MLBD Assignment-3

By:- Kwanit Gupta (B19EE046)

Description:-

Download the MNIST dataset from here. Pick its training and test sets. Each sample in the dataset is represented by a 784-dimensional vector. There are ten classes (the ten digits, `0' to `9'), and each sample is associated with one class.

(1) Using the raw binary features, implement the streaming Naive Bayes algorithm and classify the test data.

(2) Project the binary features into a lower dimensional space using a dimensionality reduction technique of your choice (such as PCA, LDA, t-SNE, etc.), by varying the dimensionality in the range {50,100,200}, and classify the test data using the same algorithm.

(3) Compare (1) and (2) in terms of classification accuracy and time required during the training and testing phase.

(4) Compare (1) and (2) with the classification accuracy you had obtained in assignment-2.

Solution:-

Streaming Naive Bayes is an online machine learning algorithm that is used for classification tasks. It is a variation of the Naive Bayes algorithm, which assumes that the features are conditionally independent given the class. The Streaming Naive Bayes algorithm can handle a large stream of data by processing data in mini-batches instead of individual data points. It updates the model incrementally as new data is received, without having to reprocess all previous data.

At a high level, the algorithm works by first calculating the prior probabilities of each class by counting the number of occurrences of each class in the training data. Then, for each feature in the dataset, the algorithm calculates the conditional probabilities for each class by counting the number of occurrences of that feature in each class. These probabilities are used to calculate the likelihood of a data point belonging to each class.

During the training phase, as new mini-batches of data are received, the algorithm updates the counts for each feature and class, which are used to re-calculate the conditional probabilities for each class. The model is then updated incrementally based on the new counts. When a new data point needs to be classified, the algorithm calculates the likelihood of the data point belonging to each class using the conditional probabilities, and then selects the class with the highest likelihood as the predicted class. Overall, Streaming Naive Bayes is a fast and efficient algorithm that is well-suited for large, streaming datasets, and can handle new data as it becomes available without the need to reprocess the entire dataset.

===

(3.A) Following were the time readings (in seconds) that Streaming Naive Bayes took with variations in the following parameters (without using dimensionality reduction techniques):-

1. Stream Window:- Number of Samples accounting into the single mini-batch
2. Alpha:- Used in Laplace Smoothing, to compensate for zero frequencies and probabilities.
3. Epsilon:- For avoiding zero denominators while calculating likelihoods and log-likelihoods.

Stream Window	Alpha	Epsilon	Time (training)	Time (testing)
10	0.01	10**-9	2.955 s	0.764 s
10	0.1	10**-9	1.359 s	0.043 s
10	1	10**-9	1.457 s	0.052 s
10	1	1	1.515 s	0.04 s
10	1	10**9	1.903 s	0.063 s
10	10	10**-9	2.687 s	0.043 s
10	100	10**-9	1.949 s	0.066 s
100	1	10**-9	0.26 s	0.042 s
100	1	1	0.229 s	0.041 s
100	1	10**9	0.378 s	0.063 s

(3.B) Following were the accuracy (in percentage) that Streaming Naive Bayes took with variations in the following parameters (without using dimensionality reduction techniques):-

Stream Window	Alpha	Epsilon	Accuracy (testing)
10	0.01	10**-9	62.36%
10	0.1	10**-9	62.35%
10	1	10**-9	62.43%
10	1	1	54.96%
10	1	10**9	11.35%
10	10	10**-9	63.43%
10	100	10**-9	64.25%
100	1	10**-9	62.43%

100	1	1	54.96%
100	1	10**9	11.35%

While keeping up the same configurations (stream window = 100, epsilon = 10**-9, alpha = 1), I tried to have variations within the n_components for dimensionality reduction techniques, as following:-

(3.C) Following were the time readings (in seconds) that Streaming Naive Bayes took with variations in the following parameters (using dimensionality reduction techniques and excluding time for data preprocessing):-

Reduction Technique	n_components	Time (training)	Time (training)	Time (testing)	Time (testing)
		No Stack	Stack	No Stack	Stack
PCA	50	0.556 s	0.281 s	0.021 s	0.005 s
PCA	100	0.289 s	0.549 s	0.006 s	0.022 s
PCA	200	0.304 s	0.136 s	0.01 s	0.013 s
LDA	2	0.16 s	0.163 s	0.001 s	0.002 s
LDA	5	0.159 s	0.157 s	0.003 s	0.002 s
LDA	8	0.159 s	0.157 s	0.004 s	0.003 s
TSNE	1	0.115 s	0.195 s	0.001 s	0.002 s
TSNE	2	0.116 s	0.131 s	0.002 s	0.003 s
TSNE	3	0.115 s	0.151 s	0.004 s	0.003 s

(3.D) Following were the accuracy (in percentage) that Streaming Naive Bayes took with variations in the following parameters (using dimensionality reduction techniques):-

Dimensionality Reduction Technique	n_components	Accuracy (testing)	Accuracy (testing)
		No Stack	Stack
PCA	50	11.99%	11.92%
PCA	100	12.25%	12.37%
PCA	200	12.22%	12.41%
LDA	2	8.61%	8.41%

LDA	5	3.74%	4.68%
LDA	8	8.68%	10.23%
TSNE	1	17.34%	18.82%
TSNE	2	10.09%	3.21%
TSNE	3	7.71%	14.92%

There could be several reasons why a normal Streaming Naive Bayes performed better in accuracy than that of Streaming Naive Bayes with dimensionality reduction techniques on binarized MNIST. Some possible reasons are:

1. Binarization of the dataset: Binarized MNIST dataset contains only black and white pixels, and each pixel is represented by a binary value. In such datasets, the feature space is already reduced, and using additional dimensionality reduction techniques may not be helpful.

2. Curse of dimensionality: The high dimensionality of the original feature space in MNIST can make it difficult to find a good representation of the data that captures its essential structure. However, applying dimensionality reduction techniques can also introduce noise or bias, which may negatively affect the performance of the model.

3. Loss of information: Dimensionality reduction techniques can lead to loss of information, which can negatively impact the performance of the model. In some cases, this loss of information may be too great, and the reduced feature space may not capture the essential structure of the data.

4. Complexity of the model: Streaming Naive Bayes with dimensionality reduction techniques can be more complex than a normal Streaming Naive Bayes model. This increased complexity may require more data to train the model effectively, and in the case of MNIST, there may not be enough data to train the model effectively.

5. Overfitting: Dimensionality reduction techniques may also lead to overfitting, especially when the reduced feature space is too small or when the data is noisy. In such cases, the model may not generalize well to new data, leading to lower accuracy.

==

(4) If compared with the time taken and accuracy by KNN Model (with and without LSH), the Streaming Naive Bayes performed as following:-

a. Assignment-2 Comparisons

Methods	Time (range)	Accuracy (range)

KNN (Jaccard Only)	350-360 seconds	96%
KNN (Jaccard + LSH)	37-50 seconds	43% - 83%

b. Assignment-3 Comparisons (excluding the data preprocessing time)

Methods	Time (range)	Accuracy (range)
Streaming Naive Bayes (Vanilla)	0.04 - 0.76 seconds	11.35% - 64.25%
Streaming Naive Bayes (PCA)	0.005 - 0.022 seconds	11.92% - 12.41%
Streaming Naive Bayes (LDA)	0.001 - 0.004 seconds	3.74% - 10.23%
Streaming Naive Bayes (T-SNE)	0.001 - 0.004 seconds	3.21% - 18.82%

The reasons why KNN performed better than Streaming Naive Bayes on binarized MNIST are:

1. KNN is a non-parametric algorithm and thus does not assume any distribution of the data. This makes it more suitable for binarized MNIST dataset, which is highly irregular and nonlinear.

2. Binarization of MNIST data reduces the number of features, which makes it easier for KNN to handle since it relies on the distance between data points.

3. KNN performs well when the dataset has a large number of samples, as in the case of the binarized MNIST dataset.

4. KNN is also less sensitive to outliers compared to Streaming Naive Bayes, which can be an advantage in the case of MNIST dataset where the data may have some noise or outliers.

5. Overall, KNN's non-parametric nature and its ability to handle datasets with fewer features and a large number of samples make it a better choice for binarized MNIST classification.

==

While evaluating the structuring of the features space, I performed the following dimensionality reduction techniques:-

1. PCA (Principal Component Analysis):-

Given a dataset X with n data points, where each data point is a d-dimensional vector, PCA seeks to find a new representation of the data in a lower-dimensional space, while still preserving as much of the original variance as possible.

First, PCA calculates the covariance matrix of the data, which is given by:

$S = (1/n) * X^T * X$

where X^T is the transpose of the data matrix X, and the division by n ensures that the covariance matrix is unbiased. Next, PCA finds the eigenvectors and eigenvalues of the covariance matrix S. An eigenvector v is a nonzero vector that satisfies the equation:

$S * v = \lambda * v$

where λ is the corresponding eigenvalue. The eigenvectors represent the directions in the original feature space that have the highest variance, while the eigenvalues represent the amount of variance in each of these directions. PCA then selects the top k eigenvectors with the highest eigenvalues, where k is the desired dimensionality of the new data representation. These k eigenvectors are the principal components of the data.

To project the data onto the new lower-dimensional space, PCA forms a new data matrix Y, where each data point in Y is a k-dimensional vector given by:

$Y = X * V$

where V is the matrix formed by the top k eigenvectors of S, and * denotes matrix multiplication. Each row of Y represents a new lower-dimensional representation of a data point in X. To reconstruct the original data from the lower-dimensional representation in Y, PCA can use the inverse transformation:

$X_{hat} = Y * V^T$

where V^T is the transpose of V. The reconstructed data X_hat will have a higher dimensionality than the original data X, but it will have a lower rank, meaning that some of the original variance in X will have been lost.

Overall, PCA is a powerful technique for reducing the dimensionality of a dataset while still preserving as much of the original variance as possible. It is widely used in machine learning and data analysis to simplify and understand complex datasets, identify important features or patterns, and prepare data for further analysis or modeling tasks.

2. LDA (Linear Discriminant Analysis):-

At a high level, LDA is a statistical method that is used to find a linear combination of features that best separates two or more classes. The goal of LDA is to find a projection of the data onto a lower-dimensional subspace that maximizes the separation between the classes. This projection is then used to classify new data points based on their position in this subspace.

To understand LDA more deeply, it's useful to start with some definitions. Let's assume we have a dataset with n samples, each with p features, and k classes. We can represent the data as an n x p matrix X, where each row represents a sample and each column represents a feature. We can also represent the class labels as a vector y of length n, where y_i is the class label of the i-th sample.

The goal of LDA is to find a projection of the data onto a lower-dimensional subspace that maximizes the separation between the classes. We can do this by finding a set of linear coefficients w that maximizes the ratio of the between-class variance to the within-class variance.

To define these variances, let's first define some notation. Let mu_k be the mean vector of the k-th class, and let N_k be the number of samples in the k-th class. Let mu be the overall mean vector of the data, and let N be the total number of samples. Finally, let S_w be the within-class scatter matrix, and let S_b be the between-class scatter matrix. The within-class scatter matrix S_w measures the variance within each class. It is defined as:

S_w = sum_{k=1}^K sum_{i:y_i=k} (x_i - mu_k) (x_i - mu_k)^T

In words, this matrix is the sum of the outer products of the differences between each sample and its class mean. It measures how much the samples in each class vary around their mean. The between-class scatter matrix S_b measures the variance between the classes. It is defined as:

S_b = sum_{k=1}^K N_k (mu_k - mu) (mu_k - mu)^T

In words, this matrix is the sum of the outer products of the differences between each class mean and the overall mean, weighted by the number of samples in each class. It measures how much the class means vary from each other.

The goal of LDA is to find a set of linear coefficients w that maximizes the ratio of the between-class variance to the within-class variance. This can be written as:

w = argmax_w (w^T S_b w) / (w^T S_w w)

To solve for w, we can use Lagrange multipliers to enforce the constraint that w is a unit vector (i.e., ||w||^2 = 1). This leads to the generalized eigenvalue problem:

S_b w = lambda S_w w

Solving this eigenvalue problem gives us the coefficients w that maximize the separation between the classes. We can use these coefficients to project the data onto a lower-dimensional subspace, where the classes are more separable. In practice, we often choose the top k eigenvectors of the eigenvalue problem, where k is the desired dimensionality of the subspace. We can then project the data onto this subspace by multiplying it by the transpose of the eigenvectors:

X_lda = X * W

This new data matrix X_lda can then be used for classification tasks, such as predicting the class label of new samples based on their position in the subspace.

3. TSNE (T-distributed Stochastic Neighbor Embedding):-

t-SNE is a powerful tool for visualizing high-dimensional data in a lower-dimensional space. The goal of t-SNE is to take a dataset with a large number of features or dimensions, and represent it in a two or three-dimensional space that is easy to visualize. The key idea behind t-SNE is to preserve the local structure of the data, meaning that data points that are close together in the high-dimensional space should also be close together in the low-dimensional space, while the global structure of the data is less important.

The t-SNE algorithm works by first constructing a probability distribution over pairs of high-dimensional data points that is based on their similarity. This is done using a Gaussian kernel to measure the similarity between each pair of data points. The Gaussian kernel assigns higher probability to pairs of data points that are close together in the high-dimensional space, and lower probability to pairs that are far apart.

Next, t-SNE constructs a similar probability distribution over pairs of points in the low-dimensional space. The algorithm starts by placing the data points randomly in the low-dimensional space, and then iteratively adjusts their positions to minimize the difference between the two probability distributions. The goal is to find a configuration of points in the low-dimensional space that is as similar as possible to the configuration of points in the high-dimensional space, while preserving the local structure of the data.

To measure the difference between the two probability distributions, t-SNE uses the Kullback-Leibler (KL) divergence. The KL divergence measures the difference between two probability distributions, and is used in t-SNE to measure the difference between the distribution over pairs of high-dimensional data points and the distribution over pairs of low-dimensional points. The algorithm uses gradient descent to iteratively adjust the positions of the data points in the low-dimensional space in order to minimize the KL divergence.

One important feature of t-SNE is that it is a non-linear algorithm, meaning that it can capture complex relationships between the data points that may not be apparent in a linear representation. Another key feature is that it is robust to outliers and can handle data with a large number of features or dimensions.

The mathematics of t-SNE can be quite complex, but the basic idea is to minimize the difference between two probability distributions using gradient descent. The algorithm uses a cost function that measures the difference between the two probability distributions, and then updates the positions of the data points in the low-dimensional space using the gradient of the cost function. The cost function involves a sum over pairs of high-dimensional data points and pairs of low-dimensional points, and includes a term that penalizes large distances between the data points in the low-dimensional space.

In summary, t-SNE is a powerful algorithm for visualizing high-dimensional data in a lower-dimensional space, and is widely used in fields such as bioinformatics, image analysis, and natural language processing. Its ability to preserve the local structure of the data and its nonlinear nature make it a valuable tool for exploratory data analysis and pattern recognition.

==

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/**