

# Practical Token Retrieval and Indexing from Binary Data: An Application in Computer Aided Design<sup>†</sup>

M. Gruber\*, R. Geschray\* and C. Hillbrand\*

\* V-Research GmbH, 6850 Dormbirn, Austria

{Matthias.Gruber, Rainer.Geschray, Christian.Hillbrand}@v-research.at

**Abstract**— In many commercial applications proprietary file formats make it difficult to access the generated data. In the worst case interoperability is impeded even further by shortcomings in interface technology. The objective of this work is to find out whether it is possible to retrieve textual data from certain binary files in a quality which is sufficient to build a useful index.

We propose a method to parse and filter binary data in multiple stages. Besides stop-words, we use whitelists and phonetic as well as phonotactic criteria to create token data while minimizing noise. The results are promising: with a few simple steps we are able to filter most of the invalid tokens while preserving abbreviations and terms like company names even though they are not in a dictionary.

## I. INTRODUCTION

The objective of this paper is to find out whether it is possible to retrieve textual data from certain binary files in a quality which is sufficient to build a useful index.

Many commercial applications store their data in a binary format which is not easily accessible to other software, often in spite of standardized file formats being available. This makes it hard to provide features which are not present in the manufacturer's software, like indexing of a large collection of documents. In the long term this may limit the market value of the software and can lead to data becoming inaccessible due to changes in the software or due to discontinuation of products or companies.

Fortunately, there are a number of ways to overcome this issue, especially if the data files are created using a method which adheres to one of the international standards governing the binary serialization of data [1], [2]. Also, text can easily be extracted if it was encoded to a well known string encoding format [3], [4]. But even if none of the aforementioned preconditions are fulfilled, it is possible to retrieve data from the files as long as some form of octet-aligned encoding was used. However, it must be considered that even if a string is octet-aligned and readable, it does not mean that it is normalized as well. In this paper we are reporting on an experiment in which we retrieve textual information from such a binary data stream and study the quality of the index we created from the retrieved data.

The study is focused on CAD data files generated by the software MegaCAD from Megatech Software GmbH. The software stores its data in a binary format using the file extension “PRT”. Unfortunately the header does not contain human readable information except for the string “MegaCad20” and the company does not provide specifications to the file format.

An interface to directly access the file contents is not available either and the format does not adhere to one of the binary encoding rules in [1] or [2]. Furthermore, the string encoding does not adhere to a well known standard as in [3] or [4]. The only interfaces provided are via automation of the manufacturer's commercial application.

Besides the obvious disadvantages of having to load the manufacturer's application, there are two further problems: Firstly, a number of important text fields are left out from the export for no apparent reason. And secondly, only a limited set of commands is available, so that parts of the export process have to be realized via simulated keystrokes (“SendKeys”). This requires that the application remains in focus while the keystrokes are sent, which cannot be guaranteed in a multitasking environment. Additionally, there is no reliable way to automatically ensure that the application has correctly received and processed a simulated keystroke.

Our approach consists of two steps. First, we open the PRT files and search the binary data stream for strings, using a sliding stream window and a number of simple parsing rules and filters. Second, we feed the potential tokens to an algorithm, which evaluates the strings based on further criteria and rules. If a certain threshold is reached, the string is tokenized and added to an inverted index.

To evaluate our method we compare our results with an index created by exporting the text fields via automation of the manufacturer's application. Furthermore, we test sensitivity and specificity of our string evaluation algorithm using a German dictionary from and randomly created strings.

Our hypothesis is that using our method it is possible to create a useful index from certain kinds of binary data in spite of a number of false tokens being included. We would like to find out how good the quality of our separation method is.

---

<sup>†</sup> This work was financed by the Austrian funding scheme COMET (COMpetence centers for Excellent Technologies) within the K-Project “Integrated Decision Support Systems for Industrial Processes (ProDSS)”

The remainder of this paper is structured as follows: In section 2 we describe the process of identifying potential tokens within the binary data stream. Section 3 describes how the strings are evaluated and filtered. In section 4 we describe how the quality of the generated index was validated and evaluated. The results of our work are presented in Section 5. We conclude with a discussion of our results and directions for future work in Section 6.

## II. IDENTIFYING TEXT STRINGS WITHIN A BINARY DATA STREAM

We iterate the original binary data stream using a sliding stream window of 256 bytes size looking for a potential word start. All German letters (upper and lower case) are considered as word start candidates. When a byte from this list is found and followed by at least two more “legal character bytes”, we continue, otherwise we discard the investigated part of the window. In addition to the word start candidates, the list of “legal characters” includes numerals, punctuation, common ligatures and other signs common in text, like arithmetic symbols, the paragraph sign and a few more.

The algorithm ignores single non legal characters, so that the three mandatory legal characters may be interjected by other signs. But as soon as two or more subsequent non legal characters are found the string will be terminated. The remaining string will only be processed if, after removal of the non legal characters, it is still at least three characters long.

Obviously, that way we discard very short words like “es” or “auf”. But in the German language, these are usually stop words and if they should be included, there is a white-list mechanism for such cases.

The mechanism successfully identifies all strings within the given length constraints. This was verified by complete visual inspection of three of the test files using a hex editor. The method is independent of character encoding and has the advantage of being able to even identify strings in non standardized encodings. The disadvantage is that all strings are extracted. In our test data (see 4.1) we observed, that in most of the files at least 50% of the retrieved strings were without any meaning and therefore useless for creating an index.

## III. SEPARATING RANDOM STRINGS FROM VALUABLE TOKENS

Even though meaningless tokens do not prevent a classic full text search from working, they increase the size of the index unnecessarily. In most scenarios random strings can easily be filtered using probabilistic methods or dictionaries as in [13]. However, this was not an option in our case because a significant number of important search terms are company names and custom abbreviations. Therefore we propose a simple algorithm to separate random strings from valuable tokens.

### A. The Basic Algorithm

To filter meaningless strings from the parsed text data we use an algorithm to calculate a characteristic value. The basic formula

$$G = w_1 p + w_2 c + w_3 u$$

is applied to each potential token, where the  $w_i$  are weight terms on the interval [0,1]. The weight terms are normalized so that  $\sum w_i = 1$  prior to the calculation. The phonetic measure  $p$  is calculated as follows:

$$p = 2(v / k - 0.5). \quad (2)$$

The term  $v$  is the number of letters which can represent vowels in the German language (a, e, i, o, u, y) and  $k$  is the number of letters which represent consonants. The combined consonants “sch”, “ch” and “ck” are counted as one.

The casing measure  $c$  in formula 1 is

$$\begin{aligned} c &= 0, \text{ if } uc = 1 \text{ or } lc = 1 + t \text{ or} \\ &c = uc / n \text{ for all other cases,} \end{aligned} \quad (3)$$

with  $uc$  being the number of upper case letters,  $lc$  the number of lower case letters,  $n$  the word length and  $t$  a tolerance for lower case letters in otherwise upper case words. In our experiments  $t$  was set to 1.

The umlaut fraction  $u$  in formula 1 is simply the number of occurrences of the letters “ä”, “ö”, “ü” and “ß”. The latter is not an actual German umlaut but an equally rare character.

Finally, all non-white listed words with  $G$  (1) greater than or equal to a certain threshold are discarded. The threshold was chosen so that no valid tokens are discarded. This still resulted in a large number of false negatives, so that we had to incorporate further rules.

### B. The Extended Algorithm

The algorithm from chapter 3.1 was modified using two phonotactic filters. The first filter identifies invalid items and is applied before any further calculations are performed. It removes all items, which match any of the following criteria:

- The string has an invalid or highly improbable starting letter (“ß” and “y” in German).
- The string has an invalid or highly improbable ending letter (“q”, “c” and “y” in German).
- The string contains subsequent umlauts (like “ää” or “üö”).
- The string contains invalid or highly improbable letter combinations (like “q” without a subsequent “u” or “hh” in German)
- The string contains none of the vowels “a”, “e”, “i”, “o” or “u”.
- The string contains no consonants.

The second filter adjusts the  $G$  value (1) up or down with a constant in the range from -0.4 to 0.4 according to a list of probable German syllables which usually yield a high  $G$  value (like “maß”), and a list of improbable German letter combinations (like “hh” and “vv”). This yields a significant improvement, reducing the number of imported random strings by nearly 50% in our test data, without adding false positives.

These syllables and the according  $G$  value adjustment constants were manually selected from the false positives of the basic algorithm, so that the given tokens were correctly identified.

#### IV. VALIDATION AND EVALUATION

To evaluate our results we have manually created another index from the test data via export by the manufacturer's application. That way we were able to compare the performance of our algorithm in extracting textual data with the strings exported by the application itself. Furthermore we have used a dictionary containing 39.005 German words and a list of 39.005 randomly created strings. This was used to fine-tune the parameters so as to make sure that no valid German words would be discarded.

##### A. Test Files

The test sample consists of a number of MegaCAD files from the years 1994 to 2010 containing different kinds of drawings. From these drawings we have manually selected a representative sample of 26 files with respect to size, creation year and content type.

Our tests and comparisons were performed on the raw index (set A), as extracted by our algorithm, without consideration for stop words because the stop words can be changed or ignored completely in our search engine and because the stop word list depends on the actual field of application.

##### B. Comparison

The MegaCAD application allows the export of drawing data to the MS Access file format. A database created through this export contains a table "TEXTZEILE" which contains part of the text fields from the drawing. Upon completion of this paper we have not been able to find out why certain fields are left out. Using these, we have created a test index (set B).

While our algorithm identified 930 distinct tokens, the exported data in set B contains only 348. In total, our algorithm found 10.428 tokens versus 2.146 which were contained in the exported data. However, it has to be taken into consideration that our algorithm falsely identifies a number of random strings from the binary data as tokens. We have manually inspected the tokens and found 90 items which we believe to be without meaning and another 31 items which are neither German nor English words but could be abbreviations or company names. These 31 items do not occur in set B, though.

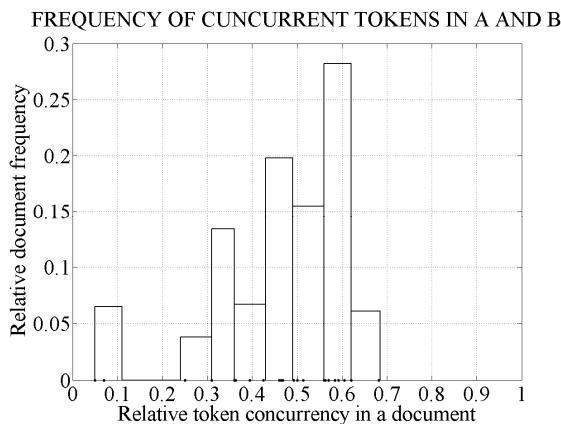


Figure 1. Relative occurrence of tokens in set A (retrieval) that can be found in set B (export). One sample corresponds to one document. The mean occurrence of tokens in A is 0.46, i.e. 46% of the tokens in A can be found in B.

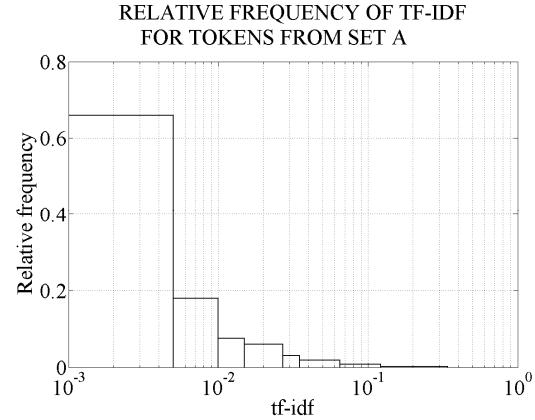


Figure 2. Figure 2. Normalized density estimation of the TF-IDF weight for tokens from set A. Be aware of the scaling difference between this figure and figure 3.

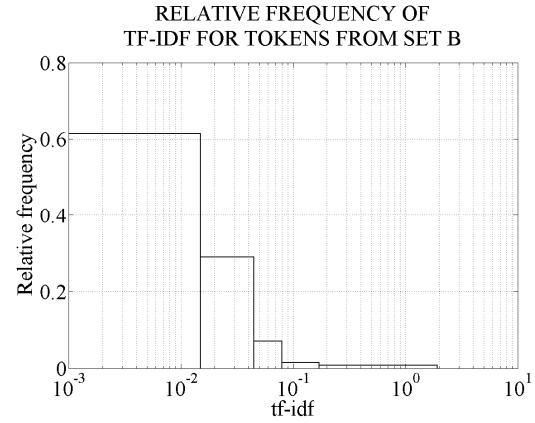


Figure 3. Figure 3. Normalized density estimation of the TF-IDF weight for tokens from set B. Be aware of the scaling difference between this figure and figure 2.

Fig. 2 and 3 illustrate the distributions of tokens between the two index files and the distribution of the TF-IDF weight as given in [5], [6], [7] and [8].

Direct comparison has revealed that a number of terms are missing from the index we created from the binary data. However, detailed investigation has shown that these are abbreviations and special terms. This can easily be solved using a white-list or additional parsing rules.

#### V. RESULTS AND INTERPRETATION

Throughout our work we have used the receiver operation characteristics method [9] to analyze the performance of our string filtering algorithm. The method was not used to calculate the ideal cut-off value, though. To be sure that no potential search term is filtered, we picked a much more conservative threshold. This was determined by estimation and iterative analysis of the confusion matrices for each of the weight terms in (1).

As test data for the ROC calculations we use a German dictionary containing 39.005 words and the same amount of randomly created strings. The length of the random strings is uniformly distributed on the interval [3, 8]. The letters of which the random strings consist are uniformly distributed on the German alphabet.

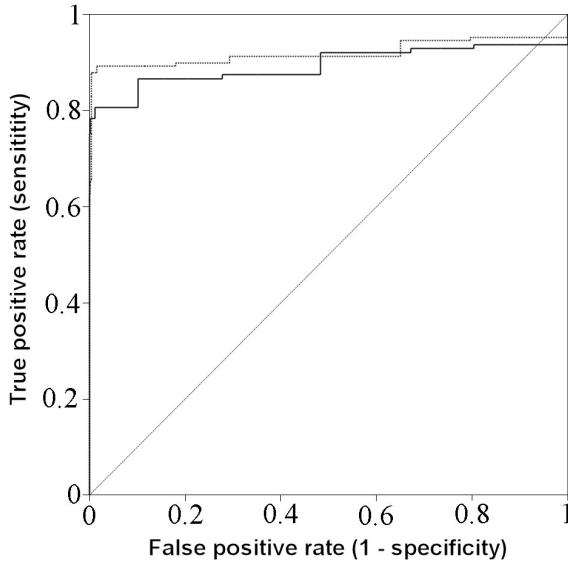


Figure 4. The continuous ROC curve shows the performance of the basic algorithm as given in III.A. The dotted curve shows the ROC of the modified algorithm (section III.B).

The curve in fig. 4 illustrates the performance of the basic algorithm and the extended version.

## VI. DISCUSSION AND FUTURE WORK

The results of this study show how token retrieval from binary CAD documents can be used for indexing the content of the textual data. Although there are still some minor drawbacks, this technique can be considered as a promising basis for building search applications as well as semantic systems on top of it. As a first area of application our technique will be used to perform full text searches on the contents of an archive of CAD documents in order to support the design process: A design engineer working on some kind of engineering task will specify a query, describing the context of his or her work to the search application. The application in turn will return existing CAD project files, satisfying the search attributes. The goal is to establish a similarity measure for attributes of a CAD project in order to retrieve valuable engineering knowledge implicitly embedded in the drawings. A highly versatile and quickly usable search engine on the textual contents of CAD data therefore promises an increase in efficiency of a design engineer's work due to the possibility of re-using engineering solutions of past projects.

The next step is to run further experiments on a larger set of data to further refine our method and the ranking of search results. Additionally, by integration of the search application into the customer's workflow, we will be able to analyze the distribution of search terms and the correlation between search terms and accepted results.

Furthermore, a text classification could help in narrowing the scope of a search for cases where too many results are generated. This could be achieved by common

machine learning methods as in [14], [15] and [16] or using a compression based method [17].

Finally, the customer will provide stop words and white-list items, which is going to influence the quality of the index and the resulting search results. These changes will have to be monitored carefully so as to avoid degradation.

## REFERENCES

- [1] International Organization for Standardization, *International Standard ISO/IEC 8825-1:2008: Information technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. Geneva, Switzerland: ISO, 2008.
- [2] International Organization for Standardization, *International Standard ISO/IEC 8825-2:2008: Information technology – ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER)*. Geneva, Switzerland: ISO, 2008.
- [3] International Organization for Standardization, *International Standard ISO/IEC 646:1991: Information technology - ISO 7-bit coded character set for information interchange*. Geneva, Switzerland: ISO, 1991.
- [4] The Unicode Consortium. *The Unicode Standard, Version 5.2.0, defined by: The Unicode Standard, Version 5.2*. Mountain View, CA: The Unicode Consortium, 2009.
- [5] K. Spärck Jones, “A statistical interpretation of term specificity and its application in retrieval,” *Journal of Documentation*, vol. 28, pp. 11–21, 1972.
- [6] G. Salton and M. J. McGill, *Introduction to modern information retrieval*. McGraw-Hill: Prentice Hall, 1983.
- [7] G. Salton, E.A. Fox and H. Wu, “Extended Boolean information retrieval,” *Communications of the ACM*, vol. 26, pp. 1022–1036, 1983.
- [8] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” *Information Processing & Management*, vol. 24, pp. 513–523, 1988.
- [9] D.M. Green and J.M. Swets, *Signal detection theory and psychophysics*. New York: Wiley, 1966.
- [10] T. Fawcett, “An introduction to ROC analysis,” *Pattern Recognition Letters*, vol. 27, pp. 861–874, 2006.
- [11] T.A. Lasko, J.G. Bhagwat, K.H. Zou and L. Ohno-Machado, “The use of receiver operating characteristic curves in biomedical informatics,” *Journal of Biomedical Informatics*, vol. 38, pp. 404–415, 2005.
- [12] K.H. Zou, A.J. O’Malley and L. Mauri, “Receiver-operating characteristic analysis for evaluating diagnostic tests and predictive models,” *Circulation* 6, vol. 115, pp. 654–657, 2007.
- [13] J.A. Zdziarski: *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification*. San Francisco, CA: No Starch Press, 2005.
- [14] T. Zhang, and F.J. Oles, “Text Categorization Based on Regularized Linear Classification Methods,” *Information retrieval*, vol. 4, pp. 5–31, 2001.
- [15] J. Zhang, R. Jin, Y. Yang and A.G. Hauptmann, “Modified Logistic Regression: An Approximation to SVM and Its Applications in Large-Scale Text Categorization,” *Proc. of the 20th International Conference on Machine Learning*, pp. 888–895, 2003.
- [16] F. Peng, D. Schuurmans and S. Wang, “Augmenting Naive Bayes classifiers with statistical language models,” *Information Retrieval*, vol. 7, pp. 317–345, 2004.
- [17] Y. Marton, N. Wu and L. Hellerstein, “On Compression-Based Text Classification,” in *Proceedings of the 27th European Conference on Information Retrieval (ECIR)*, pp 300-314, 2005.