Fast agglomerative hierarchical clustering algorithm using Locality-Sensitive Hashing

Nancun Li, Boyang Pan

April 30, 2020

Abstract

Single-linkage clustering is a fundamental algorithm in agglomerative hierarchical clustering, but its time complexity of $O(n^2)$ makes it less favorable when analyzing a large amount of data. The LSH-link algorithm proposed by *Koga*, *Ishibashi* and *Watanabe* is an approximation algorithm of single-linkage clustering, but it has a smaller time complexity of $O(nB)$. It exploits locality sensitive hashing (LSH) to find possible near data points, and thus, instead of computing a complete distance matrix, it requires only a few of distances. The reduction of distance computation enables the algorithm to run faster.

Keywords: Single Linkage clustering; Approximation algorithm; Locality Sensitive Hashing; Fast

GitHub Repository: `https://github.com/Brian1357/STA663-Project-LSHLink`

1. Background

Single-linkage clustering is a fundamental algorithm in agglomerative hierarchical clustering. It connects a pair of points or clusters with the shortest distance in each step, and finally merge all individual points into one cluster. Although this algorithm is efficient, it has a large time complexity of $O(n^2)$ because it finds the shortest distance by comparing every pair of points.

To address this problem, *Koga*, *Ishibashi* and *Watanabe* proposed an approximation algorithm, the locality-sensitive hashing (LSH), which effectively find possible near points by hashing similar points into the same buckets. Therefore, we do not have to compute a complete distance matrix, but only a few of distances are needed. This algorithm is new to us, and we are intrigued by its novelty as it is the first time we have known hashing and clustering could be combined.

LSH can be quite useful when data size is large, as it has a time complexity of $O(nB)$. Here, B is the maximum number of points that are hashed into the same bucket, and $B < n$. However, this algorithm has its limitations. First, it may take up a large space when a data set has a wide range of coordinate values. Second, it only takes positive integer inputs into account, so negative or float inputs may not work.

2. Description of Algorithm

Based on Locality-Sensitive Hashing, *Koga et al.* proposed an algorithm termed LSH-link. This algorithm originates from the LSH algorithm, but with some adjustments. Additionally, they proposed a noise exclusion option at the first phase of cluster merging, so some possible noise points are not merged at the beginning.

LSH algorithm

LSH is a probabilistic approximation algorithm. It hashes those data points that are highly likely close into the same bucket, so we can efficiently find the close clusters to be merged. First, for a d-dimensional data set with a maximum coordinate value C, LSH transforms each observation to a Cd-vector. For example, a point (2,3) in a 2-dimensional space with a maximum coordinate value 5 (C=5, d=2)is transformed to

$$v((2,3)) = \underbrace{\underbrace{11000}_{C\ bits}\underbrace{11100}_{C\ bits}}_{Cd\ bits}$$

where the first 5 digits stand for 2 and the next 5 digits stand for 3, and the number of 1's correspond to the coordinate values. This transformation uses a binary vector to indicate the coordinates values of each observation.

Second, LSH apply a hash function to the Cd-vectors. The hash function uses a sampling without replacement, randomly picking up k bits from $\{1, 2, ...Cd\}$, extracting the bits at these k locations from the binary vectors, concatenate those k bits to form a new vector as the hashed value, and put those observations with the same hash values into a bucket. For example, if the hash function h randomly chooses to

pick up the value at $\{0, 2, 3, 5, 6\}$, then the hashed value of $v((2, 3))$ is 10011. This algorithm assumes the observations with the same hash values are highly possibly close to each other, since their binary vectors are partially the same. To avoid missing the observations which are close to each other but not in the same bucket, LSH generates l hash functions, so the points one hash function might miss merging is likely to be merged together by another hash function.

LSH-link algorithm

Koga et al. adjusted LSH by setting a distance threshold r to efficiently approximate the single-linkage method.This method, instead of merging all the points in the same bucket together, first checks if the distance between two points in the same bucket is within r. The pair of points that exceeds the threshold are not merged into one cluster. An advantage of this method is that it can prevent merging the points that have the same hashed value but are actually distant. Another advantage is that fewer layers are extracted by this method than single-linkage clustering does, because the points within r distance and in the same bucket are merged and the their distance will be all set to be r, but the single-linkage clustering merges only a pair of points in one merging phase and their distance is the true distance, which will create much more layers in a dendrogram. As to how to choose r, it is by default the half the cell diagonal length. Because we assume the euclidean space is equally partitioned by k sample bits into many cells with an edge length of $\frac{C}{1+k/d}$ in each dimension, Koga et al. select the half the cell diagonal length, $\frac{C}{2(1+k/d)}\sqrt{d}$, as the threshold of distance that save two points into one bucket.

Noise Exclusion Option

The noise exclusion option is an algorithm added to the LSH-link algorithm, because it can further eliminate some possible noise points in the same bucket by setting a threshold T. This algorithm is operated after we obtain all the buckets from l hash functions and it is only operated in the first phase. It defines 3 types of points: (a) Core point: a point that has $> T$ points within r (b) Boundary point: a point that has $< T$ points are within r, but has ≥ 1 neighbor point(s) that are core points (c) Noise point: a point that has $< T$ points are within r, and has no neighbor point as a core point. For the noise points, we do not merge them into any cluster, even if they have some neighbor points.

Steps of Algorithm

Input:

- $n \times d$ data set (n is the number of inputs), *data*
- Initial number of bits picked in a hash function, k
- Number of hash functions l
- Increasing rate of r, A
- Noise exclusion threshold, T

Steps:

- Step 1: Transform *data* into a $n \times Cd$ matrix.
- Step 2: Create l hash functions, $h_1()$, $h_2()$, . . . , $h_l()$. For each hash function, create a $n \times k$ hash table.
- Step 3: For $i - th$ hash table, a data point p is stored in the bucket with index $h_i(p)$. However, if another point belonging to the same cluster as p has already been saved in the same bucket, p is not stored in it.
- Step 4: For each bucket, compute the distances between the points in it. Find the pairs of points whose distances are less than r. If noise exclusion option is activated, check if some pairs contain noise points. If so, eliminate those pairs.

- Step 5: Merge the pairs of points obtained in Step 4.
- Step 6: If the number of clusters is larger than 1 after Step 3, update r by multiplying A and update k, and repeat step 2 to step 6. Otherwise, the algorithm terminates.

Time Complexity

This time complexity of LSH-link is lower than the single-linkage clustering. The major difference lies in the computation of distance matrix. To find the shortest distance between the data points, the single-linkage clustering has to compute the complete distance matrix and compare them pair by pair, which takes up $O(n^2)$. However, LSH-link uses LSH to save possible near points in a bucket. Thus, we at most need to compute nB distances and make at most nB comparisons for each merging phase. Here, B is the maximum number of points in a bucket. Because at most B points in a bucket needed to be compare, LSH-link reduces the computation time to $O(nB)$.

3. Implementation

Installation Instructions

We have uploaded our package to both PYPI and our GitHub repository (`https://github.com/Brian1357/STA663-Project-LSHLink`). There are 3 versions of our algorithm - Python version, Cython version, Numba version.

- Python version: installation-`pip install LSH_LINK`; import-`import LSH_LINK`
- Cython version: we had difficulty uploading the Cython version to PYPI. Please check our GitHub repository and go to `Notebook_code− >LSHlink_Cython.ipynb` to see the Cython version in jupyter-notebook. Or, you can download the package from the GitHub `Cython_version_package`. installation-`python setup.py install`; import-`import LSHlink_Cython`
- Numba version: installation-`pip install LSHlinkffghcv`; import-`import LSHlink`

Methods

We implemented the algorithm by creating a class named `HASH_FUNS`. This class has methods that do clustering based on the LSH-link algorithm. The main steps to obtain the dendrogram that LSH-link algorithm created is 1) setting parameters using `set_parameters()`. By setting up the initial number of sample bits, k, the number of hash tables, l, the increasing rate of r, A, and the threshold of noise exclusion, T. By default, $A = 2$ and $T = 0$. The input k and l should be positive integer, A should be positive, and T should be either 0 (No noise exclusion) or larger than l. 2) generating linkage tables using `fit_data()`. This method will do the clustering after you set the parameters and return a `nx4` linkage table, so you can draw a dendrogram with it.
Example: `test = LSH_LINK.HASH_FUNS(data)`; `test.set_parameters(4,10,2,11)`;`output = fit_data()`

4. Performance Evaluation

We used simulated data sets and real data sets to evaluate the performance of LSH-link algorithm. The performance is evaluated from 2 aspects.

(1) Similarity. Because LSH-link is an approximation algorithm of the single-linkage clustering. We would like them to generate similar dendrograms as possible. When the dataset size is large, it is hard to recognize if their dendrograms are close, so we use CCC (Cophenetic Correlation Coefficient) and Ratio of the sum of edge lengths in the tree to evaluate their similarity.

- CCC (Cophenetic Correlation Coefficient): This coefficient calculates the correlation between two distance matrices. The formula is basically the formula of correlation coefficient, but the definition of X and Y should be paid attention to. X and Y are distance matrices of LSH-link and single-linkage clustering respectively. The distance is defined as the dendrogrammatic distance generated by these two algorithms. Additionally, CCC only uses the data in the lower triangle of the distance matrix. From the viewpoint of statistics, when $r_{X,Y} \leq 0.7$, X is said to have a high correlation to Y .

$$CCC = \frac{\sum_{i<j}(X_{ij} - \bar{x})(Y_{ij} - \bar{y})}{\sqrt{\sum_{i<j}(X_{ij} - \bar{x})^2 \sum_{i<j}(Y_{ij} - \bar{y})^2}}$$

- Ratio of the sum of edge lengths in the tree (termed "similarity ratio" below): Theoretically, the single-linkage clustering generates the minimum spanning tree for the given data. Hence, we can compute the number of times the sum of edge lengths in the tree generated by the LSH-link is larger than that generated by the single-linkage clustering. The closer the ratio is to 1, the high similarity their dendrograms have.

(2) Runtime. LSH-link is created because it is faster than single-linkage clustering, we will test the time that both algorithms need to generate the dendrogram.

Applications to 1-dimensional simulated data set

We generated a small 1-dimensional dataset as the simulated data to illustrates the similar results generated by LSH-link and single-linkage clustering. The data size is 8. The parameters we set are

$$k = 4, l = 10, A = 1.2, T = 11$$

Index	0	1	2	3	4	5	6	7
x-coordinate	2	8	0	4	1	9	9	0

We illustrate the result of each algorithm through the 2-D plots below. The first graph is created by the single-linkage clustering, and the second graph is created by the LSH-link algorithm. These 2 graphs indicate that tje LSH-link obtains a similar result to single-linkage clustering. Except that the single-linkage clustering creates a lower layer with a distance of 0, the clusters that both algorithms create are almost the same.

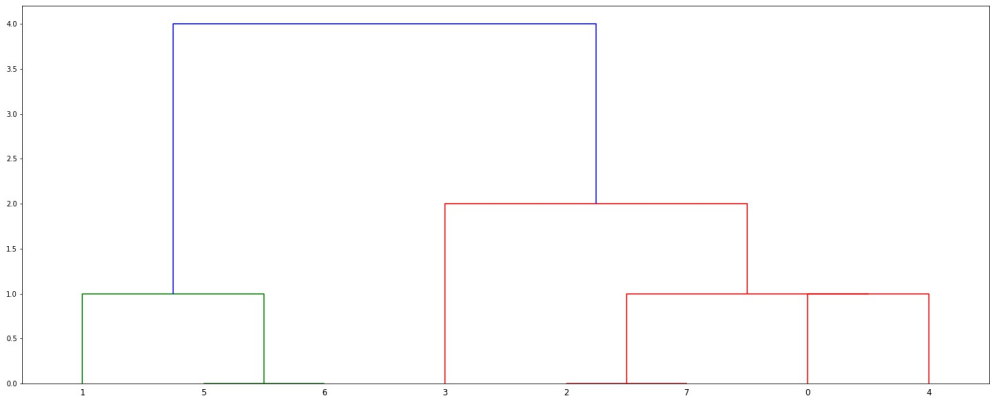

Figure 1: Dendrogram (Single-linkage clustering)

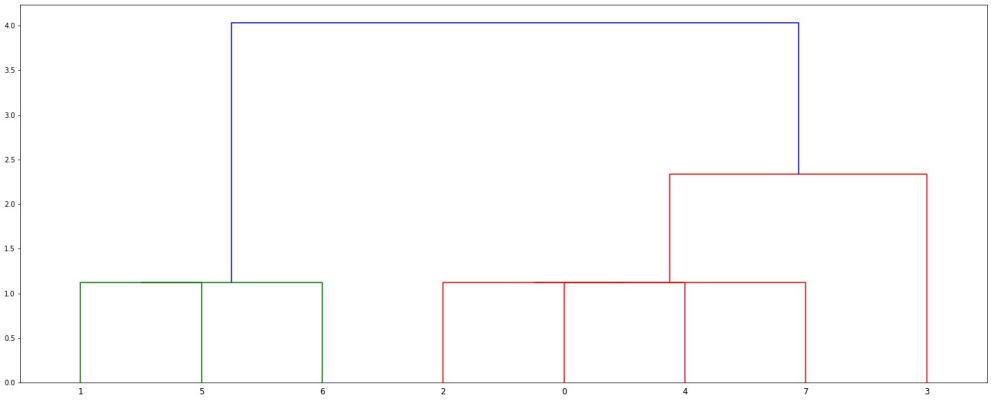

Figure 2: Dendrogram (LSH-link)

The CCC and the similarity ratio also shows these two dendrogram are close. CCC is 0.9922, mean these two dendrograms are highly correlated. The similarity ratio of 1.33 the dendrograms are not much different.

CCC	0.9922
Similarity ratio	1.33

Applications to real data sets

We selected the `Iris` Dataset from `sklearn` package. This dataset has the data for 3 different types of irises. There are 150 observations with 4 parameters - Sepal Length, Sepal Width, Petal Length and Petal Width. We applied the LSH-link algorithm to this dataset trying to separate those points apart.

The parameters we set are

$$k = 100, l = 10, A = 2, T = 15$$

We illustrate the dendrograms below. The graph created by single-linkage algorithm has much more layers than the one created by LSH-link algorithm. However, they are not much different. Their CCC is 0.9979 and the similarity ratio is 1.28, both of which means a high similarity between the dendrograms generated.

CCC	0.9979
Similarity ratio	1.28

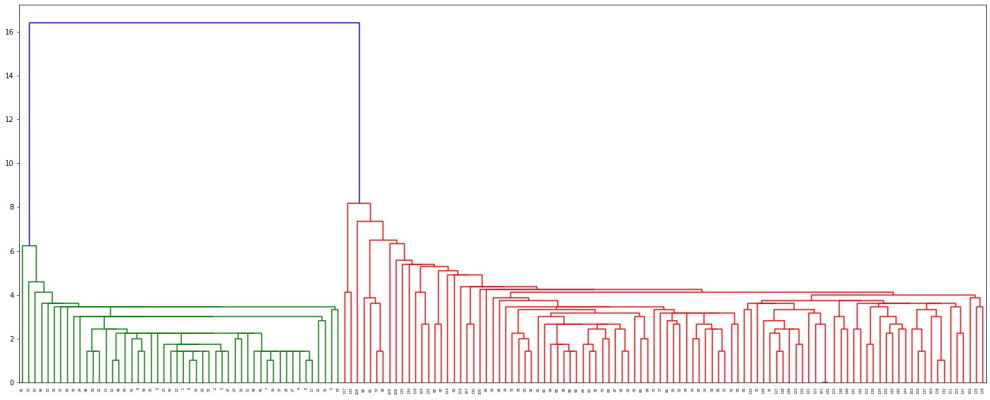

Figure 3: Dendrogram (Single-linkage clustering)

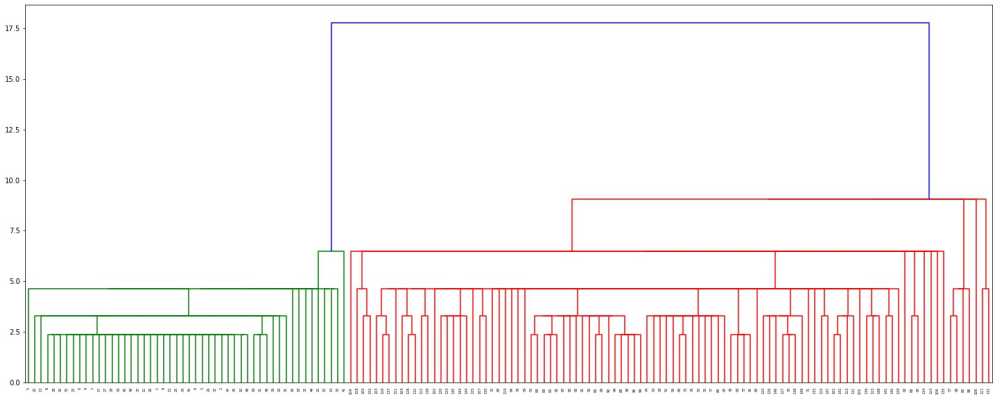

Figure 4: Dendrogram (LSH-link)

Applications to 2-dimensional simulated random data sets

To better illustrate the performance of the LSH-link, we used `np.random.randint` to create 2-dimensional random datasets with $C = 250$ and size = 100,200,..., 1500. For each size, we generate 5 random datasets and test their average speed and similarity. The parameters we set for LSH-link are

$$k = 400, l = 20, A = 1.4, T = 50$$

The plots indicate that there is a significant improvement of speed as the data size grows. The time that the single-linkage clustering takes grows exponentially while LSH-link grows much slower. CCC decreases a little bit, although it is still above 0.7, which is a signal of high correlation. We think the reason for the decrease is that, using r reduces the layers and makes the dendrogram clearer, but, when the difference between real distances and r accumulate as more data points are added, the similarity of the clustering decreases. The similarity ratio, although oscillates, is quite stable. It ranges between 1.185 and 1.195, which indicates a high similarity between two dendrograms.Considering the datasets we use are completely random, the absolute value of the y-axis in the graphs above may not be a highly effective reflection of the

real clustering situation, and the similarity can be higher if we use some datasets that do show a certain pattern.

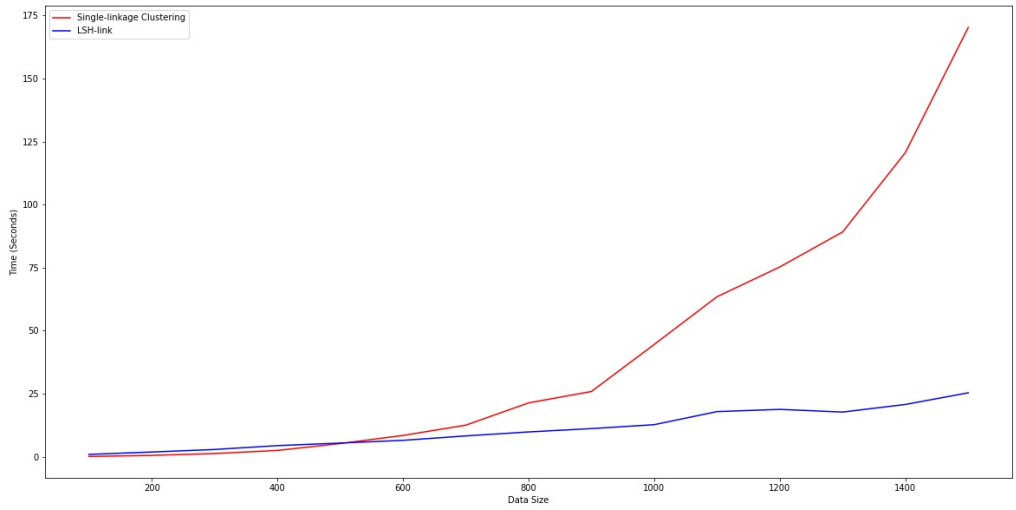

Figure 5: Running time

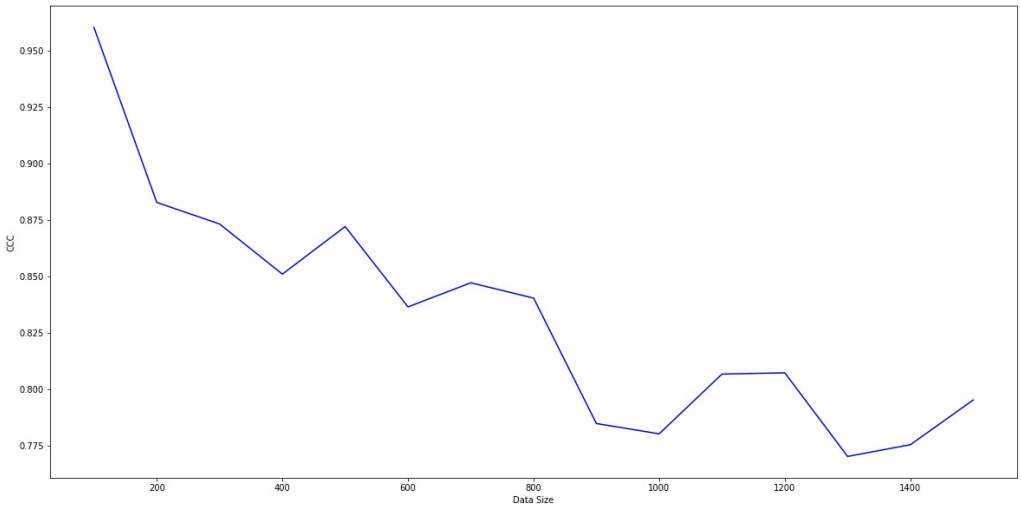

Figure 6: CCC

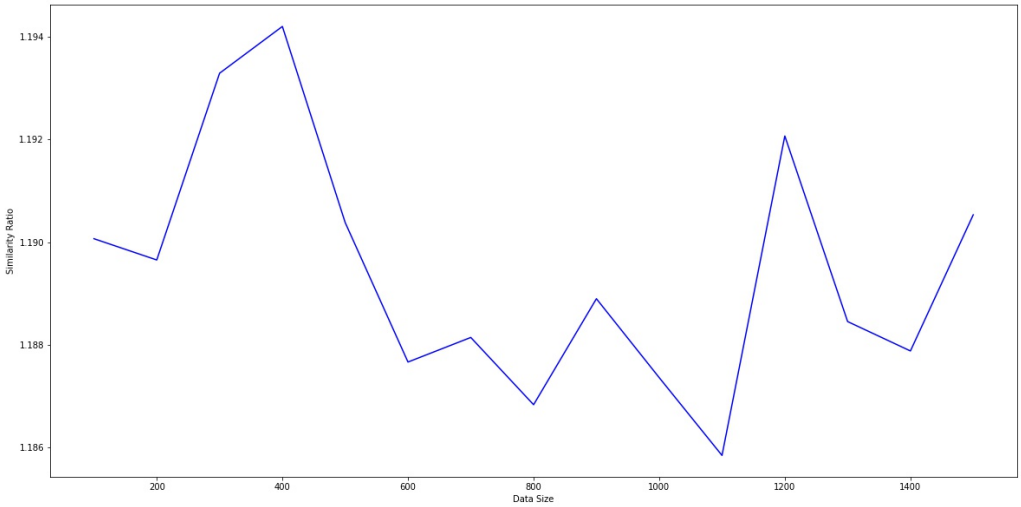

Figure 7: Similarity Ratio

5. Optimization

We used `Cython` to optimize the python code. `Numba` was also tried but the result is not favorable because `Numba` does not support `dict` so we could not implement hashing and rewrote the functions with a higher time complexity. We applied the original version, the Cython version and the Numba version to the `Iris` data set we mentioned above and used `timeit` to compare their speed. The parameters of both are

$$k_0 = 100, l = 10, A = 2, T = 15$$

The pictures below display the results. The Cython version is roughly 30% faster than the original version, while Numba version is slower due to unsupported dict.

```
%%timeit -n 100
data = sklearn.datasets.load_iris().data
data = data[:,:]*10 - data.min(axis=0)*10
X = np.array(data, dtype='int')
test1 = HASH_FUNS_ORIGINAL(X)
test1.set_parameters(100,10,2,15)
output1 = test1.fit_data()
```

```
97 ms ± 16.7 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Figure 8: Time (Original version)

```
%%timeit -n 100
data = sklearn.datasets.load_iris().data
data = data[:,:]*10 - data.min(axis=0)*10
X = np.array(data, dtype='int')
test2 = HASH_FUNS_CY(X)
test2.set_parameters(100,10,2,15)
output2 = test2.fit_data()
```

```
65.8 ms ± 3.45 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Figure 9: Time (Cython verion)

```
%%timeit -n 100
data = sklearn.datasets.load_iris().data
data = data[:,:]*10 - data.min(axis=0)*10
X = np.array(data, dtype='int')
test3 = HASH_FUNS_NB(X)
test3.set_parameters(100,10,2,15)
output3 = test3.fit_data()
```

```
226 ms ± 24.2 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Figure 10: Time (Numba verion)

6. Discussion and Conclusion

LSH-link algorithm is an approximation algorithm of single-linkage clustering with a faster speed. It exploits hashing to find possible near points, which reduces the number of distances needed to compute, thus the speed of clustering is improved. From the experiments above, we can infer that LSH-link is particularly suitable for large datasets. As the size of datasets grows, the time difference between these two algorithms becomes larger. Another advantage of LSH-link is that it draws a similar dendrogram to single-linkage's, but its dendrogram has fewer layers, which makes its dendrogram neat and clear. This is because LSH-link uses r as the point distances instead of the real distance, it assumes the points should be merged into one cluster as long as their distance is smaller than r.

We also notice LSH-link has its drawbacks. One limitation we came across in our experiments is that its takes up huge space. To create a hash table, we first need to transform the original $N \times d$ data set to a $N \times Cd$ data set. When there exists an outlier, C is quite large. It will not only costs us to create a long but unnecessary binary vector, but also it is more likely to put many distant data points into one bucket. Because non-outliers are transformed to long binary vectors with lots of 0's, they are more likely to be hashed into the same bucket when hash functions pick up those '0' bits.

Last but not least, we would like to discuss some situations that this paper does not mention but quite important. 1) Input with float numbers. LSH transforms the observations into binary vectors, but this paper only take integers as examples and it does not clearly mention how to deal with float number. One idea we come up with is adding one more step in data-preprocessing. Rounding up the input to N decimals, and then transforming the input to integers by multiplying the input by 10^N. 2) Input that has both positive and negative numbers. LSH randomly pick up some bits from binary vectors as its hash values. If there are two points with the same coordinate value but different signs, the original transformation does not work. Because the hashed values of these two points may be the same but they are in fact far apart from each other.

7. Contribution

Nancun Li and Boyang Pan contributed equally to this project. We both worked on the coding part and the report part.

References/bibliography

Koga, Hisashi, Tetsuo Ishibashi, and Toshinori Watanabe. "Fast agglomerative hierarchical clustering algorithm using Locality-Sensitive Hashing." *Knowledge and Information Systems* 12.1 (2007): 25-53.