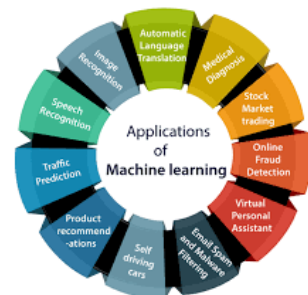
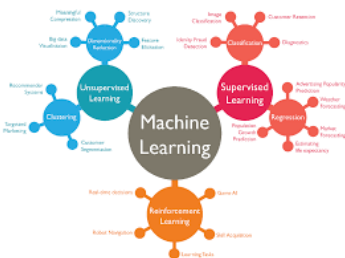


Manual

Machine Learning

Using Python Lab



Dr. G.N V G Sirisha
Sri. V V Sivarama Raju
Sri. B N V Narasimha Raju
Sri. S. Suryanarayana Raju



Department of Computer Science and Engineering
SRKR Engineering College (A), Bhimavaram, India

This lab manual is intended to aid third-year undergraduate computer science and engineering students in their course *Machine Learning Lab* [B20CS3209].

About the author

G N V G Sirisha: She got her PhD, M.Tech. and B.Tech. degrees from Andhra University. Presently she is working as an associate professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India. Her research interests include Data Science, Information Retrieval and Machine Learning.

V V Sivarama Raju: He is currently pursuing a Ph.D. at BPUT University, M.Tech From Andhra University and B.Tech From Andhra University. Presently he is working as an assistant professor in the Department of Computer Science and Engineering at SRKR Engineering College, Bhimavaram, India. He has prominent experience in Machine Learning and Deep Learning.

B N V Narasimha Raju: He is currently pursuing a Ph.D. at KL University, M.Tech From Andhra University and B.Tech From Andhra University. Presently he is working as an assistant professor in the Department of Computer Science and Engineering at SRKR Engineering College, Bhimavaram, India. He has prominent experience in Machine Learning and Deep Learning.

S. Suryanarayanaraju: He is currently pursuing a Ph.D. at GIET University, M.Tech From Andhra University and B.Tech From JNTUH. Presently he is working as an assistant professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India. He has a prominent experience in Big Data Analytics and Mining.

For private circulation among 3/4 B.Tech.(CSE) students.

Preface

The term machine learning refers to the automated detection of meaningful patterns in data. In the past couple of decades, it has become a common tool in almost any task that requires information extraction from large data sets.

We are surrounded by a machine learning-based technology: Search engines learn how to bring us the best results (while placing profitable ads), antispam software learns to filter our email messages, and credit card transactions are secured by software that learns how to detect frauds. Digital cameras learn to detect faces and intelligent personal assistance applications on smartphones learn to recognize voice commands. Cars are equipped with accident-prevention systems that are built using machine learning algorithms.

Machine learning is also widely used in scientific applications such as bioinformatics, medicine, and astronomy. One common feature of all of these applications is that, in contrast to more traditional uses of computers, in these cases, due to the complexity of the patterns that need to be detected, a human programmer cannot provide an explicit, fine-detailed specification of how such tasks should be executed.

Taking examples from intelligent beings, many of our skills are acquired or refined through learning from our experience (rather than following explicit instructions given to us). Machine learning tools are concerned with endowing programs with the ability to “learn” and adapt.

Evaluation of Practical subjects

For practical subjects there shall be continuous evaluation during the semester for 15 internal marks and 35 end examination marks. The internal 15 marks shall be awarded as follows: day to day work - 5 marks, Record-5 marks and the remaining 5 marks to be awarded by conducting an internal laboratory test. The end examination shall be conducted by the teacher concerned and external examiner appointed by Dean Evaluation/ Controller of examinations

Evaluation Scheme	
Examination	Marks
Exercise Programs	5
Record	5
Internal Exam	5
External Exam	35

Machine Learning Using Python Lab

Course Objectives

1. Implement different mechanisms in pre-processing and model evaluation & implementation
- 2 Implement different dimensionality reduction techniques
- 3 Implement different clustering & classification techniques
- 4 Evaluate the model.
- 5 Implement simple linear, logistic regressions and Feed-Forward Network

Course Outcomes

- CO1. Design Pre-processing model for their own data sets.
- CO2. Apply dimensional reduction techniques for their own datasets.
- CO3. Develop different clustering & classification techniques.
- CO4. Evaluate the model with Lasso and Ridge Regularization.
- CO5. Design neural network for structured, unstructured data classification and Regression.

List of Experiments

- 1 Vector addition.
- 2 Data pre-processing: Handling missing values, handling categorical data, bringing features to the same scale, and selecting meaningful features.
- 3 Regression model.
- 4 Write a program to implement the KNN classifier and logistic regression for binary classification and multiclass classification.
- 5 Ensemble Learning, grid search and learning and validation curves.
- 6 Write a program for Data Clustering (K-Means) and evaluate the clustering model.
- 7 Compressing data via dimensionality reduction: PCA, LDA.
- 8 Model Evaluation and Optimization: K-fold cross-validation.
- 9 Write a program to reduce the variance of a linear regression model using Lasso and Ridge Regularization.
- 10 Perceptron for digits.
- 11 Feed-Forward Network for wheat seeds dataset.
- 12 Write a program to implement a neural network for regression.
- 13 Write a program to save and load a trained machine learning model



Additional Experiments

1. Write a program to implement data pre-processing techniques like data sampling, data discretization and data augmentation.
2. Write a program to implement a Naïve Bayes algorithm.
3. Write a program to implement classification using SVM.
4. Write a program to implement a regression tree.
5. Write a program to implement Boosting techniques.
6. Write a program to implement Hierarchical clustering.
7. Write a program to implement a Multilayer perceptron.

Session #1

Vector addition

Learning Objective

To implement the Vector addition.

Learning Context

Here we shall learn how to perform Vector addition and subtraction in Python. A vector in programming terms refers to a one-dimensional array. An array is one of the data structures that stores similar elements i.e elements having the same data type.

The general features of the array include

- An array can contain many values based on the same name.
- Accessing the elements is based on the index number. If the array size is "n" the last index value is [n-1] and the starting index is always [0].
- We can also slice the elements in the array [start: end] based on the start and end position.

Addition and Subtraction of Vectors in Python

Now let's learn how to perform the basic mathematical operations such as addition and subtraction on arrays in Python. To perform this task we need to know about the Numpy module in Python. The Numpy is the Numerical Python that has several inbuilt methods that shall make our task easier. The easiest and simplest way to

create an array in Python is by adding comma-separated literals in matching square brackets. For example

```
A = [1, 2,3]
```

```
B = [4,5,6]
```

We can even create multidimensional arrays, for example, a two-dimensional array as shown below :

```
A = ([1,2,3], [4,5,6])
```

```
B = ([2,-4,7] , [5,-20,3])
```

To use this amazing module, we need to import it. Let's look into the code to utilize this module for vector addition and subtraction in Python.

```
import numpy as NP
```

```
A = [4, 8, 7]
```

```
B = [5, -4, 8]
```

```
print("The input arrays are :\n","A:",A ,"\n","B:",B)
```

```
Res= NP.add(A,B)
```

```
print("After addition the resulting array is :",Res)
```

So, in the above code, variables A and B are used to store the array elements. To perform the addition we need to call the add() method of the NumPy module as NP.add(). Here we have aliased the NumPy as NP which is not necessary. We can

directly write as `NumPy.add()`. To perform subtraction on the same array elements, we just need to write another line of code invoking the subtract method i.e `NP.subtract()` and print the result obtained after the subtraction.

Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from Preprocessing to Deep Learning", O'REILLY Publisher, 2018
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017
3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Write a program to Vector addition.

Solutions

```
import numpy as NP
A = [4, 8, 7]
B = [5, -4, 8]
print("The input arrays are :\n","A:",A ,"\n","B:",B)
Res= NP.add(A,B)
print("After addition the resulting array is :",Res)
Output:
```

The input arrays are :

A: [4, 8, 7]

B: [5, -4, 8]

After addition the resulting array is : [9 4 15]

```
import numpy as NP
```

```
A = [4, 8, 7]
```

```
B = [5, -4, 8]
```

```
print("The input arrays are :\n","A:",A ,"\n","B:",B)
```

```
Res1= NP.add(A,B)
```

```
Res2= NP.subtract(A,B)
```

```
print("Result of Addition is :",Res1,"\n","Result of Subtraction is:",Res2)
```

Output:

The input arrays are :

A: [4, 8, 7]

B: [5, -4, 8]

Result of Addition is : [9 4 15]

Result of Subtraction is: [-1 12 -1]


```
# create a vector
```

```
from numpy import array
```

```
v = array([1, 2, 3])
```

```
print(v)
```

Output:



```
[1 2 3]
```

```
from numpy import array
```

```
a = array([1, 2, 3])
```

```
print(a)
```

```
b = array([1, 2, 3])
```

```
print(b)
```

```
c = a + b
```

```
print(c)
```

Output:

```
[1 2 3]
```

```
[1 2 3]
```

```
[2 4 6]
```

```
from numpy import array
```

```
a = array([1, 2, 3])
```

```
print(a)
```

```
b = array([0.5, 0.5, 0.5])
```

```
print(b)
```


```
c = a - b
```

```
print(c)
```

Output:

```
[1 2 3]
```

```
[0.5 0.5 0.5]
```



```
[0.5 1.5 2.5]
```

```
# multiply vectors
```

```
from numpy import array
```

```
a = array([1, 2, 3])
```

```
print(a)
```

```
b = array([1, 2, 3])
```

```
print(b)
```

```
c = a * b
```

```
print(c)
```

```
Output:
```

```
[1 2 3]
```

```
[1 2 3]
```

```
[1 4 9]
```

```
from numpy import array
```

```
a = array([1, 2, 3])
```

```
print(a)
```

```
b = array([1, 2, 3])
```

```
print(b)
```

```
c = a / b
```

```
print(c)
```

```
Output:
```

```
[1 2 3]
```

```
[1 2 3]
```

```
[1. 1. 1.]
```

```
# dot product vectors
from numpy import array
a = array([1, 2, 3])
print(a)
b = array([1, 2, 3])
print(b)
c = a.dot(b)
print(c)
```

Output:

```
[1 2 3]
```

```
[1 2 3]
```

```
14
```

```
# VECTOR AND SCALAR
```

```
from numpy import array
a = array([1, 2, 3])
print(a)
s = 0.5
print(s)
c = s * a
print(c)
```

Output:

```
[1 2 3]
```

```
0.5
```

```
[0.5 1.  1.5]
```

Session #2

Data pre-processing

Learning Objective

To perform Data pre-processing like Handling missing values, handling categorical data, bringing features to the same scale, and selecting meaningful features.

Learning Context

Checking Null Values



`DataFrame.isnull()` function detects missing values in the given object. It returns a Boolean same-sized object indicating the values of NA. `isnull()` is the method that returns true if the value is null and false otherwise.

Handling Missing Values

Missing data is defined as the values or data that is not stored (or not present) for some variables in the given dataset. The first step in handling missing values is to look at the data carefully and find out missing values using `isnull()`. Some of the measures to handle it by using `dropna()` and `fillna()`

Handling Categorical Data

When your data has categories represented by strings, it will be difficult to use them to train machine learning models which often only accept numeric data.

Instead of ignoring the categorical data and excluding the information from our model, you can transform the data so it can be used in your models. Categorical data is a type of data that is used to group information with similar characteristics, while numerical data is a type of data that expresses information in the form of numbers. Most machine learning algorithms cannot handle categorical variables unless we convert them to numerical values. Many algorithm's performances even vary based on how the categorical variables are encoded. categorical values are divided into two categories:

- Nominal - No particular order
- Ordinal - there is some order between values

Bringing Features to the Same Scale

Feature scaling is the process of normalizing the range of features in a dataset. Real-world datasets often contain features that vary in degrees of magnitude, range and units. Therefore, for machine learning models to interpret these features on the same scale, we need to perform feature scaling. Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing.

More specifically, we will be looking at 3 different scalers in the Scikit-learn library for feature scaling and they are:

- MinMax Scaler
- Standard Scaler

- Robust Scaler

Selecting Meaningful Features

The `iloc()` function in Python is defined in the Pandas module that helps us to select a specific row or column from the data set. Using the `iloc` method in Python, we can easily retrieve any particular value from a row or column by using index values.

Syntax:

```
pandas.dataset.iloc[row, column]
```

parameters:

The `iloc` function in Python takes two parameters. However, both parameters of the `iloc()` method are optional. Let us discuss both of these parameters:

- The row parameter is an optional parameter that specifies the index position of the row in the form of an integer or list of integers.
- The column parameter is also an optional parameter that specifies the index position of the column in the form of an integer or list of integers.

If we specify only row value, then the `iloc` function returns a Pandas Series. If we specify the row value and column value, then the `iloc` function returns all the content of the specified cell. If we specify a list of values, the python `iloc` function returns a Pandas DataFrame.

Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from preprocessing to Deep learning", O'REILLY Publisher, 2018
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017
3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Data pre-processing: Handling missing values, handling categorical data, bringing features to same scale, selecting meaningful features

Solution

1. Importing the necessary libraries

->import numpy as np

->import pandas as pd

2. Importing the dataset:

-> data1=pd.read_csv('dataset(1).csv')

data1

Output:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

-> data1.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Country     10 non-null    object
1   Age         9 non-null     float64
2   Salary      9 non-null     float64
3   Purchased   10 non-null    object
dtypes: float64(2), object(2)
memory usage: 448.0+ bytes
```

3. Checking for the null values in the data:

-> data1.isnull().sum()

Output:

```
Country      0
Age          1
Salary       1
Purchased    0
dtype: int64
```

3.1. Dropping of the null records from the original data

-> newdata= data1.dropna()

newdata

Output:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
5	France	35.0	58000.0	Yes
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

4. Filling the missing values with mean of the data

```
-> mean_value=data1['Age'].mean()
```

```
data1['Age'].fillna(value=mean_value, inplace=True)
```

```
data1.head(10)
```

Output:

	Country	Age	Salary	Purchased
0	France	44.000000	72000.0	No
1	Spain	27.000000	48000.0	Yes
2	Germany	30.000000	54000.0	No
3	Spain	38.000000	61000.0	No
4	Germany	40.000000	NaN	Yes
5	France	35.000000	58000.0	Yes
6	Spain	38.777778	52000.0	No
7	France	48.000000	79000.0	Yes
8	Germany	50.000000	83000.0	No
9	France	37.000000	67000.0	Yes

```
-> mean_value=data1['Salary'].mean()
```

```
data1['Salary'].fillna(value=mean_value, inplace=True)
```

```
data1.head(20)
```

Output:

	Country	Age	Salary	Purchased
0	France	44.000000	72000.000000	No
1	Spain	27.000000	48000.000000	Yes
2	Germany	30.000000	54000.000000	No
3	Spain	38.000000	61000.000000	No
4	Germany	40.000000	63777.777778	Yes
5	France	35.000000	58000.000000	Yes
6	Spain	38.777778	52000.000000	No
7	France	48.000000	79000.000000	Yes
8	Germany	50.000000	83000.000000	No
9	France	37.000000	67000.000000	Yes

5. Handling Categorical Data:

-> from sklearn.preprocessing import LabelEncoder

```
le=LabelEncoder()
```

```
data1['Country']=le.fit_transform(data1[['Country']])
```

```
data1['Purchased']=le.fit_transform(data1[['Purchased']])
```

Output:

	Country	Age	Salary	Purchased
0	0	44.0	72000.000000	0
1	2	27.0	48000.000000	1
2	1	30.0	54000.000000	0
3	2	38.0	61000.000000	0
4	1	40.0	63777.777778	1

6. Feature Scaling:

-> from sklearn.preprocessing import MinMaxScaler

```
s=MinMaxScaler(feature_range=(0, 1))
```

```
name=data1.columns
```

```
data2=s.fit_transform(data1)
```

```
data2=pd.DataFrame(data2,columns=name)
```

```
display(data2)
```

Output:

	Country	Age	Salary	Purchased
0	0.0	0.739130	0.685714	0.0
1	1.0	0.000000	0.000000	1.0
2	0.5	0.130435	0.171429	0.0
3	1.0	0.478261	0.371429	0.0
4	0.5	0.565217	0.450794	1.0
5	0.0	0.347826	0.285714	1.0
6	1.0	0.512077	0.114286	0.0
7	0.0	0.913043	0.885714	1.0
8	0.5	1.000000	1.000000	0.0
9	0.0	0.434783	0.542857	1.0

7. Feature selection:

```
-> from sklearn.feature_selection import SelectKBest
```

```
from sklearn.feature_selection import chi2
```

```
bestfit=SelectKBest(score_func=chi2,k=3)
```

```
fit=bestfit.fit(data2.iloc[:,0:-1],data2.iloc[:,-1])
```

```
pd.DataFrame({"columns":["Country","Age","Salary"], "Scores":fit.scores_})
```

Output:

	columns	Scores
0	Country	0.500000
1	Age	0.070076
2	Salary	0.007011

Session #3

Regression model

Learning Objective

To Implement the Regression model

Learning Context

This is about the basics of linear regression and its implementation in the Python programming language. Linear regression is a statistical method for modeling relationships between a dependent variable with a given set of independent variables.

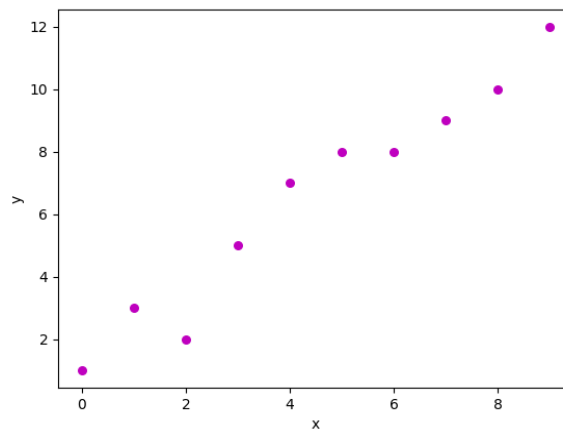
Note: In this, we refer to dependent variables as responses and independent variables as features for simplicity. In order to provide a basic understanding of linear regression, we start with the most basic version of linear regression, i.e. Simple linear regression.

Simple Linear Regression

Simple linear regression is an approach for predicting a response using a single feature. It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x). Let us consider a dataset where we have a value of response y for every feature x :

x	0	1	2	3	4	5	6	7	8	9
y	1	3	2	5	7	8	8	9	10	12

For generality, we define x as feature vector, i.e $x = [x_1, x_2, \dots, x_n]$, y as response vector, i.e $y = [y_1, y_2, \dots, y_n]$ for n observations (in above example, $n=10$). A scatter plot of the above dataset looks like



Now, the task is to find a line that fits best in the above scatter plot so that we can predict the response for any new feature values. (i.e a value of x not present in a dataset). This line is called a regression line. The equation of regression line is represented as:

$$h(x_i) = \beta_0 + \beta_1 x_i$$

Here, $h(x_i)$ represents the predicted response value for i th observation. β_0 and β_1 are regression coefficients and represent y -intercept and slope of regression line respectively. To create our model, we must learn or estimate the values of regression coefficients β_0 and β_1 and once we've estimated these coefficients, we can use the model to predict responses.

In this, we are going to use the principle of Least Squares. Now consider:

$$h(x_i) = \beta_0 + \beta_1 x_i + \varepsilon_i = h(x_i) + \varepsilon_i = \varepsilon_i = y_i - h(x_i)$$

Here, ε_i is a residual error in the i^{th} observation. So, our aim is to minimize the total residual error. We define the squared error or cost function, J as:

$$J(\beta_0, \beta_1) = \frac{1}{2n} \sum_{i=1}^n \varepsilon_i^2$$

Our task is to find the value of β_0 and β_1 for which $J(\beta_0, \beta_1)$ is minimum. Without going into the mathematical details, we present the result here:

$$\beta_1 = \frac{SS_{xy}}{SS_{xx}}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

where SS_{xy} is the sum of cross-deviations of y and x

$$SS_{xy} = \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^n y_i x_i - n\bar{x}\bar{y}$$

and SS_{xx} is the sum of squared deviations of x

$$SS_{xx} = \sum_{i=1}^n (x_i - \bar{x})^2 = \sum_{i=1}^n x_i^2 - n(\bar{x})^2$$

Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from Preprocessing to Deep Learning", O'REILLY Publisher, 2018
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017

-
3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Write a program to Regression model

Solutions

```
import pandas as pd
dataset=pd.read_csv('F:\ML\mldatasets\Realestate.csv')
dataset
x = dataset.iloc[:,[2,3,4]]
y = dataset.iloc[:,-1]
print(x)
print(y)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.1,
random_state=0)
from sklearn.linear_model import LinearRegression
regr = LinearRegression()
regr.fit(x_train, y_train)
print(regr.score(x_test, y_test))
y_pred = regr.predict(x_test)
print(y_pred)
```

Session #4

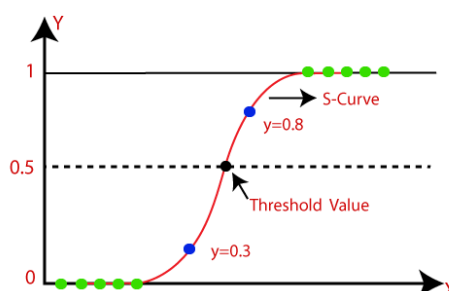
Logistic regression and KNN Classification

Learning Objective

To implement the KNN Classifier, logistic regression for binary and multiclass classification

Learning Context

Logistic Regression: Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.



In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1). The equation of the straight line can be represented as

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y} ; 0 \text{ for } y=0, \text{ and infinity for } y=1$$

But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$\log \left[\frac{y}{1-y} \right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

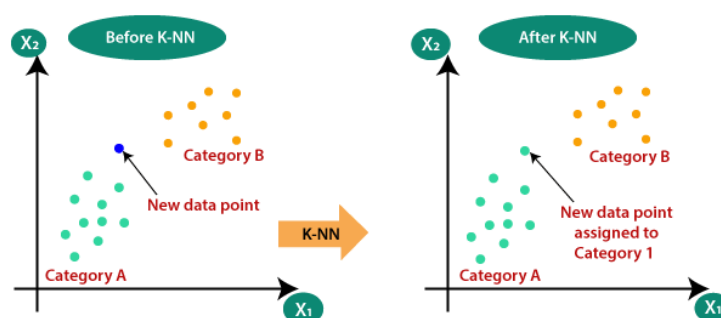
Type of Logistic Regression

On the basis of the categories, Logistic Regression can be classified into three types:

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High"

K-Nearest Neighbours Classifier

K -Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.K-NN is a non- parametric algorithm, which means it does not make any assumption on underlying data.It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.



Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from Preprocessing to Deep Learning", O'REILLY Publisher,2018

-
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017
 3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Write a program to implement KNN Classifier, logistic regression for binary and multiclass classification

Solutions

Binary Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
iris = datasets.load_iris()
features = iris.data[:100,:]
target = iris.target[:100]
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)
logistic_regression = LogisticRegression(random_state=0)
model = logistic_regression.fit(features_standardized,target)
y_pred=lr_iris.predict(features_standardized)
print(metrics.accuracy_score(y_pred,target))
```

Output: 0.5

Multinomial Logistic Regression

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import datasets, metrics
from sklearn.linear_model import LogisticRegression
iris=datasets.load_iris()
iris_data=iris.data
iris_data=pd.DataFrame(iris_data,columns=iris.feature_names)
iris_data['species']=iris.target
iris_data['species'].unique()
features =iris.feature_names
target = 'species'
X=iris_data[features]
y=iris_data[target]
lr_iris=LogisticRegression() # default value for multi class problem is
multinomial
lr_iris =lr_iris.fit(X,y)
y_pred=lr_iris.predict(X)
print(metrics.accuracy_score(y_pred,y))
```

Output: 0.9733333333333334

KNN

```
-> data=pd.read_csv("Social_Network_Ads.csv")
-> data.head()
```

Output:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
-> X = data.iloc[:, [2, 3]].values y = data.iloc[:, 4].values
```

```
-> from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,  
random_state = 0)
```

```
-> from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

```
-> from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
```

```
classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
```

```
classifier.fit(X_train, y_train)
```

Output:

```
▼ KNeighborsClassifier  
KNeighborsClassifier()
```

```
-> y_pred = classifier.predict(X_test)
```

```
-> y_pred
```


Output:

```
array([[0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1])
```

-> from sklearn.metrics import confusion_matrix

```
cm = confusion_matrix(y_test, y_pred)
```

cm

Output:

```
array([[64,  4],
       [ 3, 29]])
```

Session #5

Ensemble Learning, Grid Search, Learning and Validation Curves

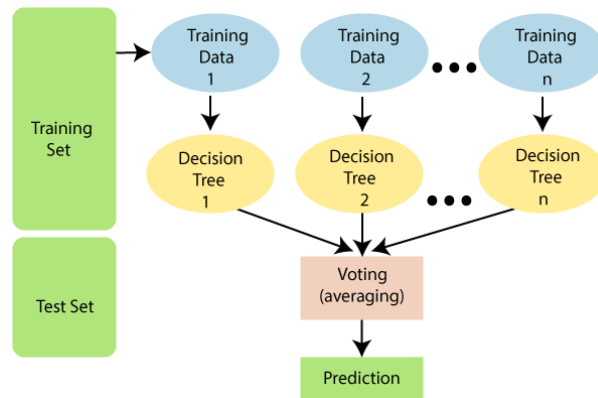
Learning Objective

To implement Ensemble Learning, grid search, learning and validation curves.

Learning Context

Random Forest Classifier

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.



Grid Search

Exhaustive search over specified parameter values for an estimator. `GridSearchCV` implements a "fit" and a "score" method. It also implements "score_samples", "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

Validation Curve

To validate a model we need a scoring function (see *Metrics and scoring*: quantifying the quality of predictions), for example accuracy for classifiers. The proper way of choosing multiple hyperparameters of an estimator is of course grid search or similar methods (see *Tuning the hyper-parameters of an estimator*) that select the hyperparameter with the maximum score on a validation set or multiple validation sets. Note that if we optimize the hyperparameters based on a validation score the validation score is biased and not a good estimate of the generalization any longer. To get a proper estimate of the generalization we have

to compute the score on another test set. However, it is sometimes helpful to plot the influence of a single hyperparameter on the training score and the validation score to find out whether the estimator is overfitting or underfitting for some hyperparameter values.

Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from Preprocessing to Deep Learning", O'REILLY Publisher, 2018
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017
3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Ensemble Learning, Data Clustering & Classification

Solutions

```
-> data=pd.read_csv("Wine.csv")
-> cols=['Alcohol','Color_Intensity','Proline','Ash_Alcanity'] -> from
sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler(feature_range=(0,1))
data[cols]=mms.fit_transform(data[cols])
-> x=data.drop('Customer_Segment',axis=1)
```

```

y=data['Customer_Segment']
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)
->from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
preds=clf.predict(X_test)
->preds

```

Output:

```

array([1, 1, 3, 1, 2, 1, 2, 3, 2, 3, 1, 3, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2,
       2, 3, 3, 3, 2, 2, 2, 1, 1, 2, 3, 1, 1, 1, 3, 3, 2, 3, 1, 2, 2, 2,
       3, 1, 2, 2, 3, 1, 2, 1, 1, 3])

```

```

->from sklearn.metrics import accuracy_score
accuracy_test_DT=accuracy_score(y_test,preds)
train_preds=clf.predict(X_train)
accuracy_train_DT=accuracy_score(y_train,train_preds)
->print('accuracy_train_DT',accuracy_train_DT)
print('accuracy_test_DT',accuracy_test_DT)

```

Output:

```

accuracy_train_DT 1.0
accuracy_test_DT 1.0

```

Grid Search CV

```

-> from sklearn.model_selection import GridSearchCV
grid_param={ 'n_estimators':[100,500,800], 'criterion':['gini','entropy'],
'bootstrap':[True,False]}

```

```

gd_sr=GridSearchCV(estimator=clf,param_grid=grid_param,scoring='accuracy',
cv=5)
gd_sr.fit(X_train,y_train)
best_parameters=gd_sr.best_score_ print(best_parameters)
best_result=gd_sr.best_score_ print(best_result)

```

Output:

```

0.976
0.976

```

Validation Curve

```

import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import validation_curve
df = pd.read_csv("F:\ML\ml datasets\wineQualityReds.csv") # Loading the data
X = df.iloc[:, :-1] # Feature matrix in pd.DataFrame format
y = df.iloc[:, -1] # Target vector in pd.Series format
# Making a Random Forest Classifier object
rf = RandomForestClassifier(n_estimators=100, criterion='gini')
train_score, test_score=validation_curve(rf, X, y,
param_name="max_depth",param_range=np.arange(1, 11), cv=10,
scoring="accuracy")
# Plot the validation curve
print(validation_curve(rf, X, y,
param_name="max_depth",param_range=np.arange(1, 11), cv=10,
scoring="accuracy"))

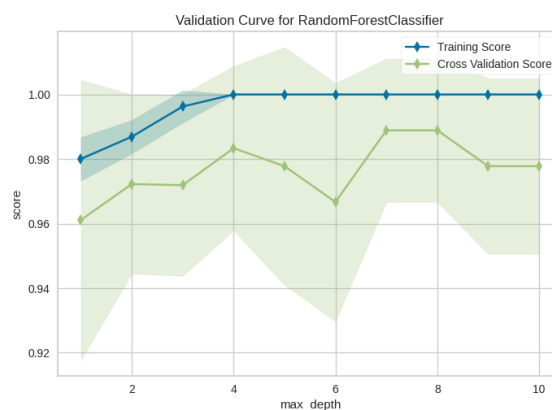
```

#PLOTING

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

param_range=np.arange(1, 11)
mean_train_score = np.mean(train_score, axis = 1)
print(mean_train_score)
mean_test_score = np.mean(test_score, axis = 1)
mtp.plot(param_range, mean_train_score, label = "Training Score", color = 'b')
mtp.plot(param_range, mean_test_score,label = "Cross Validation Score", color
= 'g')
mtp.title("Validation Curve with randomforest Classifier")
mtp.xlabel("max depth")
mtp.ylabel("Accuracy")
mtp.tight_layout()
mtp.legend(loc = 'best')
mtp.show()
```

Output:



Session #6

Data Clustering and Evaluation

Learning Objective

To implement K-Means Clustering and evaluate the clustering model.

Learning Context

K-Means

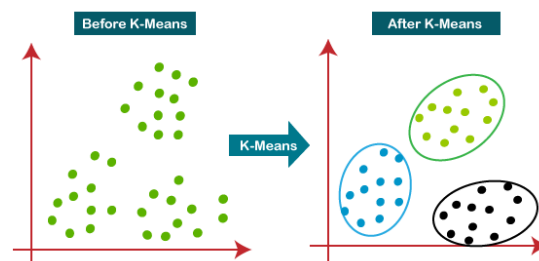
K-means is a data clustering approach for unsupervised machine learning that can separate unlabeled data into a predetermined number of disjoint groups of equal variance - clusters - based on their similarities. It's a popular algorithm thanks to its ease of use and speed on large datasets. K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if $K=2$, there will be two clusters, and for $K=3$, there will be three clusters, and so on. "It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties". It is a

centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters. The k-means clustering algorithm mainly performs two tasks:

1. Determines the best value for K center points or centroids by an iterative process.
2. Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

The below diagram explains the working of the K-means Clustering Algorithm



Predicting optimal clusters is of utmost importance in Cluster Analysis. For a given data, we need to evaluate which Clustering model will best fit the data, or which parameters of a model will give optimal clusters. We often need to compare two clusters or analyze which model would be optimal to deal with outliers. Different performance and evaluation metrics are used to evaluate clustering methods. In this Silhouette Index is one of these evaluation metrics.

Silhouette Index

The Silhouette score is the measure of how similar a data point is to its own cluster as compared to other clusters. A higher Silhouette score value indicates

that the data point is better matched to its own cluster and badly matched to other clusters. The best score value is 1 and -1 is the worst.

- 1: Means clusters are well apart from each other and clearly distinguished.
- 0: Means clusters are indifferent, or we can say that the distance between clusters is not significant.
- -1: Means clusters are assigned in the wrong way.

Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from Preprocessing to Deep Learning", O'REILLY Publisher, 2018
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017
3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Write a program to evaluate clustering model

Solutions

K Means Clustering:

```
-> data=pd.read_csv("/content/drive/MyDrive/ML LAB/Iris.csv")  
-> data.head()
```

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
-> x=data.iloc[:,3:5].values
```

```
-> wcss=[]
```

```
for i in range(1,10):
```

```
    kmeans=KMeans(n_clusters = i, init = 'k-means++', max_iter = 100, n_init =  
    10, random_state = 0).fit(x)
```

```
    wcss.append(kmeans.inertia_)
```

```
plt.plot(range(1, 10), wcss, 'bx-', color='red')
```

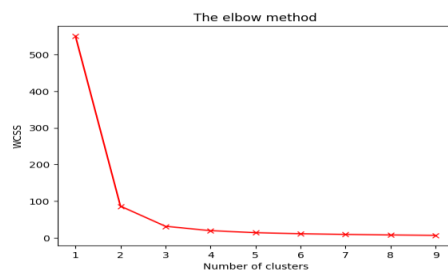
```
plt.title('The elbow method')
```

```
plt.xlabel('Number of clusters')
```

```
plt.ylabel('WCSS')
```

```
plt.show()
```

Output:



```
-> from sklearn.cluster import KMeans
```

```
kmeans=KMeans(n_clusters=3,init='k-means++',random_state=42)
```

```
-> y_predict=kmeans.fit_predict(x)
```

Output:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
       1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,  
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2,  
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0,  
       0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,  
       0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

-> kmeans.cluster_centers_

Output:

```
array([[5.59583333, 2.0375   ],
       [1.464      , 0.244    ],
       [4.26923077, 1.34230769]])
```

```
-> plt.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'red', label = 'Iris-setosa')
```

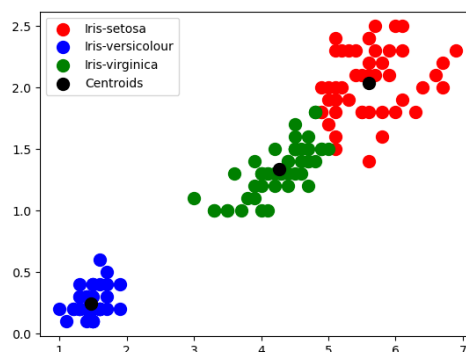
```
plt.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'blue', label =
'Iris-versicolour')
```



```
plt.scatter(x[y_predict == 2, 0], x[y_predict == 2, 1], s = 100, c = 'green', label =
'Iris-virginica')
```

```
plt.scatter(kmeans.cluster_centers_[0], kmeans.cluster_centers_[0], s = 100,  
c = 'black', label = 'Centroids')
```

```
plt.legend()
```

Output:



```
import numpy as np
from sklearn.metrics import silhouette_score
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
features, _ = make_blobs(n_samples=1000, n_features=10, centers=2, cluster_std=
0.5, shuffle=True, random_state=1)
model = KMeans(n_clusters=2, random_state=1).fit(features)
target_predicted = model.labels_
silhouette_score(features, target_predicted)
Output: 0.8916265564072141
```

Session #7

Compressing data via dimensionality reduction


Learning Objective

To implement Compressing data via dimensionality reduction like PCA and LDA

Learning Context

Principal Component Analysis

The Principal Component Analysis is a popular unsupervised learning technique for reducing the dimensionality of data. It increases interpretability yet, at the same time, it minimizes information loss. It helps to find the most significant features in a dataset and makes the data easy for plotting in 2D and 3D. The Principal Component Analysis is a popular unsupervised learning technique for reducing the dimensionality of data. It increases interpretability yet, at the same time, it minimizes information loss. It helps to find the most significant features in a dataset and makes the data easy for plotting in 2D and 3D. PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are image processing, movie recommendation system, optimizing the power allocation in various



communication channels. It is a feature extraction technique, so it contains the important variables and drops the least important variable.

Linear Discriminant Analysis

LDA is a supervised classification technique that is considered a part of crafting competitive machine learning models. This category of dimensionality reduction is used in areas like image recognition and predictive analysis in marketing. Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is also known as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA). This can be used to project the features of higher dimensional space into lower-dimensional space in order to reduce resources and dimensional costs. In this topic, "Linear Discriminant Analysis (LDA) in machine learning", we will discuss the LDA algorithm for classification predictive modeling problems, limitation of logistic regression, representation of linear Discriminant analysis model, how to make a prediction using LDA, how to prepare data for LDA, extensions to LDA and much more. So, let's start with a quick introduction to Linear Discriminant Analysis (LDA) in machine learning.

Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from Preprocessing to Deep Learning", O'REILLY Publisher, 2018
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017

3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Compressing data via dimensionality reduction: PCA, LDA

Solutions

```
-> d = pd.read_csv('/content/drive/MyDrive/ML LAB/wineQualityReds.csv')
d.head()
```

Output:

	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

```
-> x = d.iloc[:, :-1]
```

```
y = d.iloc[:, -1]
```

```
-> from sklearn.preprocessing import MinMaxScaler
```

```
s = MinMaxScaler()
```

```
x = pd.DataFrame(s.fit_transform(x), columns = x.columns)
```

```
-> from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

PCA

```
-> from sklearn.decomposition import PCA
```

```
pca = PCA(n_components = 8)
```

```
X_train = pca.fit_transform(x_train)
```



```
X_test = pca.transform(x_test)
explained_variance = pca.explained_variance_ratio_ print(explained_variance)
```

Output:

```
[0.36050266 0.18884274 0.1509921 0.07257581 0.05621668 0.05012505
0.04092108 0.03433217]
```

```
-> from sklearn.neighbors import KNeighborsClassifier
```

```
KNN_mod = KNeighborsClassifier(n_neighbors=10)
```

```
KNN_mod.fit(X_train,y_train)
```

```
pred = KNN_mod.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix,accuracy_score
```

```
accuracy_score(y_test,pred)*100
```

Output:

```
57.8125
```

LDA

```
-> from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

```
lda = LDA(n_components=5)
```

```
X_train = lda.fit_transform(x_train,y_train)
```

```
X_test = lda.transform(x_test)
```

```
explained_variance = lda.explained_variance_ratio_ print(explained_variance)
```

Output:

```
[0.88278411 0.07919032 0.02560692 0.00748796 0.00493068]
```

```
-> from sklearn.neighbors import KNeighborsClassifier
```

```
KNN_mod = KNeighborsClassifier(n_neighbors=10)
```

```
KNN_mod.fit(X_train,y_train)
```

```
pred = KNN_mod.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score  
accuracy_score(y_test, pred)*100
```

Output:

```
57.8125
```

Session #8

Model Evaluation and Optimization

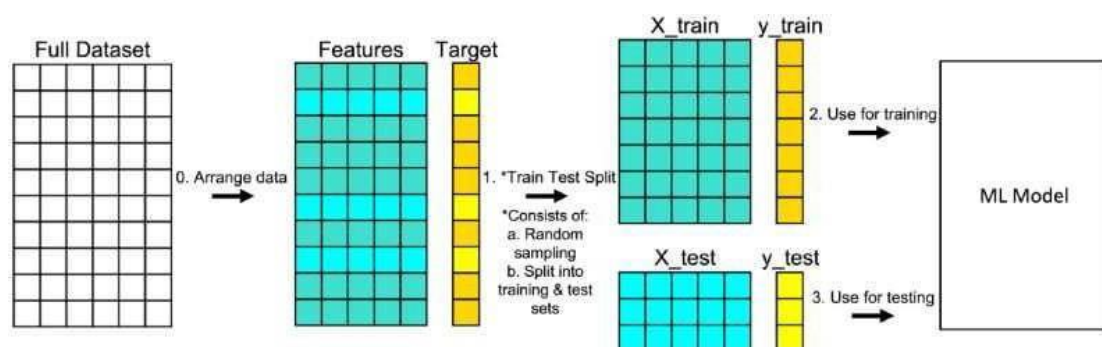
Learning Objective

To implement Model Evaluation and optimization by using K-fold cross-validation.


Learning Context

Splitting of data into train and test data

Data splitting is when data is divided into two or more subsets. Typically, with a two-part split, one part is used to evaluate or test the data and the other to train the model. Data splitting is an important aspect of data science, particularly for creating models based on data. This technique helps ensure the creation of data models and processes that use data models such as machine learning are accurate.



In a basic two-part data split, the training data set is used to train and develop models. Training sets are commonly used to estimate different parameters or to compare different model performance. The testing data set is used after the



training is done. The training and test data are compared to check that the final model works correctly. Scikit-learn alias sklearn is the most useful and robust library for machine learning in Python. The scikit-learn library provides us with the `model_selection` module in which we have the splitter function `train_test_split()`.

Syntax: `train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)`

Parameters:

1. `*arrays`: inputs such as lists, arrays, data frames, or matrices
2. `test_size`: This is a float value whose value ranges between 0.0 and 1.0. It represents the proportion of our test size. its default value is none.
3. `test_size`: This is a float value whose value ranges between 0.0 and 1.0. It represents the proportion of our test size. its default value is none.
4. `random_state`: This parameter is used to control the shuffling applied to the data before applying the split. it acts as a seed.
5. `shuffle`: This parameter is used to shuffle the data before splitting. Its default value is true
6. `stratify`: This parameter is used to split the data in a stratified fashion.

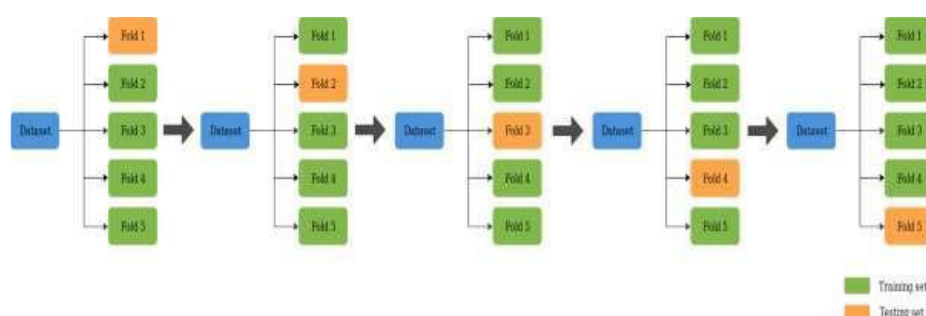
Cross Validation

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. We can also say that it is a technique to check how a statistical model

generalizes to an independent dataset. In machine learning, there is always the need to test the stability of the model. It means based only on the training dataset; we can't fit our model on the training dataset. For this purpose, we reserve a particular sample of the dataset, which was not part of the training dataset. After that, we test our model on that sample before deployment, and this complete process comes under cross-validation. This is something different from the general train-test split.

K-Fold Cross-Validation: K-fold cross-validation approach divides the input dataset into K groups of samples of equal sizes. These samples are called folds. For each learning set, the prediction function uses k-1 folds, and the rest of the folds are used for the test set. This approach is a very popular CV approach because it is easy to understand, and the output is less biased than other methods. The steps for k-fold cross-validation are:

1. Split the input dataset into K groups
2. For each group:
 - a. Take one group as the reserve or test data set.
 - b. Use remaining groups as the training dataset
 - c. Fit the model on the training set and evaluate the performance of the model using the test set.



Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from Preprocessing to Deep Learning", O'REILLY Publisher, 2018
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017
3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Model Evaluation and optimization: K-fold cross-validation, learning and validation curves, grid search

Solutions

1. Importing the libraries

```
->import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

2. Loading the data

```
->data=pd.read_csv("bank.csv")
->data.shape
```

Output:

```
(11162, 17)
```

```
->data
```

Output:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	yes
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	yes
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	yes
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	yes
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	yes
...
11157	33	blue-collar	single	primary	no	1	yes	no	cellular	20	apr	257	1	-1	0	unknown	no
11158	39	services	married	secondary	no	733	no	no	unknown	16	jun	83	4	-1	0	unknown	no
11159	32	technician	single	secondary	no	29	no	no	cellular	19	aug	156	2	-1	0	unknown	no
11160	43	technician	married	secondary	no	0	no	yes	cellular	8	may	9	2	172	5	failure	no
11161	34	technician	married	secondary	no	0	no	no	cellular	9	jul	628	1	-1	0	unknown	no

11162 rows x 17 columns

3. Categorical data Encoding:

->from sklearn.preprocessing import LabelEncoder

encoder= LabelEncoder()

data['target_var'] = encoder.fit_transform(data['target_var'])

->data.head()

Output:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	target_var
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	1
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	1
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	1
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	1
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	1

->data_c=data.select_dtypes(include=['object'])

data_c.columns

data_c.head()

Output:

	job	marital	education	default	housing	loan	contact	month	poutcome
0	admin.	married	secondary	no	yes	no	unknown	may	unknown
1	admin.	married	secondary	no	no	no	unknown	may	unknown
2	technician	married	secondary	no	yes	no	unknown	may	unknown
3	services	married	secondary	no	yes	no	unknown	may	unknown
4	admin.	married	tertiary	no	no	no	unknown	may	unknown

```
→categorical_Attributes=data.select_dtypes(include=['object'])
```

columns

categorical_Attributes

Output:

```
Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',  
      'month', 'poutcome'],  
      dtype='object')
```

```
->data_n=data.select_dtypes(include=['int64'])
```

```
->numerical_Attributes = data.select_dtypes(include=['int64'])
```

columns

numerical_Attributes

Output:

```
Index(['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous'], dtype='object')
```

```
->num_attr=pd.DataFrame(data_n)
```

```
->num_attr
```

Output:

	age	balance	day	duration	campaign	pdays	previous
0	59	2343	5	1042	1	-1	0
1	56	45	5	1467	1	-1	0
2	41	1270	5	1389	1	-1	0
3	55	2476	5	579	1	-1	0
4	54	184	5	673	2	-1	0
...
11157	33	1	20	257	1	-1	0
11158	39	733	16	83	4	-1	0
11159	32	29	19	156	2	-1	0
11160	43	0	8	9	2	172	5
11161	34	0	9	628	1	-1	0

11162 rows × 7 columns

Creating dummy variables for Categorical data

```
->data = pd.get_dummies(columns=categorical_Attributes, data=data,  
prefix=categorical_Attributes, prefix_sep="_")  
print (data.columns, data.shape)
```

Output:

```
Index(['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous',  
      'target_var', 'job_admin.', 'job_blue-collar', 'job_entrepreneur',  
      'job_housemaid', 'job_management', 'job_retired', 'job_self-employed',  
      'job_services', 'job_student', 'job_technician', 'job_unemployed',  
      'job_unknown', 'marital_divorced', 'marital_married', 'marital_single',  
      'education_primary', 'education_secondary', 'education_tertiary',  
      'education_unknown', 'default_no', 'default_yes', 'housing_no',  
      'housing_yes', 'loan_no', 'loan_yes', 'contact_cellular',  
      'contact_telephone', 'contact_unknown', 'month_apr', 'month_aug',  
      'month_dec', 'month_feb', 'month_jan', 'month_jul', 'month_jun',  
      'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep',  
      'poutcome_failure', 'poutcome_other', 'poutcome_success',  
      'poutcome_unknown'],  
      dtype='object') (11162, 52)
```

```
->data.head()
```

Output:

	age	balance	day	duration	campaign	pdays	previous	target_var	job_admin.	job_blue-collar	...	month_jun	month_mar	month_may	month_nov	month_oct
0	59	2343	5	1042	1	-1	0	1	1	0	...	0	0	1	0	0
1	56	45	5	1467	1	-1	0	1	1	0	...	0	0	1	0	0
2	41	1270	5	1389	1	-1	0	1	0	0	...	0	0	1	0	0
3	55	2476	5	579	1	-1	0	1	0	0	...	0	0	1	0	0
4	54	184	5	673	2	-1	0	1	1	0	...	0	0	1	0	0

5 rows x 52 columns

4. Splitting of data:

```
->y=data['target_var'] X=data.drop('target_var',axis=1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=0.25,random_state=123)
```

```
-> print(pd.value_counts(y_train)) print(pd.value_counts(y_test))
```

Output:

```
0    4434
1    3937
Name: target_var, dtype: int64
0    1439
1    1352
Name: target_var, dtype: int64
```

5. Fitting the model

-> from sklearn.naive_bayes import GaussianNB

ModelGNB = GaussianNB()

MM = [ModelGNB]

for models in MM:

models.fit(X_train, y_train)

y_pred = models.predict(X_test)

y_pred_prob = models.predict_proba(X_test)

print('Model Name: ', models)

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

actual = y_test

predicted = y_pred

matrix = confusion_matrix(actual, predicted,
labels=[1,0], sample_weight=None, normalize=None)

print('Confusion matrix : \n', matrix)

C_Report=classification_report(actual, predicted, labels=[1,0])

print('Classification report : \n', C_Report)

Output:

```
Model Name: GaussianNB()
Confusion matrix :
[[ 828  524]
 [ 258 1181]]
Classification report :
              precision    recall  f1-score   support

     1       0.76       0.61       0.68       1352
     0       0.69       0.82       0.75       1439

 accuracy          0.72       2791
 macro avg         0.73       0.72       0.72       2791
weighted avg         0.73       0.72       0.72       2791
```

6. Evaluation metrics:

Confusion matrix:

```
->confusion_matrix_test = confusion_matrix(y_test,y_pred)
```

```
print(confusion_matrix_test)
```

Output:

```
[[1181  258]
 [ 524  828]]
```

```
>Accuracy_Test=(confusion_matrix_test[0,0]+confusion_matrix_test[1,1])/(confusion_matrix_test[0,0]+confusion_matrix_test[0,1]+confusion_matrix_test[1,0]+confusion_matrix_test[1,1])
```

```
TNR_Test=confusion_matrix_test[0,0]/(confusion_matrix_test[0,0]+confusion_matrix_test[0,1])
```

```
TPR_Test=confusion_matrix_test[1,1]/(confusion_matrix_test[1,0]+confusion_matrix_test[1,1]) #Recall
```

```
print("Test TNR: ",TNR_Test)
```

```
print("Test TPR: ",TPR_Test)
```

```
print("Test Accuracy: ",Accuracy_Test)
```

Output:

```
Test TNR:  0.8207088255733148
Test TPR:  0.6124260355029586
Test Accuracy:  0.7198136868505912
```

7.K-fold cross validation:

-> from sklearn import datasets

```
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.model_selection import KFold, cross_val_score clf =GaussianNB()
```

```
k_folds = KFold(n_splits = 3)
```

```
scores = cross_val_score(clf, X, y, cv = k_folds)
```

```
print("Cross Validation Scores: ", scores)
```

```
print("Average CV Score: ", scores.mean())
```

```
print("Number of CV Scores used in Average: ", len(scores))
```

Output:

```
Cross Validation Scores:  [0.5122279  0.76941682 0.82123656]
Average CV Score:  0.7009604261004335
Number of CV Scores used in Average:  3
```

```
>Accuracy_Test=(confusion_matrix_test[0,0]+confusion_matrix_
test[1,1])/(confusion_matrix_test[0,0]+confusion_matrix_test[0,
1]+confusion_matrix_test[1,0]+confusion_matrix_test[1,1])
```

```
TNR_Test=confusion_matrix_test[0,0]/(confusion_matrix_ test[0,0]
+confusion_matrix_test[0,1])
```

```
TPR_Test=confusion_matrix_test[1,1]/(confusion_matrix_ test[1,0]
+confusion_matrix_test[1,1]) #Recall
```

```
print("Test TNR: ",TNR_Test)
```

```
print("Test TPR: ",TPR_Test)
```

```
print("Test Accuracy: ",Accuracy_Test)
```

Output:

```
Test TNR:  0.8207088255733148
Test TPR:  0.6124260355029586
Test Accuracy:  0.7198136868505912
```

Session #8

Regularization

Learning Objective

To implement for reducing the variance of a linear regression model using Lasso and Ridge Regularization evaluation of clustering model

Learning Context

Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it. Sometimes the machine learning model performs well with the training data but does not perform well with the test data. It means the model is not able to predict the output when deals with unseen data by introducing noise in the output, and hence the model is called overfitted. This technique can be used in such a way that it will allow to maintain all variables or features in the model by reducing the magnitude of the variables. Hence, it maintains accuracy as well as a generalization of the model. It mainly regularizes or reduces the coefficient of features toward zero. In simple words, "In regularization technique, we reduce the magnitude of the features by keeping the same number of features."

How does Regularization Work?

Regularization works by adding a penalty or complexity term to the complex model. Let's consider the simple linear regression equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_n x_n + b$$

In the above equation, Y represents the value to be predicted X1, X2, ...Xn are the features for Y.

$\beta_0, \beta_1, \dots, \beta_n$ are the weights or magnitude attached to the features, respectively. Here represents the bias of the model, and b represents the intercept.

Linear regression models try to optimize the β_0 and b to minimize the cost function. The equation for the cost function for the linear model is given below:

$$\sum_{i=1}^M (y_i - y'_i)^2 = \sum_{i=1}^M (y_i - \sum_{j=0}^n \beta_j * X_{ij})^2$$

Techniques of Regularization

There are mainly two types of regularization techniques, which are given below:

1. Ridge Regression
2. Lasso Regression

Ridge Regression

Ridge regression is one of the types of linear regression in which a small amount of bias is introduced so that we can get better long-term predictions. Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called as L2 regularization. In this technique, the cost function is altered by adding the penalty term to it. The amount of bias added to the model is called Ridge Regression penalty. We can calculate it by multiplying with the lambda to the squared weight of each individual feature. The equation for the cost function in ridge regression will be:

$$\sum_{i=1}^M (y_i - y'_i)^2 = \sum_{i=1}^M \left(y_i - \sum_{j=0}^n \beta_j * x_{ij} \right)^2 + \lambda \sum_{j=0}^n \beta_j^2$$

As we can see from the above equation, if the values of λ tend to zero, the equation becomes the cost function of the linear regression model. Hence, for the minimum value of λ , the model will resemble the linear regression mode.

Lasso Regression

Lasso regression is another regularization technique to reduce the complexity of the model. It stands for Least Absolute and Selection Operator. It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square of weights. Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0. It is also called as L1 regularization. The equation for the cost function of Lasso regression will be:

$$\sum_{i=1}^M (y_i - y'_i)^2 = \sum_{i=1}^M \left(y_i - \sum_{j=0}^n \beta_j * x_{ij} \right)^2 + \lambda \sum_{j=0}^n |\beta_j|$$

Ridge regression is mostly used to reduce the overfitting in the model, and it includes all the features present in the model. It reduces the complexity of the model by shrinking the coefficients. Lasso regression helps to reduce the overfitting in the model as well as feature selection.

Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from Preprocessing to Deep Learning", O'REILLY Publisher, 2018
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017
3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Write a program to reduce variance of a linear regression model using Lasso and Ridge Regularization

Solutions

```
from sklearn.linear_model import Lasso
from sklearn.datasets import load_diabetes
from sklearn.preprocessing import StandardScaler
diabetes=load_diabetes()
features = diabetes.data
target = diabetes.target
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)
regression = Lasso(alpha =0.5)
model =regression.fit(features_standardized, target)
```



```
print(model.coef_)
```

Output:

```
[ -0.      -10.28809083  24.98390085  14.66877173 -7.76136877  
 -0.      -8.44971502   3.28432608  24.95304334  2.90702924]
```

```
from sklearn.linear_model import Ridge  
from sklearn.datasets import load_diabetes  
from sklearn.preprocessing import StandardScaler  
boston=load_diabetes()  
features = diabetes.data  
target = diabetes.target  
scaler = StandardScaler()  
features_standardized = scaler.fit_transform(features)  
regression = Ridge(alpha=0.5)  
model = regression.fit(features_standardized, target)  
print(model.coef_)
```

Output:

```
[ -0.45160034 -11.36796054  24.75403875  15.39943346 -33.44143756  
 19.31313255   2.93639178   7.91553777  34.12396693   3.24318124]
```

Session #10

Perceptron for digits

Learning Objective

To Implement Perceptron for digits.

Learning Context

Perceptron

It consists of summation, Sigmoid Function and it takes n no. of inputs and generate output. Frank Rosenblatt (1928 - 1971) was an American psychologist notable in the field of Artificial Intelligence. In 1957 he started something really big. He "invented" a Perceptron program, on an IBM 704 computer at Cornell Aeronautical Laboratory. Scientists had discovered that brain cells (Neurons) receive input from our senses by electrical signals. The Neurons, then again, use electrical signals to store information, and to make decisions based on previous input. Frank had the idea that Perceptrons could simulate brain principles, with the ability to learn and make decisions. The original Perceptron was designed to take a number of binary inputs, and produce one binary output (0 or 1). The idea was to use different weights to represent the importance of each input, and that the sum of the values should be greater than a threshold value before making a decision like yes or no (true or false) (0 or 1).

Frank Rosenblatt suggested this algorithm:

1. Set a threshold value

-
2. Multiply all inputs with its weights
 3. Sum all the results
 4. Activate the output

Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from Preprocessing to Deep Learning", O'REILLY Publisher, 2018
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017
3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Write a program to implement Perceptron for digits

Solutions

```
from sklearn.datasets import load_digits
from sklearn.linear_model import Perceptron
X, y = load_digits(return_X_y=True)
clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(X, y)
Perceptron()
clf.score(X, y)
Output: 0.9510294936004452
```

Session #11

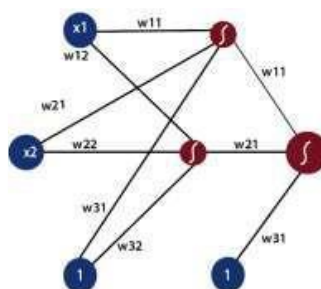
Feed-Forward Network

Learning Objective

To Implement Feed-Forward Network for wheat seeds dataset.

Learning Context

We are still making use of a gradient descent optimization algorithm which acts to minimize the error of our model by iteratively moving in the direction with the steepest descent, the direction which updates the parameters of our model while ensuring the minimal error. It is important to recognize the subsequent training of our neural network. Recognition is done by dividing our data samples through some decision boundary". The process of receiving an input to produce some kind of output to make some kind of prediction is known as Feed Forward." Feed Forward neural network is the core of many other important neural networks such as convolution neural network. In the feed-forward neural network, there are not any feedback loops or connections in the network. Here is simply an input layer, a hidden layer, and an output layer. If every model in every single layer. We will talk more about optimization algorithms and backpropagation later.



Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from Preprocessing to Deep Learning", O'REILLY Publisher, 2018
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017
3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Write a program to implement Feed-Forward Network for wheat seeds datas.

Solutions

```
-> df=pd.read_csv("wheat-seed.csv",index_col=None)
print(df.shape)
-> df.dtypes
print(df.head())
print()
print()
print(df.isnull().sum())
```

Output:

	area	perimeter	compactness	length	width	asymmetry	groove	wheat_type
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1

```
area          0
perimeter     0
compactness   0
length        0
width         0
asymmetry     0
groove        0
wheat_type    0
dtype: int64
```

```
-> X = df.iloc[:, 0:7].values
```

```
y = df.iloc[:, 7].values
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
```

```
feature_scaler = StandardScaler()
```

```
X_train = feature_scaler.fit_transform(X_train)
```

```
X_test = feature_scaler.transform(X_test)
```

```
-> from sklearn.preprocessing import LabelBinarizer
```

```
lb=LabelBinarizer()
```

```
y_train = lb.fit_transform(y_train)
```

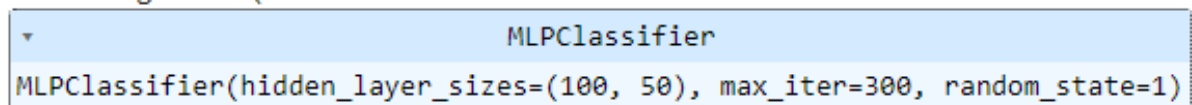
```
y_test=lb.fit_transform(y_test)
```

```
-> from sklearn.neural_network import MLPClassifier
```

```
clf = MLPClassifier(hidden_layer_sizes=(100,50), max_iter=300, activation =
'relu', solver='adam', random_state=1)
```

`clf.fit(X_train,y_train)`

Output:



```
MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=300, random_state=1)
```

`-> ypred=clf.predict(X_test)`

`from sklearn.metrics import accuracy_score`

`accuracy_score(y_test,ypred)`

Output:

`0.9047619047619048`

Session #12

Neural Network for Regression

Learning Objective

To Implement a neural network for regression.

Learning Context

Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It not only supports Convolutional Networks and Recurrent Networks individually but also their combination. It cannot handle low-level computations, so it makes use of the Backend library to resolve it. The backend library act as a high-level API wrapper for the low-level API, which lets it run on TensorFlow, CNTK, or Theano.

Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from Preprocessing to Deep Learning", O'REILLY Publisher, 2018
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017

-
3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Write a program to implement a neural network for regression.

Solutions

```
#Load libraries
import numpy as np
from keras.preprocessing.text import Tokenizer
from keras import models
from keras import layers
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
# Set random seed
np.random.seed(0)
# Generate features matrix and target vector
features, target = make_regression(n_samples = 10000,
n_features = 3,
n_informative = 3,
n_targets = 1,
noise = 0.0,
random_state = 0)
```

```

# Divide our data into training and test sets
features_train, features_test, target_train, target_test = train_test_split(
    features, target, test_size=0.33, random_state=0)

# Start neural network
network = models.Sequential()

# Add fully connected layer with a ReLU activation function
network.add(layers.Dense(units=32, activation="relu", input_shape=(features_train.shape[1],)))

# Add fully connected layer with a ReLU activation function
network.add(layers.Dense(units=32, activation="relu"))

# Add fully connected layer with no activation function
network.add(layers.Dense(units=1))

# Compile neural network
network.compile(loss="mse", # Mean squared error
                optimizer="RMSprop", # Optimization algorithm
                metrics=["mse"]) # Mean squared error

# Train neural network
history = network.fit(features_train, # Features
                      target_train, # Target vector
                      epochs=10, # Number of epochs
                      verbose=0, # No output
                      batch_size=100, # Number of observations per batch
                      validation_data=(features_test, target_test))

predicted_target = network.predict(features_test)

```

```
print(predicted_target)
```

Output:

```
104/104 [=====] - 0s 833us/step
```

```
[[ 88.99878]
```

```
[-86.64332]
```

```
[-196.5548 ]
```

```
...
```

```
[-184.40088]
```

```
[-49.22299]
```

```
[ 116.30347]]
```

```
import numpy as np
```

```
from sklearn.metrics import r2_score
```

```
print("RMS: %r " % np.sqrt(np.mean((predicted_target - target_test) ** 2)))
```

```
print("R2= ", r2_score(predicted_target, target_test))
```

```
# print(r2_score(features_train, target_train))
```

Output:

```
RMS: 190.10593971919286
```

```
R2= 0.9830793172486822
```

Session #13

Machine Learning Model

Learning Objective

To Implement a save and load trained machine learning model.

Learning Context

JOBLIB

Joblib is a python library for running computationally intensive tasks in parallel. It provides a set of operations in parallel on large datasets and for caching the results of the computationally expensive functions. Joblib is a set of tools to provide lightweight pipelining in Python. "If you are running heavy grid search cross validation or other forms of hypertuning.(Sklearn+Joblib). Joblib is part of the SciPy ecosystem and provides utilities for pipelining Python jobs. It provides utilities for saving and loading Python objects that make use of NumPy data structures, efficiently.

Saving and Loading

In machine learning, while working with scikit learn library, we need to save the trained models in a file and restore them in order to reuse them to compare the model with other models, and to test the model on new data. The saving of data is called Serialization, while restoring the data is called Deserialization. Also, we deal with different types and sizes of data. Some datasets are easily trained i.e.

they take less time to train but the datasets whose size is large (more than 1GB) can take a very large time to train on a local machine even with GPU. When we need the same trained data in some different project or later sometime, to avoid the wastage of the training time, store the trained model so that it can be used anytime in the future.

Materials & Resources

1. Chris Albon, "Machine Learning with Python Cookbook-practical solutions from Preprocessing to Deep Learning", O'REILLY Publisher, 2018
2. Sebastian Raschka & Vahid Mirjalili, "Python Machine Learning", Packt Publisher, 2017
3. Ian Good Fellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2017.

Exercise

Write a program to save and load a trained machine learning model.

Solutions

```
-> from sklearn import datasets  
  
x,y = datasets.load_iris(return_X_y=True)  
  
-> from sklearn.preprocessing import MinMaxScaler  
  
s = MinMaxScaler()
```

```
x = pd.DataFrame(s.fit_transform(x))
```

```
x.head()
```

Output:

	0	1	2	3
0	0.222222	0.625000	0.067797	0.041667
1	0.166667	0.416667	0.067797	0.041667
2	0.111111	0.500000	0.050847	0.041667
3	0.083333	0.458333	0.084746	0.041667
4	0.194444	0.666667	0.067797	0.041667

```
-> from sklearn.neighbors import KNeighborsClassifier as KNN
```

```
model = KNN(n_neighbors=10)
```

```
model.fit(x,y)
```

Output:

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=10)
```

```
-> import pickle
```

```
pickle.dump(model, open("model.pkl", 'wb'))
```

```
-> pickled_model = pickle.load(open('model.pkl', 'rb'))
```

```
pickled_model.predict(x.tail())
```

Output:

```
array([2, 2, 2, 2, 2])
```



Department of Computer Science and Engineering
SRKR Engineering College (A), Bhimavaram, India