EXPERIMENT-1

AIM: Classification using MLP on MNIST dataset for digit classification

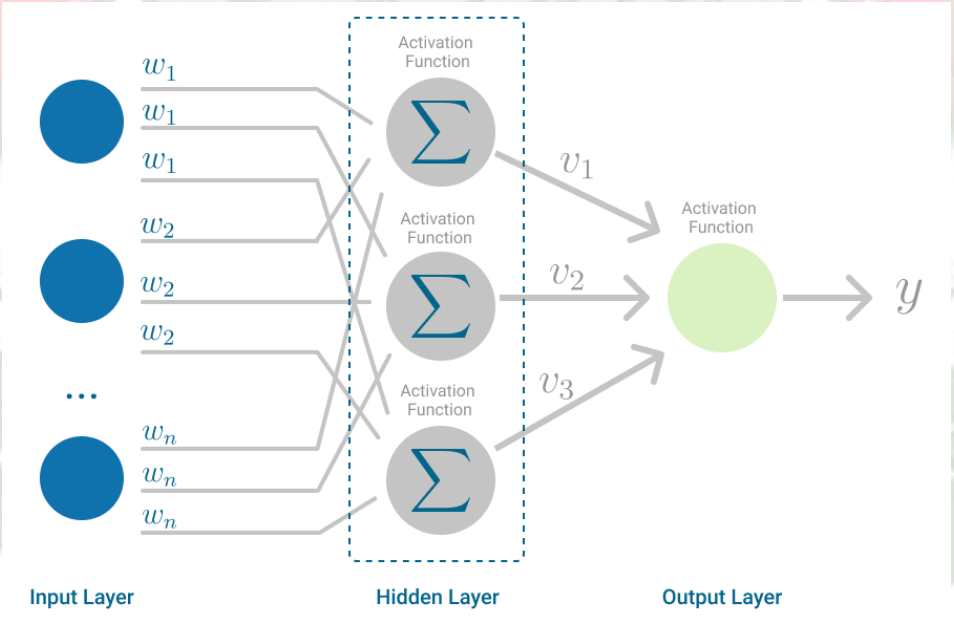
DESCRIPTION:

Introduction:

Multi-Layer Perceptron Learning:

Multi-layer perception is also known as MLP. It is fully connected dense layers, which transform any input dimension to the desired dimension. A multi-layer perception is a neural network that has multiple layers. To create a neural network we combine neurons together sothat the outputs of some neurons are inputs of other neurons.

A multi-layer perceptron has one input layer and for each input, there is one neuron(or node), it has one output layer with a single node for each output and it can have any numberof hidden layers and each hidden layer can have any number of nodes. A schematic diagram of a Multi-Layer Perceptron (MLP) is depicted below.



(reference: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141)

Classification:

Classification is a process of categorizing data or objects into predefined classes or categories based on their features or attributes. Machine Learning classification is a type of

supervised learning technique where an algorithm is trained on a labeled dataset to predict the class or category of new, unseen data.

The main objective of classification machine learning is to build a model that can accurately assign a label or category to a new observation based on its features.

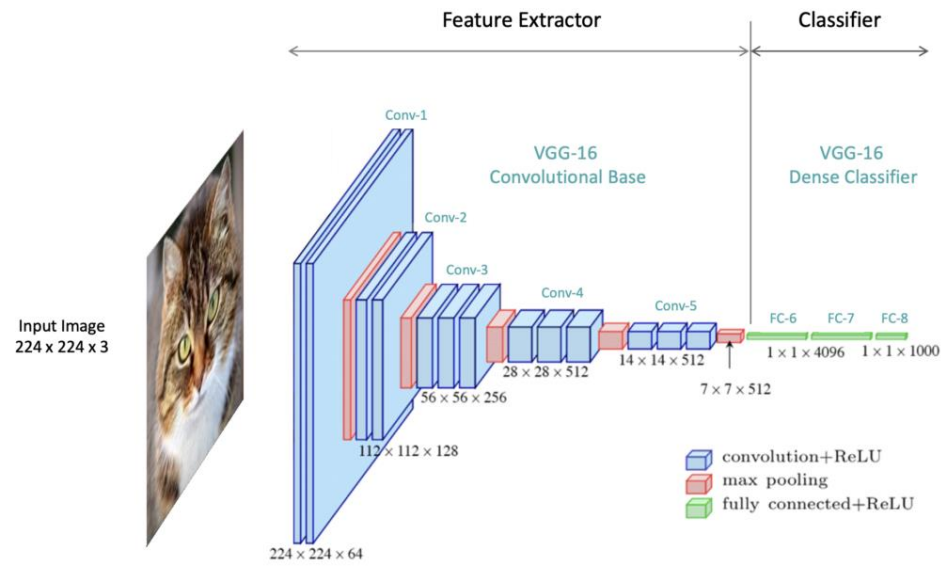
For example, a classification model might be trained on a dataset of images labeled as either dogs or cats and then used to predict the class of new, unseen images of dogs or cats based on their features such as color, texture, and shape.

About the MNIST dataset:

The MNIST (Modified National Institute of Standards and Technology) database is a large database of handwritten numbers or digits that are used for training various image processing systems. The dataset also widely used for training and testing in the field of machine learning. The set of images in the MNIST database are a combination of two of NIST's databases: Special Database 1 and Special Database 3.

The MNIST dataset has 60,000 training images and 10,000 testing images.

The MNIST dataset can be online, and it is essentially a database of various handwritten digits. The MNIST dataset has a large amount of data and is commonly used to demonstrate the real power of deep neural networks. Our brain and eyes work together to recognize any numbered image. Our mind is a potent tool, and it's capable of categorizing any image quickly. There are so many shapes of a number, and our mind can easily recognize these shapes and determine what number is it, but the same task is not simple for a computer to complete. There is only one way to do this, which is the use of deep neural network which allows us to train a computer to classify the handwritten digits effectively.



(reference: https://learnopencv.com/understanding-convolutional-neural-networks-cnn/)

So, we have only dealt with data which contains simple data points on a Cartesian coordinate system. From starting till now, we have distributed with binary class datasets. And when we use multiclass datasets, we will use the Softmax activation function is quite useful for classifying binary datasets. And it was quite effective in arranging values between 0 and 1. The sigmoid function is not effective for multicausal datasets, and for this purpose, we use the softmax activation function, which is capable of dealing with it.

Implementation of Classification with Multilayer Perceptron using Scikit-learn with MNIST Dataset

NumPy: NumPy stands for Numerical Python. It is one of the basic Python Library that is used for creating arrays, filling null values, statistical calculations and computations. Pandas is built on the top of the NumPy library.

Matplotlib: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python

CODE:

```
# Importing modules
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# Loading dataset
```

```
X, y = load_digits(return_X_y=True)
y = y.astype(int)
X /= 255.0  # Normalizing gray image to floats

# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print(len(X_train), len(X_test), len(y_train), len(y_test))  # Output:
(1437, 360, 1437, 360)

# Training the model
m = MLPClassifier(hidden_layer_sizes=(64, 32, 16), max_iter=100)
m.fit(X_train, y_train)

# Making inferences
y_pred = m.predict(X_test)
print(len(y_pred))  # Output: 360

# Evaluating the model
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)

accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

Ouput:

98.5

RESULT:

The result of this implementation is a trained Perceptron model that successfully finds a decision boundary, or straight line, that separates the two classes in the Iris dataset: Iris-setosaand Iris-versicolor. By iteratively adjusting the weights based on the training examples, the Perceptron algorithm converges, meaning it finds a set of weights that correctly classifies all the data points in this linearly separable dataset. The scatter plot visualization confirms that the two classes are distinctly separated by this boundary, demonstrating the effectiveness ofthe Perceptron in handling linearly separable data. This outcome showcases the algorithm's ability to learn and create a model that accurately distinguishes between the two flower species based on their features.

EXPERIMENT-2

AIM:

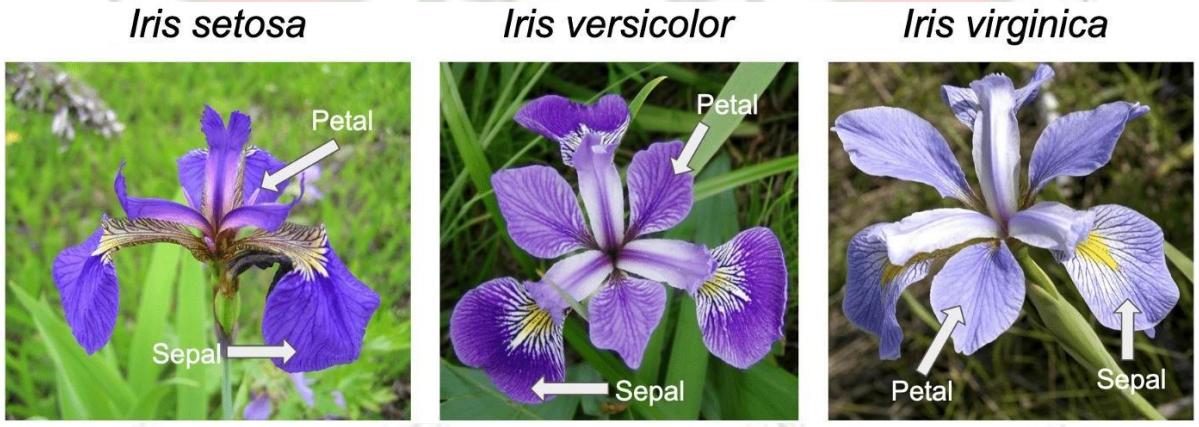
To implement and analyze the Perceptron learning algorithm for binary classification using a linearly separable subset of the Iris dataset, focusing on its ability to learn optimal weights and achieve accurate classification.

DESCRIPTION:

Introduction:

Perceptron Learning is one of the earliest and simplest forms of machine learning algorithms, designed for binary classification tasks. Developed by Frank Rosenblatt in 1958, the Perceptron is a foundational algorithm in the field of artificial intelligence, serving as the building block for more complex neural networks. The Perceptron works by learning a linear decision boundary, or hyperplane, that separates two classes in a dataset based on input features.

The Perceptron algorithm operates through an iterative process where it adjusts the weights assigned to each feature of the input data. Initially, the weights are set to zero or small random values. For each training example, the algorithm calculates a weighted sum of the input features and passes it through a step function to make a prediction. If the prediction matchesthe actual label, the weights remain unchanged. However, if the prediction is incorrect, the algorithm updates the weights by adding or subtracting a fraction of the input features based on the difference between the actual and predicted labels. This process continues for a predefined number of iterations or until the weights no longer change, indicating convergence.



(reference: https://towardsdatascience.com/the-iris-dataset-a-little-bit-of-history-and-biology-fb4812f5a7b5)

[dataset contains 4 features that describe the flower and classify them as belonging to one ofthe 3 classes.

We strip the last 50 rows of the dataset that belongs to the class ‘Iris-virginica’ and use only 2 classes ‘Iris-setosa’ and ‘Iris-versicolor’ because these classes are linearly separable and the algorithm converges to a local minimum by eventually finding the optimal weights.]

The Perceptron is particularly effective for linearly separable datasets, where it can successfully find a hyperplane that separates the classes. For example, in the classic Iris dataset, the Perceptron can be used to distinguish between the Iris-setosa and Iris-versicolorclasses based on features such as petal length and sepal length.

Importance:

The Perceptron Learning algorithm is significant not only as a historical milestone in AI but also as a practical tool for understanding the principles of linear classifiers. It forms the basis for more sophisticated models, such as multi-layer perceptrons and deep neural networks. By mastering the Perceptron, one gains insight into how more advanced machine learning models learn and adapt, making it an essential concept in the study of artificial intelligence and machine learning.

Procedure:

The provided program demonstrates the implementation of the Perceptron learning algorithm, a fundamental machine learning algorithm for binary classification. Here’s abreakdown of what the program does:

1. Data Loading and Preprocessing:

The program starts by loading the Iris dataset from the UCI Machine Learning Repository. The dataset originally contains 150 samples with four features, describing three classes of iris flowers: Iris-setosa, Iris-versicolor, and Iris-virginica.

To simplify the problem and ensure that the data is linearly separable, the program strips thelast 50 rows corresponding to the Iris-virginica class. This leaves only two classes: Iris-setosa and Iris-versicolor.

The labels are then converted to binary values: 0 for Iris-setosa and 1 for Iris-versicolor.

2. Data Visualization:

A scatter plot is generated to visualize the dataset using two features: petal length and sepal length. The plot shows that the two classes can be clearly separated by a straight line, confirming that they are linearly separable.

3. Perceptron Algorithm Implementation:

The core of the program is the implementation of the Perceptron learning algorithm. The algorithm initializes the weights to zero and iteratively updates them based on the classification errors it encounters.

For each training example, the Perceptron calculates a weighted sum of the input features and predicts the class label based on whether the sum is greater than zero. If the prediction is incorrect, the weights are adjusted accordingly.

The algorithm runs for a specified number of iterations (`num_iter`), or until the weights stabilize, meaning the classification accuracy no longer improves.

4. Tracking Misclassifications:

The program keeps track of the number of misclassified examples in each iteration and plots this information to visualize the learning process. As the algorithm progresses, the number of misclassifications decreases, indicating that the Perceptron is learning the optimal weights to separate the two classes.

Important Notes:

The program highlights that a single-layer Perceptron can only solve problems where the datais linearly separable.

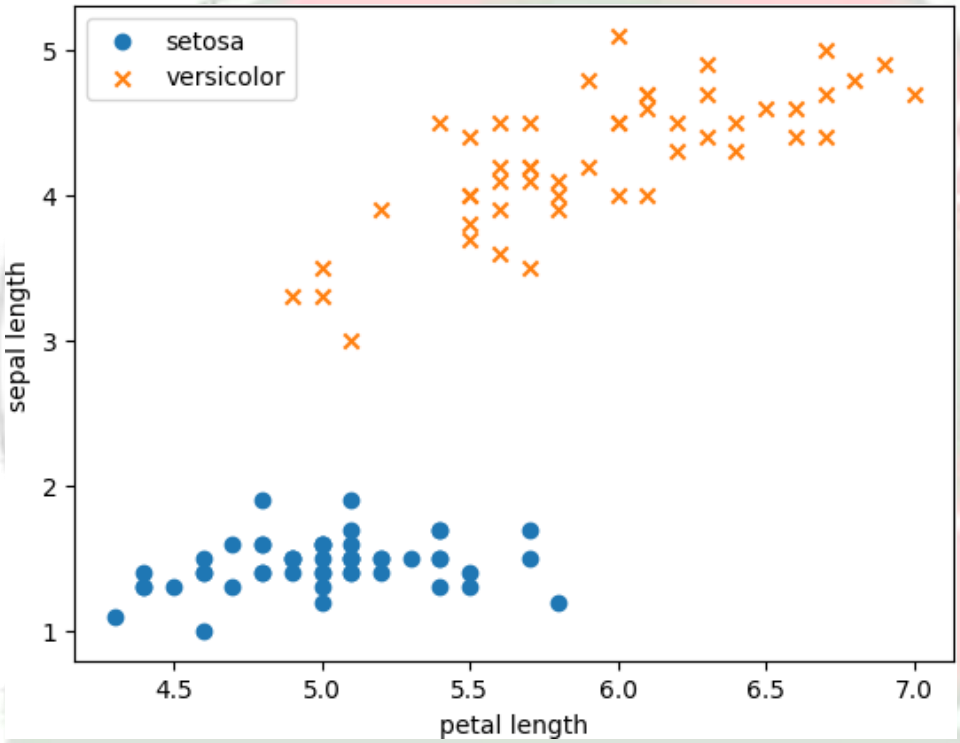
It also mentions that while the Perceptron is designed for binary classification, it can be extended to multiclass classification by using one Perceptron per class.

Overall, the program provides a clear demonstration of how the Perceptron algorithm works, from data preprocessing to visualizing the learning process, emphasizing its application in binary classification tasks.

CODE:

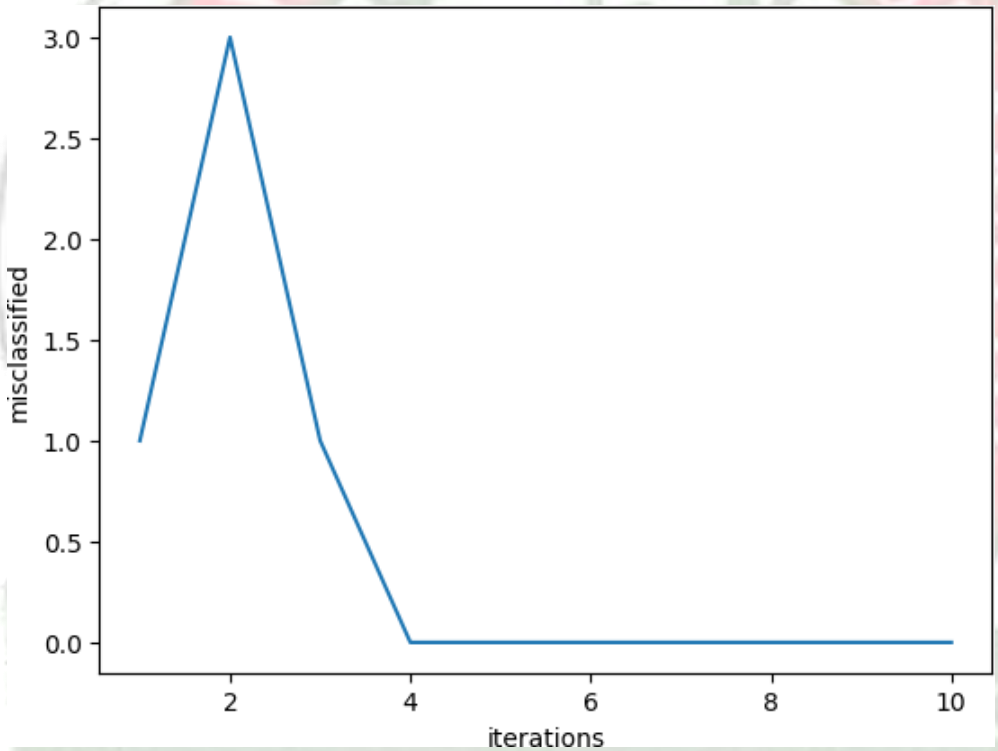
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def load_data():
    URL_ = 'https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data'
    data = pd.read_csv(URL_, header=None)
    print(data)

    # Make the dataset linearly separable
    data = data[:100]
    data[4] = np.where(data.iloc[:, -1] == 'Iris-setosa', 0, 1)  # 4
input features
    data = np.asmatrix(data, dtype='float64')
    return data

data = load_data()
```

```
plt.scatter(np.array(data[:50, 0]), np.array(data[:50, 2]), marker='o',
label='setosa')
plt.scatter(np.array(data[50:, 0]), np.array(data[50:, 2]), marker='x',
label='versicolor')
plt.xlabel('petal length')
plt.ylabel('sepal length')
plt.legend()
plt.show()
```




```
def perceptron(data, num_iter):
    features = data[:, :-1]
    labels = data[:, -1]

    # Set weights to zero
    w = np.zeros(shape=(1, features.shape[1] + 1))

    misclassified_ = []
    for epoch in range(num_iter):
        misclassified = 0
        for x, label in zip(features, labels):
            x = np.insert(x, 0, 1)
            y = np.dot(w, x.transpose())
            target = 1.0 if (y > 0) else 0.0

            delta = (label.item(0, 0) - target)

            if delta:  # Misclassified
                misclassified += 1
                w += (delta * x)
```

```
        misclassified_.append(misclassified)

    return (w, misclassified_)

num_iter = 10
w, misclassified_ = perceptron(data, num_iter)

epochs = np.arange(1, num_iter + 1)
plt.plot(epochs, misclassified_)
plt.xlabel('iterations')
plt.ylabel('misclassified')
plt.show()
```



RESULT:

The result of this implementation is a trained Perceptron model that successfully finds a decision boundary, or straight line, that separates the two classes in the Iris dataset: Iris-setosaand Iris-versicolor. By iteratively adjusting the weights based on the training examples, the Perceptron algorithm converges, meaning it finds a set of weights that correctly classifies all the data points in this linearly separable dataset. The scatter plot visualization confirms that the two classes are distinctly separated by this boundary, demonstrating the effectiveness ofthe Perceptron in handling linearly separable data. This outcome showcases the algorithm's ability to learn and create a model that accurately distinguishes between the two flower species based on their features.


CHAITANYA BHARATHI
INSTITUTE OF TECHNOLOGY
స్వయం తేజస్వినే భవ
1979