A Comparison between Textual Similarity Functions Applied to a Query Library for Inspectors of Extractive Industries

Junior Zilles[a], Jonata C. Wieczynski[b], Giancarlo Lucca[c] and Eduardo N. Borges[d]
Centro de Ciências Computacionais, Universidade Federal do Rio Grande, Av. Itália, km 8, 96203-900, Rio Grande, Brazil

Keywords: Textual Similarity, Engineering Inspection, RESTful API.

Abstract: The mineral extraction industries, including oil and gas extraction, use a series of equipment that requires several inspections before they are ready to be used. These inspections should only be carried by certified inspectors, since the equipment works under a lot of pressure and deals with toxic and dangerous materials for both human life and the environment, increasing the risk of accidents. In order to facilitate the search for qualified professionals with different certifying techniques, this article presents the construction of a RESTful API that implements a Web service for querying by inspectors, using textual similarity. The experiments included 48,374 inspectors, containing 74,134 certifications and 85,512 techniques. We evaluated the performance and quality of the system using a set of distinct similarity functions.

1 INTRODUCTION

One of the most critical activities in the Brazilian maritime industry is the inspection of welds, equipment, and works. This activity involves hiring professionals with diverse qualifications, including specific non-destructive testing techniques.

The inspection is an essential part of the process of integrity management, in addition to be a resource to monitor the performance of the structure and make sure about its safety (Carvalho et al., 2009). Welding and gluing must be continuously monitored and supervised at every stage of the application (Rogalski et al., 2019). The testing of the welding technology (the so-called qualification) and the certification of the personnel using the joining process allow these requirements to be met. For this reason, many subject standards and international and national regulations have been created that control these issues. This approach and behavior allow us to prove that the welding technology used guarantees the achievement of welded or brazed joints that meet specific or assumed acceptance criteria.

Consulting the enabled techniques of a specific inspector can be done checking the website of each certifying institution. These organizations publish information on certified professionals and qualified techniques in different HTML structures, and none of them has an Application Programming Interface (API) to retrieve the data easily. In addition, inspectors are identified by separate registration numbers, or keys, in each certifying agency. A company in need of such professionals would search for them on all these agencies. Also, queries are performed without considering more challenging searches. For instance, the name searched must have the same inner construction as the one saved in the institution's database. Variations in spelling or typos are not considered.

The idea for this project came from a Brazilian extraction company's need to validate, in a single and unified database, the records of construction projects and assembly of extraction and processing units. The digital records were extracted with an Optical Character Recognition (OCR) process, which may contain errors from the pattern detection. Therefore, this article presents a Representation State Transfer (REST) API for a web service that allows search of inspectors in multiple sources, i.e. certifying institutions. Additionally, the API is able to handle typos and spelling variations applying a set of effective and efficient textual similarity measures.

The text is organized as follows. Preliminary concepts and related work are introduced in sections 2 and 3. Section 4 specifies our methodology. Last sections present the results achieved and the conclusion.

[a] https://orcid.org/0000-0001-5748-4788
[b] https://orcid.org/0000-0002-8293-0126
[c] https://orcid.org/0000-0002-3776-0260
[d] https://orcid.org/0000-0003-1595-7676

Zilles, J., Wieczynski, J., Lucca, G. and Borges, E.
A Comparison between Textual Similarity Functions Applied to a Query Library for Inspectors of Extractive Industries.
DOI: 10.5220/0010437202000207
In *Proceedings of the 23rd International Conference on Enterprise Information Systems (ICEIS 2021) - Volume 1*, pages 200-207
ISBN: 978-989-758-509-8

2 PRELIMINARY CONCEPTS

Information retrieval (IR) (Baeza-Yates and Ribeiro-Neto, 2011) is a subarea of computer science that studies the storage and automatic retrieval of documents.Textual similarity functions (Cohen et al., 2003) are widely used in information retrieval systems to deal with typos, spelling variations, and different metadata patterns. The principal use is to reference the similarity of queries and documents, of which we want to retrieve.

Strings are lexically similar if they have a close character sequence. They can be semantically equivalent, i.e., similar based on meaning as opposed to form (Hahn and Heit, 2015). Different types of algorithms were proposed in the literature (Gomaa and Fahmy, 2013), such as Longest Common Substring (LCS), Levenshtein, Jaro-Winkler, n-gram, euclidean distance, Jaccard, cosine, and Latent Semantic (LSA).

Among the main components of an IR system, the indexer stands out. Its function is to summarize the collection of documents and speed up the search.The most common techniques for indexing are inverted and signature files (Zobel et al., 1998). Relational Database Management Systems (RDBMS) support to index columns. They use different types of indexing techniques, like generalized index search tree (GiST) (Hellerstein et al., 1995) and Generalized Inverted Index (GIN) (Borodin et al., 2018).

Although many different string similarity functions have been proposed in the literature, only a few allow the efficient calculation of textual similarity using the index. N-gram is an example of an indexable function implemented by several Database Systems.

The DBMS PostgreSQL provides a module named pg_trgm, which provides functions and operators for determining similarity of string based on 3-gram matching (Shresthaa and Behrb, 2011). It also supports indexes such as GiST and GIN for very fast similarity searches. The module fuzzystringstrmatch provides the Levenshtein distance (Levenshtein, 1966), which returns the minimum number of single-element edits required to change one string into the other. The edit operations are insertion, deletion, and substitution of a single element. To convert the Levenshtein Distance in the similarity range [0,1] we applied the equation (1), that normalize the distance using the maximum length of the compared strings.

$$sim(s_a, s_b) = 1 - \left(\frac{lev(s_a, s_b)}{\max(len(s_a), len(s_b))} \right) \quad (1)$$

Let ζ be an alphabet and $\Re$ be the set of real numbers in the closed range $[0,1]$. The trigram matching (Borges et al., 2012) is a function $similarity : \{\zeta \times \zeta\} \rightarrow \Re$, that receives as parameter two strings $s \in \zeta$ and returns a similarity score. Equation (2) defines the function as the ratio between the number of elements into the intersection between the sets of trigrams T_{sa}, T_{sb} extracted from parameters s_a, s_b and the union between the same sets. This is a commutative function, because $similarity(s_a, s_b) = similarity(s_b, s_a)$.

$$similarity(s_a, s_b) = \frac{|T_{s_a} \cap T_{s_b}|}{|T_{s_a} \cup T_{s_b}|} \quad (2)$$

The function *word similarity* returns a score that can be understood as the greatest similarity between the first string s_a and any of the n substrings of the second string $s_{b_i} | 1 \leq i \leq n$. Using this function, the number of additional characters present in the second string is not considered, except for the mismatched word boundaries. Equation (3) defines *word similarity*.

$$word\ similarity(s_a, s_b) = \frac{\max_i^n |T_{s_a} \cap T_{s_{b_i}}|}{|T_{sa}|} \quad (3)$$

The *strict word similarity* function makes use of the greater similarity in the same way as the *word similarity* function with a differential of taking into account the word boundaries. The score is divided by the intersection of the trigrams T_{sa} and T_{sb}. Equation (4) defines *strict word similarity*.

$$strict\ word\ similarity(s_a, s_b) = \frac{\max_i^n |T_{s_a} \cap T_{s_{b_i}}|}{|T_{s_a} \cup T_{s_b}|} \quad (4)$$

The quality of an Information Retrieval system can be evaluated using different measures. A Recall vs. Precision curve is often used. Equation (5) defines Recall as the fraction of relevant documents (True Positives - TP) that are successfully retrieved by the system. Relevant documents not retrieved are False Negatives (FN).

$$Recall = \frac{|\{docs_{rel}\} \cap \{docs_{ret}\}|}{|\{docs_{rel}\}|} = \frac{TP}{TP + FN} \quad (5)$$

Precision is defined by Eq. (6) as the fraction of retrieved documents that are, in fact, relevant. False Positives (FP) are non-relevant retrieved documents.

$$Precision = \frac{|\{docs_{rel}\} \cap \{docs_{ret}\}|}{|\{docs_{ret}\}|} = \frac{TP}{TP + FP} \quad (6)$$

The Mean Average Precision (MAP) (Cormack and Lynam, 2006) is one of the most widely-used metrics because it gives a single numerical value to represent system effectiveness (Turpin and Scholer, 2006). MAP can be defined by Eq. (7), where Q is the number of queries evaluated, and the $avgP$ is the average of precision of rankings defined by the positions of the relevant documents retrieved by a single query q. MAP is also often used for evaluating computer vision detection algorithms (Shanmugamani, 2018).

$$MAP = \frac{\sum_{q=1}^{Q} avgP(q)}{Q} \tag{7}$$

3 RELATED WORK

We were unable to find related work that deals explicitly with a library of industrial engineering inspectors. This was precisely the primary motivation of the work presented in this paper and shows our contribution to the naval and extractive industry. Despite that, several studies have similar proposals in distinct areas, providing users data from several heterogeneous sources requiring only a single query. We selected some studies that present a solution to share research data.

GNData (Sobolev et al., 2014) tries to solve problems when sharing research data in electrophysiology. One of the problems encountered by the authors is the difficulty of replicating research, as metadata is incomprehensible or not present. A second problem is sharing research data with other researchers. GNData overcame this by making available data management, an API, and client tools in most common languages. The system stores data and metadata together, making it easier for researchers to share.

Serpa et al. (Serpa et al., 2018) present a web services architecture that integrates oceanographic numerical modeling systems and autonomous computational systems. They highlight the need for independence and flexibility of the services, which correspond to three APIs with different processing steps. Mazzonetto et al. (Mazzonetto et al., 2017) address the problem of getting climatic data from the Center for Weather Forecast and Climate Studies of the Brazilian National Institute for Space Research (INPE). Data is stored in binary format inside the server, causing several issues. Researchers often need to make changes in the program that collects data to meet the users' queries. They proposed to develop an API to make the data available in an automated way.

Reisinger et al. (Reisinger et al., 2015) introduce the PRoteomics IDEntifications (PRIDE) Archive, a new version of the system developed to handle a massive workload since it has become a worldwide leader repository of mass spectrometry. PRIDE allows recovering peptide and protein identifications, project and assay metadata, and the originally submitted files. Searching and filtering are also possible by metadata information, such as sample details (e.g., species and tissues), instrumentation (mass spectrometer), keywords, and other provided annotations. PRIDE fully supports the storage of tandem mass spectrometry data (by far, the primary approach used in the field today). However, data coming from other proteomics workflows can also be stored (e.g., top-down proteomics or data-independent acquisition approaches).

Khan et al. (Khan and Mathelier, 2017) introduce the JASPAR RESTful API, a widely used open-access database of curated, non-redundant transcription factor binding profiles. The authors show their solution based on a RESTful API, which makes available data in different formats from an input query without relying on a specific programming language or platform.

Finally, Shresthaa and Behrb (Shresthaa and Behrb, 2011) describe a full text search functionality in Opengeocoding.org where they tested both PostgreSQL modules fuzzystringstrmatch and pg_trgm to build the functionality, of which they found that fuzzystringstrmatch functions weren't suitable to mistyping words, and pg_trgm functions weren't fast enough to their case which was to build a autocomplete full text search functionality.

As the related work presented in this section, the proposed solution aims to provide data, employing a web service, who can be easily accessible, allowing to build applications that can use it in a simplified way. We use some technologies in common with some of the works, which will be detailed in the subsequent section. The differential of our proposal is that the data comes from multiple heterogeneous industrial sources.

4 METHODOLOGY

This section presents our methodology to build the query library for inspectors of extractive industries. Firstly, we introduce the data model and the process of collecting and harvesting data. Then, the technologies used to implement the API in a microservice architecture are described. Finally, we present the experimental evaluation setup.

4.1 Modeling and Harvesting Data

The construction of the library of inspectors began by gathering the essential requirements and data sources: Brazilian Association of Non-Destructive Testing and Inspection (ABENDI)[1], Brazilian Corrosion Association (ABRACO)[2], Brazilian Welding Technology

[1]https://abendi.org.br/abendi

[2]https://abraco.org.br

Foundation (FBTS)[3], and American Petroleum Institute (APInst)[4]. From the analysis of these sources, we could observe that although they are different certification institutions, their data structure is quite similar. It describes the attributes of a certified inspector and the set of techniques in which he/she is qualified.

Figure 1 shows the integrated relational data model. We stored only the name of each inspector. An inspector certification includes the following attributes: certification *institution*, registration *number* (alphanumeric code), data *source*, and an optional *register* date. For each institution, an inspector can specialize in more than one technique using the same registration number. Each technique has the fields *due* date (expiration date of the specified technique), *type* and an optional *level*, which are stored only when the source is a databook (PDF document). We implemented this model using the PostgreSQL[5], since it has similarity functions already integrated, in addition to being the most advanced open-source relational database management system.

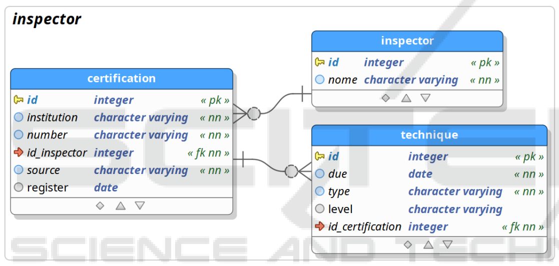

inspector
inspector
id integer « pk »
nome character varying « nn »
certification
id integer « pk »
institution character varying « nn »
number character varying « nn »
id_inspector integer « fk nn »
source character varying « nn »
register date
technique
id integer « pk »
due date « nn »
type character varying « nn »
level character varying
id_certification integer « fk nn »

Figure 1: Relational model of the proposed inspector library.

The next step was to create and populate the database with valid records, which could be used to test the API. We harvested from the certification institutions' web pages information about all inspectors (web scraping), and we also collected it from a set of PDF files available by an extractive industry company. Additionally about the data sources used in the experiments (ABENDI, ABRACO, FBTS, and APInst) are published by trusted institutions, so the data is not usually inconsistent. We have used simple pre-processing operations, such as: tokenization, removing accents, changing to lower case, and removing characters except alphanumeric. Figure 2 shows the flowchart of the algorithm, which was developed using Python using the libraries *beautifulsoup4*[6] for

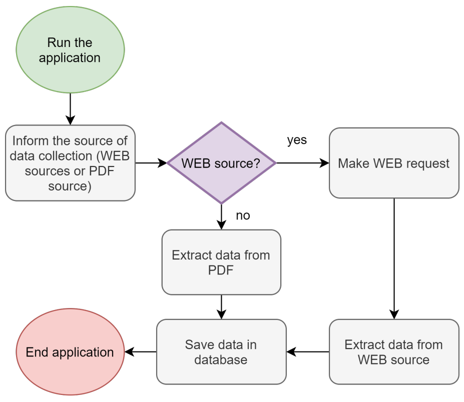


Figure 2: Flowchart of data collecting and harvesting.

handling HTML, *camelot-py*[7], which extracts tables from PDF files and transform them into *datatables*, and *pandas*[8] to handle those *datatables*.

4.2 RESTful API

The API allows users to query by name, registration number, or both, returning one or more inspectors. The results are sorted by similarity. We use the PostgreSQL fuzzystringstrmatch and pg_trgm module, the last of which provides a set of operators and similarity functions that compare strings with 3-gram matching (Ščavnický et al., 2018). This package was chosen because it allows indexing 3-grams with GiST, speeding up queries (Obe and Hsu, 2012). The database was chosen mainly because of its features and because other services from the microservice architecture also use the same Data Base Management System.

Consider the following example route as a query: ./inspector?name=mateus&registration=5827&limit=1&method=ws. The arguments are: name = "mateus"; registration number = 5827; limit = 1; and method = "ws".

Figure 3 presents the returned data in JSON format, containing a list of inspectors and their techniques, collected in the certifying organizations and the PDF files. The list is sorted by the name similarity score (line 2). For each data source, a PDF file or a website (line 9), a set of techniques is retrieved (lines 12-15).

The API was developed using *Node.js*[9] and *Swagger UI Express*[10] for creating standard documentation. Also, it is possible to perform tests on the API in the generated user interface.

[3]http://www.fbts.org.br
[4]https://www.api.org/
[5]https://www.postgresql.org
[6]https://pypi.org/project/beautifulsoup4/

[7]https://pypi.org/project/camelot-py/
[8]https://pypi.org/project/pandas/
[9]https://nodejs.org
[10]https://www.npmjs.com/package/swagger-ui-express

```
    "inspector": [  {
      "similarity": 0.53846157,
      "id": 6394,
      "name": "Mateus Lopes",
      "certifications": [  {
          "id": 6644,
          "number": "12238",
          "register": "2019-10-02
              T03:00:00.000Z",
          "source": "WEB",
          "institution": "ABENDI",
          "techniques": [  {
              "id": 12002,
              "due": "2024-09-30",
              "type": "LP-N2-G",
              "level": null    } ]
                 } ] } ]
```

Figure 3: Ranking of inspectors returned by a query.

Finally, we adapted the services to work independently, following the principles of microservice architecture.This task was done by using docker-compose[11] with services in three containers: our API, PostgreSQL, and pgAdmin4[12]. The API was also configured to accept requests from different sources by means of CORS. Because our API is a part of an architecture, we don't need to create an interface using technologies like Angular or React, since our main purpose is just to make a search tool available.

4.3 Experimental Evaluation

The proposed API's primary objective is to return queries of inspector names and/or registration numbers obtained from an OCR process. In order to evaluate the quality of the proposed API, we performed an experiment simulating the OCR errors. We built a set of queries sampling the inspector database and randomly replacing multiple characters in different positions of the first and last names. For each name, we generate two distinct queries, one using the function similarity another using word_similarity. The gold-standard dataset was composed of these queries and the expected correct result, i.e., the original name's identifiers in the four web sources and all PDF files. When there were homonyms, these identifiers also compose the result, since we do not consider the middle names when building the queries.

We varied the parameters: the size of the sample, the limit of documents retrieved in the ranking, the number of random replacements per token, and the similarity functions. Then, for each combination of the parameters, we have performed MAP. An averaged 11-point precision-recall curve across all queries was plotted concerning the best results for each similarity function. We also calculated the area under the curve (AUC). In the following section, we present all the parameter values used in the evaluation.

We performed the experiments on a computer with an Intel I5 8250U processor, 8 GB of RAM, and SSD storage. The application and API were run locally, while the PostgreSQL instance was loaded into a Docker container inside a VM running Ubuntu Server 18.04.

5 RESULTS

The API was developed with three base routes that made resources available through a GET request. The first one, ./docs-api, allows access to the documentation screen generated by Swagger. Figure 4 shows the routes available with the accepted HTTP methods. The second base route is ./inspector. This route is used to query for inspectors, given the name or the registration number, or both information. In addition to these parameters, it is possible to inform the limit of returned records and the similarity search method (see Table 1): substring (lk), equality (eq), levenshtein similarity (lv), defined by Eq.(1), 3-gram similarity (s), defined by Eq.(2), 3-gram word similarity (ws), defined by Eq.(3), or 3-gram strict word similarity (sws), defined by Eq.(4). These last three methods are lexical character-based similarity functions based on n-grams (see Section 2). The last route ./inspector/{id} is used to return an inspector from the database using his or her identifier.

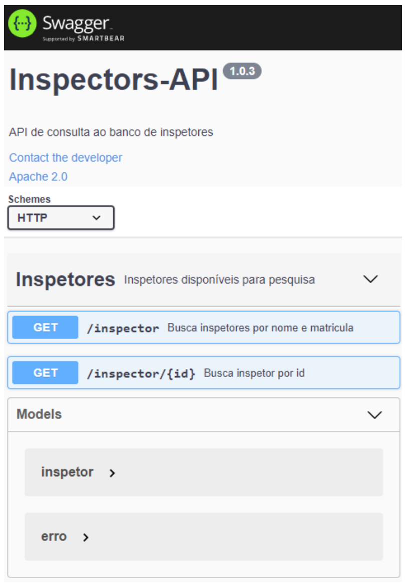
The database feeding algorithm resulted in the inclusion of $48,374$ inspectors, containing $74,134$ cer-

Table 1: API methods for querying inspectors.

Identifier	**Operation**
lk	Query using part of the name
eq	Query the name by equality, e.g. Name = John
lv	Query the name using *levenshtein*
s	Similarity query using the function *similarity**
ws	Similarity query using the function *word_similarity**
sws	Similarity query using the function *strict_word_similarity** *distance**

*Functions implemented by the modules pg_trgm and fuzzystrmatch from PostgreSQL.

[11]https://www.docker.com
[12]https://www.pgadmin.org


Swagger
Supported by SMARTBEAR

Inspectors-API 1.0.3

API de consulta ao banco de inspetores

Contact the developer
Apache 2.0

Schemes
HTTP

Inspetores Inspetores disponíveis para pesquisa

GET /inspector Busca inspetores por nome e matricula

GET /inspector/{id} Busca inspetor por id

Models

inspetor >

erro >

Figure 4: Swagger documentation screen.

tifications and 85.512 techniques. Table 2 shows the result of time execution of the extraction algorithm from each source. Rows are sorted by the column seconds per certification. The best result was achieved by ABRACO, which spent 0.5309 seconds per certification, followed by ABENDI, FBTS, PDF, and APInst. Getting data from API can be expensive because to get all names, we need to iterate through all vowels, which can bring us more than once the same inspector to handle again.

Table 2: Web scraping results.

Source	**Time(hours)**	**Included certifica-tions**	**Seconds per certi-fication**
ABRACO	0.1359	922	0.5309
ABENDI	2.0797	9,007	0.8312
FBTS	0.7970	2,765	1.0377
PDF	0.0225	56	1.4512
APInst	32.5245	61,384	1.9074
	$\sum =$ 35.5596	$\sum =$ 74,134	*avg* = 1.7267

The best results were achieved setting the parameters: size of the sample = 10,000; limit of documents retrieved in the ranking = 20; and number of random replacements per token = 1 and 2.

Table 3 summarizes the quality results. It shows, for each similarity function, the Mean Average Precision considering one (MAP_1) and two (MAP_2) character replacements per token. As we limited the rank to maximum of 20 documents it can be seen that the increase of wrong characters causes the decrease of the quality of the results. The best function was 3-gram similarity (*s*) which achieved MAP_1 = 0.8073, and MAP_2 = 0.5341. It was followed by strict word similarity (*sws*) with MAP_1 = 0.6486 and MAP_2 = 0.3771. Finally, word similarity and Levenshtein alternated 3rd and 4th place.

Table 3: Best MAP and AUC results for each similarity function, considering one and two character replacements per token.

Function	**MAP_1**	**MAP_2**	**AUC_1**	**AUC_2**
s	0.8073	0.5341	0.5786	0.2716
sws	0.6486	0.3771	0.4070	0.1699
ws	0.5661	0.3040	0.3169	0.1251
lv	0.3893	0.3298	0.1739	0.1423

Figure 5 shows, for each similarity function, the averaged 11-point precision-recall curve for the 10,000 queries with one random replacement per token. Analyzing the curves, we can see detailed results. All 3-gram based functions achieved the maximum recall of 90%. This behavior happens because we limited the API to retrieve 20 positions in the rank. Similarity (*s*) performed the best result dropping the precision to almost 78% to achieve 10% of recall. This precision was stable until it reaches Recall = 50%. After this point, the precision drops almost linearly. We can see an analogous behavior for strict_word_similarity (*sws*) and word_similarity (*ws*), where stable levels of precision, 58 and 47%, occurred up to Recall = 40 and 30% respectively. Levenshtein (*lv*) performed poorly, dropping the precision to a value lower than 30% to achieve 10% of the relevant documents. Besides, this function was able to retrieve only 70% of them. The area under the curves, for each similarity function, was reported in Table 3, column AUC_1.

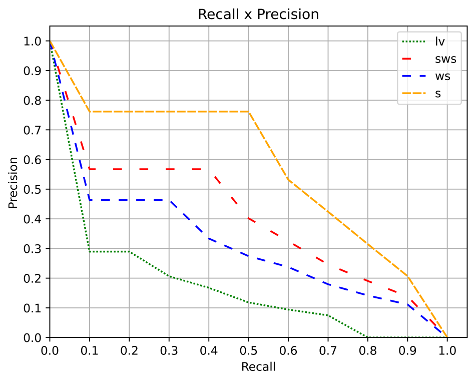


Figure 5: The averaged 11-point precision-recall curves for 10,000 queries comparing the functions, considering one character replacement per token.

When setting two character replacements per token, we obtained the results reported in Figure 6. In this case, levenshtein (*lv*), strict_word_similarity (*sws*) and similarity (*s*) two of which 3-gram based functions achieved the maximum recall of 70%. This behavior happens because we limited the API to retrieve 20 positions in the rank. In this experiment, similarity (*s*) only 3-gram similarity function (*s*) was able to satisfactorily rank the correct inspectors dropping the precision to almost 46% to achieve 10% of recall. This precision was stable until it reaches Recall = 30%. After this point, the precision drops almost linearly. We can see an analogous behavior for strict_word_similarity (*sws*), levenshtein (*lv*) and word_similarity (*ws*), where stable levels of precision, 29, 24 and 21%, occurred up to Recall = 20, 20 and 10% respectively. Word_similarity (*ws*) performed poorly, dropping the precision to a value nearly to 20% to achieve 10% of the relevant documents. Table 3 presents MAP_2 and AUC_2 scores. Considerable quality degradation can be seen when more than one character is incorrectly recognized by OCR. MAP decreased from 15% (*lv*) to 42% (*sws*) and AUC from 18% (*lv*) to 58% (*sws*).

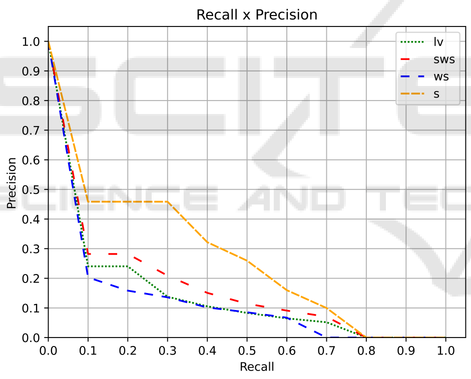


Figure 6: The averaged 11-point precision-recall curves for 10,000 queries comparing the functions, considering two character replacements per token.

In short, the 3-gram similarity function outperformed the well-know Levenshtein by up to 107% considering MAP and 232% considering AUC.

The evaluation program (API client) took approximately 27 hours to run the 10,000 queries. Table 4 shows the average processing time for each query and similarity function. It is possible to notice that there is no significant difference between max and min values in the average time of the evaluated 3-ngram functions. Levenshtein varied by 0.604 when compared to the faster function because it does not use indexing. This experiment also evaluated the client's processing time, which was almost irrelevant (≤ 0.0055 s) and associated with the network delay.

Table 4: Average processing time (seconds) for each query and similarity function.

Function	**Client***	**API**	**Difference**
s	1.0398	1.0357	0.0041
ws	1.0919	1.0878	0.0041
sws	1.0741	1.0687	0.0054
lv	1.6438	1,6383	0.0055
Difference	0.604	0.6026	0.0014

*Client program that performed the API requests.

6 CONCLUSION

In this paper, we presented the process of developing an API for an engineering inspector search library, motivated by a big company of extraction industry. The implemented web service allowed search of inspectors in multiple certifying institutions websites and from PDF files.

The experimental evaluation made us understand that the 3-ngram approach using the function similarity handles better for this case comparing to levenshtein distance, we acknowledge that there are other approaches like using Apache Solr or Elasticsearch as backend search engine that can in some way improve the request time. Additionally, with the increase of wrong letters on the searched query there will be a decrease in results that are relevant to the search. In our use case these results have shown to be enough to work with, but we are going to keep checking the results gathered when the API is going to face the real querys with OCR problems.

The experimental evaluation shows a comparison between four textual similarity functions. The results reported using precision-recall curves, MAP, and AUC make it clear that 3-gram similarity function can deal better in the simulated scenarios, therefore making this function a better choice for dealing with OCR interpreted words. The study showed that the requirements of handling variation in spelling and typos were fulfilled.

Future work will focus on including more features such as query by certified techniques, improving the exception handling, improving the container to allow database recovery automatically, deploying the API to the cloud, and using a search engine to discover if it is a better option. Also, a routine to update the collected data needs to be programmed since inspectors can renew their certifications or leave the office. Besides, new professionals can become certified. Finally, we are working on adding more data sources.

ACKNOWLEDGMENTS

This study was supported by CAPES Financial Code 001, PNPD/CAPES (464880/2019-00), CNPq (301618/2019-4), and FAPERGS (19/2551-0001279-9, 19/2551-0001660).

REFERENCES

Ščavnický, J., Karolyi, M., Růžičková, P., Pokorná, A., Harazim, H., Štourač, P., and Komenda, M. (2018). Pitfalls in users' evaluation of algorithms for text-based similarity detection in medical education. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 109–116, Poznan, Poland. IEEE.

Baeza-Yates, R. and Ribeiro-Neto, B. (2011). *Modern Information Retrieval: The Concepts and Technology behind Search*. Addison-Wesley Publishing Company, USA, 2nd edition.

Borges, E. N., Pereira, I. A., Tomasini, C., and Vargas, A. P. (2012). Argosearch: An information retrieval system based on text similarity and extensible relevance criteria. In *2012 31st International Conference of the Chilean Computer Science Society*, pages 133–141, Valparaiso, Chile. IEEE.

Borodin, A., Mirvoda, S., Porshnev, S., and Ponomareva, O. (2018). Improving generalized inverted index lock wait times. *Journal of Physics: Conference Series*, 944:012022.

Carvalho, A. A., Pereira, M. C., Bouchonneau, N., Farias, J. A. B., and Brito, J. L. F. (2009). Inspeção submarina: perspectivas e avanços. *Revista Tecnologia*, 30(2):198–209.

Cohen, W. W., Ravikumar, P., and Fienberg, S. E. (2003). A comparison of string distance metrics for name-matching tasks. In *Proceedings of the 2003 International Conference on Information Integration on the Web*, IIWEB'03, page 73–78, Acapulco, Mexico. AAAI Press.

Cormack, G. V. and Lynam, T. R. (2006). Statistical precision of information retrieval evaluation. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, page 533–540, New York, NY, USA. Association for Computing Machinery.

Gomaa, W. H. and Fahmy, A. A. (2013). A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18.

Hahn, U. and Heit, E. (2015). Semantic similarity, cognitive psychology of. In Wright, J. D., editor, *International Encyclopedia of the Social & Behavioral Sciences (Second Edition)*, pages 579 – 584. Elsevier, Oxford, second edition edition.

Hellerstein, J. M., Naughton, J. F., and Pfeffer, A. (1995). Generalized search trees for database systems. In Dayal, U., Gray, P. M. D., and Nishio, S., editors, *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases*, pages 562–573, Zurich, Switzerland. Morgan Kaufmann.

Khan, A. and Mathelier, A. (2017). JASPAR RESTful API: accessing JASPAR data from any programming language. *Bioinformatics*, 34(9):1612–1614.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8):707–710.

Mazzonetto, A., Borella, F., Chitolina, P., Tochetto, G., Casiraghi, J., de Oliveira, F. A. A., Chan, C. S., Pavan, W., and Holbig, C. A. (2017). Plataforma web para acesso e disponibilização de dados climáticos. In *Anais do VIII Workshop de Computação Aplicada a Gestão do Meio Ambiente e Recursos Naturais*, pages 805–808, Porto Alegre, RS. SBC.

Obe, R. and Hsu, L. (2012). *PostgreSQL - Up and Running: a Practical Guide to the Advanced Open Source Database*. O'Reilly, Sebastopol, CA, USA.

Reisinger, F., del Toro, N., Ternent, T., Hermjakob, H., and Vizcaíno, J. A. (2015). Introducing the PRIDE Archive RESTful web services. *Nucleic Acids Research*, 43(W1):W599–W604.

Rogalski, G., Landowski, M., Świerczyńska, A., Łabanowski, J., and Tomków, J. (2019). Qualification of brazing procedure for furnace brazing of austenitic steel according to requirements of the asme bpvc section ix. *Welding Technology Review*, 91(9):13–24.

Serpa, P., Weitzel, L., and Calado, L. (2018). Integração de sistemas de modelagem numérica oceanográfica e sistemas computacionais autônomos por meio de web service. *Revista de Sistemas de Informação da FSMA*, 1(22):26–34.

Shanmugamani, R. (2018). *Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras*. Packt Publishing Ltd, UK, 1st edition.

Shresthaa, S. and Behrb, F.-J. (2011). Implementation of full text search for opengeocoding.org. In *AGSE 2011 Applied Geoinformatics for Society and Environment Fourth International Summer School and Conference*, volume 1, pages 81–89, Nairobi, Kenya. AGSE.

Sobolev, A., Stoewer, A., Leonhardt, A., Rautenberg, P. L., Kellner, C. J., Garbers, C., and Wachtler, T. (2014). Integrated platform and api for electrophysiological data. *Frontiers in Neuroinformatics*, 8:32.

Turpin, A. and Scholer, F. (2006). User performance versus precision measures for simple search tasks. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, page 11–18, New York, NY, USA. Association for Computing Machinery.

Zobel, J., Moffat, A., and Ramamohanarao, K. (1998). Inverted files versus signature files for text indexing. *ACM Trans. Database Syst.*, 23(4):453–490.