# Week 1

## Dataset 1 - Heart Attack Analysis And Prediction

This Heart attack dataset is a survey among 1025 induviduals from ages ranging between 29 and 77 years. The main objective of this study is to see how susceptible an induvidual is to have an heart attack. Our Dataset has 13 attributes and 1 target class which is the likeliness of an induvidual getting an heart attack. The given data set is a completely numeric data set. Using this information , a machine learning model can be developed to find the probabilty of any individual suffering from an heart attack.

Link: https://heriotwatt.sharepoint.com/:x:/r/sites/DL/Shared%20Documents/DM%26%20ML/Heart%20Attack%20Analysis%20%26%20Prediction/heart.csv?d=w6d103271cf0e493e8aed526d710c709c&csf=1&web=1&e=O7ekFT

## Dataset 2 - Netflix Movies and TV Shows

Netflix is a production company based in United States of America in Los Gatos, California. It is a well known American paid subscription streaming service. Reed Hashtings ans Marc Ranolph founded the company on August 29, 1997. It offers a coolection of films and televisions through famous distribution deal and they have their own productions known as Netflix Originals

With the growing popularity of smart phones and Smart TVs, Netflix like any of the other streaming/broadcasting services can be easily accessed from tablets, laptops and any other smart device. All shows and movies can be viewed in 4K resolution as well. Initially netfix distributed Blu-Ray rentals and DVDs as well.

The indicated tabular dataset contains lists for all TV episodes, movies linked with Netflix along with the details about the performers and directors, as well as ratings and other aspects.

It consists of the following attributes:

- show id - each Film / Television how has its own ID
- type - Television Show / Film
- title - name of the Film / Television Show
- director - Film / Television Show director
- cast - peformers in the movie or television show
- country - the province in which the film or television show is set
- date_added - the date the title was added to Netflix
- release_year - actual release year for the Movie / Television Show
- rating - film / television show rating

- duration - number of seasons or timespan in minutes
- listed_in - the genre
- description - description of the Movie / TV Show

This is a nominal dataset. There are 8088 rows (records) and 12 columns

Link : https://heriotwatt.sharepoint.com/:x:/s/DL/EYFuXjqsXLtAnmeJy8UYzCoB9jKD-Xn1qh0GZph6-aT7CA?e=01W6S9,

## Dataset 3 - Vehicle Detection Image Set

This dataset contains images of vehicles taken from different angle i.e., taken from a car camera, surveillance camera,etc. There are two labels for the image set:

- Vehicles ( 8968 )
- Non-Vehicles ( 8792 )

This dataset has no vehicle type provided and consist of images taken under different lighting conditions and different time of the day.

Link : https://heriotwatt.sharepoint.com/:f:/s/DL/EqShixf5XbZHlsJ2GeV3C6MBuq_ZLI5Uvs4lhpHh3m6xow?e=y4BjWD

# Week 2

## Importing Libraries

```
# Common imports
import numpy as np
import os
import tarfile
import urllib
import pandas as pd
import urllib.request
import seaborn as sns
#import pandoc

import sys
assert sys.version_info >= (3, 5)
# Python ≥3.5 is required|

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score ,roc_curve, roc_auc_score
from sklearn import metrics
from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt

import matplotlib.pyplot as plt
from matplotlib.ticker import StrMethodFormatter
```

## Get the Data

```
#DOWNLOAD_ROOT = "https://github.com/SAL6910/DL/blob/main/heart.csv"
heart=
pd.read_csv("https://raw.githubusercontent.com/SAL6910/DL/main/heart.c
sv")
heart.head()
# To get the first 5 lines of the dataset

---------------------------------------------------------------------
-----
gaierror                                  Traceback (most recent call
last)
File ~\anaconda3\lib\urllib\request.py:1346, in
AbstractHTTPHandler.do_open(self, http_class, req, **http_conn_args)
   1345 try:
-> 1346     h.request(req.get_method(), req.selector, req.data,
headers,
   1347               encode_chunked=req.has_header('Transfer-
encoding'))
   1348 except OSError as err: # timeout error

File ~\anaconda3\lib\http\client.py:1285, in
HTTPConnection.request(self, method, url, body, headers,
encode_chunked)
   1284 """Send a complete request to the server."""
-> 1285 self._send_request(method, url, body, headers, encode_chunked)

File ~\anaconda3\lib\http\client.py:1331, in
HTTPConnection._send_request(self, method, url, body, headers,
encode_chunked)
   1330     body = _encode(body, 'body')
```

```
-> 1331 self.endheaders(body, encode_chunked=encode_chunked)

File ~\anaconda3\lib\http\client.py:1280, in
HTTPConnection.endheaders(self, message_body, encode_chunked)
   1279     raise CannotSendHeader()
-> 1280 self._send_output(message_body, encode_chunked=encode_chunked)

File ~\anaconda3\lib\http\client.py:1040, in
HTTPConnection._send_output(self, message_body, encode_chunked)
   1039 del self._buffer[:]
-> 1040 self.send(msg)
   1042 if message_body is not None:
   1043
   1044     # create a consistent interface to message_body

File ~\anaconda3\lib\http\client.py:980, in HTTPConnection.send(self,
data)
    979 if self.auto_open:
--> 980     self.connect()
    981 else:

File ~\anaconda3\lib\http\client.py:1447, in
HTTPSConnection.connect(self)
   1445 "Connect to a host on a given (SSL) port."
-> 1447 super().connect()
   1449 if self._tunnel_host:

File ~\anaconda3\lib\http\client.py:946, in
HTTPConnection.connect(self)
    945 """Connect to the host and port specified in __init__."""
--> 946 self.sock = self._create_connection(
    947     (self.host,self.port), self.timeout, self.source_address)
    948 # Might fail in OSs that don't implement TCP_NODELAY

File ~\anaconda3\lib\socket.py:823, in create_connection(address,
timeout, source_address)
    822 err = None
--> 823 for res in getaddrinfo(host, port, 0, SOCK_STREAM):
    824     af, socktype, proto, canonname, sa = res

File ~\anaconda3\lib\socket.py:954, in getaddrinfo(host, port, family,
type, proto, flags)
    953 addrlist = []
--> 954 for res in _socket.getaddrinfo(host, port, family, type,
proto, flags):
    955     af, socktype, proto, canonname, sa = res

gaierror: [Errno 11001] getaddrinfo failed
```

```
During handling of the above exception, another exception occurred:

URLError                                  Traceback (most recent call
last)
Input In [368], in <cell line: 2>()
      1 #DOWNLOAD_ROOT =
"https://github.com/SAL6910/DL/blob/main/heart.csv"
----> 2 heart=
pd.read_csv("https://raw.githubusercontent.com/SAL6910/DL/main/heart.c
sv")
      3 heart.head()

File ~\anaconda3\lib\site-packages\pandas\util\_decorators.py:311, in
deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*arg
s, **kwargs)
    305 if len(args) > num_allow_args:
    306     warnings.warn(
    307         msg.format(arguments=arguments),
    308         FutureWarning,
    309         stacklevel=stacklevel,
    310     )
--> 311 return func(*args, **kwargs)

File ~\anaconda3\lib\site-packages\pandas\io\parsers\readers.py:680,
in read_csv(filepath_or_buffer, sep, delimiter, header, names,
index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine,
converters, true_values, false_values, skipinitialspace, skiprows,
skipfooter, nrows, na_values, keep_default_na, na_filter, verbose,
skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col,
date_parser, dayfirst, cache_dates, iterator, chunksize, compression,
thousands, decimal, lineterminator, quotechar, quoting, doublequote,
escapechar, comment, encoding, encoding_errors, dialect,
error_bad_lines, warn_bad_lines, on_bad_lines, delim_whitespace,
low_memory, memory_map, float_precision, storage_options)
    665 kwds_defaults = _refine_defaults_read(
    666     dialect,
    667     delimiter,
   (...)
    676     defaults={"delimiter": ","},
    677 )
    678 kwds.update(kwds_defaults)
--> 680 return _read(filepath_or_buffer, kwds)

File ~\anaconda3\lib\site-packages\pandas\io\parsers\readers.py:575,
in _read(filepath_or_buffer, kwds)
    572 _validate_names(kwds.get("names", None))
    574 # Create the parser.
--> 575 parser = TextFileReader(filepath_or_buffer, **kwds)
    577 if chunksize or iterator:
    578     return parser
```

```
File ~\anaconda3\lib\site-packages\pandas\io\parsers\readers.py:933,
in TextFileReader.__init__(self, f, engine, **kwds)
    930     self.options["has_index_names"] = kwds["has_index_names"]
    932 self.handles: IOHandles | None = None
--> 933 self._engine = self._make_engine(f, self.engine)

File ~\anaconda3\lib\site-packages\pandas\io\parsers\readers.py:1217,
in TextFileReader._make_engine(self, f, engine)
   1213     mode = "rb"
   1214 # error: No overload variant of "get_handle" matches argument
types
   1215 # "Union[str, PathLike[str], ReadCsvBuffer[bytes],
ReadCsvBuffer[str]]"
   1216 # , "str", "bool", "Any", "Any", "Any", "Any", "Any"
-> 1217 self.handles = get_handle(  # type: ignore[call-overload]
   1218     f,
   1219     mode,
   1220     encoding=self.options.get("encoding", None),
   1221     compression=self.options.get("compression", None),
   1222     memory_map=self.options.get("memory_map", False),
   1223     is_text=is_text,
   1224     errors=self.options.get("encoding_errors", "strict"),
   1225     storage_options=self.options.get("storage_options", None),
   1226 )
   1227 assert self.handles is not None
   1228 f = self.handles.handle

File ~\anaconda3\lib\site-packages\pandas\io\common.py:670, in
get_handle(path_or_buf, mode, encoding, compression, memory_map,
is_text, errors, storage_options)
    667     codecs.lookup_error(errors)
    669 # open URLs
--> 670 ioargs = _get_filepath_or_buffer(
    671     path_or_buf,
    672     encoding=encoding,
    673     compression=compression,
    674     mode=mode,
    675     storage_options=storage_options,
    676 )
    678 handle = ioargs.filepath_or_buffer
    679 handles: list[BaseBuffer]

File ~\anaconda3\lib\site-packages\pandas\io\common.py:339, in
_get_filepath_or_buffer(filepath_or_buffer, encoding, compression,
mode, storage_options)
    337 # assuming storage_options is to be interpreted as headers
    338 req_info = urllib.request.Request(filepath_or_buffer,
headers=storage_options)
--> 339 with urlopen(req_info) as req:
```

```
    340     content_encoding = req.headers.get("Content-Encoding",
None)
    341     if content_encoding == "gzip":
    342         # Override compression based on Content-Encoding
header

File ~\anaconda3\lib\site-packages\pandas\io\common.py:239, in
urlopen(*args, **kwargs)
    233 """
    234 Lazy-import wrapper for stdlib urlopen, as that imports a big
chunk of
    235 the stdlib.
    236 """
    237 import urllib.request
--> 239 return urllib.request.urlopen(*args, **kwargs)

File ~\anaconda3\lib\urllib\request.py:214, in urlopen(url, data,
timeout, cafile, capath, cadefault, context)
    212 else:
    213     opener = _opener
--> 214 return opener.open(url, data, timeout)

File ~\anaconda3\lib\urllib\request.py:517, in
OpenerDirector.open(self, fullurl, data, timeout)
    514     req = meth(req)
    516 sys.audit('urllib.Request', req.full_url, req.data,
req.headers, req.get_method())
--> 517 response = self._open(req, data)
    519 # post-process response
    520 meth_name = protocol+"_response"

File ~\anaconda3\lib\urllib\request.py:534, in
OpenerDirector._open(self, req, data)
    531     return result
    533 protocol = req.type
--> 534 result = self._call_chain(self.handle_open, protocol, protocol
+
    535                           '_open', req)
    536 if result:
    537     return result

File ~\anaconda3\lib\urllib\request.py:494, in
OpenerDirector._call_chain(self, chain, kind, meth_name, *args)
    492 for handler in handlers:
    493     func = getattr(handler, meth_name)
--> 494     result = func(*args)
    495     if result is not None:
    496         return result

File ~\anaconda3\lib\urllib\request.py:1389, in
```

```
HTTPSHandler.https_open(self, req)
   1388 def https_open(self, req):
-> 1389     return self.do_open(http.client.HTTPSConnection, req,
   1390         context=self._context,
check_hostname=self._check_hostname)

File ~\anaconda3\lib\urllib\request.py:1349, in
AbstractHTTPHandler.do_open(self, http_class, req, **http_conn_args)
   1346         h.request(req.get_method(), req.selector, req.data,
headers,
   1347                   encode_chunked=req.has_header('Transfer-
encoding'))
   1348     except OSError as err: # timeout error
-> 1349         raise URLError(err)
   1350     r = h.getresponse()
   1351 except:

URLError: <urlopen error [Errno 11001] getaddrinfo failed>
```

```
heart.info()
```

```
heart["age"].value_counts()
# Number of times the first and last five values in column 'age' has
been repeated
```

```
heart.describe()
```

```
heart_null = heart[heart.isnull().any(axis=1)].head()
heart_null
# display rows with missing values
# axis=1 (represents columns) , axis=0 (represents rows)

# The data does not contain any null values, hence it requires no
further cleaning
```

## Edited Dataset with Null Values

```
# heart 2
```

```
heart2=
pd.read_csv("https://raw.githubusercontent.com/SAL6910/DL/main/heart2.
csv")
heart2.head(12)
```

```
heart2_null = heart2[heart2.isnull().any(axis=1)]
heart2_null
# display rows with missing values
# axis=1 (represents columns) , axis=0 (represents rows)
```

```
heart2.iloc[5:12,0:33]
# iloc commands is used to print a specific range of data
```

```
heart2_null.dropna()
# Helps to remove all values with NUll

heart2.dropna()
# shows the original dataset without null values

median = heart2["age"].median()
heart2["age"].fillna(median, inplace=True)
heart2.head(15)

#df[['col1', 'col2']] = df[['col1', 'col2']].fillna(df[['col1',
'col2']].median())
# to fill up the null values with the median of every coloumn

heart2
# To view the initial edited dataset complete with median values in
place of Null Values (no empty records)
# for example column 4 of 1991 is '5.41'

# Cleaning the edited Data set with null values are done
```

## Plotting Histogram

```
#To plot a histogram for each numerical attribute
heart.hist(bins=15, figsize=(20,15), color='g', edgecolor='k')
# plt.xlabel ('Topic title rate')
# plt.ylabel('Frequency')
plt.show()

# y-axis is the Frequency
# Bins is group count of values in a column
```

## Visualizing Data

```
#Data Visualization
heart.plot(kind="scatter", x="age", y="trestbps", color='g')

heart.plot(kind="scatter", x="age", y="trestbps", alpha=0.1)
# alpha dictates the gradient of the scatter plot ( 1,0.5 or 0.25.)
```

## Week 3

## Correlation

```
# Correlation of all the attributes with the expected class attribute
(target)
corr_matrix = heart.corr() # computes the standard correlation
coefficient (Pearson’s r) between every pair of attributes
```

```
corr_matrix["target"].sort_values(ascending=False)
# Listing the highest correlated attributes in acending arder

from pandas.plotting import scatter_matrix

attributes = ["target", "cp", "oldpeak",
              "exang","thalach"]
# Picked the top 2 and inversly top 2 attributes and visualizing
against target
scatter_matrix(heart[attributes], figsize=(12, 8), alpha=0.1)

plt.figure(figsize=(11, 11))
corr_heatmap = sns.heatmap(heart.corr(),annot=True,cmap="BuPu",
linewidths = 1.0)
# To plot the heat map of thr correlated values
```

## Week 4

## Prediction with Logistic Regression

Trying with top 6 attributes

```
corr_attributes =  ['target','age', 'sex','oldpeak', 'slope',
'ca','cp', 'thalach', 'exang']
# Selecting only the important attributes with high correlation

new_heart = heart[corr_attributes]
# Making a new dataset with the corelated attributes

X = new_heart.drop(['target'], axis = 1)
y = new_heart[['target']]
# X refers to al our input parameters, y refers to the target class

X.head()

y.head(10)

X_train, X_test, y_train, y_test = train_test_split(X,y, train_size =
0.7, random_state = 42)
# Splitting the dataset to 7:3 ratio for training and testing
# Train_size parameter defines the size percent for training and
testing
# Random_state parameter defines the rate of shuffle between the
dataset to increase accuracy

X_train

y_train = np.ravel(y_train)
y_train
# To print the values in an array
```

```
heart_classifier=LogisticRegression(max_iter=500)
heart_classifier.fit(X_train,y_train)
y_predict=heart_classifier.predict(X_test)

# Using Logistic Regression as a model with the iteration of 500

print(classification_report(y_test, y_predict))
print()
print('Accuracy', accuracy_score(y_test, y_predict))
```

Trying with top 4 attributes attributes

```
corr_attributes =  [ 'target','age', 'sex','oldpeak', 'cp', 'thalach',
'exang']
# Selecting only the important attributes with high correlation

new_heart = heart[corr_attributes]
# Making a new dataset with the corelated attributes

X = new_heart.drop(['target'], axis = 1)
y = new_heart[['target']]
# X refers to al our input parameters, y refers to the target class

X.head()

y.head(10)

X_train, X_test, y_train, y_test = train_test_split(X,y, train_size =
0.7, random_state = 42)

# Splitting the dataset to 7:3 ratio for training and testing
# Train_size parameter defines the size percent for training and
testing
# Random_state parameter defines the rate of shuffle between the
dataset to increase accuracy

X_train

y_train = np.ravel(y_train)
y_train
# To print the values in an array

heart_classifier=LogisticRegression(max_iter=500)
heart_classifier.fit(X_train,y_train)
y_predict=heart_classifier.predict(X_test)

# Using Logistic Regression as a model with the iteration of 500

print(classification_report(y_test, y_predict))
print()
print('Accuracy', accuracy_score(y_test,y_predict))
```

Trying with top 2 attributes

```
corr_attributes =  [ 'target','age', 'sex','oldpeak', 'cp',]
# Selecting only the important attributes with high correlation

new_heart = heart[corr_attributes]
# Making a new dataset with the corelated attributes

X = new_heart.drop(['target'], axis = 1)
y = new_heart[['target']]
# X refers to al our input parameters, y refers to the target class

X.head()

y.head(10)

X_train, X_test, y_train, y_test = train_test_split(X,y, train_size =
0.7, random_state = 42)
# Splitting the dataset to 7:3 ratio for training and testing
# Train_size parameter defines the size percent for training and
testing
# Random_state parameter defines the rate of shuffle between the
dataset to increase accuracy

X_train

y_train = np.ravel(y_train)
y_train
# To print the values in an array

heart_classifier=LogisticRegression(max_iter=500)
heart_classifier.fit(X_train,y_train)
y_predict=heart_classifier.predict(X_test)

# Using Logistic Regression as a model with the iteration of 500

print(classification_report(y_test, y_predict))
print()
print('Accuracy', accuracy_score(y_test, y_predict))
```

Trying with top 8 attributes

```
corr_attributes =  ['target','age', 'sex','oldpeak', 'slope',
'ca','cp', 'thalach', 'exang', 'restecg', 'thal']
# Selecting only the important attributes with high correlation

new_heart = heart[corr_attributes]
# Making a new dataset with the corelated attributes

X = new_heart.drop(['target'], axis = 1)
y = new_heart[['target']]
# X refers to al our input parameters, y refers to the target class

X.head()
```

```
y.head(10)

X_train, X_test, y_train, y_test = train_test_split(X,y, train_size =
0.7, random_state = 42)
# Splitting the dataset to 7:3 ratio for training and testing
# Train_size parameter defines the size percent for training and
testing
# Random_state parameter defines the rate of shuffle between the
dataset to increase accuracy

X_train

y_train = np.ravel(y_train)
y_train
# To print the values in an array

heart_classifier=LogisticRegression(max_iter=500)
heart_classifier.fit(X_train,y_train)
y_predict=heart_classifier.predict(X_test)

# Using Logistic Regression as a model with the iteration of 500

print(classification_report(y_test, y_predict))
print()
print('Accuracy', accuracy_score(y_test, y_predict))
```

You can see by taking 8 attributes the accuracy percentage has increased to 0.83%

```
# Plotting the confussion matrix between y_prect and y_test
conf_matrix = confusion_matrix(y_test, y_predict)

plt.figure(figsize=(6, 6))
sns.heatmap(conf_matrix, annot=True,fmt='d', cmap='coolwarm')  # Fmt
represents the type of text (d = decimal)
plt.ylabel("y_test", fontsize=20)
plt.xlabel("y_predict", fontsize=20)
plt.title("Confusion Matrix", fontsize=25)
plt.show()

true_pos, false_pos, Thresh =roc_curve(y_test, y_predict)

plt.plot(true_pos, false_pos)
plt.title('ROC Curve')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

print('area under ROC curve is', roc_auc_score(y_test, y_predict))
# Map to display the relationship between TP rate and FP rate
# The higher the Tp rate the better
```

# Week 5

## Prediction with Multinominal Naive Bayes

```
from sklearn.datasets import load_iris
from sklearn.naive_bayes import MultinomialNB

mnb = MultinomialNB()
y_predict = mnb.fit(X_train, y_train).predict(X_test)

print(classification_report(y_test, y_predict))
print()
print('Accuracy', accuracy_score(y_test, y_predict))


conf_matrix = confusion_matrix(y_test, y_predict)

plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix, annot=True,fmt='d', cmap='coolwarm')
plt.ylabel("y_test", fontsize=20)
plt.xlabel("y_predict", fontsize=20)
plt.title("Confusion Matrix", fontsize=25)
plt.show()

true_pos, false_pos, Thresh =roc_curve(y_test, y_predict)

plt.plot(true_pos, false_pos)
plt.title('ROC Curve')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

print('area under ROC curve is', roc_auc_score(y_test, y_predict))
```

## Prediction with Gaussian Naive Bayes

```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
y_predict = gnb.fit(X_train, y_train).predict(X_test)

print(classification_report(y_test, y_predict))
print()
print('Accuracy', accuracy_score(y_test, y_predict))

conf_matrix = confusion_matrix(y_test, y_predict)

plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix, annot=True,fmt='d', cmap='coolwarm')
plt.ylabel("y_test", fontsize=20)
```

```
plt.xlabel("y_predict", fontsize=20)
plt.title("Confusion Matrix", fontsize=25)
plt.show()

true_pos, false_pos, Thresh =roc_curve(y_test, y_predict)

plt.plot(true_pos, false_pos)
plt.title('ROC Curve')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

print('area under ROC curve is', roc_auc_score(y_test, y_predict))
```

## Prediction with Complement Naive Bayes

```
from sklearn.naive_bayes import ComplementNB

cnb = ComplementNB()
y_predict = cnb.fit(X_train, y_train).predict(X_test)

print(classification_report(y_test, y_predict))
print()
print('Accuracy', accuracy_score(y_test, y_predict))

conf_matrix = confusion_matrix(y_test, y_predict)

plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix, annot=True,fmt='d', cmap='coolwarm')
plt.ylabel("y_test", fontsize=20)
plt.xlabel("y_predict", fontsize=20)
plt.title("Confusion Matrix", fontsize=25)
plt.show()

true_pos, false_pos, Thresh =roc_curve(y_test, y_predict)

plt.plot(true_pos, false_pos)
plt.title('ROC Curve')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

print('area under ROC curve is', roc_auc_score(y_test, y_predict))
```

## Prediction with Bernouli Naive Bayes

```
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB()
y_predict = bnb.fit(X_train, y_train).predict(X_test)
```

```
print(classification_report(y_test, y_predict))
print()
print('Accuracy', accuracy_score(y_test, y_predict))

conf_matrix = confusion_matrix(y_test, y_predict)

plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix, annot=True,fmt='d', cmap='coolwarm')
plt.ylabel("y_test", fontsize=20)
plt.xlabel("y_predict", fontsize=20)
plt.title("Confusion Matrix", fontsize=25)
plt.show()

true_pos, false_pos, Thresh =roc_curve(y_test, y_predict)

plt.plot(true_pos, false_pos)
plt.title('ROC Curve')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

print('area under ROC curve is', roc_auc_score(y_test, y_predict))
```

## Prediction with Categorical Naive Bayes

```
from sklearn.naive_bayes import CategoricalNB

ctnb =CategoricalNB()
y_predict = ctnb.fit(X_train, y_train).predict(X_test)
print(classification_report(y_test, y_predict))
print()
print('Accuracy', accuracy_score(y_test, y_predict))

conf_matrix = confusion_matrix(y_test, y_predict)

plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix, annot=True,fmt='d', cmap='coolwarm')
plt.ylabel("y_test", fontsize=20)
plt.xlabel("y_predict", fontsize=20)
plt.title("Confusion Matrix", fontsize=25)
plt.show()

true_pos, false_pos, Thresh =roc_curve(y_test, y_predict)

plt.plot(true_pos, false_pos)
plt.title('ROC Curve')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
print('area under ROC curve is', roc_auc_score(y_test, y_predict))
```

## Comparison between different Naive Bayes Algorithms

| Naïve Bayes Algoithm | Accuracy | TP | FP | TN | FN | Sensitivity (TP/(TP+FN)) | Specificity (TN/(TN+FP)) | Precission (TP/(TP+FP)) | Recall (TP/(TP+FN)) | Area under ROC Curve |
|---|---|---|---|---|---|---|---|---|---|---|
| Multinomial Naive Bayes | 0.731 | 80 | 32 | 70 | 23 | 0.776699029 | 0.68627451 | 0.714285714 | 0.776699029 | 0.731 |
| Gaussian Naive Bayes | 0.79 | 89 | 29 | 73 | 14 | **0.86407767** | 0.715686275 | 0.754237288 | **0.86407767** | 0.7898 |
| Complement Naive Baye | 0.731 | 80 | 32 | 70 | 23 | 0.776699029 | 0.68627451 | 0.714285714 | 0.776699029 | 0.731 |
| Bernoulli Naive Bayes | **0.814** | 89 | 24 | 78 | 14 | **0.86407767** | 0.764705882 | 0.787610619 | **0.86407767** | 0.81439 |
| Categorical Naive Bayes | **0.814** | 88 | 23 | 79 | 15 | 0.854368932 | **0.774509804** | **0.792792793** | 0.854368932 | **0.8144** |

## Week 7

## K- Means Clustering

```
from sklearn.cluster import KMeans

k = 5 # number of clusters (default)
kmeans = KMeans(n_clusters=k, random_state=42).fit(X_train)
y_predict = kmeans.fit_predict(new_heart[['target','cp']])

# Accuracy of clusters
kmeans.score(new_heart[['target','cp']])

y_predict

y_predict is kmeans.labels_

# Predicts the closest cluster each sample in X belongs to.
kmeans.cluster_centers_

kmeans.labels_
```

## Gaussian Mixture Model

```
# checks whether a cluster contains only samples belonging to a single
class
from sklearn.metrics import homogeneity_score
from sklearn.mixture import GaussianMixture as EM

em = EM(n_components = 2, max_iter = 50, n_init = 1).fit(X_train)
y_predict = em.predict(X_train)
print(homogeneity_score(y_train, y_predict))
```

## Elbow Method

This method helps to decide on an optimal value for k i.e., the number of clusters

```
kmeans_per_k = [KMeans(n_clusters=k, random_state=42).fit(X_train)
                for k in range(1, 10)]
inertias = [model.inertia_ for model in kmeans_per_k]

# It is calculated by measuring the distance between each data point
# And its centroid, squaring this distance, and summing these squares
across one cluster.
inertias

plt.figure(figsize=(8, 3.5))
plt.plot(range(1, 10), inertias, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Inertia", fontsize=14)
plt.annotate('Elbow',
             xy=(2, inertias[1]),
             xytext=(0.2, 0.2),
             textcoords='figure fraction',
             fontsize=16,
             arrowprops=dict(facecolor='black', shrink=0.1)
            )

plt.show()
```

From the above graph k is best taken as 2

```
# Updated k value
k = 2 # number of clusters (default)
kmeans = KMeans(n_clusters=k, random_state=42).fit(X_train)
y_predict = kmeans.fit_predict(new_heart[['target','cp']])

kmeans.score(new_heart[['target','cp']])

y_predict

y_predict is kmeans.labels_

kmeans.cluster_centers_

kmeans.labels_
```

## Week 8

## Decision Tree

```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import os

# Display the first five rows of the dataset
heart.head(5)

from sklearn.model_selection import RandomizedSearchCV,
cross_val_score
from scipy.stats import randint
from sklearn.tree import export_text
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier,plot_tree

decision_tree = DecisionTreeClassifier(random_state=0, max_depth=3,
criterion="gini")
decision_tree.fit(X_train, y_train)
y_predict = decision_tree.predict(X_test)
X_predict = decision_tree.predict(X_train)
print("Testing Accuracy: ",accuracy_score(y_test, y_predict))
print("Training Accuracy: ",accuracy_score(y_train,X_predict))

plt.figure(figsize=(70,50))
plot_tree(decision_tree)
```

## 10-Fold Cross Validation

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(decision_tree, X_train, y_train, cv=10)
scores

print("%0.2f accuracy with a standard deviation of %0.2f" %
(scores.mean(), scores.std()))

# Precision measure
from sklearn.metrics import precision_score, recall_score

precision_score(y_test, y_predict)

# Recall score
recall_score(y_test, y_predict)

# F measure
from sklearn.metrics import f1_score

f1_score(y_test, y_predict)

# ROC area
from sklearn.metrics import roc_auc_score
```

```
roc_auc = roc_auc_score(y_test, y_predict)
roc_auc
```

## Changing Parameter

```
decision_tree = DecisionTreeClassifier(max_depth=4, splitter="best",
min_samples_leaf=2)
decision_tree.fit(X_train, y_train)
y_predict = decision_tree.predict(X_test)
predx = decision_tree.predict(X_train)
print("Testing Accuracy: ",accuracy_score(y_test,y_predict))
print("Training Accuracy: ",accuracy_score(y_train,predx))
```

```
decision_tree = DecisionTreeClassifier(max_depth=4, splitter="best",
min_samples_leaf=6)
decision_tree.fit(X_train, y_train)
y_predict = decision_tree.predict(X_test)
predx = decision_tree.predict(X_train)
print("Testing Accuracy: ",accuracy_score(y_test, y_predict))
print("Training Accuracy: ",accuracy_score(y_train,predx))
```

```
decision_tree = DecisionTreeClassifier(max_depth=2, splitter="best",
min_samples_leaf=8)
decision_tree.fit(X_train, y_train)
y_predict = decision_tree.predict(X_test)
predx = decision_tree.predict(X_train)
print("Testing Accuracy: ",accuracy_score(y_test, y_predict))
print("Training Accuracy: ",accuracy_score(y_train,predx))
```

```
decision_tree = DecisionTreeClassifier(max_depth=10, splitter="best",
min_samples_leaf=8)
decision_tree.fit(X_train, y_train)
y_predict = decision_tree.predict(X_test)
predx = decision_tree.predict(X_train)
print("Testing Accuracy: ",accuracy_score(y_test, y_predict))
print("Training Accuracy: ",accuracy_score(y_train,predx))
```

```
decision_tree = DecisionTreeClassifier(max_depth=10,
splitter="random", min_samples_leaf=8)
decision_tree.fit(X_train, y_train)
y_predict = decision_tree.predict(X_test)
predx = decision_tree.predict(X_train)
print("Testing Accuracy: ",accuracy_score(y_test, y_predict))
print("Training Accuracy: ",accuracy_score(y_train,predx))
```

```
decision_tree = DecisionTreeClassifier(max_depth=10, splitter="best",
min_samples_leaf=15)
decision_tree.fit(X_train, y_train)
y_predict= decision_tree.predict(X_test)
predx = decision_tree.predict(X_train)
```

```
print("Testing Accuracy: ",accuracy_score(y_test, y_predict))
print("Training Accuracy: ",accuracy_score(y_train,predx))

decision_tree = DecisionTreeClassifier(max_depth=15, splitter="best",
min_samples_leaf=4)
decision_tree.fit(X_train, y_train)
y_predict = decision_tree.predict(X_test)
predx = decision_tree.predict(X_train)
print("Testing Accuracy: ",accuracy_score(y_test,y_predict))
print("Training Accuracy: ",accuracy_score(y_train,predx))
```

## Visual Conclusion

| Parameter Change Results | | | | | |
|---|---|---|---|---|---|
| **SL.no** | **max_depth** | **splitter** | **min_samples_leaf** | **Testing Accuracy** | **Training Accuracy** |
| 1 | 4 | best | 2 | 0.827922078 | 0.891213389 |
| 2 | 4 | best | 6 | 0.814935065 | 0.881450488 |
| 3 | 2 | best | 8 | 0.701298701 | 0.772663877 |
| 4 | 10 | best | 8 | 0.883116883 | 0.914923291 |
| 5 | 10 | random | 8 | 0.873376623 | 0.906555091 |
| 6 | 10 | best | 15 | 0.840909091 | 0.877266388 |
| 7 | 15 | best | 4 | 0.915584416 | 0.960948396 |

## Tree

```
X_train2, X_test2, y_train2, y_test2 = train_test_split(X,y,
train_size = 0.7, random_state = 42)

# Create Decision Tree classifer object
clf = DecisionTreeClassifier(max_depth=15, splitter="best",
min_samples_leaf=4)

# Train Decision Tree Classifer
clf = clf.fit(X_train2,y_train2)

#Predict the response for test dataset
y_pred2 = clf.predict(X_test2)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test2, y_pred2))
```

# Week 9

## Linear and Logistic Regression

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
lin_reg.intercept_, lin_reg.coef_

from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

linear_clf =
make_pipeline(StandardScaler(),SGDClassifier(max_iter=20000))
linear_clf.fit(X_train, y_train)

pred = linear_clf.predict(X_test)
predx = linear_clf.predict(X_train)

print("Testing Accuracy: ",accuracy_score(y_test, pred))
print("Training Accuracy: ",accuracy_score(y_train,predx))

from sklearn.model_selection import cross_val_score
scores = cross_val_score(linear_clf, X_train, y_train, cv=10)
scores
```

| | No. of Iterations | Testing Accuracy | Training Accuracy |
|---|---|---|---|
| 1 | 1000 | 0.795454545 | 0.80195258 |
| 2 | 2000 | 0.779220779 | 0.806136681 |
| 3 | 5000 | 0.788961039 | 0.817294282 |
| 4 | 10000 | 0.75974026 | 0.783821478 |
| 5 | 15000 | 0.824675325 | 0.854951185 |
| 6 | 20000 | 0.808441558 | 0.829846583 |


# Week 10

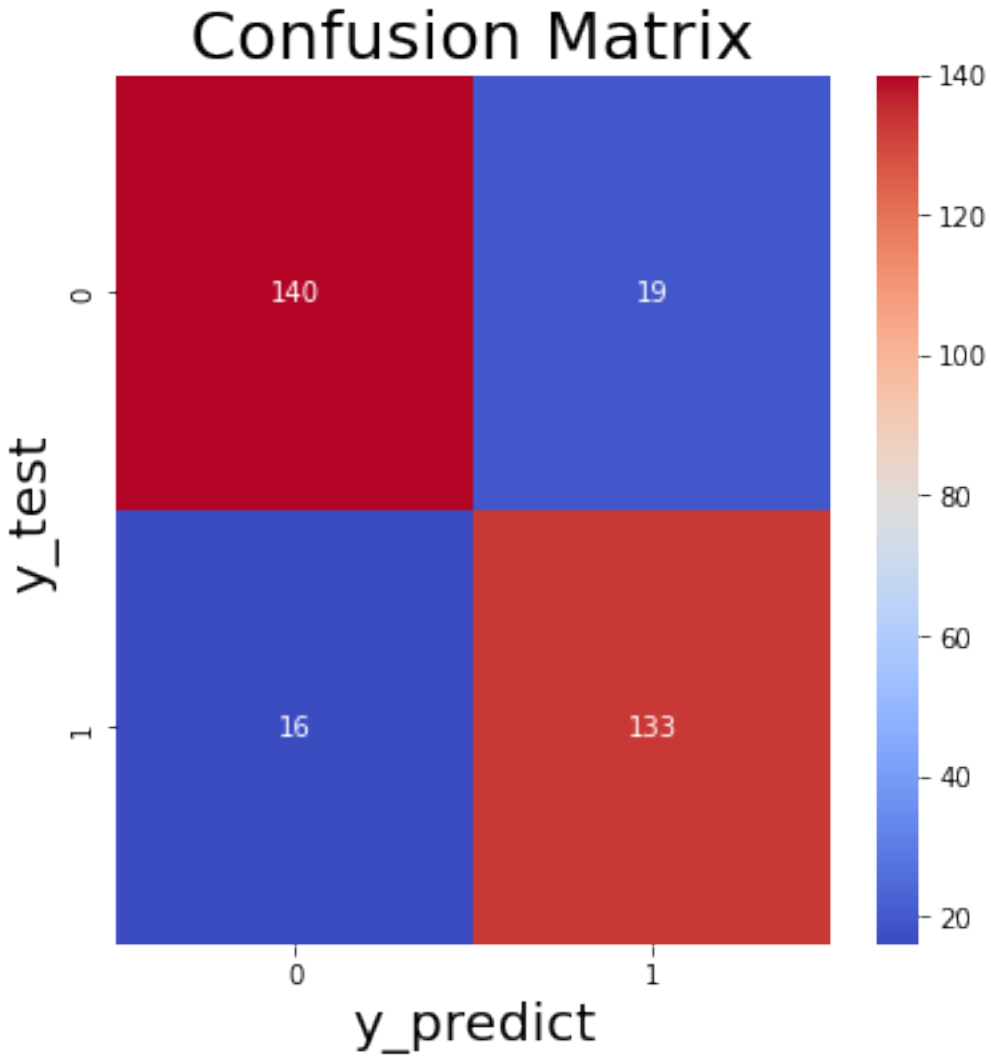
## Classification using MLP and perceptron

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification

y_predict =
MLPClassifier(random_state=1,hidden_layer_sizes=(20,),solver='adam',ac
tivation ='logistic',learning_rate_init=0.001,
max_iter=5000).fit(X_train, y_train).predict(X_test)

print(classification_report(y_test, y_predict))
print()
print('Accuracy', accuracy_score(y_test, y_predict))
```

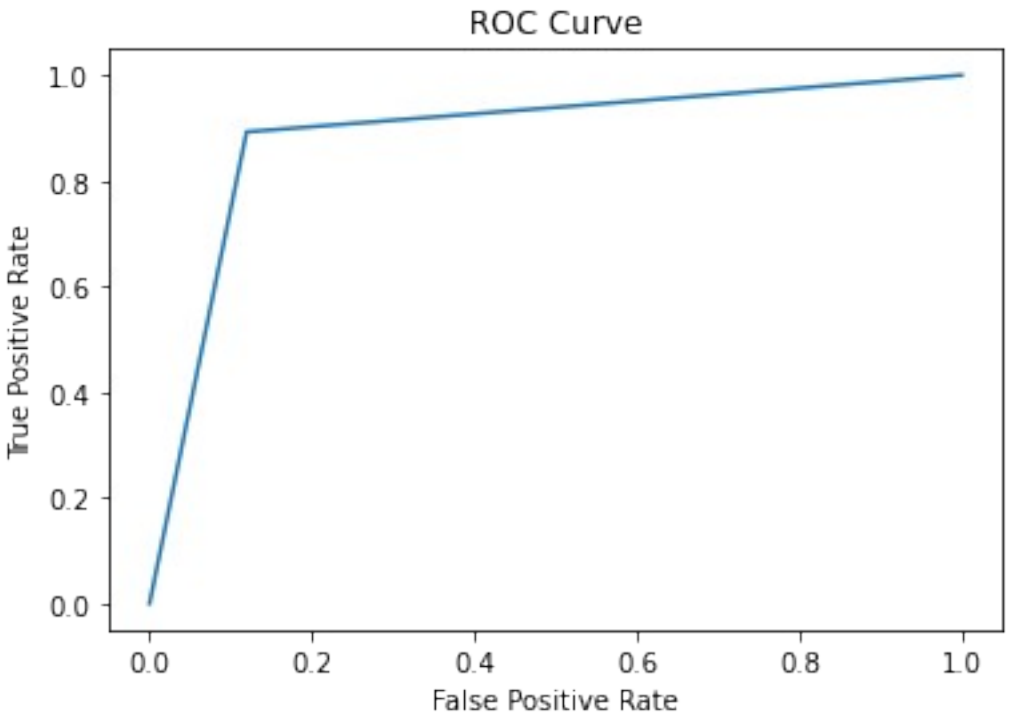
```
              precision    recall  f1-score   support

           0       0.90      0.88      0.89       159
           1       0.88      0.89      0.88       149

    accuracy                           0.89       308
   macro avg       0.89      0.89      0.89       308
weighted avg       0.89      0.89      0.89       308


Accuracy 0.8863636363636364
```

```
conf_matrix = confusion_matrix(y_test, y_predict)

plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix, annot=True,fmt='d', cmap='coolwarm')
```

```
plt.ylabel("y_test", fontsize=20)
plt.xlabel("y_predict", fontsize=20)
plt.title("Confusion Matrix", fontsize=25)
plt.show()
```




```
true_pos, false_pos, Thresh =roc_curve(y_test, y_predict)

plt.plot(true_pos, false_pos)
plt.title('ROC Curve')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

print('area under ROC curve is', roc_auc_score(y_test, y_predict))
```




area under ROC curve is 0.8865602971592589

| Multilayer Perceptron | Accuracy | TP | FP | TN | FN | Sensitivity | Specificity | Precision | Recall | AUC |
|---|---|---|---|---|---|---|---|---|---|---|
| Architecture 1 (Layers = 20, learning rate η =0.001, No of iterations = 300, optimisation algorithm= 'adam', activation functions = 'relu') | 82.46 | 139 | 44 | 115 | 10 | 0.932885906 | 0.72327044 | 0.759562842 | 0.932885906 | 0.82807 |
| Architecture 2(Layers = 20, learning rate η =0.5, No of iterations = 300, optimisation algorithm= 'adam', activation functions = 'relu') | 48.37 | 149 | 159 | 0 | 0 | 1 | 0 | 0.483766234 | 1 | 0.5 |
| Architecture 3 (Layers = 20, learning rate η =0.001, No of iterations = 300, optimisation algorithm= 'adam', activation functions = 'logistic') | 85.38 | 138 | 34 | 125 | 11 | 0.926174497 | 0.786163522 | 0.802325581 | 0.926174497 | 0.856 |
| Architecture 4 (Layers = 20, learning rate η =0.001, No of iterations = 5000, optimisation algorithm= 'adam', activation functions = 'logistic') | 88.63 | 133 | 19 | 140 | 16 | 0.89261745 | 0.880503145 | 0.875 | 0.89261745 | 0.886 |
| Architecture 5 (Layers = 5, learning rate η =0.001, No of iterations = 5000, optimisation algorithm= 'adam', activation functions = 'logistic') | 82.79 | 133 | 37 | 122 | 16 | 0.89261745 | 0.767295597 | 0.782352941 | 0.89261745 | 0.829 |
| Architecture 6 (Layers = 20, learning rate η =0.001, No of iterations = 5000, optimisation algorithm= 'sgd', activation functions = 'logistic') | 78.57 | 132 | 49 | 110 | 17 | 0.88590604 | 0.691823899 | 0.729281768 | 0.88590604 | 0.788 |