Text based classification of companies in CrunchBase

Fernando Batista
INESC-ID
ISCTE – Instituto Universitário de Lisboa
Portugal
fernando.batista@iscte.pt

Joao Paulo Carvalho
INESC-ID
Instituto Superior Técnico - Universidade de Lisboa
Portugal
joao.carvalho@inesc-id.pt

Abstract— **This paper introduces two fuzzy fingerprint based text classification techniques that were successfully applied to automatically label companies from CrunchBase, based purely on their unstructured textual description. This is a real and very challenging problem due to the large set of possible labels (more than 40) and also to the fact that the textual descriptions do not have to abide by any criteria and are, therefore, extremely heterogeneous. Fuzzy fingerprints are a recently introduced technique that can be used for performing fast classification. They perform well in the presence of unbalanced datasets and can cope with a very large number of classes. In the paper, a comparison is performed against some of the best text classification techniques commonly used to address similar problems. When applied to the CrunchBase dataset, the fuzzy fingerprint based approach outperformed the other techniques.**

Keywords — Text classification; Fuzzy Fingerprints; Text Mining; Crunchbase; Document Classification;

I. Introduction

Text based classification is a relevant and difficult problem that has been around for a long time. It is a particular class of Text Mining problems that comprise numerous well-known Natural Language Processing (NLP) applications, such as: sentiment analysis [1][15], topic classification and detection [11][17], readability assessment, language identification, spam filtering, authorship identification [5][6], textual event detection [13], etc.

Many different classification techniques have been applied to text classification problems with varying degrees of success. The difficulty of a classification task varies across tasks and becomes substantially high as the number of categories/classes increase. Moreover, in multiclass text classification tasks, an increased number of classes demand larger sets of training data, and some of those classes will always be more difficult than others to classify. Reasons for that may be: i) few positive training examples for the class, and/or ii) lack of good predictive features for that class [9]. The existing literature concerning text classification is rich, but reported experiments using more than a few categories or classes are rather rare. The work of [11], focusing on topic classification, performs a classification of tweets into 18 general categories, such as *sports*, *politics*, *technology*, etc.; and uses TF-IDF weights together with Multinomial Naïve Bayes (MNB), achieving about 65% accuracy. Other literature dealing with large number of labels includes the work reported in [21] that introduces a new algorithm, based on *k* means, for multi-label classification in domains with large number of labels. However, the paper does not perform an extensive study of the proposed algorithm in real data. A considerable number of classes is also used in [20] where a system for tag suggestions for new blog posts based on the existing tagged posts is presented.

This paper introduces two fuzzy fingerprint based text classification techniques, and analyses their performance in automatically assigning one of 42 possible categories to a text. Fuzzy fingerprints are a recently introduced technique that can be used for classification. Fuzzy fingerprint methods are fast, perform well in the presence of unbalanced datasets and can cope with a very large number of classes [6][13][17]. The paper uses CrunchBase, an extensive database maintained by tens of thousands of active contributors, and currently a leading statistical resource for technology companies [3]. Each company in the CrunchBase dataset is described using unstructured text, ranging from a few words to entire pages, created by different contributors, and classified with one of 42 predefined categories. This is a real and very difficult problem, not only because of the lack of criteria in creating the texts, but also because it involves a very large number of categories. This paper compares the two proposed fuzzy fingerprint based classification techniques with several of the best text classification techniques, commonly used to address similar problems, and show that proposed techniques outperform the other techniques when applied to CrunchBase.

The paper is organized as follows: Section I describes the problem we are trying to address; Section II summarizes the most common methods used in text based classification and its applications; Section III introduces the Fuzzy Fingerprint method and its short text variants; Section IV introduces CrunchBase, the text database used in our experiments; Section V concerns experiments and results, and finally section VI presents conclusions and future work.

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under project PTDC/IVC-ESCT/4919/2012 and funds with reference UID/CEC/50021/2013.

II. Text Classification

For years, a wide range of methods has been applied to Text Classification problems, ranging from hand-coded rules to supervised and unsupervised machine learning. Some of the most well-known and commonly applied methods for text classification tasks [9] include: Naïve Bayes variants, k-Nearest Neighbor (kNN) [11], Logistic Regression or Maximum Entropy [1][11][22], Decision Trees [11], Neural Networks, and Support Vector Machines (SVM) [11][22]. Although many approaches have been proposed, automatic text classification is still an active area of research, mostly because existing text classifiers are still far from perfect in these tasks.

Naïve Bayes classifiers are a family of simple probabilistic classifiers based with strong independence assumptions between the features. Multinomial Naïve Bayes (MNB), a variant of Naïve Bayes that considers the frequency of words, is a popular and competitive method for text categorization. Being commonly used for text classification tasks [11][22], it can be denoted as:

$$p(c|d) \propto P(c) \prod_{1<k<n_d} P(t_k|c) \qquad (1)$$

where $p(c|d)$ is probability of a document d being in class c, $p(c)$ is the prior probability of a document belonging to c, and $P(t_k|c)$ is the probability of the term t_k occurring in class c. Several Naïve Bayes variants have been analyzed by [14] and [22], concluding that MNB is usually more stable and is usually the better choice.

Support Vector Machines are commonly used for text classification tasks [2][11][17][22]. An SVM classification is based on set of hyperplanes in a high-dimensional space. A good separation is achieved by the hyperplane that has the largest functional margin, the distance to the nearest training data point of any class. In general, a larger margin means a lower classifier generalization error. SVMs can efficiently perform linear and non-linear classifications using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. Usually SVM outperforms MNB in classification tasks, but that may depend on task and on the properties of data being used. In fact, several papers have been found to compare SVM with MNB in text classification tasks that conclude that MNB outperforms SVM and provide the best results [11][22]. Distinct analyses performed by [22] go deeper in such assessment and, using both review snippets and full-length reviews, show that SVM is better to full-reviews, but Naïve Bayes is more suitable for short snippet sentiment tasks.

Another common approach to text classification is to use example-based classifiers that do not explicitly build “declarative representations of categories”, but instead computes the similarity between the document to be classified and the training documents. *k*NN is an example-based classifier, commonly referred in the literature [2][9][12][17]. The representations of training data are simply stored together with their category labels. In order to decide whether a document *d* belongs to a category *c*, *k*NN checks if the *k* training documents most similar to *d* belong to *c*. If the answer is positive for a sufficiently large proportion of documents then the result is positive. An appropriate value of k is of the utmost importance. While *k*=1 can be too simplistic, as the decision is made according only to the nearest neighbor, a high value of *k* can lead to too much noise and favor dominant categories. kNN is known to be affected by noisy data, still it is considered one of the simplest and best performing text classifiers, whose main drawback is the relatively high computational cost of classification, because for each test document it must compute its similarity to all of the training documents. The training is fast, but the classification is slow because it implies computing all the similarities between a document that has not been categorized and the existing a collection of training documents [4]. The work by [2] reports kNN and SVMs to perform better than Latent Semantic Analysis (LSA) in text categorization tasks.

Alternative approaches to text classification include decision trees, logistic regression, and neural networks. According to [22], logistic regression gives results similar to SVM in what concerns text classification problems. Finally, neural networks have been regaining special attention during the recent years, motivated by recent discoveries in deep learning approaches [19]. Such approaches are now constantly beating existing state of the art results, but they usually require enormous quantities of data and require significant amount of time to build the models.

In what concerns the features, text classification tasks usually rely on unigrams, also known as bag-of-words representation [1][11][20][22]. Despite not being so commonly used for text classification tasks, a considerable amount of research using word bigram features can be found in the literature [1][22], but the use of higher order n-grams are rarely reported. Benefits of moving from unigram-based to bigram-based features depend on the tasks and are not consistent in the literature. For example, [22] reports small but consistent performance gains, while [1] reports a statistically insignificant decrease of performance for sentiment analysis. The very same papers report opposite, but statistically insignificant, results concerning a topic classification task: [1] reports a slightly increase of performance, while [22] reports a decrease of performance for an equivalent task. Such results suggest that word bigrams have limited utility for topic classification, perhaps because certain topic keywords are indicative alone [22].

The bag-of-words representation can be combined with different weighting schemes, including: i) binary or Bernouli weights that indicate whether a given word occurs in the document; ii) the words frequency that provide more information to the model; and iii) TF-IDF that is a scoring method that can measures the importance of a word or term in a collection of documents. TF-IDF is a powerful weighing schema, commonly reported in the literature to text classification tasks [17][18][20][22].

TF-IDF depends not only on the word/term frequency (*tf*), but also on the number of occurences of that term in the collection of documents, expressed by *idf* in eq. (2).

$$TFIDF = tf * idf \qquad (2)$$

Term frequency (*tf*) can be expressed simply by the number of occurrences of the word in the document, or by the logarithm of that count, thus penalizing very common words, such as pronouns and other functional words that occur frequently on a text. On the other hand, words that occur in few documents are probably richer in details that can better characterize the document. The inverse document frequency (*idf*) spans from the principle that a word that occurs in many documents is not relevant in differing each document from each other. *idf* can be obtained by dividing the total number of documents N by the number of documents t_n containing the term, and then taking the logarithm of that quotient, as expressed in:

$$idf = log\frac{N}{t_n} \qquad (3)$$

By combining Eqs. (2) and (3), the TF-IDF of a word in a document can be expressed by Eq. (4).

$$TFIDF = tf \times log\frac{N}{t_n} \qquad (4)$$

To summarize, TF-IDF provides a score to a term that is higher when it occurs many times within a small number of documents, and a lower when it occurs fewer times or in a larger number of documents. TF-IDF can be used together with a similarity measure to provide a simple way of performing text classification.

III. Fuzzy Fingerprints

Fingerprint identification is a well-known and widely documented technique in forensic sciences. In computer sciences a fingerprint is a procedure that maps an arbitrarily large data item (such as a computer file, or author set of texts) to a much compact information block, its fingerprint, that uniquely identifies the original data for all practical purposes, just as human fingerprints uniquely identify people for practical purposes.

In computer sciences, fingerprints are typically used to avoid the comparison and transmission of bulky data. For example, a web browser or proxy server can efficiently check if a remote file has been modified simply by fetching its fingerprint and comparing it with the fingerprint of the previously fetched copy. Fingerprints are a fast and compact way to identify items.

In order to serve for classification purposes, a fingerprint must be able to capture the identity of a given class. In other words, the probability of a collision, i.e., two classes yielding the same fingerprint, must be small.

For text classification purposes, we consider a set of texts associated with a given class to build the class fingerprint. Each word in each text represents a distinctive event in the process of building the class fingerprint. Distinct word frequencies are used as a proxy for the class associated with a specific text. The set of the fuzzy fingerprints of all classes is known as the fingerprint library. Given a fingerprint library and a text to be classified, we obtain the text fingerprint using a process similar to the one used to create the fingerprint of each class, and then find the class that has the most similar fingerprint. Details regarding fuzzy fingerprint creation and detection are given in the following sections.

A. Fuzzy Fingerprint Creation and Fuzzy Fingerprint Libraries

The full set of known texts (i.e. properly classified texts) are processed to compute the top-*k* feature list for each class. Consider F_j is the set of events of class j (simplistic example: the set of all words for all texts belonging to class j). The result consists of a list of k tuples $\{v_i, n_i\}$ where v_i is the i-th most frequent feature and n_i the corresponding count (simplistic example: an ordered k-sized list containing the most frequent distinct words). For extensive databases the top-*k* list can be approximated without any significant loss in accuracy in order to improve performance [7][8].

Then we fuzzify each top-*k* list in order to obtain the class fingerprint. The choice of the fuzzifying function is critical: the chosen approach is to assign a membership value to each feature in the set based only on the order in the list. The reason for using the order instead of the frequency results from empirical experiments that show that the order of the frequency seems more relevant than the frequency actual value [6]. The more frequent features will have a higher membership value. We tested for several alternative membership functions, and all results presented in this work use a function μ_{par} inspired in the Pareto rule, where roughly 80% of the membership value is assigned to first 20% elements in the ranking (5).

$$\mu_{par}(i) = \begin{cases} 1-(1-b)\dfrac{i}{k} & if\, i < a \\ a\left(1-\dfrac{i-a}{k-a}\right) & if\, i \geq a \end{cases} \qquad (5)$$

The fingerprint (Φ), which is based on the top-*k* list, consists on a size-*k* fuzzy vector where each position i contains an element v_i and a membership value μ_i representing the fuzzified value of v_i's rank (the membership of the rank).

A class j will be represented by its fingerprint $\Phi_j = \Phi(F_j)$. Formally, fingerprint $\Phi_j = \{(v_{ji}, \mu_{ji}) | i = 1..k_j\}$ has length k_j, with $S_j = \{v_{ji} | i = 1..k_j\}$ representing the set of v's in Φ_j. The set of all class fingerprints will constitute the fingerprint library.

B. Fuzzy Fingerprint Detection

In order to find the class of an unknown text T, we start by computing the size-*k* fingerprint of T, Φ_T. Then we compare the fingerprint of T with the fingerprints Φ_j of all classes present in the fingerprint library. The unknown text is classified as j if it has the most similar fingerprint to Φ_j. Fingerprint comparison, $sim(\Phi_T, \Phi_j)$, is calculated using (6):

$$sim(\Phi_T,\Phi_j) = \sum_{v \in S_T \cup S_j} \frac{\min\left(\mu_v(\Phi_T),\mu_v(\Phi_j)\right)}{k}, \qquad (6)$$

where $\mu_v(\Phi_x)$ is the membership value associated with the rank of element v in fingerprint x. This function is based on the fuzzy AND. In this case we use the minimum or Gōdel t-norm in accordance with [6], but other t-norms could also be used.

C. Small Texts Fuzzy Fingerprints

The presented fuzzy fingerprint method is not applicable to very small texts, such as for example, tweets, since the word frequencies in each text to be classified are not distinctive enough to create a fingerprint. In order to address such cases, an adapted fuzzy fingerprint based method was used for Twitter topic detection in [17]. Here we propose to extend the method to the classification of generic small texts.

1) Building the Small Texts Fingerprint Library

The main difference towards the regular fuzzy fingerprint method is based on the assumption that that due to the small size of each text, its features should be as unique as possible in order to make the fingerprints distinguishable amongst the various classes. Therefore, in addition to counting each feature occurrence, we also account for of its Inverse Class Frequency (icf), an adaptation of the well-known Inverse Document Frequency (idf) [18] where classes are used instead of documents to distinguish common feature occurrence:

$$icf_v = \frac{J}{J_v}, \qquad (7)$$

where J is the class fingerprint library size (i.e., the total number of classes), and J_v is the number of classes where feature v is present.

$tficf_v = n_v \times icf$, i.e., the product of the frequency of feature v with its inverse class frequency, is used to order the k-sized feature list.

After obtaining the top-k list for a given class, we use approach the approach described in the previous section to obtain the small text's fingerprint.

Once again, the fingerprint is a k sized bi-dimensional array containing in the first column the list of the top-k words, and in the second column its membership value $\mu_{ab}(i)$, obtained by applying (2).

2) Text-Class Similarity Score

In the previous method, in order to check the class of a given text, a fingerprint would be built for the text (using the procedure described above), and then the text fingerprint would be compared with each fingerprint present in the library. In the case of very small texts such approach does not work since it does not make sense to count the number of individual word occurrences in each text. Therefore we use a Text-Class Similarity Score (TCS2) that tests how much a text fits to a given class. The TCS2 score (8), does not take into account the size of the text to be classified (i.e., its number of features).

$$TCS2(T,\Phi_j) = \sum_v \mu_{\Phi_j}\left(v\right) : v \in \left(T \cap S\Phi_j\right) \qquad (8)$$

In (4), Φ_j is the class fingerprint, T is the preprocessed text, $S_{\Phi j}$ is the set of features of the class fingerprint, and $\mu_\Phi(v)$ is the membership degree of feature v in the class j fingerprint. Essentially, TCS2 sums the membership value of every feature v that is common between the text and the class fingerprint. Note that in [17] T would be the set of distinct words in the text, and the result was normalized based on the size of T. Such normalization does not occur here.

IV. Dataset

CrunchBase [3] is a public database containing information about start-up companies, people and investors. It started as a simple crowd-sourced database to track startups covered on TechCrunch, and has become a leading statistical resource for technology companies and transactions. The CrunchBase dataset contains about 650k profiles of people and companies and is maintained by tens of thousands of contributors. Descriptions consist of unstructured text, and all the information can be consulted using the CrunchBase API.

The corpus used for the experiments was collected using the CrunchBase API during April 2014. Each company is tagged with one of 42 possible categories and contains a textual description. The unstructured textual information may contain different types of information, ranging from products that a company sells, to its address, or even to the people involved. The description is usually written in English, but other languages are also being used. Moreover, some descriptions may include several languages, usually corresponding to translations. During a preprocessing stage we removed descriptions in languages other than English, and discarded descriptions with less than four words.

The resulting corpus contains about 119k companies and was randomly split into three different subsets: train (70%), development (15%) and test (15%). TABLE I shows statistics about each of the categories in the corpus. The table reveals that this is a very unbalanced corpus, with the two first categories corresponding to about 25% of all data, and the top 10 categories corresponding to almost 70% of all data. An important aspect about the categories is that the second most representative category (*Other*) is actually the most ambiguous data category.

Most of the descriptions contain a considerable number of words. However, about 10% contain less than 20 words. Fig. 1 shows a distribution of the number of words in the description of each company.

TABLE I. Number OF COMPANIES IN THE CORPUS

Category	**Training**	**Devel.**	**Test**	**Total**
Software	11968	2538	2585	17091
Other	8937	1838	1898	12673
Web	8556	1828	1804	12188
Ecommerce	5587	1213	1189	7989
Games video	4666	1054	978	6698
Mobile	4464	983	951	6398
Advertising	3833	827	863	5523
Biotech	3318	675	722	4715
Consulting	3205	706	701	4612
Enterprise	3017	683	644	4344
Education	2159	444	494	3097
Hardware	2102	406	465	2973
Health	1574	348	340	2262
Public relations	1483	336	308	2127
Network hosting	1486	323	305	2114
Finance	1358	330	283	1971
Clean tech	1368	285	294	1947
Search	1255	229	268	1752
Social	1180	272	249	1701
Local	1123	243	201	1567
Analytics	954	180	185	1319
Medical	931	192	193	1316
Security	852	189	183	1224
Travel	800	163	169	1132
Manufacturing	767	162	171	1100
Legal	685	160	154	999
News	650	150	162	962
Hospitality	650	131	164	945
Fashion	515	126	116	757
Sports	493	100	114	707
Real estate	506	102	97	705
Semiconductor	486	103	108	697
Photo video	485	107	84	676
Music	427	100	102	629
Transportation	446	92	86	624
Automotive	300	82	78	460
Messaging	299	71	69	439
Design	300	69	61	430
Nonprofit	200	38	28	266
Pets	84	15	17	116
Nanotech	71	12	20	103
Government	35	5	6	46
Total	**83575**	**17910**	**17909**	**119394**

V. EXPERIMENTS AND RESULTS

Different methods have been tested on the described dataset: ZeroR, which is a commonly used baseline [11] that consists in classifying all candidates using the most frequent category; TF-IDF and a similarity measure; two variants of Multinomial Naïve Bayes [14]; an SVM implementation based on SMO [10][16]; and finally the two versions of fuzzy fingerprints presented in Section III.

Experiments concerning Naïve Bayes and SMO were conducted using Weka version 3-6-8[1], a collection of open source machine learning algorithms and tools for data pre-processing and visualization.

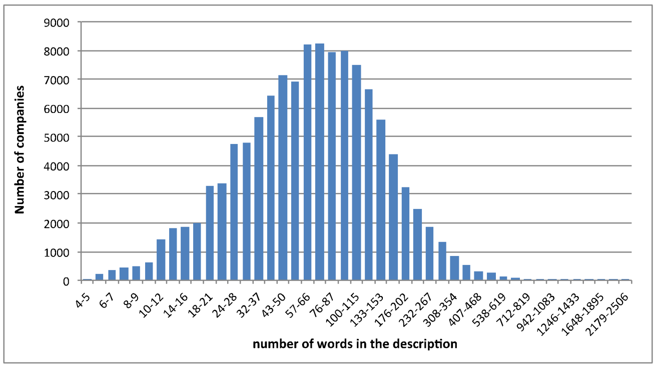


Fig. 1. Distribution of companies by the number of words in the corresponding description.

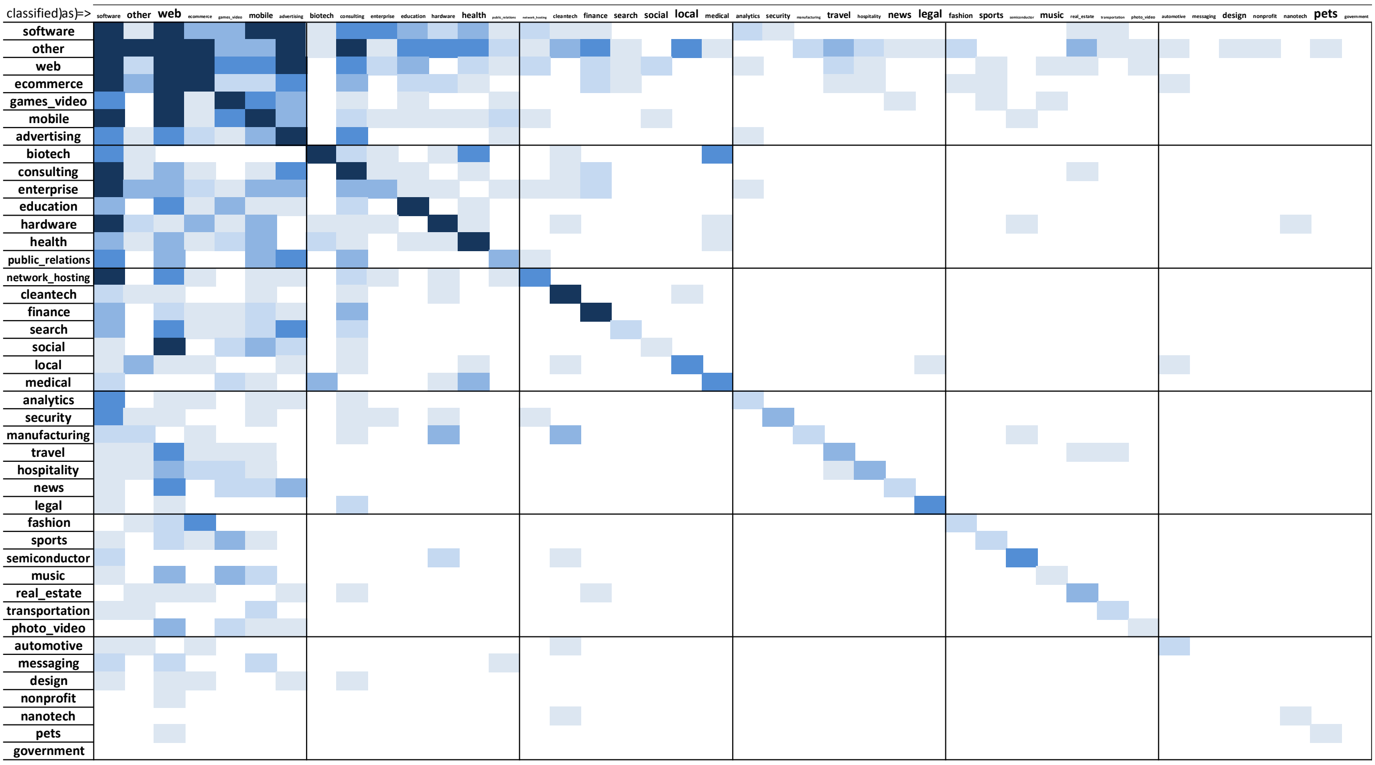
Each method was trained and optimized using the training and development sets, and the test set was used only after the ideal parameters for each method were found. We used word unigram based features only. The two variants of Multinomial Naïve Bayes do not support configuration parameters. SVMs were tested with different complexity parameters, but the best results were actually achieved using the default parameters, where the complexity level (C) is defined as 1.0 and the kernel is PolyKernel. Apart from the SMO implementation of SVMs, another SVM implementation based on the LIBSVM was also tested, but the performance was lower. In order to limit the computational complexity of the above models, especially the SVM models, the number of features was restricted to the 10K most relevant words, based of their TF-IDF. The same ARFF files (weka input files) were shared amongst the above methods, making experiments comparable. The TF-IDF weighting scheme does not use the same feature set, but we have set the minimum term frequency to 10, resulting into a similar number of features.

In what concerns the fuzzy fingerprints methods (FFP), the only relevant parameter is the size of the top-*k* list. Experiments were performed on the development set for k varying from 50 to 5000. The best results were obtained for *k*=4000 for both FFP approaches.

TABLE II. shows the performance (number of correct classifications and accuracy) for each of the methods, when using the 42 different company categories. As expected, given the large number of categories, the baseline has rather poor results. Both FFP versions give clearly the best results, with an improvement of around 10% over Multinomial Naïve Bayes, and 20% over both SVM and TF-IDF. Note that the features used in the FFP methods are necessarily different form the remaining methods, since each fingerprint contains the Top 4000 most relevant features for the corresponding class. The

[1] http://www.cs.waikato.ac.nz/ml/weka

TABLE III. CONFUSION MATRIX. DARKER TONES REPRESENT HIGHER NUMBERS



similar result for both fuzzy fingerprint versions is rather interesting, since both methods end up being quite different. The result can probably be explained by the large variation of the text descriptions. One of the methods performs better for the shorter descriptions, and the other for the larger descriptions. Since the distribution of the number of words used for the descriptions roughly approaches a normal distribution, the overall results end up being identical.

TABLE II.
CLASSIFICATION RESULTS USING 42 CATEGORIES (17910 EXAMPLES)

	#Correct	Accuracy
Baseline (most common)	2585	0.144
TF-IDF only	6146	0.343
Multinomial Naive Bayes	6760	0.377
Updatable Multinomial Naive Bayes	6761	0.378
SVM (based on SMO)	6097	0.340
FFP, K=4000	**7274**	**0.406**
Small texts FFP, K=4000	**7282**	**0.407**

TABLE III. shows the confusion matrix for the results achieved using our best method, based on fingerprints. The order of the columns follows the same order of the rows. The figure reveals that a considerable number of categories are being incorrectly classified as "software" (first column), the most common class. Other interesting result is the apparent confusion between "social" and "web". Finally the matrix also shows that many companies tagged as "other" are being classified as one of the other classes. Based on this fact, we tried to see how the methods would perform when removing the "other" category. Note that CrunchBase users choose "other" when they are undecided regarding what category to use. Results are presented in TABLE IV.

It is possible to see a considerable improvement in all categories even if the best to worse ranking is kept.

TABLE IV.
CLASSIFICATION RESULTS WITHOUT "OTHER" (16011 EXAMPLES)

	#Correct	Accuracy
Baseline (most common)	2585	0.162
TF-IDF only	5696	0.356
Multinomial Naive Bayes	6627	0.414
Updateable Multinomial Naive Bayes	6628	0.414
SVM (based on SMO)	5778	0.361
FFP, K=4000	**7099**	**0.443**
Small texts FFP, K=4000	**7093**	**0.443**

An additional experiment has been performed in order to check whether our results were being much influenced by the presence of specific mentions of the corresponding category within the texts. We have checked how many of our features were one of the 42 categories. In fact 14 of our features also correspond to category names. However, these features occur only 168 times in our test set, revealing that they have little influence in the results.

VI. Conclusions

Classification of unstructured texts is a very difficult problem, especially when the number of classes is high. This paper uses CrunchBase data to check how efficiently can a company be properly classified into one of the available 42 categories, based only on its unstructured textual description – which is provided by each company responsible without using any guidelines, and therefore is very heterogeneous both in size and contents. We tested the performance of several text classification methods, including the most popular techniques in the literature, and proposed to perform the classification using two distinct Fuzzy Fingerprints based methods, one more adapted to longer texts that was previously presented in [6], and a novel approach adapted to shorter texts.

Overall it was possible to find that both Fuzzy fingerprint based methods performed very similarly (probably due to the high variability in text size) and allowed for accuracy improvements between 10% and 20% when compared against what are usually considered as the best text classification methods.

Even if the results show a distinct advantage in the use of Fuzzy Fingerprints, overall accuracy values are still low, and there is room for improvement. Since both Fuzzy Fingerprint approaches performed similarly, one future development is to use an adaptive procedure that selects one of the methods based on the text characteristics, namely its size. Since one of the methods is more adapted to longer texts, and the other to shorter ones, this should bring accuracy improvements.

References

[1] Batista, F., Ribeiro, R., “Sentiment analysis and topic classification based on binary maximum entropy classifiers,” Procesamiento de Lenguaje Natural, vol. 50, no. 0, pp. 77–84, 2013.

[2] Cardoso-Cachopo, A., Oliveira, A., "An empirical comparison of text categorization methods." In *String processing and information retrieval*, pp. 183-196. Springer Berlin Heidelberg, 2003.

[3] Crunchbase, https://info.crunchbase.com/about/, last accessed on February, 10th, 2015.

[4] Feldman, R., Sanger, J., eds. *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge University Press, 2007.

[5] Fung, G., “The disputed federalist papers: SVM feature selection via concave minimization”, New York City, ACM Press, 2003..

[6] Homem N., Carvalho J.P., Authorship Identification and Author Fuzzy Fingerprints”, Proc. of the NAFIPS2011 - 30th Annual Conference of the North American Fuzzy Information Processing Society, 2011, IEEE Xplorer

[7] Homem, N. Carvalho, J.P., “Finding top-k elements in a time-sliding window” Evolving Systems, 2(1), pp. 51-71, Jan. 2011, Springer

[8] Homem, N. Carvalho, J.P., “Finding top-k elements in data streams”, Information Sciences, 180(24), pp. 4958-4974, Dec. 2010, Elsevier

[9] Ikonomakis, M., S. Kotsiantis, and V. Tampakas. "Text classification using machine learning techniques". *WSEAS Transactions on Computers* 4, no. 8: 966-974. 2005

[10] Keerthi, S., Shevade, S.K., Bhattacharyya, C., Murthy, K., "Improvements to Platt's SMO algorithm for SVM classifier design." *Neural Computation* 13, no. 3 2001: pp. 637-649.

[11] Lee, K., et al, "Twitter trending topic classification." In *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*, pp. 251-258. IEEE, 2011.

[12] Lim, H.S. "Improving kNN based text classification with well estimated parameters." In *Neural Information Processing*, pp. 516-523. Springer Berlin Heidelberg, 2004.

[13] Marujo, L. et al, “Textual Event Detection using Fuzzy Fingerprints”, IEEE Intelligent Systems IS’14, Advances in Intelligent Systems and Computing, Sep. 2014 , Vol 322, pp. 825-836, Springer.

[14] McCallum, A., Nigam, K., "A comparison of event models for naive bayes text classification." In *AAAI-98 workshop on learning for text categorization*, vol. 752, pp. 41-48. 1998.

[15] Pang, B., Lee, L., Vaithyanathan, S., ”Thumbs up?: sentiment classification using machine learning techniques”, EMNLP '02 Proc. of the ACL-02 conference on Empirical methods in natural language processing – Vol.10, pp 79-86, 2002

[16] Platt, J. "Fast training of support vector machines using sequential minimal optimization." *Advances in kernel methods—support vector learning* 3, 1999.

[17] Rosa, H., and Batista, F. and Carvalho, J.P., “Twitter Topic Fuzzy Fingerprints”, WCCI2014, FUZZ-IEEE, 2014 IEEE World Congress on Computational Intelligence, International Conference on Fuzzy Systems, Jul. 2014, pp. 776-783

[18] Salton, G., Buckley, C., "Term-weighting approaches in automatic text retrieval". Information Processing & Management 24 (5): 513–523, 1988.

[19] Socher, R. et al., "Recursive deep models for semantic compositionality over a sentiment treebank." In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, vol. 1631, p. 1642. 2013.

[20] Sood, S. et al., "TagAssist: Automatic Tag Suggestion for Blog Posts." In *ICWSM*. 2007.

[21] Tsoumakas, G., Katakis, I., Vlahavas, I., "Effective and efficient multilabel classification in domains with large number of labels." In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD’08)*, pp. 30-44. 2008.

[22] Wang, S., Manning, C.D., "Baselines and bigrams: Simple, good sentiment and topic classification." *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*. Association for Computational Linguistics, 2012.