|  |  |
| --- | --- |
| **Ex. No: 1** | **Regression** |
| **Date: 30-12-2022** |

# Aim:

To build a multiple linear regression model to predict the sale price of the house.

# Algorithm:

**Multiple Linear Regression:**

* Import the necessary modules and the csv file.
* Fill the null values present in the data and pre-process the data.
* Use variance influence factor in the data and check for correlation and select the columns that have variance influence factor lesser than 3.
* Split the data into train and test using train-test-split from sklearn library.
* Import Linear regression from sklearn and fit the model.
* Predict the values using the test data and print the accuracy metrics.

# Simple Linear Regression:

* Import the necessary modules and the csv file.
* Fix the dependent variable and independent variable calculate the slope and y-intercept using the mathematical formula calculate the variances of dependent and independent variable.
* Calculate the root mean square of the predicted values and find the line of best fit.

# Program:

import numpy as np import pandas as pd

import matplotlib.pyplot as plt df=pd.read\_csv('house\_pred.csv') df.head()

df.info() df.describe()

columns\_numerical=df.select\_dtypes(exclude='object') column\_numerical=columns\_numerical.columns column\_numerical

columns\_object=df.select\_dtypes(include='object') column\_object=columns\_object.columns column\_object

columns\_numerical.info()

df['LotArea'].mean()

df['LotArea'].median()

columns\_numerical['LotFrontage'].fillna(value=columns\_numerical['LotFront age'].mean(),inplace=True)

columns\_numerical['MasVnrArea'].fillna(value=columns\_numerical['MasVnrAre a'].mean(),inplace=True)

columns\_numerical['GarageYrBlt'].fillna(value=columns\_numerical['GarageYr Blt'].median(),inplace=True)

columns\_numerical.info() columns\_object.info()

columns\_object.drop(['Alley','FireplaceQu','PoolQC','Fence','MiscFeature'

],inplace=True,axis=1) df1=columns\_object.isna().sum() df1

columns\_object.fillna(method='ffill',inplace=True) columns\_object.columns df\_uni=columns\_object.describe(include='object').T

def dummies(x): a=pd.get\_dummies(columns\_object[x],drop\_first=True) return a

column\_object=list(columns\_object.columns) len(column\_object) columns\_object.describe(include='object').T

a=pd.DataFrame()

for i in column\_object: c=dummies(i) a=pd.concat([a,c],axis=1)

#c.drop\_duplicates(inplace=True)

#l=a.columns[a.columns.duplicated()] #s=list(a.columns.drop\_duplicates())

a=a.loc[:,~a.columns.duplicated()] len(l)

len(s) df\_final=a.join(columns\_numerical) df\_final

def iqr(x):

q1=df\_final[x].quantile(0.25) q3=df\_final[x].quantile(0.75) iqr=q3-q1

upper=q3+1.5\*iqr lower=q1-1.5\*iqr print(i) print(upper,lower) return upper,lower

df\_f=df\_final.copy()

for i in column\_numerical: a,b=iqr(i)

df\_f=df\_f[(df\_f[i]>b) | (df\_f[i]<a)] df\_f plt.boxplot(df\_final['EnclosedPorch'])

for i in column\_numerical: plt.boxplot(df\_final[i])

import seaborn as sns

sns.boxplot(columns\_numerical)

plt.figure(figsize=(8,250)) c=1

for i in columns\_numerical: ax=plt.subplot(38,1,c)

sns.boxplot(columns\_numerical[i],orient='h',x=columns\_numerical[i]) c+=1

plt.show() df\_final

x=df\_final.iloc[:,:-1].values y=df\_final.iloc[:,-1].values x.shape

x=pd.DataFrame(x) x.columns=df\_final.columns[:-1] x['intercept']=1

vif=pd.DataFrame() vif['variables']=x.columns

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor vif['values']=[variance\_inflation\_factor(x.values,i) for i in range(x.shape[1])]

vif e=vif[vif['values']>3]

out=e['variables'].values[:-1] df\_final.drop(out,axis=1,inplace=True)

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_s tate=42)

from sklearn.linear\_model import LinearRegression reg=LinearRegression()

reg.fit(x\_train,y\_train)

y\_pred=reg.predict(x\_test) from sklearn.metrics import

r2\_score,mean\_absolute\_error,mean\_squared\_error print("R2 score:",r2\_score(y\_pred,y\_test))

print("Mean squared error:",mean\_squared\_error(y\_pred,y\_test)) print("Mean absolute error:",mean\_absolute\_error(y\_pred,y\_test))

x=x.values x

df\_test=pd.read\_csv('test.csv') df\_test.info()

df\_test.drop(['Alley','FireplaceQu','PoolQC','Fence','MiscFeature'],axis= 1,inplace=True)

df\_test\_num=df\_test.select\_dtypes(exclude='object') df\_test\_obj=df\_test.select\_dtypes(include='object')

df\_test\_num.info()

df\_test\_num['LotFrontage'].fillna(value=columns\_numerical['LotFrontage']. mean(),inplace=True) df\_test\_num['MasVnrArea'].fillna(value=columns\_numerical['MasVnrArea'].me an(),inplace=True) df\_test\_num['GarageYrBlt'].fillna(value=columns\_numerical['GarageYrBlt']. median(),inplace=True)

df\_test\_obj.describe().T

df\_test\_obj.fillna(method='ffill',inplace=True) def dummies(x):

a=pd.get\_dummies(df\_test\_obj[x],drop\_first=True) return a

b=pd.DataFrame()

for i in column\_object: c=dummies(i) b=pd.concat([b,c],axis=1)

#c.drop\_duplicates(inplace=True)

l=b.columns[b.columns.duplicated()] s=list(b.columns.drop\_duplicates()) b=b.loc[:,~b.columns.duplicated()]

len(b.columns) b=b.join(df\_test\_num) len(b.columns) w=vif[vif['values']<3] w=w['variables'].values

for i in w:

if i not in b.columns: b[i]=0

test\_x=b[w] test\_x.columns=x\_train.columns test\_x['intercept']=1

test\_x

x\_train house\_price\_predict=reg.predict(test\_x) test\_x['SalePrice']=house\_price\_predict test\_x.head()

df\_test['SalePrice']=house\_price\_predict df\_test.to\_csv('Predicted House Price.csv')

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

import math df=pd.read\_csv('data1.csv') df.head()

x=df['x'].values

y=df['y'].values mean\_x=np.mean(x) mean\_y=np.mean(y) r=f=d=0

f=((x-mean\_x)\*(y-mean\_y)).sum()

d=(np.sqrt(((x-mean\_x)\*\*2)\*((y-mean\_y)\*\*2))).sum() r=f/d

S\_y=((np.sqrt((y-mean\_y)\*\*2))).sum()/(len(y)-1)

S\_x=((np.sqrt((x-mean\_x)\*\*2))).sum()/(len(x)-1) b1=r\*(S\_y/S\_x)

b1

b0=mean\_y-b1\*mean\_x b0

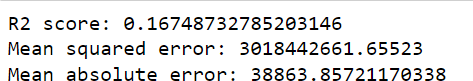
r

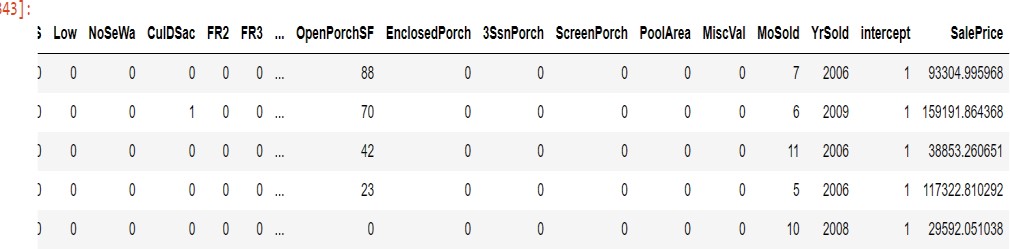
rmse=math.sqrt((1/len(y))\*((y-mean\_y)\*\*2).sum()) rmse

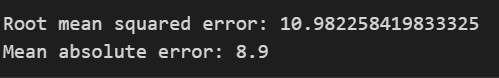
mae=(abs(y-mean\_y).sum())/len(y) mae

print("Root mean squared error:",rmse) print("Mean absolute error:",mae)

# Output Screenshot:







**Result:**

Hence, the multiple regression model is built and the sale price of the house is predicted and a simple linear regression model is built.

|  |  |
| --- | --- |
| **Ex. No: 2** | **Logistic Regression** |
| **Date: 06-01-2023** |

# Aim:

To build a logistic regression model to classify customer status.

# Algorithm:

* Import the necessary modules and the csv file.
* Fill the null values present in the data and pre-process the data.
* Split the data into train and test using train-test-split from sklearn library.
* Using the standard scaler normalize the train data and the test data.
* Import Logistic regression from sklearn and fit the model.
* Predict the values using the test data and print the confusion matrix and classification report.

# Program:

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

df=pd.read\_csv('telecom\_customer\_churn.csv') df.head()

df.info()

df.drop(['Churn Category','Churn Reason'],axis=1,inplace=True) obj\_col=df.select\_dtypes(include='object')

for i in obj\_col: print(i,":",obj\_col[i].unique())

obj\_col.drop(['Customer ID','City'],axis=1,inplace=True) obj\_col['Offer'].value\_counts().index[0]

for i in obj\_col: obj\_col[i].fillna(method='ffill',inplace=True)

for i in obj\_col: print(i,":",obj\_col[i].unique())

df\_obj=pd.DataFrame()

from sklearn.preprocessing import LabelEncoder le=LabelEncoder()

for i in obj\_col: df\_obj[i]=le.fit\_transform(obj\_col[i])

df\_obj.head() df\_obj.info()

num\_col=df.select\_dtypes(exclude='object') num\_col.head()

num\_col.info() num\_col.describe()

plt.figure(figsize=(8,250)) c=1

for i in num\_col: ax=plt.subplot(38,1,c)

sns.boxplot(num\_col[i],orient='h',x=num\_col[i]) c+=1

plt.show()

num\_col.drop(['Zip Code','Latitude','Longitude'],axis=1,inplace=True) sns.kdeplot(num\_col['Avg Monthly Long Distance Charges']) sns.kdeplot(num\_col['Avg Monthly GB Download'])

num\_col['Avg Monthly Long Distance Charges'].fillna(num\_col['Avg Monthly Long Distance Charges'].mean(),inplace=True)

num\_col['Avg Monthly GB Download'].fillna(num\_col['Avg Monthly GB Download'].mean(),inplace=True)

num\_col.info() df\_final=df\_obj.join(num\_col) df\_final.head()

x=df\_final.iloc[:,:-1] y=df\_final.iloc[:,-1].values

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_s tate=42)

l=x\_train.select\_dtypes(include=['int64','float64']).columns from sklearn.preprocessing import StandardScaler sc=StandardScaler()

x\_train[l]=sc.fit\_transform(x\_train[l]) x\_test[l]=sc.transform(x\_test[l])

x\_train=x\_train.values x\_test=x\_test.values

x\_train x\_test

from sklearn.linear\_model import LogisticRegression reg=LogisticRegression()

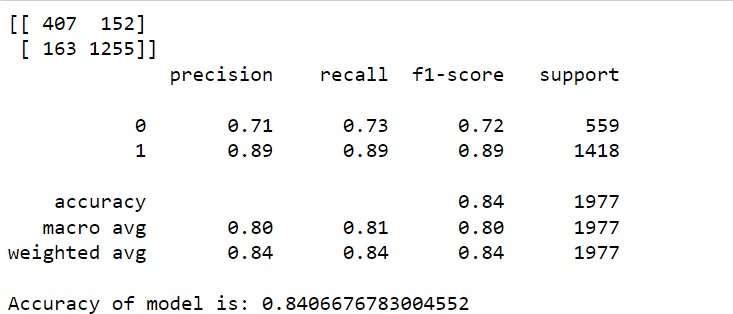
reg.fit(x\_train,y\_train) y\_pred=reg.predict(x\_test)

from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report

print(confusion\_matrix(y\_pred,y\_test)) print(classification\_report(y\_pred,y\_test))

print("Accuracy of model is:",accuracy\_score(y\_pred,y\_test))

# Output Screenshot:



**Result:**

Hence, the logistic regression model is built, and the customer status is classified.

|  |  |
| --- | --- |
| **Ex. No: 3** | **Decision Tree from scratch** |
| **Date: 13-01-2023** |

# Aim:

To build a decision tree classifier from scratch.

# Algorithm:

* Import the necessary modules and the csv file.
* Fill the null values present in the data and pre-process the data.
* Create a class and define the functions for building the tree, calculating information gain and gini index, print the tree, fit and predict the values.
* Split the data into train and test data using train test split.
* Create a object for Decision Tree Classifier and fit the model.
* Predict the values using the test data and print the accuracy score.

# Program:

import numpy as np import pandas as pd

df=pd.read\_csv('classification.csv') df.head()

class Node():

def

init (self,feature\_index=None,threshold=None,left=None,right=None,i nfo\_gain=None,value=None):

self.feature\_index=feature\_index self.threshold=threshold self.left=left

self.right=right self.info\_gain=info\_gain

self.value=value

class DecisionTreeClassifier():

def

init (self, min\_samples\_split=2, max\_depth=2): self.root = None

self.min\_samples\_split = min\_samples\_split self.max\_depth = max\_depth

def build\_tree(self, dataset, curr\_depth=0): X, Y = dataset[:,:-1], dataset[:,-1]

num\_samples, num\_features = np.shape(X) if num\_samples>=self.min\_samples\_split and

curr\_depth<=self.max\_depth:

best\_split = self.get\_best\_split(dataset, num\_samples, num\_features)

if best\_split["info\_gain"]>0: left\_subtree =

self.build\_tree(best\_split["dataset\_left"], curr\_depth+1) right\_subtree =

self.build\_tree(best\_split["dataset\_right"], curr\_depth+1) return Node(best\_split["feature\_index"],

best\_split["threshold"],left\_subtree, right\_subtree, best\_split["info\_gain"])

leaf\_value = self.calculate\_leaf\_value(Y) return Node(value=leaf\_value)

def get\_best\_split(self, dataset, num\_samples, num\_features): best\_split = {}

max\_info\_gain = -float("inf")

for feature\_index in range(num\_features): feature\_values = dataset[:, feature\_index] possible\_thresholds = np.unique(feature\_values) for threshold in possible\_thresholds:

dataset\_left, dataset\_right = self.split(dataset, feature\_index, threshold)

if len(dataset\_left)>0 and len(dataset\_right)>0:

y, left\_y, right\_y = dataset[:, -1], dataset\_left[:,

-1], dataset\_right[:, -1]

curr\_info\_gain = self.information\_gain(y, left\_y,

right\_y, "gini")

if curr\_info\_gain>max\_info\_gain: best\_split["feature\_index"] = feature\_index best\_split["threshold"] = threshold best\_split["dataset\_left"] = dataset\_left best\_split["dataset\_right"] = dataset\_right best\_split["info\_gain"] = curr\_info\_gain max\_info\_gain = curr\_info\_gain

return best\_split

def split(self, dataset, feature\_index, threshold): dataset\_left = np.array([row for row in dataset if

row[feature\_index]<=threshold])

dataset\_right = np.array([row for row in dataset if row[feature\_index]>threshold])

return dataset\_left, dataset\_right

def information\_gain(self, parent, l\_child, r\_child,mode='gini'): weight\_l = len(l\_child) / len(parent)

weight\_r = len(r\_child) / len(parent) if mode=="gini":

gain = self.gini\_index(parent) - (weight\_l\*self.gini\_index(l\_child) + weight\_r\*self.gini\_index(r\_child))

else:

gain = self.entropy(parent) - (weight\_l\*self.entropy(l\_child)

+ weight\_r\*self.entropy(r\_child)) return gain

def gini\_index(self, y): class\_labels = np.unique(y) gini = 0

for cls in class\_labels:

p\_cls = len(y[y == cls]) / len(y) gini += p\_cls\*\*2

return 1 - gini

def calculate\_leaf\_value(self, Y): Y = list(Y)

return max(Y, key=Y.count)

def print\_tree(self, tree=None, indent=" "): if not tree:

tree = self.root

if tree.value is not None: print(tree.value)

else:

print("X\_"+str(tree.feature\_index), "<=", tree.threshold, "?", tree.info\_gain)

print("%sleft:" % (indent), end="") self.print\_tree(tree.left, indent + indent) print("%sright:" % (indent), end="") self.print\_tree(tree.right, indent + indent)

def fit(self, X, Y):

dataset = np.concatenate((X, Y), axis=1) self.root = self.build\_tree(dataset)

def predict(self, X):

preditions = [self.make\_prediction(x, self.root) for x in X] return preditions

def make\_prediction(self, x, tree):

if tree.value!=None: return tree.value feature\_val = x[tree.feature\_index] if feature\_val<=tree.threshold:

return self.make\_prediction(x, tree.left) else:

return self.make\_prediction(x, tree.right) x=df.iloc[:,:-1].values

y=df.iloc[:,-1].values.reshape(-1,1)

from sklearn.model\_selection import train\_test\_split x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,random\_state=42,test\_s

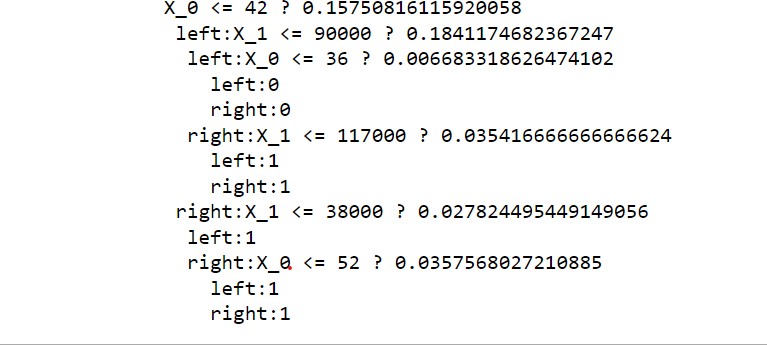
ize=0.3) tree=DecisionTreeClassifier(min\_samples\_split=3) tree.fit(x\_train,y\_train)

tree.print\_tree()

y\_pred=tree.predict(x\_test)

from sklearn.metrics import accuracy\_score accuracy\_score(y\_pred,y\_test)

# Output Screenshot:

Text  Description automatically generated

**Result:**

Hence, the decision tree classifier is built from scratch and the purchased status is classified.

|  |  |
| --- | --- |
| **Ex. No: 4** | **Hyper parameter Tuning** |
| **Date: 20-01-2023** |

# Aim:

To build an hyperparameter tuning model using grid search and random search.

# Algorithm:

* Import the necessary modules and the csv file.
* Fill the null values present in the data and pre-process the data.
* Split the data into train and test data using train test split.
* Import GridSearchCV and RandomSearchCV and declare the parameters for each model.
* Train the model with the predicted values and predict the values and print the accuracy score.

# Program:

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import warnings warnings.filterwarnings('ignore')

df=pd.read\_csv('Telco-Customer-Churn.csv') df.head()

df.info() df.describe()

df.drop('customerID',inplace=True,axis=1)

df['TotalCharges']=df['TotalCharges'].apply(pd.to\_numeric,errors='coerce'

)

df.isnull().sum() import seaborn as sns

sns.kdeplot(df['TotalCharges']) print(df['TotalCharges'].mean(),df['TotalCharges'].median())

df['TotalCharges'].fillna(df['TotalCharges'].mean(),inplace=True) a=df.select\_dtypes(include='object')

for i in a:

print(i,':',df[i].unique())

df['MultipleLines']=df['MultipleLines'].replace('No phone service','No') df['OnlineBackup']=df['OnlineBackup'].replace('No internet service','No') df['OnlineSecurity']=df['OnlineSecurity'].replace('No internet

service','No') df['DeviceProtection']=df['DeviceProtection'].replace('No internet

service','No')

df['StreamingTV']=df['StreamingTV'].replace('No internet service','No') df['TechSupport']=df['TechSupport'].replace('No internet service','No') df['StreamingMovies']=df['StreamingMovies'].replace('No internet

service','No')

from sklearn.preprocessing import LabelEncoder,StandardScaler le=LabelEncoder()

a=a.columns

df[a]=df[a].apply(lambda x : le.fit\_transform(x)) df.info()

x=df.iloc[:,:-1].values

y=df.iloc[:,-1].values

from sklearn.model\_selection import train\_test\_split x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_s

tate=42)

s=StandardScaler() x\_train=s.fit\_transform(x\_train) x\_test=s.transform(x\_test)

## KNN

from sklearn.model\_selection import GridSearchCV,RandomizedSearchCV from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier() knn.fit(x\_train,y\_train)

knn\_p={'n\_neighbors':np.arange(1,50)} print("Grid Search:")

grid\_knn=GridSearchCV(knn,knn\_p,scoring='accuracy',cv=9,n\_jobs=-1) grid\_knn.fit(x\_train,y\_train)

print(grid\_knn.best\_params\_) print(grid\_knn.best\_score\_)

print("\nRandomized Search:") rand\_knn=RandomizedSearchCV(knn,knn\_p,scoring='accuracy',cv=9,n\_jobs=-1)

rand\_knn.fit(x\_train,y\_train) print(rand\_knn.best\_params\_) print(rand\_knn.best\_score\_)

from sklearn.metrics import accuracy\_score y\_pred=knn.predict(x\_test) y\_pred\_grid=grid\_knn.predict(x\_test) y\_pred\_rand=rand\_knn.predict(x\_test)

print("Normal KNN:",accuracy\_score(y\_pred,y\_test)) print("Grid Search KNN:",accuracy\_score(y\_pred\_grid,y\_test))

print("Random Search KNN:",accuracy\_score(y\_pred\_rand,y\_test)) ## Logistic Regression

from sklearn.linear\_model import LogisticRegression

lr=LogisticRegression() lr.fit(x\_train,y\_train)

lr\_params={'solver':['newton-cg','lbfgs','sag'],'C':np.logspace(-

3,2.45,7),'penalty':['l1','l2']}

print("Grid Search:") grid\_lr=GridSearchCV(lr,lr\_params,scoring='accuracy',cv=9,n\_jobs=-1) grid\_lr.fit(x\_train,y\_train)

print(grid\_lr.best\_params\_) print(grid\_lr.best\_score\_)

print("\nRandomized Search:") rand\_lr=RandomizedSearchCV(lr,lr\_params,scoring='accuracy',cv=9,n\_jobs=-

1)

rand\_lr.fit(x\_train,y\_train) print(rand\_lr.best\_params\_) print(rand\_lr.best\_score\_)

y\_pred=lr.predict(x\_test) y\_pred\_grid=grid\_lr.predict(x\_test) y\_pred\_rand=rand\_lr.predict(x\_test)

print("Normal Logistic Regression:",accuracy\_score(y\_pred,y\_test)) print("Grid Search Logistic

Regression:",accuracy\_score(y\_pred\_grid,y\_test)) print("Random Search Logistic

Regression:",accuracy\_score(y\_pred\_rand,y\_test)) ## Naive Bayes Classification

from sklearn.naive\_bayes import GaussianNB

gnb=GaussianNB() gnb.fit(x\_train,y\_train)

gnb\_params={'var\_smoothing':np.logspace(5,-7,79)} print("Grid Search:")

grid\_gnb=GridSearchCV(gnb,gnb\_params,scoring='accuracy',cv=9,n\_jobs=-1)

grid\_gnb.fit(x\_train,y\_train) print(grid\_gnb.best\_params\_) print(grid\_gnb.best\_score\_)

print("\nRandomized Search:") rand\_gnb=RandomizedSearchCV(gnb,gnb\_params,scoring='accuracy',cv=9,n\_jobs

=-1)

rand\_gnb.fit(x\_train,y\_train) print(rand\_gnb.best\_params\_) print(rand\_gnb.best\_score\_)

y\_pred=gnb.predict(x\_test) y\_pred\_grid=grid\_gnb.predict(x\_test) y\_pred\_rand=rand\_gnb.predict(x\_test)

print("Normal Naive Bayes:",accuracy\_score(y\_pred,y\_test)) print("Grid Search Naive Bayes:",accuracy\_score(y\_pred\_grid,y\_test))

print("Random Search Naive Bayes:",accuracy\_score(y\_pred\_rand,y\_test)) ## Decision Trees

from sklearn.tree import DecisionTreeClassifier dtc=DecisionTreeClassifier() dtc.fit(x\_train,y\_train)

dtc\_params={'criterion':['gini','entropy'],'max\_depth':np.arange(1,20),'m in\_samples\_split':np.arange(1,10)

,'max\_features':np.arange(1,7)}

print("Grid Search:") grid\_dtc=GridSearchCV(dtc,dtc\_params,scoring='accuracy',cv=9,n\_jobs=-1) grid\_dtc.fit(x\_train,y\_train)

print(grid\_dtc.best\_params\_) print(grid\_dtc.best\_score\_)

print("Randomized Search:") rand\_dtc=RandomizedSearchCV(dtc,dtc\_params,scoring='accuracy',cv=9,n\_jobs

=-1)

rand\_dtc.fit(x\_train,y\_train) print(grid\_dtc.best\_params\_)

print(grid\_dtc.best\_score\_)

y\_pred=dtc.predict(x\_test) y\_pred\_grid=grid\_dtc.predict(x\_test) y\_pred\_rand=rand\_dtc.predict(x\_test)

print("Normal Decision Tree:",accuracy\_score(y\_pred,y\_test)) print("Grid Search Decision Tree:",accuracy\_score(y\_pred\_grid,y\_test)) print("Random Search Decision Tree:",accuracy\_score(y\_pred\_rand,y\_test))

## Support Vector Machines

from sklearn.svm import SVC

svc=SVC() svc.fit(x\_train,y\_train)

svc\_params={'C':[0.1,1,10,100],'gamma':[0.1,0.001,0.01,0.001],'kernel':['

poly','sigmoid','rbf']}

print("Grid Search:") grid\_svc=GridSearchCV(svc,svc\_params,scoring='accuracy',cv=9,n\_jobs=-1) grid\_svc.fit(x\_train,y\_train)

print(grid\_svc.best\_params\_) print(grid\_svc.best\_score\_)

print("\nRandomized Search:") rand\_svc=RandomizedSearchCV(svc,svc\_params,scoring='accuracy',cv=9,n\_jobs

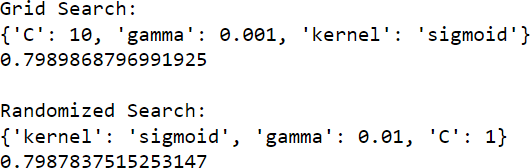
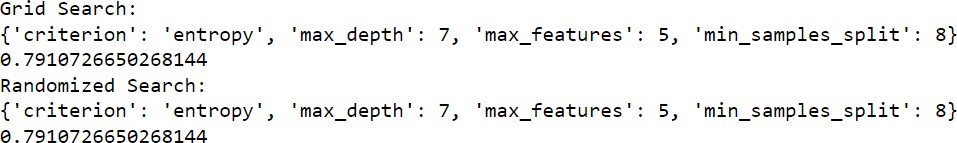
=-1)

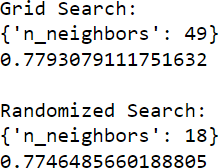
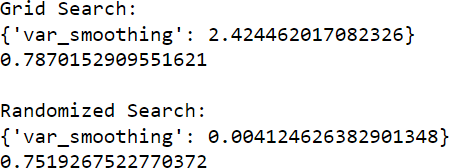
rand\_svc.fit(x\_train,y\_train) print(rand\_svc.best\_params\_) print(rand\_svc.best\_score\_)

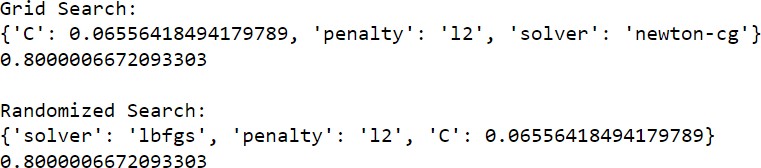
y\_pred=svc.predict(x\_test) y\_pred\_grid=grid\_svc.predict(x\_test) y\_pred\_rand=rand\_svc.predict(x\_test)

print("Normal Support Vector Machines:",accuracy\_score(y\_pred,y\_test)) print("Grid Search Decision Tree:",accuracy\_score(y\_pred\_grid,y\_test)) print("Random Search Decision Tree:",accuracy\_score(y\_pred\_rand,y\_test))

# Output Screenshot:







**Result:**

Hence, parameters for each classification model are selected using Grid Search and Random Search.

|  |  |
| --- | --- |
| **Ex. No: 5** | **Ensemble methods** |
| **Date: 27-01-2023** |

# Aim:

To build an ensemble of classification models.

# Algorithm:

* Import the necessary modules and the csv file.
* Fill the null values present in the data and pre-process the data.
* Split the data into train and test data using train test split.
* Create 5 bags consisting of Logistic regression, K-Nearest Neighbours, Naïve bayes, Decision Tree and Support Vector Machines and predict the classes.
* Create a separate data frame consisting of the predicted values for each model and use max voting to predict the appropriate final class.

# Program:

import numpy as np import pandas as pd import warnings

warnings.filterwarnings('ignore') df=pd.read\_csv('telecom\_customer\_churn.csv') df.head()

df.info()

df.drop(['Churn Category','Churn Reason'],axis=1,inplace=True) df.drop(['Customer ID','City'],axis=1,inplace=True) df['Offer'].value\_counts().index[0]

obj\_col=df.select\_dtypes(include='object') for i in obj\_col:

df[i].fillna(method='ffill',inplace=True)

for i in obj\_col: print(i,":",df[i].unique())

df\_obj=pd.DataFrame()

from sklearn.preprocessing import LabelEncoder le=LabelEncoder()

for i in obj\_col: df[i]=le.fit\_transform(obj\_col[i])

df.drop(['Latitude','Longitude','Zip Code'],inplace=True,axis=1) df['Avg Monthly GB Download']=df['Avg Monthly GB

Download'].fillna(value=df['Avg Monthly GB Download'].mean()) df['Avg Monthly Long Distance Charges']=df['Avg Monthly Long Distance

Charges'].fillna(value=df['Avg Monthly Long Distance Charges'].mean())

x=df.iloc[:,:-1]

y=df.iloc[:,-1]

from sklearn.model\_selection import train\_test\_split x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3)

x\_train.reset\_index(inplace=True,drop=True) y\_train.reset\_index(inplace=True,drop=True)

x\_train.columns x\_train.info()

b1\_x\_train=x\_train.sample(n=992) b1\_y\_train=y\_train[b1\_x\_train.index]

b2\_x\_train=x\_train.sample(n=992) b2\_y\_train=y\_train[b2\_x\_train.index]

b3\_x\_train=x\_train.sample(n=992) b3\_y\_train=y\_train[b3\_x\_train.index]

b4\_x\_train=x\_train.sample(n=992) b4\_y\_train=y\_train[b4\_x\_train.index]

b5\_x\_train=x\_train.sample(n=644) b5\_y\_train=y\_train[b5\_x\_train.index]

## Logistic Regression -Bag 1

from sklearn.linear\_model import LogisticRegression from sklearn.model\_selection import GridSearchCV

lr=LogisticRegression() lr.fit(b1\_x\_train,b1\_y\_train)

lr\_params={'solver':['newton-cg','lbfgs','sag'],'C':np.logspace(-

10,2.45,10),'penalty':['l1','l2']}

print("Grid Search:") grid\_lr=GridSearchCV(lr,lr\_params,scoring='accuracy',cv=9,n\_jobs=-1) grid\_lr.fit(b1\_x\_train,b1\_y\_train)

print(grid\_lr.best\_score\_)

## KNN-Bag 2

from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier() knn.fit(b2\_x\_train,b2\_y\_train)

knn\_p={'n\_neighbors':np.arange(1,50)}

grid\_knn=GridSearchCV(knn,knn\_p,scoring='accuracy',cv=9,n\_jobs=-1) grid\_knn.fit(b2\_x\_train,b2\_y\_train)

print(grid\_knn.best\_score\_)

## Naive Bayes-Bag 3

from sklearn.naive\_bayes import GaussianNB

gnb=GaussianNB() gnb.fit(b3\_x\_train,b3\_y\_train)

gnb\_params={'var\_smoothing':np.logspace(5,-7,79)}

grid\_gnb=GridSearchCV(gnb,gnb\_params,scoring='accuracy',cv=9,n\_jobs=-1) grid\_gnb.fit(b3\_x\_train,b3\_y\_train)

print(grid\_gnb.best\_score\_)

## Decision Tree-Bag 4

from sklearn.tree import DecisionTreeClassifier dtc=DecisionTreeClassifier() dtc.fit(b4\_x\_train,b4\_y\_train)

dtc\_params={'criterion':['gini','entropy'],'max\_depth':np.arange(1,20),'m in\_samples\_split':np.arange(1,10)

,'max\_features':np.arange(1,7)} grid\_dtc=GridSearchCV(dtc,dtc\_params,scoring='accuracy',cv=9,n\_jobs=-1)

grid\_dtc.fit(b4\_x\_train,b4\_y\_train) print(grid\_dtc.best\_score\_)

## Support Vector Machine-Bag 5 from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score svc=SVC()

svc.fit(b5\_x\_train,b5\_y\_train) print(accuracy\_score(svc.predict(x\_test),y\_test)) print()

predicted\_data=pd.DataFrame()

predicted\_data['Logistic Regression']=grid\_lr.predict(x\_test) predicted\_data['KNN']=grid\_knn.predict(x\_test) predicted\_data['Naive Bayes']=grid\_gnb.predict(x\_test) predicted\_data['Decision Tree']=grid\_dtc.predict(x\_test) predicted\_data['Support Vector Machines']=svc.predict(x\_test)

predicted\_data

for i in predicted\_data: print(i)

df1=predicted\_data.iloc[0] l=[]

for i in range(0,len(predicted\_data)):

c=d=0 pred=0

for i in predicted\_data.iloc[i]: if i==0:

c+=1

elif i==1:

d+=1

if c>d:

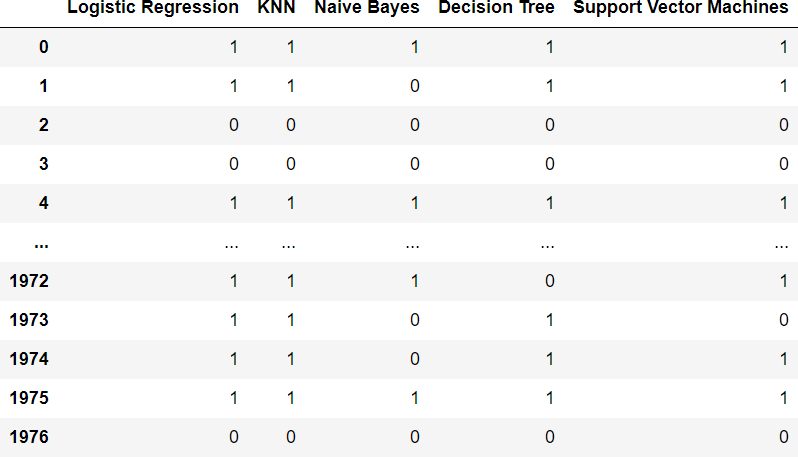
pred=0 elif d>c:

pred=1 l.append(pred)

predicted\_data predicted\_data['Final']=l

from sklearn.metrics import accuracy\_score print(accuracy\_score(predicted\_data['Final'],y\_test))

# Output Screenshot:



**Result:**

Hence, using ensemble methods of 5 classification models, churn is classified.

|  |  |
| --- | --- |
| **Ex. No: 6** | **K Means Clustering** |
| **Date: 03-02-2023** |

# Aim:

To build a clustering model using K-Means algorithm.

# Algorithm:

* Import the necessary modules and the csv file.
* Fill the null values present in the data and pre-process the data.
* Find the optimal number of clusters using WCSS graph.
* Retrain the model with the optimal number of clusters and predict the labels of each data point.
* Visualize the clusters in 3-dimensional using mpl\_toolkits and print the silhouette score.

# Program:

import numpy as np import pandas as pd import seaborn as sns import warnings

import matplotlib.pyplot as plt warnings.filterwarnings('ignore') ### IRIS dataset df=sns.load\_dataset('iris') df.head()

sns.pairplot(df) dataset=df.iloc[:,[1,2,3]].values from sklearn.cluster import KMeans wcss=[]

for i in range(1,15): km=KMeans(n\_clusters=i,random\_state=42) km.fit(dataset) wcss.append(km.inertia\_)

plt.plot(range(1,15),wcss) km=KMeans(n\_clusters=3,random\_state=42) y=km.fit\_predict(dataset)

y

%matplotlib widget

from mpl\_toolkits.mplot3d import Axes3D

fig=plt.figure() #ax=fig.add\_subplot(111,projection='3d') ax=Axes3D(fig)

ax.scatter(dataset[y==0,0],dataset[y==0,1],dataset[y==0,2],c='red',label= 'Cluster-1')

ax.scatter(dataset[y==1,0],dataset[y==1,1],dataset[y==1,2],c='green',labe l='Cluster-2')

ax.scatter(dataset[y==2,0],dataset[y==2,1],dataset[y==2,2],c='blue',label

='Cluster-3')

ax.scatter(km.cluster\_centers\_[:,0],km.cluster\_centers\_[:,1],km.cluster\_c enters\_[:,2],c='black',s=100,depthshade=False)

ax.set\_xlabel('sepal\_width') ax.set\_ylabel('petal\_length') ax.set\_zlabel('petal\_width') plt.legend(bbox\_to\_anchor=(1.05, 1), loc=2)

from sklearn.metrics import silhouette\_score score=silhouette\_score(dataset,y) print(score)

### data.csv df1=pd.read\_csv('data.csv') df1.head() dataset1=df1.iloc[:,[2,3,4]].values from sklearn.cluster import KMeans wcss=[]

for i in range(1,15): km=KMeans(n\_clusters=i,random\_state=42) km.fit(dataset1) wcss.append(km.inertia\_)

%matplotlib inline plt.plot(range(1,15),wcss) km1=KMeans(n\_clusters=6,random\_state=42) y1=km1.fit\_predict(dataset1)

y1

%matplotlib widget

from mpl\_toolkits.mplot3d import Axes3D

fig1=plt.figure() #ax=fig.add\_subplot(111,projection='3d') ax=Axes3D(fig1)

ax.scatter(dataset1[y1==0,0],dataset1[y1==0,1],dataset1[y1==0,2],c='red', label='Cluster-1')

ax.scatter(dataset1[y1==1,0],dataset1[y1==1,1],dataset1[y1==1,2],c='green ',label='Cluster-2')

ax.scatter(dataset1[y1==2,0],dataset1[y1==2,1],dataset1[y1==2,2],c='blue'

,label='Cluster-3') ax.scatter(dataset1[y1==3,0],dataset1[y1==3,1],dataset1[y1==3,2],c='yello

w',label='Cluster-4') ax.scatter(dataset1[y1==4,0],dataset1[y1==4,1],dataset1[y1==4,2],c='magen

ta',label='Cluster-5') ax.scatter(dataset1[y1==5,0],dataset1[y1==5,1],dataset1[y1==5,2],c='orang

e',label='Cluster-6')

ax.scatter(km1.cluster\_centers\_[:,0],km1.cluster\_centers\_[:,1],km1.cluste r\_centers\_[:,2],c='black',s=100,depthshade=False)

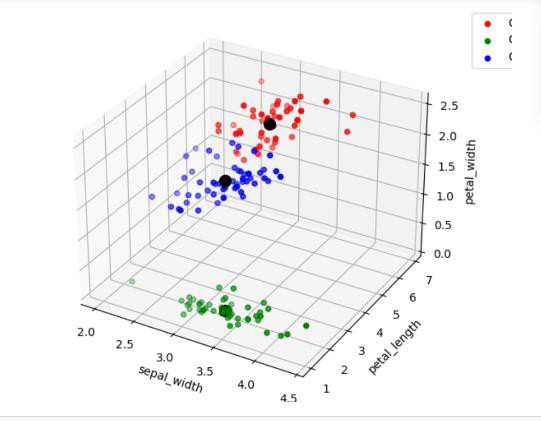
ax.set\_xlabel('Age') ax.set\_ylabel('Annual Income') ax.set\_zlabel('Spending Score')

plt.legend(bbox\_to\_anchor=(1.05, 1), loc=2)

from sklearn.metrics import silhouette\_score score=silhouette\_score(dataset1,y1)

print(score)

# Output Screenshot:

A picture containing shape  Description automatically generated

**Result:**

Hence, cluster classification model using K-Means is built and the labels of the data points are predicted.

|  |  |
| --- | --- |
| **Ex. No: 7** | **Principal Component Analysis** |
| **Date: 17-02-2023** |

# Aim:

To build a principal component analysis from scratch.

# Algorithm:

* Import the necessary modules and the necessary datasets.
* Fill the null values present in the data and pre-process the data.
* Normalize the data and compute the covariance matrix.
* Calculate the eigen values and eigen vectors of the covariance matrix.
* Reproject the data by scalar dot operation between the highest eigen vector and the normalized data i.e., mean is subtracted from all the data points.
* Calculate the variance and plot it using scree plot and select the number of features.

# Program:

import numpy as np import pandas as pd

from sklearn.datasets import load\_digits

digit=load\_digits() df=pd.DataFrame(digit.data) df.columns=digit.feature\_names df

x=df mean\_l=np.mean(x.T,axis=1) mean\_l

x\_center=x-mean\_l x\_center

q=[]

for i in df:

w=np.array(df[i]) q.append(w)

cov=np.cov(q,bias=False) cov

cov.shape eigen=np.linalg.eig(cov) eigen\_values=eigen[0] eigen\_vectors=eigen[1]

print("Eigen values:",eigen\_values) print("Eigen vectors:",eigen\_vectors) pca\_x=eigen\_vectors.T.dot(x\_center.T) pca\_x.T

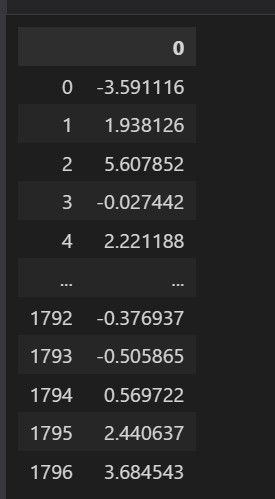
df\_pca=pd.DataFrame(pca\_x.T)

df\_pca eigen\_value\_max=np.sort(eigen\_values)[-1] eigen\_value\_max eigen\_vector\_max=np.sort(eigen\_vectors)[-1] eigen\_vector\_max pca1\_x=eigen\_vector\_max.T.dot(x\_center.T) pca1\_x

df1\_pca=pd.DataFrame(pca1\_x) df1\_pca

np.cumsum(pca1\_x) np.cumsum(df\_pca.iloc[:,:1])

# Output Screenshot:



**Result:**

Hence, principal component analysis is built from scratch and the feature are selected.

|  |  |
| --- | --- |
| **Ex. No: 8** | **Recommendation System** |
| **Date: 3-03-2023** |

# Aim:

To build a content based and collaborative filter-based recommendation system.

# Algorithm:

* Import the necessary modules and the csv file.
* Fill the null values present in the data and pre-process the data.

# Content based recommendation system:

* Create a tfidf sparse matrix using TfidfVectorizer from sklearn library.
* Create a cosine similarity matrix using linear kernel function and store the keywords in a data frame.
* Create a user defined recommendation function to check if the keyword is present and sort the cosine similarity values and return the top recommended the top n items.

# Collaborative filter-based recommendation system:

* Create a csr matrix using the scipy module and use the nearest neighbours algorithm to fit the csr matrix.
* Find similar users using the nearest neighbour model trained with the csr matrix.
* Predict the rating a user would give to the item and recommend the top n-items.

# Program:

import numpy as np import pandas as pd

## Content based recommendation system df=pd.read\_csv('result\_final.csv') df.head()

df.info()

df.drop(columns=['Unnamed: 0.1','link','date'],axis=1,inplace=True) def string(x):

x=x[1:-1] s=''

for i in x.split(','): s+=i+','

return s df.fillna(method='ffill',inplace=True) df['keywords']=df['keywords'].agg(string)

from sklearn.feature\_extraction.text import TfidfVectorizer tfidf=TfidfVectorizer(stop\_words='english')

tfidf\_mat=tfidf.fit\_transform(df['keywords']) tfidf\_mat

from sklearn.metrics.pairwise import linear\_kernel

cosine\_sim=linear\_kernel(tfidf\_mat,tfidf\_mat) cosine\_sim

df1=pd.DataFrame(pd.Series(df['keywords'],index=df.index)) df1

import re

def recomendation(x):

ind=df1[df1['keywords'].str.contains(x,flags=re.IGNORECASE,regex=True)

].index[0] sim\_score=list(enumerate(cosine\_sim[ind]))

sim\_score=sorted(sim\_score,key=lambda a:a[1],reverse=True) sim\_score=sim\_score[1:8]

final\_ind=[i[0] for i in sim\_score] return final\_ind

r=input('Enter keyword for news title:') ind=recomendation(r)

print('Recommended news are:') for i in ind:

print('>>',df['title'][i],'\n \t>>>',df['title\_summary'][i])

## Collaborative filtering recommendation system

df2=pd.read\_csv('Netflix\_Dataset\_Movie.csv') df3=pd.read\_csv('Netflix\_Dataset\_Rating.csv') df3.head() users=df3['User\_ID'].unique().shape[0] movie=df3['Movie\_ID'].unique().shape[0] df3.sort\_values('User\_ID',inplace=True) user\_uni

df3

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder() df3['User\_ID']=le.fit\_transform(df3['User\_ID']) df3.sort\_values('Movie\_ID',inplace=True) df3['Movie\_id']=le.fit\_transform(df3['Movie\_ID']) a=np.zeros((users,movie))

for i in df3.itertuples(): a[i[1]-1,i[4]-1]=i[2]

a

from scipy.sparse import csr\_matrix sparse\_matrix=csr\_matrix(a)

from sklearn.neighbors import NearestNeighbors

nn=NearestNeighbors(n\_neighbors=5,algorithm='brute',metric='cosine',n\_job s=-1)

nn.fit(sparse\_matrix) df3

data=df3.sort\_values('User\_ID',ascending=True) colab\_filter=data[data['User\_ID']==69].Movie\_ID colab\_filter=colab\_filter.tolist() len(colab\_filter)

item\_id=[]

for i in colab\_filter: dist,ind=nn.kneighbors(sparse\_matrix[i],n\_neighbors=5) ind=ind.flatten()

ind=ind[:1] item\_id.extend(ind)

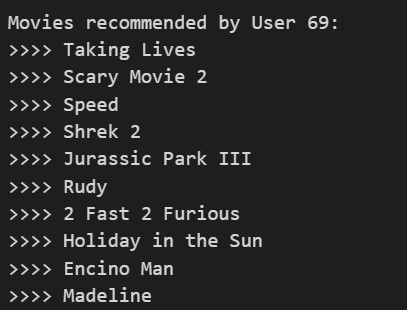
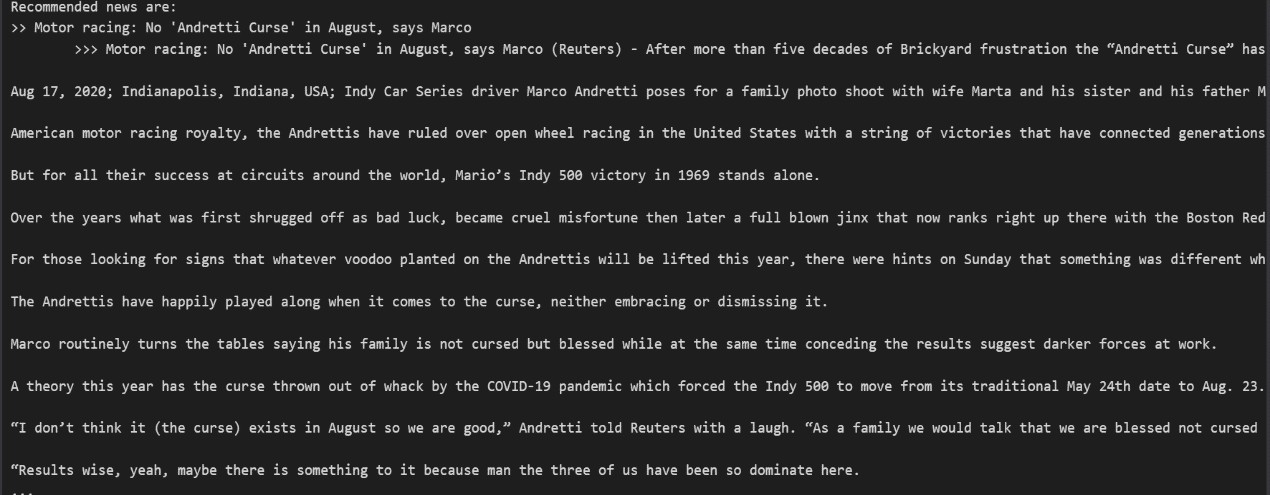
item\_id=item\_id[:10] item\_id

df2

print("Movies recommended by User 69:") for i in item\_id:

print(">>>>",df2['Name'][i-1])

# Output Screenshot:



**Result:**

Hence, for the given keyword input the items are content based recommendations are recommended to the user and according to the given user collaborative filter-based recommendations are recommended.

|  |  |
| --- | --- |
| **Ex. No: 9** | **Neural Network** |
| **Date: 17-03-2023** |

# Aim:

To build a neural network using tensorflow and pytorch modules.

# Algorithm:

* Import the necessary modules and the csv file.
* Fill the null values present in the data and pre-process the data.
* Standardize the data using Standard Scalar from skleran.preprocessing module.

# Tensorflow neural network:

* Create the neural network by adding the layers and the activation functions of each layer.
* Compile the neural network and declare early stopping to avoid overtraining of the model.
* Fit the neural network and predict the values using validation data and print the classification report and confusion matrix.

# Pytorch neural network:

* Change the datatype of the train, test data and to torch datatypes and concert too tensor dataset.
* Create the neural network by adding the hidden layers and declare the optimizer and loss functions and train the network.
* Predict the values and print the accuracy of the network.

# Program:

import numpy as np import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt df=pd.read\_csv('Bank\_Personal\_Loan\_Modelling.csv') df.head()

df.info() df.isnull().sum() df.duplicated().sum()

sns.countplot(x='Personal Loan',data=df) df.describe() df['Experience']=abs(df['Experience']) df['Annual\_CCAvg']=df['CCAvg']\*12

df.drop(['ID','ZIP Code','CCAvg'],axis=1,inplace=True) x=df.drop('Personal Loan',axis=1).values y=df['Personal Loan'].values.reshape(-1,1)

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_s tate=42)

from sklearn.preprocessing import StandardScaler sc=StandardScaler() x\_train=sc.fit\_transform(x\_train) x\_test=sc.fit\_transform(x\_test)

from sklearn.decomposition import PCA

pca=PCA() x\_train=pca.fit\_transform(x\_train) x\_test=pca.fit\_transform(x\_test) np.cumsum(pca.explained\_variance\_ratio\_) from sklearn.decomposition import PCA

pca=PCA(n\_components=10) x\_train=pca.fit\_transform(x\_train) x\_test=pca.fit\_transform(x\_test) x\_train.shape

## Using keras classification import tensorflow.keras

from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense model=Sequential()

model.add(Dense(9,activation='relu')) model.add(Dense(5,activation='relu')) model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='adam',loss='categorical\_crossentropy',metrics=[' accuracy']) model.fit(x\_train,y\_train,epochs=10,validation\_data=(x\_test,y\_test),verbo se=1)

model\_loss=pd.DataFrame(model.history.history) model\_loss.plot()

model=Sequential()

model.add(Dense(9,activation='relu')) model.add(Dense(5,activation='relu')) model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='adam',loss='categorical\_crossentropy',metrics=[' accuracy'])

from tensorflow.keras.callbacks import EarlyStopping early\_stop=EarlyStopping(monitor='val\_loss',mode='min',verbose=1,patience

=25)

model.fit(x\_train,y\_train,epochs=100,validation\_data=(x\_test,y\_test),verb ose=1,callbacks=[early\_stop]) model\_loss=pd.DataFrame(model.history.history)

model\_loss.plot() pred=model.predict(x\_test)

threshold=0.5

pred=np.where(pred>threshold,1,0)

from sklearn.metrics import classification\_report,confusion\_matrix

print(confusion\_matrix(y\_test,pred)) print('\n') print(classification\_report(y\_test,pred)) ## Using PyTorch classification

import torch

import torch.nn as nn

from torch.optim import SGD

from torch.utils.data import TensorDataset,DataLoader x=df.drop('Personal Loan',axis=1).values y=df['Personal Loan'].values.reshape(-1,1)

from sklearn.model\_selection import train\_test\_split x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_s tate=42)

from sklearn.preprocessing import StandardScaler sc=StandardScaler() x\_train=sc.fit\_transform(x\_train) x\_test=sc.fit\_transform(x\_test) x\_train=torch.tensor(x\_train).to(torch.float32) x\_test=torch.tensor(x\_test).to(torch.float32) y\_train=torch.tensor(y\_train).to(torch.float32) y\_test=torch.tensor(y\_test).to(torch.float32) dataset=TensorDataset(x\_train,y\_train) data=DataLoader(dataset,batch\_size=24,shuffle=True)

model=nn.Sequential(nn.Linear(11,8),nn.ReLU(),nn.Linear(8,4),nn.ReLU(),nn

.Linear(4,1),nn.Sigmoid()) model fn\_loss=nn.BCELoss()

optimr=SGD(model.parameters(),lr=0.001) def train(model,epoch,data):

train\_loss=[0]\*epoch train\_acc=[0]\*epoch

for i in range(epoch):

for x\_batch,y\_batch in data:

pred=model(x\_batch) loss=fn\_loss(pred,y\_batch)

loss.backward()

optimizer.step() optimizer.zero\_grad()

train\_loss[i]+=loss.item()\*x\_batch.size(0)

crt=(torch.where(model(x\_batch)>=0.5,1,0)==y\_batch).sum() train\_acc[i]+=crt

train\_loss[i] = train\_loss[i]/len(data.dataset) train\_acc[i]=train\_acc[i]/len(data.dataset)

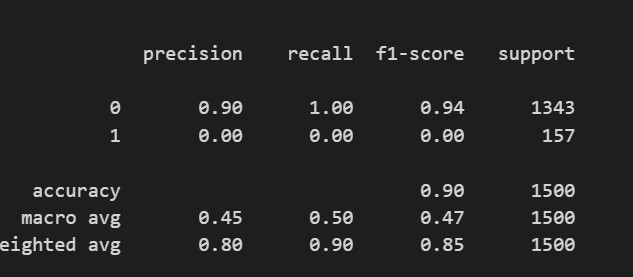
print(f'Epoch:{i+1}|Loss:{train\_loss[i]}|Accuracy:{train\_acc[i]}') return train\_loss,train\_acc

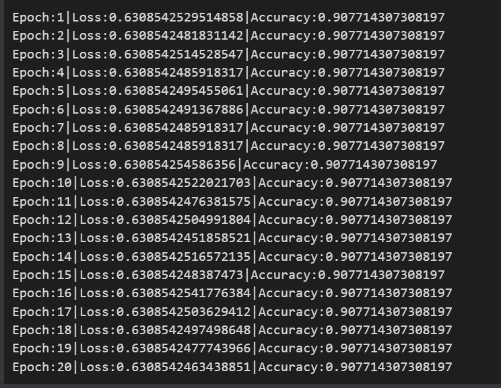
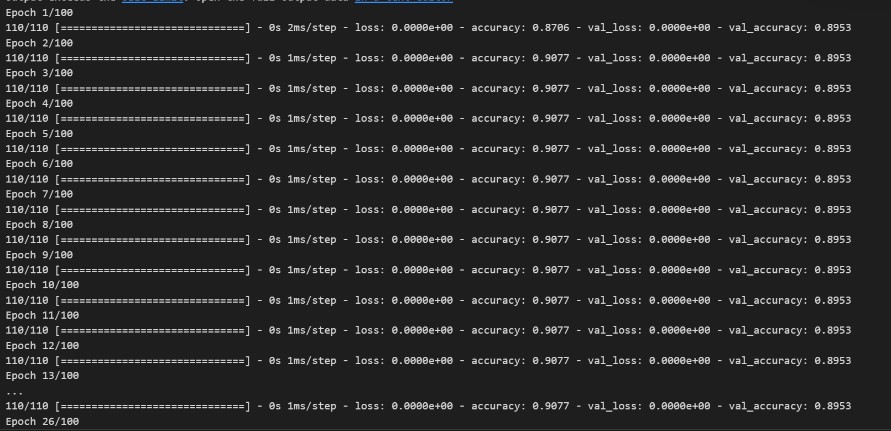
train\_loss,train\_acc=train(model,20,data) testdata=TensorDataset(x\_test,y\_test) tdata=DataLoader(testdata,batch\_size=1500) with torch.no\_grad():

for x\_whole,y\_whole in tdata: pred=model(x\_whole)

crt\_cnt=(torch.where(pred>=0.5,1,0)==y\_whole).sum() print(f'Accuracy:{crt\_cnt/len(tdata.dataset)}')

# Output Screenshot:





**Result:**

Hence, neural networks are built using tensorflow and pytorch and the data is classsified.

|  |  |
| --- | --- |
| **Ex. No: 10** | **Convolutional Neural Network** |
| **Date: 24-03-2023** |

# Aim:

To build a convolutional neural network using tensorflow for image classification.

# Algorithm:

* Import the necessary modules and the database.
* Label the images of each class and spilt the data into train and test data.
* Create the convolutional neural network by adding the layers of Conv2D and MaxPooling2D and the activation functions of each layer.
* Create an early stop to avoid overtraining of the model and compile the model and fit the model with training data.
* Predict the accuracy of the model using the test data and print the accuracy.

# Program:

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import tensorflow as tf

from tensorflow.keras import datasets,layers,models (train\_images,train\_labels),(test\_images,test\_labels)=datasets.cifar10.lo ad\_data()

train\_images,test\_images=train\_images/255.0,test\_images/255.0 class\_names=['airplane','automobile','bird','cat','deer','dog','frog','ho rse','ship','truck']

plt.figure(figsize=(10,10)) for i in range(10):

plt.subplot(5,5,i+1)

plt.xticks([])

plt.yticks([]) plt.grid(False) plt.imshow(train\_images[i])

plt.xlabel(class\_names[train\_labels[i][0]]) plt.show()

from keras.layers.core import Dense,Activation,Dropout,Flatten from tensorflow.keras.models import Sequential

from keras.layers.convolutional import Conv2D,MaxPooling2D model=Sequential()

model.add(Conv2D(32,(3,3),activation='relu',input\_shape=(32,32,3)))

model.add(MaxPooling2D(2,2)) model.add(Conv2D(64,(3,3),activation='relu')) model.add(MaxPooling2D(2,2)) model.add(Conv2D(64,(3,3),activation='relu')) model.add(layers.Flatten()) model.add(Dense(128,activation='relu')) model.add(Dense(10))

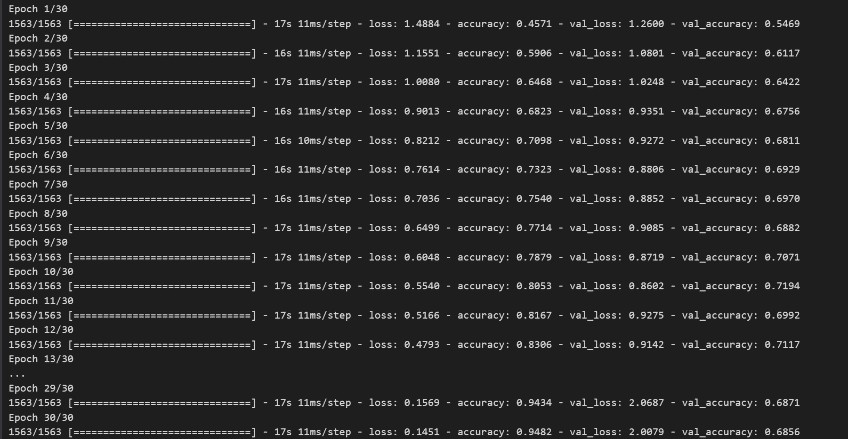
model.summary()

from keras.callbacks import EarlyStopping early\_stop=EarlyStopping(monitor='val\_loss',mode='min',verbose=1,patience

=25)

model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCros sentropy(from\_logits=True),metrics=['accuracy']) history=model.fit(train\_images,train\_labels,epochs=30,validation\_data=(te st\_images,test\_labels),callbacks=[early\_stop]) plt.plot(history.history['accuracy'], label='accuracy') plt.plot(history.history['val\_accuracy'], label='val\_accuracy') plt.xlabel(['Epoch'])

plt.ylabel(['Accuracy']) plt.ylim([.5,1]) plt.legend(loc='lower right') **Output Screenshot:**



# Result:

Hence, the convolutional neural network is built, and the images are classified.

|  |  |
| --- | --- |
| **Ex. No: 11** | **MLOps-Using Flask** |
| **Date: 23-03-2023** |

# Aim:

To build a flask model that implements a recommendation system.

# Algorithm:

* Import the necessary modules and the necessary datasets.
* Fill the null values present in the data and pre-process the data.
* Create a recommendation model that receives user and input and recommends items and rename the file as model.py.
* Connect mysql using pymysql library to verify the login credentials.
* Create html page to display the output, receive input and a login page and render all the html pages using appropriate functions.
* Pass the input values to python from html to check the login credentials using sql.
* Pass the input for recommendation and print the recommendations in the html page.

# Program:

from flask import Flask,render\_template,request import pickle

from model import Recommendation import pymysql as sql

app= Flask( name )

from flask import Flask, render\_template, request

import numpy as np

import pickle

app = Flask(\_\_name\_\_)

model = pickle.load(open('model.pkl', 'rb'))

@app.route('/')

def index():

    return render\_template('index.html')

@app.route('/predict', methods=['GET', 'POST'])

def predict():

    val1 = request.form['bedrooms']

    val2 = request.form['bathrooms']

    val3 = request.form['floors']

    val4 = request.form['yr\_built']

    arr = np.array([val1, val2, val3, val4])

    arr = arr.astype(np.float64)

    pred = model.predict([arr])

    return render\_template('index.html', data=int(pred))

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

## Recommendation system:

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

import pickle

df = pd.read\_csv(r"C:\Users\rysha\Downloads\house\_data.csv")

columns = ['bedrooms', 'bathrooms', 'floors', 'yr\_built', 'price']

df = df[columns]

X = df.iloc[:, 0:4]

y = df.iloc[:, 4:]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25)

lr = LinearRegression()

lr.fit(X\_train, y\_train)

pickle.dump(lr, open('model.pkl', 'wb'))

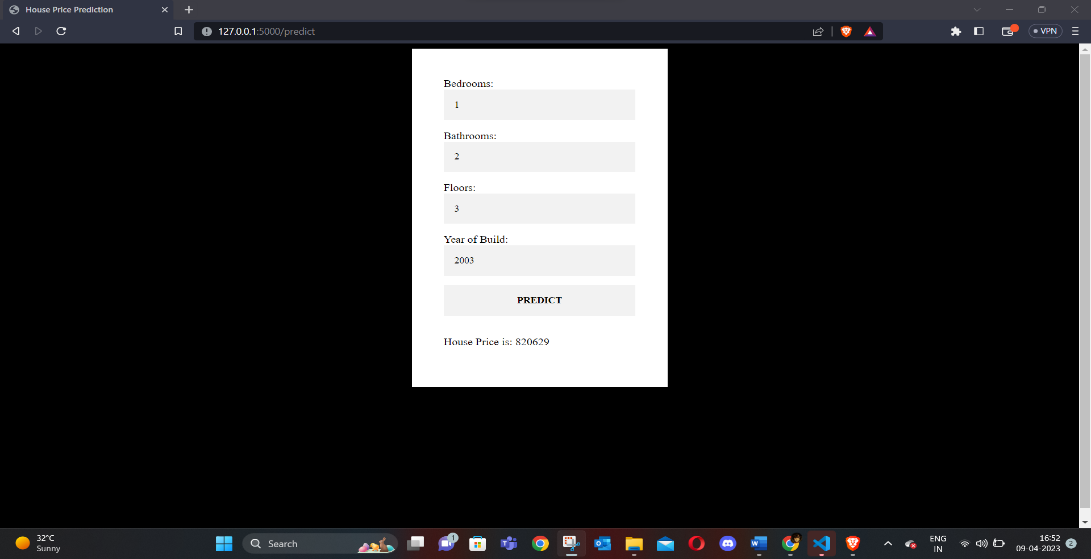
import pickle rec=Recommendation()

pickle.dump(rec,open('model.pkl','wb'))

# Output Screenshot:

A screenshot of a computer

Description automatically generated



**Result:**

Hence, the recommendation system is built and implemented using flask.