MLBD Assignment-2

By:- Kwanit Gupta (B19EE046)

Description:-

Download the MNIST dataset from here. Pick its training and test sets. A 784-dimensional vector represents each sample in the dataset. There are ten classes (the ten digits, `0’ to `9’), and each sample is associated with one class. In this assignment, we will use the raw binary feature vectors, assuming they are “shingles”.

(1) Write a code to classify the test samples using the kNN algorithm and Jaccard similarity by varying the value of k in {1,2,3,4,5}. Report the classification accuracy and time required to classify all the test samples using one CPU core.

(2) Using any publicly available code for LSH, classify the test samples using the kNN algorithm. Vary the length of the signature vector in {40,60} and ‘s’ in {0.8, 0.9}. For each combination, run the experiment multiple times to calculate the average and standard deviation of the classification accuracy and time required to classify all the test samples in all the set-ups using one CPU core.

(3) Compare (1) and (2) in terms of classification accuracy, the time required, and peak RAM required.

Solution:-

KNN Classifier is a supervised machine learning algorithm that can be used for classification and regression tasks. It works by identifying the k nearest neighbors of a data point and using their labels to predict the label of the data point in question. The choice of distance metric is crucial in KNN, as it determines which data points are considered to be nearest neighbors. One common choice of distance metric for categorical data is the Jaccard distance.

The Jaccard distance measures the dissimilarity between two sets by calculating the ratio of the size of their intersection to the size of their union. In the context of KNN classification, the Jaccard distance between two data points can be calculated by treating each attribute as a binary value indicating whether it is present or not. The Jaccard distance between two data points is then the ratio of the number of attributes that differ between them to the total number of attributes.

As the first task, I flattened all the MNIST Images and binarized them for non-zero pixel values, thereby setting them directly to 1. Then I tried 3 different methodologies to perform KNN with Jaccard Distance:-

1. Using Custom Jaccard Distance Function
2. Sklearn’s KNN with metric as ‘jaccard’
3. Sklearn’s KNN with metric as Jaccard-Needham Dissimilarity

The key difference between Jaccard distance and Jaccard-Needham dissimilarity is in their calculation formula. Jaccard distance is calculated as 1 minus the Jaccard similarity coefficient, while Jaccard-Needham dissimilarity is calculated directly from the Jaccard similarity coefficient.

(A) Following were the time readings (in seconds) that KNN took with different Jaccard Methods:-

(i) For 10 images

Value of K	Custom Function	Metric='jaccard'	Metric='Jaccard-Needham'
1	627.65 seconds	0.379 seconds	14.841 seconds
2	574.25 seconds	0.387 seconds	13.177 seconds
3	567.76 seconds	0.381 seconds	13.689 seconds
4	619.80 seconds	0.386 seconds	18.516 seconds
5	646.04 seconds	0.397 seconds	16.303 seconds

(ii) For 60 images

Value of K	Custom Function	Metric='jaccard'	Metric='Jaccard-Needham'
1	3179.38 seconds	2.820 seconds	76.352 seconds
2	3038.49 seconds	2.034 seconds	70.529 seconds
3	3135.65 seconds	2.019 seconds	69.468 seconds
4	8688.90 seconds	2.038 seconds	70.957 seconds
5	3188.63 seconds	2.033 seconds	68.639 seconds

(iii) For 90 images

Value of K	Custom Function	Metric='jaccard'	Metric='Jaccard-Needham'
1	4202.45 seconds	4.124 seconds	103.350 seconds
2	4262.35 seconds	2.987 seconds	104.262 seconds
3	4253.20 seconds	2.994 seconds	105.842 seconds
4	4249.50 seconds	3.203 seconds	102.301 seconds
5	4312.88 seconds	3.910 seconds	103.944 seconds

(iv) For all 10000 images

Value of K	Custom Function	Metric='jaccard'	Metric='Jaccard-Needham'
1	Around 460 Hours (As Calculated with previous trends)	351.071 seconds	Around 7 Hours (As Calculated with previous trends)
2		349.315 seconds	
3		352.932 seconds	
4		353.139 seconds	
5		357.854 seconds	

Following were the accuracy readings (overall) that KNN took with different Jaccard Methods:-

(i) For 10 images

Value of K	Custom Function	Metric='jaccard'	Metric='Jaccard-Needham'
1	100%	100%	100%
2			
3			
4			
5			

(ii) For 60 images

Value of K	Custom Function	Metric='jaccard'	Metric='Jaccard-Needham'
1	98%	98%	98%
2	100%	100%	100%
3	98%	98%	98%
4	100%	100%	100%
5	100%	100%	100%

(iii) For 90 images

Value of K	Custom Function	Metric='jaccard'	Metric='Jaccard-Needham'

1	99%	99%	99%
2	99%	99%	99%
3	99%	99%	99%
4	100%	100%	100%
5	99%	99%	99%

(iv) For all 10000 images

Value of K	Custom Function	Metric='jaccard'	Metric='Jaccard-Needham'
1	Not Possible to Calculate	96%	Not Possible to Calculate
2			
3			
4			
5			

From this, we can say that the methods only differed the time of operations (because in-built have their internal finetuned configurations associated with workings, whereas the custom function isn't finetuned to optimized run), not the accuracy levels.

(B) Locally Sensitive Hashing (LSH) is a technique used in data mining and information retrieval to efficiently find similar items in large datasets. LSH is based on the concept of hashing, which is a function that maps input data of arbitrary size to a fixed-size output. Minhashing is a specific LSH technique used for finding similar sets, such as documents or images.

The basic idea behind LSH with Minhashing is to represent each item in the dataset as a set of features, and then map these features to hash codes using a hashing function. Similar items are then identified by comparing their hash codes. The key advantage of LSH with Minhashing is that it can quickly identify similar items without having to compare every item to every other item in the dataset.

The process of Minhashing involves representing each item in the dataset as a set of binary values, where each value corresponds to the presence or absence of a particular feature. For example, in the case of text documents, each feature might correspond to the presence or absence of a particular word in the document. A set of hash functions are then applied to each feature, resulting in a set of hash codes for each item.

The LSH step involves partitioning the hash codes into multiple "buckets" based on their similarity. Items with similar hash codes are more likely to be placed in the same bucket, and can therefore be quickly compared to one another to identify potential matches.

The key mathematical equation used in LSH with Minhashing is the Jaccard similarity coefficient, which is a measure of the similarity between two sets A and B:

$J(A,B) = |A \cap B| / |A \cup B|$

Minhashing relies on the fact that the Jaccard similarity between two sets can be approximated by the probability that the sets have the same minimum hash code value:

$J(A,B) \approx Pr[minhash(A) = minhash(B)]$

Following were the time readings (in seconds) that KNN took with different Jaccard Methods for length=40:-

(i) For 10 images

Value of K	Metric='jaccard' (s=0.8)	Metric='jaccard' (s=0.9)	Metric='Jaccard-Needham' (s=0.8)	Metric='Jaccard-Needham' (s=0.9)
1	0.058 seconds	0.037 seconds	7.97 seconds	9.82 seconds
2	0.044 seconds	0.046 seconds	10.22 seconds	7.99 seconds
3	0.072 seconds	0.075 seconds	9.84 seconds	9.69 seconds
4	0.078 seconds	0.045 seconds	9.77 seconds	8.73 seconds
5	0.043 seconds	0.046 seconds	8.16 seconds	8.56 seconds

(ii) For 60 images

Value of K	Metric='jaccard' (s=0.8)	Metric='jaccard' (s=0.9)	Metric='Jaccard-Needham' (s=0.8)	Metric='Jaccard-Needham' (s=0.9)
1	0.321 seconds	0.328 seconds	52.89 seconds	50.85 seconds
2	0.18 seconds	0.187 seconds	50.96 seconds	51.53 seconds
3	0.21 seconds	0.198 seconds	52.69 seconds	50.83 seconds
4	0.203 seconds	0.217 seconds	51.13 seconds	52.63 seconds
5	0.199 seconds	0.382 seconds	50.42 seconds	52.27 seconds

(iii) For 90 images

Value of	Metric='jaccard'	Metric='jaccard'	Metric='Jaccard-Needha	Metric='Jaccard-Needha

K	(s=0.8)	(s=0.9)	m’ (s=0.8)	m’ (s=0.9)
1	0.251 seconds	0.248 seconds	75.6 seconds	75.53 seconds
2	0.497 seconds	0.374 seconds	76.79 seconds	77.41 seconds
3	0.311 seconds	0.297 seconds	76.83 seconds	75.43 seconds
4	0.295 seconds	0.33 seconds	75 seconds	76.58 seconds
5	0.319 seconds	0.304 seconds	77.11 seconds	75.72 seconds

(iv) For all 10000 images

Value of K	Metric=’jaccard’ (s=0.8)	Metric=’jaccard’ (s=0.9)	Metric=’Jaccard-Needham’ (s=0.8)	Metric=’Jaccard-Needham’ (s=0.9)
1	37 seconds	32.74 seconds	Not Possible	Not Possible
2	31.63 seconds	31.97 seconds		
3	35.24 seconds	34.15 seconds		
4	37.08 seconds	36.7 seconds		
5	35.7 seconds	34.77 seconds		

Following were the time readings (in seconds) that KNN took with different Jaccard Methods for length=60:-

(i) For 10 images

Value of K	Metric=’jaccard’ (s=0.8)	Metric=’jaccard’ (s=0.9)	Metric=’Jaccard-Needham’ (s=0.8)	Metric=’Jaccard-Needham’ (s=0.9)
1	0.095 seconds	0.055 seconds	9.91 seconds	8.28 seconds
2	0.06 seconds	0.054 seconds	9.8 seconds	9.92 seconds
3	0.089 seconds	0.102 seconds	8.42 seconds	9.06 seconds
4	0.056 seconds	0.058 seconds	9.85 seconds	9.43 seconds
5	0.065 seconds	0.062 seconds	8.8 seconds	9.92 seconds

(ii) For 60 images

Value of	Metric=’jaccard’	Metric=’jaccard’	Metric=’Jaccard-Needha	Metric=’Jaccard-Needha

K	(s=0.8)	(s=0.9)	m’ (s=0.8)	m’ (s=0.9)
1	0.257 seconds	0.243 seconds	51.12 seconds	51.48 seconds
2	0.448 seconds	0.257 seconds	53.33 seconds	51.77 seconds
3	0.266 seconds	0.274 seconds	51.89 seconds	52.31 seconds
4	0.483 seconds	0.461 seconds	51.08 seconds	49.94 seconds
5	0.287 seconds	0.278 seconds	51.81 seconds	51.56 seconds

(iii) For 90 images

Value of K	Metric=’jaccard’ (s=0.8)	Metric=’jaccard’ (s=0.9)	Metric=’Jaccard-Needham’ (s=0.8)	Metric=’Jaccard-Needham’ (s=0.9)
1	0.355 seconds	0.357 seconds	75.26 seconds	76.35 seconds
2	0.375 seconds	0.38 seconds	77.44 seconds	76.4 seconds
3	0.395 seconds	0.413 seconds	77.34 seconds	77.39 seconds
4	0.406 seconds	0.433 seconds	75.26 seconds	78.17 seconds
5	0.425 seconds	0.704 seconds	78.44 seconds	76.7 seconds

(iv) For all 10000 images

Value of K	Metric=’jaccard’ (s=0.8)	Metric=’jaccard’ (s=0.9)	Metric=’Jaccard-Needham’ (s=0.8)	Metric=’Jaccard-Needham’ (s=0.9)
1	43.23 seconds	43.04 seconds	Not Possible	Not Possible
2	44.73 seconds	44.81 seconds		
3	47.48 seconds	46.82 seconds		
4	49.46 seconds	48.07 seconds		
5	50.58 seconds	49.44 seconds		

Following were the accuracy readings (overall) that KNN took with different Jaccard Methods for length=40:-

(i) For 10 images

Value of	Metric=’jaccard’	Metric=’jaccard’	Metric=’Jaccard-Needha	Metric=’Jaccard-Needha

K	(s=0.8)	(s=0.9)	m’ (s=0.8)	m’ (s=0.9)
1	64	55	100	73
2	55	64	100	91
3	91	82	100	91
4	64	82	100	100
5	64	91	91	100

(ii) For 60 images

Value of K	Metric=’jaccard’ (s=0.8)	Metric=’jaccard’ (s=0.9)	Metric=’Jaccard-Needham’ (s=0.8)	Metric=’Jaccard-Needham’ (s=0.9)
1	61	62	93	93
2	84	70	93	97
3	72	82	93	97
4	74	69	97	92
5	67	82	95	97

(iii) For 90 images

Value of K	Metric=’jaccard’ (s=0.8)	Metric=’jaccard’ (s=0.9)	Metric=’Jaccard-Needham’ (s=0.8)	Metric=’Jaccard-Needham’ (s=0.9)
1	58	57	95	92
2	57	75	92	93
3	68	87	93	95
4	75	64	92	91
5	75	80	98	92

(iv) For all 10000 images

Value of K	Metric=’jaccard’ (s=0.8)	Metric=’jaccard’ (s=0.9)	Metric=’Jaccard-Needham’ (s=0.8)	Metric=’Jaccard-Needham’ (s=0.9)

1	66	72	Not Possible	Not Possible
2	71	64		
3	67	43		
4	75	67		
5	77	73		

Following were the accuracy readings (overall) that KNN took with different Jaccard Methods for length=60:-

(i) For 10 images

Value of K	Metric='jaccard' (s=0.8)	Metric='jaccard' (s=0.9)	Metric='Jaccard-Needham' (s=0.8)	Metric='Jaccard-Needham' (s=0.9)
1	73	64	100	91
2	100	100		91
3	100	91		64
4	73	82		100
5	64	100		100

(ii) For 60 images

Value of K	Metric='jaccard' (s=0.8)	Metric='jaccard' (s=0.9)	Metric='Jaccard-Needham' (s=0.8)	Metric='Jaccard-Needham' (s=0.9)
1	74	80	97	100
2	85	74	98	97
3	85	85	95	97
4	89	87	95	97
5	67	79	97	98

(iii) For 90 images

Value of K	Metric='jaccard' (s=0.8)	Metric='jaccard' (s=0.9)	Metric='Jaccard-Needham' (s=0.8)	Metric='Jaccard-Needham' (s=0.9)

1	80	68	92	99
2	76	78	97	91
3	77	76	96	97
4	78	80	97	96
5	89	81	91	98

(iv) For all 10000 images

Value of K	Metric='jaccard' (s=0.8)	Metric='jaccard' (s=0.9)	Metric='Jaccard-Needham' (s=0.8)	Metric='Jaccard-Needham' (s=0.9)
1	82	74	Not Possible	Not Possible
2	76	73		
3	77	78		
4	83	79		
5	80	82		

For both (A) and (B), other details like Precision, Recall, F1-Score, and Support were also shown in the colab notebook mentioned below.

(c) Please refer to https://colab.research.google.com/drive/1_cinCrgK8DM1oylwFmUZuAn3CSWSqv0v?usp=sharing, since in this notebook I printed all the information corresponding to each and every parametric combination, with the following format:-

```
Peak memory usage during fit function: 9456.63 MiB
Peak memory usage during predict function: 9498.13 MiB
L1 cache size: 32.00 KiB
L2 cache size: 256.00 KiB
Memory usage of KNN object and its attributes: 0.00 MiB
Resident memory usage of Python process: 9456.63 MiB
Virtual memory usage of Python process: 10671.32 MiB
CPU usage of Python process: 0.00%
Number of threads of Python process: 17
Number of open file descriptors of Python process: 61
```

The name of the corresponding sections would be "Memory-related information for imgs=Num_Images"

The Configuration of the Google Colab’s CPU used hereby is as follows:-

- Intel Xeon CPU
- 1 or 2 vCPU cores (depending on the instance type)
- Clock speed ranges from 2.2 to 2.7 GHz
- Support for Hyper-Threading (simultaneous multithreading) technology
- Cache memory: L1 32 KB, L2 256 KB, and L3 35 MB

Reference:-

- **https://yann.lecun.com/exdb/mnist/**