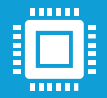
Introduction to MLP

Middlesex University Dubai

Multilayer Perceptron is a type of ANN(Artificial Neural Network).

It is supervised learning used for Classification tasks.



In order to classify data into various categories or classes, MLPClassifier's function in machine learning is to discover complex nonlinear relationships among input features and target labels.

It differs from logistic regression in that one or more non-linear layers, also referred to as hidden layers, exist between the input and output layers.

Data Preprocessing

- Initialize the python libraries like numpy, MLPClassifier, LabelEncoder, train_test_split, pd and Loaded the dataset using pandas library.
- Data Analysis: Removed duplicates(19), there are no null values, Then standardized dataset using LabelEncoder().
- Selected target variable as Recurred and others as input variable.
- O/P variable represents Predicting the possibility of a disease recurring is frequently crucial for treatment planning and patient management.
- Split data into train(80%) and test(20%).

```python
# Import the libraries:
from sklearn.linear_model import Perceptron
from sklearn.neural_network import MLPClassifier
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
# Load the dataset
data = pd.read_csv("ClassificationDataset.csv")
data.head()
```

```
[39]: data.duplicated().sum()

[39]: 19

[40]: data.drop_duplicates(inplace=True)
```

```python
# Encode categorical variables
label_encoder = LabelEncoder()
categorical_cols = ['Gender', 'Smoking', 'Hx Smoking', 'Hx Radiothreapy', 'Thyroid Function',
                    'Physical Examination', 'Adenopathy', 'Pathology', 'Focality', 'Risk',
                    'T', 'N', 'M', 'Stage', 'Response']
for col in categorical_cols:
    data[col] = label_encoder.fit_transform(data[col])

# Split data into features and target
X = data.drop(columns=['Recurred'])
y = data['Recurred']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
data.head()
```

	Age	Gender	Smoking	Hx Smoking	Hx Radiothreapy	Thyroid Function	Physical Examination	Adenopathy	Pathology	Focality	Risk	T	N	M	Stage	Response	Recurred
0	11	0	0	0	0	2	3	3	2	1	2	0	0	0	0	2	0
1	18	0	0	1	0	2	1	3	2	1	2	0	0	0	0	1	0
2	14	0	0	0	0	2	4	3	2	1	2	0	0	0	0	1	0
3	46	0	0	0	0	2	4	3	2	1	2	0	0	0	0	1	0
4	46	0	0	0	0	2	1	3	2	0	2	0	0	0	0	1	0

Model Evaluation before tuning

Randomly selecting the Parameters values shown in the code:

- **hidden_layer_sizes** -specifies the number of layers and the number of neurons in each layer. I have took (10,) it signifies that single hidden layer with 10 neurons.
- **Activation**-Used in hidden layer, that introduces the nonlinearity into the model, 'logistic': Indicates that the hidden layer's activation function is the logistic sigmoid function.
- **Solver-** Explains the optimization algorithm that is used to adjust the network's weights during training. 'sgd': The optimization algorithm makes use of stochastic gradient descent (SGD).
- **Learning rate-** Scheduling for any updates in weights. It is only used when solver is "sgd" , 'constant': During training, the learning rate is kept constant.
- Setting the initial learning rate to 0.001 with **learning_rate**_init=0.001 and the maximum number of iterations to 100 for model training is known as max_iter =1000.

Training and Testing Accuracy:

75.6%, shows how accurate the model was on the training dataset. It displays the percentage of instances that are properly identified, indicating how well the model fits the training set.

This percentage, which is roughly 79.5%, shows how accurate the model is on the testing dataset. It indicates the percentage of correctly identified instances and assesses the model's ability to generalize to new scenarios.

This shows that your model is not severely overfitting, as there is not much of a difference between these two accuracies.

```
from sklearn.neural_network import MLPClassifier
# Initialize MLPClassifier
model = MLPClassifier(hidden_layer_sizes=(10,),
                      activation='logistic',
                      solver='sgd',
                      learning_rate='constant',
                      learning_rate_init=0.001,
                      max_iter=1000)

# Fit the model
model.fit(X_train, y_train.values.ravel())

# Testing
train_accuracy = model.score(X_train, y_train)
test_accuracy = model.score(X_test, y_test)
print("Training Accuracy before tuning:", train_accuracy)
print("Testing Accuracy before tuning:", test_accuracy)
```

```
Training Accuracy before tuning: 0.7560137457044673
Testing Accuracy before tuning: 0.7945205479452054
c:\users\nishu\appdata\local\programs\python\python39\lib\si
stic Optimizer: Maximum iterations (1000) reached and the op
  warnings.warn(
```

Hyperparameter

- Hyperparameter tuning: With **GridSearchCV**, specify a grid of parameters to be optimized. It looks through each possible combination of these hyperparameters once I have specified them along with its possible values.
- Cross-Validation(GridSearchcv)- Divide the training dataset into several folds, or subsets. One fold is kept out as the validation set for each cross-validation iteration, while the model is trained on the remaining folds. For every set of hyperparameters, this procedure is repeated.
- Grid of parameters: I Explored alternative configurations of the MLPClassifier's hyperparameters, such as learning rates, maximum iterations, activation functions, solvers, regularization strengths (alpha), and hidden layer sizes, put in a variable param_grid.
- Model Iteration: An MLPClassifier was initialized with a 1000 iteration limit.
- Initialization of GridSearchCV: Created a GridSearchCV object (grid_search) to use parallel processing (n_jobs=-1) and 5-fold cross-validation to find the optimal hyperparameters.
- Found the following hyperparameter combinations to be the best_parameters:
  'relu' as the activation function
  Alpha: 0.0001 (regularization strength)
  Layers hidden: (50, 50) –(2 hidden layer each with 50 neurons)
  'Constant' learning rate
  solver= Adam
  100 iterations is the maximum.

After tuning the parameters

```
]: from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 50, 25)],
    'activation': ['logistic', 'tanh', 'relu'],
    'solver': ['adam', 'sgd'],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate': ['constant', 'adaptive'],
     'max_iter': [100]
}

# Initialize MLPClassifier
model = MLPClassifier(max_iter=1000)

# Initialize GridSearchCV with cross-validation
grid_search = GridSearchCV(model, param_grid, cv=5, n_jobs=-1)

# Fit the model
grid_search.fit(X_train, y_train.values.ravel())

# Print best parameters
print("Best Parameters:", grid_search.best_params_)

# Print accuracy after tuning
print("\nAccuracy After Tuning:")
```

Model Evaluation After tuning

Middlesex University Dubai

Training Accuracy:

The 91% percentage of accurately predicted occurrences among all the examples in the training dataset is represented by this accuracy. A high training accuracy shows that the model has made good learnings from the training set and is very confident in its ability to predict the training instances.

Testing Accuracy:

A high testing accuracy shows how well the model works on new, untested examples and how well it generalizes to unknown data. Given that the testing and training accuracy are almost equal, it is likely that the model has not overfit to the training set and is capable of producing reliable predictions when faced with new data.

```
Best Parameters: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'constant', 'max_iter': 100, 'solver': 'adam'}

Accuracy After Tuning:
Training Accuracy: 0.9104617182933957
Testing Accuracy: 0.9041095890410958
```

Conclusion:

After hyperparameter modification, the model performs much better, obtaining high accuracies on the testing and training sets.

With 91.0% training accuracy and 90.4% testing accuracy, the model demonstrates excellent accuracy for prediction and the ability to generalize to new data.

The model's performance was effectively enhanced by the hyperparameters chosen using crossvalidation - GridSearchCV.

It's possible that the model was able to understand the underlying patterns in the data more effectively and without overfitting due to the combination of parameters selected during tuning.

THANK YOU