Document similarity for error prediction

Péter Marjai a, Péter Lehotay-Kéry a and Attila Kiss a, b

a Department of Information Systems, ELTE Eötvös Loránd University, Budapest, Hungary; b Department of Informatics, J. Selye University, Komárno, Slovakia

ABSTRACT
In today’s rushing world, there’s an ever-increasing usage of networking equipment. These devices log their operations; however, there could be errors that result in the restart of the given device. There could be different patterns before different errors. Our main goal is to predict the upcoming error based on the log lines of the actual file. To achieve this, we use document similarity. One of the key concepts of information retrieval is document similarity which is an indicator of how analogous (or different) documents are. In this paper, we are studying the effectiveness of prediction based on cosine similarity, Jaccard similarity, and Euclidean distance of rows before restarts. We use different features like TFIDF, Doc2Vec, LSH, and others in conjunction with these distance measures. Since networking devices produce lots of log files, we use Spark for Big data computing.

1. Introduction
Finding similar documents has become a common task in the past few years and it is a major problem of data mining. It can be useful for a set of problems. One of these problems is for cross-document co-reference resolution. This means the identification of co-referring entity mentions in different documents that refer to the same entity (Keshtkaran et al., 2017). In Mayfield et al. (2009), they use cosine similarity to score the mentions and the words around the entity. In Rao et al. (2010), TFIDF weighted vectors are used with cosine similarity. The same is used in the proposed CROCS framework in Dutta and Weikum (2015).

Another problem similarity is often used for is clustering. Clustering can be used to establish groups of similar documents from a large volume of documents. Such smaller clusters can be used for browser algorithms, information navigation, and recommending engines. In Huang (2008), they represent documents with TFIDF vectors and use various similarity measures like Cosine similarity, Jaccard Similarity, Euclidean distance, Pearson Correlation Coefficient, and Averaged Kullback–Leiber Divergence. A simple transformation is applied to the values to make them fit into the K-means algorithm which is
used for clustering. In Karypis et al. (2000), they focus on the different clustering algorithms, while TFIDF and cosine similarity is still used here. In Chim and Deng (2008), instead of using TFIDF with cosine similarity they use Phrase-Bases Document Similarity with clustering algorithms.

Similarity can also be used for plagiarism detection. This problem became of interest because of its importance. In Zechner et al. (2009), cosine similarity is used with various measures. In Gustafson et al. (2008), plagiarism is recognized using the Sentence Similarity. In Potthast et al. (2011), various models like the Cross-language character n-Gram model (CL-CNG), Cross-language explicit semantic analysis (CL-ESA), and Cross-language alignment-based similarity analysis (CL-ASA) are used to find plagiarism.

A similar problem is to find articles from the same source or finding mirror pages on the web. Document similarity is used to address this problem in Fetterly et al. (2003). In Haveliwala et al. (2002), methods with document similarity are evaluated to solve this problem.

### 2. Related works

In this paper, evaluate different document similarity metrics to get a more comprehensive picture of their efficiency. These metrics are Cosine similarity with TFIDF, Cosine similarity with Doc2Vec, Jaccard similarity with MinHashing and LSH, and Euclidean distance with OneHotEncoding and Bucketed Random Projection. For a randomly chosen set of lines, we are comparing the ten most likely predictions for each measure to give a basic picture of the predictions. We then compare the prediction rate of the measures. After that, we compare the frequency of valid predictions on the top 10 predictions. Lastly, we study the speed of the measures.

The organization of the paper is the following. Section 3 contains the definition of similarity and gives an overview of the used distance measures. The word encoding algorithms are also explained here. In Section 4, the performed experiments are described, respectively, and numerical examples based on the log lines of real-life networking devices are represented to compare the effectiveness of the different measures. Section 5 is the conclusion about the issue in question and the possible future works are also listed here.

### 3. Concepts and problems

#### 3.1. Similarity

A similarity measure is a real-valued function that evaluates the resemblance between two items. There is not a single definition for similarity measure; however, these measures are analogous to the inverse of distance metrics: they produce either zero or a negative value for dissimilar objects, while large values for similar items.

We use similarity to find textually alike log lines before the upcoming error and in a large corpus of lines that occurred before specific errors. The aspect of similarity we use in our research is character-level similarity not semantical so we examine the words in the log lines, not their meanings.
### 3.1.1. Cosine similarity

Cosine similarity is a similarity measure between two cardinal vectors of an inner product space. It is equal to the inner product of the vectors normalized to both have length 1, which is the cosine of the angle between the vectors. The positive space, where the result is between [0, 1] is where the cosine similarity is especially used. If the unit vectors are parallel, they are ultimately similar and dissimilar if they are orthogonal. This is correspondent to the cosine which is maximum when the components span a zero angle and null when they are perpendicular. Given the vectors A and B, the cosine similarity, \( \cos(\theta) \), is represented using a dot product and magnitude as,

\[
similarity = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}.
\]

where \( A_i \) and \( B_i \) are segments of the vectors \( A \) and \( B \).

For any number of dimensions, these bounds could be applied which is why the cosine similarity is frequently used in high-dimensional positive spaces. Text mining and information retrieval use this feature as a different dimension is appointed to each word. After this, the document can be represented by a vector with values in each dimension corresponding to the number of usage of the word in the document. In our experiments, the cosine similarity is used with TF-IDF and with Doc2vec.

### 3.1.2. Jaccard similarity

Jaccard similarity addresses well the problems of finding documents that are textually similar in a large corpus such as log lines. The dissimilarity and similarity of two sets can be measured with this statistic, which was developed and published by Paul Jaccard in Jaccard (1912). It can be defined as the size of the intersection divided by the size of the union of the sets:

\[
J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad \text{(2)}
\]

(If \( A \) and \( B \) are both empty, define \( J(A, B) = 1 \).) This means that the Jaccard similarity is bounded as:

\[
0 \leq J(A, B) \leq 1 \quad \text{(3)}
\]

To efficiently compute a proper estimation of the Jaccard similarity for sets, we use MinHash min-wise independent permutations and Locality-Sensitive hashing scheme. From the values of the hash function, fix-sized signatures are made, which represent the sets.

### 3.1.3. Euclidean distance

In mathematics, the best-known distance measure is the Euclidean distance, which is a straight-line distance between two points in Euclidean space. In an n-dimensional Euclidean space points are vectors of n real numbers. Let \( x \) and \( y \) be two points. The length of the line section connecting them (denoted as \( xy \), is the Euclidean distance of the two points. In Cartesian coordinates, the two points are \( x = (x_1, x_2, \ldots, x_n) \) and \( y = (y_1, y_2, \ldots, y_n) \) in the Euclidean n-space, then their distance \( (d) \) is given by the Pythagorean formula
(Anton & Rorres, 2013):

\[
d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}
\]

This is also referred as \(L_2\)-norm.

Other distance measures can be used for Euclidean spaces. For any constant \(r\), \(L_r\)-norm can be defined to be the distance measure \(d\):

\[
d([x_1, x_2, \ldots, x_n], [y_1, y_2, \ldots, y_n]) = \left( \sum_{i=1}^{n} |x_i - y_i|^r \right)^{\frac{1}{r}}
\]

The case \(r=1\) or \(L_1\)-norm is the Manhattan distance. \(L_\infty\)-norm is another measure, it being the limit as \(r\) approaches infinity of the \(L_r\)-norm. \(L_\infty\)-norm is defined as the maximum of \(|x_i - y_i|\), over all dimensions \(i\). This is because only the dimension with the largest difference matters, as \(r\) gets larger. To encode the words into dense vectors, we use OneHotEncoder. We use this measure with Bucketed Random Projection.

### 3.2. TFIDF

In information retrieval, the importance of a word taken from the corpus to a document can be represented with TFIDF, short for term frequency-inverse document frequency, which is a numerical statistic. For a given word, it is a measure of how intense are the occurrences of it.

#### 3.2.1. Term frequency (TF)

Term Frequency measures how frequently a term takes place in a document. Suppose we have a couple of documents and we want to rate them based on their relevance to the query, ‘the white cat’. Eliminating the documents not containing all three words still leaves many documents. We can count the number of occurrences of each term in each document to further distinguish them. Since these documents vary in length, it is also possible that a long document contains the term much more than a sort one, so the frequency should be normalized. More formally Term Frequency can be defined as:

\[
TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}
\]

where \(f_{ij}\) is the number of occurrences (frequency) of a word (term) \(i\) in a document \(j\), which is divided by the maximum number of occurrences of any word in that document. This means that the most frequent word gets a TF value of 1 and others get fractions as their value for this document.

#### 3.2.2. Inverse document frequency (IDF)

Inverse Document Frequency measures the importance of a term. All terms are considered to be equally important while computing TF. Since there are some very common words like
‘the’ Term Frequency is likely to falsely bear down on documents containing ‘the’ more frequently, without giving enough weight to the meaningful terms like ‘white’ and ‘cat’. That is why weighing down the frequent terms and scaling up the rare ones are needed. Let $N$ be a collection of documents. Suppose that term $i$ appears in $n_i$ of the $N$ documents in the collection. Inverse Document Frequency then can be computed as:

$$IDF_i = \log_2 \left( \frac{N}{n_i} \right)$$

(7)

Then Term Frequency-Inverse Document Frequency for term $i$ in document $j$ is calculated as:

$$TF-\text{IDF} = TF_{ij} \times IDF_i$$

(8)

The terms with higher values are the terms that best characterize the document.

3.3. Word2Vec and Doc2Vec

Word2Vec is a natural language processing technique. It is an open-source tool introduced by Google researchers in Mikolov et al. (2013). It proposes effective algorithms to represent words as N-dimensional real vectors, which is also known as word embedding. It can also measure the similarity of words. The algorithm uses a simple neural network model with a single hidden layer. It has two models, continuous bag-of-words (CBOW) and Skip-Gram. The algorithm takes a large corpus of text as its input and creates a vector space, usually of hundreds of dimensions where each distinct word is assigned with a related vector in the space. Similar context words are located close to each other in the vector space. These vectors are selected cautiously so the mathematical function, cosine similarity can state the semantic similarity between the words that are represented by vectors.

Doc2Vec is an extension to Word2vec proposed in Le and Mikolov (2014). It constructs embeddings regardless of documents regardless of their lengths. An another vector called Paragraph ID (or doc ID) had been added. The Doc2Vec can be calculated by two algorithms. The first one is similar to the CBOW model and is called Distributed Memory version of Paragraph Vector (PV-DM). The other one, which is similar to Skip-Gram called Distributed Bag of Words version of Paragraph Vector(PV-DBOW) can also be used.

In our experiments, we use the Skip-Gram approach. Each word is converted into a feature vector using Word2Vec, then Doc2Vec is represented as the average of these vectors, and its length will also be the same as the length of Word2Vec vectors. Formally:

$$D2V = \frac{W2W(w_1) + W2W(w_2) + \ldots + W2W(w_n)}{n}$$

(9)

where $W2W(w_i)$ represents the $i^{th}$ word’s W2W vector and $n$ is the number of vectors.

3.4. MinHashing

MinHash is a technique used to rapidly calculate the similarity of two sets. It was invented by Andrei Broder (1997). It was used to detect duplicate web pages, and it also has been
applied on large-scale clustering problems, for example, clustering documents based on their similarity of sets of words. There is a noteworthy relationship between the Jaccard similarity of the sets that have been minhashed. The Jaccard similarity of two sets is the same as the probability of producing the same value for two sets with the minhash function for a random permutation of rows.

Let $A$ and $B$ be two subsets of set $U$. The minimal member of any set $S$ with regard to $h \circ \text{perm}$ (the member $x$ of $S$ with the minimal value of $h(\text{perm}(x))$ is defined as $h_{\min}(S)$, where $\text{perm}$ is a random permutation of the elements of $U$ and $h$ is a hash function mapping the members of $U$ to individual numbers. Applying $h_{\min}$ on $A$ and $B$ both assuming there is no hash conflict, $h_{\min}(A)$ and $h_{\min}(B)$ values are equal if and only if amidst all elements of $A \cup B$, $A \cap B$ intersection contains the minimum hash value element. The Jaccard index, and the probability of the above-described case are the same, therefore:

$$\Pr[h_{\min}(A) = h_{\min}(B)] = J(A, B)$$  (10)

### 3.5. Locality-Sensitive hashing (LSH)

While minhashing can be used to create small signatures from large documents and to maintain the excepted similarity of any pair of documents, efficiently finding the pairs with the highest similarity could be still impossible since the number of pairs of documents could be too large. Locality-Sensitive Hashing (Rajaraman & Ullman, 2011) is a technique that hashes input items several times with a high probability of similar items being hashed in the same bucket than dissimilar are. The number of buckets is considerably less than the number of possible input items.

An LSH family (Indyk & Motwani, 1998) $\mathcal{F}$ is defined for a threshold $R>0$, a metric space $\mathcal{N} = (N, d)$, and $c>1$ an approximation factor. This family is made of $h: N \rightarrow S$ functions, which map elements from the metric space (input) to a bucket $s \in S$. For any two points $x, y \in N$ using a function $h \in \mathcal{F}$ chosen indiscriminately, the LSH family fulfils the conditions below:

- if $d(x, y) \leq R$, then $h(x) = h(y)$ with the probability at least $P_1$
- if $d(x, y) \geq cR$, then $h(x) = h(y)$ with the probability at most $P_2$

Families where $P_1 > P_2$ are points of interests and are called $(R, cR, P_1, P_2) − sensitive$

After the hashing of the items, any pairs in the same bucket are considered to be a candidate pair, and only those pairs are reviewed for similarity. After minhashing the items, dividing the signature matrix into $b$ bands composed of $r$ rows each is an efficient method to select the hashing. Every band is hashed into some great amount of buckets with a hash function taking vectors of $r$ integers. Columns with the same vector in different bands cannot hash into a specific bucket, since different bucket arrays are used for each band. This means, only equivalent vectors are hashed into the same bucket. Columns not matching in a band could become a candidate pair by other bands. Consequently, this approach makes similar columns become candidate pairs with greater likelihood.
3.6. One-Hot encoding

In computer science, a one-hot is a group of bits among which only legal combinations of values are marked with 1 and all the others are marked with 0 (Harris & Harris, 2010). In natural language processing, an $1 \times N$ vector is called one-hot vector and is used to differentiate each word in a corpus from every other words in it. This vector only contains one single 1 in a cell that is used particularly to determine the word, and every other cell contains 0. For example:

\[
\begin{pmatrix}
  \text{red} & \text{green} & \text{blue} \\
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1 \\
\end{pmatrix}
\]

It also guarantees that machine learning does not suppose that larger numbers are more significant. For example, 7 is bigger than 2 but that does not make 7 more crucial than 2. The same goes for words, the word ‘apple’ is not more essential than the word ‘cat’.

3.7. Bucketed random projection

Bucketed Random Projection is an LSH class for Euclidean distance metrics. This technique is used to scale down the dimensionality of a set of points lying in Euclidean space. Points in the Euclidean distance space are represented with sparse or dense vectors which are the input of this method. Vectors of configurable dimension are the output. The hash values in the same dimension are calculated by the same hash function. This method has low error rates and great power and it preserves distances well, while empirical results are sparse (Bingham & Mannila, 2001). Points in a vector space can be projected into a lower-dimensional space if they are of adequately high dimension. This is achieved in a way that approximately keeps the distances between points. The idea behind this is the Johnson–Lindenstrauss lemma (Johnson & Lindenstrauss, 1984).

In random projection, using a random $k \times d$ dimensional matrix $R$ which has columns with unit lengths, the original $d$- dimensional data is projected to a $k$-dimensional subspace. Let $X_{d \times N}$ be the original set of information, then

\[
X_{k \times N}^{\text{RP}} = R_{k \times d} X_{d \times N},
\]

which is the data’s projection into a lower $k$-dimension subspace. In Euclidean distance, to solve the search problem, the random matrix $R$ is generated using Gaussian distribution (Wang et al., 2014).

For a real-valued random variable Gaussian distribution is a continuous probability distribution, and its general form of the probability density function is:

\[
f(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
\]

where $\sigma$ is the standard deviation, $\mu$ is the expectation of the distribution and $\sigma^2$ is the variance of the distribution.
4. Experiments

4.1. Data

Our data consisted of log lines produced by different network devices used at the Ericsson-ELTE Software Technology Lab. We tested the similarity measures on three distinct sets consisting of groups of log lines with error codes assigned to them. The first consisted of 69,211 lines in total and 12,070 groups of log lines with error codes. The second consisted of 46,877 lines in total which produced 8024 groups of log lines. The third one consists of 47,612 lines in total which made 6862 groups of log lines. Overall 163,700 log lines were organized into 26,956 groups.

Our test data was also distinct from these sets and consisted of 150 groups of error lines with error codes.

4.2. Experimental analysis

Several experiments were conducted to verify the efficiency of the investigated methods in pair with distance methods. Cosine similarity with Doc2Vec and TFIDF approach, Jaccard similarity with MinHashing and Euclidean distance with OneHot Encoding and Bucketed Random Projection were chosen to predict an error code for a group of log lines. The validity of these predictions is compared with each other. The experimental analyses are divided into four parts and are explained below.

4.2.1. Experiment 1: comparing the five most likely prediction by each measure for a set of log lines

First of all, to visualize the results of our rankings, a simple example is represented (Table 1), where predictions were made on a random set of lines based on its similarity to other documents. The randomly chosen set of lines has the error code 192, which is a fairly regular code. The higher the Cosine similarity and Jaccard similarity values, the more similar the documents are. Euclidean distance represents the distance between two documents, so the lower, the more similar the documents are.

This randomly chosen example perfectly showcases that there could be a lot of false predictions. All of the investigated methods assign high values to not valid error codes.

Table 1. Top-ten predictions by Cosine similarity, with D2V and TFIDF, Jaccard similarity with MinHashing and Euclidean distance with OneHotEncoding and BRP.

| Rank | Cosine D2V | Cosine TFIDF | Jac. MinHash | Eu. OH. BRP |
|------|------------|--------------|--------------|-------------|
| 1    | 182 (0.99815) | 192 (0.46996) | 192 (0.79545) | 201 (5.83095) |
| 2    | 182 (0.99624) | 182 (0.46678) | 201 (0.79070) | 201 (5.83095) |
| 3    | 192 (0.99614) | 201 (0.46095) | 201 (0.76744) | 182 (5.83095) |
| 4    | 201 (0.99405) | 182 (0.45096) | 201 (0.76744) | 182 (5.83095) |
| 5    | 201 (0.99383) | 193 (0.44449) | 201 (0.76744) | 192 (6.00000) |
| 6    | 192 (0.98882) | 192 (0.44163) | 192 (0.76087) | 201 (6.63325) |
| 7    | 182 (0.98850) | 182 (0.43980) | 182 (0.75000) | 182 (6.70820) |
| 8    | 192 (0.98406) | 182 (0.43891) | 182 (0.75000) | 193 (6.85565) |
| 9    | 182 (0.98071) | 191 (0.43852) | 182 (0.75000) | 193 (6.92820) |
| 10   | 193 (0.08313) | 182 (0.43589) | 182 (0.75000) | 100 (6.92820) |

Note: Italics are the prediction scores.
This could be because of the length of the lines. The similar prerequisites of the errors also can result from this. For example, error 192 means *Software upgrade completed, request for cold restart* while 193 means *Software upgrade completed, request for warm restart*. The only exception is the type of restart so it is likely that the lines resulting in these error codes can be very similar and resulting in false high values.

4.2.2. **Experiment 2: comparing the prediction rate of the measures**

To evaluate the performance of the measures we take the first predictions by the methods and comparing them with the actual error code. This way the prediction rate can be estimated. The results are obtained by averaging over 450 implementations. The results are shown in **Figure 1**.

In the case of our log line groups, it can be seen that the Euclidean OneHot BRP method slightly overpowers all other methods; however, it is still likely that it gives false predictions. This could be because of the combination of both OneHot Encoding and Bucketed Random Projection. Not only the words are converted into bit vectors these vectors are also projected into a lower-dimensional space, while cosine methods have higher dimensions, especially in the case of D2V vectors.

4.2.3. **Experiment 3: comparing the frequency of valid prediction on the top 10 predictions**

When making a decision, while the first prediction is the most likely, it is important to analyse the other likely predictions as well because they could provide other useful information. We used the first 10 predictions to compare the efficiency of the measures. The results are obtained by averaging over 450 implementations. **Figure 2** shows the average of the valid predictions at each rank.

In the case of all measures, it can be seen that there are predictions with better value than the predictions in the first place. This implies that false predictions could be made by any of the methods. It can be also seen that Jaccard similarity with MinHashing performs

![Figure 1](image.png)

**Figure 1.** The validity of the first predictions by Cosine similarity with D2V, Cosine similarity with Tfidf, Jaccard similarity with MinHashing and Euclidean distance with OneHot Encoding and Bucketed Random Projection.
better than any other methods in the case of the first 10 predictions. While Cosine similarity with D2V, Jaccard similarity with MinHashing and Euclidean distance with OneHot Encoding and BRP tend to predict with similar performance, Cosine similarity with TFIDF has performance drops.

The average of the first 10 predictions can be seen in Figure 3. The cosine methods barely have 50% performance; however, Jaccard similarity with MinHashing overpowers Euclidean with OneHot Encoding and BRP. From these experiments, the conclusion can be drawn that to get a valid prediction the usage of these measures is not enough.
4.2.4. Experiment 4: comparing the speed of the measures

The speed of a method is also an important feature. Some measures could be performed faster than others which is important when the quickest possible reaction is needed. We made three speed comparisons. The results are obtained by averaging over 450 implementations. Firstly, we studied the speed of distance measures. The details can be seen in Figure 4.

From the three investigated distance measures, Cosine similarity is the fastest, greatly overpowering the other two being more than 10 times faster than Jaccard similarity and Euclidean distance. This could be because of the vector representation used in Cosine measures. The comparison between the speed of used methods, namely D2V, TFIDF, MinHashing, OneHot Encoding, and Bucketed Random Projection is seen in Figure 5.

While OneHot Encoding is outstandingly slow, MinHashing BRP and TFIDF are the fastest methods. This could be because of the low calculation complexity of these methods. Doc2Vec is noticeably slower is a result of first turning words into vectors.

Figure 4. The speed of cosine similarity, Euclidean distance and Jaccard similarity distance measures.

Figure 5. The speed of D2V, TFIDF, MinHashing, OneHot Encoding and Bucketed Random Projection methods.
and only then turning these vectors into D2V vectors. It can be also seen that, with the exception of MinHashing and BRP, all of the Max values are significantly higher than the Avg values. This is because of the first initialization of the models.

To get a more detailed picture of the speed of the investigated measures, the overall speed should be taken into account. This can be seen in Figure 6.

Euclidean distance with OneHot Encoding takes the most time to find the prediction. While Cosine TFIDF needs the least time to make a prediction, it still has a high Max value. Overall Jaccard similarity with MinHashing is the most effective, which takes the double amount of time as Cosine TFIDF; however, its Max value is significantly slower.

5. Conclusion and future work

In this paper, we surveyed the error prediction capability of several similarity measures. These measures calculate similarity based on how different two sentences are. Instead of sentences, we used sets of log lines created before errors that were produced by networking devices. We used different methods to create vectors and hash values which are the inputs of the similarity measures. We then gave a detailed example of a random set of log lines to represent how predictions look like. To analyse the performance of these measure-method pairs, several experiments were conducted. The experimental results showed that there is around 10% difference in prediction rates between the best and worst measure, which is very low. It was also discovered that speed depends on the combination of the used measure and the used method. Overall, Jaccard similarity with MinHashing had the best performance.

During our experiments, we came to the conclusion that the single use of these measures is not enough to correctly predict the upcoming error, as all of the measures have around 50% success rate.

While we only investigated these measures with one single method each, it is possible that other combinations could result in better performances. We also think that a specific
encoder that takes important information from the log lines and encodes it could amplify the performance as not necessary words in the log lines would not be used when making a prediction.

**Disclosure statement**

No potential conflict of interest was reported by the authors.

**Funding**

The project has been supported by the European Union, co-financed by the European Social Fund [grant number EFOP-3.6.3-VEKOP-16-2017-00002].

This publication is the partial result of the Research & Development Operational Programme for the project “Modernisation and Improvement of Technical Infrastructure for Research and Development of J. Selye University in the Fields of Nanotechnology and Intelligent Space” [grant number ITMS 26210120042], co-funded by the European Regional Development Fund.

The project was also supported by the Ericsson-ELTE Software Technology Lab.

**Notes on contributors**

**Péter Marjai** received a B.Sc. degree in computer science from Eötvös Loránd University in 2019 and currently pursuing an M.Sc. degree.

**Péter Lehotay-Kéry** received his MSc degree in computer science at Eötvös Loránd University Faculty of Informatics in Budapest, 2018 and currently doing his PhD studies with specialization in information systems. His scientific research is focusing on databases, security, big data, data mining and bioinformatics.

**Attila Kiss** was born in 1960. In 1985 he graduated (MSc) as a mathematician at Eötvös Loránd University, in Budapest, Hungary. He defended his Ph.D. in the field of database theory in 1991. Since 2010 he is working as the head of Information Systems Department at Eötvös Loránd University. Since 2015 he has been teaching at J. Selye University, Komárno, Slovakia, too. His scientific research is focusing on database theory and practice, security, semantic web, big data, data mining, artificial intelligence, and bioinformatics. He was the supervisor of 7 Ph.D. students. He has more than 160 scientific publications.

**ORCID**

Péter Marjai http://orcid.org/0000-0003-1462-5629

Péter Lehotay-Kéry http://orcid.org/0000-0002-0884-4297

Attila Kiss http://orcid.org/0000-0001-8174-6194

**References**

Anton, H., & Rorres, C. (2013). *Elementary linear algebra: applications version*. John Wiley & Sons.

Bingham, E., & Mannila, H. (2001). Random projection in dimensionality reduction: Applications to image and text data. In *Proceedings of the Seventh ACM Sigkdd International Conference on Knowledge Discovery and Data Mining* (KDD ’01). Association for Computing Machinery, New York, NY, USA (pp. 245–250). https://doi.org/10.1145/502512.502546

Broder, A. Z. (1997). On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences 1997* (Cat. No. 97tb100171), Salerno, Italy (pp. 21–29). https://doi.org/10.1109/SEQUEN.1997.666900
Chim, H., & Deng, X. (2008). Efficient phrase-based document similarity for clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(9), 1217–1229. https://doi.org/10.1109/TKDE.2008.50

Dutta, S., & Weikum, G. (2015). Cross-document co-reference resolution using sample-based clustering with knowledge enrichment. *Transactions of the Association for Computational Linguistics*, 3, 15–28. https://doi.org/10.1162/tacl_a_00119

Fetterly, D., Manasse, M., & Najork, M. (2003). On the evolution of clusters of near-duplicate web pages. In *Proceedings of the IEEE/LEOS 3rd International Conference on Numerical Simulation of Semiconductor Optoelectronic Devices (IEEE Cat. No. 03ex726)*, Santiago, Chile (pp. 37–45). https://doi.org/10.1109/LAWEB.2003.1250280

Gustafson, N., Pera, M. S., & Ng, Y. K. (2008). Nowhere to hide: Finding plagiarized documents based on sentence similarity. In *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Sydney, NSW, Australia (Vol. 1, pp. 690–696). https://doi.org/10.1109/WIAT.2008.16

Harris, D., & Harris, S. (2010). *Digital design and computer architecture*. Morgan Kaufmann.

Haveliwala, T. H., Gionis, A., Klein, D., & Indyk, P. (2002). Evaluating strategies for similarity search on the web. In *Proceedings of the 11th International Conference on World Wide Web (WWW ’02)*, Association for Computing Machinery, New York, NY, USA (pp. 432–442). https://doi.org/10.1145/511446.511502

Huang, A. (2008, April). Similarity measures for text document clustering. In *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Sydney, NSW, Australia (Vol. 1, pp. 690–696). https://doi.org/10.1109/WIIAT.2008.16

Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC ’98)*, Association for Computing Machinery, New York, NY, USA (pp. 604–613). https://doi.org/10.1145/276698.276876

Jaccard, P. (1912). The distribution of the flora in the alpine zone. 1. *New Phytologist*, 11(2), 37–50. https://doi.org/10.1111/nph.1912.11.issue-2

Johnson, W. B., & Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26(189–206), 1. https://doi.org/10.1090/conm/026/737400

Karypis, M. S. G., Kumar, V., & Steinbach, M. (2000, May). A comparison of document clustering techniques. In *Textmining Workshop at kdd2000*.

Keshtkaran, A., Yuhaniz, S. S., & Ibrahim, S. (2017). An overview of cross-document coreference resolution. In *2017 International Conference on Computer and Drone Applications (Iconda)*, Kuching, Malaysia (pp. 43–48). https://doi.org/10.1109/ICONDA.2017.8270397

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML’14)*. JMLR.org, II–1188–II–1196.

Mayfield, J., Alexander, D., Dorr, B. J., Eiser, J., Elsayed, T., Finin, T., Fink, C., Freedman, M., Garera, N., McNamee, P., & Mohammad, S. (2009). Cross-document coreference resolution: A key technology for learning by reading. In *Aaai Spring Symposium: Learning by Reading and Learning to Read*, California, USA (Vol. 9, pp. 65–70). AAAI PressPalo Alto.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

Potthast, M., Barrón-Cedeño, A., Stein, B., & Rosso, P. (2011). Cross-language plagiarism detection. *Language Resources and Evaluation*, 45(1), 45–62. https://doi.org/10.1007/s10579-009-9114-z

Rajaraman, A., & Ullman, J. D. (2011). *Mining of massive datasets*. Cambridge University Press.

Rao, D., McNamee, P., & Dredze, M. (2010). Streaming cross document entity coreference resolution. In *Coling 2010: Posters*, Coling 2010 Organizing Committee Beijing, China (pp. 1050–1058). https://www.aclweb.org/anthology/C10-2121/

Wang, J., Shen, H. T., Song, J., & Ji, J. (2014). Hashing for similarity search: A survey. arXiv preprint arXiv:1408.2927.

Zechner, M., Muhr, M., Kern, R., & Granitzer, M. (2009). External and intrinsic plagiarism detection using vector space models. In *Proceedings of the SEPLN’09 Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse (PAN-2009)* and *1st International Competition on Plagiarism Detection*, Graz (Vol. 32, pp. 47–55). https://graz.pure.elsevier.com/en/publications/external-and-intrinsic-plagiarism-detection-using-vector-space-mo