Name: Vinay Singh

Roll No : 177 B

Registration No : 20010232

Subject : Machine Learning

# Experiment No. 9

**Aim:** Implementing a Neural Network based estimation using Scikit learn.

**Problem Statement**: Wine classification is a very famous multi-class classification problem.Build a ML model to classify the type of wine.

**Dataset Description**:

The Wine dataset is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The Dataset comprises of 13 features: alcohol, malic acid, ash, alcalinit of ash, magnesium, flavanoids,hue,color_intensity,od280/od315_of_diluted_wines,nonflavanoid_phenols, total_phenols, proanthocyanins, proline and type of wine cultivar.

This data has three type of wine Class_0, Class_1, and Class_2.

| Class | 3 |
|---|---|
| Samples per class | [59,71,48] |
| Samples total | 178 |
| Dimensionality | 13 |
| Features | real, positive |

**Problem Analysis**:

Upon critical Analysis of the given Problem, it can be seen that there's a Relationship between the class of wine and the features of the wine. This type of problem can easily be dealt with by using Multi-Layer Perceptron Classifier.

Multi-layer perceptron (MLP) is a supplement of feed forward neural network. It consists of three types of layers—the input layer, output layer and hidden layer. The input layer receives the input signal to be processed. The required task such as prediction and classification is performed by the output layer. An arbitrary number of hidden layers that are placed in between the input and output layer are the true computational engine of the MLP. Similar to a feed forward network in a MLP the data flows in the forward direction from input to output layer. The neurons in the MLP are trained with the back propagation

learning algorithm. MLPs are designed to approximate any continuous function

and can solve problems which are not linearly separable. The major use cases of MLP are pattern classification, recognition,prediction and approximation.

**Machine Learning Task**:

*Classification*

**Predictor**:

- *Alcohol*
- *Malic acid*
- *Ash*
- *Alcalinity of ash*
- *Magnesium*
- *Total phenols*
- *Flavanoids*
- *Nonflavanoid phenols*
- *Proanthocyanins*
- *Color intensity*
- *Hue*
- *OD280/OD315 of diluted wines*
- *Proline*

**Response Variable**: EITHER OF THE THREE Classes:

*Class_0, Class_1, Class_2*

**ModelParameter**

*Weights* (Learned During Training)

**HyperParameter**

- *Bias Inputs*
- *Number of Layers*
- *Number of Neurons in each Layer*
- *Activation Function*
- *Learning rule/Algorithm*
- *Learning Rate*
- *Epochs*
- *Batch Size*

**Program** :

```
from sklearn.datasets import load_winewine = load_wine()

  # printing class distribution of wine dataset
  import numpy as np
  print(f'Classes: {np.unique(wine.target)}')
  print(f'Class distribution of the dataset: {np.bincount(wine.target)}')

  # from sklearn.cross_validation import train_test_split(Hold
  out method)
  from sklearn.model_selection import train_test_split as SPLIT
  X_train, X_test, y_train, y_test = SPLIT(wine.data, wine.target,
  test_size=0.25,stratify=wine.target, random_state=123)

  # printing class distribution of test dataset
  print(f'Classes: {np.unique(y_test)}')
  print(f'Class distribution for test data: {np.bincount(y_test)}')

  # MLP is sensitive to feature scaling, hence performing
  scaling # Options: MinmaxScaler and Standardscaler
  # from sklearn.preprocessing import
  MinMaxScaler # scaler = MinMaxScaler()
  from sklearn.preprocessing import StandardScaler as SS
  X_train_stdsc = SS().fit_transform(X_train)
  X_test_stdsc = SS().fit_transform(X_test)
  # Setting of hyperparameters of the network
  from sklearn.neural_network import MLPClassifier as
  MLP mlp =
  MLP(hidden_layer_sizes=(10,),learning_rate_init=0.001,max_iter=5000)

  # Calculating Training Time : more neurons, more time
  from time import
  time start = time()
  # Train the model using the scaled training sets
  mlp.fit(X_train_stdsc, y_train)
  end = time()
  print(f'Training Time: {(end-start)*1000:.3f}ms')

  # Predict the response for test dataset
  y_pred = mlp.predict(X_test_stdsc) # scaled

  # Import scikit-learn metrics module for evaluating model performance
  from sklearn.metrics import
  classification_report,confusion_matrix,accuracy_score

  # Model Accuracy, how often is the classifier correct?
  print(f'Accuracy:{accuracy_score(y_test, y_pred)}')

  # display the confusion matrix
  print('Confusion Matrix is:\n',confusion_matrix(y_test, y_pred))
  print('Classification Report:\n',classification_report(y_test, y_pred))

  from sklearn.metrics import plot_confusion_matrix as PCM
  PCM(mlp, X_test_stdsc, y_test)
```

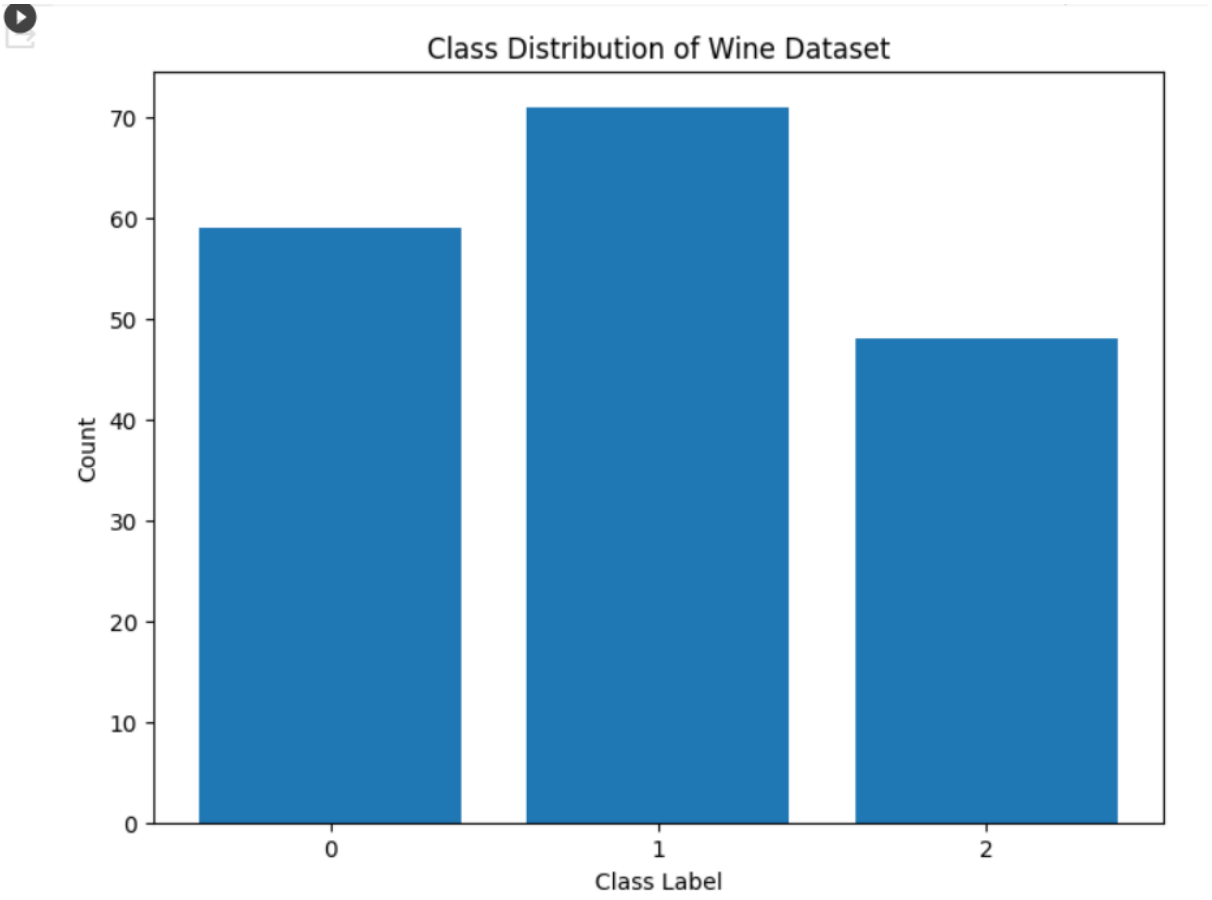
**Output** :

```
Classes: [0 1 2]
Class distribution of the dataset: [59 71 48]
Classes: [0 1 2]
Class distribution for test data: [15 18 12]
Classes: [0 1 2]
Class distribution of the dataset: [59 71 48]
Classes: [0 1 2]
Class distribution for test data: [15 18 12]
```

**Program:**

```
import matplotlib.pyplot as plt

# Class labels and their counts
class_labels = np.unique(wine.target)
class_counts = np.bincount(wine.target)

# Create a bar chart to visualize the class
distribution plt.figure(figsize=(8, 6))
plt.bar(class_labels, class_counts,
tick_label=class_labels) plt.title('Class Distribution of
Wine Dataset') plt.xlabel('Class Label')
plt.ylabel('Count'
) plt.show()

# Now, create a bar chart for the test
dataset test_class_labels = np.unique(y_test)
test_class_counts = np.bincount(y_test)

plt.figure(figsize=(8, 6))
plt.bar(test_class_labels,
test_class_counts,
tick_label=test_class_labels)
plt.title('Class Distribution of Test
Dataset') plt.xlabel('Class Label')
plt.ylabel('Count'
) plt.show()
```

**Graph** :


Class Distribution of Wine Dataset
Count
70
60
50
40
30
20
10
0
0
1
2
Class Label


**Conclusion** :

I have successfully implemented Neural Network based estimation using Scikit learn for multiclass classification Problem.

# Experiment No . 10

**Aim:** Experiment on classification using transfer learning (Deep learning).

**Problem Statement**:

Image Classification using a Pretrained Convolutional Neural Network.

**CNN Description**:

**Problem Analysis**:

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks make them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNsare on the lower extreme.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage.

**Machine Learning Task**: *Classification*

**Predictor**: Image of size 224 X 224 X3

**Response Variable**: *One of 1000 object categories, such as keyboard, mouse, pencil, andmany animals.*

**Model Parameters**: *Weights* (Learned During Training)

**Hyper Parameters**:

- *Bias Inputs*
- *Number of Layers*
- *Number of Neurons in each Layer*
- *Activation Function*
- *Learning rule/Algorithm*
- *Learning Rate*
- *Epochs*
- *Batch Size*
- *Dropout for regularization*

**Describe the pretrained CNN used:**

EfficientNet was introduced by a group of researchers at Google Research. The paper titled "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks" was authored by Mingxing Tan and Quoc V. Le. It was published in June 2019, and it quickly gained attention in the deep learning community for its innovative approach to model scaling and its impressive performance on various computer vision tasks.

Mingxing Tan and Quoc V. Le's work on EfficientNet has had a significant impact on the field of deep learning, and it has inspired further research in the design of efficient neuralnetwork architectures.

EfficientNetV2B0 is a part of the broader EfficientNet family, which includes larger and more powerful models (e.g., B1, B2, B3, and so on) that can achieve even higher accuracy but require more computational resources. When selecting a model, consider the trade-off between model size and computational efficiency based on the specific requirements of yourtask.

**Program:**

```
from tensorflow import keras

# preprocess_input is a pass-through function for EffNets from
keras.applications.efficientnet import
preprocess_input,decode_predictions from tensorflow.keras.preprocessing
import image

#urllib is a package that collects several modules for workingwith URLs: urllib.
request is for opening and reading URLs. import urllib.request
 import matplotlib.pyplot as
 pltimport numpy as np

 # Take a Public domain image
url =
```

```
'https://upload.wikimedia.org/wikipedia/commons/0/02/Black_bear_large.jpg'
 #PIL is a popular image processing library. Here it is used toload images as
 NumPy arrays
 import PIL
 img = PIL.Image.open(urllib.request.urlopen(url))img
 = img.resize((224, 224))
 img_batch = np.expand_dims(img, 0)

 # Load Pretrained model trained using ‘imagenet’ dataset and#
 run prediction and include the fully connected (dense)
 # layers at the top of the networkeffnet =
keras.applications.EfficientNetV2B0(weights='imagenet',include_top=True)
pred = effnet.predict(img_batch) print(decode_predictions(pred))# prints
class number, classname, and confidence of prediction
 plt.imshow(img
 )
 plt.title(f'Class:
{decode_predictions(pred)[0][0][1]}\nConfidence:
 {decode_predictions(pred)[0][0][2]*100}%'
 ) plt.show()
```

**Output** :

**Conclusion** :

I have Successfully Predicted the class of the image by using the EfficientNet Convolutional Neural Network.