Massive Text Normalization via an Efficient Randomized Algorithm

Nan Jiang
jiang631@purdue.edu
Purdue University
IN, USA

Chen Luo
cheluo@amazon.com
Amazon Search
CA, USA

Vihan Lakshman
vihan@amazon.com
Amazon Search
CA, USA

Yesh Dattatreya
ydatta@amazon.com
Amazon Search
CA, USA

Yexiang Xue
yexiang@purdue.edu
Purdue University
IN, USA

ABSTRACT

Current popular machine learning techniques in natural language processing and data mining rely heavily on high-quality text sources. Nevertheless, real-world text datasets contain a significant amount of spelling errors and improperly punctuated variants where the performance of these models would quickly deteriorate. Moreover, existing text normalization methods are prohibitively expensive to execute over web-scale datasets, can hardly process noisy texts from social networks, or require annotations to learn the corrections in a supervised manner. In this paper, we present Flan (Fast LSH Algorithm for Text Normalization), a scalable randomized algorithm to clean and canonicalize massive text data. Our approach suggests corrections based on the morphology of the words, where lexically similar words are considered the same with high probability. We efficiently handle the pairwise word-to-word comparisons via locality sensitive hashing (LSH). We also propose a novel stabilization process to address the issue of hash collisions between dissimilar words, which is a consequence of the randomized nature of LSH and is exacerbated by the massive scale of real-world datasets. Compared with existing approaches, our method is more efficient, both asymptotically and in empirical evaluations, does not rely on feature engineering, and does not require any annotation. Our experimental results on real-world datasets demonstrate the efficiency and efficacy of Flan. Based on recent advances in densified Minhash, our approach requires much less computational time compared to baseline text normalization techniques on large-scale Twitter and Reddit datasets. In a human evaluation of the quality of the normalization, Flan achieves 5% and 14% improvement against the baselines over the Reddit and Twitter datasets, respectively. Our method also improves performance on Twitter sentiment classification applications and the perturbed GLUE benchmark datasets, where we introduce random errors into the text.

CCS CONCEPTS

• **Information systems → Information retrieval**.

KEYWORDS

Lexical Normalization, Locality-Sensitive Hashing, Natural Language Processing

ACM Reference Format:
Nan Jiang, Chen Luo, Vihan Lakshman, Yesh Dattatreya, and Yexiang Xue. 2022. Massive Text Normalization via an Efficient Randomized Algorithm. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3485447.3512015

1 INTRODUCTION

Many natural language processing (NLP) algorithms rely on high-quality text sources to obtain state-of-the-art results [23, 46]. Recent studies have witnessed that model performance deteriorates when these deep learning models are evaluated on real-world noisy texts [19, 29]. Specifically, text data extracted from web sources such as Twitter, Reddit, and search query logs contain numerous instances of spelling errors, typos, and non-standard punctuation marks [50, 55]. These noises can render pretrained neural models (like BERT [20]) trained on clean data sources ineffective.

This challenge motivates the need for *lexical normalization*, which is the task of cleaning noisy input words into canonicalized forms. Prior techniques for lexical normalization involve 1) combining a rich set of features, such as phonetic similarity, lexical edit distances, *n*-gram probabilities, and word-embedding features [25, 28], 2) supervised learning, where annotated datasets are required to learn a correction mapping from unnormalized words to normalized ones [13], and 3) similarity search with word-embeddings, where the top-ranked words under a vector similarity measure are considered as the correction [16].

In this paper, we present a scalable randomized Flan (Fast LSH Algorithm for Text Normalization). Compared with existing methods, Flan can 1) *eliminate the need for additional annotation* for supervised learning. 2) *scale better on large datasets*, especially those with hundreds of millions or billions of lines of text. 3) *be robust to noise*, by reducing the likelihood of normalizing a word into a dissimilar one due to our proposed graph stabilization technique.

Flan harnesses locality-sensitive hashing (LSH) [57] to find normalized words in a graph and consists of *indexing* and *inference*

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France.
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512015

stages. The input to the indexing stage is the set of tokens found in the data (word unigrams in our experiments). The output is a vocabulary with word corrections, represented as a graph and built via LSH. During inference, we use LSH again to hash an unknown word to its appropriate graph component and substitute this noisy word with the canonicalized representative from the graph.

We boost the probability of LSH bucketing similar words together by taking independent repetitions of the hashing process and building a weighted word-to-word graph where the weights are the number of repetitions in which two tokens share the same hash code. We remove those insignificant edges with weight below a predefined threshold as a stabilization step. In the pruned graph, the words in every connected component are regarded as sharing the same meaning. This edge pruning step reduces the likelihood of a word being normalized to a dissimilar one.

Our experiments compare Flan with several popular text normalization methods. In terms of empirical running time comparison, we find that Flan is much faster than baselines across the indexing and inference stages, on large-scale Twitter and Reddit datasets. In a human evaluation on the correction quality across the Twitter and Reddit datasets, Flan achieves a 5% and 15% higher F1-Score, respectively, against the baselines. We also demonstrate the impact of Flan on downstream NLP tasks. On the Twitter sentiment analysis and perturbed GLUE benchmark, Flan demonstrates consistent improvement over the baselines. We further provide a case study of applying Flan in an industrial setting to normalize search queries and product titles on a dataset sampled from the search logs of a large e-commerce website. These experiments all show a consistent advantage of the proposed Flan approach against all the competing baselines.

Our contributions in this paper can be summarized as follows:

(1) We present an efficient algorithm for lexical normalization that uses the morphological similarity between words for lexical correction. To the best of our knowledge, this similarity measurement has not been fully explored in this domain and offers compelling advantages over existing word embedding search and lexical/phonetic edit distance models. In particular, our technique does not require supervised training or annotated data. Flan also scales better to large datasets thanks to the efficiency of densified LSH over competing algorithmic primitives.
(2) The randomized nature of LSH introduces the possibility of dissimilar words sharing the same signature due to undesirable hash collisions. This problem becomes very prevalent at massive scales. We address this challenge of dealing with unfavorable collisions through a novel approach of modeling the LSH outputs as a word-to-word graph with multiple repetitions and edge pruning. We show that Flan is robust to noise and scales well to large datasets theoretically and empirically.
(3) We compare Flan with several existing popular methods over different datasets, comparing the average running time, examining the quality of the word corrections via human evaluations, and providing several case studies for the performance over perturbed GLUE benchmark datasets, Twitter sentiment analysis, and a large-scale product search dataset.

2 BACKGROUND & RELATED WORK

2.1 Lexical Normalization

Lexical normalization has received a surge of interest with the advent of mobile computing [8, 14], where typing on small keyboards increases the opportunity for typos, and the rise of social media [6], where users are accustomed to using slang, abbreviations, and other types of informal languages. Lexical normalization refers to the process of transferring non-standard, informal, or misspelled tokens into their standardized counterparts as well as converting words of various tenses or pluralization into a consistent representation [38]. This process has emerged as a crucial step to utilize neural models in NLP, which are often pretrained on clean text corpora, on noisy real-world datasets.

Prior techniques in lexical normalization all involve either: 1) combining features, like phonetic similarity and lexical distances with standard word and *n*-gram probabilities [25, 26, 28], 2) supervised learning, where annotated datasets are required to learn a correction mapping [13], or 3) nearest neighbor search with pretrained word-embeddings, where the top-ranked words under cosine similarity measure are considered as the correction candidates [16].

In the literature, the classic approaches for lexical normalization usually encompass a combination of spelling correction, stemming [34], and regular expression filtering [10]. More recent works have introduced unsupervised statistical models for text cleaning [5, 17] or combining multiple heuristics to identify and normalize out of vocabulary words [24]. Another explored learning robust word representations through end-to-end neural networks as opposed to normalizing the data beforehand [22, 35] or directly fine-tuning the BERT models for lexical normalization tasks [38]. Another group of works focuses on directly learning over the subword level information, where character sequences or subword pairs are directly used for learning the representation without any correction steps [38].

However, several unavoidable issues limit the approaches as mentioned above. The pattern of typos may vary across data sources and languages, possibly requiring training separate deep models or collecting additional annotations. Existing normalization methods are either prohibitively slow when applied over massive datasets or require expensive and time-consuming model training.

2.2 Locality-Sensitive Hashing Family

Locality-sensitive hashing (LSH) is widely used to find approximate nearest neighbors in high dimensional space [3]. A common hash process encodes an input x into a *hash code* $h(x) \in \mathbb{N}$ by a *hash function h*. The *hash table* is composed of buckets (or bins) that use the hash code as the index, where each input item x is placed into the $h(x)$th bucket. A *hash collision* occurs when the hash code for two different inputs are equal: $h(x_i) = h(x_j), x_i \neq x_j$. The *collision probability* is proportional to some monotonic function of the similarity between the two: $P[h(x_i) = h(x_j)] \propto f(\mathtt{sim}(x_i, x_j))$, where $\mathtt{sim}(x_i, x_j)$ is the similarity under consideration and f is a monotonically increasing function.

More precisely, let $\mathcal{H}$ be a family of hash functions mapping an input vector to a discrete integer set, namely $h : \mathbb{R}^d \to \mathbb{N}$ where $h \in \mathcal{H}$. A locality-sensitive hash family is a family of functions with the following property: under the hash mapping, similar inputs have a higher probability of having the same hash code than those dissimilar ones [57].

Definition 2.1 (Locality-Sensitive Hashing Family [30]). Given inputs $\{x_i \in \mathbb{R}^d\}_{i=1}^N$ and a similarity metric $\mathtt{sim}(x_i, x_j)$, a hashing family $\mathcal{H}$ is called $(r, \alpha r, p, q)$-sensitive if a function h chosen uniformly from $\mathcal{H}$ satisfies the following properties: 1) If $\mathtt{sim}(x_i, x_j) \geq r$, then $P_{h \sim \mathcal{H}}[h(x_i) = h(x_j)] \geq p$; 2) If $\mathtt{sim}(x_i, x_j) \leq \alpha r$, then $P_{h \sim \mathcal{H}}[h(x_i) = h(x_j)] \leq q$. In practice, we assume $p > q$ and $\alpha \in (0, 1)$.

Minwise Hashing. For set resemblance, also known as Jaccard similarity, minwise hashing (MinHash) is a popular form in the LSH family [9]. The MinHash applies a random permutation π on the given set $\mathcal{S}$, and stores only the minimum value after the permutation mapping. The MinHash value of set $\mathcal{S}$ is computed as: $h_\pi^{\min}(\mathcal{S}) = \min \pi(\mathcal{S})$. Given two sets $\mathcal{S}_i$ and $\mathcal{S}_j$, the probability of the two sets having a hash collision is their Jaccard similarity:

$$P\left[h_\pi^{\min}(\mathcal{S}_i) = h_\pi^{\min}(\mathcal{S}_j)\right] = \frac{|\mathcal{S}_i \cap \mathcal{S}_j|}{|\mathcal{S}_i \cup \mathcal{S}_j|} \tag{1}$$

where the probability comes from the randomly sampled permutation π. One major concern with MinHash is the time efficiency, as it requires applying a large number of permutations on the data. The current best approach based on the fast-Johnson-Lindenstrauss transformation [1], calculates T hash codes of the input vector with dimensions d in time $O(d \log d + T)$.

Densified One Permutation Hashing. Recent advance in densification based hashing shows that several minwise hashes can be efficiently computed in $O(d+T)$ time [48, 49]. Subsequently, Shrivastava [47] improved upon this result to present a densification-based method that is variance-optimal, providing the most efficient algorithm, both statistically and in terms of runtime, available for computing approximate near neighbor problems on sets.

This research leverage the most recent advances in the drastic reductions in hashing time. We explore the morphological similarity between words [4]. This measure of similarity makes densification-based MinHash an ideal subroutine for our proposed algorithm.

3 METHODOLOGY

3.1 Motivations

To measure the distance or similarity between two words, extensive research has been conducted over two metrics: edit distance and cosine similarity. Edit distance and its variations, including Levenshtein, Damerau–Levenshtein, and Jaro-Winkler distance [15] are all defined around computing the minimal sequence of edit operations (i.e., deletion, insertion, and replacement) for changing a given word into another. Information on neighboring characters on keyboards as well as phonetic relationships is needed to adjust the cost of deletion, insertion, and replacement. In the cosine similarity paradigm [21], words are embedded into the high-dimensional unit ball in the Euclidean space, and the distance between two words is the angle between their corresponding word vectors. In the domain of lexical normalization, these two metrics require prohibitive computational costs when dealing with large data [11, 25]. In this work, we consider Jaccard similarity as the similarity measurement between word pairs. Here, the Jaccard similarity is the ratio of character spans (or subwords) that two words share. The advantage of this metric is that it can handle web-scale data via recent algorithmic advances in computing LSH signatures [49, 57]. Our method focuses solely on the structures and patterns within words and does not incorporate the semantic or syntactic meaning of words from the context.

3.2 Lexical Normalization by Densified MinHash

Densified MinHash for Words. This step takes a word as input and generates hash code via densified MinHash pipeline, an example of which is shown in Figure 1(a). Given a word of n characters $w_i = c_1 c_2 \ldots c_n$, it is partitioned into a set of subwords $\mathcal{S}(w_i)$:

$$\mathcal{S}(w_i) = \{c_i\}_{i=1}^n \cup \ldots \{c_i \ldots c_{i+k}\}_{i=1}^{n-k} \ldots \cup \{c_1 c_2 \ldots c_n\} \tag{2}$$

Here, $\mathcal{S}(w_i)$ is the of union all cardinality of subwords. In our experiments, this is controlled by hyper-parameter `CHARS`, denoting which substrings are included in the set. For example, `CHARS` = $[1, 3]$ signifies that the character-level unigram and trigram sets are included in set $S(w_i)$. If the substring length is longer than the input word's length, its subword set is defined as $\emptyset$.

After obtaining the subwords set, we use a hash function h from a 2-universal hash family $\mathcal{H}_{univ}$ [51] to map every string into a discrete integer value in a large universe $\{1, \ldots, U\}$:

$$\mathcal{U}(w_i) = \{h(s) \in \{1, \ldots, U\} | s \in \mathcal{S}(w_i), h \sim \mathcal{H}_{univ}\} \tag{3}$$

As discussed in Section 2.2, for efficiency concern, we use densified MinHash instead of permutation-based MinHash [31] to generate the hash code for the set $\mathcal{U}(w_i)$. We divide the universe L into non-overlapping buckets and the elements in $\mathcal{U}(w_i)$ are correspondingly partitioned. Then we preserve the minimum value in every bucket as the hash code for this discrete range. The preserved value at j^{th} bucket is:

$$\mathcal{L}_j(w_i) = \min\left\{u \middle| u \in \mathcal{U}(w_i), u \in \left[\frac{j}{L}U, \frac{j+1}{L}U\right]\right\} \tag{4}$$

One issue unresolved is that we cannot have a signature for those empty bins. Shrivastava [47] proposes an optimal strategy to reassign every empty bin with a value from one of the existing non-empty bins. In particular, fix one empty bin, we flip a coin and borrow the first non-empty bin value from either the left or the right. After that, we obtain a signature array $\mathcal{L}(w_i)$ to represent the input word w_i.

To further represent every word as one single hash code, we adopt another strategy that randomly hashes array $\mathcal{L}(w_i)$ into an integer in another universe $\{1, \ldots, U'\}$ [32]. Here, we pick another hash function h' at random from the universal hash family $\mathcal{H}_{univ}$ that recursively hashes the array of signature values into one element:

$$\mathcal{A}_j(w_i) = \begin{cases} h'(\mathcal{L}_j(w_i)) & \text{If } j = 1 \\ h'(\mathcal{L}_j(w_i) + \mathcal{A}_{j-1}(w_i)) & \text{If } 1 < j \leq L \end{cases} \tag{5}$$

where $h' \sim \mathcal{H}_{iniv}$. The last step output $\mathcal{A}_L(w_i)$ is the final hash code for input word w_i. We show, for example, the detailed process of mapping an input word into a hash code in Example 3.1.

Example 3.1. In Figure 1(a), given a input word "*goo0d*", we first generate the subword set $\mathcal{S}(goo0d) = \{goo, oo0, o0d\}$. It is later hashed into discrete values as $\mathcal{U}(goo0d) = \{2, 23, 28\}$. the universe

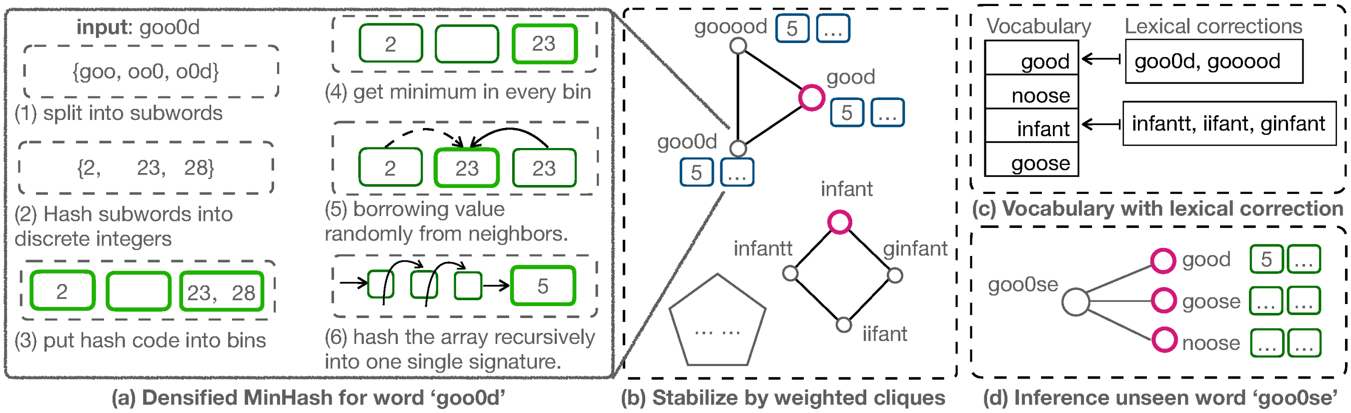
input: goo0d
{goo, oo0, o0d}
(1) split into subwords
{2, 23, 28}
(2) Hash subwords into discrete integers
2
23, 28
(3) put hash code into bins
2
23
(4) get minimum in every bin
2
23
23
(5) borrowing value randomly from neighbors.
5
(6) hash the array recursively into one single signature.
(a) Densified MinHash for word 'goo0d'
gooood
5
...
good
5
...
goo0d
5
...
infant
infantt
ginfant
iifant
... ...
(b) Stabilize by weighted cliques
Vocabulary
Lexical corrections
good
goo0d, gooood
noose
infant
infantt, iifant, ginfant
goose
(c) Vocabulary with lexical correction
goo0se
good
5
...
goose
...
...
noose
...
...
(d) Inference unseen word 'goo0se'

Figure 1: The architecture of the proposed algorithm. (a) Steps of hashing a word into signatures via densified MinHash algorithm. (b) Representing all words in the same buckets as cliques in the undirected graph, where every component of words is regarded as sharing the same meaning. (c) The output is a linked-list style vocabulary, with all the similar words pointing to their representatives. (d) In inference, the new word 'goo0se' uses densified MinHash to validate if it can be canonized.

$U = \{1, \dots, 30\}$ is partitioned into 3 parts: $[1, 10)$, $[10, 20)$, $[20, 30]$. After we preserve the minimum value in every bucket and randomly fill the second bin, which is initially empty, we obtain the sequence $\mathcal{L} = [2, 23, 23]$. The final hash code for the word "goo0d" is computed as $\mathcal{A}_3(goo0d) = 5$ by Equation (5).

To conclude, given two words w_i and w_j, the probability of the event that two words w_i, w_j has hash collision $(h(w_i) = h(w_j))$ by densified MinHash is proportional to their subwords' Jaccard similarity, and further proportional to the morphological similarity of the two words [57]:

$$P[h(w_i) = h(w_j)] \propto \frac{|\mathcal{S}(w_i) \cap \mathcal{S}(w_j)|}{|\mathcal{S}(w_i) \cup \mathcal{S}(w_j)|} \propto \mathtt{sim}(w_i, w_j) \tag{6}$$

Here we assume that words located in the same bucket are lexically similar. These grouped words usually are the variant of one canonical representation, which we call the *representative* word. Our idea for lexical normalization is to use this representative to replace all of the rest words in the same bucket.

However, due to the other property of LSH families introduced in Definition 2.1, dissimilar words can also have the same signature with an unavoidable small probability. For real-world datasets with millions of distinct words, the chance of at least one pair of dissimilar words sharing a signature becomes significant, which leads to a high rate of improper corrections. Motivated by the Count-Min Sketch data structure [12, 18, 33], we propose a graphical stabilization method that can greatly decrease the likelihood of a word mapping to any dissimilar words while at the same time maintaining a high likelihood among similar words.

Stabilization by Weighted Cliques. We stabilize by repeating the above hashing process for T ($T \geq 1$) times and aggregating the results with a graph data structure [33]. In a graph $\mathcal{G} = (V, E)$, let every vertex w_i represent a word in the dataset such that the number of vertices in the graph is equal to the number of unique words in the dataset. Let the edge weight $e(w_i, w_j)$ be the number of times two words w_i, w_j are located in the same bucket, where $e(w_i, w_j) \geq 0$. If two words have no hash collision, they do not have an edge. In one repetition, words sharing the same signature form a connected component. For T independent repetitions, we have a weighted graph where the weight of each edge represents the number of repetitions in which two words share the same signature. Figure 1(b) provides an illustration of this stabilization process.

To handle the unavoidable small probability of undesirable hash collisions, we propose removing edges with significantly small weights. To attain that goal, we introduce a criterion with a threshold parameter $\alpha \in [0, 1]$ to determine if an edge is significant. Insignificant edges are pruned to decrease the likelihood of a word mapping to any dissimilar words, which is formalized as:

$$E' = E \setminus \{e(w_i, w_j) | e(w_i, w_j) \leq \alpha T, e(w_i, w_j) \in E\} \tag{7}$$

To be specific, if $e(w_i, w_j) \geq \alpha T$, then the two words w_i, w_j are assumed to be sufficiently similar; if there is no edge between words w_i and w_j or $e(w_i, w_j) < \alpha T$, then these two words are considered as distinct entities and we remove all edges between them in the graph. Note that the $\alpha = 0$ case indicates that no edges will be pruned and is simply the union of the edges over all the repetitions. For $\alpha = 1$ case, only the words with the same signature across all the repetitions are preserved, which is the intersection for all the repetitions. After T repetitions and edge pruning, we interpret the words left in every connected component as sharing the same meaning.

Finally, the output of the algorithm is a linked list-style vocabulary $\mathcal{V}_{\mathcal{G}}$, where the lists of misspelled words are pointed to their representatives. Here we let the most frequent token in every connected component be the representative for this group of similar words. We denote the whole process that takes in all the words and outputs a vocabulary along with lexical corrections as the *Indexing* procedure. Figure 1(c) gives one example of the output vocabulary.

Inference Criterion. Once we create a vocabulary $\mathcal{V}_{\mathcal{G}}$, we can go through every dataset and remap words to their morphological representatives. However, in the inference stage, there may exist words in the testing set that are not covered if they were not present in the indexing corpus. Thus, we apply LSH again to cover as many new unseen words as possible and determine which word, if any, in the vocabulary would be the best fit.

Formally, given vocabulary $\mathcal{V}_{\mathcal{G}}$ from the indexing procedure and an unseen word w_r in the testing set, we apply the densified MinHash for the word w_r and check if the word w_r would have a collision with any word in the vocabulary. After T repetitions, we would have $l \geq 0$ edges that link from a set of words $\{w_i\}_{i=1}^{l}$ in the vocabulary to this word w_r. Then we reuse our prior criterion:

$$e(w_r, w_i) > \alpha T,\ \forall i \in \{1, \dots, l\}, e(w_r, w_i) \in \mathcal{G}$$

for every edge that link to w_r. If there are no edges (i.e., $l = 0$) or none of them satisfy the criterion, this word cannot be canonized. If we find several satisfying words, we pick the word with the highest frequency. Figure 1(d) present an example of this procedure.

3.3 Analysis of Similarity Estimation

Running Time Analysis. Let N be the number of words in the dataset, T be the number of repetitions, and L the average number of subwords in the data set. The time of applying densified MinHash for all words is $O(LNT)$. Afterward, the complexity of constructing the graph is $O(M^2B)$, where M is the expected number of items in each bucket of the hash table, and B is the number of buckets. The final graph pruning takes $O(N)$ time to finish. Thus, the overall computational complexity is $O(LNT + M^2B + N)$. In practice, N is in the order of billions and thus dominates asymptotically, so the simplified time complexity of our method is $O(LNT)$. Note that the major speedup of this method comes from the prior breakthrough in efficiently computing the MinHash signatures [9].

For comparison, a spell correction algorithm based on edit distance runs in time $O(LNX)$, where X is the number of possible characters to be deleted, replaced, or inserted. Given a word, a spelling corrector will consider all the neighboring words with, for example, one and two steps of edit distances, then pick the neighboring word with maximum score in the dictionary. Usually, the number of possible characters X is much larger than the number of repetitions T. Flan would further improve upon the speed of edit distance-based algorithms in distributed settings where we can compute these repetitions in parallel.

Robust Analysis. We analyse the robustness of Flan via the planted clique model in undirected graphs [7]. Given N distinct words $\{w_i\}_{i=1}^{N}$ and several clusters $\{c_1, c_2, \dots\}$, each word belongs to one and only one cluster. Let $|c_j|$ be cluster c_j's size. Let $C(\cdot)$ be a mapping from a word w_i to its appropriate cluster c_j: $c_j = C(w_i)$. Each cluster c_j is a connected component of lexically similar words. Following the same notation as Definition 2.1, let p denote the minimum probability of an intra-cluster edge and q be the maximum probability of an inter-cluster edge. Let $\mathcal{G}$ denote the graph produced by our algorithm. The probability of the graph $\mathcal{G}$ having the edge $e(w_i, w_j)$ is:

$$P\left[e(w_i, w_j) \in \mathcal{G}\right] \begin{cases} \geq p & \text{if } C(w_i) = C(w_j) \\ \leq q & \text{otherwise} \end{cases} \tag{8}$$

In practice, we have $p \gg q$, as we expect the lexical similarity between words in the same component to be larger than those across the components. LSH calculates p and q by estimating modeling the Jaccard similarity between words. The stabilization step is T *i.i.d* coin flips with probabilities p or q. The threshold α for removing insignificant edges are chosen empirically so that $p > \alpha > q$.

We first upper bound the probability of an unrelated word being included in the wrong connected component, noted as *False Positive probability*. Then, we bound the probability that a word will not be assigned to its proper cluster by edge pruning, referred to as the *False Negative probability*.

Proposition 3.2. *1) False Positive Probability. Fix a node w_i, the probability that Flan connect w_i to a node in cluster c_j where $c_j \neq C(w_i)$ is at most $|c_j| \exp\left(\frac{-T(q-\alpha)^2}{3q}\right)$ where $|c_j|$ is cluster c_j's size. 2) False Negative Probability. Fix a node w_i, the probability that Flan does not add an edge from w_i to any of the other nodes in $c_j = C(w_i)$ is at most $\exp\left(\frac{-|c_j|T(p-\alpha)^2}{2p}\right)$.*

Proposition 3.2 implies that the probability of the wrong correction due to the second property of LSH or edge pruning procedure decreases exponentially with repetitions. With an appropriate number of repetitions, our method attains robust estimation for the morphological similarity between all pairs of words. Appendix A provides a detailed proof of Proposition 3.2.

3.4 Connection to Existing Approaches

The distance measurement used in our method is an extension and relaxation of classic stemming operations [41], where two words with the same stem would be of identical meaning. Our method would not only identify two words sharing the same prefix or suffix strings with high similarity, but also any subsequences of the word based on the composition of subword set $\mathcal{S}(w)$.

The Flan graph $\mathcal{G}$ also captures common tendencies in human errors, such as substituting adjacent characters on a keyboard or similar-sounding characters. It reduces the effort of generating features for finding patterns in typos. For a connected component of the graph, words with adjacent or similar sounding characters are included with high probability. These misspelled words are then mapped to the representative word in the final pruned graph.

Popular spell correction methods like Hunspell and Aspell [2] find words that have a "sounds like" word within two steps of edit distance to the original word. For Flan, the words in a given connected component of the graph include those with small edit-distance with high probability. Furthermore, this component is also likely to include words with longer edit distances, offering a dynamic and generalized way for correction. Experimental evidence for this property is collected in Table 5.

Moreover, supervised approaches that build upon rich feature sets about human typing and spelling patterns work well on small-scale and domain-specific datasets. However, different languages and various data domains usually require adjustments, additional annotations, and further feature engineering. Such expert knowledge becomes quite expensive to acquire when we scale to massive data and various languages. Our method, with no such features over typing, spelling, devices, or languages, uses multiple repetitions and pruned edge weights as statistical estimators. Flan can effectively and efficiently normalize words to a canonical form without any supervised learning, annotations, or feature engineering.

4 EXPERIMENTAL STUDY AND ANALYSIS

4.1 Experiment Setups

Datasets. We consider datasets from Twitter, Reddit, the GLUE benchmark [56] with perturbed text, and data sampled from the logs of a large e-commerce search engine. The Twitter sentiment140 dataset contains 1.6 million tweets with 0.7 million distinct words [43]. On the Twitter dataset, we use 80% for the training 10% validation, and 10% for testing. The Reddit dataset has 10 million sentences with 2.7 million unique words [55]. For the GLUE benchmark, we consider MRPC, STSB, RTE, CoLA and SST2 subtasks, covering single sentence prediction, sentence similarity and paraphrasing along with the language inference tasks. Note that the Reddit dataset is unlabeled, so we only use this corpus to measure the time efficiency and correction quality of various normalization techniques but not for downstream applications evaluation.

Baselines. We consider those methods with different similarity measurements for comparison: 1) edit-distance with standard word dictionary, including a current popular algorithm [2]as well as a classic method [40]. 2) cosine similarity over pretrained word-embeddings. We use Glove [42] and Fasttext [36] as the word-embeddings and apply maximum inner-product search via the FAISS library for searching over the high-dimensional space [27]. Note that there are several lexical normalization methods that are not included in this research, because either the source codes are not shared [52], the methods require annotated lexical normalization datasets [54], a long pipeline with several human-defined rules are needed [11], the methods are built upon morphological and phonetic features that are defined by domain experts [24] or the dependencies of code were out of maintenance [53].

Evaluation Metrics. We evaluate Flan as well as the aforementioned lexical normalization baselines mainly in terms of 1) computational efficiency, evaluating the empirical running time of every algorithm, 2) correction quality, measuring the goodness of correction with human evaluators, and 3) impact on downstream applications, namely Twitter sentiment classification and perturbed GLUE benchmark.

We consider the rate of words to be corrected as the criterion of choosing hyper-parameter α. Given fixed vocabulary size $|\mathcal{V}|$, define $\phi(\mathcal{W},\mathcal{V})$ as the ratio of unique words that get corrected by lexical normalization:

$$\phi(\mathcal{W},\mathcal{V}) = \left(\frac{1}{|\mathcal{W}|}\sum_{w\in\mathcal{W}} \mathbb{I}\{w\in\mathcal{V}\} - 1\right)\times 100\% \qquad (9)$$

the indicator function $\mathbb{I}$ evaluates to 1 if word w is in the vocabulary $\mathcal{V}$ and $\mathcal{W}$ represents the set of unique words from the training and validation sets. The vocabulary is constructed by selecting top $|\mathcal{V}|$ number of frequent words from the training and validation sets.

4.2 Effectiveness of Lexical Correction

Empirical Running Time Benchmark. We compare the running time of all the methods over large-scale datasets. For the "Indexing" procedure, we first extract all the words from the text corpora along with the frequencies of the words. Then, the words are fed into every algorithm, where the output is either the original word or the corrected one. This measures the overall time to create the correction mapping for the whole training set. Only the time used for lexical normalization is calculated for these benchmarks. Specifically, for the "Single" case, the whole algorithm is applied over one process. For the "Multi" case, we partition the workload equally over 20 processors. For the "Inference" step, we benchmark the overall time for mapping words to their normalized form following the indexing stage. As shown in Table 1, we observe that Flan has a faster running time and scales better to the dataset size than the baseline methods across both the indexing and inference stages.

Table 1: Running time of lexical normalization methods over Twitter and Reddit Datasets. Flan scales better to the dataset size and is faster over Indexing and Inference scenarios than the competing approaches.

Datasets	Methods	Indexing (mins)		Inference (mins)
		Single	Multi	
Twitter	Flan (α = 0.2)	**40●**	**3●**	**18●**
	Al-Hussaini [2]	171	16	49
	Norvig [40]	510	41	154
	FAISS-Glove	408	25	83
	FAISS-Fasttext	44	6	29
Reddit	Flan (α = 0.2)	**59●**	**12●**	**26●**
	Al-Hussaini [2]	520	46	71
	Norvig [40]	731	93	221
	FAISS-Glove	514	29	101
	FAISS-Fasttext	70	19	42

For the implementation of the proposed approach and the hyper-parameter settings, please see Appendix B.1. We acknowledge that the computation time is impacted by choice of programming language, specific libraries, and software engineering optimizations. For comparison fairness, we use the same procedure of feeding the inputs into and extracting the outputs from the algorithms. All the methods we benchmark are implemented in Python.

Quality Comparison. To evaluate the quality of the corrections made by a given lexical normalization method, we conducted a study with native English speakers to evaluate the quality of the correction methods. We first select 100 sentences each from the Twitter and Reddit datasets, feed the sentences into every algorithm, and then extract the corrected output sentences. We create a questionnaire for the corrected sentences and deploy it to the Amazon Mechanical Turk. Five different native speakers evaluated the quality of each sentence after correction. Each reviewer was asked to label every corrected sentence as either "Good", "Neutral", "Bad", or "Not Sure". We define the label "Good" as signifying the corrections make the meaning of the text more clear or more grammatically correct. The label "Bad" denotes that the corrections make the meaning of the text less clear or less grammatically correct. "Neutral" case signifies that the corrections do not improve or diminish the clarity of the text. See Appendix B.2 for details.

Since we do not have ground-truth labels for this dataset, we cannot easily classify a correction as "correct" or "incorrect," which makes measuring recall a challenge. Thus, we focus on evaluating if the correction helps the human evaluator better understand the meaning of the sentence. To assess the results from this study, we consider "good" and "neutral" as a correct result and regard "bad" as an incorrect one, since all these methods are "unsupervised". The

Table 2: Human evaluation for the quality of word corrections. For Twitter dataset, Flan has a higher Recall and F1-score. For the Reddit dataset, and Flan has a higher Precision and F1-Score value than the baselines.

Datasets	Methods	Precision	Recall	F1-Score
Twitter	Flan ($\alpha = 0.2$)	60.45%	**41.76%**●	**49.39%**●
	Al-Hussaini [2]	37.93%	35.71%	36.79%
	Norvig [40]	51.79%	28.57%	36.83%
	FAISS-Glove	**71.43%**●	9.34%	16.52%
	FAISS-Fasttext	65.28%	24.18%	35.28%
Reddit	Flan (α=0.2)	**84.85%**●	**34.33%**●	**48.88%**●
	Al-Hussaini [2]	42.53%	**34.33%**●	37.99%
	Norvig [40]	66.00%	32.84%	43.85%
	FAISS-Glove	63.64%	17.16%	27.04%
	FAISS-Fasttext	75.71%	22.39%	34.56%

precision is calculated as the ratio between the number of correct results to the total number of corrections. The Recall is defined as the fraction of problematic sentences that are corrected to good. The F1-Score is calculated based on Precision and Recall [38, 54].

The results are presented in Table 2. On the Twitter dataset, Flan has the highest recall and F1 score while the FAISS-Glove method has the highest precision score, when compared to all the approaches. For the Reddit dataset, Flan has the highest precision and F1 score when compared to the baselines.

Furthermore, Table 5 gives several successful examples on the correction made by Flan, which shows that Flan can corrects words with small and multiple character difference. However, we still observe some failure cases with Flan, such as mapping "evga" and "vga" together. Disambiguation for such pairs would likely require more information on the surrounding context of a word. We defer this investigation for future work.

4.3 Impact on Downstream Applications

Twitter Sentiment Analysis. We evaluate the impact of lexical normalization over real-world noisy tweets. The task is to classify the sentiment of a given tweet as positive or negative. For the neural learning model, we use the summation of word vectors as the sentence representation, which is then mapped to a two-dimensional output vector via an affine transformation. The learning objective is to minimize the logistic loss between the predicted label and the ground truth label. The word vectors inside the model are randomly initialized and we set the dimension to 256. Prior to training the model, we apply the various lexical normalization techniques we study in our experiments. We refer to "No corr." as the model with no correction. We report the accuracy on the validation and testing sets, which we also normalize, when we reach the best result on the corresponding validation set.

As shown in Table 3, we observe that Al-Hussaini [2] and Norvig [40] do not improve the classification result, because of the large percentage of mismatch between the language style on Twitter and formal writing. For the FAISS-Glove and FAISS-Fasttext models, they can convert unseen words into semantically-similar words. Flan does not introduce such a domain mismatch and attains the best accuracy performance among the competing approaches.

Table 3: Accuracy comparison on the Twitter dataset. The Flan improve the Accuracy on the validation set by 0.1% and testing set by 0.2% against all the baselines.

Methods	Valid Accuracy	Test Accuracy
No Corr.	79.40%	79.39%
Flan ($\alpha = 0.2$)	**79.54%**●	**79.62%**●
Al-Hussaini [2]	79.08%	79.16%
Norvig [40]	79.06%	79.18%
FAISS + Glove	79.42%	79.41%
FAISS + Fasttext	79.44%	79.41%

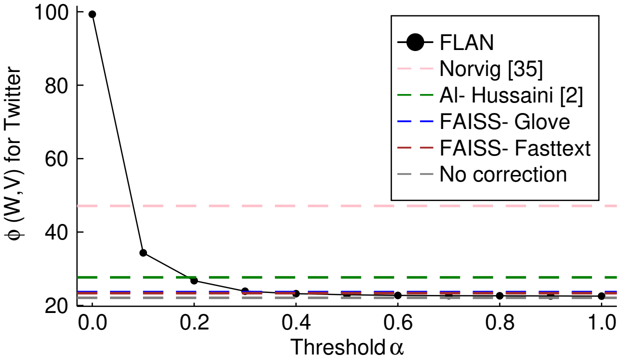

Figure 2: Words covered ($\phi(\mathcal{W}, \mathcal{V})$) on the Twitter dataset. Flan's coverage is determined by the threshold α.

Perturbed GLUE Benchmark. To further investigate the impact of lexical normalization tools over the related NLP tasks, we consider MRPC, STSB, RTE, CoLA and SST2 subtasks from the popular GLUE benchmark [56]. As the GLUE datasets are of high quality, we follow previous approaches [22, 45] in randomly perturbing the words in the validation and testing dataset while keeping the training set clean. We synthetically generate lexical errors at 20, 40, and 60% rates of noise such that we perturb a sentence with probability equal to this rate and then select 1-2 characters uniformly at random in every word of the sentence to delete or replace with another random character. Note that different from the real-world application on noisy tweets, the synthesized typos are different from the real errors that follow a more structured distribution. We use DistilBert [44] as the pretrained model, which is then fine-tuned over the training set with 10 epochs. We then evaluate the perturbed test sets after applying a normalization algorithm as a cleaning step. We refer to "No corr." baseline as the DistilBert taking those perturbed sentences into the model.

From the result in Table 4, we observe that, for all the subtasks, with the rate of noises becoming higher, the relative improvement of Flan over the "No corr." and also the rest competing approaches become larger on all the chosen subtasks. The main reasons are that these perturbed words are not natural and the other correction methods can hardly capture them. It shows that Flan has better capability to recover the words under the random perturbation and improve the quality of the sentences for downstream learning tasks.

Table 4: Preturbed GLUE benchmark with all the lexical normalization algorithms. We observe that when the noisy level become higher, the FLAN can help to recover more words and get better results than all the competing methods.

Subtasks	Perturbed Rate	Metrics	No corr.	FLAN (α=.2)	Norvig [40]	Al-Hussaini [2]	FAISS-Glove	FAISS-Fasttext
MRPC	20%	Accuracy	78.67%	**78.92%**●	**78.92%**●	74.26%	78.18%	78.18%
		F1-score	84.26%	84.83%	84.07%	82.98%	**84.89%**●	84.83%
MRPC	40%	Accuracy	76.22%	**77.94%**●	77.69%	74.51%	77.43%	77.69%
		F1-score	84.24%	**85.09%**●	84.17%	83.38%	84.71%	84.49%
MRPC	60%	Accuracy	67.89%	**69.11%**●	67.11%	65.44%	67.64%	67.89%
		F1-score	74.10%	**78.64%**●	74.80%	72.62%	73.60%	73.85%
STSB	20%	Pearson	72.02%	**72.55%**●	71.93%	61.70%	69.49%	69.54%
		Spearman	71.61%	**72.39%**●	71.83%	61.74%	69.43%	69.13%
STSB	40%	Pearson	70.80%	**72.76%**●	71.30%	62.57%	71.11%	70.70%
		Spearman	69.70%	**71.67%**●	70.48%	62.09%	70.39%	69.64%
STSB	60%	Pearson	65.73%	**70.25%**●	68.93%	60.57%	66.75%	67.30%
		Spearman	65.02%	**69.90%**●	68.39%	60.65%	66.65%	66.81%
RTE	20%	Accuracy	59.57%	62.09%	58.85%	58.07%	**61.46%**●	59.21%
RTE	40%	Accuracy	57.76%	61.07%	57.04%	56.32%	59.57%	**61.31%**●
RTE	60%	Accuracy	56.32%	**60.29%**●	58.85%	54.15%	57.40%	57.40%
CoLA	20%	Matthew's Corr.	46.00%	**46.50%**●	42.82%	12.25%	39.98%	41.03%
CoLA	40%	Matthew's Corr.	29.71%	**30.92%**●	30.21%	10.41%	29.98%	28.97%
CoLA	60%	Matthew's Corr.	9.00%	**16.35%**●	13.44%	15.66%	12.86%	15.18%
SST2	20%	Accuracy	77.18%	78.72%	78.93%	76.76%	79.16%	**79.23%**●
SST2	40%	Accuracy	69.73%	**71.23%**●	70.02%	68.02%	70.74%	70.14%
SST2	60%	Accuracy	57.31%	**59.42%**●	58.22%	57.32%	57.67%	58.12%

4.4 Case Studies

Ablation Study on Hyper-parameter α. Hyper-parameter α has the most significant impact on the generated FLAN graph. Thus we study the effect of the graph pruning threshold α on the behavior of FLAN, evaluating the rate of correction by Equation 9. Results on Twitter dataset are shown in Figure 2 and Reddit dataset are shown in Figure 3. When $\alpha = 0$, we see that FLAN corrects nearly every word in the corpus. However, when we change from $\alpha = 0.1$ to $\alpha = 0.2$, we notice that this correction coverage drops rapidly, which empirically demonstrates the exponential decay from applying more repetitions that we discussed previously in Section 3.3. We also plot the correction coverage of our baseline methods for references. Based on these results, we selected $\alpha = 0.2$ as the pruning threshold for all the experiments in this paper since it provided a balance between covering words and not introducing too much noise.

Large-Scale Product Search. We further conducted offline experiments applying FLAN to normalize search queries and product titles sampled from the logs of a large e-commerce search engine [39]. The e-commerce product search logs contain 100 million lines of product descriptions and search queries with 3.2 million unique words. Due to confidentiality concerns, we are limited in the amount of detail we can provide. Datasets at this extreme scale present a significant challenge to existing normalization techniques. We observed that the edit-distance-based methods [2, 40], were prohibitively slow to apply at this scale, requiring days to complete. Meanwhile, FLAN finished normalizing the entire dataset in roughly 4 hours. Moreover, the FAISS-based methods require the additional overhead of learning these word representations on the e-commerce query-product logs.

5 CONCLUSION

In this work, we investigate lexical normalization for cleaning the real-world massive text corpora. We propose FLAN, a scalable randomized algorithm for cleaning massive text data. By leveraging recent advance of densified MinHash, the approximated similarities of all word pairs are efficiently computed. Compared with existing approaches, FLAN does not need extra annotation, rule definition and feature generation. Moreover, we propose using a graphical structure to detect and remove undesirable word associations due to random hash collisions to stabilize the correction quality. We further provide theoretical guarantees on the robustness of FLAN with upper bounds on the false positive and false negative probabilities.

In experiments, we benchmark with several prevalent methods and several large-scale datasets. In running time analysis, FLAN demonstrates a faster computation speed over competing methods from edit-distance models and nearest neighbor models with pretrained word-embedding. When measuring the quality of corrections, FLAN has achieved a relatively higher recall and F1 score against the baselines as measured by human evaluation. Finally, we evaluate the end-to-end benefit of FLAN on two machine learning tasks: Twitter sentiment analysis and perturbed GLUE benchmarks, where we find FLAN consistently improves the quality of noisy texts. To conclude, FLAN mitigates the gap between deep models trained on clean sources and noisy real-world NLP applications.

ACKNOWLEDGMENTS

This work was done while N.J. was a intern at Amazon Search. In addition, N. J. and Y. X. disclose support from NSF grants IIS-1850243, CCF-1918327. We thank reviewers for their comments.

REFERENCES

[1] Nir Ailon and Bernard Chazelle. 2009. The Fast Johnson–Lindenstrauss Transform and Approximate Nearest Neighbors. *SIAM J. Comput.* 39, 1 (2009), 302–322.

[2] Leena Al-Hussaini. 2017. Experience: Insights into the Benchmarking Data of Hunspell and Aspell Spell Checkers. *ACM J. Data Inf. Qual.* 8, 3-4 (2017), 13:1–13:10.

[3] Alexandr Andoni and Piotr Indyk. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM* 51, 1 (2008), 117–122.

[4] Mark Aronoff. 1994. *Morphology by itself: Stems and inflectional classes.* Number 22. MIT press.

[5] AiTi Aw, Min Zhang, Juan Xiao, and Jian Su. 2006. A Phrase-Based Statistical Model for SMS Text Normalization. In *ACL*. The Association for Computer Linguistics.

[6] Timothy Baldwin, Marie-Catherine de Marneffe, Bo Han, Young-Bum Kim, Alan Ritter, and Wei Xu. 2015. Shared Tasks of the 2015 Workshop on Noisy User-generated Text. In *Proceedings of the Workshop on Noisy User-generated Text*. 126–135.

[7] Béla Bollobás and Paul Erdös. 1976. Cliques in random graphs. In *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 80. Cambridge University Press, 419–427.

[8] Francesco Bonchi, Ophir Frieder, Franco Maria Nardini, Fabrizio Silvestri, and Hossein Vahabi. 2012. Interactive and context-aware tag spell check and correction. In *CIKM*. ACM, 1869–1873.

[9] Andrei Z. Broder and Michael Mitzenmacher. 2001. Completeness and robustness properties of min-wise independent permutations. *Random Struct. Algorithms* 18, 1 (2001), 18–30.

[10] Martine Cadot and Joseph di Martino. 2003. A Data Cleaning Solution by Perl Scripts for the KDD Cup 2003 Task 2. *SIGKDD Explor. Newsl.* 5, 2 (2003), 158–159.

[11] Jhon Adrián Cerón-Guzmán and Elizabeth León-Guzmán. 2016. Lexical Normalization of Spanish Tweets. In *WWW (Companion Volume)*. ACM, 605–610.

[12] Beidi Chen, Anshumali Shrivastava, and Rebecca C. Steorts. 2018. Unique entity estimation with application to the Syrian conflict. *Ann. Appl. Stat.* 12, 2 (06 2018), 1039–1067.

[13] Monojit Choudhury, Rahul Saraf, Vijit Jain, Animesh Mukherjee, Sudeshna Sarkar, and Anupam Basu. 2007. Investigation and modeling of the structure of texting language. *Int. J. Document Anal. Recognit.* 10, 3-4 (2007), 157–174.

[14] Nicole Coddington. 2014. Correction of typographical errors on touch displays. US Patent 8,739,055.

[15] William W Cohen, Pradeep Ravikumar, Stephen E Fienberg, et al. 2003. A Comparison of String Distance Metrics for Name-Matching Tasks.. In *IIWeb*, Vol. 3. 73–78.

[16] Richard Cole and Ramesh Hariharan. 2002. Approximate String Matching: A Simpler Faster Algorithm. *SIAM J. Comput.* 31, 6 (2002), 1761–1782.

[17] Danish Contractor, Tanveer A. Faruquie, and L. Venkata Subramaniam. 2010. Unsupervised cleansing of noisy text. In *COLING (Posters)*. Chinese Information Processing Society of China, 189–196.

[18] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms* 55, 1 (2005), 58–75.

[19] Matthias Damaschk, Tillmann Dönicke, and Florian Lux. 2019. Multiclass Text Classification on Unbalanced, Sparse and Noisy Data. In *Proceedings of the First NLPL Workshop on Deep Learning for Natural Language Processing*. 58–65.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. Association for Computational Linguistics, 4171–4186.

[21] Qin Ding, Hsiang-Fu Yu, and Cho-Jui Hsieh. 2019. A Fast Sampling Algorithm for Maximum Inner Product Search. In *AISTATS (Proceedings of Machine Learning Research, Vol. 89)*. PMLR, 3004–3012.

[22] Yerai Doval, Jesús Vilares, and Carlos Gómez-Rodríguez. 2020. Towards Robust Word Embeddings for Noisy Texts. *Applied Sciences* 10, 19 (2020).

[23] Venkat Gudivada, Amy Apon, and Junhua Ding. 2017. Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations. *International Journal on Advances in Software* 10, 1 (2017), 1–20.

[24] Bo Han and Timothy Baldwin. 2011. Lexical Normalisation of Short Text Messages: Makn Sens a #twitter. In *ACL*. The Association for Computer Linguistics, 368–378.

[25] Bo Han, Paul Cook, and Timothy Baldwin. 2013. Lexical normalization for social media text. *ACM Trans. Intell. Syst. Technol.* 4, 1 (2013), 5:1–5:27.

[26] Aminul Islam and Diana Inkpen. 2009. Real-word spelling correction using Google web 1Tn-gram data set. In *CIKM*. ACM, 1689–1692.

[27] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Trans. Big Data* 7, 3 (2021), 535–547.

[28] Nobuhiro Kaji and Masaru Kitsuregawa. 2014. Accurate Word Segmentation and POS Tagging for Japanese Microblogs: Corpus Annotation and Joint Modeling with Lexical Normalization. In *EMNLP*. ACL, 99–109.

[29] R. Andrew Kreek and Emilia Apostolova. 2018. Training and Prediction Data Discrepancies: Challenges of Text Classification with Noisy, Historical Data. In *NUT@EMNLP*. Association for Computational Linguistics, 104–109.

[30] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2014. *Mining of Massive Datasets* (3nd ed.). Cambridge University Press, USA.

[31] Ping Li, Art B. Owen, and Cun-Hui Zhang. 2012. One Permutation Hashing. In *NIPS*. 3122–3130.

[32] Chen Luo and Anshumali Shrivastava. 2018. Arrays of (locality-sensitive) Count Estimators (ACE): Anomaly Detection on the Edge. In *WWW*. ACM, 1439–1448.

[33] Chen Luo and Anshumali Shrivastava. 2018. Arrays of (locality-sensitive) Count Estimators (ACE): Anomaly Detection on the Edge. In *WWW*. ACM, 1439–1448.

[34] Dimitrios P. Lyras, Kyriakos N. Sgarbas, and Nikolaos D. Fakotakis. 2007. Using the Levenshtein Edit Distance for Automatic Lemmatization: A Case Study for Modern Greek and English. In *ICTAI*. IEEE Computer Society, 428–435.

[35] Valentin Malykh, Varvara Logacheva, and Taras Khakhulin. 2018. Robust Word Vectors: Context-Informed Embeddings for Noisy Texts. In *NUT@EMNLP*. Association for Computational Linguistics, 54–63.

[36] Tomás Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in Pre-Training Distributed Word Representations. In *LREC*. European Language Resources Association (ELRA).

[37] Michael Mitzenmacher and Eli Upfal. 2017. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis* (2nd ed.). Cambridge University Press.

[38] Benjamin Müller, Benoît Sagot, and Djamé Seddah. 2019. Enhancing BERT for Lexical Normalization. In *W-NUT@EMNLP*. Association for Computational Linguistics, 297–306.

[39] Priyanka Nigam, Yiwei Song, Vijai Mohan, Vihan Lakshman, Weitian Allen Ding, Ankit Shingavi, Choon Hui Teo, Hao Gu, and Bing Yin. 2019. Semantic Product Search. In *KDD*. ACM, 2876–2885.

[40] Peter Norvig. 2009. Natural language corpus data. *Beautiful data* (2009), 219–242.

[41] Chris D. Paice. 1994. An Evaluation Method for Stemming Algorithms. In *SIGIR*. ACM/Springer, 42–50.

[42] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *EMNLP*. ACL, 1532–1543.

[43] Tapan Sahni, Chinmay Chandak, Naveen Reddy Chedeti, and Manish Singh. 2017. Efficient Twitter sentiment classification using subjective distant supervision. In *COMSNETS*. IEEE, 548–553.

[44] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *EMC2@NeurIPS*. IEEE, 1–5.

[45] Kira A. Selby, Yinong Wang, Ruizhe Wang, Peyman Passban, Ahmad Rashid, Mehdi Rezagholizadeh, and Pascal Poupart. 2021. Robust Embeddings Via Distributions. *CoRR* abs/2104.08420 (2021). arXiv:2104.08420

[46] Victor S. Sheng, Foster J. Provost, and Panagiotis G. Ipeirotis. 2008. Get another label? improving data quality and data mining using multiple, noisy labelers. In *KDD*. ACM, 614–622.

[47] Anshumali Shrivastava. 2017. Optimal Densification for Fast and Accurate Minwise Hashing. In *ICML (Proceedings of Machine Learning Research, Vol. 70)*. PMLR, 3154–3163.

[48] Anshumali Shrivastava and Ping Li. 2014. Densifying One Permutation Hashing via Rotation for Fast Near Neighbor Search. In *ICML (JMLR Workshop and Conference Proceedings, Vol. 32)*. JMLR.org, 557–565.

[49] Anshumali Shrivastava and Ping Li. 2014. Improved Densification of One Permutation Hashing. In *UAI*. AUAI Press, 732–741.

[50] Sujoy Kumar Sikdar, Byungkyu Kang, John O'Donovan, Tobias Hollerer, and Sibel Adal. 2013. Cutting through the noise: Defining ground truth in information credibility on twitter. *Human* 2, 3 (2013), 151–167.

[51] Douglas R. Stinson. 1991. Universal Hashing and Authentication Codes. In *CRYPTO (Lecture Notes in Computer Science, Vol. 576)*. Springer, 74–85.

[52] Dmitry Supranovich and Viachaslau Patsepnia. 2015. IHS_RD: Lexical Normalization for English Tweets. In *NUT@IJCNLP*. Association for Computational Linguistics, 78–81.

[53] Rob van der Goot. 2019. MoNoise: A Multi-lingual and Easy-to-use Lexical Normalization Tool. In *ACL*. Association for Computational Linguistics, 201–206.

[54] Rob van der Goot, Alan Ramponi, Tommaso Caselli, Michele Cafagna, and Lorenzo De Mattei. 2020. Norm It! Lexical Normalization for Italian and Its Downstream Effects for Dependency Parsing. In *LREC*. European Language Resources Association, 6272–6278.

[55] Michael Völske, Martin Potthast, Shahbaz Syed, and Benno Stein. 2017. TL;DR: Mining Reddit to Learn Automatic Summarization. In *NFiS@EMNLP*. Association for Computational Linguistics, 59–63.

[56] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *ICLR (Poster)*. OpenReview.net.

[57] Kang Zhao, Hongtao Lu, and Jincheng Mei. 2014. Locality Preserving Hashing. In *AAAI*. AAAI Press, 2874–2881.

A PROOF FOR ROBUST ANALYSIS

We give detailed proofs about the theoretical result in Section 3.3. The primary tool for our analysis will be the following Chernoff bounds [37], which we can apply since the weight of each edge is the sum of independent indicator random variables.

Lemma A.1 (Chernoff Bound on Edge Weight Estimation). *Define the weight of a particular edge be $X = \sum_{i=1}^{T} X_i$, where all X_i are independent and T is the number of repetitions in Flan. If $X_i = 1$ with probability $\geq p$ and $X_i = 0$ with probability $\leq 1 - p$, then $\mathbb{E}[X] \geq \sum_{i=1}^{T} p = Tp$. The lower tail bound is:*

$$P[X \leq (1-\delta)Tp] \leq \exp\left(\frac{-Tp\delta^2}{2}\right) \qquad \forall \delta \in (0,1) \quad (10)$$

If $X_i = 1$ with probability $\leq q$ and $X_i = 0$ with probability $\geq 1 - q$, then $\mathbb{E}[X] \leq \sum_{i=1}^{T} q = Tq$. The upper tail bound is:

$$P[X \geq (1+\delta)Tq] \leq \exp\left(\frac{-Tq\delta^2}{3}\right) \qquad \forall \delta > 0 \quad (11)$$

Following the same notation as Definition 2.1, let p denote the minimum probability of an intra-cluster edge and q be the maximum probability of an inter-cluster edge. Let $\mathcal{G}$ denote the graph produced by our algorithm. The probability of the graph $\mathcal{G}$ having the edge $e(w_i, w_j)$ is:

$$P\left[e(w_i, w_j) \in \mathcal{G}\right] \begin{cases} \geq p & \text{if } C(w_i) = C(w_j) \\ \leq q & \text{otherwise} \end{cases} \quad (12)$$

In practice, we have $p \gg q$, as we expect the lexical similarity between words in the same component to be larger than those across the components. LSH seeks to estimate p and q as modeling the Jaccard similarity between words. The stabilization step is T *i.i.d* coin flips with probabilities p or q. The threshold α for removing insignificant edges are chosen empirically so that $p > \alpha > q$.

Proposition A.2 (False Positive Probability). *Fix a node w_i. The probability that Flan will connect w_i to a node in a cluster c_j where $c_j \neq C(w_i)$ is at most $|c_j| \exp\left(\frac{-T(q-\alpha)^2}{3q}\right)$. $|c_j|$ is the size of cluster c_j.*

Proof. Recall that $e(w_i, w_k)$ denotes the weight assigned to edge (w_i, w_k). The probability of w_i connecting to any of the word in cluster c_j:

$$\begin{aligned} &P[\exists w_k \in c_j, e(w_i, w_k) \in \mathcal{G}] \\ &\qquad \leq \sum_{j=1}^{|c_j|} P\left[e(w_i, w_k) \geq \alpha T\right] && \text{Union Bound} \\ &\qquad \leq |c_j| \exp\left(\frac{-Tq(\alpha/q - 1)^2}{3}\right) && \text{Equation (11)} \\ &\qquad \leq |c_j| \exp\left(\frac{-T(q-\alpha)^2}{3q}\right) \end{aligned}$$

where the second inequality follows by setting $\delta = \alpha/q - 1 > 0$. □

The above proposition implies that the probability of a false positive event decreases exponentially with more repetitions T. As a practical illustration of the power of this bound, if we take $q = 0.05, T = 10, \alpha = 1/2$, and $|c| = 100$, we find that the probability of this bad event is at most 0.000014.

Proposition A.3 (False Negative Probability). *Fix a node w_i. The probability that Flan will not add an edge from w_i to any of the other nodes in $c_j = C(w_i)$ is at most $\exp\left(\frac{-|c_j|T(p-\alpha)^2}{2p}\right)$.*

Proof. We note that w_i does not share edge with some other word $w_k \in c_j$ in Flan graph if the edge weight is smaller than αT after applying T repetitions. By the fact that the presence of each edge is an independent event and another Chernoff bound, we have that

$$\begin{aligned} &P[\forall w_k \in c_j, e(w_i, w_k) \notin \mathcal{G}] \\ &\qquad = \prod_{k=1}^{|c_j|} P[e(w_i, w_k) \leq \alpha T] \\ &\qquad \leq \left(\exp\left(\frac{-Tp(1-\alpha/p)^2}{2}\right)\right)^{|c_j|} && \text{Equation (11)} \\ &\qquad \leq \exp\left(\frac{-|c_j|T(p-\alpha)^2}{2p}\right) \end{aligned}$$

where the first inequality follows by setting $\delta = 1 - \alpha/p \in (0,1)$. □

B EXPERIMENTAL CONFIGURATIONS

B.1 Implementation Details

For comparison fairness, we use the same procedure of feeding the inputs into and extracting the outputs from the algorithms. In addition, we use parallelism to address computational bottlenecks in the two methods, albeit in different ways. In Flan, we use every process to run one repetition of the LSH algorithms. In spell-correction, we let every process do spell-correction for a batch of the words. For the `autocorrect`[1] and Hunspell[2] libraries, they apply one and two steps of operation (i.e., replace, delete, replace and insert) for every input word. For all the word variants in the dictionary, it outputs a word with the highest frequency.

For the hyper-parameters in our experiments, we set `CHARS` to be $[1, 2, 3, 5, 7]$. The hash library for converting a string into a hash code is `xxhash`[3]. The number of repetitions of LSH is set to $T = 20$. We note that the number of repetitions determines the memory and also the running time of the Flan algorithm. A higher number of repetitions give us a higher quality normalization while a lower number gives us a faster algorithm. We set the threshold ratio $\alpha = 0.2$ for removing low weight edges. Furthermore, we set the universe size to $U = U' = 2^{32}$ and partition the space into 4 bins. The 2-universal hashing function we use in our experiments is $h(x) = (ax + b) \mod P$, where $a \sim [1, U], b \sim [0, U - 1]$ and the prime number $P = 2^{31} - 1$. As there are several random functions used in Flan, we fix a random seed value for reproducibility. Every algorithm runs over 24 cores CPU with a frequency of 3.8 GHz.

The twitter dataset is the sentiment140 dataset[4] and the Reddit dataset is constructed from Reddit's comments dump[5]. The datasets

[1] https://github.com/filyp/autocorrect

[2] https://github.com/tokestermw/spacy_hunspell

[3] https://github.com/ifduyue/python-xxhash

[4] https://www.kaggle.com/kazanova/sentiment140

[5] https://files.pushshift.io/reddit/comments/RC_2021-06.zst

for GLUE benchmark is from[6]. The DistilBert we use for finetuning GLUE subtasks are from[7].

B.2 Quality Comparison Experiment Details

We evaluate the correction results from different algorithms by native speakers on Amazon Turk[8]. To measure the quality of the corrections by Flan, we conduct a human evaluation study to evaluate the algorithms’ performance. We first select 300 testing instance dataset that were corrected by either Flan or the spell-correction method. Then we deploy the questionnaire to the AmazonTurk for native speaker to labels it. The evaluators did not know if a correction came from Flan or the spell corrector, and each reviewer was tasked with labeling word corrections as either “good", “neutral”, “bad”, and “not sure”. We define a “good" corrections as one that query is better for human understanding or one that eliminates unnecessary punctuation; “neutral” denotes a correction does not help to make the meaning of the query more clear, but does not hurt either, such as converting a word from plural to singular; “bad” corrections are those that make the query less coherent. Finally, we use “not sure" to capture the remaining cases. To give a quantitative result, we consider “good", “neutral” as correct correction and regard “bad” and “not sure” as incorrect correction.

C CASE STUDIES

C.1 Ablation Study on Threshold

In Figure 3, we plot the effect of the graph pruning threshold α on the behavior of Flan on Reddit dataset. When $\alpha = 0$ we see that Flan corrects nearly every word in the corpus. However, when we set $\alpha = 0.1$ or $\alpha = 0.2$, we note that this correction coverage drops rapidly, which empirically demonstrates the exponential decay from applying more repetitions that we discussed previously. Based on the results on two datasets, we selected $\alpha = 0.2$ as the pruning threshold in our experiences since it provided a balance between covering words and not introducing too much noise.

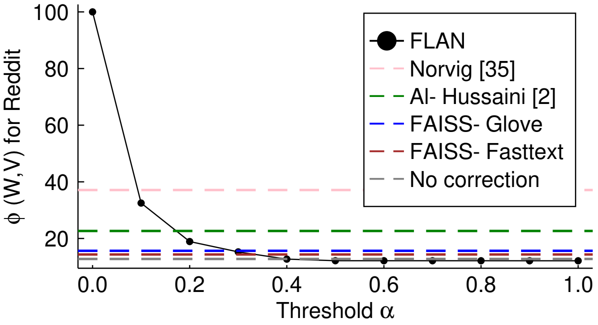

Figure 3: Percentage of words covered ($\phi(\mathcal{W}, \mathcal{V})$) on the Twitter and Reddit datasets. Flan’s coverage is determined by the threshold α.

C.2 Connected Components in Flan Graph

We collect the results of select connected components after the LSH mapping, repetition, and pruning steps. The left column in Table 5 is the representative word for the connected component while the right column illustrates other words in the connected component that are mapped to the representative.

Table 5: Connected Components in the constructed graph of Flan over Twitter Dataset. The left column is the representative word for every connected component and the right column shows other words in the corresponding connected component. We observe that Flan can capture patterns from typographical errors on keyboards.

Representative	Connected Components
there	thereâ, therem, therea, ithere, therer
night	gnight, nightï, nightâ, gnightâ, dnight, nighti
friends	friend, friendsss, friendz, friendss, friendzz, friendsssss, myfriends, friendssss, vfriends, myfriend, friendâ, friend1
feeling	feelin, feelingz, feelingg, feelinga, feelinf, feelinfg
morning	mornings, gmorning, morningg, gmornin, morningss, morningo, gmorningg, smorning, morningstar, morningâ, morningon
amazing	amazingg, amazinggg, mazing, mazinggg, amazinggggg, amazinggggggg, amazingggg, amazinggggggggggg, amazingggggggggg, mazingggg, amazinggggggggg, soamazing, amazings, amazingggggg

From Table 5, we observe that Flan can successfully group words with minor character difference into the same connected component. These results also provide evidence to the effectiveness of our graph pruning strategy in preventing spurious hash collisions from leading to unrelated word matches. This table also shows that Flan tends to convert words of plural form into singular form or vice versa based on the frequency distribution of these variations in the dataset. In addition, Flan is able to map infrequent words to a meaningful and frequent words in the indexed vocabulary, such as “amazinggggg” and “amazingggggg”. In addition, Flan captures typos related to the characters in close keyboard proximity such as “feelinf” as well as fixing the habit of double typing (“feelingg” and “gmorningg”).

Ultimately, these results suggest that lexical normalization can aid in improving the quality of text-based models applied to noisy data, and Flan provides a computationally scalable alternative to existing methods.

[6] https://huggingface.co/datasets/glue
[7] https://huggingface.co/distilbert-base-uncased
[8] https://www.mturk.com/