2019 6th International Conference on Control, Decision and
Information Technologies (CoDIT'19) | Paris, France / April 23-26, 2019

String similarity algorithms for a ticket classification system

Malgorzata Pikies[1] and Junade Ali[2]

Abstract— **Fuzzy string matching allows for close, but not exactly, matching strings to be compared and extracted from bodies of text. As such, they are useful in systems which automatically extract and process documents. We summarise and compare various existing algorithms for achieving string similarity measures: Longest Common Subsequence (LCS), Dice coefficient, Cosine Similarity, Levenshtein distance and Damerau distance. Based on previously classified customer support enquiries (tickets), we considered the effectiveness of different algorithms and configurations to automatically identify keywords of interest (such as error phrases, product names and warning messages) in instances where such key phrases are misspelled, copied incorrectly or are otherwise differently formed. An optimal algorithm selection is made based on novel studies of the aforementioned similarity measures on text strings tokenised into characters. Such analysis also allowed for an optimum similarity threshold to be identified for various categories of enquiries, to reduce mismatched strings whilst allowing optimal coverage of the correctly matched key phrases. This led to a 15% improvement in the ratio of false positives to true positive classifications over the existing approach used by a customer support system.**

I. Introduction

Maintaining a high customer satisfaction benchmark is one of the main priorities of every company, and a customer support team is often the primary frontier for customers to contact a business. In order to provide the best customer service, agents have to prioritise tickets, reply quickly and accurately. With a growing customer base, the average waiting time for a reply can elongate. Classifiers based on string matching algorithms can shorten a ticket response time, hence help with agents' performance and reduce costs of the business. In practice it can be accomplished by using an automatic classification system linked with a database of replies to the most frequent enquiries or run technical diagnostics based on the error information provided by a customer. A system like that can immediately (subject to the text processing time) reply to tickets, which can reduce workload of customer support agents.

The purpose of this paper is an introduction to mechanisms behind the chosen string similarity algorithms. Given two strings (sequences of characters) X and Y, the difficulty of finding a quantity to measure the relationship between them comes down to two things. One is finding the correct similarity function $S(X,Y)$ and the second is finding a threshold $t_{S/D}$. Based on these values, two strings can be classified as

- similar: $S(X,Y) \geq t_S$ or $D(X,Y) \leq t_D$,
- different: $S(X,Y) < t_S$ or $D(X,Y) > t_D$,

where $D(X,Y)$ is a string dissimilarity function.

II. Current knowledge

There are numerous examples of fuzzy string matching or string similarity algorithms being used in customer support environments for extracting relevant information. [1] proposed an automated labelling system for bug trackers and customer support. They describe their recurrent neural network solution, where the text is tokenised into vectors of words and sentences. [2] describes using a Natural Language Processing (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. [3] 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. Despite using approaches for string similarity, there has been very little existing work comparing string similarity techniques or their configuration parameters when used in customer support automation.

String classification systems has been studied to label and understand a variety of text strings. Prior to any string analysis one has to decide on the text tokenisation. A string of text can be divided into items, such as words, phrases, letters etc. Items can be used to create *n*-grams. A set of all strings of an integer length n, in a finite alphabet Σ is denoted by Σ^n. An *n*-gram (sometimes called a shingle or a *q*-gram) based on letters is simply an any string from Σ^n [5]. In practice, a sequence of *n*-grams is created from a text of interest (see Tab. I for examples). Once strings are divided into substrings, the measurement of their similarity is possible.

To the best of our knowledge, there exist a gap in the literature that we want to fill. This paper is the first one to compare performance of different string similarity algorithms for a keyword extraction using test strings tokenised into characters.

TABLE I

n-GRAM EXAMPLES FOR A STRING "ALAMAKOTA" TOKENISED INTO LETTERS.

Name	*n*	*n*-grams
unigram	1	(a, l, a, m, a, k, o, t, a)
bigram	2	(al, la, am, ma, ak, ko, ot, ta)
trigram	3	(ala, lam, ama, mak, ako, kot, ota)

[1]Malgorzata Pikies is with Cloudflare, London, United Kingdom malgorzata@cloudflare.com

[2]Junade Ali is with Cloudflare, London, United Kingdom junade@cloudflare.com

A. Longest Common Subsequence

The Longest Common Subsequence (LCS) problem concerns finding the longest subsequence which is present in two strings [6]. Formally, a sequence $(x_{i_1}x_{i_2}...x_{i_p})$ is a subsequence of $X = (x_1x_2...x_k)$ if $\forall_j \in \{1,...,p\} : i_j \in \{1,...,k\}$, where $i_1 < i_2 < ... < i_p$ is an increasing sequence of indices [4]. The LCS can be generalised into the *n*-gram similarity measure. The recursive *n*-gram similarity between X and $Y = (y_1y_2...y_l)$ is defined as [7]:

$$s_n(\Gamma_{k,l}) = max_{i,j}(s_n(\Gamma_{k-1,l}), s_n(\Gamma_{k,l-1}), s_n(\Gamma_{k-1,l-1}) + s_n(\Gamma^n_{k-n,l-n})). \quad (1)$$

$\Gamma^n_{i,j} = (x_{i+1}...x_{i+n}, y_{j+1}...y_{j+n})$ and $\Gamma_{i,j} = (x_1...x_i, y_1...y_j)$ are pairs of strings of length n, and i and j, respectively. If both strings contain exactly one *n*-gram:

$$s_n(\Gamma^n_{n,n}) = s_n(\Gamma^n_{0,0}) = \begin{cases} 1, & \text{if } \forall_{1\le u\le n} x_u = y_u \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

If one of the strings is empty then the similarity is equal to zero. When $n = 1$, the *n*-gram similarity problem becomes LCS [7].

The brute-force LCS algorithm, which checks every possible subsequence of X and compares it with Y has exponential time $\mathcal{O}(2^k \cdot l)$. There exist a tabular solution [8] to this problem (shown on Fig. 1) and its cost is $\mathcal{O}(k \cdot l)$.

The major flaw of the LCS similarity is its insensitivity to a context, because it only pays attention to characters that are part of the searched sequence. It can also be an advantage in case of common human spelling mistakes. In this paper, the measurement using the LCS algorithm is performed as a distance, where the longest common subsequence is normalised to the length of the longer string:

$$d_n = 1 - s_n/max(k,l). \quad (3)$$

B. Dice coefficient

The similarity of two strings X and Y can be quantified by the intersection of the corresponding two sets of *n*-grams.

```
INPUT: two strings $X$, $Y$
OUTPUT: $T(K,L)$
  $K \leftarrow length(X) + 1$
  $L \leftarrow length(Y) + 1$
  $T \leftarrow$ table $K \times L$ of zeros
  for $i \leftarrow 1, K$ do
      for $j \leftarrow 1, L$ do
          if $X(i-1) = Y(j-1)$ then
              $T(i,j) \leftarrow T(i-1,j-1) + 1$
          else
              $T(i,j) \leftarrow T(i,j-1), T(i-1,j)$
          end if
      end for
  end for
```

Fig. 1. Dynamic Programming implementation for the Longest Common Subsequence problem for two non-empty strings X and Y.

The context sensitive approach is using the Dice coefficient [7], which is the ratio:

$$J_D(X,Y) = 2 \cdot \frac{|n\text{-}grams(X) \cap n\text{-}grams(Y)|}{|n\text{-}grams(X)| + |n\text{-}grams(Y)|}, \quad (4)$$

where the nominator is the number of *n*-grams shared between X and Y, and the denominator is the total number of *n*-grams in both strings. E.g. in case of trigrams ($n = 3$), sets would be $P_X = \{x_1x_2x_3, x_2x_3x_4, .., x_{k-2}, x_{k-1}, x_k\}$ and $P_Y = \{y_1y_2y_3, y_2y_3y_4, ..., y_{l-2}y_{l-1}y_l\}$.

The solution to this problem is shown on Fig. 2. Sets of *n*-grams can be calculated by external function with cost $\mathcal{O}(k)$ for X and $\mathcal{O}(l)$ for Y. The total cost of the algorithm is $\mathcal{O}(k + l)$.

Depending on the size of *n*-grams, the algorithm might have problems finding similarity between strings that are alike. In addition to that, the order of *n*-grams is not relevant thus the similarity might be equal to unity for different strings with the same sets.

C. Cosine Similarity

The Cosine Similarity [9] is a common vector-based method, where each expression (e.g. it can be a word or a letter) is a separate dimension. A multi-dimensional space is created, and as a result each text can be represented as a vector in that space. The similarity between two vectors can be measured as a distance between them. However, this particular approach is slightly faulty because two similar strings can be labelled as different due to their vectors lengths [9]. A way to get rid of this effect is to normalise vectors to unity [10] as can be seen on Fig. 3. The way to do that is to simply calculate the cosine between these two vectors [9], defined as:

$$s(X,Y) = \frac{\vec{U}(X) \cdot \vec{V}(Y)}{|\vec{U}(X)||\vec{V}(Y)|} = cos\theta, \quad (5)$$

```
INPUT: two strings $X$, $Y$
OUTPUT: $2 \cdot I/L$
  $P_X \leftarrow n\text{-}grams(X)$
  $P_Y \leftarrow n\text{-}grams(Y)$
  $U \leftarrow$ empty set
  $I \leftarrow 0$
  $L \leftarrow length(P_X) + length(P_Y)$
  for all $P$ in $P_X$ do
      $U \leftarrow U \cup \{P\}$
  end for
  for all $P$ in $P_Y$ do
      $U \leftarrow U \cup \{P\}$
  end for
  for all $P$ in $U$ do
      if $P$ in $P_X$ and $P$ in $P_Y$ then
          $I \leftarrow I + 1$
      end if
  end for
```

Fig. 2. The similarity of two non-empty strings X and Y calculated as a Dice coefficient.

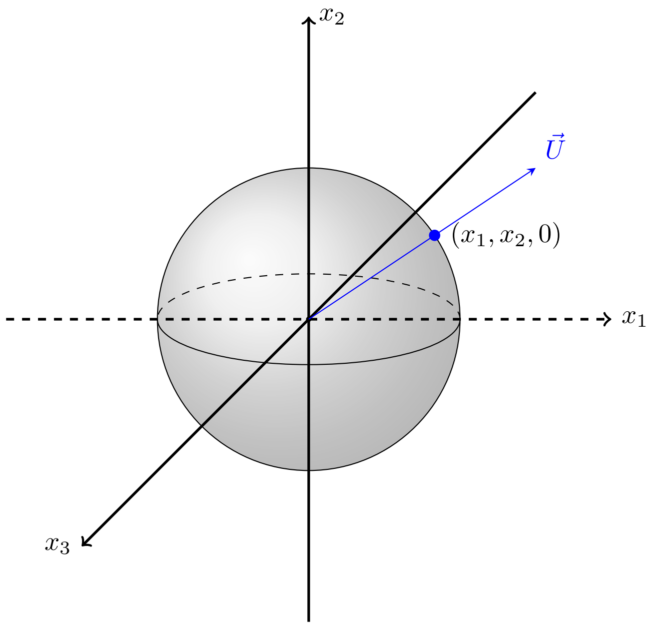

Fig. 3. A unit sphere as an envelope of the vector space and a vector $\vec{U}(X)$ normalised to the $[x_1, x_2, 0]$, where $\sqrt{x_1^2 + x_2^2} = 1$. The third component of the vector $\vec{U}$ is equal to zero for simplicity.

where θ is the angle between the two vectors corresponding to two strings. The inner product in (5) is normalised with respect to vectors' lengths, thus the final value does not depend on them.

The implementation of the algorithm dedicated to n-grams is shown on Fig. 4 and its cost is $\mathcal{O}(k + l)$.

The cosine function is equal to unity for two identical (unit) vectors, and -1 for two opposite vectors. While searching for the matching strings, one looks for the highest possible cosine similarity value. The biggest flaw of this algorithm is the fact it is not sensitive for ordering of terms.

```
INPUT: two strings $X$, $Y$
OUTPUT: $C/(\sqrt{L_X} \cdot \sqrt{L_Y})$
  $G_X \leftarrow profile(X)$
  $G_Y \leftarrow profile(Y)$
  $C \leftarrow 0$
  for all $V, M$ in $G_X$ do          ▷ If $G_X$ is shorter than $G_Y$
      $C \leftarrow C + V \cdot G_Y[M]$
  end for
  $L_X \leftarrow 0$
  $L_Y \leftarrow 0$
  for all $V$ in $G_X$ do
      $L_X \leftarrow L_X + V^2$
  end for
  for all $V$ in $G_Y$ do
      $L_Y \leftarrow L_Y + V^2$
  end for
```

Fig. 4. The Cosine Similarity of two non-empty strings X and Y.

D. The edit distance

The edit distance (Levenstein) is just a special case of the n-gram distance in case of $n = 1$ [7]. The difference between definitions of n-gram similarity and distance is very subtle. The minimum distance is defined as follows:

$$d_n(\Gamma_{k,l}) = min(d_n(\Gamma_{k-1,l}) + 1, d_n(\Gamma_{k,l-1}) + 1, \\ d_n(\Gamma_{k-1,l-1}) + d_n(\Gamma^n_{k-n,l-n})). \quad (6)$$

In case where both strings have exactly one n-gram, the distance is defined as:

$$d_n(\Gamma^n_{n,n}) = d_n(\Gamma^n_{0,0}) = \begin{cases} 1, & \text{if } \forall_{1 \leq u \leq n} x_u \neq y_u \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

When there is only one complete n-gram in either of the strings:

$$d_n(\Gamma_{k,l}) = 1 \text{ if}(k = l \wedge l < n) \vee (k < n \wedge l = n). \quad (8)$$

The problem is to find the minimum number of elementary edit operations, necessary to transform one string into another. Depending on the definition and treatment of elementary operations, the Levenstein distance has different variations.

In the classical edit distance problem, the elementary edit operation for two strings ($|X| \neq |Y|$) $X \rightarrow Y$ is defined as:

- 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 one X into Y.

The algorithm solving this problem is shown on Fig. 5. The cost of this algorithm is $\mathcal{O}(k \cdot l)$.

The number of edits does not always do us justice, as the distance is not normalised [11] (3 edits in a string made of 3 characters to the same number of edits in a string made of 10 characters). The distance can be normalised to its length L (the number of elementary edit operations), thus

```
INPUT: two strings $X$, $Y$
OUTPUT: $G_X(len(Y))$
  $G_X \leftarrow [0, 1, 2, ..., length(Y)]$
  $G_Y \leftarrow [0] \times length(Y)$
  for $I$ in $[0, 1, ..., length(X)]$ do
      $G_Y(0) \leftarrow I + 1$
      for $J$ in $[0, 1, ..., len(Y)]$ do
          $C \leftarrow 1$
          if X(I) = Y(J) then
              $C \leftarrow 0$
          end if
          $G_Y(J+1) \leftarrow min(G_Y(J)) + 1, G_X(J+1) + 1, G_X(J) + C)$
      end for
      $G_X, G_Y = G_Y, G_X$
  end for
```

Fig. 5. The Levenshtein distance of two non-empty strings X and Y.

the Normalised Levenshtein distance is the minimum value of the weighted sum over L [12]. It can also be normalised to the sum of sequences' lengths($|X|$ and $|Y|$) or using the length of the longest string as a normalisation constant, then it gets values from 0.0 to 1.0. Authors of [11] defined a normalised Levenstein distance for the transition $X \rightarrow Y$:

$$d_{norm}(\Gamma_{k,l}) = \frac{2 \cdot d_1(\Gamma_{k,l})}{\alpha \cdot (|X| + |Y|) + d_1(\Gamma_{k,l})}, \quad (9)$$

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.

When a transposition of two adjacent characters is considered as an addition to the above-mentioned edit operations, the distance is called the Damerau-Levenshtein distance [13].

III. Choosing an optimal algorithm

The Levenshtein algorithm was used in the existing classification system, with the distance normalised to the length of the string. When a ticket is submitted, our algorithm scans strings taken from ticket's subject (a title) and body (a message) in order to find a match with one of the keywords included in our model. A scan is performed in windows of the keyword's length. The maximum edit distance of 0.2 is allowed for the majority of keywords, exceptions are client and server error codes [14] where we require an exact match. When the match is found the algorithm stops the search. In order to save the computing time, the algorithm takes only the first 200 characters of the ticket. Our goal is to find a time efficient and accurate algorithm that performs as well as Levenshtein. The algorithm should allow us to process the full text from a ticket (without restricting it to 200 characters). Ideally we would like to obtain the best match rate with the smallest possible false positive match ratio (misclassified tickets). Hence, algorithms are compared based on their ability to properly classify tickets but not necessarily on the same keywords.

A. The model

With the variety of products, defining an accurate model which corresponds to the customers' most common issues is essential. Categories included in our model are listed below, together with short explanations and example keywords used to trigger them on tickets.

1) Errors: This category corresponds to two types of common HTTP status codes: client and server errors. Tickets that contribute to this category can be triggered if one of the following example keywords is found: 'error 403', '403 error', 'Forbidden', 'error 502', '502 error', 'Gateway Error', 'Internal Server Error' etc.

2) Abuse: This category includes tickets concerning banned domains and can be triggered for example by one of these keywords: 'code 1097', 'zone banned', 'error 1097'.

3) Crypto: Problems concerning encrypting traffic with SSL [15] go to this category. Example keywords are: 'ssl', 'redirect loop', 'mixed content', 'certificate', 'https'

4) DNS: Issues related to setting up domain name servers [16] are gathered in the DNS category and can be triggered by keywords like 'cname setup', 'website not active', 'dns pending', 'name server' etc.

5) Account: Customers having problems with their two factor authentication, submit tickets that are categorised as 'account' and usually include one of the following keywords: '2fa', 'two factor authentication', 'mfa', 'google authenticator'.

B. Dataset

The used dataset corresponds to closed English tickets submitted between March and November 2018. The ticket submission system allows customers to tag each question to a specific category and requires the support agent to adjust this to an accurate value prior to the ticket being closed. Categories included in our model contribute significantly to the overall pool of tickets, which can be seen on Fig. 6. Other categories include mostly general questions and billing issues, and some less popular categories which are going to be implemented into our model in the future.

C. Accuracy measurements

For each ticket from a given category five algorithms were tested. Algorithms were required to have at least 0.8 similarity (Dice, Cosine) or 0.2 distance (Levenshtein, Damerau, LCS). As mentioned previously, some of the keywords that belong to the 'Errors' category were required to have an exact match. These keywords include word 'error' together with an error number, e.g. 'error 500' or '500 error'. For Cosine and Dice algorithms, different *n*-gram sizes were tested (strings were divided into letters). A match means that either in the ticket's body or in its subject one keyword from the corresponding category in our model was found, but it was not necessarily the same word for each algorithm. Empty spaces were omitted for the calculation of Cosine and Dice similarities.

For the *n*-gram based algorithms, the results were suspiciously high (100 %) for the smallest *n*-gram size. It might seem like a desired result but in reality our model is not that perfect. As a cross-check the same measurements were done on a subset of tickets from the category 'Crypto' to test for instances where the classifier triggered on keywords belonging to the 'DNS' category. The investigation showed the Cosine and Dice algorithms found a match with

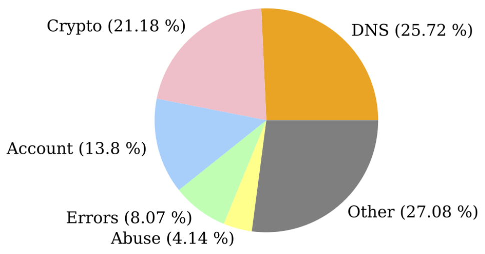

Fig. 6. The distribution of categories in closed tickets.

sometimes almost random words (see Tab. II and III for example results). A solution to this problem was adjusting the n-gram size. Fig. 7 shows performance of all considered algorithms, where for Dice and Cosine used strings divided into bigrams. With the change of the n-gram size to $n = 2$, the performance of the first two algorithms dropped to values closer to Levenshtein's result. With the same n-gram configuration, the ratio of false positives is smaller for the first two algorithms, that for the original one. The smallest and highest false positives ratios are observed for the Cosine and LCS algorithms, respectively (see Fig. 8). Hence, the LCS algorithm was excluded from the further analysis. Adding transpositions to the list of elementary operations did not improved the performance, thus we do not proceed with the Damerau algorithm. Increasing n-gram size gives an improvement of only relative 4.2 %, thus we do not examine higher values of n. A summary of measurements made for tickets from the two most popular categories is given in Tab. IV. Based on this measurement we decided to explore a performance of the Cosine similarity algorithm.

TABLE II
EXAMPLE 'DNS' MATCHES OF COSINE ALGORITHM, n-GRAM SIZE = 1. TICKETS WERE TAKEN FROM THE 'DNS' CATEGORY.

Keyword	String matched	Result, [%]
a record	s record	88.88
cname setup	please cont	83.33
a name	ve nam	84.51
dns	dns	100.00
dns modification pending	is records for do domain ni	80.01
name server	my name serv	80.40

TABLE III
EXAMPLE 'DNS' MATCHES OF COSINE ALGORITHM, n-GRAM SIZE = 1. TICKETS WERE TAKEN FROM THE 'CRYPTO' CATEGORY.

Keyword	String matched	Result, [%]
a name	ma gen	84.52
website not active	p to ssl isnt active	80.00
website not active	ted ssl certificat	80.00
website not active	to enable ssl certi	82.81
name server	e requireme	81.79
dns modification pending	p and now the domain settin	84.68

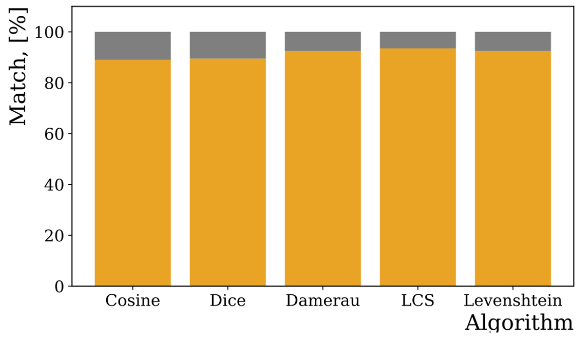

Fig. 7. The plot corresponds to a 200 closed tickets tagged as a 'DNS' category. Orange colour corresponds to enquiries matched as a DNS problem (true positives) and grey corresponds to unmatched tickets (false negatives). For Cosine and Dice: n-gram size is equal to 2.

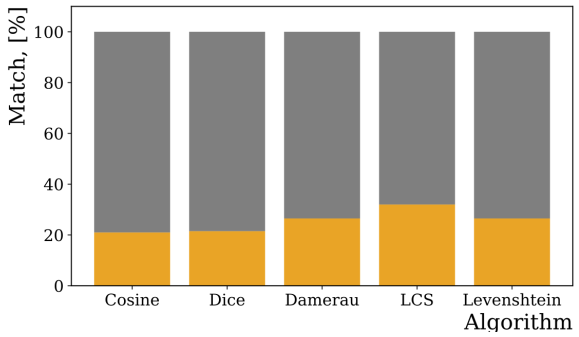

Fig. 8. The plot corresponds to a 200 closed tickets tagged as a 'Crypto' category. Orange colour corresponds to enquiries matched as a DNS problem (false positives) and grey corresponds to unmatched tickets (true negatives). For Cosine and Dice: n-gram size is equal to 2.

D. Choice of the best configuration

In this section we inspect the dependence of Cosine similarity accuracy on its threshold, for distinct 'DNS' false positive matches. We scan the true and false positives ratios as a function of t_S, where threshold can obtain values: 0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.67, 0.7, 0.71, 0.73, 0.75, 0.77, 0.79, 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.0. Fig. 9 shows that with the growing similarity threshold, ratio of true positive matches drops down. It is an expected behaviour because no model can completely categorise every user-generated text input. Simultaneously the percentage of misclassified tickets drops (1-FP grows). The density of the above-mentioned t_S values grows for higher thresholds because this area is interesting in terms of an intersection of this two functions. The point where these functions cross tells us what threshold gives us the smallest false positives ratio with the highest true positives ratio. The same measurements were performed for tickets from other categories included in our model. The obtained intersection points are reported in Tab. V.

The optimal threshold value was calculated as a weighted sum:

$$t_{S_{opt}} = \sum_i w_i \cdot t_{S_i} / \sum_i w_i, \qquad (10)$$

where the sum runs over categories reported in Tab. V and weights are equal to their contribution to the overall poll of

TABLE IV
TRUE (TP) AND FALSE (FP) POSITIVE MATCHES, AND THEIR RATIO. 'DNS' AND 'CRYPTO' TICKETS MATCHED FOR THE 'DNS' CATEGORY.

Algoritm	n	TP, [%]	FP, [%]	FP/TP
Cosine	1	100.0	99.0	0.99
	2	89.0	21.0	0.24
	3	81.5	19.0	0.23
Dice	1	100.0	98.0	0.98
	2	89.5	21.5	0.24
	3	88.0	20.0	0.23
Damerau		92.5	26.5	0.29
LCS		93.5	32.0	0.34
Levenshtein		92.5	26.5	0.29

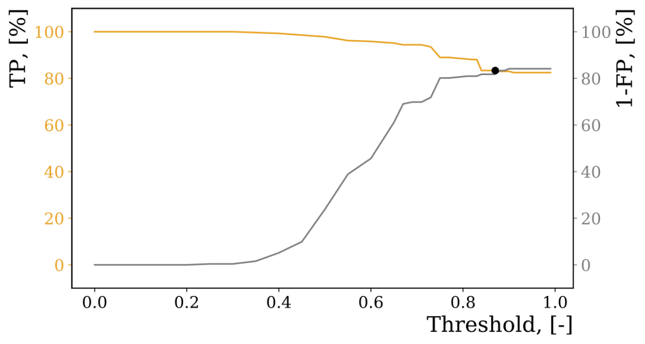

Fig. 9. The orange line corresponds to the true positives (TP) ratio and the grey one to the distance from unity of the false positives ratio for tickets matched as 'DNS'. The black dot marks the intersection point of these two lines. TP and FP were calculated using tickets from the 'DNS' and 'Crypto' categories, respectively.

TABLE V
INTERCESSION POINTS OF TP AND 1-FP FOR TICKETS TRIGGERED ON KEYWORDS FROM THE 'DNS' CATEGORY.

Ticket category	Threshold
Crypto	0.87
Account	0.83
Abuse	0.74
Errors	0.83

tickets (Fig. 6). As a result, the optimal similarity threshold is $t_{S_{opt}} = 0.84$. We calculated the FP/TP ratio using 'DNS' and 'Crypto' tickets triggered on the 'DNS' category with the Cosine similarity algorithm on text strings tokenised into sets of bigrams with threshold equal to $t_{S_{opt}}$. The resulting ratio is 24.4%, where in the original configuration it was equal to 28.7%.

IV. CONCLUSION

We have presented measurements and comparison of various chosen algorithms using a keyword search based classification system. We showed that with a simple tune of the algorithm's parameters it is possible to obtain satisfying results. We found the Cosine algorithm to have the best results in our use-case and by increasing *n*-gram size from 1 to 2, we found an approach more optimal to other considered. Additionally, the total cost of the similarity algorithm dropped from $\mathcal{O}(k \cdot l)$ (for the Levenshtein algorithm) to $\mathcal{O}(k + l)$. Since the optimal approach requires less rounds of processing, it allows for more computationally efficient processing of bigger text strings.

We have estimated an optimal configuration for the Cosine similarity approach, tuned to our most frequent ticket category. The improvement was estimated using a ratio of true positive and false positive matches using tickets from our two most popular categories. We have achieved a performance improvement of a relative 15.0%. Nevertheless, building an automatic key phrase extraction system is a process that requires continuous improvement. Automatic classification of support tickets is a complex problem and there is still much to explore.

REFERENCES

[1] Lyubinets, V., Boiko, T., Nicholas, D., “Automated labeling of bugs and tickets using attention-based mechanisms in recurrent neural networks,” 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP) pp. 271 - 275, August 2018

[2] Weerasooriya, T., Perera, N., Liyanage, S., “A method to extract essential keywords from a tweet using NLP tools,” 16th Interna-tional Conference on Advances in ICT for Emerging Regions (IC-Ter), pp. 29-34, September 2016

[3] Han, J., Goh, K. H., Sun, A., Akbari, M., “Towards Effective Extraction and Linking of Software Mentions from User-Generated Support Tickets,” Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM ’18, pp. 2263-2271, October 2018

[4] D’Angelo, J. P., West, D. B., “Mathematical Thinking: Problem-Solving and Proofs,” pp. 277-279, Prentice Hall, 1997, ISBN: 0130144126

[5] Ukkonen, E., “Approximate string-matching with q-grams and maximal matches,” Theoretical Computer Science, vol. 92, no. 1, pp. 191-211, January 1992

[6] Wagner, R. A., Fisher, M. J., “The string-to-string correction problem,” J. ACM, vol. 21, no. 1, pp. 168-173, January 1974

[7] Kondrak, G., “N-Gram Similarity and Distance,” Proceedings of the 12th international conference on String Processing and Information Retrieval, pp. 115-126, November, 2005

[8] Larsen., K. S., “‘Length of Maximal Common Subsequences,” DAIMI Report Series. 21., November 1998

[9] C. D. Manning, “‘Introduction to Information Retrieval,” pp. 109-134, Cambridge University Press, 2008, ISBN: 0521865719

[10] Salton, G., Wong, A., Yang, C.S, “A Vector Space Model for Automatic Indexing,” ACM, vol. 18, no. 11, pp. 613-620, November 1975

[11] Yujian, L., Bo, L. “A Normalized Levenshtein Distance Metric,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29 , no. 6 , pp. 1091-1095, June 2007

[12] Marzal, A., Vidal, E., “Computation of normalized edit distance and applications,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 15, no. 9, pp. 926-932, September 1993

[13] Mays., E., Damerau, F. J., Mercer, R. L., “‘Context Based Spelling Correction,” Information Processing and Management, vol. 27, no. 5, pp. 517-522, September 1991

[14] Fielding, R., Reschke, J., “‘Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content,” RFC 7231, June 2014

[15] Freier, A., Kralton, P., Kocher, P., “‘The Secure Sockets Layer (SSL) Protocol Version 3.0,” RFC 6101, August

[16] Mockapetris, P., “‘Domain names - concepts and facilities,” RFC 1034, November 1987