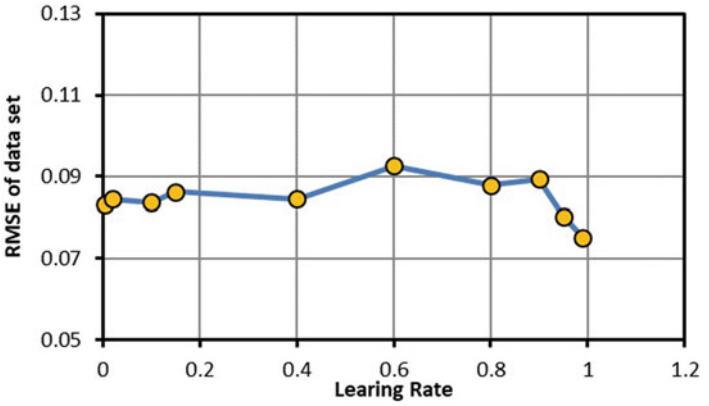


Fig. 8.6 Effect of various learning rates on ANN performance for testing (Hidden nodes = 1 and Momentum term = 0.8), for R_f

is shown in Tables 8.5 and 8.6 (models 29–31). It can be seen that the performance of ANN models is relatively insensitive to transfer functions although a slightly better performance is obtained when the hyperbolic tangent (tanh) transfer function is used for the hidden layer, and the sigmoid transfer function is used for the output layer.

Ismail and Jeng [20] reported similar observations for the case of predicting settlement of shallow foundations on cohesionless soils by artificial neural networks. Furthermore, Al-Janabi [21] also stressed similar observations for the case of predicting dissolved gypsum and leaching strain during leaching process by artificial

neural networks and prediction of ultimate bearing capacity of shallow foundation on cohesionless soils by using back-propagation neural networks.

8.2.6 *Parametric Study*

In order to confirm the generalization ability and robustness of models, logarithm of modulus number, modulus exponent, and failure ratio, (log k − 1 and R_f − 1), an additional validation approach is proposed. The approach includes carrying out a parametric study in which the response of the ANN model output to changes in its inputs is investigated. The first three input variables (i.e., plasticity index, dry unit weight, and water content), except one (i.e., confining stress), are fixed to their mean values used for training and a set of synthetic data, between the minimum and maximum values used for model training, and are generated for the input that is not set to a fixed value at different intervals. The synthetic data are generated by increasing their values in increments equal to 25% of the total range between the minimum and maximum values. The response of the model is examined. This process is repeated using another input variable and so on until the model response is tested for all input variables. The robustness of the model can be determined by examining how well the predicted logarithm of modulus number, modulus exponent, and failure ratio are in agreement with the known underlying physical process over a range of inputs.

The above approach is applied to models (log k − 1, and R_f − 1) and the results are shown in Figs. 8.7, 8.8, and 8.9. It can be seen that the behavior of models (log k − 1, and R_f − 1) is expected, which indicates that the models may be considered to be robust. For example, the figures show that the logarithm

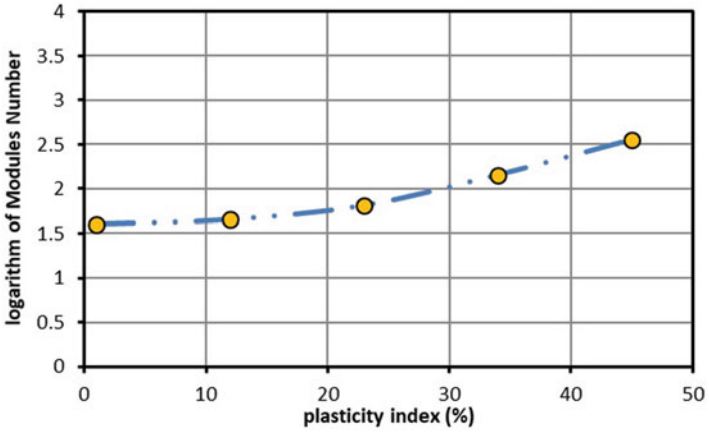


Fig. 8.7 Result of parametric study of log k model under different plasticity indices

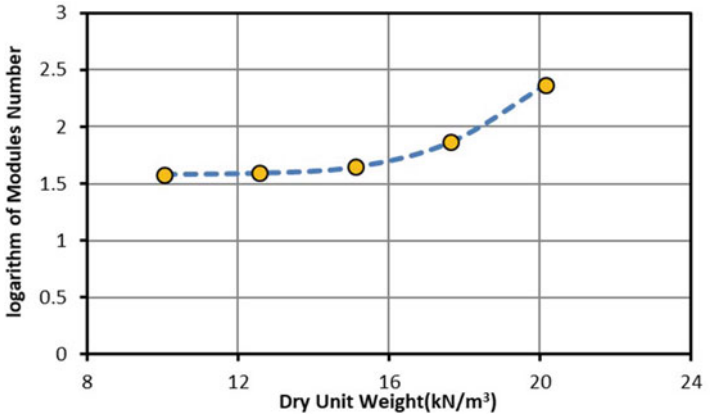


Fig. 8.8 Result of parametric study of log k model under different dry unit weights

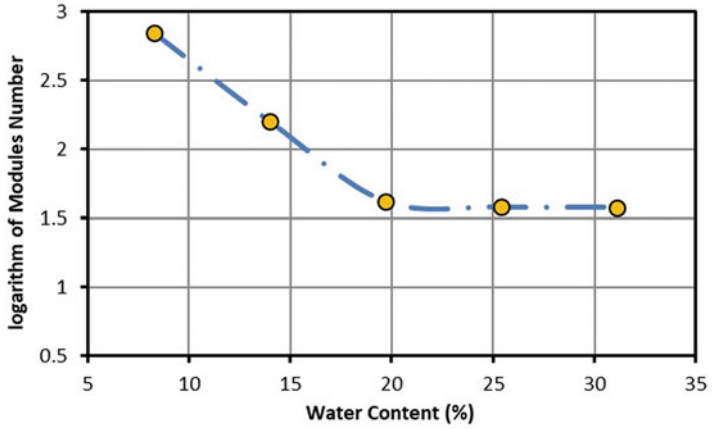


Fig. 8.9 Result of parametric study of log k model under different water contents

of modulus number increases as plasticity index is increased. Also, there is an increase in the logarithm of modulus number as dry unit weight increases: see Figs. 8.8 and 8.12, respectively. On the other hand, in Fig. 8.9, the logarithm of modulus number decreases as the water content is increased in range to (20)% after which it remains constant. Figure 8.10 shows the relationship between the logarithm of modulus number and the confining stress. It can be noticed that the logarithm of modulus number remarkably decreases as the confining stress is increased.

Figures 8.11 and 8.12 show that the variation of failure ratio (R_f) with the increase in plasticity index and a decrease in dry unit weight respectively is too small, while the failure ratio decreases with the increase in water content as shown in Fig. 8.13. On the other hand, Fig. 8.14 shows that the failure ratio (R_f) increases with increase

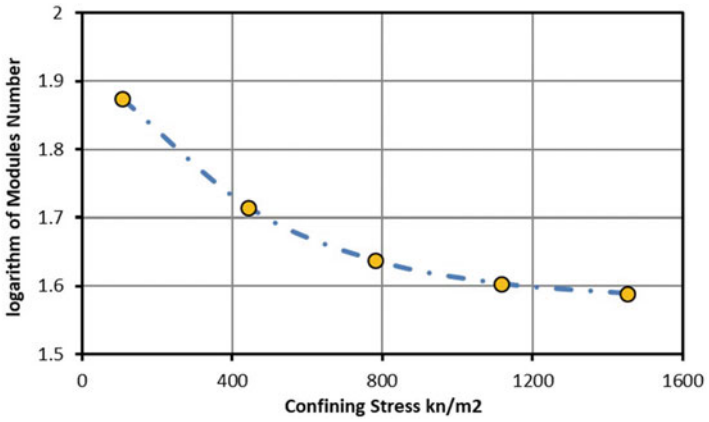


Fig. 8.10 Result of parametric study of log k model under different confining stresses

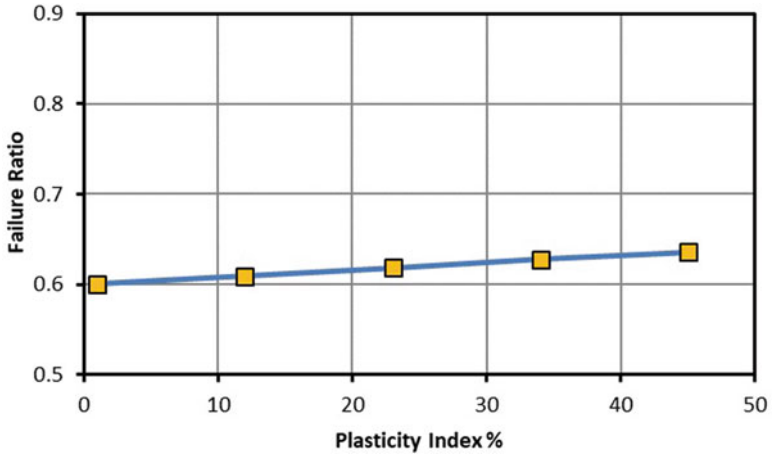


Fig. 8.11 Result of parametric study of (R_f) model under different plasticity indices

in the confining stress. Note: The preceding observations are logical and agree with physical behavior of soil sample.

8.2.7 Sensitivity Analysis of the ANN Model Inputs

In an attempt to identify which of the input variables has the most significant impact on the logarithm of modulus number, modulus exponent, and failure ratio (log k and R_f), a sensitivity analysis is carried out on the ANN models (log k and R_f). A simple and innovative technique proposed by Garson [22] is used to interpret the relative

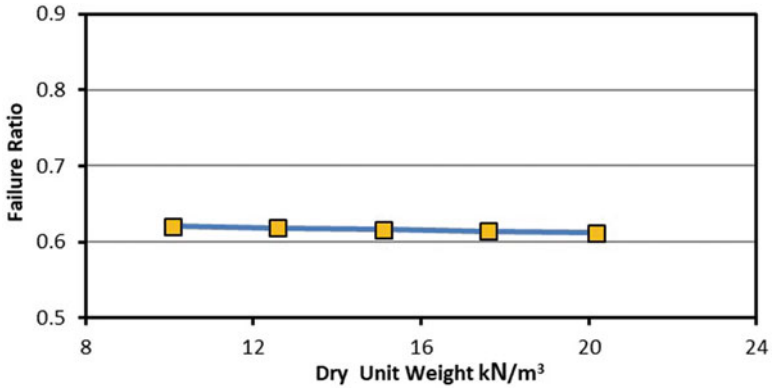


Fig. 8.12 Result of parametric study of (R_f) model under different dry unit weights, (kN/m^3)

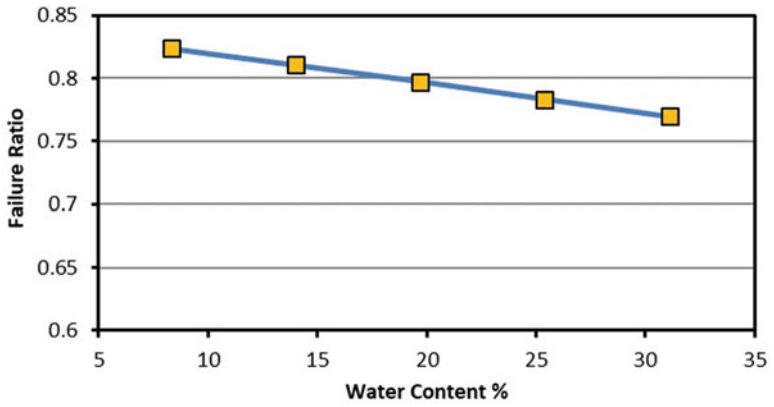


Fig. 8.13 Result of parametric study of (R_f) model under different water contents, (ω_o)%

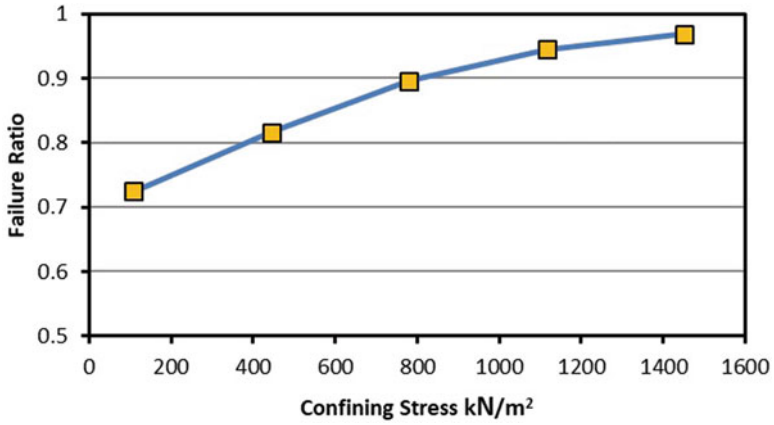


Fig. 8.14 Result of parametric study of (R_f) model under different confining stresses, (kN/m^2)

Table 8.7 The optimum connection weight for model log *k*

	Connection weights				
Hidden nodes	Plasticity index (PI) (%)	Dry unit weight (kN/m^3)	Water content ω_o (%)	Confining stress (kN/m^2)	log *k*
Hidden-1	2.425	3.139	−6.346	−1.8	1

Table 8.8 The absolute value of the connection weight for each model input parameter

	Plasticity index (PI) (%)	Dry unit weight (kN/m^3)	Water content ω_o (%)	Confining stress (kN/m^2)
Hidden-1	2.425	3.139	6.346	1.8

Table 8.9 The optimum connection weight for each model input parameter divided by the sum of all input parameter

	Plasticity index (PI) (%)	Dry unit weight (kN/m^3)	Water content ω_o (%)	Confining stress (kN/m^2)
Hidden-1	0.176	0.229	0.462	0.131

importance of the input variables by examining the connection weights of the trained network. For a network with one hidden layer, the technique involves a process of partitioning the hidden output connection weights into components associated with each input node. For models (log *k* and R_f), the method is illustrated as follows. The models have four input nodes and one output node for each model with connection weight, as follows (Table 8.7).

The computational process proposed by Garson [22] is as follows:

1. For each hidden node *I*, get the products p_{ij} (where *j* represents the column number of the weight mentioned above) by multiplying the absolute value of the hidden-output layer connection weight by the absolute value of the hidden-input layer connection weight of each input variable *j*. As an example (Table 8.8):

$$P_{11} = 2.425 \times 1.00 = 2.425$$

2. For each hidden node *I*, divide p_{ij} by the sum of all input variables to obtain Q_{ij}. As an example (Table 8.9):

$$Q_{11} = 2.425/\left(2.425 + 3.139 + 6.346 + 1.8\right)$$

$$Q_{11} = 0.176$$

3. Divide Q_{ij} by the sum for all input variables to get the relative importance of all output weights attributed to the given input variable.
4. As an example, the relative importance (see Tables 8.10, 8.11 and 8.12, respectively) for input node 1 is equal to:

$$(0.176 \times 100) / (0.176 + 0.229 + 0.462 + 0.131) = 17.6\%$$

Table 8.10 Sensitive analyses of input variables for model of (log k)

For log $k - 1$	Plasticity index (PI) (%)	Dry unit weight (kN/m^3)	Water content ω_o (%)	Confining stress (kN/m^2)
Relative importance (%)	17.6	22.9	46.2	13.1

Table 8.11 Sensitive analyses of input variables for model of (n)

For $n - 1$	Plasticity index (PI) (%)	Dry unit weight (kN/m^3)	Water content ω_o (%)	Confining stress (kN/m^2)
Relative importance (%)	9.586	48	24.37	18

Table 8.12 Sensitive analyses of input variables for model of (R_f)

For $R_f - 1$	Plasticity index (PI) (%)	Dry unit weight (kN/m^3)	Water content ω_o (%)	Confining stress (kN/m^2)
Relative importance (%)	40	9.1	7	43.7

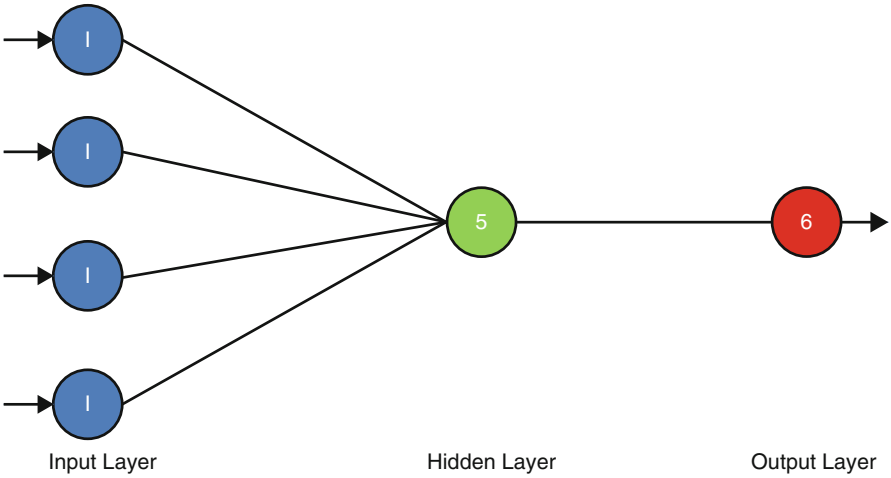


Fig. 8.15 Structure of the ANN optimal model (log $k - 1$)

8.3 ANN Model Equations

8.3.1 ANN Model Equation for log k

The small number of connection weights obtained for the optimal ANN model (log $k - 1$) enables the network to be translated into relatively simple formula. To demonstrate this, the structure of the ANN model is shown in Fig. 8.15, while connection weights and threshold levels are summarized in Table 8.13.

Table 8.13 Weights and threshold levels for the ANN optimal model (log $k-1$)

Hidden layer	w_{ji} (weight from node I in the input layer to nodes j in the hidden layer)				Hidden layer
Nodes	$I=1$	$I=2$	$I=3$	$I=4$	Threshold θ_j
$J=5$	2.42	3.139	−6.346	−1.8	−0.984
Output					Output
Layer					Layer
Nodes	$I=5$	–	–	–	Threshold θ_j
$J=6$	1.00				−0.411

Using the connection weight and the threshold levels shown in Table 8.13, the predicted logarithm of modulus number (log k) can be expressed as follows:

$$\log k = \frac{1}{1+\mathrm{e}^{(0.411-1\times\tanh\ x)}} \tag{8.12}$$

In which

$$X = \theta_5 + w51 \times I_1 + w52 \times I_2 + w53 \times I_3 + w54 \times I_4 \tag{8.13}$$

$$X = \theta_5 + w51 \times \mathrm{PI} + w52 \times \gamma_{\mathrm{dry}} + w53 \times \omega_{\mathrm{o}} + w54 \times \sigma_3 \tag{8.14}$$

$$\{X\} = \{\theta\} + [W]\{I\} \tag{8.15}$$

where

I_1 = Plasticity index %; I_2 = Dry unit weight (kN/m^3); I_3 = Water content % and I_4= Confining stress (kN/m^2).

It should be noted that before using Eq. (8.11), all input variables need to be scaled between 0.0 and 1.0 using Eq. (8.9), and all data range in the ANN model, see Table 8.1. It should also be noted that the predicted value of (log k) obtained from Eq. (8.9) is scaled between 0.0 and 1.0 and in order to obtain the actual value of log k has be re-scaled using Eq. (8.10) and the data range in Tables 8.5 and 8.6. The procedure for scaling and the substituting the values of the weights and threshold levels from Table 8.10, Eq. (8.12), and Eq. (8.13) can be rewritten as follows:

$$\log k = \frac{2.949}{1+\mathrm{e}^{(0.411-1\times\tanh\ x)}} + 1 \tag{8.16}$$

And

$$X = 10^{-3}\left(55 \times \mathrm{PI} + 310 \times \gamma_{\mathrm{dry}} - 278 \times \omega_{\mathrm{o}} - 1.3 \times \sigma_3\right) - 1.708 \tag{8.17}$$

A numerical example is provided to better explain the implementation of modulus number formula. The equation is tested against different types of the result by Wong and Duncan [23], which are given below:

Plasticity index % = 1
Dry unit weight (kN/m^3) = 16.65
Water content ω_{o} % = 11.6
Confining stress (kN/m^2) = 349.648
From Eq. (8.16). $x = -0.1708$
From Eq. (8.19)

$$\log k = \frac{2.949}{1 + \mathrm{e}^{(0.411 - 1 \times \tanh\ x)}} + 1 = 2.05 \tag{8.18}$$

The predicted value is compared well with measured value ($\log\ k = 2.04$). Appendix 2, case No. (45).

8.3.2 *ANN Model Equation for R_f*

The small number of connection weights obtained for the optimal ANN model ($R_{\mathrm{f}} - 1$) enables the network to be translated into relatively simple formula. To demonstrate this, the structure of the ANN model is shown in Fig. 8.16, while connection weights and threshold levels are summarized in Table 8.14.

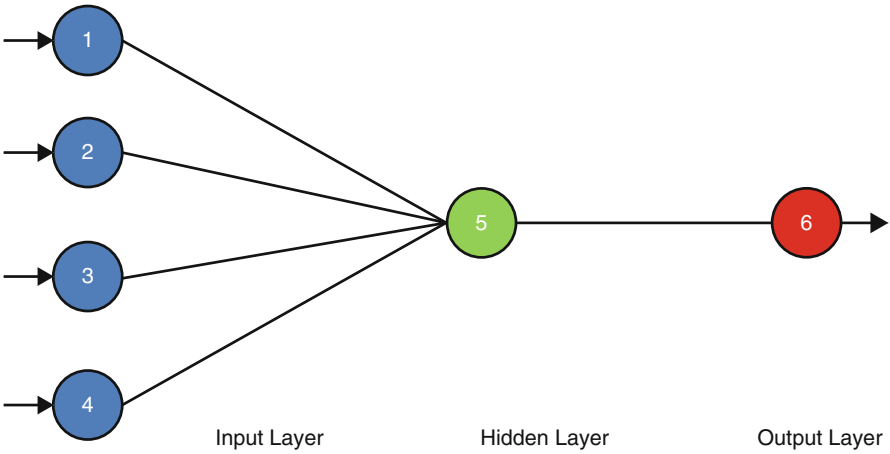


Fig. 8.16 Structure of the ANN optimal model ($R_{\mathrm{f}} - 1$)

Table 8.14 Weights and threshold levels for the ANN optimal model, ($R_f - 1$)

Hidden layer	w_{ji} (weight from node I in the input layer to nodes j in the hidden layer)				Hidden layer
Nodes	$I = 1$	$I = 2$	$I = 3$	$I = 4$	Threshold θ_j
$J = 5$	0.883	−0.2	0.153	−0.96	0.033
Output					Output
Layer					Layer
Nodes	$I = 5$	–	–	–	Threshold θ_j
$J = 6$	3.45-				0.753

Using the connection weight and the threshold levels shown in Table 8.14, the prediction of failure ratio (R_f) can be expressed as follows:

$$R_f = \frac{1}{1 + e^{(-0.753 + 3.45 \times \tanh\ x)}} \tag{8.19}$$

in which

$$X = \theta_5 + w51 \times I_1 + w52 \times I_2 + w53 \times I_3 + w54 \times I_4 \tag{8.20}$$

$$X = \theta_5 + w51 \times \mathrm{PI} + w52 \times \gamma_{dry} + w53 \times \omega_o + w54 \times \sigma_3 \tag{8.21}$$

$$\{X\} = \{\theta\} + [W]\{I\} \tag{8.22}$$

It should be noted that before using Eq. (8.17), all input variables need to be scaled between 0.0 and 1.0 using Eq. (8.9) and all data range in the ANN model, see Table 8.14. It should also be noted that the predicted value of (R_f), obtained from Eq. (8.14), is scaled between 0.0 and 1.0 and in order to smooth the data get high generalization ability and to obtain the actual value of (R_f) and has been re-scaled using Eq. (8.10) and the data range in Table 8.2 [5, 24, 25]. Equations (8.14) and (8.15), can be re-written as follows:

$$R_f = \frac{1}{1 + e^{(-0.753 + 3.45 \times \tanh\ x)}} + 0.52 \tag{8.23}$$

And

$$X = 10^{-3}\left(3.5 \times \mathrm{PI} - 19.7 \times \gamma_{dry} + 6 \times \omega_o - 0.7 \times \sigma_3\right) + 0.236 \tag{8.24}$$

A numerical example is provided to better explain the implementation of failure ratio formula. The equation is tested against different types of the result by Wong and Duncan [23], which are given below:

Plasticity index % = 19
Dry unit weight (kN/m^3) = 17
Water content ω_o % = 16.2
Confining stress (kN/m^2) = 946.74
From Eq. (8.22). $x = -0.276$
From Eq. (8.23)

$$R_f = \frac{0.48}{1 + e^{(-0.753+3.45\times\tanh(-0.276))}} + 0.52 = 0.924$$

The predicted value is compared well with the measured value ($R_f = 0.92$). Appendix 2, case No. 14.

8.4 Validity of the ANN Models Equation

To assess the validity of the derived equations for the logarithm of modulus number, modulus exponent, and failure ratio (log k, n and R_f) models, the equations are used to predict these values on the basis of all, training, and validation data sets used. The predicted values of the (log k and R_f), are plotted against the measured (observed) values, (log k and R_f), in Figs. 8.17 and 8.18, respectively for the three data sets.

8.5 Comparison Between Measured and Predicted Stress–Strain Relationship

In this section, the reliability and the performance of the proposed ANNs method is further examined graphically using unseen experimental dataset that has not been involved in the process of learning. Figures 8.19 and 8.20 depict the comparison between the predicted nonlinear soil stress–strain parameters and compared with experimental results to determine the nonlinear soil stress–strain parameters conducted by Boscardin et al. [26]. Based on hyperbolic parameters under different factors (plasticity index, dry unit weight, water content, and confining stress), as shown below. ANNs, models equations developed to find hyperbolic parameters, logarithm of modulus number (log k), and failure ratio (R_f). According to the results, significant agreement can be observed between the measured versus predicted values with low scatter around equality line, which confirms that the trained network has the ability to successfully reproduce the results of the experimental soil stress–strain parameters with high consistency.

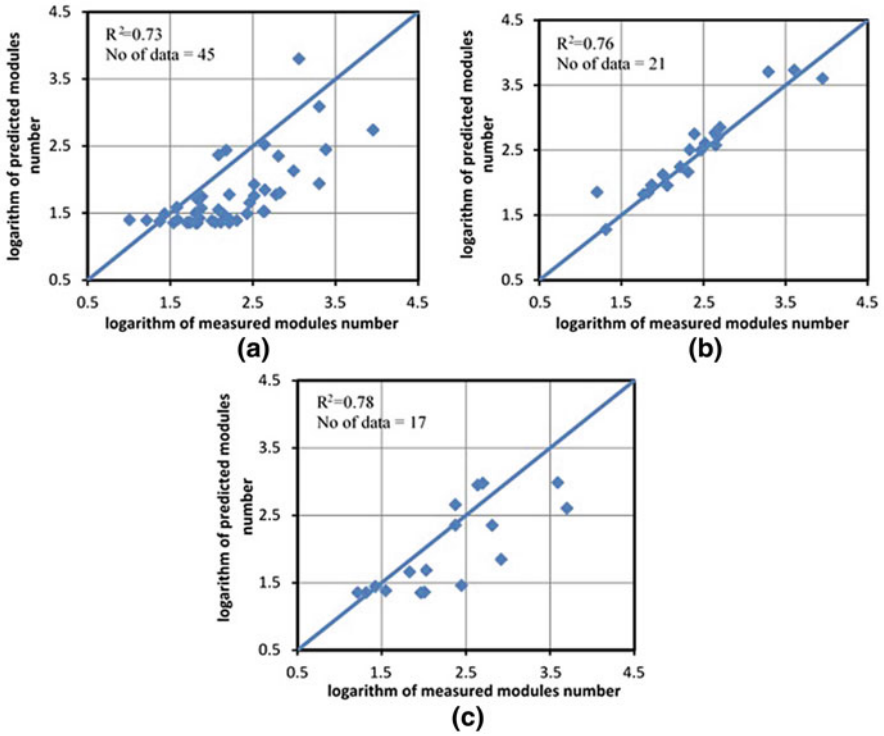


Fig. 8.17 Comparison of predicted and measured logarithm of modulus number for (**a**) training data set, (**b**) testing data set, (**c**) validation data set

8.6 Concluding Remarks

The analyses carried out in this study have given rise to the following results and conclusions:

1. Several factors affect the hyperbolic stress–strain relationship parameters, such as plasticity index (PI), dry unit weight (γ_d), water content (ω_o), and confining stress.
2. ANNs have the ability to predict the hyperbolic stress–strain relationship parameters, modulus number (log k), modulus exponent (n), and failure ratio (R_f), with a good degree of accuracy within the range of data used for developing ANN models.

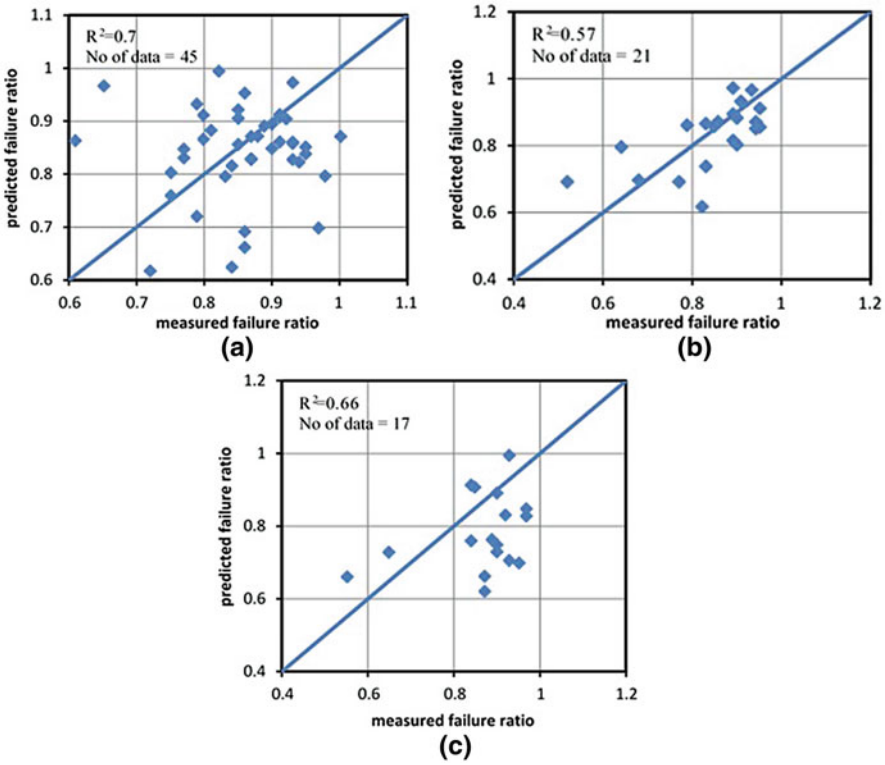


Fig. 8.18 Comparison of predicted and measured failure ratios for (**a**) training data set, (**b**) testing data set, (**c**) validation data set

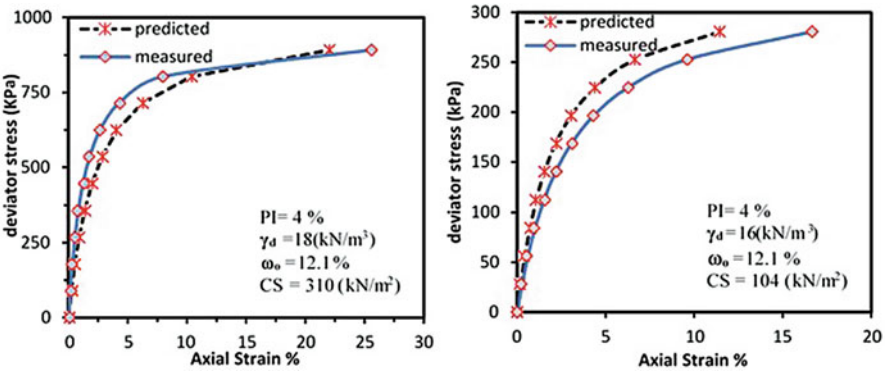


Fig. 8.19 Comparison between measured and predicted stress–strain relationship for (ML) soil

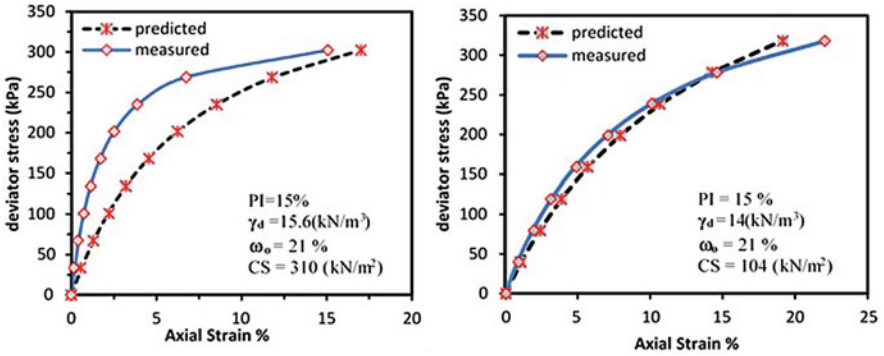


Fig. 8.20 Comparison between measured and predicted stress–strain relationship for (CL) soil

3. Study the impact of the internal network parameters on model performance, which indicates that

   ANN performance is sensitive to the number of hidden layer nodes, momentum terms, learning rate, and transfer functions.
4. The results of parametric study carried out in this work show that the ANN model developed. This indicates that the model could reliably be considered to be robust.
5. The sensitivity analysis indicated the following:

   Logarithm of Modulus Number (*log k*): The results indicate that the water content has the most significant effect on the prediction of the logarithm of modulus number followed by dry unit weight with a relative importance of 46.2% and 22.9%, respectively. The results also indicate that plasticity index, and confining stress, have relative importance (17.6%, 13.1%) respectively.

   Failure Ratio (R_f): The results of sensitivity analysis indicate that the confining stress has the most significant effect on the predicted failure ratio, followed by plasticity index with a relative importance 43.7% and 40%, respectively. The results also indicate that dry unit weight and water content have moderate impact on the failure ratio with a relative importance equal to 9.1% and 7%, respectively.

Acknowledgments The authors would like to acknowledge the Iraqi Ministry of Higher Education and Scientific Research and Wasit University for the grant provided to carry out this research under the grant agreement number 162575, dated 28/05/2013, with the Liverpool John Moores University, university reference number (744221).

A.1 Appendix 1

Case no.	Reference	Unified soil classif.	Cohesion (kN/m^2)	Friction angle	PI (%)	Dry unit weight γ_d (kN/m^3)	Water content ω_c (%)	Confining stress (kN/m^2)	log k	n	R_f
1	Wong and Duncan [23]	ML	193.68	19	4	17.4	15.6	860.672	2.301	0.59	0.86
2	Wong and Duncan [23]	ML	41.964	30	4	17.5	12.7	860.672	1.4314	1.43	0.72
3	Wong and Duncan [23]	ML	45.192	31	1	16.65	11.6	349.648	2.3802	0.31	0.83
4	Wong and Duncan [23]	ML	20.444	31	1	16.65	13.6	403.44	2.4314	0.38	0.82
5	Wong and Duncan [23]	ML	58.104	27	1	16.65	16.6	403.44	2	0.84	0.77
6	Wong and Duncan [23]	CL	57.028	29	20	17.4	16.7	715.433	2.415	0.6	0.87
7	Wong and Duncan [23]	CL	129.12	14	20	17.13	19.5	494.886	1.5911	0.48	0.58
8	Wong and Duncan [23]	CL	102.22	0	23	17.15	19.1	193.651	1.8195	0	0.75
9	Wong and Duncan [23]	CL	45.192	0	23	16.65	21.2	193.651	1	0.03	0.52
10	Wong and Duncan [23]	CL	107.6	31	22	16.33	21.7	193.651	1.5563	0	0.57
11	Wong and Duncan [23]	CL	98.992	17	16	16.87	11.5	215.168	2.8129	−0.68	0.9
12	Wong and Duncan [23]	CL	161.4	6	16	17.46	14.3	376.544	2.8261	−0.14	0.93
13	Wong and Duncan [23]	CL	139.88	24	16	17.45	16.8	376.544	2.6335	0.1	0.93
14	Wong and Duncan [23]	CL	193.68	13	16	18	11.5	376.544	3.3802	−0.74	0.92
15	Wong and Duncan [23]	CL	204.44	32	16	18.36	14.5	376.544	3.301	−0.3	0.97
16	Wong and Duncan [23]	CL	161.4	18	16	17.41	8.71	376.544	3.9494	−1.1	0.94
17	Wong and Duncan [23]	CL	139.88	29	16	19.1	11.7	376.544	3.699	−0.28	0.95
18	Wong and Duncan [23]	CL	68.864	25	15	16.81	12.5	403.44	2.5051	−0.21	0.8
19	Wong and Duncan [23]	CL	53.8	2	15	16.81	14.5	349.648	2.2788	0.02	0.81
20	Wong and Duncan [23]	CL	107.6	1	30	17.13	17.2	349.648	1.8692	0.23	0.87

(continued)

Case no.	Reference	Unified soil classif.	Cohesion (kN/m^2)	Friction angle	PI (%)	Dry unit weight γ_d (kN/m^3)	Water content ω_c (%)	Confining stress (kN/m^2)	log k	n	R_f
21	Wong and Duncan [23]	CL	107.6	1	30	16.36	17	349.648	1.8325	−0.05	0.84
22	Wong and Duncan [23]	CL	48.42	25	30	16.42	20	349.648	1.4314	0.18	0.85
23	Wong and Duncan [23]	CL	61.332	4	16	17.33	14.6	349.648	2.5051	0.29	0.85
24	Wong and Duncan [23]	CL	161.4	3	32	15.54	23.2	349.648	2.301	0.29	0.89
25	Wong and Duncan [23]	CL	129.12	1	32	14.68	23.3	403.44	2	0.18	0.86
26	Wong and Duncan [23]	CL	68.864	22	32	14.53	26.7	349.648	1.7243	0.14	0.9
27	Wong and Duncan [23]	CL	90.384	22	16	17.9	15.1	349.648	2.2041	0.34	0.79
28	Wong and Duncan [23]	CL	59.18	28	16	17	15	349.648	2.4624	0.27	0.91
29	Wong and Duncan [23]	CL	83.928	25	12	16	13.5	349.648	2.8325	−0.36	0.84
30	Wong and Duncan [23]	CL	161.4	6	12	17	13.3	349.648	2.7782	0.18	0.68
31	Wong and Duncan [23]	CL	79.624	18	12	16.42	19.3	349.648	1.3617	0.32	0.61
32	Wong and Duncan [23]	CL	97.916	20	12	17	16.7	349.648	2.4472	0.6	0.93
33	Wong and Duncan [23]	CL	71.016	8	12	16.25	16.3	349.648	2.3424	0.23	0.9
34	Wong and Duncan [23]	CL	139.88	13	25	16.8	18.6	349.648	2.1461	0.2	0.84
35	Wong and Duncan [23]	CL	107.6	2	25	16.31	17.1	349.648	2.0792	0.09	0.83
36	Wong and Duncan [23]	CL	86.08	24	25	16.5	19.7	349.648	1.6721	0.33	0.82
37	Wong and Duncan [23]	CL	161.4	8	25	17	13.9	349.648	2.9777	−0.15	0.9
38	Wong and Duncan [23]	CL	161.4	4	25	17.33	16.9	349.648	2.6721	0	0.95
39	Wong and Duncan [23]	CL	72.092	23	23	15.8	20.8	349.648	1.8751	0.44	0.88
40	Wong and Duncan [23]	CL	193.68	12	23	16.8	14.8	349.648	2.9243	−0.19	0.84
41	Wong and Duncan [23]	CL	129.12	29	23	16.2	17.4	349.648	2.4314	0.6	0.87
42	Wong and Duncan [23]	CL	150.64	13	23	16	14.2	349.648	3.0414	−0.36	0.83
43	Wong and Duncan [23]	CL	150.64	2	23	16.71	17.5	349.648	2.6128	0.15	0.87
44	Wong and Duncan [23]	CL	82.852	1	27	15.68	24	322.752	1.7559	0.43	0.86
45	Wong and Duncan [23]	CL	104.372	2	18	15.96	22.9	215.168	2.0414	0.43	0.9

46	Wong and Duncan [23]	CL	118.36	1	20	15.86	22.7	430.336	2	0.27	0.89
47	Wong and Duncan [23]	CL	106.524	3	24	15.7	23.9	430.336	2.2041	0.54	0.97
48	Wong and Duncan [23]	CL	118.36	2	24	15.83	22.7	430.336	2.1139	0.46	0.91
49	Wong and Duncan [23]	CL	83.928	0	24	15.5	22.7	430.336	1.7243	0.41	0.85
50	Wong and Duncan [23]	CL	129.12	0	26	15.62	23.4	860.672	2.3802	0	0.95
51	Wong and Duncan [23]	CL	102.22	12	24	17.19	18.1	860.672	2.2041	0	0.93
52	Wong and Duncan [23]	CL	172.16	20	18	18.38	12.2	403.44	2.1761	0.16	0.79
53	Wong and Duncan [23]	CL	215.2	20	20	17.75	13	823	2.6435	0.17	0.85
54	Wong and Duncan [23]	CL	269	16	19	18.54	13.1	823	2.6435	0.34	0.86
55	Wong and Duncan [23]	CL	107.6	11	19	18	16.2	392.681	2.0414	0.94	0.91
56	Wong and Duncan [23]	CL	150.64	9	19	17.96	16.6	274.339	1.8261	0.71	0.77
57	Wong and Duncan [23]	CL	107.6	3	19	17.65	17.3	279.718	1.5682	0.37	0.65
58	Wong and Duncan [23]	CL	236.72	4	19	17	16.2	946.739	1.8513	1.06	0.98
59	Wong and Duncan [23]	CL	65.636	0	19	14.4	28.8	215.168	1.9638	0.21	0.89
60	Wong and Duncan [23]	CL	39.812	1	38	15.73	31.1	193.651	1.3222	0	0.65
61	Wong and Duncan [23]	CL	54.876	1	36	14.77	28.6	193.651	1.8261	0.02	0.79
62	Wong and Duncan [23]	CL	67.788	0	45	10.63	26.5	215.168	1.8129	0.14	0.77
63	Wong and Duncan [23]	CL	129.12	2	36	14.45	27.4	860.672	1.5563	0.72	0.91
64	Wong and Duncan [23]	SC	279.76	26	36	14.52	24.4	860.672	1.716	0.66	0.89
65	Wong and Duncan [23]	SC	193.68	4	11	19.72	9.6	1452.38	3.5911	−0.08	0.93
66	Wong and Duncan [23]	CL	98.992	31	18	20.17	8.3	107.584	2.7076	0.37	0.64
67	Wong and Duncan [23]	CL	161.4	17	16	16.87	11.5	215.168	2.8129	−0.68	0.9
68	Wong and Duncan [23]	CL	139.88	6	16	17.46	14.3	376.544	2.8261	−0.14	0.93
69	Wong and Duncan [23]	CL	193.68	24	16	17.45	16.8	376.544	2.6335	0.1	0.93
70	Wong and Duncan [23]	CL	204.44	13	16	18.04	11.5	376.544	3.3802	−0.74	0.92

(continued)

Case no.	Reference	Unified soil classif.	Cohesion (kN/m^2)	Friction angle	PI (%)	Dry unit weight γ_d (kN/m^3)	Water content ω_c (%)	Confining stress (kN/m^2)	log k	n	R_f
71	Wong and Duncan [23]	CL	161.4	32	16	18.36	14.5	215.168	3.301	−0.3	0.97
72	Wong and Duncan [23]	CL	139.88	18	16	17.41	8.71	376.544	3.9494	−1.1	0.94
73	Wong and Duncan [23]	CL	68.864	29	16	19	11.7	376.544	3.699	−0.28	0.95
74	Boscardin [26]	ML	28	34	4	18.05	12.1	207.5	2.6435	0.4	0.95
75	Boscardin [26]	ML	24	32	4	17.1	12.1	207.5	2.301	0.26	0.89
76	Boscardin [26]	ML	21	30	4	16.15	12.1	207.5	2.0414	0.25	0.85
77	Boscardin [26]	ML	17	28	4	15.2	12.1	207.5	1.8751	0.25	0.8
78	Boscardin [26]	ML	0	23	4	11.59	12.1	207.5	1.2041	0.95	0.55
79	Boscardin [26]	CL	62	13	15	15.67	21	207.5	2.0792	0.45	1
80	Boscardin [26]	CL	48	15	15	14.85	21	207.5	1.8751	0.54	0.94
81	Boscardin [26]	CL	41	14	15	14.02	21	207.5	1.699	0.6	0.9
82	Boscardin [26]	CL	35	13	15	13.2	21	207.5	1.5441	0.66	0.87
83	Boscardin [26]	CL	0	19	15	10.06	21	207.5	1.2041	0.95	0.75

B.1 Appendix 2

Case no.	Input variables				Output variables		
	PI	Dry unit weight (kN/m^3)	Water content ω_o (%)	Confining stress (kN/m^2)	R_f	n	log k
1	4	17.4	15.6	860.672	0.86	0.59	2.30103
2	4	17.5	12.7	860.672	0.72	1.43	1.43136
3	1	16.65	11.6	349.648	0.83	0.31	2.38021
4	1	16.65	13.6	403.44	0.82	0.38	2.43136
5	1	16.65	16.6	403.44	0.77	0.84	2
6	20	17.4	16.7	715.433	0.87	0.6	2.41497
7	20	17.13	19.5	494.886	0.58	0.48	1.59106
8	23	17.15	19.1	193.651	0.75	0	1.81954
9	23	16.65	21.2	193.651	0.52	0.03	1
10	22	16.33	21.7	193.651	0.57	0	1.5563
11	16	16.87	11.5	215.168	0.9	−0.68	2.81291
12	16	17.46	14.3	376.544	0.93	−0.14	2.82607
13	16	17.45	16.8	376.544	0.93	0.1	2.63347
14	16	18	11.5	376.544	0.92	−0.74	3.38021
15	16	18.36	14.5	376.544	0.97	−0.3	3.30103
16	16	17.41	8.71	376.544	0.94	−1.1	3.94939
17	16	19.1	11.7	376.544	0.95	−0.28	3.69897
18	15	16.81	12.5	403.44	0.8	−0.21	2.50515
19	15	16.81	14.5	349.648	0.81	0.02	2.27875
20	30	17.13	17.2	349.648	0.87	0.23	1.86923
21	30	16.36	17	349.648	0.84	−0.05	1.83251
22	30	16.42	20	349.648	0.85	0.18	1.43136
23	16	17.33	14.6	349.648	0.85	0.29	2.50515
24	32	15.54	23.2	349.648	0.89	0.29	2.30103
25	32	14.68	23.3	403.44	0.86	0.18	2
26	32	14.53	26.7	349.648	0.9	0.14	1.72428
27	16	17.9	15.1	349.648	0.79	0.34	2.20412
28	16	17	15	349.648	0.91	0.27	2.4624
29	12	16	13.5	349.648	0.84	−0.36	2.83251
30	12	17	13.3	349.648	0.68	0.18	2.77815
31	12	16.42	19.3	349.648	0.61	0.32	1.36173
32	12	17	16.7	349.648	0.93	0.6	2.44716
33	12	16.25	16.3	349.648	0.9	0.23	2.34242
34	25	16.8	18.6	349.648	0.84	0.2	2.14613
35	25	16.31	17.1	349.648	0.83	0.09	2.07918
36	25	16.5	19.7	349.648	0.82	0.33	1.6721
37	25	17	13.9	349.648	0.9	−0.15	2.97772

(continued)

Case no.	Input variables				Output variables		
	PI	Dry unit weight (kN/m^3)	Water content ω_o (%)	Confining stress (kN/m^2)	R_f	n	log k
38	25	17.33	16.9	349.648	0.95	0	2.6721
39	23	15.8	20.8	349.648	0.88	0.44	1.87506
40	23	16.8	14.8	349.648	0.84	−0.19	2.92428
41	23	16.2	17.4	349.648	0.87	0.6	2.43136
42	23	16	14.2	349.648	0.83	−0.36	3.04139
43	23	16.71	17.5	349.648	0.87	0.15	2.61278
44	27	15.68	24	322.752	0.86	0.43	1.75587
45	18	15.96	22.9	215.168	0.9	0.43	2.04139
46	20	15.86	22.7	430.336	0.89	0.27	2
47	24	15.7	23.9	430.336	0.97	0.54	2.20412
48	24	15.83	22.7	430.336	0.91	0.46	2.11394
49	24	15.5	22.7	430.336	0.85	0.41	1.72428
50	26	15.62	23.4	860.672	0.95	0	2.38021
51	24	17.19	18.1	860.672	0.93	0	2.20412
52	18	18.38	12.2	403.44	0.79	0.16	2.17609
53	20	17.75	13	823	0.85	0.17	2.64345
54	19	18.54	13.1	823	0.86	0.34	2.64345
55	19	18	16.2	392.681	0.91	0.94	2.04139
56	19	17.96	16.6	274.339	0.77	0.71	1.82607
57	19	17.65	17.3	279.718	0.65	0.37	1.5682
58	19	17	16.2	946.739	0.98	1.06	1.85126
59	19	14.4	28.8	215.168	0.89	0.21	1.96379
60	38	15.73	31.1	193.651	0.65	0	1.32222
61	36	14.77	28.6	193.651	0.79	0.02	1.82607
62	45	10.63	26.5	215.168	0.77	0.14	1.81291
63	36	14.45	27.4	860.672	0.91	0.72	1.5563
64	36	14.52	24.4	860.672	0.89	0.66	1.716
65	11	19.72	9.6	1452.38	0.93	−0.08	3.59106
66	18	20.17	8.3	107.584	0.64	0.37	2.70757
67	16	16.87	11.5	215.168	0.9	−0.68	2.81291
68	16	17.46	14.3	376.544	0.93	−0.14	2.82607
69	16	17.45	16.8	376.544	0.93	0.1	2.63347
70	16	18.04	11.5	376.544	0.92	−0.74	3.38021
71	16	18.36	14.5	215.168	0.97	−0.3	3.30103
72	16	17.41	8.71	376.544	0.94	−1.1	3.94939
73	16	19	11.7	376.544	0.95	−0.28	3.69897
74	4	18.05	12.1	207.5	0.95	0.4	2.64345
75	4	17.1	12.1	207.5	0.89	0.26	2.30103

(continued)

Case no.	Input variables				Output variables		
	PI	Dry unit weight (kN/m^3)	Water content ω_o (%)	Confining stress (kN/m^2)	R_f	n	log k
76	4	16.15	12.1	207.5	0.85	0.25	2.04139
77	4	15.2	12.1	207.5	0.8	0.25	1.87506
78	4	11.59	12.1	207.5	0.55	0.95	1.20412
79	15	15.67	21	207.5	1	0.45	2.07918
80	15	14.85	21	207.5	0.94	0.54	1.87506
81	15	14.02	21	207.5	0.9	0.6	1.69897
82	15	13.2	21	207.5	0.87	0.66	1.54407
83	15	10.06	21	207.5	0.75	0.95	1.20412

References

1. Abad, S., Yilmaz, M., Armaghani, D. J., & Tugrul, A. (2018). Prediction of the durability of limestone aggregates using computational techniques. *Neural Computing and Applications, 29*, 423–433.
2. Douma, O. B., Boukhatem, B., Ghrici, M., & Tagnit-Hamou, A. (2017). Prediction of properties of self-compacting concrete containing fly ash using artificial neural network. *Neural Computing and Applications, 28*, S707–S718.
3. Shirazi, A. Z., & Mohammadi, Z. (2017). A hybrid intelligent model combining ANN and imperialist competitive algorithm for prediction of corrosion rate in 3C steel under seawater environment. *Neural Computing and Applications, 28*, 3455–3464.
4. Millie, D. F., Weckman, G. R., Young II, W. A., Ivey, J. E., Carrick, H. J., & Fahnenstiel, G. L. (2012). Modeling microalgal abundance with artificial neural networks: Demonstration of a heuristic 'grey-box' to deconvolve and quantify environmental influences. *Environmental Modelling and Software, 38*, 27–39.
5. Jebur, A. A., Atherton, W., Al Khaddar, R. M., & Loffill, E. (2018b). Settlement prediction of model piles embedded in sandy soil using the Levenberg–Marquardt (LM) training algorithm. *Geotechnical and Geological Engineering, 36*(5), 2893–2906.
6. Tarawneh, B. (2017). Predicting standard penetration test N-value from cone penetration test data using artificial neural networks. *Geoscience Frontiers, 8*(1), 199–204.
7. Moayedi, H., & Rezaei, A. (2017). An artificial neural network approach for under-reamed piles subjected to uplift forces in dry sand. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-017-2990-z
8. Singh, G., & Walia, B. S. (2017). Performance evaluation of nature-inspired algorithms for the design of bored pile foundation by artificial neural networks. *Neural Computing and Applications, 28*, 289–298.
9. Alizadeh, B., Najjari, S., & Kadkhodaie-Ilkhchi, A. (2012). Artificial neural network modeling and cluster analysis for organic facies and burial history estimation using well log data: A case study of the South Pars Gas Field, Persian Gulf, Iran. *Computers and Geosciences, 45*, 261–269.
10. Mohammed, M. A., Ghani, M. K. A., & Hamed, R. I. (2017). Analysis of an electronic methods for nasopharyngeal carcinoma: Prevalence, diagnosis, challenges and technologies. *Journal of Computational Science, 21*, 241–254.
11. Tarawneh, B. (2013). Pipe pile setup: Database and prediction model using artificial neural network. *Soils and Foundations, 53*(4), 607–615.

12. Yadav, A. K., Malik, H., & Chandel, S. S. (2014). Selection of most relevant input parameters using WEKA for artificial neural network based solar radiation prediction models. *Renewable and Sustainable Energy Reviews, 31*, 509–519.
13. Feng, Y., Barr, W., & Harper, W. F. (2013). Neural network processing of microbial fuel cell signals for the identification of chemicals present in water. *Journal of Environmental Management, 120*, 84–92.
14. Jaeel, A. J., Al-wared, A. I., & Ismail, Z. Z. (2016). Prediction of sustainable electricity generation in microbial fuel cell by neural network: Effect of anode angle with respect to flow direction. *Journal of Electroanalytical Chemistry, 767*, 56–62.
15. Nguyen-Truong, H. T., & Le, H. M. (2015). An implementation of the Levenberg–Marquardt algorithm for simultaneous-energy-gradient fitting using two-layer feed forward neural networks. *Chemical Physics Letters, 629*, 40–45.
16. Kriesel, D. (2011). *Brief introduction to neural networks* [Online]. University of Bonn, Germany. Retrieved July 20, 2018, from http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf.
17. Jebur, A. A., Atherton, W., & Al Khaddar, R. M. (2018a). Feasibility of an evolutionary artificial intelligence (AI) scheme for modelling of load settlement response of concrete piles embedded in cohesionless soil. *Ships and Offshore Structures, 13*(7), 705–718.
18. Chokshi, P., Dashwood, R., & Hughes, D. J. (2017). Artificial Neural Network (ANN) based microstructural prediction model for 22MnB5 boron steel during tailored hot stamping. *Computers and Structures, 190*, 162–172.
19. Caudill, M. (1988). Neural networks primer. Part III. *AI Expert, 3*(6), 53–59.
20. Ismail, A., & Jeng, D. S. (2011). Modelling load settlement behaviour of piles using High-Order Neural Network (HON-PILE model). *Engineering Application of Artificial Intelligence, 24*(5), 813–821.
21. Al-Janabi, K. R. (2006). *Laboratory leaching process modeling in gypseous soils using Artificial Neural Networks (ANNs)*. PhD thesis, Building and Construction Engineering Department, University of Technology, Iraq.
22. Garson, G. D. (1991). Interpreting neural-network connection weights. *AI Expert, 6*, 46–51.
23. Wong, K. S., & Duncan, J. M. (1974). *Hyperbolic stress–strain parameters for nonlinear finite element analyses of stresses and movements in soil masses*. La Jolla, CA: College of Engineering, Office of Research Services, University of California.
24. Abdellatif, M. E. M. (2013). *Modelling the impact of climate change on urban drainage system*. PhD thesis, Faculty of Engineering and Technology, Liverpool John Moores University, Liverpool, UK.
25. Jebur, A. A., Atherton, W., Alkhadar, R. M., & Loffill, E. (2017). Piles in sandy soil: A numerical study and experimental validation. *Procedia Engineering, 196*, 60–67.
26. Boscardin, M. D., Selig, E. T., Lin, R.-S., & Yang, G.-R. (1990). Hyperbolic parameters for compacted soils. *Journal of Geotechnical Engineering, 116*, 88–104.

Index

M. W. Berry et al. (eds.), *Supervised and Unsupervised Learning for Data Science*,
Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-030-22475-2