University of Athens
Dept. of Informatics & Telecommunications
K17c: Software Development for Algorithmic Problems

Project 1

Ioanna Kontemeniotou – 1115202000227
Panagiotis Chatzimichos – 1115202000211

Introduction

In this project, we explore various strategies to solve the approximate k-NN (k-nearest neighbors) problem and the Approximate Range Search problem, focusing on the MNIST dataset of handwritten digits as our primary data source. Two main methods are applied: Locality-Sensitive Hashing (LSH) and Randomized Projection on Hypercubes.

The Euclidean distance was used to measure the distance between points across all algorithms used in this project.

We have also implemented three versions of vector clustering techniques, initializing them with the K-Means++ algorithm. These versions utilize the methods mentioned earlier, along with the standard Lloyd's algorithm, to assign data points to clusters. Finally, to assess how well our clusters were created, we used the Silhouette method for evaluation.

Our code was written in C++.

What is included

Project 1

- build - Files created by Makefile
- include - Header files
  - brute_force.h
  - cluster.h
  - common.h - Definition of some commonly used functions and structures
  - cube_projection.h
  - dataset_input.h - Structures and utilities used for reading the MNIST dataset
  - lsh.h
- modules
  - brute_force.cpp
  - cluster.cpp
  - common.cpp - Implementation of some commonly used functions and structures
  - cube_projection.cpp
  - dataset_input.cpp - Utility functions for reading the MNIST dataset
  - lsh.cpp
- src
  - main_cluster.cpp
  - main_cube.cpp
  - main_lsh.cpp
- Makefile
- cluster.conf

Usage

Build and run:

A Makefile is provided for compilation of the project. Build and run the project as follows:

```
$ make                  # Build the project
$ make run_lsh          # Run LSH
$ make run_cube         # Run Hypercube
$ make run_cluster      # Run Cluster
$ make clean            # Clean the house
```

Parameters description:

You can change the parameter values in the Makefile for each program.

Parameters for LSH:

- -d <input_file> : MNIST dataset.
- -q <query_file> : Points in the MNIST dataset format.
- -k <int> : Number of hash functions.
- -L <int> : Number of hash tables.
- -o <output_file> : File to print the results.
- -N <number_of_neighbors> : Number of nearest neighbors to be found.
- -R <radius> : Range of search.
- -labels_print <0 or 1>: Used to print the label of the queries points and neighbour points. When it is applied the two following parameters MUST be given also.
- -labels <labels file>: Path to training dataset labels.
- -qlabels <labels file>: Path to testing dataset labels.

Note: The number of query points can be tweaked inside "main_lsh.cpp" in the corresponding defined variable.

Parameters for Hypercube:

- -d <input_file> : MNIST dataset.
- -q <query_file> : Points in the MNIST dataset format.
- -k <int> : The dimension in which the points are projected.
- -M <int> : Maximum number of candidates to check. M = -1 ignores this upper bound completely.
- -probes <int> : Number of vertices.
- -o <output_file> : File to print the results.
- -N <number_of_neighbors> : Number of nearest neighbors to be found.
- -R <radius> : Range of search.
- -labels_print <0 or 1>: Used to print the label of the queries points and neighbour points. When it is applied the two following parameters MUST be given also.
- -labels <labels file>: Path to training dataset labels.
- -qlabels <labels file>: Path to testing dataset labels.

Note: The number of query points can be tweaked inside "main_lsh.cpp" in the corresponding defined variable.

Parameters for Clustering:

- -i <input_file> : MNIST dataset.
- -c <configuration_file> : A configuration file.
- -o <output_file> : File to print the results.
- -complete <optional> : Add for extra clustering info (WARNING: output file can gets very big).
- -m <method: Classic OR LSH or Hypercube> : Method to be used to find the Nearest Neighbors.

Experiments

To evaluate the performance of our k-NN solvers we have experimented with the values of the parameters focusing on two main metrics: function runtime and accuracy. Our main dataset had 60,000 points. To see how different parameters affected the solvers, we used a test set with 100 "query" points. We then computed the average values for runtime and accuracy.

1. **Run time:** Measured by the time taken by the solvers to find the nearest neighbors.
2. **Accuracy:** Checked by how well the solvers match the true nearest neighbors found by the Exhaustive Search algorithm. We have used the following formula:

$$Relative\ error = (Approximate\ distance - True\ Distance)/True\ Distance.$$

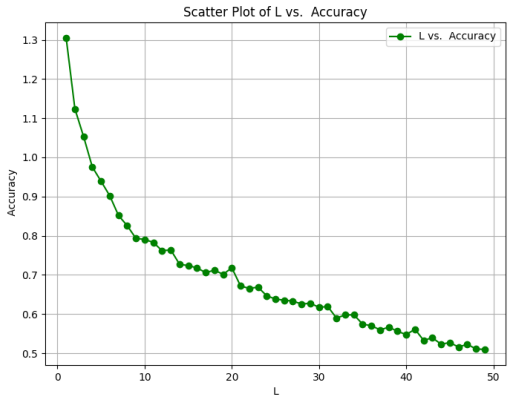
   Smaller values mean better accuracy.

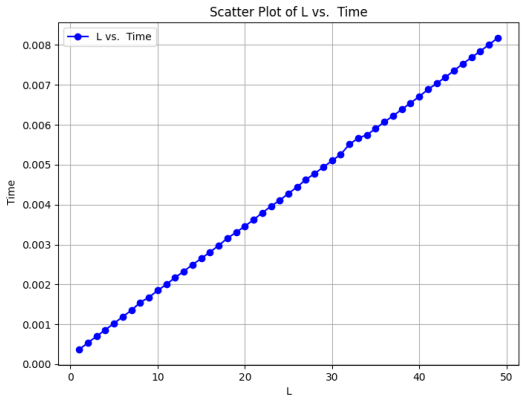
Nearest Neighbor Search

LSH Implementation

Number of hash tables (L parameter)

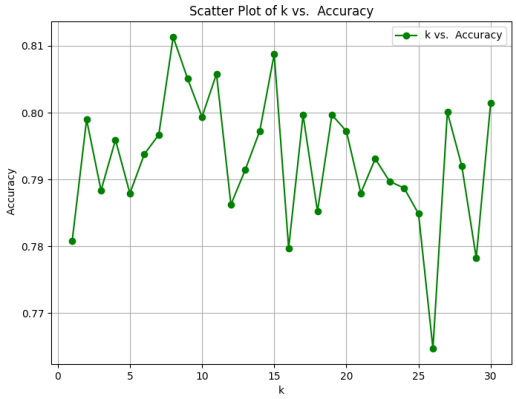
k = 5, W = 10, Table size = n/16

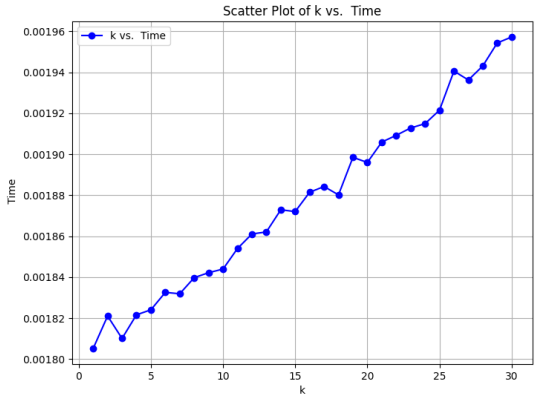




The behavior of L, which represents the number of hash tables, is consistent with our predictions. As the number of hash tables increases, more candidates are evaluated, enhancing accuracy. Additionally, the linear relationship between L and time is expected, as that the number of points checked aligns with the formula: $L \times (n/table\_size)$.

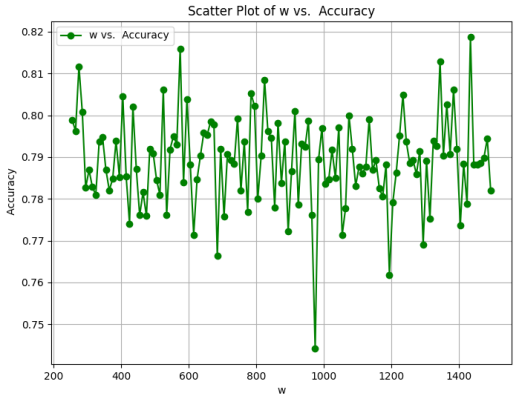
Number of hash functions (k parameter)
L = 10, W = 10, Table size = n/16

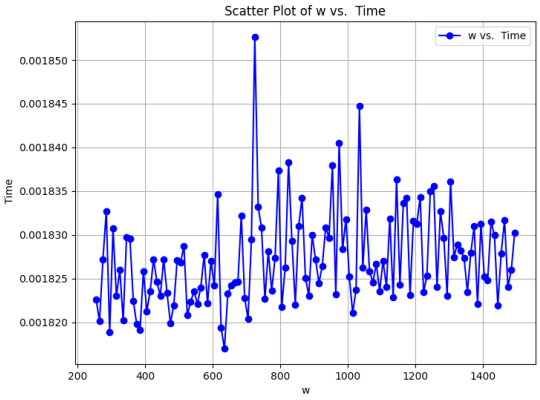




While a large number of k, which represents the number hash functions, does not lead to better accuracy, it leads to an increase in computational time. Therefore, to achieve optimal performance without compromising too much on accuracy, it is better to select a lower number of hash functions.

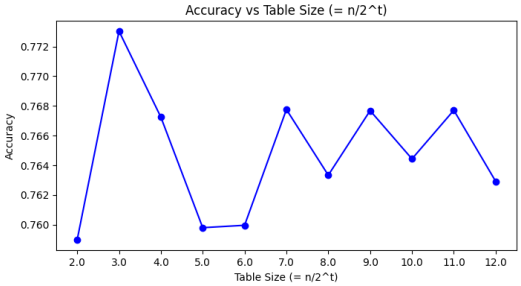
Window size (w parameter)
k = 5, L = 10, Table size = n/16

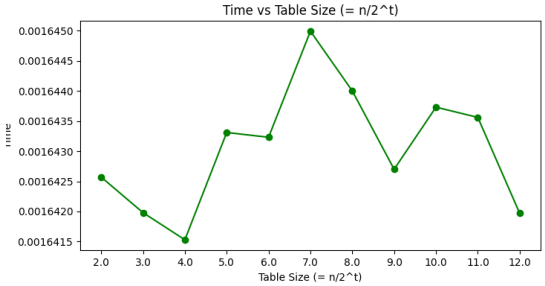




There doesn't seem to be a direct correlation where increasing w (window size) consistently improves or degrades accuracy or run time.

Table size
L = 10, k = 5, W = 10, N = 10

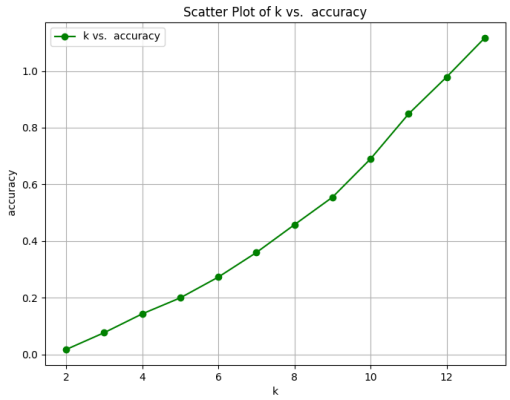


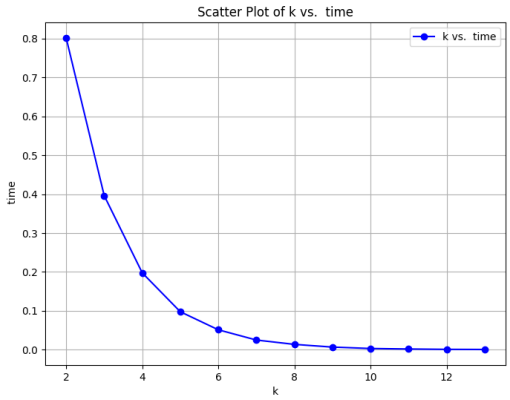


It's evident that table size does not affect accuracy greatly. We are looking for 10 nearest neighbors (N=10). The plots suggest that regardless of the table size the quality of these neighbors remains relatively unchanged.

Hypercube Implementation

Dimension in which the points are projected (k parameter)
W = 10, probes = 2

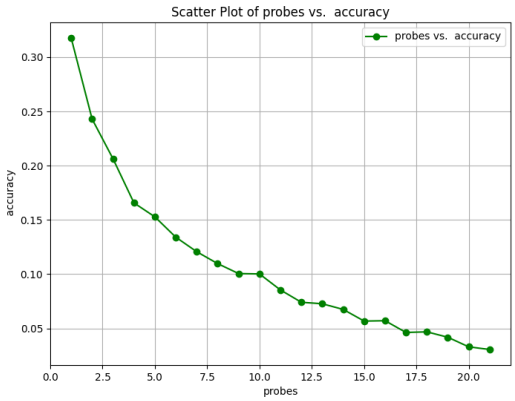


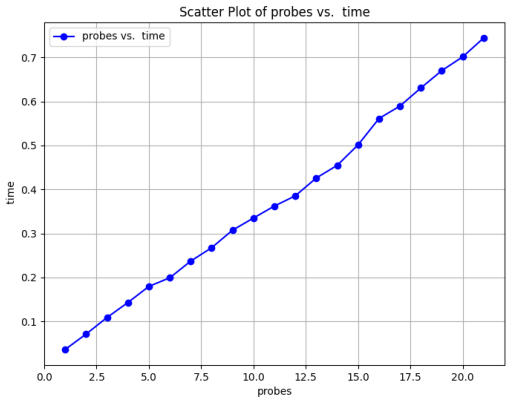


As k increases, indicating a higher hypercube dimension, the vectors are divided in more buckets. Consequently, as k increases, the bucket size gets smaller. With probes set at 2 (indicating checks on two buckets), a greater number of points are examined for smaller k values, and fewer for larger k values, so the difference in accuracy is justified.

Given that our dataset is uniformly distributed, we should expect a uniform distribution across the buckets. Therefore, each bucket approximately contains $\frac{n}{2^k}$ points. Increasing the value of k by 1 results in the bucket size being reduced in half. This offers an explanation for the observed exponential relationship between k and run time.

Probes parameter
W = 10, k = 5

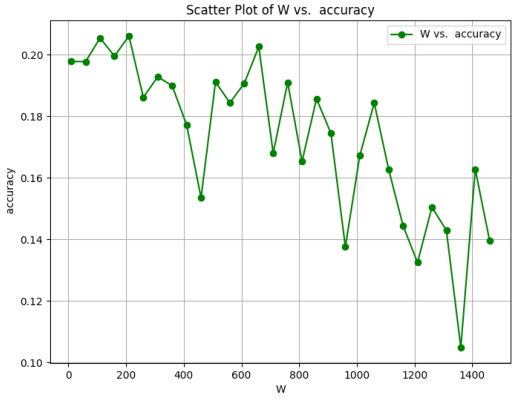


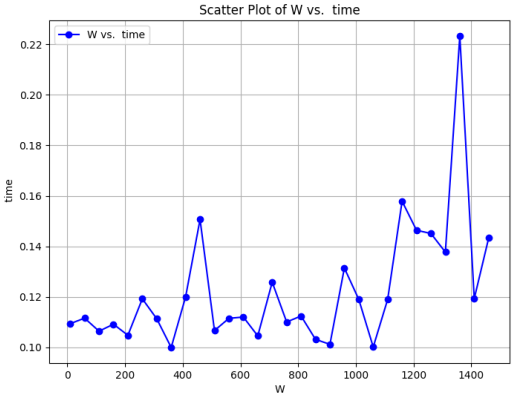


As expected, as the number of probes increases, it's harder to find meaningful neighbors which leads to increased accuracy. (Note: Smaller values mean better accuracy).

The relationship between the number of probes and time appears to be almost linear. This suggests that as we use more probes, the search becomes more time consuming.

Window Size (w parameter)
k = 5, probes = 2



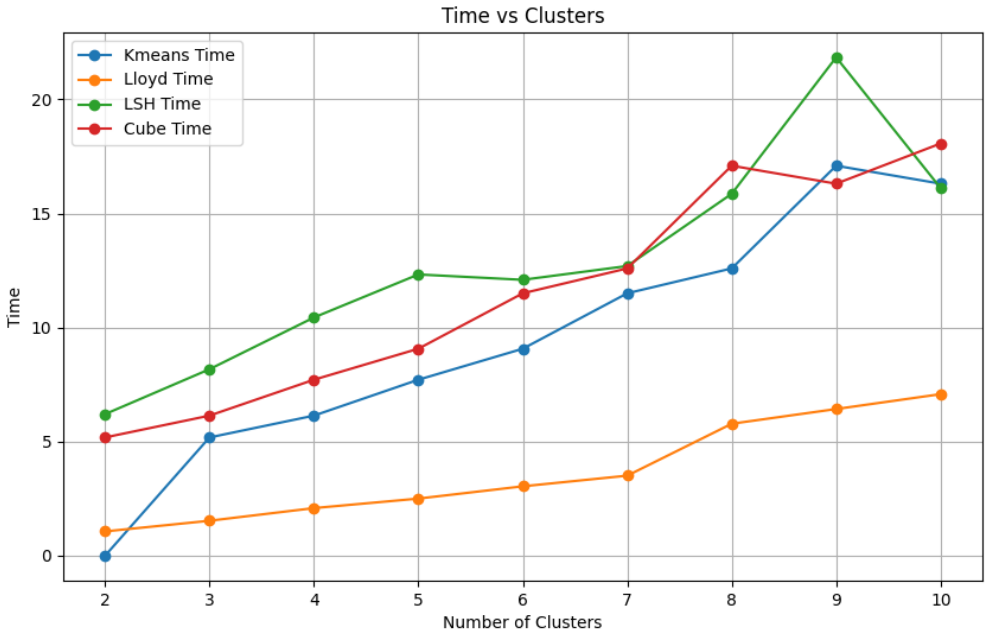


It is observed that there is no consistent relationship between w and both time and accuracy.

While there might be slight improvements in accuracy for increased w values, the difference isn't significant.

Clustering

10000 points were used.
Parameters for LSH: $L = 10$, $k = 4$, table size = 10000/32.
Parameters for Hypercube: $k = 4$, probes = 2, max candidates = no limit.



We expected the LSH and Hypercube methods to run faster than Lloyd's algorithm. However, Lloyd's proved to be the most efficient in all tests, converging quickly with only 2-3 point reassignments.

In comparison, LSH and Hypercube took much longer to converge, forcing us to set a 5-loop limit. Their slower run times might be due to the high computational cost and the larger overhead in calculating their values and depending on their parameters.

Silhoutte

Clusters = 10	Lloyd	LSH	Hypercube
Silhoutte Score	0.0608004	0.0298109	0.0178627
Average Time for 10.000 points	334.838 sec		

The results suggest that Lloyd's algorithm is the most effective at clustering for this dataset. Additionally, while the LSH and Hypercube methods often produce positive results, they are occasionally inconsistent, sometimes yielding negative outcomes.