ISA Transactions xxx (xxxx) xxx

Contents lists available at ScienceDirect

ISA Transactions

journal homepage: www.elsevier.com/locate/isatrans

Research article

Analysis and safety engineering of fuzzy string matching algorithms

Malgorzata Pikies *, Junade Ali

Cloudflare, London, United Kingdom

ARTICLE INFO

Article history:
Received 14 October 2019
Received in revised form 11 May 2020
Accepted 3 October 2020
Available online xxxx

Keywords:
String similarity
Fuzzy string matching
Safety engineering
Natural language processing
Binary classification
Neural network

ABSTRACT

In this paper we explore fuzzy string matching in an automatic ticket classification and processing system. We compare performance of the following string similarity algorithms: Longest Common Subsequence (LCS), Dice coefficient, Cosine Similarity, Levenshtein (edit) distance and Damerau distance. Through optimisation, we accomplished a 15% improvement in the ratio of false positives to true positive classifications over the existing approach used by a customer support system for free customers. To introduce greater safety; we compliment fuzzy string matching algorithms with a second layer Convolutional Neural Network (CNN) binary classifier, achieving an improved keyword classification ratio for two ticket categories by a relative 69% and 78%. Such an approach allows for classification to only be applied where a desired level of safety achieved, such as in instances where automated answers.

1. Introduction

A key benefit of using an automated ticket classification system in a customer support context is the ability to introduce automated responses to improve the customer experience by decreasing customers wait times. The reduced ticket response time, together with a qualitatively reliable reply can reduce costs of the business and raise the overall customer satisfaction. Our original research was motivated by a need of accuracy and computation time improvements. The algorithm used prior to our research was based on the edit distance calculation. In [1] we showed that the cosine similarity measure performed better than other algorithms and ran at an improved computational performance ($O(n + m)$ difficulty instead of $O(n * m)$ for two strings n and m). Driven by needs to expand our ticket classification system to areas where a greater degree of certainty was required before classification, we introduced a few novel innovations. We include Name Entity Recognition to identify and sanitise meta-information like email footers, define a modified implementation of the Cosine similarity algorithm to account for numeric values in strings and we define a novel approach using Convolutional Neural Networks for safety-engineering for fuzzy string matching. We provide comparative analysis for the safety-engineering strategy and comparative data for string similarity. To the best of our knowledge, no other approaches for safety-engineering string similarity algorithms. Our improved system presented in this paper is made of three components:

1. a database with replies to the most frequent customers' enquiries,
2. a string similarity based classifier,
3. a new binary classifier based on neural networks.

In Section 2 we introduce the theoretical description of the chosen string similarity algorithms with the focus on the edit distance and the cosine similarity. The other algorithms used in this paper are discussed in more details in our original work [1]. We further introduce theoretical principles behind neural networks, which is concluded with a summary of a related work. In Section 3 we present the two classification models and in Section 4 we describe the dataset used for our studies, together with the data sanitisation used prior to neural network training and application. In Section 5 we focus on the improvements accomplished by using the chosen string similarity algorithm [1] and by introducing a second layer classification system (the binary classifier). The research is summarised in Section 6.

2. Current knowledge

2.1. Fuzzy String matching

Fuzzy string matching is a technique used to find approximate matches between two strings. Algorithms may be divided into two categories due to the feature they measure:

- similarity algorithms: the match is found if $S(X, Y) \geq t_S$,
- dissimilarity algorithms: the match is found if $D(X, Y) \leq t_D$,

where $t_{S/D}$ is a string similarity/dissimilarity threshold, $S(X, Y)$ and $D(X, Y)$ are the similarity and dissimilarity functions, X and

* Corresponding author.
E-mail address: malgorzata@cloudflare.com (M. Pikies).

https://doi.org/10.1016/j.isatra.2020.10.014
0019-0578/

Y are the two strings in question. A string of a size k is a sequence of characters $(x_1x_2 \dots x_k)$ and it can be divided into groups of characters or words called n-grams. An n-gram based on single characters is simply a substring of a length n from the original string [2]. A string containing multiple words can also be divided into n-grams made of phrases or words, instead of characters.

2.1.1. The edit distance

The edit distance (known also as the Levenstein distance) is a special case of a unigram (an n-gram of a unit size) distance [3], where strings are tokenised into n-grams made of characters. The distance is equal to the minimum number of elementary edit operations, necessary to transform one string into another. Depending on a definition and treatment of elementary operations, the Levenstein distance has different variations. In the classical edit distance problem, a set of elementary edit operations for two strings ($|X| \neq |Y|$) $X \rightarrow Y$ contains:

- a change operation, if $X \neq \emptyset$ and $Y \neq \emptyset$;
- a delete operation, if $Y = \emptyset$;
- an insert operation, if $X = \emptyset$.

Hence, the edit distance is the minimal number of changes, deletions, and insertions needed to transform the string X into the string Y.

The authors of [3] defined the minimum distance d between strings $X = (x_1x_2 \dots x_k)$ and $Y = (y_1y_2 \dots y_l)$ divided into n-grams of a size n:

$$d_n(\Gamma_{k,l}) = min(d_n(\Gamma_{k-1,l}) + 1, d_n(\Gamma_{k,l-1}) + 1, d_n(\Gamma_{k-1,l-1}) + d_n(\Gamma^n_{k-n,l-n})), \quad (1)$$

where $\Gamma_{i,j} = (x_1 \dots x_i, y_1 \dots y_j)$ is a pair of strings of size i and j and $\Gamma^n_{i,j} = (x_{i+1} \dots x_{i+n}, y_{j+1} \dots y_{j+n})$ is a pair of n-grams in the two strings. The cost of the dynamic programming solution to this problem is $\mathcal{O}(k \cdot l)$.

As mentioned before, there are many variations of the edit distance algorithm. In case where the above mentioned set of elementary operations is enriched by adding a transposition of two adjacent characters, the algorithms is called he Damerau–Levenshtein distance [4]. The distance can be normalised to its length L (the number of elementary edit operations), thus the Normalised Levenshtein distance is the minimum value of the weighted sum over L [5]. It can also be normalised to the sum of stings' lengths using the length of the longest string as a normalisation constant, then it gets values from 0.0 to 1.0. Authors of [6] defined a normalised Levenstein distance for the transition $X \rightarrow Y$:

$$d_{norm}(\Gamma_{k,l}) = \frac{2 \cdot d_1(\Gamma_{k,l})}{\alpha \cdot (|X| + |Y|) + d_1(\Gamma_{k,l})}, \quad (2)$$

where α is equal to a maximum value of elementary edit operation transforming string X to the null string, or the null string to Y.

2.1.2. Cosine similarity

The Cosine Similarity [7] is a method which can measure a similarity between two strings based on their similarity angle in a multi-dimensional space. In this method, strings are tokenised into n-grams of words or characters, where each unique n-gram is a separate dimension. As a result, each text can be represented as a vector in the multi-dimensional space. With vectors normalised to unity [8], we get a cosine of the angle θ between them [7]:

$$s(X, Y) = \frac{\vec{U}(X) \cdot \vec{V}(Y)}{|\vec{U}(X)||\vec{V}(Y)|} = \cos\theta. \quad (3)$$

The inner product in (3) is normalised with respect to vectors' lengths, thus the final value does not depend on them. The closer to unity the more similar strings are. The computational cost of this algorithm is $\mathcal{O}(k+l)$ but this approach has one significant disadvantage — the similarity measure is not sensitive for ordering of terms.

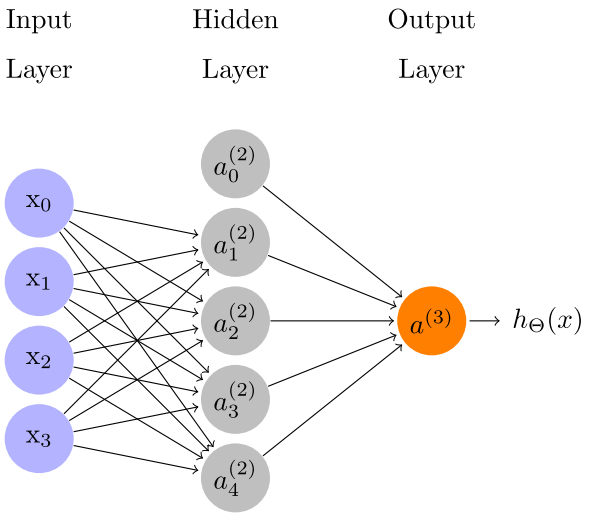

Fig. 1. A simplified NN architecture with one hidden layer and one output node.

2.2. Other algorithms

In the original research paper, next to the edit distance and the cosine similarity, we considered using the Dice coefficient [9] and the Longest Common Subsequence [10] algorithms. Dice measures the similarity between two strings as the intersection of the corresponding two sets of n-grams that make up two compared strings and in LCS the similarity measure is the length of the longest subsequence which is present in two strings. The disadvantage of the Dice coefficient is its insensitivity to the ordering of n-grams and the LCS is insensitive to a context. The two algorithms are described in more details in [1]. Both of them have as an advantage the computational time, which is $\mathcal{O}(k + l)$.

2.3. Neural networks

Machine learning systems whose architecture was inspired by the nervous system of the human brain are called Artificial Neural Networks (hereafter known as Neural Networks). Each NN is made of an input layer, output layer, and one or more hidden layers. A simplified Neural Network (NN) architecture is shown on Fig. 1.

Layers are made of so-called nodes, where each node in the input layer $a^{(0)}$ represent a feature from the training set (with the exception of the bias node x_0). Input nodes are connected with hidden layer nodes, and the transition from layer j to a layer $j+1$ is performed using weights Θ:

$$a^{(j+1)} = g(\Theta^{(j)}a^{(j)}), \quad (4)$$

where g is the activation function. The output layer gives us an activation of the last node, which is our desired answer. The NN is made of three layers, the input layer is made of one bias unit and 3 input nodes, the hidden layer has 4 nodes and one bias unit, and the output layer gives as an output an activation of the final node $h_\Theta(x) = a^{(3)}$.

The NN structure presented so far, where each subsequent layer can be connected to its predecessor and no single processing unit depends on its own output, is known as a *feedforward* network [11]. In case of more complex architectures with more

hidden layers, the architecture type determines possible connections between nodes and node types. A neural network that has internal loops which can memorise their internal state is called a Recurrent Neural Network (RNN) [12]. This feature makes RNN an very useful tool in studies of sequenced data (such as text, speech, or videos).

A multi-layer architecture that was designed to image recognition tasks is called Convolutional Neural Network (CNN). This is a type of a feedforward neural network that uses a mathematical function called convolution [13] to extract local features which later are combined to detect higher-order features with subsequent layers. With the growing popularity of word embeddings to represent words in a multidimensional vector-space, CNNs became a attractive method to extract high-level text features from their words or n-grams.

2.4. Related work

There are many applications of fuzzy string matching or string similarity algorithms, and NN techniques in text classification systems. Similarity algorithms have applications in many customer support environments. [14] proposed an automated labelling system for bug trackers and customer support based on their recurrent neural network solution, where the text is tokenised into vectors of words and sentences. [15] describes using a NLP based tool for a keyword extraction and mentions usage of the Levenshtein distance for word matching, yet the study focuses on the enhancement of the Machine Learning (ML) tagger with a Twitter model using previous customer service interactions. [16] uses word and character embeddings with neural models. They compare different linking methods with the fuzzy string matching, which computes the Levenshtein Distance between their queries using support tickets. [17] describe their ticket resolution system built using historical information of similar events. They propose two approaches to improve the traditional k-nearest neighbour (KNN) algorithm. First classify tickets as true or false and proposes resolution. The second enriches their KNN by enriching the algorithm with topic-level features, resolution information, top-level features and similarity measures from metric learning, where in case of textual features they use the Jaccard index for the bag of words model [7]. They improve their similarity measurements using event and resolution historical information. [18] showed that character-level CNN can be an efficient tool for text classification. [19] present a combination of convolutional and recurrent neural networks C-LSTM, where the first one phrase-level features and their sequences are later used in the long short-term memory network (LSTM). Machine Learning and string similarity has been widely used in the text classification system, yet there has been very little existing work comparing string similarity techniques or their configuration parameters when used in customer support automation together with neural network approaches as a second layer classification system.

3. Classification models

Our improved ticket classification model is divided into two stages. The first stage follows our original idea of the ticket classification based on the keyword search using a string similarity algorithm. Successive a ticket submission, ticket's text is scanned in search for a keyword match. The second stage is applied only on paid tickets and its goal is to assess the ticket's text as either informative enough or not to make a successful classification. Both stages are described in more detail in the following sections.

3.1. Keyword search based model

The model is made from a dataset of the most common ticket categories, where each category has a set of characteristic keywords. The ticket submission system allows customers to describe their enquiries in two main fields: subject and description (also called a ticket's body). The less popular way is to submit a ticket via the email channel, in this case the subject is the email's title and the description is its content. The subject should contain a short information of the existing difficulties thus it has a priority in the keyword search.

Categories included in this analysis are *Errors*, *Abuse*, *Crypto*, *DNS*, and *Account* (see [1] for descriptions). Their order is not accidental. Keyword search prioritises the most unique categories and rare keywords. The scan on the ticket's string is performed in windows of the keyword's length and when the match is found the algorithm stops the search. Once the category is set, the free customer gets a prepared response. In case of paying customers, only *DNS* and *Errors* tickets are automated and a safety check based on the binary classifier is applied.

3.2. The binary classification model

This model is based on a binary classifier that was trained using spaCy's [20] ensemble of a Convolutional Neural Network (CNN) and a bag-of-words model. We automate paid tickets for two categories (*DNS* and *Errors*), thus two separate classifiers are trained. Prior to the machine learning training, the primary keyword classification for each ticket has to be labelled by an agent either as *positive* (the keyword classification was correct) or *negative* (the keyword classifier misfired). The manual labelling is considered to be very conservative because in case of ambiguities in the ticket (e.g. one ticket was concerning more that one problem) the ticket is labelled as *negative*. As this is a binary problem, the default classification is based on a 50% score threshold. In our application we want to be even more conservative, thus every ticket with the greater score smaller than 80% is tagged as *unclassified*. As a result, paid tickets are automated if these conditions are met: the ticket was triggered by a keyword belonging to a given category (*DNS* or *Errors*) and the binary classifier's score for a *positive* label is greater than 80%. Fig. 2 depicts obtained confusion matrices before and after neglecting tickets marked as *unclassified*. The improvement achieved by the 80% requirement is clearly visible. The same behaviour can be seen on Fig. 3. Many tickets were misclassified as negative in the *DNS* category, this was to be expected as many of the problems relating to *DNS* are unique and difficult to predict, the 80% requirement lowered the number of tickets misclassified as positive from 240 to 73 but the price was the coverage of true positive matches. The same procedure for *Error* tickets reduced the number of tickets misclassified as positive from 410 to 230 with a small cost in true positive classifications. We consider this to be sufficient for our application.

4. Data sample and processing

The first dataset used in our analysis is made of automated tickets from our original research. Since tickets are changing with every new product release, the model also has to evolve. In order to be consistent with the original paper, we collected a second dataset of tickets corresponding to the same period of time (between March and November 2018). The second dataset was gathered with a restriction that the tickets should concern zones with paid plans. The ticketing system requires the support agent to adjust the classification to an accurate value prior to the ticket being closed. The distribution of categories chosen for this

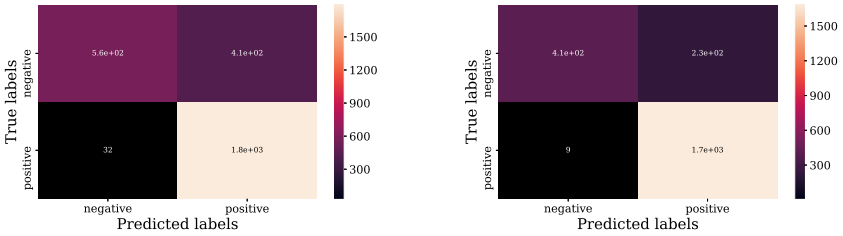

Fig. 2. Confusion matrices for *Errors* tickets using 50% (left) and 80% thresholds.

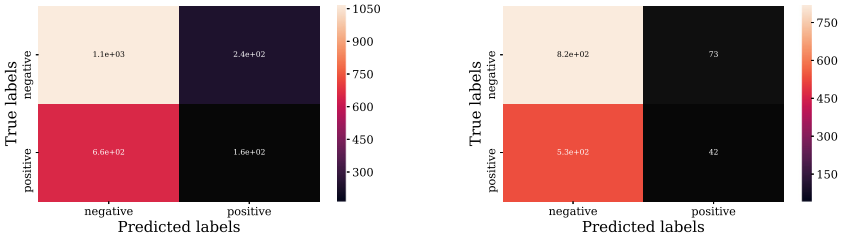

Fig. 3. Confusion matrices for *DNS* tickets using 50% (left) and 80% thresholds.

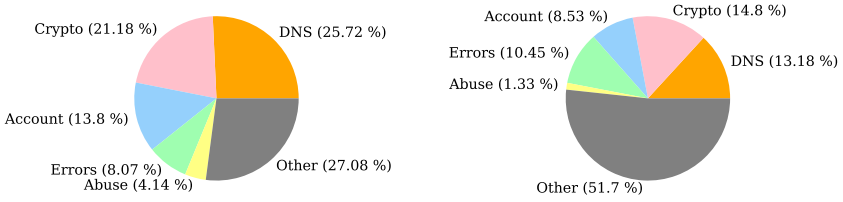

Fig. 4. The distribution of categories in closed tickets in free (left) [1] and paid (right) zones. The left plot corresponds to 26 195 tickets and the right plot corresponds to 38 884 tickets, all submitted between March and November 2018.

studies is given in Fig. 4. The coverage is higher on the first data sample (left chart) because this system was originally designed to automate tickets from free customers. The majority of paid tickets not covered by chosen categories belongs to: *Billing* (9.61%), *Firewall* (7.26%), *General Question* (5.01%), *Caching* (4.49%), and *Network* (3.58%).

Since the second layer classification is based on machine learning techniques, a comprehensive data sanitisation is crucial. Hence, prior to the binary classification, tickets have to be sanitised from technical information such as IP numbers, emails, URLs, WHOIS [21] lookup outputs etc. In addition we tuned spaCy's Name Entity Recogniser (NER) and used it for a footer extraction. In this procedure we look for entities such as names, geographical locations, numbers etc. in the tickets' description. A dummy example ticket with detected entities can be seen on Fig. 5.

With the tuned NER we expect an elevated named entity density at the end of a ticket's description in case of presence of a footer. We define the entity density as:

$$d_{\mathrm{NE}} = \frac{n_{\mathrm{NE}}}{n_{\mathrm{words}}}, \tag{5}$$

where n_{NE} stands for the number or named entities found and n_{words} stands for the total number of words in the analysed text (multi word entities are accounted as one). We start with the footer size set to a maximum of 5 lines, at the end of the ticket. The footer size is then decreased until the density reaches 30%, if this does not happen we assume that there is no footer at all. The 30% threshold was chosen using a named entity density distribution of a subset of 2313 tickets from the paid dataset, known to have footers. The clear separation in densities can be seen on Fig. 6.

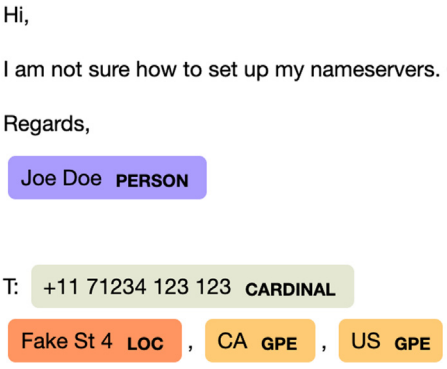

Fig. 5. Example ticket with fake data showing NER's application with entities detected.

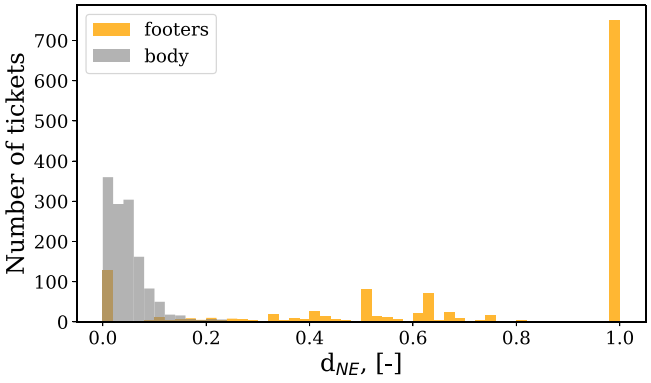

Fig. 6. The named entity density distribution in tickets' bodies and their footers.

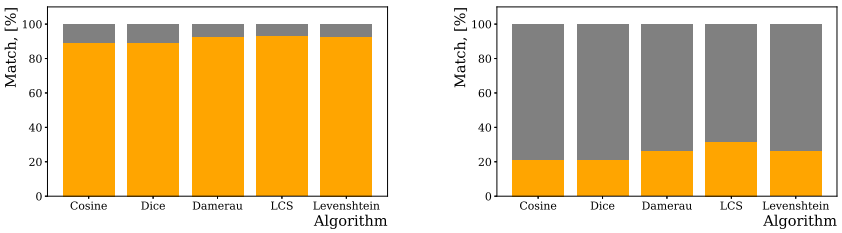

Fig. 7. The percentage of tickets triggered by keywords from the *DNS* category in the *DNS* (left) and *Crypto* (right) tickets [1]. Values correspond to 200 closed tickets from each category. For Cosine and Dice: *n*-gram size is equal to 2.

Table 1
Intercession points of TP and 1-FP for tickets triggered on keywords from the '*DNS*' category [1].

FP ticket category	t_S threshold
Crypto	0.87
Account	0.83
Abuse	0.74
Errors	0.83

Table 2
Execution time per character for chosen algorithms.

Algorithm	Execution time, [s]
Cosine	0.0775 ± 0.0019
Dice	0.0722 ± 0.0018
Damerau	0.6159 ± 0.0208
LCS	0.5460 ± 0.0193
Levenshtein	0.2525 ± 0.0085

5. Analysis

5.1. The keyword search classification

With the first dataset we measure the accuracy for chosen string similarity algorithms: Cosine, Dice, Damerau, Longest Common Subsequence, and Levenshtein (edit distance) [1]. In the accuracy measurements we look for keywords from the *DNS* category, and the search is performed on tickets from the two most common (see Fig. 4) categories: *DNS* (true positive (TP) matches) and *Crypto* (false positive (FP) matches). Two strings are matched if the minimum similarity threshold of 80% (or the maximum dissimilarity threshold of 20%) is fulfilled. In order to save the computing time, all five algorithms analyse only the first 200 characters of each ticket, and we randomly choose a small subsets of tickets (200 tickets for each category). The measurements are performed for different *n*-gram sizes for Cosine and Dice algorithms. These two algorithms are not sensitive for the ordering of words, hence the false positive rate for tickets matched with *n*-gram's of size $n = 1$ is not acceptable [1]. The reason for that is the fact that the algorithms sometimes can find a match with unrelated strings based only on their letter composition, e.g. *'dns modification pending'* would have 84.7% Cosine similarity with *'p and now the domain settin'*. This behaviour disappears when we increase the *n*-gram size by one. Using higher *n*-gram sizes does not introduce significant improvement, thus we chose to work with bigrams. The performance of all considered algorithms is shown on Fig. 7. The best (smallest) ratio of false positive to true positive matches (0.24), with the smallest false positive rate (21%) is observed for the Cosine algorithm, where the Dice algorithm had approximately the same FP/TP ratio with a higher false positive rate of 21.5%. The same ratio for the original algorithm (Levenshtein) is equal to 0.29, and the false positive rate is equal to 26.5%. Damerau and Damerau and LCS algorithms had false positive rates of 26.5%, 32.0% and 26.5%, with FP/TP ratios of 0.29 and 0.34, respectively [1].

In the next step, we perform threshold scans in order to check for a better minimum Cosine similarity (t_S) requirement. The point where the percentage of *'Crypto'* tickets not classified as *'DNS'* (1-FP) meets the TP ratio is equal to the best similarity threshold that distincts between the *'Crypto'* category ($t_{S_{Crypto}}$) from our most common category (*'DNS'*). The same *'DNS'* keywords searches are performed for other categories (see Table 1 for results). The obtained values are used to estimate the optimal working point as a weighted sum:

$$t_{S_{opt}} = \sum_i w_i \cdot t_{S_i} / \sum_i w_i, \tag{6}$$

where weights are equal to categories' contribution to the overall fraction of tickets (Fig. 4 (left)). As a result we obtain the optimal similarity threshold of $t_{S_{opt}} = 0.84$.

5.1.1. Execution time improvements

For a subset of matched *DNS* tickets' we measure execution times of the aforementioned algorithms. We use their keywords and their test strings known to have a match, to measure only the part of the programme which compares the similarity (omitting looping through a ticket to find a match). Our study shows that the Cosine and Dice algorithms are faster by an order of magnitude that Damerau, LCS and Levenshtein. The results are summarised in Table 2.

The median length of a ticket's body from this subset is 221 characters (prior to string sanitisation), where the maximum length is 20 026 and the minimum being 18. Assuming the above-mentioned median as a ticket length, and the worst case scenario, when no technical or unnecessary information (like an email footer) is found in a ticket and the keyword is at the end of the ticket's body; Damerau, LCS, and Levenshtein algorithms would have 136.11, 120.67, and 55.80 s ticket processing time, respectively. Under the same conditions, the improvement in execution times by using Cosine and Dice is clearly seen: the two algorithms would take less than 20 s to process an average ticket (17.13 s for Cosine and 15.96 s for Dice). Our requirements are that time spent on a single ticket's keyword search should not be greater than one minute. Cosine, Dice and Levenshtein algorithms fit our criteria, but the first two algorithms are more than three times faster than the Levenshtein. The difference between Dice and Cosine is very small, with Dice being faster, yet our previous studies shown better FP/TP ratio with smaller fraction of false positives for the Cosine algorithm (as described in Section 5.1). Thus, we believe that the Cosine algorithm is the best choice as it gives us a good accuracy with the smallest ratio of false positive matches for ticket classification and a small processing time.

5.2. Binary classification as a safety check

With the best working point estimated in previous steps we now consider application of our model on the tickets received

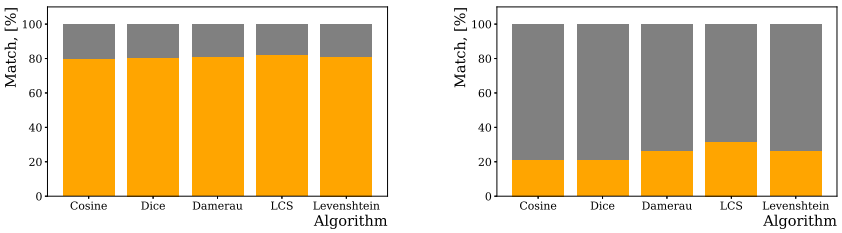

Fig. 8. The percentage of tickets triggered by keywords from the *DNS* category in the *DNS* (left) and *Crypto* (right) tickets. For Cosine and Dice: *n*-gram size is equal to 2.

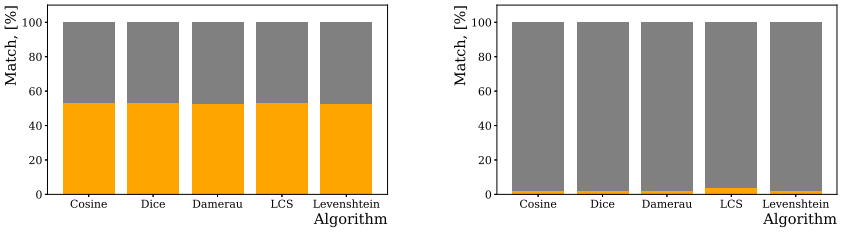

Fig. 9. The percentage of tickets triggered by keywords from the *Errors* category in the *Errors* (left) and *DNS* (right) tickets. For Cosine and Dice: *n*-gram size is equal to 2.

Table 3
True (TP) and false (FP) positive matches, and their ratio. *'DNS'* and *'Crypto'* tickets matched with the *'DNS'* category.

	Keyword classification			Binary classification		
Algorithm	TP, [%]	FP, [%]	FP/TP	TP, [%]	FP, [%]	FP/TP
Cosine	80.12	22.59	0.2820	25.54	1.72	0.0674
Dice	80.41	22.67	0.2820	25.54	1.72	0.0674
Damerau	81.09	25.03	0.3087	25.40	1.77	0.0697
LCS	82.46	28.21	0.3421	25.83	2.02	0.0780
Levenshtein	80.99	25.07	0.3095	25.44	1.77	0.0697

Table 4
True (TP) and false (FP) positive matches, and their ratio. 'Errors' and 'DNS' tickets matched for the 'Errors' category.

	Keyword classification			Binary classification		
Algorithm	TP, [%]	FP, [%]	FP/TP	TP, [%]	FP, [%]	FP/TP
Cosine	53.26	1.95	0.0366	36.82	0.49	0.0132
Dice	53.04	1.95	0.0368	36.71	0.49	0.0133
Damerau	52.72	2.24	0.0425	36.71	0.49	0.0133
LCS	53.15	3.7	0.0697	36.82	0.49	0.0132
Levenshtein	52.83	2.34	0.0443	36.71	0.49	0.0133

from paying customers. The reason for the distinction here is that the second group of customers has an access to a larger product range, thus their enquires may differentiate.

To verify our original studies, the same accuracy measurements are performed for *DNS* and *Errors* categories (we automate only server error codes), and their results can be seen on Figs. 8 and 9. Since the model was tuned to other types of tickets, we can see a drop in the percentage of tickets matched with the *DNS* category. Some of the missed customers' problems cannot be automated because they require a manual debugging from a human agent. Hence, expanding our model with new keywords is not a solution. In order to get the best possible false positive to true positive ratio, we introduce the binary classifier. The classifier is applied on top of the keyword classification. As mentioned in Section 3.2, the classifier is trained using labels supplied by our agents. The manual labelling was rigorous, thus after requirement on the binary tag to be *positive* we lose on the TP rate, but the TP/FP ratio gets better with a very small FP value. For all results see Tables 3 and 4.

It is important to notice that final values in Table 4 are approximately the same for all algorithms but we choose to stay with our choice of the Cosine similarity algorithm as it proved to be accurate and significantly faster than other algorithms (with an exception of the Dice algorithm). The keyword search step which uses the string similarity algorithm is shared between free and paid tickets. Free tickets do not have the binary classifier safety check (to maximise the classification coverage) thus we chose Cosine over Dice because the resulting FP/TP ratio is smaller for the Cosine algorithm (as described in Section 5.1). It is also important to mention that in the final version of our system for paid tickets we choose to modify the behaviour of our string similarity algorithm to further reduce the amount of misclassified tickets related to similarity being applied to numeric values. In case of keywords which contain numbers the requirement on 84% similarity might not be enough. More than 50% of keywords belonging to the *Errors* category includes number that can differ with only one digit, e.g. '*error 503*' could be match with a ticket containing '*error 502*' with the 85.71% cosine similarity. This is an important issue because these two error codes should trigger different automatic replies. For this reason we require that all keywords that contain digits have to be matched with the equality operator (as can be seen in the last *if* statement in Fig. 10). This forms the modified Cosine Similarity Algorithm, which accounts for the fact that numeric values require equality even where text components of the string should be matched with a fuzzy approach. With the improved Cosine algorithm and the second layer binary classification we obtained the true positive to false positive ratio of 0.0670 for the *'DNS'* category and 0.0132 for the *'Errors'* category.

6. Summary and conclusions

This paper presents a novel approach for using binary classification as a mechanism for safety engineering with fuzzy string matching algorithms together with comparison of various chosen algorithms using a keyword search approach. The false positive to true positive ratios have been measured using two layers of classification. The measurements have been enriched in the execution time comparison for the string similarity algorithms. We have estimated an optimal configuration for the Cosine similarity approach, tuned to our most frequent ticket category among free customers. The improvement was estimated using the aforementioned ratio, where false positive matches were taken from

INPUT: strings X, Y, threshold T

OUTPUT: boolean R

$G_X \leftarrow n\text{-}gram(X)$ ▷ Dictionary: n-gram occurrences in text

$G_Y \leftarrow n\text{-}gram(Y)$ ▷ Dictionary: n-gram occurrences in keyword

$C \leftarrow 0$ ▷ Cosine similarity value

$R \leftarrow False$ ▷ Result

for all unique K **in** G_X and G_Y **do** ▷ K is a dictionary key (n-gram)

$C \leftarrow C + G_X[K] \cdot G_Y[K]$

end for

$L_X \leftarrow 0$

$L_Y \leftarrow 0$

for all values V **in** G_X **do**

$L_X \leftarrow L_X + V^2$

end for

for all values V **in** G_Y **do**

$L_Y \leftarrow L_Y + V^2$

end for

$C \leftarrow C/(\sqrt{L_X} \cdot \sqrt{L_Y})$ ▷ Normalisation

if $C \geq T$ **then**

$R \leftarrow True$

end if

if Y has digits **then**

$R \leftarrow X == Y$

end if

Fig. 10. The modified Cosine Similarity Algorithm of two non-empty strings X and Y.

our second most popular ticket category. We have achieved a performance improvement of a relative 15.0% for free tickets. The average execution time improvement per character is 0.0775 s. With the new, faster algorithm we are able to use the full information provided by a customer (without restricting the number of characters to be processed). We showed that the second layer classifier reduced significantly the false positive to true positive ratio: 0.0670 for the *'DNS'* category and 0.0132 for the *'Errors'* category. This corresponds to a relative ratio improvement of 78% and 69% for the *'DNS'* and *'Errors'* categories respectively. Our conclusions are the following:

1. n-gram based algorithms (with $n > 1$) like Dice and Cosine outperform Levenshtein, Damerau, and LCS algorithms;
2. a neural-network based binary classifier works well as a second layer safety check for ticket classification;
3. the decrease in false positives was greater than the decrease in true positive matches when the neural network based binary classifier is used.

Our innovative approach to a multi-layer classification system using the modified Cosine algorithm and a neural-network based binary classifier reduced a fraction false positive classifications to a negligible level. Whilst using a second layer classifier results in a drop in the classification coverage, we find it is an acceptable trade-off for a greater reduction in classification false positives.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Pikies M, Ali J. String similarity algorithms for a ticket classification system. In: 2019 6th international conference on control, decision and information technologies. 2019, p. 36–41. http://dx.doi.org/10.1109/CoDIT.2019.8820497.

[2] Ukkonen E. Approximate string-matching with Q-grams and maximal matches. Theoret Comput Sci 1992;92(1):191–211. http://dx.doi.org/10.1016/0304-3975(92)90143-4.

[3] Kondrak G. N-gram similarity and distance. In: Proceedings of the 12th international conference on string processing and information retrieval. Berlin, Heidelberg: Springer-Verlag; 2005, p. 115–26. http://dx.doi.org/10.1007/11575832_13.

[4] Mays E, Damerau FJ, Mercer RL. Context based spelling correction. Inf Process Manage 1991;27(5):517–22.

[5] Marzal A, Vidal E. Computation of normalized edit distance and applications. IEEE Trans Pattern Anal Mach Intell 1993;15(9):926–32. http://dx.doi.org/10.1109/34.232078.

[6] Yujian L, Bo L. A normalized levenshtein distance metric. IEEE Trans Pattern Anal Mach Intell 2007;29(6). http://dx.doi.org/10.1109/TPAMI.2007.1078.
[7] Manning CD, Raghavan P, Schütze H. Introduction to information retrieval. New York, NY, USA: Cambridge University Press; 2008.
[8] Salton G, Wong A, Yang CS. A vector space model for automatic indexing. Commun ACM 1975;18(11):613–20. http://dx.doi.org/10.1145/361219.361220.
[9] Brew C, McKelvie D. Word-pair extraction for lexicography. Proceedings of the 2nd international conference on new methods in language processing. 1996.
[10] Wagner RA, Fischer MJ. The string-to-string correction problem. J ACM 1974;21(1):168–73. http://dx.doi.org/10.1145/321796.321811.
[11] Nilsson NJ. Artificial intelligence: A new synthesis. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 1998.
[12] Graves A, Liwicki M, Fernández S, Bertolami R, Bunke H, Schmidhuber J. A novel connectionist system for unconstrained handwriting recognition. IEEE Trans Pattern Anal Mach Intell 2009;31(5):855–68. http://dx.doi.org/10.1109/TPAMI.2008.137.
[13] Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc IEEE 1998;86(11):2278–324. http://dx.doi.org/10.1109/5.726791.
[14] Lyubinets V, Boiko T, Nicholas D. Automated labeling of bugs and tickets using attention-based mechanisms in recurrent neural networks. In: 2018 IEEE Second International Conference on Data Stream Mining Processing (DSMP). 2018, p. 271–5. http://dx.doi.org/10.1109/DSMP.2018.8478511.
[15] Weerasooriya T, Perera N, Liyanage SR. A method to extract essential keywords from a tweet using NLP tools. In: 2016 sixteenth international conference on advances in ICT for emerging regions. 2016, p. 29–34. http://dx.doi.org/10.1109/ICTER.2016.7829895.
[16] Han R, Goh K, Sun A, Akbari M. Towards effective extraction and linking of software mentions from user-generated support tickets. 2018, p. 2263–71. http://dx.doi.org/10.1145/3269206.3272026.
[17] Zhou W, Tang L, Zeng C, Li T, Shwartz L, Ya. Grabarnik G. Resolution recommendation for event tickets in service management. IEEE Trans Netw Serv Manag 2016;13(4):954–67. http://dx.doi.org/10.1109/TNSM.2016.2587807.
[18] Zhang X, Zhao J, LeCun Y. Character-level convolutional networks for text classification. In: Proceedings of the 28th international conference on neural information processing systems - Volume 1. Cambridge, MA, USA: MIT Press; 2015, p. 649–57, URL http://dl.acm.org/citation.cfm?id=2969239.2969312.
[19] Zhou C, Sun C, Liu Z, Lau F. A c-LSTM neural network for text classification. 2015. arXiv:1511.08630.
[20] Honnibal M, Montani I. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. 2017, To appear.
[21] Daigle L. WHOIS protocol specification. RFC 3912, 2004, http://dx.doi.org/10.17487/RFC3912.