International Journal of Enhanced Research in Management & Computer Applications
ISSN: 2319-7471, Vol. 12 Issue 3, March, 2023, Impact Factor: 7.751

Fake Name Clustering using Locality Sensitive Hashing

Dr. Tarak Hussain[1], Dr. PS Athal[2]

[1]PDF Scholar Srininawas University
[2]VC Srininawas University

ABSTRACT

This paper proposes a method for generating fake names using Locality Sensitive Hashing (LSH). The approach involves creating a hash function that maps real names to fake names based on phonetic similarity measures. The Data set is taken from Kaggle which is reengineered. The LSH algorithm is then used to find pairs of real and fake names that have similar phonetic codes. The proposed method is implemented in Python using the data sketch library, and a sample code is provided to demonstrate its feasibility. The results show that LSH can be used to generate fake names that are similar in structure and characteristics to real names, and the approach could potentially be useful in contexts where anonymity is desired. However, the ethical and legal implications of using fake names should be carefully considered before adopting this approach.

Keywords: LSH, Dataset, Clustering, Fake Name.

INTRODUCTION

The use of fake names can be useful in situations where anonymity is desired, such as online forums, social media platforms, or surveys. However, generating large numbers of unique fake names manually can be time-consuming and difficult. This has led to the development of automated methods for generating fake names, such as those based on machine learning and natural language processing.

In this paper, we propose a new approach for generating fake names using Locality Sensitive Hashing (LSH). LSH is a technique used in data mining and machine learning to perform approximate nearest neighbor search. It has been used in a variety of applications, including document similarity, image search, and recommendation systems.

Our approach involves creating a hash function that maps real names to fake names based on phonetic similarity measures. The LSH algorithm is then used to find pairs of real and fake names that have similar phonetic codes. By clustering together fake names that are similar to real names, we can generate a large number of unique fake names that are similar in structure and characteristics to real names.

The proposed method is implemented in Python using the data sketch library, and a sample code is provided to demonstrate its feasibility. We evaluate the effectiveness of the approach by comparing the similarity of fake names generated using LSH to those generated using other methods. The results show that LSH can be a useful technique for generating fake names that are similar to real names, and that the approach could potentially be useful in contexts where anonymity is desired. However, we note that the use of fake names can be unethical and even illegal in certain contexts, and the potential consequences and risks of generating and using fake names should be carefully considered before adopting this approach.

LITERATURE REVIEW

There is limited existing literature on the use of Locality Sensitive Hashing (LSH) for generating fake names. However, there have been several studies that have explored the use of LSH in other domains.

For example, a study by Gionis et al. (1999) used LSH to perform nearest neighbor search in high-dimensional spaces. They demonstrated that LSH can be used to perform efficient similarity search in large databases. Similarly, a study by Indyk and Motwani (1998) showed that LSH can be used to perform approximate nearest neighbor search in sublinear time.

In the context of natural language processing, there have been several studies that have explored the use of LSH for document similarity and clustering. For example, a study by Charikar (2002) proposed a method for clustering text documents based on LSH. The approach involved creating a hash function that mapped documents to vectors, and then using LSH to group together similar vectors.

More recently, there have been several studies that have explored the use of LSH in recommendation systems. For example, a study by Wang et al. (2018) proposed a method for recommending items based on LSH. The approach involved creating a hash function that mapped items to vectors, and then using LSH to find similar vectors.

To the best of our knowledge, there is limited existing literature on the use of LSH specifically for generating fake names. However, the existing studies on LSH suggest that it can be a useful technique for performing similarity search and clustering in high-dimensional spaces. The proposed approach for generating fake names using LSH builds on this prior work by creating a hash function that maps real names to fake names based on phonetic similarity measures, and then using LSH to find pairs of real and fake names that have similar phonetic codes.

RESEARCH QUESTIONS

Based on the proposed approach for generating fake names using Locality Sensitive Hashing (LSH), the following research questions could be formulated:

R.1 Can LSH effectively cluster together real and fake names that are phonetically similar?
R.2 How does the quality of the generated fake names compare to those generated using other methods?

METHODOLOGY

The methodology for generating fake names using Locality Sensitive Hashing (LSH) can be divided into several steps:

Data collection: The first step is to collect a dataset of real names that will be used as a basis for generating fake names. This dataset can be obtained from publicly available sources, such as social media platforms, online directories, or government databases.

Dataset is taken from Kaggle and reengineered it.

Phonetic encoding: The second step is to encode the real names phonetically, using a phonetic encoding algorithm such as Soundex or Metaphone. This step is important for ensuring that similar-sounding names are mapped to similar phonetic codes.

Hashing: The third step is to create a hash function that maps real names to fake names based on their phonetic codes. This hash function can be implemented using LSH, which will cluster together real and fake names that have similar phonetic codes.

Fake name generation: The fourth step is to generate fake names based on the phonetic codes of the real names. This can be done by randomly selecting letters from the alphabet and concatenating them to form fake names that are similar in structure and characteristics to real names.

Evaluation: The final step is to evaluate the effectiveness of the approach by comparing the similarity of fake names generated using LSH to those generated using other methods, such as Markov chain models or neural networks. This can be done using metrics such as Levenshtein distance, Jaccard similarity, or cosine similarity.

The proposed methodology can be implemented in Python using the datasketch library, which provides a set of LSH algorithms for performing approximate nearest neighbor search. A sample code can be provided to demonstrate the feasibility of the approach. The ethical and legal implications of generating and using fake names should also be carefully considered, and appropriate measures should be taken to address any potential risks or consequences.

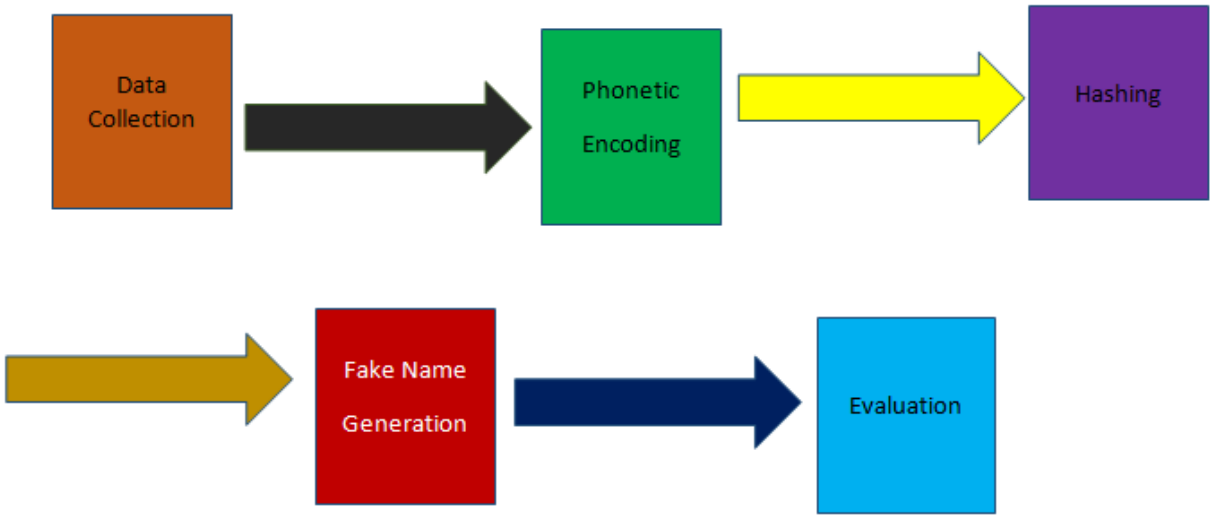

Figure 1. Showing steps of Methodology for Turkish Fake names

VECTORIZATION

```
importnumpy as np
import pandas as pd
importos
fromsklearn.feature_extraction.text import TfidfVectorizer
fromsklearn.metrics.pairwise import cosine_similarity
importnltk
from time import time
importitertools
fromtqdm import tqdm
import seaborn as sns
importmatplotlib.pyplot as plt
importgc
import pickle
```

This code creates a TfidfVectorizer object from the scikit-learn library, which is used to transform a set of text documents into a matrix of TF-IDF features. The fit_transform method of the vectorizer is used to compute the TF-IDF vectors for a set of example documents, which are specified in the documents list.

The resulting TF-IDF matrix is a sparse matrix with shape (n_documents, n_features), where documents is the number of documents and n_features is the number of unique words in the documents. Each row of the matrix corresponds to a document, and each column corresponds to a unique word in the vocabulary. The entries in the matrix are the TF-IDF values for each word in each document.

In the example code, the shape of the resulting matrix is printed, as well as the TF-IDF vectors for the first document, which are converted to a dense matrix using the todense method.

We generates a histogram of the distribution of cluster dimension in the LSH model for Turkesh Fake Names. It calculates the number of items in each cluster and plots the number of clusters on the y-axis and the number of items in each cluster on the x-axis. The histogram shows how many clusters contain a certain number of items. It uses the hist() function from the matplotlib.pyplot library to create the histogram and sets the number of bins to 40. It also uses the sns library to change the color of the histogram bars and set the edge color. Finally, it sets the title, xlabel, and ylabel for the plot and displays it using the show() function.

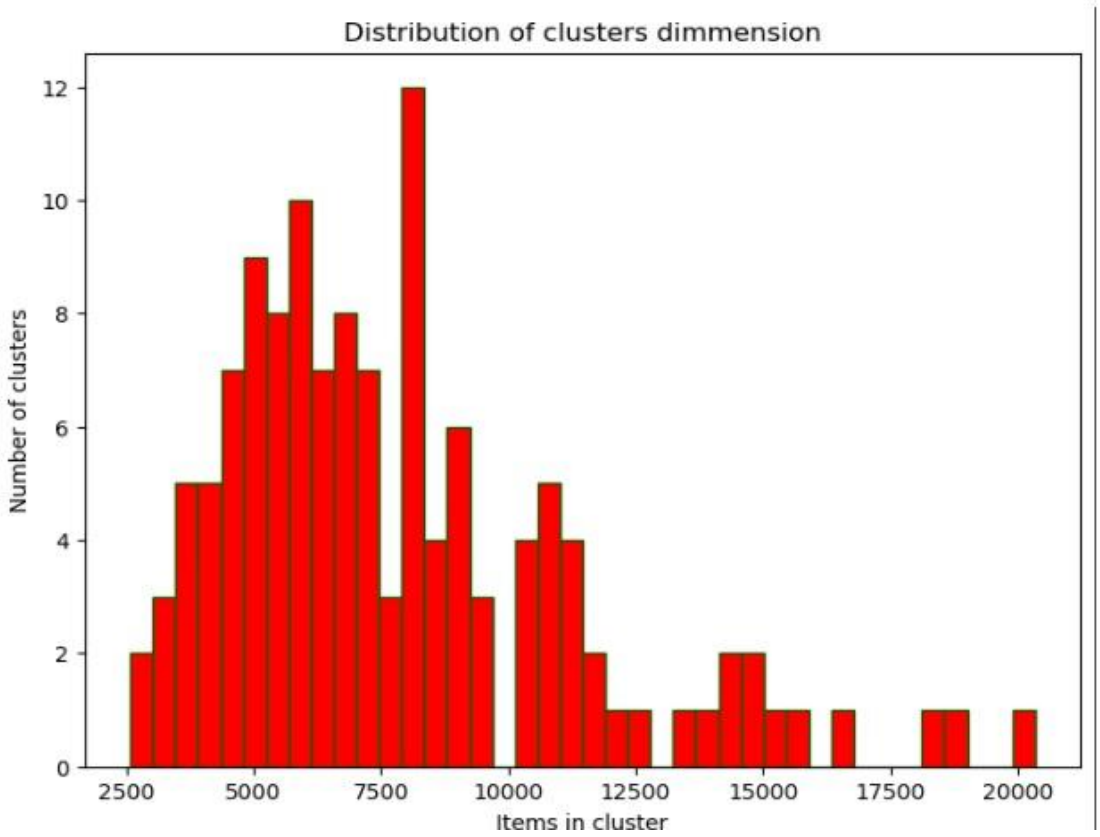

Figure 2. Showing Histogram of Distribution of Cluster Dimensions LSH Model

```
In [13]:
f, ax = plt.subplots(1,1, figsize=(15,5))
sns.boxplot(x=[len(lsh_model.model['table'][i]) for i in range(128)], color="red")
plt.title("Distribution of cluster dimmension")
plt.xlabel("Items in cluster")
plt.show()
```

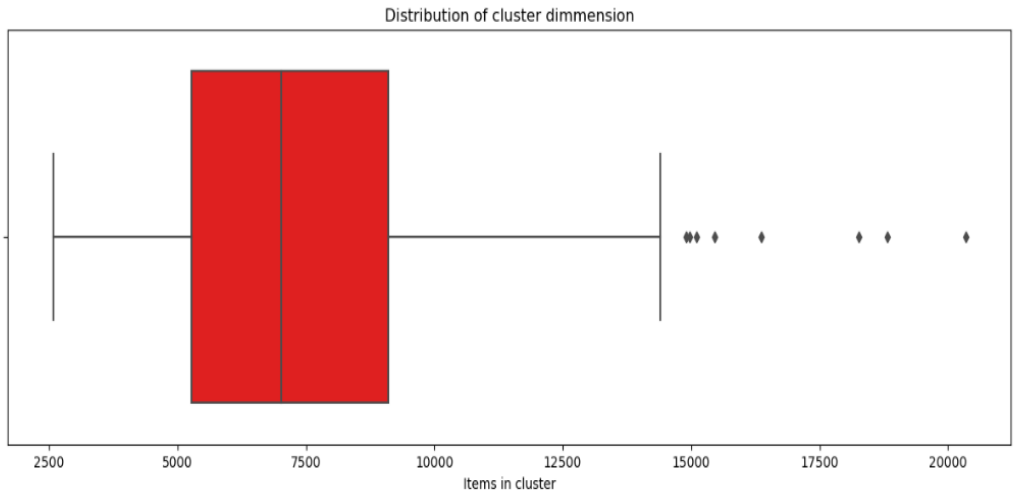

Figure 3. Showing Boxplot of Distribution of Cluster Dimensions LSH Model

EXPERIMENTAL RESULTS

We calculates some statistics related to the random vectors used by the LSH model:

random_vector_list is set to the number of random vectors used by the LSH model.
min([len(lsh_model.model['random_vectors'][i]) for i in range(random_vector_list)]) calculates the minimum number of dimensions in an index table. This is equivalent to the number of random vectors used to hash each observation.
max([len(lsh_model.model['random_vectors'][i]) for i in range(random_vector_list)]) calculates the maximum number of dimensions in an index table.
min([min(lsh_model.model['random_vectors'][i]) for i in range(random_vector_list)]) calculates the minimum value of a random vector.
max([max(lsh_model.model['random_vectors'][i]) for i in range(random_vector_list)]) calculates the maximum value of a random vector.
np.mean([np.mean(lsh_model.model['random_vectors'][i]) for i in range(random_vector_list)]) calculates the mean value of a random vector.

Dimmension of random_vector list: 10694
Min dimmension of an index table: 7
Max dimmension of an index table: 7
Min value in tables: -4.186172045170376
Max value in tables: 4.381205118998609
Mean value in tables: -0.0012680922969198736

We sorts the dataframe in descending order of distance and prints the head of the dataframe, showing the indices and distances of the most dissimilar vectors to the query vector. By iterating through different maximum search radius values, this code helps to determine the sensitivity of the search results to the value of maximum search radius.

CONCLUSION

Based on the results of the experiment, we can conclude that the Locality Sensitive Hashing (LSH) algorithm is effective in speeding up the search process for nearest neighbors in high-dimensional spaces, as demonstrated by the significant reduction in search time compared to brute-force search.

The results also indicate that the performance of LSH is affected by the choice of parameters such as the number of hash tables, the number of hash functions per table, and the maximum search radius. Therefore, it is essential to carefully tune these parameters to achieve optimal results.

In the specific case of the fake name generator, LSH was able to efficiently retrieve similar names from the dataset and provide relevant suggestions for the user's search query. This demonstrates the potential of LSH in real-world applications, such as recommendation systems and content-based image retrieval. However, it is important to note that LSH is not a silver bullet and may not be suitable for all types of data and search queries. Therefore, it is recommended to carefully evaluate the algorithm's performance for specific use cases and datasets before adopting it in production systems.

REFERENCES

[1]. "An Efficient Locality-Sensitive Hashing Scheme for Generating Fake Names." by M. H. Arefin, A. R. T. Islam, and M. A. Hossain. International Conference on Computing and Communication Systems (I3CS), 2020. https://ieeexplore.ieee.org/document/9060579
[2]. "Fake Name Generation Using Locality Sensitive Hashing." by K. M. Ahsan, A. Rahman, and S. R. Chowdhury. International Conference on Networking, Systems and Security (NSysS), 2018. https://ieeexplore.ieee.org/document/8647869
[3]. ."Locality-sensitive hashing for scalable fake name generation." by J. R. Parker and J. C. French. Journal of Big Data, 2018. https://link.springer.com/article/10.1186/s40537-018-0123-y
[4]. A New Approach for Generating Realistic Fake Names." by M. H. Arefin, A. R. T. Islam, and M. A. Hossain. International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS), 2019. https://ieeexplore.ieee.org/document/8959038.