Learning Domain Terms - Empirical Methods to Enhance Enterprise Text Analytics Performance

Gargi Roy, Lipika Dey, Mohammad Shakir, Tirthankar Dasgupta
TCS Research and Innovation, India
{roy.gargi,lipika.dey,m.shakir,dasgupta.tirthankar}@tcs.com

Abstract

Performance of standard text analytics algorithms are known to be substantially degraded on consumer generated data, which are often very noisy. These algorithms also do not work well on enterprise data which has a very different nature from News repositories, storybooks or Wikipedia data. Text cleaning is a mandatory step which aims at noise removal and correction to improve performance. However, enterprise data need special cleaning methods since it contains many domain terms which appear to be noise against a standard dictionary, but in reality are not so. In this work we present detailed analysis of characteristics of enterprise data and suggest unsupervised methods for cleaning these repositories after domain terms have been automatically segregated from true noise terms. Noise terms are thereafter corrected in a contextual fashion. The effectiveness of the method is established through careful manual evaluation of error corrections over several standard data sets, including those available for hate speech detection, where there is deliberate distortion to avoid detection. We also share results to show enhancement in classification accuracy after noise correction.

1 Introduction

A large part of enterprise data such as customer complaints, project management documents, client communications, risk reports, emails etc. can yield rich insights and actionable intelligence for improving enterprise processes. Text analytics solutions built with machine learning methods are employed for the purpose. There is also an increasing emphasis on robotic process automation (RPA) where the focus is on automating enterprise tasks. Tasks that do not involve active or deep cognitive abilities can be easily automated. However, if the task involves reading and interpretation of natural language content, there is a need to ensure that the machine interpretation of the content is correct before it is sent for automated downstream processing. Given the volumes of such communication that is generated for any organization in the digital world, automated processing of these content is beneficial to ensure timely response to customers, timely resolution and also generate predictive insights.

It is well known that the quality of analytical results are largely dependent on the quality of input text. Thus, all text analytics solutions are preceded by pre-processing and cleaning steps to help yield better results. Though there exists standard dictionaries to aid the cleaning process, these dictionaries are not enough to deal with enterprise text, due to their inherent nature. Internal enterprise communication like emails, messages etc. contains words, abbreviations and terms that are very domain specific and not available in general purpose thesaurus. This includes terms representing names of services, products or groups etc. They also contain lots of acronyms that are not always standardized but get created along the way and are well understood within a community. Consumer generated content like customer complaints, call logs etc. additionally contain spelling distortions of products and services. The real problem posed by enterprise text is that there is no demarcation between domain words that "appear to be noisy" since they are not part of a dictionary and words that are "true noise" because it is wrongly spelt or misused. For any cleaning to be done for enterprise text, it has to be first resolved whether the unrecognizable term is an enterprise term or a true noise term and then correct it appropriately. Without this, even word or phrase frequencies cannot be generated properly.

Interestingly we found a resonance of the problem while dealing with social media text. There are not only spelling mistakes and typos - but deliberate distortion of traditional words and phrases while writing them down. Some of these are conventional short-hands and some are done to avoid online detection. This is one of the problems faced by online hate speech detectors. Hence detecting and replacing these words with the correct word is an important problem in that scenario also.

Let us illustrate the different types of errors and their significance through the following examples.

- Let us assume a certain repository refers often to the name "Flipkart." This is a non-dictionary term, but may be recognized as a Named Entity if written correctly in a text. However, the more crucial issue is detecting distortions of it in consumer generated text. While analyzing a set of customer communications involving the organization, we found the following mentions - "fipkart", "flipcart", "flipcardt" etc. which are easily recognizable as erroneous mentions of the name by a human, but not by a computer. The performance of many downstream text analytics solutions are affected by this since the frequency computation of the term is affected by these errors.

- The second example will show that not all errors are with respect to named entities only. Analysis of a text repository for a bank yields the following different references to the term "account" - accont, acnt, act etc. While the last two are abbreviated references, the first one is a typo. A standard spell-correction algorithm employing shortest edit-distance based correction corrects "accont" as "accost", which is obviously wrong. Such situations need context-based correction.

- In a third example, we take the case of a specific issue where there is no error, but only "apparent error" introduced by enterprise terms. In a particular repository, a term called "gess" appears very often. This is a company internal term for the organization, referring to a particular service. There are multiple references to this term in the company's internal text repository. A standard spell checker would tend to correct this term to "guess" - which would once again be incorrect.

- Social media abounds in such examples. The terms such as "ebloa", "snapchatt" which are misspellings of non-dictionary yet important, non-noise terms - "ebola" and "snapchat" respectively, where the first one is a virus name that caused epidemic and the second one is a multimedia messaging application. Where as, "presidrnt" is a misspelling of the dictionary term "president". In hate speech data, we found lots of spelling distortions of an Internet slang - "lmao", such as "lmfao", "lmaoooooo", "lmmfao", "lmfaooo", and our method was able to capture them.

The above examples illustrate that text cleaning, especially while dealing with enterprise text, involves segregation of non-dictionary domain terms from true noise words. Since terminologies differ within groups in an enterprise, and also evolve over time, one time dictionary creation is not a solution. As products, processes and tasks evolve over time, the terminologies also change. Hence, there is a need for designing methods that can ingest a given enterprise or social media repository and clean it correctly in an efficient fashion before embarking on a text analytics journey.

In this work, we propose novel unsupervised method that can analyze any enterprise document repository and clean it effectively. Initially, all words in the repository are segregated into two categories - dictionary and non-dictionary words. Any standard linguistic dictionary can be used for the purpose. The key idea is to segregate the domain words from true noise words and then correct the noise words accordingly. The domain words, when recognized, get included in the dictionary. The remaining noise words are corrected incrementally. The whole act is achieved via an iterative method using the following two steps of Analyze and Infer.

Analyze - This step analyzes the word distributions of the dictionary and non-dictionary words and classifies them into different frequency groups.

Infer - This step facilitates the movement of non-dictionary words to the dictionary and finally correct the noise words, based on word structures, their frequency groups and contextual similarities. Similarities between words are established using multiple distance functions that combine traditional edit-distance based measures, phoneme based similarity as well as contextual similarities computed using word embeddings that are created for each repository. The word2vec skip-gram model (Mikolov et al., 2013) is used for creating the word embeddings. The vector dimensions are adjusted to take care of small repositories.

The above steps are applied in an iterative fashion till no movement of word is possible. The validity and effectiveness of the method are established through multiple experiments. We also provide performance analysis of the proposed algorithms to show that the proposed method is highly efficient. Thus it can be used to clean any new repository before applying text analytics algorithms to achieve the best results.

2 Related Work

Most of the noise detection and correction work for text center around spellchecking and spelling correction (Subramaniam et al., 2009). A standard way to do so is to use edit distance based methods (Ristad and Yianilos, 1998) with respect to a standard dictionary. Other methods include noisy channel model for detecting misspellings (Brill and Moore, 2000; Lai et al., 2015), linguistic and statistical approach (Schierle et al., 2008) and graph based approach to correct spelling errors in domain-centric search for emails (Bao et al., 2011). Neural word embeddings for context sensitive spelling correction have also been considered in (Gong et al., 2019; Fivez et al., 2017; Flor, 2012). The work by (Flor et al., 2019) also uses orthographic and phonetic similarity and contextual information for automatic spelling correction. However, none of the above works consider segregation of non-dictionary domain words from noise. Hence, most of these methods are not applicable for enterprise text which deserves special attention. In (Li et al., 2018), methods have been specified for acronym disambiguation in enterprise text. However other errors are not addressed in this. In (Lu et al., 2019), methods are presented for correcting errors contextually, based on neighboring words and edit distance measures. However, these also do not take care of enterprise domain words and consider them as errors. Although, (Shakir et al., 2019) has attempted to identify domain terms over true error terms based on the assumption that domain and correct terms are more uniformly distributed across documents, this assumption does not work always. It is often found that some spelling mistakes also occur uniformly and with high frequency. Thus there is a need to work on unsupervised, robust and repeatable methods for detecting true noise terms in enterprise text and correct them. This work addresses that need.

3 Enterprise text cleaning - a prelude to analytics

Consumer generated text is inherently noisy. Hence all text analytics tasks start with mandatory preprocessing and cleaning steps as follows. These texts are found to be fraught with random use of upper cases, spaces and punctuation marks. Social media has also given rise to a set of new terms that represent phrases, and may be expressed by a non-traditional combination of characters. For example, the term 4u represents “for you”. Certain errors are also introduced due to the underlying digital platforms. Incorrect conversion of characters to symbols or icons fall under this category. Rule based pre-processors are implemented to remove the above noises. After the removal of unwanted elements, the text is tokenized using the modified twitter tokenizer (Dey and Roy, 2015). The modification is done in such a manner that it suits the semi-formal enterprise text tokenization. Afterwards, stop words are removed and words are normalised to lower case letters. The next step is to perform noise correction, which involves recognition of noisy terms and restoring them to their correct forms. The next section explains the proposed steps to do this for enterprise text.

4 Correcting Noisy terms in Enterprise Text

In this section we present the details of the proposed noise correction algorithm. To begin with, all tokens in the text repository are segregated into two groups, dictionary and non-dictionary, depending on whether they belong to a standard language dictionary or not. Each token is tagged with their category. Further, the frequency of each token is computed and ranked.

For a token w, its frequency is given by the number of occurrence of w in the corpus and is denoted by w_f. The rank of w, denoted by w_r, is determined by ordering the words in descending order of frequency. The word with highest frequency gets rank one, second highest will have rank two and so on.

Let the set of tokens that are detected as dictionary words be indicated by $\mathbf{V}$, and the set of non-dictionary tokens be denoted by $\mathbf{N}$. The task now is to reduce the cardinality of $\mathbf{N}$ by identifying tokens that can be moved from it to $\mathbf{V}$. This happens iteratively through the following actions.

Analyze - This step analyzes the relative frequencies and ranks of tokens and classifies them into different frequency groups.

Infer - This step checks the similarities among tokens in terms of their structures, frequency groups and contextual presence in text. The final movement is inferred based on a weighted similarity computed using multiple distance functions that combine traditional edit-distance based measures, phoneme based similarity as well as contextual similarities computed using word embeddings that are created for each repository.

The next two subsections describe the above steps in detail.

4.1 Analyzing word frequencies in Repository

Zipf's law (Zipf, 1932; Manning et al., 1999) states that the log-log plot of the rank versus frequency of words in a corpus usually follows a linear pattern. This distribution is computed from all words in the corpus. Usually, the noise words occur less frequently, hence their individual frequencies are very low. Consequently, it is assumed that their ranks are very high and are concentrated at the tail of the distribution. Though the Zipf's law has been found to represent word distributions from different kinds of repositories like News, story books etc. quite well, we observed that it fails to represent enterprise text correctly. The distributions shown in top row of Figure 1 shows this. The distortion is noticeable towards the side of high-frequency or top ranked words. Delving deeper, we found that the dictionary and non-dictionary words show two different trends. While the relative occurrence of high-frequency dictionary words in the repository is less than expected, the relative occurrence of high-frequency non-dictionary words is much higher than expected. The log-log plot of Zipf's distribution for dictionary and non-dictionary words plotted separately in middle and lower bands of Figure 1 illustrate this. The reason obviously lies in the fact that enterprise content, especially parts of it, are far more repetitive in nature and hence the pattern breaks. Intuition therefore suggests that, analyzing the repository to find the tokens that cause the deviation from linearity should be a good idea to spot domain words.

Given that the repository is already segregated into two sets of tokens $\mathbf{V}$ and $\mathbf{N}$, this is accomplished as follows:

(a). For each set $\mathbf{V}$ and $\mathbf{N}$, the zipf's distribution is plotted using log-log scale of rank w_r versus frequency w_f of the tokens belonging to the set. A linear regression minimizing the standard error, is then applied, to determine the best fitting line to both the distributions independently. The standard error is defined by the absolute value of the difference between predicted and actual values.

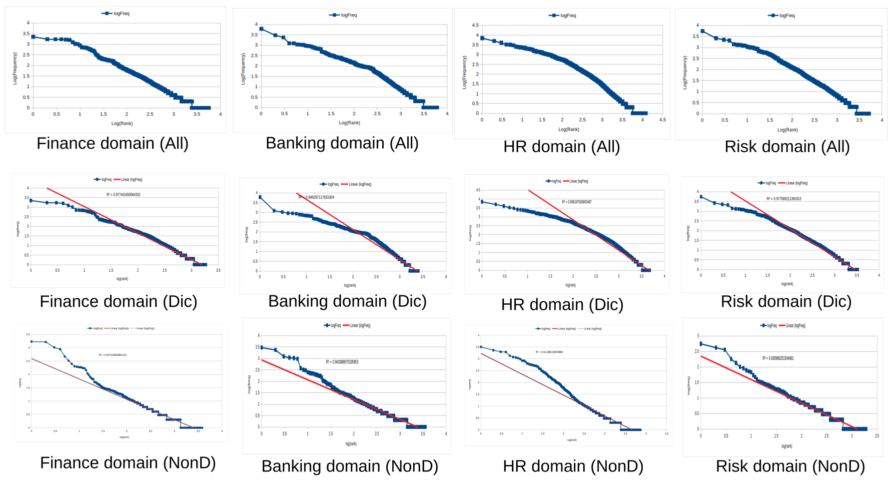


Figure 1: Log-log plot of rank versus frequency of all, dictionary (Dic) and non-dictionary (NonD) terms in four different enterprise content.

The next task was to find natural groups of words in each set, such that all words in a group are likely to belong to the same error category, indicating that these words would have similar kind of distribution and hence similar roles to play in the repository. The natural breaks are determined using Jenks optimization method (Jenks, 1977) on the standard error between the regression line and the original frequency curve. The optimal number of natural breaks are determined by applying Jenks optimization method. Jenks optimization method, which is also called goodness of variance fit (GVF), divides a set of values into groups or classes such that intra-class variance is minimized while maximizing inter-class variance. The value of GVF ranges from 0 to 1 where 0 signifies "No Fit" and 1 signifies "Perfect Fit". The value of GVF, denoted by η, is a parameter fed to the process which helps the process terminate with an optimal number of classes within an acceptable error limit. The method starts with class number equal to 2, and then iteratively increases the number of classes till the error limit crosses η. In our implementations, we have used 0.85 as the value of η. Function 2 depicts this.

Jenks natural breaks, also called classes play a very useful role in determining the role of words within a repository. Each class in each set is assigned a unique identity. In this work, the class associated to the highest-frequency side of the regression line is assigned the identity 1 in both the cases. Thus, for both the sets $\mathbf{V}$ and $\mathbf{N}$, it can be stated that a word belonging to Jenk's class i in that set appears with higher frequency in the repository than a word belonging to class $i+1$ in the same set. Each token w in each set can now be associated to a class identity, based on the class it belongs to. For token w, its class is denoted by w_i^S, where the superscript S takes value $\mathbf{V}$ or $\mathbf{N}$, depending on the parent set of w and i denotes the class number.

The class identity of a token w is utilized by the inference process to decide whether the token should be retained as a domain term or be declared a noise and hence corrected.

4.2 Inferring nature of words - segregating true noise terms from domain terms

As stated in the earlier section, the objective of the inference mechanism is to determine whether a non-dictionary token is a wrongly spelt dictionary word or a domain word, that should be retained as it is. The decision is taken based on the Jenk's class of the noise word and also it's similarity to other words in the repository.

Traditional noise detection and correction methods have assumed all wrongly spelt words in a text to be errors and attempt to correct them using similarity measures with known words. Some commonly used similarity measures used in literature are Jaro-Winkler distance (Jaro, 1989; Winkler, 1990), Fuzzy distance, Levenshtein distance (Levenshtein, 1966), orthographic similarity (Weber, 1970; Van Orden, 1987) etc. Table 1 shows comparative results for these measures. It is seen that Jaro-Winkler distance is the most effective measure.

A second type of similarity that can be used to determine the likelihood of two tokens being same is based on phonetic similarity. Phonetic similarity plays a very important role in enterprise text which contains many non-familiar words. These words are often spelt wrong, but have a spelling which "sounds similar" to the original word. This is the reason why we see many occurrences of "Flipcart" for the term is "Flipkart". There are standard ways of deriving the phonetic representation of any word and thereafter compute pair-wise phonetic similarity of words. For phonetic representation (denoted by $ph(w)$), we have used Double Metaphone algorithm (Philips, 2000) that has a large code length and we used up to 25, which we have found to be sufficient for encoding the words.

However none of the above measures can capture contextual similarity or dissimilarity of words. So we propose to use word vectors to capture context of words. Since enterprise text has non-standard vocabulary, no existing model serves the purpose and word vectors have to be built for every repository. Unlike, other embedding generation models, our focus is on building word vectors from small repositories rather than large ones.

To obtain the word vectors for all words in a repository, we start with a pre-processed tokenized corpus from which stop words are removed. The word2vec (Mikolov et al., 2013) algorithm which uses the skip-gram model is utilized to generate the word vectors. The word vector for token w is denoted by w^v. The skip-gram model is specifically chosen since it has shown to work well for small data and can represent rare terms also. We have used window size of five and vector dimension of 20, as the vocabulary sizes for each repository is not very large. The dimension can be varied.

While word embeddings can encapsulate context, character embeddings for words have been found to help in recognizing new words in a vocabulary that have partial overlaps with known words. They play an important role in recognizing different lemmatizations of the same form, different entities of the same type by recognizing a particular suffix or prefix and so on. To generate the character embedding vectors (denoted by w^c), we have used a standard Chars2vec library (Engineering, 2019) that comes with a pre-trained model with different vector dimensions. We have used 50 as the dimension since the vocabularies are small.

For a word w_i, its vector embedding is denoted by w_i^v and it's character vector is denoted by w_i^c. We have used cosine similarity as a measure to compute similarity between two vectors.

4.3 Recognizing domain terms and correcting Noise terms

We now present an approach that can detect domain terms in a completely unsupervised way and further assist in cleaning the repository text by correcting noise. It may be noted that, domain terms need not be only non-dictionary, terms like "account" or "banks" are domain terms for financial content.

In order to achieve this, the first step is to characterize the words as belonging to one of the following four categories, which are maintained as different sets - *i*) Dictionary domain term - denoted by $\mathbf{D}'$, *ii*) Non–dictionary domain term denoted by $\mathbf{D}''$, *iii*) A noise word which is a distortion of a dictionary term $\mathbf{E}'$ and *iv*) A noise term which is a distortion of a non-dictionary term $\mathbf{E}''$.

Noise	Correction	JW	FD	LD	OS
acccessible	accessible	0.98	26	1	0.97
calculte	calculate	0.98	16	1	0.88
accont	account	0.97	10	1	0.87
acconut	account	0.97	11	2	0.83
accout	account	0.97	13	1	0.87
balnce	balance	0.97	7	1	0.85
adjustmentrs	adjustment	0.97	28	2	0.77
hoildays	holidays	0.97	5	2	0.85
cahnged	changed	0.96	2	2	0.57
ccorrect	correct	0.96	17	1	0.7
deatils	details	0.96	5	2	0.83
calim	claim	0.94	2	2	0.49
ddeduction	deduction	0.94	23	1	0.72
excempeted	exempted	0.91	18	2	0.82
apalgamation	amalgamation	0.9	5	1	0.72
kundly	kindly	0.9	1	1	0.55
aalowance	allowance	0.86	2	1	0.66

Table 1: Different spelling similarities between a pair of words. JW: Jaro-Winkler distance, FD: Fuzzy distance, LD: Levenshtein Distance, OS: Orthographic Similarity

The movement of words from one set to another set occurs throughout the process until no more movement is possible. As edit distance based spelling similarity is computationally expensive, we have designed our algorithm in a way that it reduces time and computational complexity and does not compute similarity measures for all possible pair of words which takes $O(n^2)$, instead efficiently we choose to compute through hierarchically relaxing constraints.

Step 1 - All terms in Jenk's class 1 of $\mathbf{V}$ are declared as dictionary domain terms and are added to $\mathbf{D}'$. Terms like *account, bank, appraisal, mobile, customer* etc. come under this category, depending on which dataset is being analyzed.

Step 2 - Target now is to detect domain words from the set of non-dictionary terms $\mathbf{N}$. For each $w_i \in \mathbf{N}$, it's phonetic representation $ph(w_i)$ is compared with phonetic representation of the dictionary words belonging to the set $\mathbf{V}$. Let $\mathbf{V_m} \subseteq \mathbf{V}$ such that each word $w_j \in \mathbf{V_m}$ satisfy all the following properties (a) w_j and w_i have exactly similar phonetic representation (b) $JW(w_i, w_j) > \alpha_1$ and $Sim(w_i^c, w_j^c) > \alpha_2$, where α_1 and α_2 are user controlled parameters, that are kept high.

Now we pick up the word w_j in $\mathbf{V_m}$ which has highest frequency as the candidate correction term. If the frequency of w_j is higher than frequency of w_i, then w_i is assumed to be a wrongly spelt occurrence of w_j. Hence w_i is added to $\mathbf{E}'$ with w_j marked as a correction for w_i.

However, if w_j has lower frequency than w_i, then there is a chance that w_i may be a domain word and not a wrongly spelt occurrence of w_j. However this has to be verified. The frequencies and Jenk's class numbers of w_i and w_j are considered for the purpose. Contextual similarities of the words are also checked.

Since frequencies and class numbers have different orders of magnitude, so an analysis is done (given in Function 3) on the relative difference (Törnqvist et al., 1985) of class numbers and frequencies as given in Equation 1 and Equation 2 The corresponding functions are given in Function 5 and Function 4 respectively. These equations have been designed using the sigmoid function, so that both the values remain within 0 to 1. Where, w_i^S and w_i^f denote the class number and the frequency of word w_i. $max\{x, y\}$ and $min\{x, y\}$ return the maximum and minimum of $\{x, y\}$ respectively. In Equation 2, logarithm to the base 10 is used to represent the large range of values in a compact way and also to capture the multiplicity of the large number i.e. "how large the number is in multiple", with respect to the small number.

$$d_s = \sigma(ln(\frac{max\{w_i^S, w_j^S\} - min\{w_i^S, w_j^S\}}{min\{w_i^S, w_j^S\}})),\ where\ \sigma(x) = \frac{e^x}{e^x + 1} \tag{1}$$

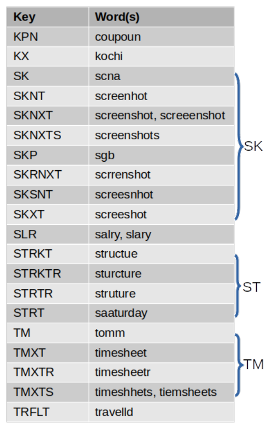
$$d_l = \sigma(log_{10}(max\{w_i^f, w_j^f\}) - log_{10}(min\{w_i^f, w_j^f\})),\ where\ \sigma(x) = \frac{e^x}{e^x + 1} \tag{2}$$

The higher the values of d_l and d_s, lower the possibility of them to be the same word. If d_l is higher than a user defined threshold, say θ_1, then we don't go for any further analysis and declare w_i to be a domain word which needs no correction. Hence, w_i is moved from $\mathbf{N}$ to $\mathbf{D''}$.

An example of this is a term *GESS* which is very close phonetically to *guess* but is actually a domain term. Hence it has a much higher frequency and rank than the dictionary word guess. Hence a correction would not be appropriate in this case. Rather the word should be retained as it is.

If the difference in relative frequencies between w_j and w_i are low, then we further check the value of d_s along with the cosine similarity between their word vector representations, $Sim(w_i^v, w_j^v)$. If $d_s > \theta_2$ and $Sim(w_i^v, w_j^v) \leq \theta_3$, then we infer that w_i is a domain term. The class number of a word captures its relative significance in a repository, independent of the actual frequency values. This value is used to control the identification of unknown words in a cautious fashion by imposing the following restriction - an unknown or error word which has a higher class in $\mathbf{N}$ is not allowed to be identified as a misspelling of a word belonging to a lower class in $\mathbf{V}$ indiscriminately. This is the key feature of the proposed work that enables detection of domain words correctly. The cosine similarity of word vectors further establish the contextual similarity, since the neighbours of a word are taken into consideration while constructing the embeddings. Occurrences of terms like *STURCTURE* which is a very common typos for the intended term STRUCTURE are corrected at this stage since both the terms have same phonetic representation and also a substantial overlap in their context.

Step 3 - Now, we concentrate on the words in $\mathbf{N}$ which do not have an exact but approximately phonetically similar counterpart in $\mathbf{V}$. Some of these words may also be domain words while others may be errors. $\mathbf{N}$ is alphabetically sorted based on the phonetic representation strings by representing it as a Red-Black tree which is self-balancing binary search tree and sorted based on the natural ordering of the keys (here, phonetic representation string of the words are the key). All words that have matching phonemes in first two positions are grouped together through obtaining the sub tree (in-order traversal) whose key ranges are specified by the above condition i.e. matching of first two characters. Function 6 depicts this where ξ is the approximation constraint and set to two in this work (as we are matching first two characters of the phonetic representations).



Key	Word(s)
KPN	coupoun
KX	kochi
SK	scna
SKNT	screenhot
SKNXT	screenshot, screeenshot
SKNXTS	screenshots
SKP	sgb
SKRNXT	scrrenshot
SKSNT	screesnhot
SKXT	screeshot
SLR	salry, slary
STRKT	structue
STRKTR	sturcture
STRTR	struture
STRT	saaturday
TM	tomm
TMXT	timesheet
TMXTR	timesheetr
TMXTS	timeshhets, tiemsheets
TRFLT	travelld

Figure 2: A part of $\mathbf{N}$ as phoneme wise sorted, balanced binary search tree

Thus $\mathbf{N}$ gets divided into several smaller subsets. Figure 2 shows a portion of the above mentioned tree and the smaller subsets obtained are shown in curly brackets along with their matched phonemes in first two positions. For each subset $\mathbf{N_i} \in \mathbf{N}$, we now compute pair-wise Jaro-wrinkler distance between the words as well as between their phonetic representations, the cosine-similarities of their character and word vector representations. These measures are then used to construct tightly connected graphical components over the text representation space, such that each node in this graph may be considered as altered representations of each other. Thus, from the three subsets in Figure 2, the following three updated subsets are obtained. $\{$*screenhot, screenshot, screeenshot, screenshots, scrrenshot, screesnhot, screeshot*$\}$, $\{$*structue, sturcture, struture*$\}$, $\{$*timesheet, timesheetr, timeshhets, tiemsheets*$\}$. If any of these nodes now represent a word which has already been identified as a noisy representation of a dictionary word, i.e. $\in \mathbf{E'}$, then all words in this component are also moved to $\mathbf{E'}$. This is repeated for all subsets containing phonetically similar words. These words are removed from their corresponding subsets in $\mathbf{N}$. For example, from the above mentioned subsets consider the second subset, *sturcture* has been identified earlier and *structure* is suggested as correction. So, now both *struture, structue* are identified as errors with *structure* as suggested correction. Thus, the terms whose phonetic representations are not exactly same as that of any dictionary word but are typos, are corrected appropriately now.

For the remaining non-empty subsets that have cardinality greater than two, the frequency distribution of the words within the subset is generated and tested for anomaly detection using Grubbs' test (Grubbs and others, 1950). If Grubb's test detects the highest frequency word as an anomaly, then the word is considered as a domain term and is moved to $\mathbf{D}''$ while the others are considered as erroneous representations of it and moved to $\mathbf{E}''$. For subsets with two elements we apply Equation 2 which computes relative difference between the frequencies with threshold γ which is set to be 0.75 in this work, and apply the same logic as earlier to decide whether a term is a domain term and move it accordingly to $\mathbf{D}''$. Otherwise we keep it as it is in $\mathbf{N}$ since no correction can be suggested for it. Continuing from the earlier example, thus, *screenshot* and *timesheet* are now recognised as non-dictionary domain term (which are very common in many enterprise repositories) respectively from the first and third sub set and the other words are identified as misspelling of the recognised domain term and suggested correction accordingly.

Step 4 - Finally, the identified erroneous terms in $\mathbf{E}'$ and $\mathbf{E}''$ are replaced by their candidate corrections in the corpus. The entire corpus is again segregated into dictionary and non-dictionary terms (say $\mathbf{V}'$, $\mathbf{N}'$ respectively) and Jenk's classes are once again generated as earlier. Finally, the words belonging to the two highest ranked classes of $\mathbf{V}'$ are considered as domain dictionary terms and maintained as $\mathbf{D}'$ while the corresponding terms from $\mathbf{N}'$ are inferred as non-dictionary domain terms and moved to $\mathbf{D}''$. The execution of the algorithm is presented in Algorithm 1.

Algorithm 1: IdentifyCorrect

```
   Input  : Corpus tokens, $\mathbf{C}$
   Output: $\mathbf{D}', \mathbf{D}'', \mathbf{E}', \mathbf{E}''$
1  $\{\mathbf{N}, \mathbf{V}\} \leftarrow$ segregate words in $\mathbf{C}$
2  for $w \in \mathbf{C}$ do
3      $\{w_f, w_r, w^v, w^c, ph(w)\} \leftarrow compute\ features(w)$
4  $\mathbf{W}^{\mathbf{N}} \leftarrow$ PlotFitBreak($\mathbf{N}$)
5  $\mathbf{W}^{\mathbf{V}} \leftarrow$ PlotFitBreak($\mathbf{V}$)
6  for $w \in \mathbf{N}\ and\ w' \in \mathbf{V}$ do
7      Efficient phonetic match s.t. $ph(w) = ph(w')$
8      if $JW(w, w') > \alpha_1\ and\ Sim(w^c, w'^c) > \alpha_2$ then
9          if $(w_f > w'_f)$ then
10             $w_i^N \leftarrow$ get class number of $w$ from $\mathbf{W}^{\mathbf{N}}$
11             $w_i'^V \leftarrow$ get class number of $w'$ from $\mathbf{W}^{\mathbf{V}}$
12             $w_{type} \leftarrow$ AnalyseRelativeDifference($w, w', w_i^N, w_i'^V,$
13             $w_f, w'_f, Sim(w^v, w'^v)$)
14             if $w_{type}\ is\ domain$ then
15                 $\mathbf{D}'' \leftarrow w$
16             else
17                 $\mathbf{E}' \leftarrow w$ and $w'$ is correction for $w$
18 Represent $\mathbf{N}$ as phonetic representation-wise sorted balanced binary search
     tree
19 for $w \in \mathbf{N}$ do
20     $\mathbf{A} \leftarrow$ GetApproxPhonemeMatch($w, \mathbf{N}, \xi$)
21     for $w' \in \mathbf{A}$ do
22         if $JW(w, w') > \beta_1\ and\ Sim(w^c, w'^c) > \beta_2\ and$
23         $Sim(w^v, w'^v) > \beta_3\ and\ JW(ph(w), ph(w')) > \beta_4$ then
24             $\mathbf{Y} \leftarrow \{w, w'\}$
25 Compute set $\mathbf{X}$ containing all such set $\mathbf{Y}$ which contains spelling wise,
     phonetically and contextually consistent non-dictionary terms
26 for $\mathbf{Z} \in \mathbf{X}$ do
27     if $\exists w \in \mathbf{Z}$ s.t. $w \in \mathbf{E}'$ then
28         Move all other words of $\mathbf{Z}$ to $\mathbf{E}'$ with correction equal to
             correction of $w$ and remove $\mathbf{Z}$ from $\mathbf{X}$
29 for $\mathbf{Z} \in \mathbf{X}$ do
30     $w$ is the word with maximum frequency $w_f$ in $\mathbf{Z}$
31     if $|\mathbf{Z}| \geq 3$ then
32         if $w_f$ is sufficiently large than frequencies of other words in $\mathbf{Z}$
             using Grubbs' test then
33             $\mathbf{D}'' \leftarrow w$ and $\mathbf{E}'' \leftarrow (\mathbf{Z} - \{w\})$
34     if $|\mathbf{z}| = 2$ then
35         if $RelDiffLargeRange(w_f, w'_f)) > \gamma$ where $w, w' \in \mathbf{z}$ then
36             $\mathbf{D}'' \leftarrow w$ and $\mathbf{E}'' \leftarrow w'$
37 for $w \in \mathbf{E}' \cup \mathbf{E}''$ do
38     Replace $w$ with its correction $w'$ and update frequency of $w'$
39 $\mathbf{N}' \leftarrow$ update $\mathbf{N}$
40 $\mathbf{V}' \leftarrow$ update $\mathbf{V}$
41 $\mathbf{W}^{\mathbf{N}'} \leftarrow$ PlotFitBreak($\mathbf{N}'$)
42 $\mathbf{W}^{\mathbf{V}'} \leftarrow$ PlotFitBreak($\mathbf{V}'$)
43 $\mathbf{D}' \leftarrow$ get words from first two classes from $\mathbf{W}^{\mathbf{V}'}$
44 $\mathbf{D}'' \leftarrow \mathbf{D}'' \cup$ get words from first two classes from $\mathbf{W}^{\mathbf{N}'}$
```

4.4 Time complexity analysis:

The algorithm is designed and implemented efficiently. At every step, we have reduced comparisons by applying the different similarity checking functions in a hierarchical fashion as discussed in Steps 1 – 4 earlier. By grouping the dictionary words according to their phonetic representation and keeping them in an hash table with the phonetic representation code as key, the search for phonetically similar words is accomplished in $O(1)$ time. The hash sets are typically small. Application of the conditions based on high Jaro-wrinkler and character vector similarities, reduces the set of possible terms for candidate matches for an error term even further. The cosine similarity computation to check for contextual similarity is therefore conducted between very few pairs of words only. For efficient computation of line 19, the non-dictionary words and their phonetic code are stored in a sorted tree map, where the sub map is retrieved based on the approximation constraint for each non-dictionary word, which takes $O(log(m''))$ time assuming that the sub map contains m'' number of non-dictionary terms where $m'' \ll m$, assuming m to be the cardinality of $\mathbf{N}$, hence the total time is $(m * log(m''))$.

Algorithm 2: PlotFitBreak

```
Input  : Set of words: $\mathbf{W}$
Output: Set containing every word from $\mathbf{W}$ with its class number
1 Plot rank versus frequency in log-log scale within $\mathbf{W}$
2 Apply linear regression on the plot
3 Compute standard error of the regression
4 Compute Jenks natural breaks on the standard error iteratively (starting
    from 2) until the goodness of variance fit reaches to $\eta$
5 Mark the class number of the words in $\mathbf{W}$ according to the natural
    breaks
```

Algorithm 3: AnalyseRelativeDifference

```
Input  : Non-dictionary word: $w$, dictionary word: $w'$, class number of the
         words: $w_i^N$, $w_i'^V$, frequency of the words: $w_f$, $w'_f$, cosine
         similarity between the word vectors of $w$, $w'$: $Sim(w^w, w'^w)$
Output: Type of a word - domain or not
1 if RelDiffLargeRange($w_f$, $w'_f$) $\geq \theta_1$ then
2     $w$ is a domain word
3 else if RelDiffSmallRange($w_i^N$, $w_i'^V$) $\geq \theta_2$ and $Sim(w^w, w'^w) \leq \theta_3$
    then
4     $w$ is a domain word
```

Algorithm 4: RelDiffLargeRange

```
Input  : Frequencies of words: $w_f$, $w'_f$
Output: Relative difference between $w_f$, $w'_f$: d
1 $max \leftarrow$ getMaximum($w_f$, $w'_f$)
2 $min \leftarrow$ getMinimum($w_f$, $w'_f$)
3 d $\leftarrow \sigma(log_{10}(max) - log_{10}(min))$,
    $\sigma(x) = \frac{e^x}{e^x+1}$
```

Algorithm 5: RelDiffSmallRange

```
Input  : Class number of words: $w_i^N$, $w_i'^V$
Output: Relative difference between
        $w_i^N$, $w_i'^V$: d
1 $max \leftarrow$ getMaximum($w_i^N$, $w_i'^V$)
2 $min \leftarrow$ getMinimum($w_i^N$, $w_i'^V$)
3 d $\leftarrow \sigma(ln(\frac{max-min}{min}))$, $\sigma(x) = \frac{e^x}{e^x+1}$
```

Algorithm 6: GetApproxPhonemeMatch

```
Input  : A word: $w$, sorted, balanced binary
         search tree: $\mathbf{W}$, approximation
         constraint: $\xi$
Output: Set of words: $\mathbf{W}'$
1 $\mathbf{W}' \leftarrow$ Return the sub tree from $\mathbf{W}$ whose
    key ranges from first letter of ph($w$) to the
    number of letters specified in $\xi$
2 $\mathbf{W}' \leftarrow \mathbf{W}' \cup w$
```

5 Results

This section presents the results and evaluation of our techniques. We have done manual evaluation for error correction as well as performed experiments to show that noise cleaning indeed enhances the performance of classification. We have done all of these for 7 sets of enterprise data. We also applied our method on 3 hate speech data sets to detect distortions of slang words. For these sets also we report enhancement in classification performance after correction of errors.

Data set description: Dataset 1 to dataset 7 are internal enterprise data sets pertaining to different domain such as, risk, contigency, finance, banking and HR. These data sets contain internal organizational mails, communications, customer complaints, customer requests. The other five data sets are as follows. *i*) Dataset 8: This data set (Weissenbacher et al., 2018) contains tweets mentioning different drug names, their side effects. *ii*) Dataset 9: This data set contains customer complaints with multi-label annotation in telecom domain (Dasgupta et al., 2016). The complaints has been generated within India. Dataset 10: This Offensive Language Identification Dataset (OLID) (Zampieri et al., 2019) contains annotated offensive social media content. *iii*) Dataset 11: This data set (Davidson et al., 2017) contains tweets with racist, sexist, homophobic and offensive content. Although this dataset contains annotations, we did not consider the annotations. *iv*) Dataset 12: This dataset (de Gibert et al., 2018) contains posts extracted from Stormfront, a white supremacist forum. This data set is annotated, however, we have used those texts which are annotated as hate speech. One of the evaluation criteria was downstream task of classification to show that the noise cleaning actually enhances the classification performance. Of the above, data sets 5, 6, 7, 9 and 10 are labeled. For these we present results for classification before and after cleaning. Classification has been done using Gradient Boosting classifier. Dataset 9 is multi-labeled, hence has been classified using Multilabel k Nearest Neighbours classifier. Classification results are presented using accuracy, precision, recall and F1-Scores.

Performance evaluation: All these experiments were run on a standard machine with Intel® Core^{TM} i7-4600U CPU @ 2.10GHz 4 with 15.6 GiB memory having 64-bit ubuntu 18.04 LTS operating system. Error correction results were manually evaluated for subsets of words for all the 12 data sets and the performance is presented in Table 2. The evaluation was done against annotated set with given ground truths. The table shows that the precision of correction is very high at around 90%. Analysis shows that most of the unresolved error words are names of people and places, land marks and house/building names etc. sometimes as a part of main text or signature in case of emails. These usually have very low

Data Set	Domain	# of documents	Corpus Tokens	Unique Non-Dictionary Terms	Domain Dictionary Terms		Error Terms							
							Resolved	Domain Terms		Misspelling of Dictionary Terms		Misspelling of NonDictionary Terms	
					Detected	Correct(%)		Detected	Correct(%)	Detected	Correct(%)	Detected	Correct(%)
1	Risk	9942	7211	3022	27	27(100%)	1457	565	546(97%)	760	636(84%)	132	124(94%)
2	Contigency	9942	6640	2525	142	139(98%)	1067	505	495(98%)	481	438(91%)	81	74(91%)
3	Risk	9387	5251	1934	100	100(100%)	746	317	303(96%)	387	367(95%)	42	32(76%)
4	Contigency	9387	5448	1965	165	165(100%)	626	240	233(97%)	353	325(92%)	33	29(88%)
5	Finance	2147	5865	3967	29	29(100%)	1092	447	430(96%)	427	316(74%)	218	158(73%)
6	Banking	2360	6843	4122	41	41(100%)	766	137	131(96%)	431	356(83%)	198	135(68%)
7	HR	16357	12201	7309	67	67(100%)	3272	381	369(97%)	2261	1884(84%)	630	339(62%)
8	Twitter	5383	10766	5192	1157	1130(98%)	370	150	143(95%)	220	170(77%)	-	-
9	Telecom	5394	17263	10009	130	129(99%)	4102	255	236(93%)	3105	2403(77%)	772	567(73%)
10	Offensive	13241	17784	5973	225	225(100%)	619	287	266(93%)	243	217(89%)	89	75(84%)
11	Hate tweet	24783	32552	21228	505	498(99%)	5701	2208	1933(88%)	2718	236(91%)	775	719(93%)
12	Hate post	958	3382	588	598	597(100%)	179	144	135(94%)	35	29(83%)	-	-

Table 2: Manual evaluation of domain and error term detection and correction for the error terms

Performance	Multi Class								Multi-label	
	Dataset5 (Finance)		Dataset6 (Banking)		Dataset7 (HR)		Dataset10 (Offensive)		Dataset9 (Telecom)	
	raw	clean	raw	clean	raw	clean	raw	clean	raw	clean
Precision	0.924	0.96	0.425	0.5	0.878	0.89	0.79	0.8	0.54	0.58
Recall	0.922	0.96	0.418	0.47	0.849	0.87	0.63	0.66	0.32	0.38
F1-Score	0.923	0.96	0.421	0.48	0.863	0.88	0.7	0.72	0.409	0.46
Accuracy	0.927	0.96	0.444	0.48	0.855	0.89	0.75	0.77	0.44	0.46

Table 3: Classification performance of the data sets with and without cleaning

frequencies and high variance. Table 3 present the classification performance (before and after cleaning) showing that performance is enhanced (around 3-13%) after cleaning.

Table 4 presents a slang term in row 1 and it's distortions encountered very often in hate speech data (dataset 11). It is purposefully distorted to avoid automatic detection by word-based filters. Columns 2 - 5 in this table show that no single distance measure would be able to capture these, since they vary a lot from the original term. Our multi-pronged approach are able to detect these types of distortions also very well. Hence this approach can be more effective in detecting such terms from social media.

Non-dictionary term	Phonetic code	JW(t,t')	Sim (t^v, t'^v)	Sim(t^c, t'^c)
motherfuckin	M0RFKN	**1**	**1**	**1**
motherfuckas	M0RFKS	**0.93**	0.75	0.88
mufuckin	MFKN	0.84	**0.97**	0.74
muthafucka	M0FK	0.78	**0.96**	0.83
mothafucka	M0FK	0.89	**0.94**	0.9
muthafuckin	M0FKN	0.87	**0.97**	0.89
mothafuckin	M0FKN	0.95	0.92	**0.97**
muthafukin	M0FKN	0.84	**0.94**	0.9
muhfuckin	MFKN	**0.87**	0.86	0.85
muthaufckin	M0FKN	0.84	0.76	**0.89**
muhhfuckin	MFKN	0.84	**0.96**	0.88

Table 4: Non-dictionary slang (in bold and denoted as t') along with its different distorted versions (denoted as t) are detected from a hate speech data set (Dataset11)

6 Conclusion

This work presents an efficient method for identifying domain terms over true noise terms from enterprise text and thereafter clean the text appropriately to enhance performance of text analytics programs. The methods, though empirical, have been derived after careful analysis of many large enterprise repositories. Experimental evaluation shows that the method proposed is highly accurate in determining domain terms and also correct errors. The effectiveness of the approach is also established through improvement in classification accuracy after correction. Though designed for enterprise text, it was found that the method can be very fruitfully engaged in detecting errors in hate speech data originating in social media. Since these words are deliberately distorted, the errors in this case do not follow any specific pattern and are more difficult to determine. We have shown that our combined approach performs very well in such scenarios also. We are now working on formalizing the error detection and correction in enterprise text as an optimization problem. We are also working on an interactive system such that the method can receive rewards for a correct action and penalized for a wrong one. This can be effectively used for designing better systems which can utilize reinforcement learning for cleaning enterprise text. Detecting and removing names and other low frequency errors is also on the agenda.

References

Zhuowei Bao, Benny Kimelfeld, and Yunyao Li. 2011. A graph approach to spelling correction in domain-centric search. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 905–914.

Eric Brill and Robert C Moore. 2000. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th annual meeting of the association for computational linguistics*, pages 286–293.

Tirthankar Dasgupta, Lipika Dey, and Ishan Verma. 2016. Fuzzy multi-label classification of customer complaint logs under noisy environment. In *International Joint Conference on Rough Sets*, pages 376–385. Springer.

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. In *Proceedings of the 11th International AAAI Conference on Web and Social Media*, ICWSM '17, pages 512–515.

Ona de Gibert, Naiara Perez, Aitor García-Pablos, and Montse Cuadros. 2018. Hate Speech Dataset from a White Supremacy Forum. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 11–20, Brussels, Belgium, October. Association for Computational Linguistics.

Lipika Dey and Gargi Roy. 2015. Auto-correction of consumer generated text in semi-formal environment. In *7th Language and Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics. Fundacja Uniwersytetu im. Adama Mickiewicza w Poznaniu*, pages 203–207.

Intuition Engineering. 2019. Chars2vec: Character-based language model for handling real world texts with spelling errors and human slang.

Pieter Fivez, Simon Suster, and Walter Daelemans. 2017. Unsupervised context-sensitive spelling correction of clinical free-text with word and character n-gram embeddings. In *BioNLP 2017*, pages 143–148.

Michael Flor, Michael Fried, and Alla Rozovskaya. 2019. A benchmark corpus of English misspellings and a minimally-supervised model for spelling correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 76–86, Florence, Italy, August. Association for Computational Linguistics.

Michael Flor. 2012. Four types of context for automatic spelling correction. *TAL*, 53(3):61–99.

Hongyu Gong, Yuchen Li, Suma Bhat, and Pramod Viswanath. 2019. Context-sensitive malicious spelling error correction. In *The World Wide Web Conference*, pages 2771–2777.

Frank E Grubbs et al. 1950. Sample criteria for testing outlying observations. *The Annals of Mathematical Statistics*, 21(1):27–58.

Matthew A Jaro. 1989. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420.

G.F. Jenks. 1977. *Optimal data classification for choropleth maps: George F. Jenks*. Occasional paper. University of Kansas. Department of Geography.

Kenneth H Lai, Maxim Topaz, Foster R Goss, and Li Zhou. 2015. Automated misspelling detection and correction in clinical free-text records. *Journal of biomedical informatics*, 55:188–195.

Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

Yang Li, Bo Zhao, Ariel Fuxman, and Fangbo Tao. 2018. Guess me if you can: Acronym disambiguation for enterprises. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1308–1317.

Chris J Lu, Alan R Aronson, Sonya E Shooshan, and Dina Demner-Fushman. 2019. Spell checker for consumer language (cspell). *Journal of the American Medical Informatics Association*, 26(3):211–218.

Christopher D Manning, Christopher D Manning, and Hinrich Schütze. 1999. *Foundations of statistical natural language processing*. MIT press.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Lawrence Philips. 2000. The double metaphone search algorithm. *C/C++ users journal*, 18(6):38–43.

Eric Sven Ristad and Peter N Yianilos. 1998. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532.

Martin Schierle, Sascha Schulz, and Markus Ackermann, 2008. *From Spelling Correction to Text Cleaning – Using Context Information*, pages 397–404. 01.

Mohammad Shakir, Gargi Roy, Aninda Sukla, Tirthankar Dasgupta, Geetika Sharma, and Lipika Dey. 2019. Framework for analyzing and improving quality of available data for enterprise automation tasks. In *DCCL workshop at ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

L Venkata Subramaniam, Shourya Roy, Tanveer A Faruquie, and Sumit Negi. 2009. A survey of types of text noise and techniques to handle noisy text. In *Proceedings of The Third Workshop on Analytics for Noisy Unstructured Text Data*, pages 115–122.

Leo Törnqvist, Pentti Vartia, and Yrjö O Vartia. 1985. How should relative changes be measured? *The American Statistician*, 39(1):43–46.

Guy C Van Orden. 1987. A rows is a rose: Spelling, sound, and reading. *Memory & cognition*, 15(3):181–198.

Rose-Marie Weber. 1970. A linguistic analysis of first-grade reading errors. *Reading Research Quarterly*, pages 427–451.

Davy Weissenbacher, Abeed Sarker, Michael Paul, and Graciela Gonzalez. 2018. Overview of the third social media mining for health (smm4h) shared tasks at emnlp 2018. In *Proceedings of the 2018 EMNLP Workshop SMM4H: The 3rd Social Media Mining for Health Applications Workshop & Shared Task*, pages 13–16.

William E Winkler. 1990. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019. Predicting the Type and Target of Offensive Posts in Social Media. In *Proceedings of NAACL*.

George Kingsley Zipf. 1932. Selected studies of the principle of relative frequency in language.