

**Department of Computer Science &
Engineering**

Lab Manual

Subject Name:

Machine Learning Lab

Subject Code:

21BTCS014

Class: - T.E.

Branch: - CSE

Prepared By: - Nagesh Jadhav

Year: - Third Year-Computer

Contributors: - Nagesh Jadhav

Examinations: - CA (40) and Final Exam (60)

**Required H/W and S/W: - 64 bit processor based machine
and Anaconda Navigator, Python**

ASSIGNMENT 1

Problem Statement-

- A. Installation and Configuration of machine learning environment with Anaconda on windows or Ubuntu (Jupyter notebook, spyder, vscode, pycharm)
- B. Installation, Configuration and checking the current Version of numpy, scipy, scikit, pandas and matplotlib lib using anaconda prompt and list their uses in machine learning.

Objective

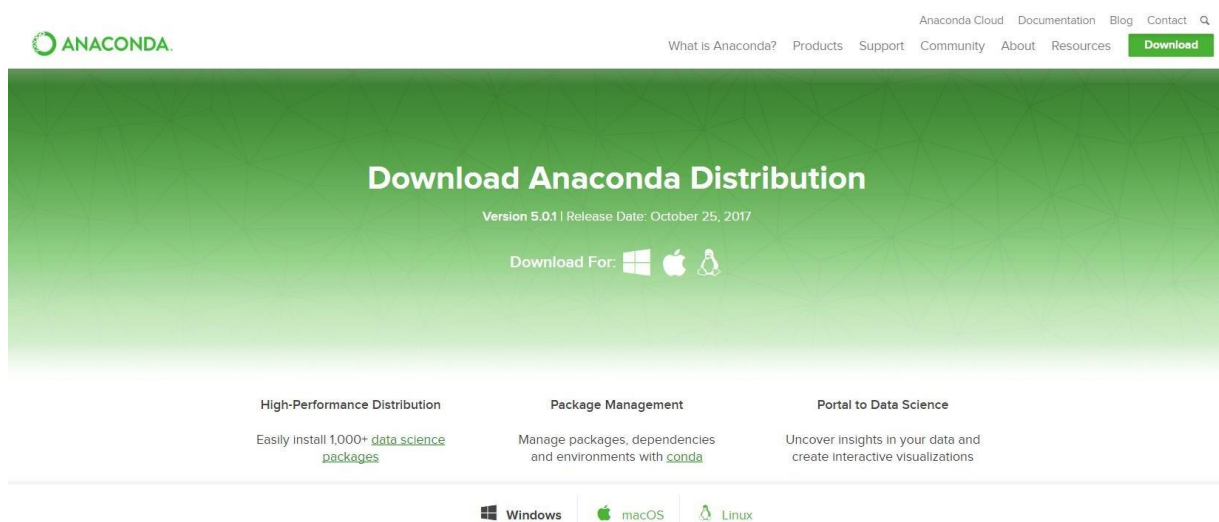
To install anaconda and configure it with Python

Theory

Installation Of Anaconda in Windows

1. Visit the Anaconda downloads page

Go to the following link: [Anaconda.com/downloads](https://anaconda.com/downloads)



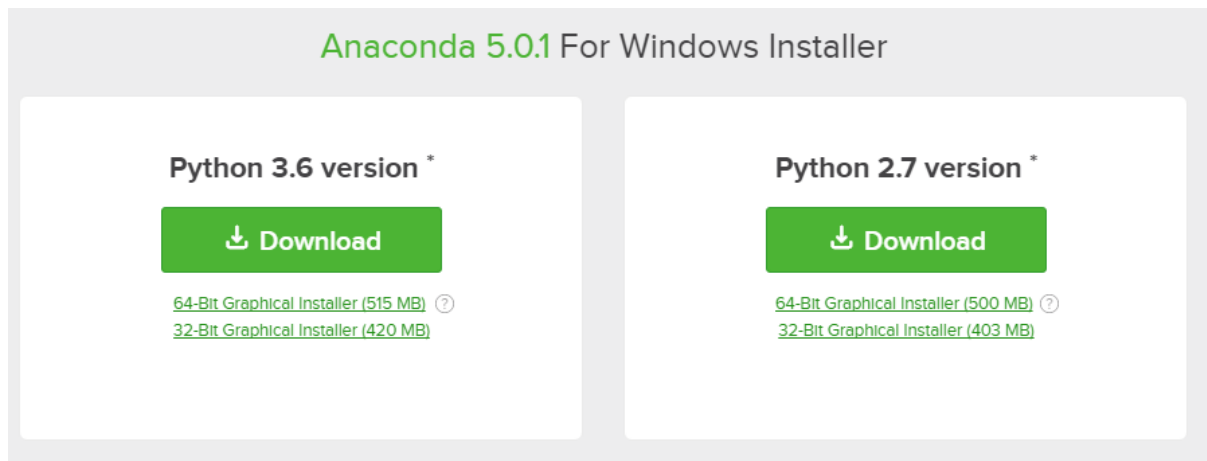
2. Select Windows

Select Windows where the three operating systems are listed.



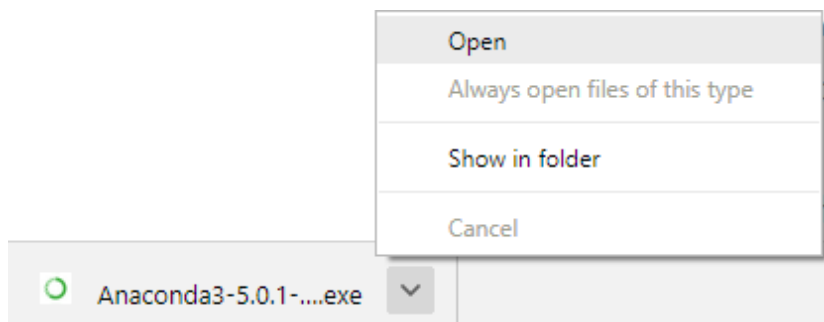
3. Download

Download the most recent Python 3 release. At the time of writing, the most recent release was the Python 3.6 Version. Python 2.7 is legacy Python. For problem solvers, select the Python 3.6 version. If you are unsure if your computer is running a 64-bit or 32-bit version of Windows, select 64-bit as 64-bit Windows is most common.



4. Open and run the installer

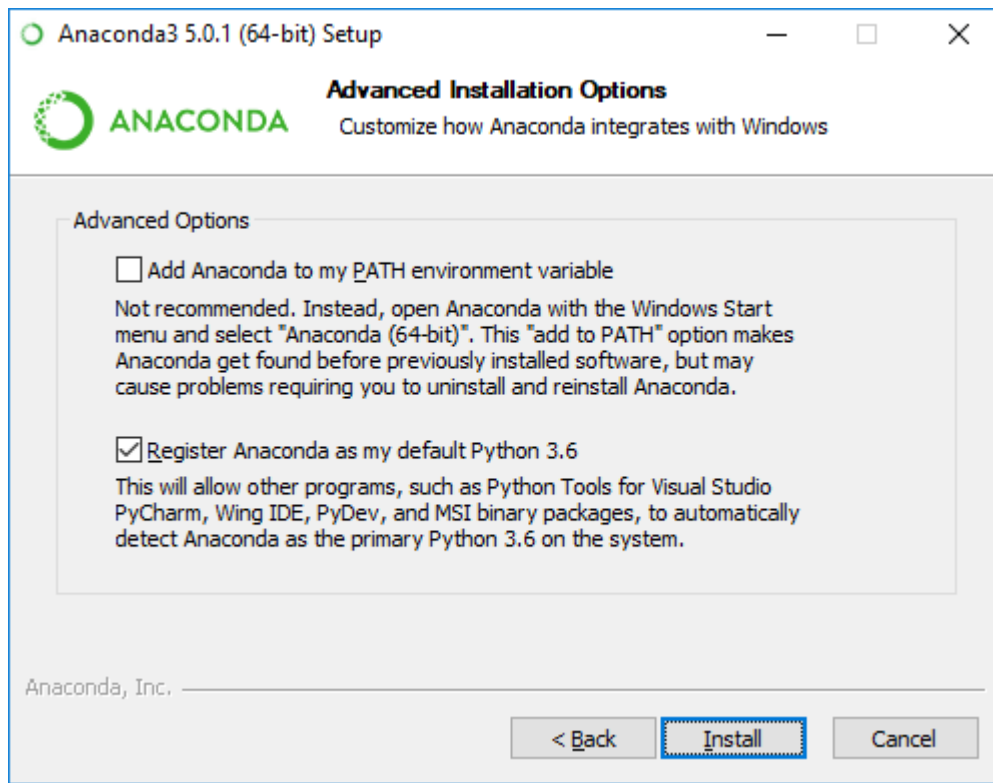
Once the download completes, open and run the *.exe* installer



At the beginning of the install, you need to click **Next** to confirm the installation.



At the Advanced Installation Options screen, I recommend that you **do not check** "Add Anaconda to my PATH environment variable"



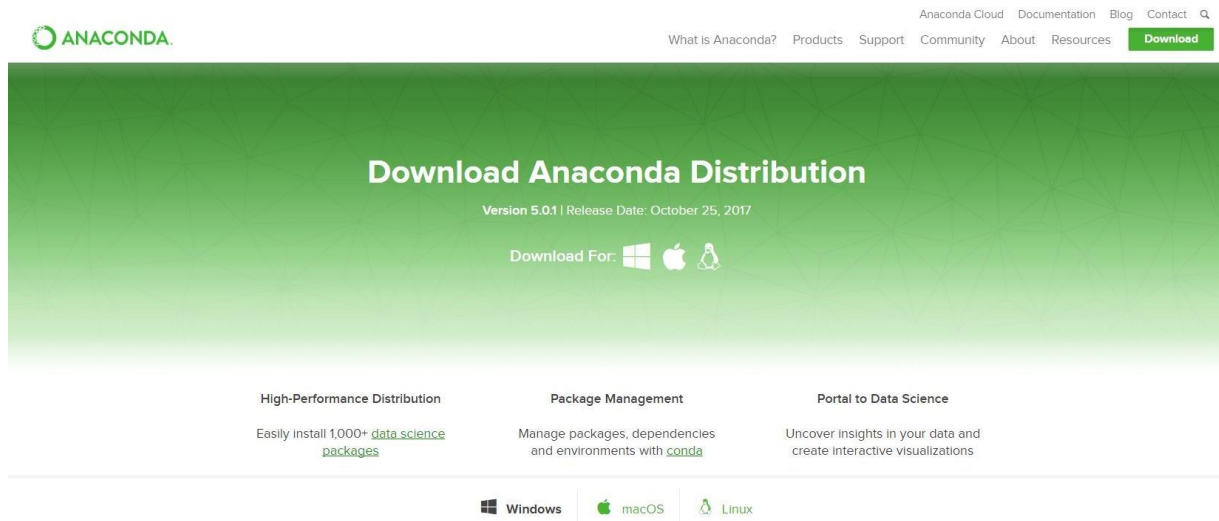
Installation Of Anaconda in Ubuntu

This section details the installation of the Anaconda distribution of Python on Linux, specifically Ubuntu 18.04, but the instructions should work for other Debian-based Linux distributions as well.

Ubuntu 18.04 comes pre-installed with Python (Version 3.6) and legacy Python (Version 2.7). You can confirm the legacy version of Python is installed by opening up a terminal.

1. Visit the Anaconda downloads page

Go to the following link: [Anaconda.com/downloads](https://anaconda.com/downloads)



2. Select Linux

On the downloads page, select the Linux operating system.



3. Copy the bash (.sh file) installer link

In the **Python 3.6 Version*** box, right-click on the [64-Bit(x86) Installer] link. Select [copy link address].



4. Use wget to download the bash installer

Now that the bash installer (.sh file) link is stored on the clipboard, use wget to download the installer script. In a terminal, cd into the home directory and make a new directory called tmp. cd into tmp and use wget to download the installer. Although the installer is a bash script, it is still quite large and the download will not be immediate (Note the link below includes <release>. the specific release depends on when you download the installer)

```
$ cd ~  
$ mkdir tmp  
$ cd tmp  
$ https://repo.continuum.io/archive/Anaconda3<release>.sh
```

5. Run the bash script to install **Anaconda3**

With the bash installer script downloaded, run the **.sh** script to install **Anaconda3**. Ensure you are in the directory where the installer script downloaded:

```
$ ls
```

Anaconda3-5.2.0-Linux-x86_64.sh Run
the installer script with bash.

```
$ bash Anaconda3-5.2.0-Linux-x86_64.sh
```

Accept the Licence Agreement and allow Anaconda to be added to your PATH. By adding Anaconda to your PATH, the Anaconda distribution of Python will be called when you type \$ python in a terminal.

6. source the .bashrc file to add Anaconda to your PATH

Now that **Anaconda3** is installed and **Anaconda3** is added to our PATH, source the .bashrc file to load the new PATH environment variable into the current terminal session. Note the .bashrc file is in the home directory. You can see it with \$ ls -a.

```
$ cd ~
```

```
$ source .bashrc
```

Installation Of NumPy Command –
pip3 install numpy To check the
version

Import numpy as np

Installation of Scipy-stack Command –
pip3 install scipy-stack

NUMPY/SCIPY:-

NumPy is a Python library, which stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object, provide tools for integrating C, C++ etc.

- NumPy array can also be used as an efficient multi-dimensional container for generic .data.
- The ndarray (NumPy Array) is a multidimensional array used to store values of same datatype. These arrays are indexed just like Sequences, starts with zero.
- The ndarrays are better than regular arrays in terms of faster computation and ease of manipulation.
- In different algorithms of Machine Learning like K-means Clustering, Random Forest etc. we have to store the values in an array. So, instead of using regular array, ndarray helps us to manipulate and execute easily.

Installation of scikit

Command `pip3 install -u scikit-learn`

SCIKIT:-

The functionality that scikit-learn provides include:

- **Regression**, including Linear and Logistic Regression
- **Classification**, including K-Nearest Neighbors
- **Clustering**, including K-Means and K-Means++
- **Model selection**
- **Preprocessing**, including Min-Max Normalization.

Installation of Pandas

Command- pip3 install pandas

PANDAS:-

- Merging and Joining Data Sets.
- Reshaping & pivoting Data Sets.
- Inserting & deleting columns in Data Structure.
- Aligning data & dealing with missing data.
- Iterating over a Data set.
- Analyzing Time Series.
- Filtering Data around a condition.
- Arranging Data in an ascending & descending.
- Reading from files with CSV, TXT, XLSX, other formats.
- Manipulating Data using integrated indexing for DataFrame objects.
- Generating Data range, date shifting, lagging, converting frequency, and other other Time Series functionality.
- Subsetting fancy indexing, & label based slicing Data Sets that are large in size.
- Performing split apply combine on Data Sets using the group by engine. With Python Pandas, it is easier to clean & wrangle with your Data. features of Pandas make it a great choice for Data Science and Analysis.

Installation of Matplotlib Command-
pip3 install matplotlib

MATPLOTLIB:-

- Matplotlib is a visualization library in Python for 2D plots of arrays. It consists of several plots like line, bar, scatter, histogram etc.
- Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It can also be used with graphics toolkits like PyQt and wxPython.
- One of the advantages of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals.

Conclusion

Thus we have successfully installed Anaconda on windows and Ubuntu; We also have described the libraries and the installation of the libraries.

Assignment 2

Aim: Download any dataset from UCI or Data.org or from any data repositories and perform the basic data pre-processing steps using Python/R

Objectives:

1. Learn to pre-process dataset
2. Learn to use pandas and sklearn

Theory

Basic steps

Step 1 : Import the libraries

Step 2 : Import the data-set

Step 3 : Check out the missing values

Step 4 : See the Categorical Values

Step 5 : Splitting the data-set into Training and Test Set

Data cleaning:

The main aim of Data Cleaning is to identify and remove errors & duplicate data, in order to create a reliable dataset. This improves the quality of the training data for analytics and enables accurate decision-making.

Needless to say, data cleansing is a time-consuming process and most data scientists spend an enormous amount of time in enhancing the quality of the data. However, there are various methods to identify and classify data for data cleansing.

There are mainly two distinct techniques, namely Qualitative and Quantitative techniques to classify data errors. Qualitative techniques involve rules, constraints, and patterns to identify errors.

On the other hand, Quantitative techniques employ statistical techniques to identify errors in the trained data.

Normalization:

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Here's the formula for normalization:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Here, Xmax and Xmin are the maximum and the minimum values of the feature respectively.

- When the value of X is the minimum value in the column, the numerator will be 0, and hence X' is 0
- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X' is 1
- If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1

Standardization:

Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Here's the formula for standardization:

$$X' = \frac{X - \mu}{\sigma}$$

μ is the mean of the feature values and σ is the standard deviation of the feature values. Note that in this case, the values are not restricted to a particular range.

```
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2,random_state=0)
//Split arrays or matrices into random train and test subsets Quick
utility that wraps input validation
and next(ShuffleSplit().split(X, y)) and application to input data into a single call for
splitting (and optionally subsampling) data in a oneliner.
```

```
Imputer(missing_values='NaN', strategy='mean', axis=0)
Imputation transformer for completing missing values.
```

```
pandas.read_csv()
Read a comma-separated values (csv) file into DataFrame.
Also supports optionally iterating or breaking of the file into chunks.
```

Program:

```
//Import the libraries
```

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

```
Importing the dataset
dataset=pd.read_csv('data.csv')
X=dataset.iloc[:, :-1].values
y=dataset.iloc[:, 3].values
```

```
//Checking out Missing data
```

```
From sklearn.preprocessing import Imputer
imputer=Imputer(missing_values='NaN', strategy='mean', axis=0)
imputer=imputer.fit(X[:, 1:3])
X[:, 1:3]=imputer.transform(X[:, 1:3])
```

```
//Checking the categorical values
from sklearn.preprocessing import LabelEncoder
labelencoder_X=LabelEncoder()
X[:, 0]=labelencoder_X.fit_transform(X[:, 0])
labelencoder_y=LabelEncoder()
y=labelencoder_y.fit_transform(y)
```

```
Splitting the dataset into Training Set and Testing Set from
sklearn.cross_validation import train_test_split
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2,random_state=0)
```

Output:

```
[1] ▶ MI
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

[2] ▶ MI
dataset=pd.read_csv('data.csv')
X=dataset.iloc[:, :-1].values
y=dataset.iloc[:, 3].values

[3] ▶ MI
from sklearn.preprocessing import Imputer
imputer=Imputer(missing_values='NaN', strategy='mean', axis=0)
imputer=imputer.fit(X[:, 1:3])
X[:, 1:3]=imputer.transform(X[:, 1:3])

[4] ▶ MI
from sklearn.preprocessing import LabelEncoder
labelencoder_X=LabelEncoder()
X[:, 0]=labelencoder_X.fit_transform(X[:, 0])
labelencoder_y=LabelEncoder()
y=labelencoder_y.fit_transform(y)

[7] ▶ MI
from sklearn.cross_validation import train_test_split
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2,random_state=0,)

[8] ▶ MI
X_train.shape

(8, 3)

[9] ▶ MI
y_train.shape

(8, )
```

Conclusion:

Thus we have successfully implemented pre-processing operations on a dataset

Assignment 3

Aim: Download the any dataset from UCI or Data.org or from any other data repositories and Implement artificial neural network on a dataset.

A. Classifier using ANN

B. Regressor using ANN

Display the loss graph for the same. Also generate classification report in terms of confusion matrix, precision, recall, and f1 score.

Objectives:

1. To learn about classification and regression
2. To learn ANN algorithm
3. To demonstrate and analyze the results

Theory:

Artificial Neural Network Tutorial provides basic and advanced concepts of ANNs.

The term "Artificial neural network" refers to a biologically inspired sub-field of artificial intelligence modeled after the brain. An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain. Similar to a human brain has neurons interconnected to each other, artificial neural networks also have neurons that are linked to each other in various layers of the networks. These neurons are known as nodes.

Artificial Neural Network primarily consists of three layers

Input Layer:

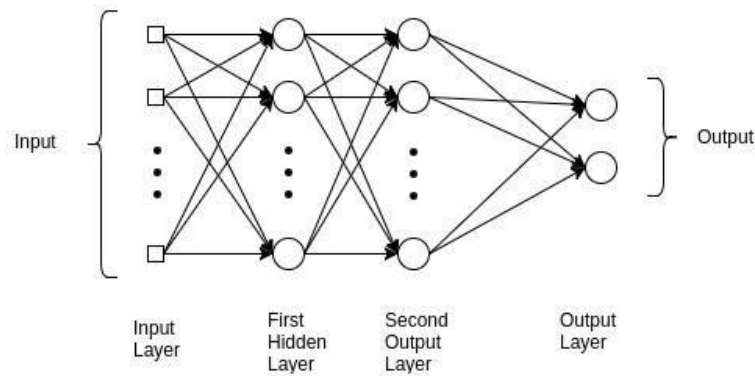
As the name suggests, it accepts inputs in several different formats provided by the programmer.

Hidden Layer:

The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

Output Layer:

The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.



Regression:

A regression problem is when the output variable is a real or continuous value, such as “salary” or “weight”. Many different models can be used, the simplest is the linear regression. It tries to fit data with the best hyper-plane which goes through the points.

Classification:

A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”. A classification model attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes. For example, when filtering emails “spam” or “not spam”, when looking at transaction data, “fraudulent”, or “authorized”.

Multilayer Perceptron:

In the Multilayer perceptron, there can be more than one linear layer (combination of **neurons**). If we take the simple example of the three-layer network, the first layer will be the *input layer* and the last will be the *output layer* and the middle layer will be called *hidden layer*. We feed our input data into the input layer and take the output from the output layer. We can increase the number of the hidden layer as much as we want, to make the model more complex according to our task.

BackPropagation Algorithm:

The algorithm is used to effectively train a neural network through a method called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model’s parameters (weights and biases).

In other words, backpropagation aims to minimize the cost function by adjusting the network’s weights and biases. The level of adjustment is determined by the gradients of the cost function with respect to those parameters.

Activation function

Activation functions, also known as non-linearity, describe the input-output relations in a non-linear way. This gives the model power to be more flexible in describing arbitrary relations. Here are some popular activation functions: Sigmoid, Relu, and TanH. I will describe these in my next blog.

Precision - Precision is the ratio of correctly predicted positive observations to the total

predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. The question recall answers is: Of all the passengers that truly survived, how many did we label? We have got recall of 0.631 which is good for this model as it's above 0.5.

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall. In our case, F1 score is 0.701.

$$\text{F1 Score} = 2 * (\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$$

Confusion matrix

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

	<i>Class 1 Predicted</i>	<i>Class 2 Predicted</i>
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

Code:

1) Program: Classifier using ANN

Titanic Survival Prediction using ANN Classifier

Note- You can download dataset from “ https://www.kaggle.com/jamesleslie/titanic-neural-network-for-beginners/data?select=train_clean.csv. “

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.pyplot import rcParams%matplotlib inline
```

```

rcParams['figure.figsize'] = 10,8
sns.set(style='whitegrid', palette='muted',
        rc={'figure.figsize': (15,10)})
import os
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from numpy.random import seed
from tensorflow import set_random_seed
train = pd.read_csv('./train_clean.csv', )
test = pd.read_csv('./test_clean.csv')
df = pd.concat([train, test], axis=0, sort=True)
df.head()

df['Sex'] = df['Sex'].astype('category')
# convert to category codes
df['Sex'] = df['Sex'].cat.codes

categorical = ['Embarked', 'Title']
for var in categorical:
    df = pd.concat([df,
                    pd.get_dummies(df[var], prefix=var)], axis=1)
    del df[var]
# drop the variables we won't be using
df.drop(['Cabin', 'Name', 'Ticket', 'PassengerId'], axis=1, inplace=True)
df.head()

continuous = ['Age', 'Fare', 'Parch', 'Pclass', 'SibSp', 'Family_Size']
scaler = StandardScaler()
for var in continuous:
    df[var] = df[var].astype('float64')
    df[var] = scaler.fit_transform(df[var].values.reshape(-1, 1))

X_train = df[pd.notnull(df['Survived'])].drop(['Survived'], axis=1)
y_train = df[pd.notnull(df['Survived'])]['Survived']
X_test = df[pd.isnull(df['Survived'])].drop(['Survived'], axis=1)

lyrs=[8]
act='linear'
opt='Adam'
dr=0.0
# set random seed for reproducibility
seed(42)
set_random_seed(42)
model = Sequential()
# create first hidden layer
model.add(Dense(lyrs[0], input_dim=X_train.shape[1], activation=act))
# create additional hidden layers
for i in range(1,len(lyrs)):
    model.add(Dense(lyrs[i], activation=act))
# add dropout, default is none

```

```

model.add(Dropout(dr))
# create output layer
model.add(Dense(1, activation='sigmoid')) # output layer
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
model = create_model()
print(model.summary())

# train model on full train set, with 80/20 CV split
training = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=0)
val_acc = np.mean(training.history['val_acc'])
print("n%s: %.2f%%" % ('val_acc', val_acc*100))
# summarize history for accuracy
plt.plot(training.history['acc'])
plt.plot(training.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# calculate predictions
test['Survived'] = model.predict(X_test)
test['Survived'] = test['Survived'].apply(lambda x: round(x,0)).astype('int')
solution = test[['PassengerId', 'Survived']]
print(solution)

```

B.Regressor using ANN

Note:- Dataset is available at The data is available in the Colab in the path
/content/sample_data/california_housing_train.csv

```

import math
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import Model
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from tensorflow.keras.losses import MeanSquaredLogarithmicError
TRAIN_DATA_PATH = '/content/sample_data/california_housing_train.csv'
TEST_DATA_PATH = '/content/sample_data/california_housing_test.csv'
TARGET_NAME = 'median_house_value'

```

The training data is in the path /content/sample_data/california_housing_train.csv and the test data is in the path /content/sample_data/california_housing_test.csv in Google Colab. The target name in the data is median_house_value.

```

train_data = pd.read_csv(TRAIN_DATA_PATH)

test_data = pd.read_csv(TEST_DATA_PATH)

x_train, y_train = train_data.drop(TARGET_NAME, axis=1), train_data[TARGET_NAME]

x_test, y_test = test_data.drop(TARGET_NAME, axis=1), test_data[TARGET_NAME]

```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250

Standard Scale test and train data

Z - Score normalization

```

"""

```

```

standard_scaler = StandardScaler()

x_train_scaled = pd.DataFrame(
    standard_scaler.fit_transform(x_train),
    columns=x_train.columns
)

x_test_scaled = pd.DataFrame(
    standard_scaler.transform(x_test),
    columns = x_test.columns
)

return x_train_scaled, x_test_scaled

x_train_scaled, x_test_scaled = scale_datasets(x_train, x_test)

hidden_units1 = 160

hidden_units2 = 480

hidden_units3 = 256

learning_rate = 0.01

# Creating model using the Sequential in tensorflow

```

```

def build_model_using_sequential():

    model = Sequential([

        Dense(hidden_units1, kernel_initializer='normal', activation='relu'),

        Dropout(0.2),

        Dense(hidden_units2, kernel_initializer='normal', activation='relu'),

        Dropout(0.2),

        Dense(hidden_units3, kernel_initializer='normal', activation='relu'),

        Dense(1, kernel_initializer='normal', activation='linear')

    ])

    return model

# build the model

model = build_model_using_sequential()

# loss function

msle = MeanSquaredLogarithmicError()

model.compile(

    loss=msle,

    optimizer=Adam(learning_rate=learning_rate),

    metrics=[msle]

)

# train the model

history = model.fit(

    x_train_scaled.values,

    y_train.values,

    epochs=10,

    batch_size=64,

    validation_split=0.2

)

def plot_history(history, key):

    plt.plot(history.history[key])

```

```
plt.plot(history.history['val_'+key])

plt.xlabel("Epochs")

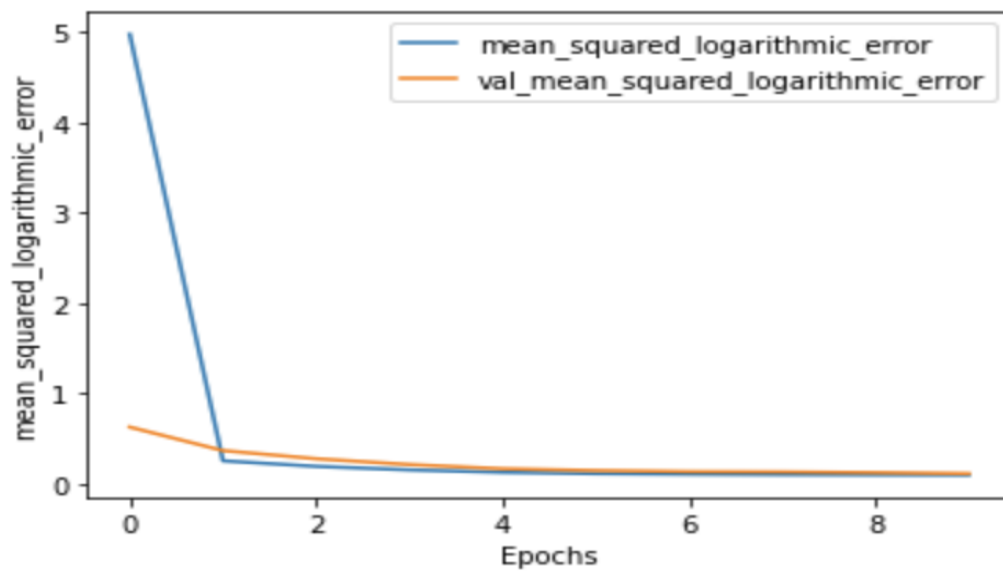
plt.ylabel(key)

plt.legend([key, 'val_'+key])

plt.show()

# Plot the history

plot_history(history, 'mean_squared_logarithmic_error')
```



```
x_test['prediction'] = model.predict(x_test_scaled)
```

Conclusion:

Thus we have successfully completed the implementation of Artificial Neural network algorithm

Assignment 4

Aim: Develop a Bayesian classifier for given dataset

Objectives

1. To learn bayes theorem
2. To implement Bayesian classifier

Theory

Bayes Theorem

Bayes' Theorem is a way of finding a probability when we know certain other probabilities.

The formula is:

$$P(A|B) = \frac{P(A) P(B|A)}{P(B)}$$

Which tells us: how often A happens *given that B happens*, written $P(A|B)$,

When we know: how often B happens *given that A happens*, written $P(B|A)$
and how likely A is on its own, written $P(A)$
and how likely B is on its own, written $P(B)$

Bayes Classifier with example

In machine learning, **naïve Bayes classifiers** are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. They are among the simplest Bayesian network models.^[1] But they could be coupled with Kernel density estimation and achieve higher accuracy levels.

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a

particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

Code

```
import pandas as pd
from sklearn.naive_bayes import GaussianNB
df_=pd.read_csv('IRIS.csv')
df_=df_.dropna(axis=1,how='any')
from sklearn.preprocessing import StandardScaler from
sklearn.model_selection import train_test_split from
sklearn.neural_network import MLPClassifier from
sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(["Iris-setosa", "Iris-versicolor", "Iris-viginica"]) le.transform(["Iris-
setosa", "Iris-versicolor", "Iris-viginica"])

x=df_.iloc[:,0:3]

obj=StandardScaler()
x_=obj.fit_transform(x)
y=df_['species']
X_train, X_test, y_train, y_test = train_test_split(x_, y, test_size=0.4,random_state=42)
X_validate, X_test1, y_validate, y_test1=train_test_split(X_test,y_test,test_s
ize=0.5,random_state=42)
bayes=GaussianNB(priors=None)
bayes.fit(X_train,y_train)
print(bayes.score(X_validate,y_validate))
print(bayes.score(X_test1,y_test1))
```

Iris dataset

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Training

```
bayes.fit(X_train,y_train)

GaussianNB(priors=None)
```

Results

```
print(bayes.score(X_validate,y_validate))
0.9333333333333333

print(bayes.score(X_test1,y_test1))
0.9
```

Conclusion

Thus we have successfully completed the implementation of Naïve Bayes Gaussian Classifier.

Assignment 5

Aim: Using inbuilt dataset of Breast cancer from scikit learn Implement PCA algorithm

Objectives:

1. To learn about dimensionality reduction techniques
2. To implement principle component analysis

Theory:

Curse of Dimensionality

The **curse of dimensionality** refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces (often with hundreds or thousands of dimensions) that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience. The expression was coined by Richard E. Bellman when considering problems in dynamic programming.^{[1][2]}

Cursed phenomena occur in domains such as numerical analysis, sampling, combinatorics, machine learning, data mining and databases. The common theme of these problems is that when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality. Also, organizing and searching data often relies on detecting areas where objects form groups with similar properties; in high dimensional data, however, all objects appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient.

PCA

Given a collection of points in two, three, or higher dimensional space, a "best fitting" line can be defined as one that minimizes the average squared distance from a point to the line. The next best-fitting line can be similarly chosen from directions perpendicular to the first. Repeating this process yields an orthogonal basis in which different individual dimensions of the data are uncorrelated. These basis vectors are called **principal components**, and several related procedures **principal component analysis (PCA)**.

PCA is mostly used as a tool in exploratory data analysis and for making predictive models. It is often used to visualize genetic distance and relatedness between populations. PCA is either done in the following 2 steps:

1. calculating the data covariance (or correlation) matrix of the original data
2. performing eigenvalue decomposition on the covariance matrix

LDA

Linear discriminant analysis (LDA), **normal discriminant analysis (NDA)**, or **discriminant function analysis** is a generalization of **Fisher's linear discriminant**, a method used in statistics, pattern recognition, and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting

combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification.

LDA is closely related to analysis of variance (ANOVA) and regression analysis, which also attempt to express one dependent variable as a linear combination of other features or measurements.

Difference between PCA AND LDA

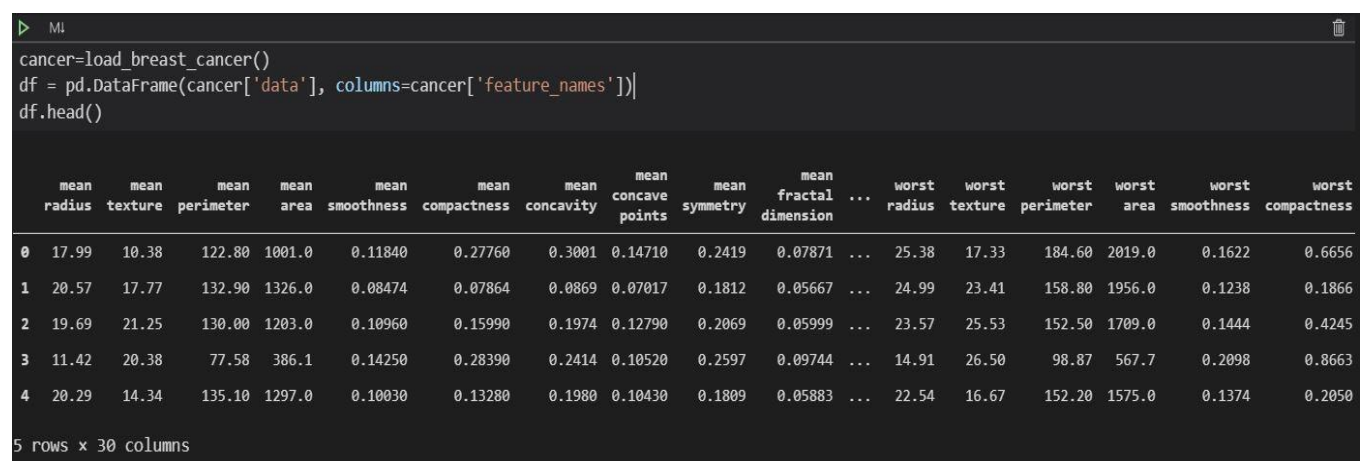
LDA => This method identifies components (i.e., linear combination of the observed variables) that **maximize class separation (i.e. between-class variance)** when such prior information is available (i.e., supervised). E.g., you have a training set containing a variable specifying the class of each observation.

PCA => Aims to find components that account for **maximum variance in the data** (including error and within-variable variance). Unlike LDA, it does not take into account class membership (i.e., unsupervised), and is used when such information is not available. Importantly, both LDA and PCA do not require any prior notion of how the variables are related among themselves, and the resulting components can not be interpreted in terms of an underlying construct.

Code

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
cancer=load_breast_cancer()
df = pd.DataFrame(cancer['data'], columns=cancer['feature_names'])df.head()
x=df.iloc[:,0:8]
y=df['mean fractal dimension']
from sklearn.decomposition import PCA
pca=PCA(n_components=2) pc=pca.fit_transform(x)
pdf=pd.DataFrame(data=pc,columns=['pc1','pc2'])
pdf.head()
```

Dataset



```
cancer=load_breast_cancer()
df = pd.DataFrame(cancer['data'], columns=cancer['feature_names'])
df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.6656
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.1866
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.4245
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.8663
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.2050

5 rows x 30 columns

Results

```

> Ml
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
pc=pca.fit_transform(x)
pdf=pd.DataFrame(data=pc,columns=['pc1','pc2'])

> Ml
pdf.head()

```

	pc1	pc2
0	347.389725	3.450407
1	672.360671	6.224443
2	549.459902	-0.291580
3	-269.152753	-4.104915
4	643.562714	6.209551

Conclusion

Thus we have successfully completed the implementation of the Principle component Analysis Algorithm.

Assignment 6

Aim: Implement decision tree classification/regression technique for given dataset

Developed model should be able to answer the given queries.

Objectives

1. To learn about decision trees
2. To implement decision trees and compare results

Theory:

Decision Tree:

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

A tree can be “*learned*” by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called *recursive partitioning*. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

Different types:

Decision trees used in data mining are of two main types:

- **Classification tree** analysis is when the predicted outcome is the class (discrete) to which the data belongs.
- **Regression tree** analysis is when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient's length of stay in a hospital).

The term **Classification And Regression Tree (CART)** analysis is an umbrella term used to refer to both of the above procedures, first introduced by Breiman et al. in 1984.^[4] Trees used for regression and trees used for classification have some similarities - but also some differences, such as the procedure used to determine where to split.^[4]

Some techniques, often called *ensemble* methods, construct more than one decision tree:

- **Boosted trees** Incrementally building an ensemble by training each new instance to emphasize the training instances previously mis-modeled. A typical example is AdaBoost. These can be used for regression-type and classification-type problems.^{[5][6]}
- **Bootstrap aggregated** (or bagged) decision trees, an early ensemble method, builds multiple decision trees by repeatedly resampling training data with replacement, and voting the trees for a consensus prediction.^[7]
 - o A **random forest** classifier is a specific type of bootstrap aggregating
- **Rotation forest** – in which every decision tree is trained by first applying principal component analysis (PCA) on a random subset of the input features.^[8]

A special case of a decision tree is a decision list,^[9] which is a one-sided decision tree, so that

every internal node has exactly 1 leaf node and exactly 1 internal node as a child (except for the bottommost node, whose only child is a single leaf node). While less expressive, decision lists are arguably easier to understand than general decision trees due to their added sparsity, permit non-greedy learning methods^[10] and monotonic constraints to be imposed.^[11]

Notable decision tree algorithms include:

- ID3 (Iterative Dichotomiser 3)
- C4.5 (successor of ID3)
- CART (Classification And Regression Tree)^[4]
- Chi-square automatic interaction detection (CHAID). Performs multi-level splits when computing classification trees.^[12]
- MARS: extends decision trees to handle numerical data better.
- Conditional Inference Trees. Statistics-based approach that uses non-parametric tests as splitting criteria, corrected for multiple testing to avoid overfitting. This approach results in unbiased predictor selection and does not require pruning.

ID3 and CART were invented independently at around the same time (between 1970 and 1980), yet follow a similar approach for learning a decision tree from training tuples.

Difference between decision tree classification and regression Comparison Chart

BASIS FOR COMPARISON	CLASSIFICATION	REGRESSION
Basic	The discovery of model or functions where the mapping of objects is done into predefined classes.	A devised model in which the mapping of objects is done into values.
Involves prediction of	Discrete values	Continuous values
Algorithms	Decision tree, logistic regression, etc.	Regression tree (Random forest), Linear regression, etc.

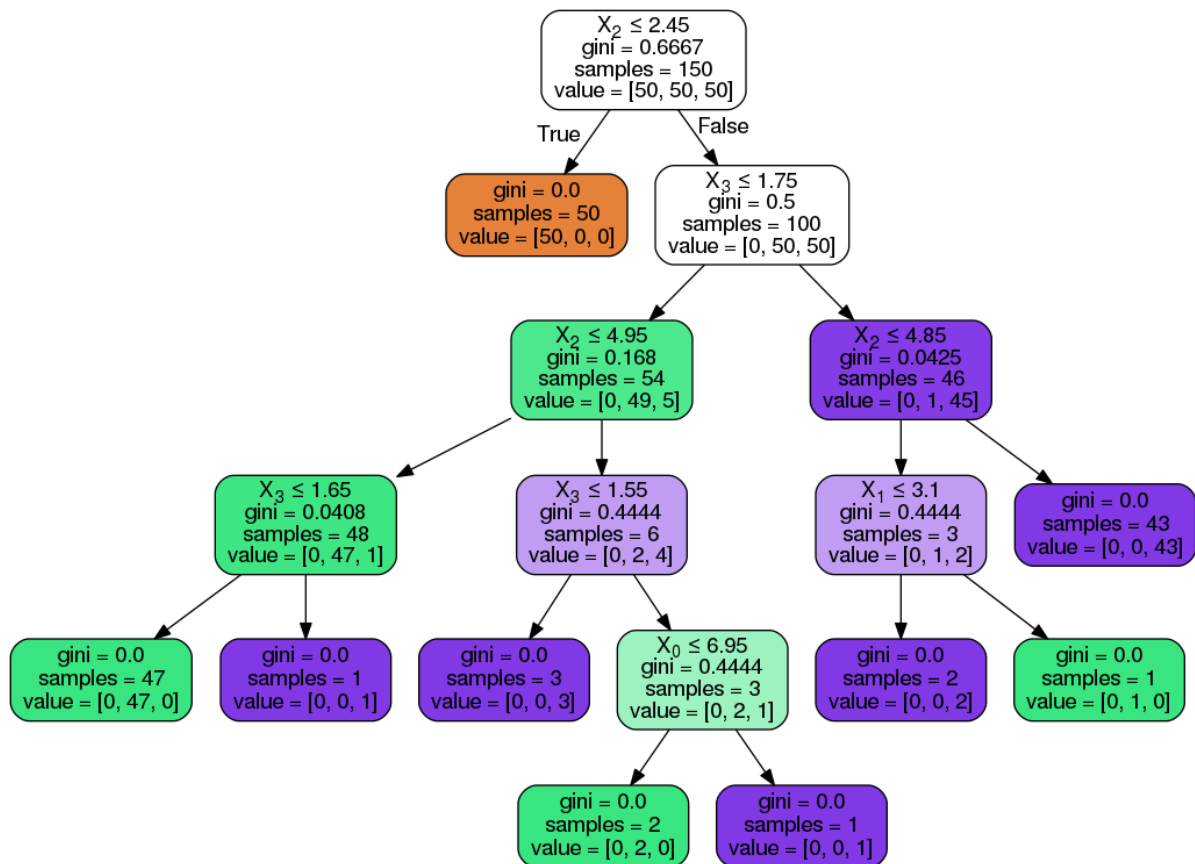
BASISFOR COMPARISON	CLASSIFICATION	REGRESSION
Nature of the predicted data	Unordered	Ordered

Visualizing decision trees

```

From sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplusdot_data = StringIO()
export_graphviz(dtree, out_file=dot_data,
filled=True, rounded=True,
special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())

```

Code

```

import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import export_graphviz
import pydotplus
from sklearn.externals.six import StringIO
from IPython.display import Image
from pydot import graph_from_dot_data
df=pd.read_csv("train.csv")
df.fillna(method="ffill",inplace=True)
df1=pd.read_csv("test.csv")
df1.fillna(method="ffill",inplace=True)
df2=pd.read_csv("gender_submission.csv")
df2.fillna(method="ffill",inplace=True)
le=LabelEncoder()
df["Name"]=le.fit_transform(df["Name"])
df["Sex"]=le.fit_transform(df["Sex"])
df["Ticket"]=le.fit_transform(df["Ticket"])

```

```

#df["Cabin"]=le.fit_transform(df["Cabin"])
df["Embarked"]=le.fit_transform(df["Embarked"])    df1["Sex"]=le.fit_transform(df1["Sex"])
df1["Ticket"]=le.fit_transform(df1["Ticket"])      #df["Cabin"]=le.fit_transform(df["Cabin"])
df1["Embarked"]=le.fit_transform(df1["Embarked"])x=df.iloc[:,[0,2,4,5,6,7,8,9,11]]
y=df["Survived"]
obj=StandardScaler()
x=obj.fit_transform(x)df
x1=df1.iloc[:,[0,1,3,4,5,6,7,8,10]]
y1=df2["Survived"]
x1=obj.fit_transform(x1)df1
x2=df2.iloc[:,0]
y2=df2["Survived"]
dt=DecisionTreeClassifier()
dt.fit(x,y) print(dt.score(x1,y1))
print(dt.score(x,y))

```

Dataset

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	108	1	22.0	1	0	523	7.2500	NaN	2
1	2	1	1	190	0	38.0	1	0	596	71.2833	C85	0
2	3	1	3	353	0	26.0	0	0	669	7.9250	C85	2
3	4	1	1	272	0	35.0	1	0	49	53.1000	C123	2
4	5	0	3	15	1	35.0	0	0	472	8.0500	C123	2
...
886	887	0	2	548	1	27.0	0	0	101	13.0000	C50	2
887	888	1	1	303	0	19.0	0	0	14	30.0000	B42	2
888	889	0	3	413	0	19.0	1	2	675	23.4500	B42	2
889	890	1	1	81	1	26.0	0	0	8	30.0000	C148	0
890	891	0	3	220	1	32.0	0	0	466	7.7500	C148	1

891 rows × 12 columns

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	1	34.5	0	0	152	7.8292	NaN	1
1	893	3	Wilkes, Mrs. James (Ellen Needs)	0	47.0	1	0	221	7.0000	NaN	2
2	894	2	Myles, Mr. Thomas Francis	1	62.0	0	0	73	9.6875	NaN	1
3	895	3	Wirz, Mr. Albert	1	27.0	0	0	147	8.6625	NaN	2
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	0	22.0	1	1	138	12.2875	NaN	2
...
413	1305	3	Spector, Mr. Woolf	1	28.0	0	0	267	8.0500	C78	2
414	1306	1	Oliva y Ocana, Dona. Fermina	0	39.0	0	0	324	108.9000	C105	0
415	1307	3	Saether, Mr. Simon Sivertsen	1	38.5	0	0	346	7.2500	C105	2
416	1308	3	Ware, Mr. Frederick	1	38.5	0	0	220	8.0500	C105	2
417	1309	3	Peter, Master. Michael J	1	38.5	1	1	105	22.3583	C105	0

418 rows x 11 columns

Training

```

dt=DecisionTreeClassifier()
dt.fit(x,y)

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                        splitter='best')

```

Results

```

print(dt.score(x1,y1))

0.8205741626794258

```

Conclusion

Thus we have successfully completed the implementation of decision tree classifier.

Assignment 7

Aim: Implement SVM Classifier for given dataset

Objectives:

1. To learn SVM and kernel functions
2. To implement SVM classifier

Theory:

SVM:

support-vector machines (SVMs, also **support-vector networks^[1]**) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

Kernel function

In machine learning, **kernel methods** are a class of algorithms for pattern analysis, whose best known member is the support vector machine (SVM). The general task of pattern analysis is to find and study general types of relations (for example clusters, rankings, principal components, correlations, classifications) in datasets. For many algorithms that solve these tasks, the data in raw representation have to be explicitly transformed into feature vector representations via a user-specified *feature map*: in contrast, kernel methods require only a user-specified *kernel*, i.e., a similarity function over pairs of data points in raw representation.

Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, *implicit* feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "**kernel trick**".^[1] Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

Kernel trick

The kernel trick seems to be one of the most confusing concepts in statistics and machine learning; it first appears to be genuine mathematical sorcery, not to mention the problem of lexical ambiguity (does kernel refer to: a non-parametric way to estimate a probability density (statistics), the set of vectors \mathbf{v} for which a linear transformation T maps to the zero vector — i.e. $T(\mathbf{v}) = 0$ (linear algebra), the set of elements in a group G that are mapped to the identity

element by a homomorphism between groups (group theory), the core of a computer operating system (computer science), or something to do with the seeds of nuts or fruit?).

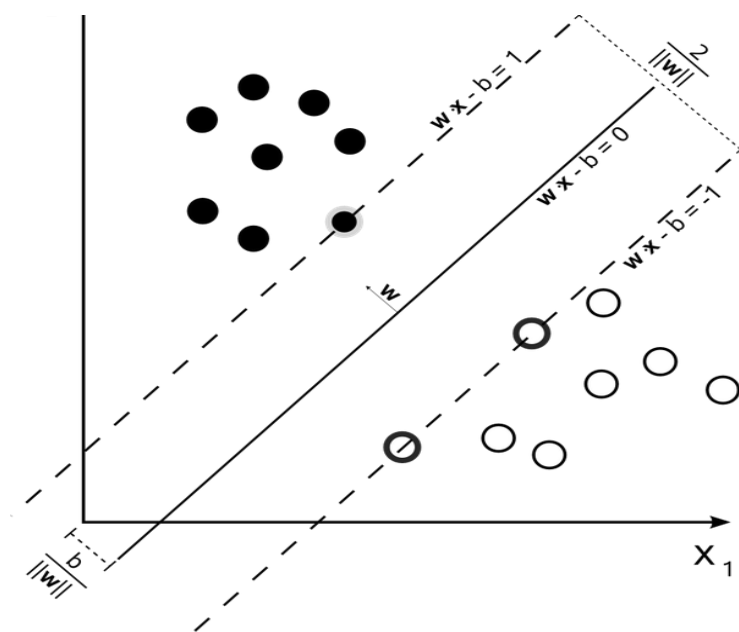
The kernel trick also illustrates some fundamental ideas about different ways to represent data and how machine learning algorithms “see” these different data representations. And finally, the seeming mathematical sleight of hand in the kernel trick just begs one to further explore what it actually means.

Significance of SVM

it capable of doing both classification and regression. In this post I'll focus on using SVM for classification. In particular I'll be focusing on non-linear SVM, or SVM using a non-linear kernel. Non-linear SVM means that the boundary that the algorithm calculates doesn't have to be a straight line. The benefit is that you can capture much more complex relationships between your datapoints without having to perform difficult transformations on your own. The downside is that the training time is much longer as it's much more computationally intensive.

Originally Answered: what is the purpose of support vector in SVM?

A support vector machine attempts to find the line that "best" separates two classes of points. By "best", we mean the line that results in the **largest margin** between the two classes. The points that lie on this margin are the support vectors.



Here we have three support vectors.

The nice thing about acknowledging these support vectors is that we can then formulate the problem of finding the "maximum-margin hyperplane" (the line that best separates the two classes) as an optimization problem that only considers these support vectors. So we can effectively throw out the vast majority of our data, which makes the classification process go much faster than, say, a neural network.

More importantly, by presenting the problem in terms of the support vectors (the so-called *dual form*), we can apply what's called the *kernel trick* to effectively transform the SVM into a non-linear classifier.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
#df = pd.read_csv('creditcard.csv', dtype={'v1':int})
df=pd.read_csv('creditcard.csv')
count1=df['Class'].astype(bool).sum(axis=0)

count2=df['Class'].count()-df['Class'].astype(bool).sum(axis=0)print(count1)
print(count2)
x=df.iloc[:,1:29]
y=df['Amount']
from sklearn.preprocessing import StandardScaler
obj=StandardScaler()
x1=obj.fit_transform(x)
plt.matshow(df.corr())

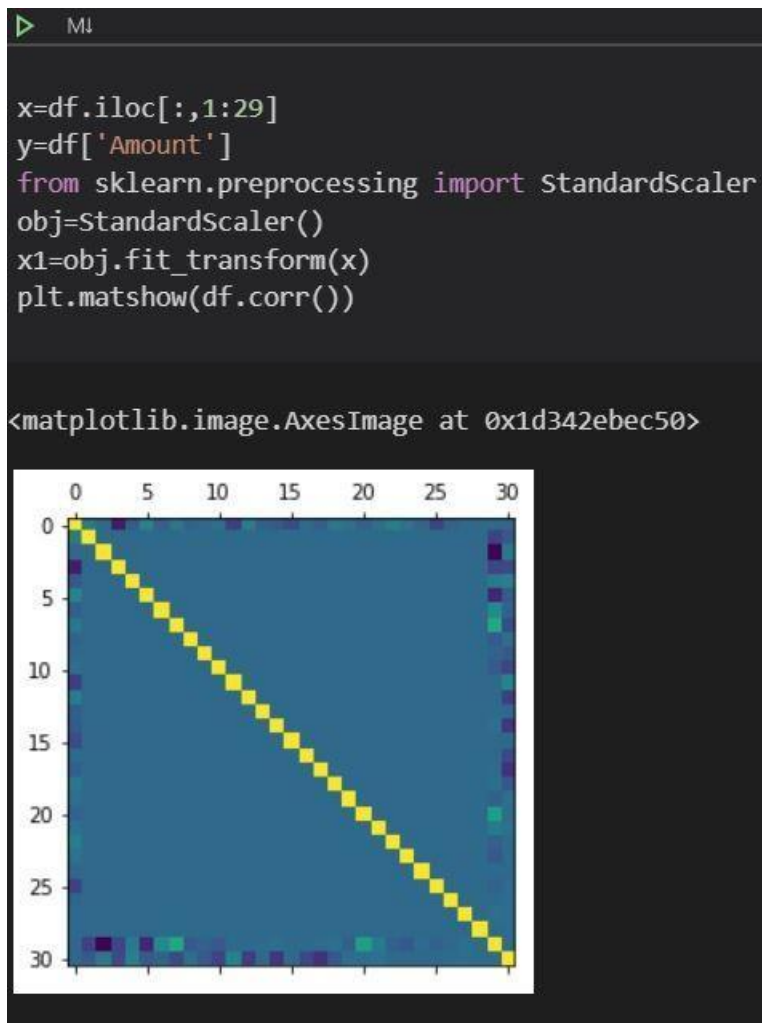
df_train_all = df[0:150000]
df_train_1 = df_train_all[df_train_all['Class'] == 1] df_train_0
= df_train_all[df_train_all['Class'] == 0]
df_sample=df_train_0.sample(300)
df_train = df_train_1.append(df_sample)
df_train = df_train.sample(frac=1)
x=df.iloc[:,0:30]
y=df['Class']
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest=train_test_split(x,y,test_size=0.2,random_state=42)
from sklearn.svm import LinearSVC
obj1=LinearSVC()
obj1.fit(Xtrain,ytrain)
print(obj1.score(Xtest,ytest))
ypred=obj1.predict(Xtest)
confusion_matrix(ytest, ypred)
confusion_matrix(ytrain, obj1.predict(Xtrain))
```

Dataset

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0

5 rows x 31 columns

Correlation Matrix



Training

```
from sklearn.svm import LinearSVC
obj1=LinearSVC()
obj1.fit(Xtrain,ytrain)

LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

Result

```
print(obj1.score(Xtest,ytest))

0.9983673326077034
```

Confusion Matrix

```
ypred=obj1.predict(Xtest)
confusion_matrix(ytest, ypred)
confusion_matrix(ytrain, obj1.predict(Xtrain))

array([[227443,      8],
       [   361,    33]], dtype=int64)
```

Conclusion:

Thus we have successfully completed the implementation of Support VectorMachine

Assignment 8

Aim: Implement K means algorithm for multidimensional data

Objectives:

1. To learn unsupervised learning
2. To implement K means algorithm

Theory:

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labelled responses. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.

Common clustering algorithms include:

- Hierarchical clustering: builds a multilevel hierarchy of clusters by creating a cluster tree
- k-Means clustering: partitions data into k distinct clusters based on distance to the centroid of a cluster
- Gaussian mixture models: models clusters as a mixture of multivariate normal density components
- Self-organizing maps: uses neural networks that learn the topology and distribution of the data
- Hidden Markov models: uses observed data to recover the sequence of states

Unsupervised learning methods are used in bioinformatics for sequence analysis and genetic clustering; in data mining for sequence and pattern mining; in medical imaging for image segmentation; and in computer vision for object recognition.

Clustering and its types:

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

Let's understand this with an example. Suppose, you are the head of a rental store and wish to understand preferences of your costumers to scale up your business. Is it possible for you to look at details of each costumer and devise a unique business strategy for each one of them? Definitely not. But, what you can do is to cluster all of your costumers into say 10 groups based on their purchasing habits and use a separate strategy for costumers in each of these 10 groups. And this is what we call clustering.

A "clustering" is essentially a set of such clusters, usually containing all objects in the data set. Additionally, it may specify the relationship of the clusters to each other, for example, a hierarchy of clusters embedded in each other. Clustering's can be roughly distinguished as:

- Hard clustering: each object belongs to a cluster or not
- Soft clustering (also: fuzzy clustering): each object belongs to each cluster to a certain degree (for example, a likelihood of belonging to the cluster)

Types:

- Strict partitioning clustering: each object belongs to exactly one cluster
- Strict partitioning clustering with outliers: objects can also belong to no cluster, and are considered outliers
- Overlapping clustering (also: alternative clustering, multi-view clustering): objects may belong to more than one cluster; usually involving hard clusters
- Hierarchical clustering: objects that belong to a child cluster also belong to the parent cluster
- Subspace clustering: while an overlapping clustering, within a uniquely defined subspace, clusters are not expected to overlap

K means algorithm

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each datapoint belongs to only one group. It tries to make the inter-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way k means algorithm works is as follows:

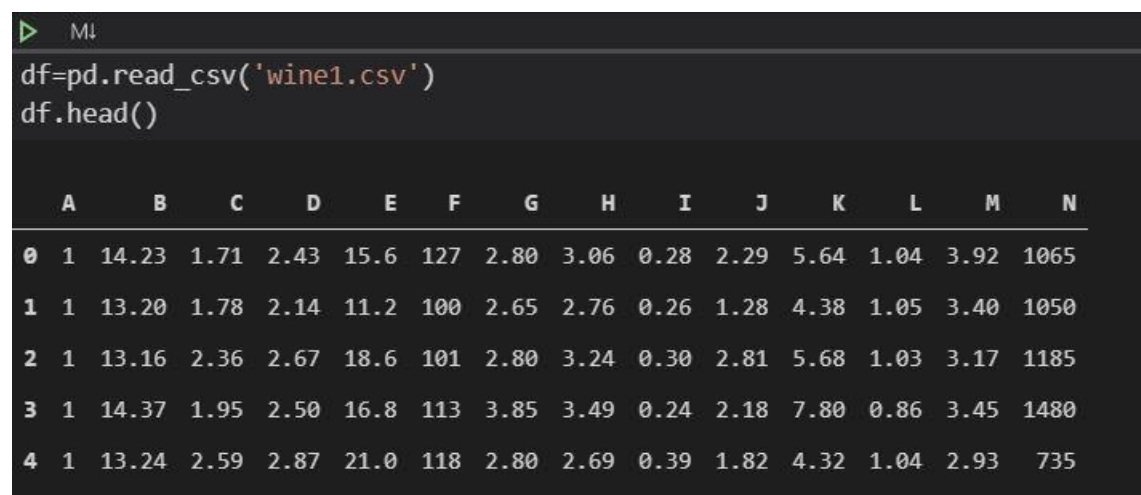
1. Specify number of clusters K.
 2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
 3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.
- Compute the sum of the squared distance between data points and all centroids.
 - Assign each data point to the closest cluster (centroid).
 - Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

The approach kmeans follows to solve the problem is called Expectation- Maximization.

Code:

```
import pandas as pd
from matplotlib import pyplot as plt
df=pd.read_csv('wine1.csv')
from sklearn.preprocessing import StandardScaler from
sklearn.model_selection import train_test_split from
sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
km=KMeans(n_clusters=3, max_iter=400,verbose=True,tol=0.2) pred
= km.fit_predict(df[['B','C']])
plt.scatter(df['B'], df['C'],c=df['A']) score =
accuracy_score(df['A'], pred) score
plt.scatter(df['B'], df['C'], c=pred)
plt.scatter(km.cluster_centers_[0],km.cluster_centers_[1], color='black')
```

Dataset



```
df=pd.read_csv('wine1.csv')
df.head()
```

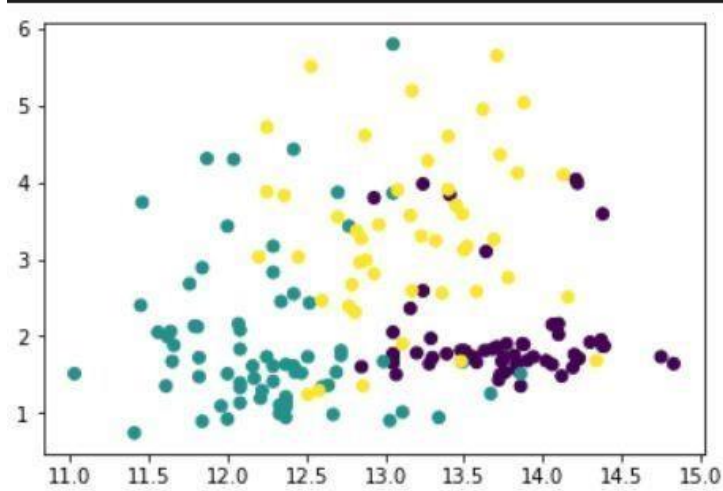
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

Points before Fitting

```

> plt.scatter(df['B'], df['C'], c=df['A'])
matplotlib.collections.PathCollection at 0x12759b752e8>

```



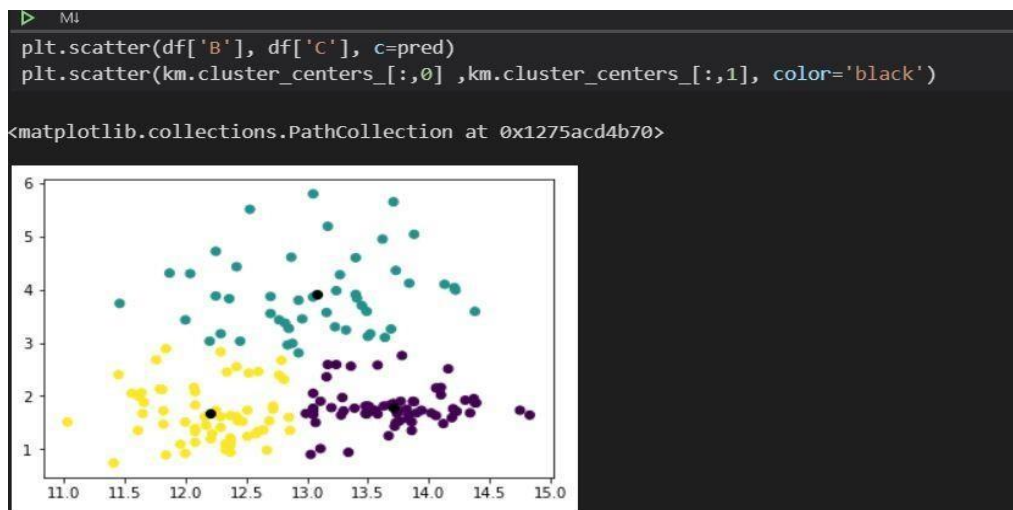
Training

```

> km.fit_predict(df[['B', 'C']])
end inner loop
Iteration 0, inertia 96.202308592161
center shift 4.182927e-01 within tolerance 1.896364e-01
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 122.93593701832567
start iteration
done sorting
end inner loop
Iteration 1, inertia 99.99686808125053
start iteration
done sorting
end inner loop
Iteration 2, inertia 96.41264585221674
center shift 1.924536e-01 within tolerance 1.896364e-01
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 154.17291868486353
start iteration
done sorting
end inner loop
Iteration 1, inertia 152.8250241227828

```

Points after implementing algorithm



Conclusion

Thus we have successfully completed the implementation of Kmeans algorithm

Assignment 9

Aim: Implement KNN algorithm on provided dataset. Calculate the performance metric and compare the error rate with K value(K value range 1 to n).

Objectives:

1. To learn KNN algorithm
2. To implement KNN classifier

Theory:

1. Lazy learners

Lazy learners simply store the training data and wait until a testing data appear. When it does, classification is conducted based on the most related data in the stored training data. Compared to eager learners, lazy learners have less training time but more time in predicting.

Eager learners

Eager learners construct a classification model based on the given training data before receiving data for classification. It must be able to commit to a single hypothesis that covers the entire instance space. Due to the model construction, eager learners take a long time for train and less time to predict.

Lazy learner:

1. Just store Data set **without** learning from it
 2. Start classifying data when it receives **Test data**
 3. So, it takes less time learning and more time classifying data
-

Eager learner:

1. When it receives data set it starts classifying (learning)
2. Then it does not wait for test data to learn
3. So, it takes long time learning and less time classifying data

KNN Algorithm

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

The KNN Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbours
3. For each example in the data
 - 3.1 Calculate the distance between the query example and the current example from the data.
 - 3.2 Add the distance and the index of the example to an ordered collection
4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
7. If regression, return the mean of the K labels
8. If classification, return the mode of the K labels

Code:

```
import pandas as pd
df=pd.read_csv('IRIS.csv')
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df.head() x=df.iloc[:,0:3]
y=df['species']
obj=StandardScaler()
x=obj.fit_transform(x)
le.fit_transform(['Iris-setosa','Iris-versicolor','Iris-virginica']) xtrain, xtest, ytrain,
ytest=train_test_split(x,y,test_size=0.3) knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(xtrain,ytrain)
print(knn.score(xtest,ytest))
from sklearn import metrics
ypred=knn.predict(xtest)
confusion_matrix(ytest,ypred)
k_range=range(1,26)
errorl={}
errorlist=[]
for k in k_range:
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(xtrain,ytrain) ypred=knn.predict(xtest)
    errorl[k]=1-metrics.accuracy_score(ytest,ypred)
    errorlist.append(1-metrics.accuracy_score(ytest,ypred))

import matplotlib.pyplot as plt
plt.plot(k_range, errorlist)
plt.xlabel("K values")
plt.ylabel("Error")
```

Dataset

```
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Training

```
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(xtrain,ytrain)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                     weights='uniform')
```

Results

```
print(knn.score(xtest,ytest))

0.9333333333333333
```

Confusion Matrix

```
from sklearn import metrics
ypred=knn.predict(xtest)
confusion_matrix(ytest,ypred)

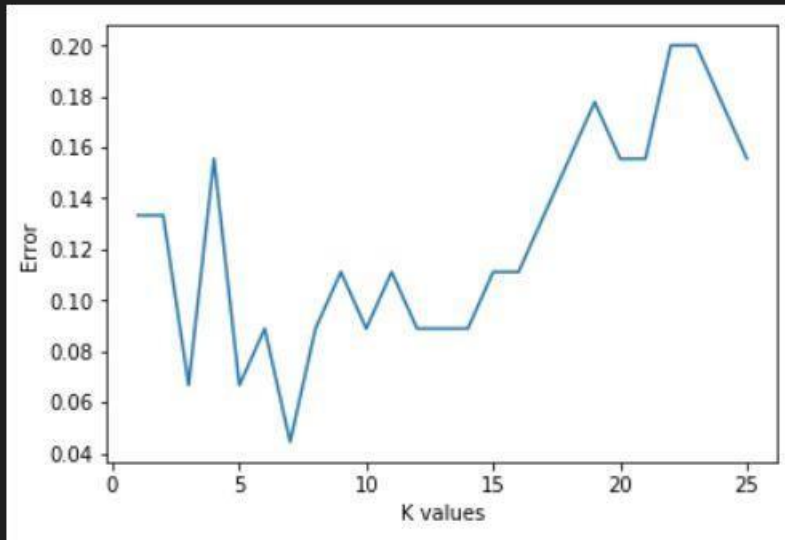
array([[10,  0,  0],
       [ 0, 12,  0],
       [ 0,  3, 20]], dtype=int64)
```


Error Graph

MI

```
plt.plot(k_range, errorlist)
plt.xlabel("K values")
plt.ylabel("Error")
```

```
Text(0,0.5,'Error')
```



Conclusion

Thus we have successfully completed the implementation of KNN algorithm

Assignment 10

Aim: Implement CNN for MNIST/CIFAR10 dataset using tensor flow

Objectives:

1. To learn basics of deep learning
2. To learn and implement CNN

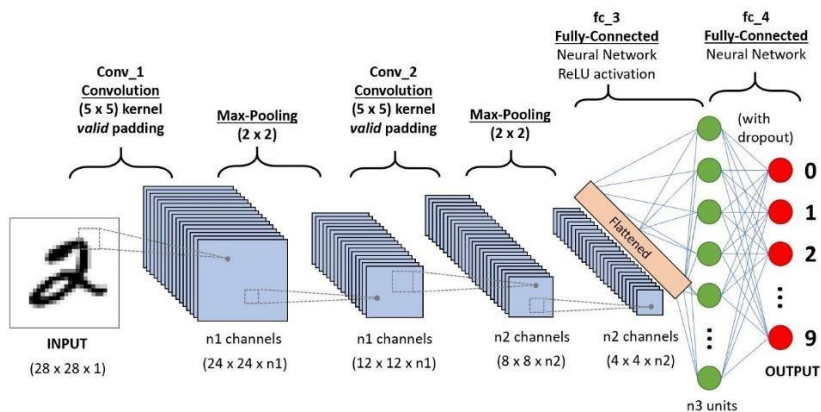
Theory:

Deep Learning

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network.

Deep learning, a subset of machine learning, utilizes a hierarchical level of artificial neural networks to carry out the process of machine learning. The artificial neural networks are built like the human brain, with neuron nodes connected together like a web. While traditional programs build analysis with data in a linear way, the hierarchical function of deep learning systems enables machines to process data with a nonlinear approach.

CNN Architecture



CNN WORKING

A Convolutional Neural Networks Introduction so to speak.

- Step 1: Convolution Operation

The first building block in our plan of attack is convolution operation. In this step, we will touch on feature detectors, which basically serve as the neural network's filters. We will also discuss feature maps, learning the parameters of such maps, how patterns are detected, the layers of detection, and how the findings are mapped out.

- Step 1(b): ReLU Layer

The second part of this step will involve the Rectified Linear Unit or ReLU. We will cover ReLU layers and explore how linearity functions in the context of Convolutional Neural Networks.

Not necessary for understanding CNN's, but there's no harm in a quick lesson to improve your skills.

- Step 2: Pooling

In this part, we'll cover pooling and will get to understand exactly how it generally works. Our nexus here, however, will be a specific type of pooling; max pooling. We'll cover various approaches, though, including mean (or sum) pooling. This part will end with a demonstration made using a visual interactive tool that will definitely sort the whole concept out for you.

- Step 3: Flattening

This will be a brief breakdown of the flattening process and how we move from pooled to flattened layers when working with Convolutional Neural Networks.

- Step 4: Full Connection

In this part, everything that we covered throughout the section will be merged together. By learning this, you'll get to envision a fuller picture of how Convolutional Neural Networks operate and how the "neurons" that are finally produced learn the classification of images.

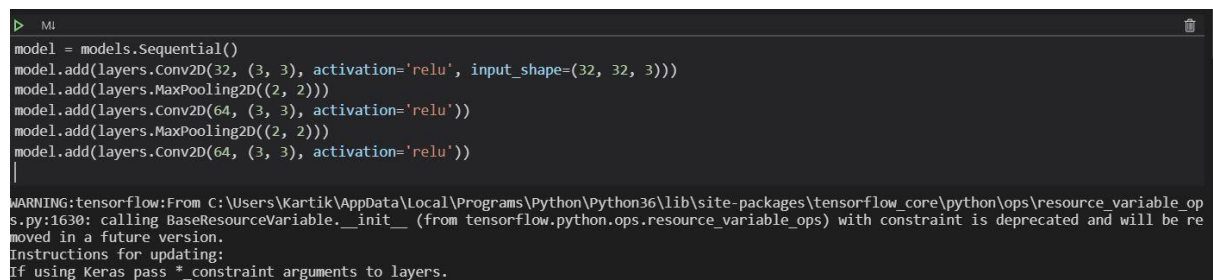
Code

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
train_images, test_images = train_images/255, test_images/255

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.compile(optimizer='SGD', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(test_acc)
```

Model making



```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

WARNING:tensorflow:From C:\Users\Kartik\AppData\Local\Programs\Python\Python36\lib\site-packages\tensorflow_core\python\ops\resource_variable_ops.py:1630: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.

Model Summary

```
ML
model.summary()

Model: "sequential"
Layer (type)                Output Shape                Param #
=====
conv2d (Conv2D)              (None, 30, 30, 32)         896
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)         0
conv2d_1 (Conv2D)            (None, 13, 13, 64)         18496
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
conv2d_2 (Conv2D)            (None, 4, 4, 64)           36928
=====
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
```

Model Modifications and summary

```
ML
model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10))

ML
model.summary()

Model: "sequential"
Layer (type)                Output Shape                Param #
=====
conv2d (Conv2D)              (None, 30, 30, 32)         896
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)         0
conv2d_1 (Conv2D)            (None, 13, 13, 64)         18496
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
conv2d_2 (Conv2D)            (None, 4, 4, 64)           36928
flatten (Flatten)            (None, 1024)                0
dense (Dense)                (None, 64)                  65600
dense_1 (Dense)              (None, 10)                  650
=====
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
```

Training

```
▶ MI
model.compile(optimizer='SGD',loss=tp.keras.losses.SparseCategoricalCrossentropy(from_logits=True),metrics=['accuracy'])
history=model.fit(train_images, train_labels, epochs=10, validation_data=(test_images,test_labels))

Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [=====] - 67s 1ms/sample - loss: 0.5307 - acc: 0.8138 - val_loss: 0.9441 - val_acc: 0.7092
Epoch 2/10
50000/50000 [=====] - 58s 1ms/sample - loss: 0.4727 - acc: 0.8336 - val_loss: 0.9893 - val_acc: 0.6975
Epoch 3/10
50000/50000 [=====] - 58s 1ms/sample - loss: 0.4483 - acc: 0.8429 - val_loss: 1.0316 - val_acc: 0.6951
Epoch 4/10
50000/50000 [=====] - 55s 1ms/sample - loss: 0.4345 - acc: 0.8468 - val_loss: 0.9480 - val_acc: 0.7192
Epoch 5/10
50000/50000 [=====] - 54s 1ms/sample - loss: 0.4199 - acc: 0.8525 - val_loss: 0.9589 - val_acc: 0.7176
Epoch 6/10
50000/50000 [=====] - 60s 1ms/sample - loss: 0.4080 - acc: 0.8563 - val_loss: 0.9990 - val_acc: 0.7182
Epoch 7/10
50000/50000 [=====] - 87s 2ms/sample - loss: 0.3965 - acc: 0.8611 - val_loss: 0.9726 - val_acc: 0.7240
Epoch 8/10
50000/50000 [=====] - 68s 1ms/sample - loss: 0.3863 - acc: 0.8656 - val_loss: 1.0029 - val_acc: 0.7178
Epoch 9/10
50000/50000 [=====] - 66s 1ms/sample - loss: 0.3787 - acc: 0.8675 - val_loss: 1.0084 - val_acc: 0.7234
Epoch 10/10
50000/50000 [=====] - 64s 1ms/sample - loss: 0.3701 - acc: 0.8701 - val_loss: 1.0067 - val_acc: 0.7185
```

Results

```
▶ MI
print(test_acc)

0.7185
```

Conclusion

Thus we have successfully completed the implementation of CNN algorithm using tensorflow

Assignment 11

Aim: Perform basic image processing using opencv

Objectives:

1. To learn about images processing
 2. To perform various operations on images
- Theory
OpenCV

OpenCV (*Open Source Computer Vision Library*) is a library of programming functions mainly aimed at real-time computer vision.^[1] Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel^[2]). The library is cross-platform and free for use under the open-source BSD license.

OpenCV supports some models from deep learning frameworks like TensorFlow, Torch, PyTorch (after converting to an ONNX model) and Caffe according to a defined list of supported layers.^[3] It promotes OpenVisionCapsules.^[4], which is a portable format, compatible with all other formats.

Image processing operations

Digital image processing is the use of a digital computer to *process* digital images through an *algorithm*.^{[1][2]} As a subcategory or field of digital signal processing, digital image processing has many advantages over analog image processing. It allows a much wider range of algorithms to be applied to the input data and can avoid problems such as the build-up of noise and distortion during processing. Since images are defined over two dimensions (perhaps more) digital image processing may be modeled in the form of multidimensional systems. The generation and development of digital image processing are mainly affected by three factors: first, the development of computers; second, the development of mathematics (especially the creation and improvement of discrete mathematics theory); third, the demand for a wide range of applications in environment, agriculture, military, industry and medical science has increased.

Digital image processing allows the use of much more complex algorithms, and hence, can offer both more sophisticated performance at simple tasks, and the implementation of methods which would be impossible by analogue means.

In particular, digital image processing is a concrete application of, and a practical technology based on:

- Classification
- Feature extraction
- Multi-scale signal analysis
- Pattern recognition
- Projection

Some techniques which are used in digital image processing include:

- Anisotropic diffusion
- Hidden Markov models
- Image editing
- Image restoration
- Independent component analysis
- Linear filtering

- Neural networks
- Partial differential equations
- Pixelation
- Point feature matching
- Principal components analysis
- Self-organizing maps
- Wavelets

OpenCV methods

Image Acquisition

OpenCV gives the flexibility to capture image directly from a pre-recorded video stream, camera input feed, or a directory path.

```
#Taking input from a directory path
img = cv2.imread('C:\Users\USER\Desktop\image.jpg',0)#Capturing input from a video stream
cap = cv2.VideoCapture(0)
```

Histogram Equalization :

Representation of intensity distribution vs no. of pixels of an image is termed as the histogram. Equalization stretches out the intensity range in order to suit contrast levels

appropriately. More resources [here](#) and [here](#).

```
#Taking input from a directory path
img = cv2.imread('wiki.jpg',0)#Applying Histogram Equalization
equ = cv2.equalizeHist(img)#Save the image
cv2.imwrite('res.png',equ)
```

Erosion and Dilation

Erosion and Dilation belong to the group of morphological transformations and widely used together for the treatment of noise or detection of intensity bumps. Here's an extensive resource on the same from the documentation.

```
#Taking input from a directory path
img = cv2.imread('wiki.jpg',0)#Applying Erosion
img_erosion = cv2.erode(img, kernel, iterations=1)#Applying Dilation
img_dilation = cv2.dilate(img, kernel, iterations=1)
```

Image Denoising

Noise has a very peculiar property that its mean is zero, and this is what helps in its removal by averaging it out.

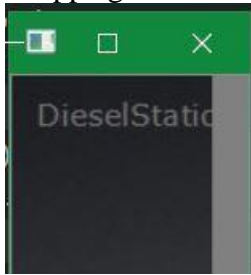
OpenCV provides four variations of this technique.

1. **cv2.fastNlMeansDenoising()** — works with a single grayscale images
2. **cv2.fastNlMeansDenoisingColored()** — works with a color image.
3. **cv2.fastNlMeansDenoisingMulti()** — works with image sequence captured in short period of time (grayscale images)
4. **cv2.fastNlMeansDenoisingColoredMulti()** — same as above, but for color images.

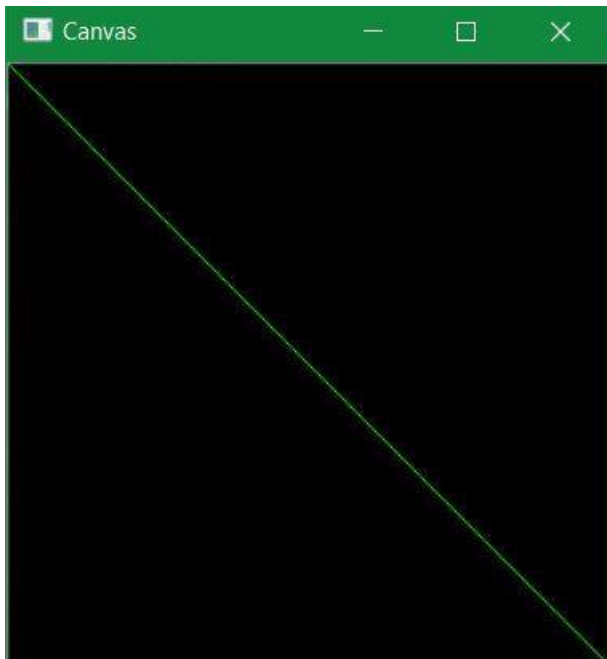
Code

```
import cv2
import numpy as np
image=cv.imread("Lamborghini_Centenario_2017_widescreen_01.jpg")
cv.imshow("Image",image)
cv.waitKey(0) image.shape
corner=image[0:100,0:100]
cv.imshow("Corner", corner)
cv.waitKey(0)
canvas = np.zeros((300, 300, 3), dtype = "uint8")
#RGB(255,255,255)
green = (0, 255, 0)
cv.line(canvas, (0, 0), (300, 300), green)
cv.imshow("Canvas", canvas)
cv.waitKey(0) red =
(0, 0, 255)
cv.line(canvas, (300, 0), (0, 300), red, 3)
cv.imshow("Canvas", canvas)
cv.waitKey(0)
M = np.float32([[1, 0, 25], [0, 1, 50]])
shifted = cv.warpAffine(image, M, (image.shape[1], image.shape[0]))
cv.imshow("Shifted Down and Right", shifted)
cv.waitKey(0)
cv.imshow("Original", image) (h,
w) = image.shape[:2] center = (w
// 2, h // 2)
M = cv.getRotationMatrix2D(center, -45, 1.0)rotated
= cv.warpAffine(image, M, (w, h))
cv.imshow("Rotated by 45 Degrees", rotated)
cv.waitKey(0)
cv.imshow("Original", image)
print(image.shape)
r = 150.0 / image.shape[1]
dim = (150, int(image.shape[0] * r))
resized = cv.resize(image, dim, interpolation = cv.INTER_AREA)
print(resized.shape)
cv.imshow("Resized (Width)", resized)
cv.waitKey(0)
cv.imshow("Original", image)
flipped = cv.flip(image, 0)
cv.imshow("Flipped Horizontally", flipped)
cv.waitKey(0)
```

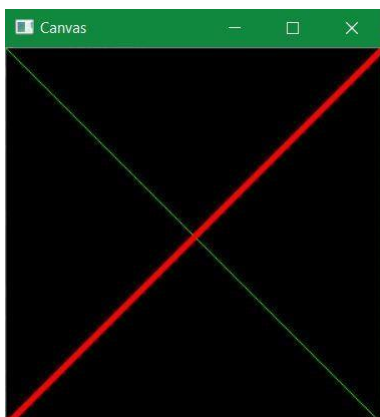
Results
Cropping:



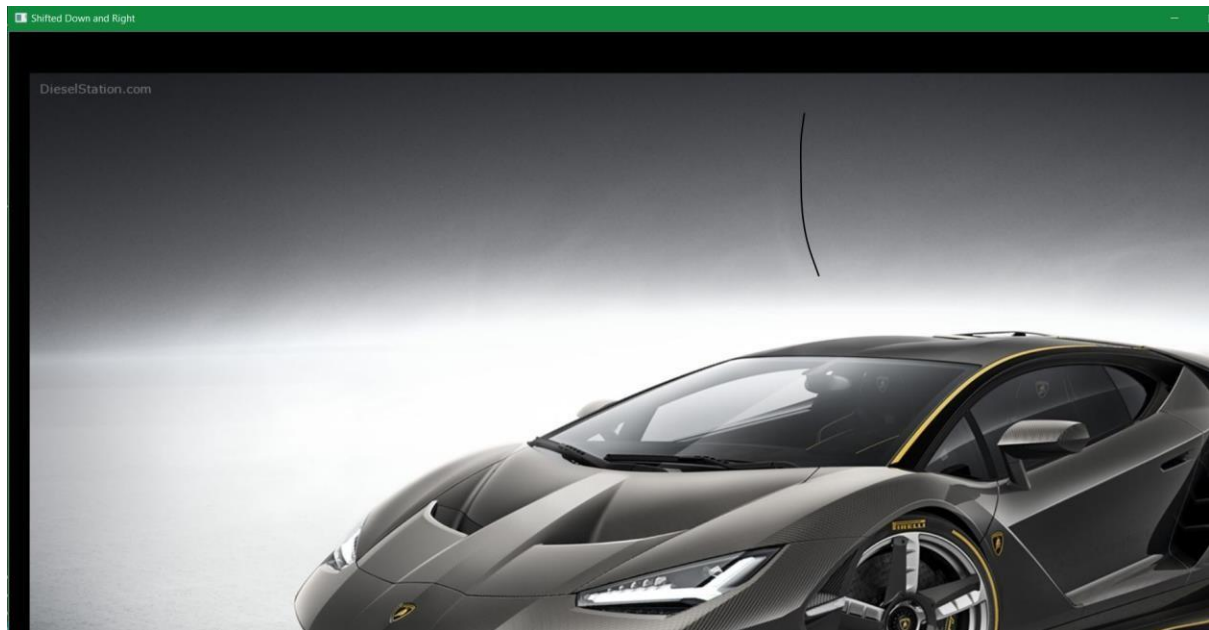
Greenline



Red Line



Shifting the image down right



Rotating the image by 45degrees



Resizing the image



Flipping the image



Conclusion

Thus we have successfully learned the usage of OpenCV and it's functions

Experiment No -12

Title :- Case study on - Generative AI And Conversational AI

Theory :-

What is Conversational AI?

In the vast realm of AI, Conversational AI stands as a bridge between machines and humans, facilitating interactions that mirror natural human conversations. With the ability to understand intent and piecing together necessary information, they ensure the right resolution to customer's queries, and fast!

Designed to help machines understand, process, and respond to human language in an intuitive and engaging manner. Conversational AI thrives on a blend of advanced technologies, ensuring bots not only grasp human language but also respond with human-like precision.

The 4 pillars of Conversational AI are:

1)Machine Learning (ML)

At the heart of Conversational AI, ML employs intricate algorithms to discern patterns from vast data sets. This continuous learning enhances the bot's understanding and response mechanism. For instance, ML powers image recognition, speech recognition, and even self-driving cars, showcasing its versatility across sectors.

2)Natural Language Processing (NLP)

NLP transforms raw user input into machine-readable data, ensuring bots comprehend and generate relevant responses. It encompasses:

- Lexical Analysis: Segmenting text into words or phrases.
- Syntax and Semantic Analysis: Deciphering word relationships and text meanings.
- Sentiment Analysis: Gauging the emotional tone of a text.
- Output Transformation: Crafting human-like responses based on input.

3) Data Mining

Beyond mere pattern recognition, data mining extracts valuable insights from conversational data. It aids in refining the AI system's functionality. For instance, by analyzing customer behaviors, AI can segment customers, enabling businesses to tailor their marketing strategies.

4) Automatic Speech Recognition (ASR)

Essential for voice interactions, ASR deciphers human voice inputs, filters background disturbances, and translates speech to text. Tools like voice-to-text dictation exemplify ASR's capability to streamline tasks.

Together, these components forge a Conversational AI engine that evolves with each interaction, promising enhanced user experiences and fostering business growth.

What is Generative AI?

Venturing into the imaginative side of AI, Generative AI is the creative powerhouse in the AI domain. Unlike traditional AI systems that rely on predefined rules, it uses vast amounts of data to generate original and innovative outputs. By analyzing patterns and learning from existing examples, generative AI models can create realistic images, music, text, and more, often surpassing human imagination.

Employs algorithms to autonomously create content, such as text, images, music, and more, by learning patterns from existing data.

The backbone of generative AI is the Generative Adversarial Network (GAN).

1) Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a class of generative AI models that have revolutionized the field of artificial intelligence by generating realistic and high-quality content. GANs consist of two parts: a generator, which creates images, and a discriminator, which evaluates them. The two work in tandem, effectively “teaching” each other to produce better results. This adversarial process leads to the generation of highly realistic content.

2) Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs) are a type of generative AI model that combine concepts from both autoencoders and probabilistic modeling. They are powerful tools for learning representations of complex data and generating new samples. VAEs allow for the creation of new instances that can be similar to your input data, making them great for tasks like image denoising or inpainting.

Comparison between Conversational AI and Generative AI

	Conversational AI	Generative AI
Objective	Facilitates interactive dialogues between humans and AI, emphasizing natural language processing.	Concentrates on generating fresh and innovative content without explicit instructions.
Inputs & outputs	Takes the user’s natural language as input and returns context-sensitive responses.	Processes various data types to craft new content based on learned patterns.
Training & learning	Primarily trained using vast conversation datasets, ensuring the model’s ability to emulate human-like interactions.	Utilizes diverse data, from text to images, to identify patterns and produce content accordingly.
Applications & use-case	Commonly employed in virtual assistants, chatbots, customer support, language translation tools, and voice-controlled interfaces.	Widely applied in realms like virtual reality, gaming, artistic creations, content generation, product innovations, and multimedia enhancements.

Interactivity	High engagement potential, adapting responses based on real-time user interactions, questions, and feedback.	Generally provides content with minimal interaction, relying heavily on its training data for content generation.
Complexity	Capable of comprehending and reacting to intricate user inquiries, ideal for real-world applications such as customer service.	Generates detailed and imaginative outputs, though might not always grasp the context or specific user inclinations.
Feedback loop	Benefits from user feedback, enabling iterative model enhancements.	Often refines outputs based on curated data, but might lack direct user feedback mechanisms.
Dependencies	Anchored by high-quality, diverse conversational data to ensure relevance.	Dependent on a wide range of data types to hone its content generation capabilities.
Latency	Engineered for real-time responses, prioritizing prompt user interactions.	Might require more processing time, especially for intricate content generation.
Scalability	Adapts well to increasing user interactions, ensuring consistent performance.	Scales effectively with data, refining the richness of generated content over time.
Versatility	Best suited for applications demanding real-time interactions and feedback.	Exhibits adaptability across a spectrum of domains needing content creation.
Data sensitivity	Requires consistent data quality; anomalies can lead to response inaccuracies.	Resilient to varied data inputs but can deviate in output quality with skewed training data.
Limitations	Might occasionally struggle with highly technical or out-of-context queries.	While innovative, can occasionally produce outputs that lack contextual relevance.

Real-life applications of Conversational AI

- **Customer support chatbots:**

These AI-driven chatbots are designed to handle a wide range of customer queries, from simple FAQs to more complex issues. They can operate 24/7, ensuring that customer concerns are addressed promptly.

- **Voice assistants:**

Platforms like Alexa, Google Assistant, and Siri use conversational AI to understand and respond to user commands. They can set reminders, play music, provide weather updates, and more.

- **Social media interactions:**

With the rise of social media platforms, businesses are leveraging conversational AI to interact with customers on platforms like Instagram, Facebook Messenger, and WhatsApp. These AI tools can answer product-related questions, provide recommendations, and even facilitate purchases.

- **Virtual agents for contact centers:**

Conversational AI can streamline the operations of contact centers by handling routine inquiries, thereby allowing human agents to focus on more complex issues.

- **Personalized marketing:**

By analyzing user interactions, conversational AI can provide personalized product recommendations and promotional offers, enhancing the shopping experience for users. For example, Yellow.ai's Engage helps retailers craft Generative AI powered digital campaigns that increase revenue per campaign by 15% and boost lead generation by 20%.

Real-life applications of Generative AI

- **Content creation:**

Generative AI can produce a variety of content, from text (ChatGPT, Google Bard) and images (DALL-E, Midjourney) to music (Musenet) and videos (Runway). This is particularly useful for creative industries, where AI can assist in generating initial drafts or ideas.

- **Enhanced customer interactions:**

Generative AI can improve customer interactions through enhanced chat and search experiences. It can provide more human-like responses and even generate content based on user preferences.

- **Data exploration:**

Generative AI can explore vast amounts of unstructured data, providing insights through conversational interfaces and summarizations. This is especially beneficial for businesses looking to derive actionable insights from their data.

- **Repetitive tasks automation:**

Tasks like replying to requests for proposals (RFPs), localizing marketing content in multiple languages, and checking customer contracts for compliance can be automated using generative AI.

- **Innovations in various fields:**

Generative AI has found applications in drug and chip design, material science development, and more. Its ability to generate novel solutions based on existing data makes it a valuable tool for research and development.