MACHINE LEARNING TASK 1

- NAME: BRINDA ARUNKUMAR
- SUBJECT: MACHINE LEARNING
- MATRICULATION NUMBER : 11026769
- PROFESSOR: milan.gnjatovic@kpu.edu.r

AIM:

Task 1: **Handwritten Digit Classification**

A program for classification of handwritten digits based on the Gaussian naive Bayes approach.

ABSTRACT:

Handwritten digit classification is a well-known problem in the field of machine learning, and the MNIST dataset is a popular dataset used to train and test models for this task. The MNIST dataset consists of 70,000 images of handwritten digits, with 60,000 images used for training and 10,000 images used for testing. The goal of this task is to classify each image into one of ten classes, corresponding to the digits from 0 to 9. In recent years, various machine learning algorithms and deep learning architectures have been developed to tackle this problem, achieving high levels of accuracy on the MNIST dataset. These models have important applications in various fields, such as character recognition, automated document processing, and computer vision.

INTRODUCTION:

Handwritten digit recognition is the process to provide the ability to machines to recognize human handwritten digits. It is not an easy task for the machine because handwritten digits are not perfect, vary from person-to-person, and can be made with many different flavors.

The accuracy of handwritten text recognition depends on trained CNN, which is considered a successful method in some handwriting and computer recognition applications.

Handwritten digit classification is a common problem in machine learning and computer vision. The goal is to correctly classify an image of a handwritten digit into one of the ten possible digits (0-9). One popular algorithm for this task is Gaussian Naive Bayes (GNB).

GNB is a simple probabilistic classifier that applies Bayes' theorem. Given an input feature vector x and a set of class labels y, the probability of x belonging to class y is given by:

$P(y|x) = P(x|y) * P(y) / P(x)$

where $P(x|y)$ is the likelihood of observing x given y, $P(y)$ is the prior probability of y, and $P(x)$ is the evidence.

In the case of handwritten digit classification, the feature vector x represents the pixel values of the input image, and y represents the class label (digit). GNB models the likelihood P(x|y) as a Gaussian distribution with mean and variance estimated from the training data. To find the confusion matrix for GNB classification, we first train the model on a training set of handwritten digit images and their corresponding labels. Then, we use the trained model to predict the labels of a separate test set of images. The confusion matrix is a table that shows the number of true positives, false positives, true negatives, and false negatives for each class label.

For example, if we have 100 test images of digit "0" and the GNB model predicts 95 of them correctly and 5 of them as "1", then the confusion matrix would show 95 true positives and 5 false positives for "0", and 0 true positives and 0 false positives for "1". We can compute the precision, recall, and F1-score for each class label using the values in the confusion matrix.

Flow of the code:

The code performs handwritten digit classification using the Gaussian Naive Bayes theorem and computes the confusion matrix, precision, recall, and F1-score to evaluate the performance of the classifier. Here is a summary of the code:

1.Import the required libraries: pandas,GaussianNB,confusion_matrix,precision_recall_fscore_support.

2.Mount the Google Drive to access the CSV files.

3.Load the training and testing data from the CSV files.

4.Split the data into features (X) and labels (y).

5.Generate a Gaussian Naive Bayes model.

6.Train the model using the training data.

7.Use the trained model to predict the labels of the testing data.

8.Compute the confusion matrix using the true labels (y_test) and predicted labels (y_pred) of the testing data.

9.Compute precision, recall, and F1-score using precision_recall_fscore_support () function of scikit-learn library.

10.Print the Confusion Matrix, Precision, Recall, and F1-score.

PARAMETERS AND VARIABLES USED IN THE CODE

SL NO	PARAMETER	SIGNIFICANCE

1	GaussianNB	GaussianNB is a function from the sklearn.naive_bayes module, which implements the Gaussian Naive Bayes algorithm. This algorithm is used for classification tasks, where the goal is to predict the category of a given input.
2	confusion_matrix	confusion_matrix is a function from the sklearn. metrics module, which computes the confusion matrix for a given set of predictions and true labels. The confusion matrix is a table that shows the number of true positives, false positives, true negatives, and false negatives.
3	precision_recall_fscore_support	precision_recall_fscore_support is a function from the sklearn. metrics module, which computes the precision, recall, F1-score, and support for a given set of predictions and true labels. These metrics are commonly used to evaluate the performance of a classification model.
4	train_data and test_data	train_data and test_data are pandas Data Frames that store the training and testing data, respectively. The read_csv function is used to read the data from CSV files.
5	X_train and X_test	X_train and X_test are numpy arrays that store the features of the training and testing data, respectively. In this case, the features are the pixel values of the images.
6	y_train and y_test	y_train and y_test are numpy arrays that store the labels of the training and testing data, respectively. In this case, the labels are the digits that the images represent.
7	gnb	gnb is an instance of the Gaussian Naive Bayes model.
8	gnb.fit(X_train, y_train)	gnb.fit(X_train, y_train) is used to train the Gaussian Naive Bayes model using the training data.
9	y_pred	y_pred is a numpy array that stores the predicted labels for the testing data, based on the trained model.

OUTPUT:

Confusion Matrix	Predicted Class									
	0	1	2	3	4	5	6	7	8	9

Actual Class	0	870	0	3	5	2	5	31	1	35	28
	1	0	1079	2	1	0	0	10	0	38	5
	2	79	25	266	91	5	2	269	4	271	20
	3	32	39	6	353	2	3	51	8	409	107
	4	19	2	5	4	168	7	63	7	210	497
	5	71	25	1	20	3	44	40	2	586	100
	6	12	12	3	1	1	7	895	0	26	1
	7	0	15	2	10	5	1	5	280	39	670
	8	13	72	3	7	3	11	12	4	648	201
	9	5	7	3	6	1	0	1	13	18	955

ACCURACY: 0.5558555855585559

Precision Matrix									
0.790191	0.845611	0.904762	0.708835	0.884211	0.55	0.649964	0.877743	0.284211	0.369582

Recall Matrix									
0.887755	0.950661	0.257752	0.349505	0.171079	0.049327	0.934238	0.272639	0.665298	0.946482

F1 Score									
0.836136	0.895064	0.401207	0.46817	0.286689	0.090535	0.766595	0.416048	0.398279	0.531589

Support									
980	1135	1032	1010	982	892	958	1027	974	1009

1. How do you handle features with a constant value across all images in the dataset?

- A dataset's features that have a fixed value for every image contribute no valuable information to the machine learning model and may potentially hinder it from performing well. The removal of such characteristics from the dataset before to model training is one method for handling them. This may be accomplished manually by analysing the data and deleting the constant features or using alternative approaches like variance thresholding. The VarianceThreshold class in Scikit-Learn may be used to exclude features that have variance below a certain threshold.

2. How do you handle features with a constant value across all images belonging to a given class

- We can use feature selection techniques like variance thresholding or correlation-based feature selection to manage characteristics with a constant value across all photos in a particular class. When using variance thresholding, features that have variances below a predetermined threshold are eliminated because they are thought to have weak or no discriminatory power. As these features are probably to have strong predictive power, correlation-based feature selection involves choosing features that are highly correlated with the class labels. The VarianceThreshold class from the sklearn.feature_selection package may be used to create variance thresholding in Python. Use the SelectKBest or Select Percentile classes from the same module along with an appropriate scoring function, such as chi-squared or mutual information, to perform correlation-based feature selection.

3. **Do you, and if so, how do you the prevent arithmetic underflow or overflow?**

- Arithmetic underflow occurs when a number is too small to be represented by the data type being used, while arithmetic overflow occurs when a number is too large to be represented by the data type. Here are some ways to prevent arithmetic underflow or overflow:
- Use appropriate data types: Choosing an appropriate data type for your variables can help prevent arithmetic underflow or overflow. For example, using a floating-point data type instead of an integer data type can prevent underflow. We can Check for overflow and underflow before performing arithmetic operations, we can check whether the result will cause an overflow or underflow. We can do this by comparing the values of the operands and the expected result with the maximum and minimum values of the data type being used. In some programming languages, you can use exception handling to catch arithmetic overflow or underflow errors. When an error occurs, the program can take appropriate action to prevent data loss or unexpected behavior. You can prevent arithmetic underflow and overflow vulnerabilities by adding modifiers to functions performing arithmetic operations that check for integer overflow/underflow.

4: **How do you explain the low classification accuracy obtained by applying the Gaussian naive Bayes approach in this particular context?**

When using the Gaussian Naive Bayes approach in a specific situation, a number of factors may result in low classification accuracy. The idea that features are independent of one another is one such factor. This might not always be the case, for example, when classifying images in which the values of nearby pixels might be correlated, producing a model with lower accuracy and less optimal performance.
Class imbalance, where some classes have a disproportionately lower number of instances, is another factor. Because of the algorithm's assumption that all classes have equal prior probabilities, inaccurate modeling and lower accuracy may result from datasets that are unbalanced.
If the actual feature distributions are not Gaussian, the assumption of Gaussian feature distributions may also result in lower accuracy. For instance, the Gaussian Naive Bayes model might not be able to capture highly skewed or multimodal feature distributions, resulting in incorrect classification.
Last but not least, the curse of dimensionality may also result in decreased accuracy when there are many more features than instances. Because of this sparsity, it may be difficult to estimate model parameters accurately, which will reduce accuracy.

<u>**REFERENCES:**</u>

1. https://openai.com/blog/chatgpt
2. https://pjreddie.com/projects/mnist-in-csv/
3. https://lazyprogrammer.me/bayes-classifier-and-naive-bayes-tutorial-using/
4. https://colab.research.google.com/#scrollTo=0YepiaKuyRaY

Thank you!

Have a good day 😊