SIR GURUDAS MAHAVIDYALAYA

DEPARTMENT OF

Computer Sc.

LAB MANUAL

Subject: Data Warehousing & Data Mining (Using Python)

Paper Code: DSE A2

INDEX

SL no.-	PROGRAM NAME-	SIGN.-
1.	**1.Linear Regression-** **a.**Compute y = ex for x = 1 to 50. **b.**Save the values generated in the CSV File [through Pandas]. **c.**Find out the Model Parameters (Slope and Intercept). **e.**Evaluate the error in terms of training set.	
2.	**2.Dataset Import-** **a.** Import Titanic: Machine Learning from Disaster dataset. **b.** Find the total number of records present in the dataset. **d.**Identify and print categorical and numerical attributes present in the dataset. **e.**Find unique values for string variables in the dataset.	
3.	**3.Dataset Pre-processing-** **a.**Find the number of missing values. **c.**Scale numeric variables into specified range.	
4.	**4.Understanding of the Dataset-** **a.**Use bar chart for plotting number of missing values for each attribute. **b.**Use pie chart for plotting percentage of survivals.	
5.	**5.Retrieve Train and Test Dataset-** **a.**Partition the original dataset into train and test dataset with the ratio of 8:2. **b.**Partition the original dataset into train and test dataset with the ratio of 7:3.	
6.	**6.Predict the chance of survival-** **a.**Apply Bayesian classification methods [Use Multinomial and Gaussian Naïve Bayes for classification]. **c.**Apply Decision Tree classifier Try this classifier model using different criterions and splitter. **d.**Apply Logistic Regression on the Titanic Dataset to predict the probability of	

	survival of the passengers.	
7.	**9.Import the Dataset "StudentPerformance" and then:** **a.**Apply pre-processing techniques (if applicable).	

1. Linear Regression-

a.Compute y = ex for x = 1 to 50.

The value of Exponential Function e^x can be expressed using following Taylor Series.
e^x = 1 + x/1! + x^2/2! + x^3/3! + ...

- **Python Source**

code-- # Python program to
calculate # e raise to the
power x

```
# Function to calculate value
# using sum of first n terms
of # Taylor Series

def exponential(n, x):


    # initialize sum of series

    sum = 1.0

    for i in range(n, 0, -1):

        sum = 1 + x * sum /

        i

    print ("e^x =", sum)

# Driver program to test above function

n = 10

x = 1.0
exponential(n, x)
```

- **Output-**

e^x = 2.718282

b.Save the values generated in the CSV File [through Pandas].

Let's see how to save a Pandas DataFrame as a CSV file using to_csv() method.

Example:

- Source code--

```
# importing pandas as

pd import pandas as pd
```

```
# list of name, degree, score

nme = ["aparna", "pankaj", "sudhir", "Geeku"]

deg = ["MBA", "BCA", "M.Tech", "MBA"]

scr = [90, 40, 80, 98]


# dictionary of lists

dict = {'name': nme, 'degree': deg, 'score': scr}



df = pd.DataFrame(dict)


# saving the

dataframe

df.to_csv('file1.csv')
```

- Output-

	name	degree	score
0	aparna	MBA	90
1	pankaj	BCA	40
2	sudhir	M.Tech	80
3	Geeku	MBA	98

d.Find out the Model Parameters (Slope and Intercept).

I am trying to predict car prices (by machine learning) with a simple linear regression (only one independent variable). The variables are "highway miles per gallon"

```
0      27
1      27
2      26
3      30
4      22
       ..
200    28
201    25
202    23
203    27
204    25
Name: highway-mpg, Length: 205, dtype: int64
```

and "price":

```
0      13495.0
1      16500.0
2      16500.0
3      13950.0
4      17450.0
       ...
200    16845.0
201    19045.0
202    21485.0
203    22470.0
204    22625.0
Name: price, Length: 205, dtype: float64
```

With the following code:

- **Source code--**

```
from sklearn.linear_model import LinearRegression
x = df["highway-mpg"]
y = df["price"]
lm = LinearRegression()

lm.fit([x],[y])
Yhat = lm.predict([x])

print(Yhat)
print(lm.intercept_)
print(lm.coef_)
```

However, the intercept and slope coefficient print commands give me the following output:

- **Output-**

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

e.Evaluate the error in terms of training set.

Example with code:

- **Source code--**

```
import numpy as np
from sklearn import linear_model
```

```
#
##############################################################################
####
# Generate sample data
n_samples_train, n_samples_test, n_features = 75, 150, 500
np.random.seed(0)
coef = np.random.randn(n_features)
coef[50:] = 0.0 # only the top 10 features are impacting the model
X = np.random.randn(n_samples_train + n_samples_test, n_features)
y = np.dot(X, coef)

# Split train and test data
X_train, X_test = X[:n_samples_train], X[n_samples_train:]
y_train, y_test = y[:n_samples_train], y[n_samples_train:]

#
##############################################################################
####
# Compute train and test errors
alphas = np.logspace(-5, 1, 60)
enet = linear_model.ElasticNet(l1_ratio=0.7, max_iter=10000)
train_errors = list()
test_errors = list()
for alpha in
alphas:
    enet.set_params(alpha=alpha)
    enet.fit(X_train, y_train)
    train_errors.append(enet.score(X_train, y_train))
    test_errors.append(enet.score(X_test, y_test))

i_alpha_optim = np.argmax(test_errors)
alpha_optim = alphas[i_alpha_optim]
print("Optimal regularization parameter : %s" % alpha_optim)

# Estimate the coef_ on full data with optimal regularization parameter
enet.set_params(alpha=alpha_optim)
coef_ = enet.fit(X, y).coef_

#
##############################################################################
####
# Plot results functions
import matplotlib.pyplot as plt
plt.subplot(2, 1, 1)
plt.semilogx(alphas, train_errors,
label="Train") plt.semilogx(alphas, test_errors,
label="Test") plt.vlines(
```

```
    alpha_optim,
    plt.ylim()[0],
    np.max(test_errors),
    color="k",
    linewidth=3,
    label="Optimum on test",
)
plt.legend(loc="lower left")
plt.ylim([0, 1.2])
plt.xlabel("Regularization parameter")
plt.ylabel("Performance")

# Show estimated coef_ vs true coef
plt.subplot(2, 1, 2)
plt.plot(coef, label="True coef")
plt.plot(coef_, label="Estimated coef")
plt.legend()
plt.subplots_adjust(0.09, 0.04, 0.94, 0.94, 0.26, 0.26)
plt.show()
```

Total running time of the script: (0 minutes 3.396 seconds)

2. Dataset Import-

a. Import Titanic: Machine Learning from Disaster dataset.

- **In[1]:**

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn import tree
from sklearn.model_selection import train_test_split
import math
import warnings
warnings.filterwarnings('ignore')
```

- **In[2]:**

```
df=pd.read_csv('../input/titanic/train.csv')
res=pd.read_csv('../input/titanic/gender_submission.csv')
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

- **Out[2]:**

- **In[3]:**

```
input1=df[['Pclass','Sex','Age','Fare']]
input1.head()
```

- **Out[3]:**

	Pclass	Sex	Age	Fare
0	3	male	22.0	7.2500
1	1	female	38.0	71.2833
2	3	female	26.0	7.9250
3	1	female	35.0	53.1000
4	3	male	35.0	8.0500

- **In[4]:**

```
target=df['Survived']
target.head()
```

- **Out[4]:**

```
0  0
1  1
2  1
3  1
4  0
Name: Survived, dtype: int64
```

b. Find the total number of records present in the dataset.

count the number of lines in a CSV file in Python-

SAMPLE.CS

V 1,2,3

4,5,6

7,8,9

- **Source code--**

```
file =
open("sample.csv")
reader = csv.reader(file)
lines= len(list(reader))

print(lines)
```

- **Output-**

```
3
```

d.Identify and print categorical and numerical attributes present in the dataset.

We take a peek at the training data with the head() method below.

- **In[1]:**

```
X_train.head()
```

- **Out[1]:**

Rooms	Distance	Postcode	Bedroom2	Bathroom	Landsize	Lattitude	Longtitude	Propertycount
1	5.0	3182.0	1.0	1.0	0.0	-37.85984	144.9867	13240.0
2	8.0	3016.0	2.0	2.0	193.0	-37.85800	144.9005	6380.0
3	12.6	3020.0	3.0	1.0	555.0	-37.79880	144.8220	3755.0
3	13.0	3046.0	3.0	1.0	265.0	-37.70830	144.9158	8870.0
3	13.3	3020.0	3.0	1.0	673.0	-37.76230	144.8272	4217.0

- **In[2]:**

```
# Get list of categorical variables
s = (X_train.dtypes == 'object')
object_cols = list(s[s].index)

print("Categorical
variables:") print(object_cols)
```

- **Out[2]:**

```
Categorical variables:
['Type', 'Method', 'Regionname']
```

e.Find unique values for string variables in the dataset.

Let’s discuss how to get unique values from a column in Pandas DataFrame.

Create a simple dataframe with dictionary of lists, say columns name are A, B, C, D, E with duplicate elements.

	A	B	C	D	E
0	A1	B1	C1	D1	E1
1	A2	B2	C2	D2	E1
2	A3	B3	C3	D2	E1
3	A4	B4	C3	D2	E1
4	A5	B4	C3	D2	E1

Now, let’s get the unique values of a column in this dataframe.

Example #1:

Get the unique values of ‘B’ column

- **Source code--**

```
# Import pandas

package import pandas

as pd


# create a dictionary with five fields each

data = {

   'A':['A1', 'A2', 'A3', 'A4', 'A5'],

   'B':['B1', 'B2', 'B3', 'B4', 'B4'],

   'C':['C1', 'C2', 'C3', 'C3', 'C3'],

   'D':['D1', 'D2', 'D2', 'D2', 'D2'],

   'E':['E1', 'E1', 'E1', 'E1', 'E1'] }


# Convert the dictionary into DataFrame
```

```
df = pd.DataFrame(data)


# Get the unique values of 'B' column
df.B.unique()
```

- **Output-**

```
[ 'B1' , 'B2' , 'B3' , 'B4' ]
```

Example #2: Get the unique values of ‘E’ column

- **Source code--**

```
# Import pandas

package import pandas

as pd


# create a dictionary with five fields each

data = {

   'A':['A1', 'A2', 'A3', 'A4', 'A5'],

   'B':['B1', 'B2', 'B3', 'B4', 'B4'],

   'C':['C1', 'C2', 'C3', 'C3', 'C3'],

   'D':['D1', 'D2', 'D2', 'D2', 'D2'],

   'E':['E1', 'E1', 'E1', 'E1', 'E1'] }


# Convert the dictionary into DataFrame

df = pd.DataFrame(data)


# Get the unique values of 'E' column
df.E.unique()
```

- **Output-**

```
[ 'E1' ]
```

3. Dataset Pre-processing-

a. Find the number of missing values.

Example1: Count total NaN at each column in DataFrame.

- **Source code--**

```
# import numpy library as np

import numpy as np


# import pandas library as pd

import pandas as pd



# List of Tuples

students = [('Ankit', 22, 'Up', 'Geu'),

           ('Ankita', np.NaN, 'Delhi', np.NaN),

           ('Rahul', 16, 'Tokyo', 'Abes'),

           ('Simran', 41, 'Delhi', 'Gehu'),

           ('Shaurya', np.NaN, 'Delhi', 'Geu'),

           ('Shivangi', 35, 'Mumbai', np.NaN ),

           ('Swapnil', 35, np.NaN, 'Geu'),

           (np.NaN, 35, 'Uk', 'Geu'),

           ('Jeet', 35, 'Guj', 'Gehu'),

           (np.NaN, np.NaN, np.NaN, np.NaN)

           ]


# Create a DataFrame object from list of tuples
# with columns and indices.

details = pd.DataFrame(students, columns =['Name', 'Age',

                                           'Place', 'College'],

                       index =['a', 'b', 'c', 'd', 'e',

                               'f', 'g', 'i', 'j', 'k'])
```

```
# show the boolean dataframe

print(" \nshow the boolean Dataframe : \n\n", details.isnull())


# Count total NaN at each column in a DataFrame

print(" \nCount total NaN at each column in a DataFrame : \n\n",

    details.isnull().sum())
```

- **Output-**

```
show the boolean Dataframe :

     Name    Age  Place  College
a  False  False  False    False
b  False   True  False     True
c  False  False  False    False
d  False  False  False    False
e  False   True  False    False
f  False  False  False     True
g  False  False   True    False
i   True  False  False    False
j  False  False  False    False
k   True   True   True     True

Count total NaN at each column in a DataFrame :

 Name       2
Age         3
Place       2
College     3
dtype: int64
```

Example2: Count total NaN in DataFrame.

- **Source code--**

```
# import numpy library as np

import numpy as np


# import pandas library as pd

import pandas as pd




# List of Tuples

students = [('Ankit', 22, 'Up', 'Geu'),

        ('Ankita', np.NaN, 'Delhi', np.NaN),

        ('Rahul', 16, 'Tokyo', 'Abes'),

        ('Simran', 41, 'Delhi', 'Gehu'),

        ('Shaurya', np.NaN, 'Delhi', 'Geu'),

        ('Shivangi', 35, 'Mumbai', np.NaN ),
```

```
        ('Swapnil', 35, np.NaN, 'Geu'),

        (np.NaN, 35, 'Uk', 'Geu'),

        ('Jeet', 35, 'Guj', 'Gehu'),

        (np.NaN, np.NaN, np.NaN, np.NaN)

         ]


# Create a DataFrame object from
# list of tuples with columns
# and indices.

details = pd.DataFrame(students, columns =['Name', 'Age',

                                    'Place', 'College'],

                index =['a', 'b', 'c', 'd', 'e',

                        'f', 'g', 'i', 'j', 'k'])


# show the boolean dataframe

print(" \nshow the boolean Dataframe : \n\n", details.isnull())


# Count total NaN in a DataFrame

print(" \nCount total NaN in a DataFrame : \n\n",

      details.isnull().sum().sum())
```

- **Output-**

```
show the boolean Dataframe :

     Name    Age  Place  College
a  False  False  False    False
b  False   True  False     True
c  False  False  False    False
d  False  False  False    False
e  False   True  False    False
f  False  False  False     True
g  False  False   True    False
i   True  False  False    False
j  False  False  False    False
k   True   True   True     True

Count total NaN in a DataFrame :

 10
```

c.Scale numeric variables into specified range.

- **Source code--**

```
# Find the min and max values for each column
def dataset_minmax(dataset):
        minmax = list()
        for i in range(len(dataset[0])):
```

```
            col_values = [row[i] for row in dataset]
            value_min = min(col_values)
            value_max = max(col_values)
            minmax.append([value_min, value_max])
    return minmax

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

# Contrive small dataset
dataset = [[50, 30], [20, 90]]
print(dataset)
# Calculate min and max for each column
minmax = dataset_minmax(dataset)
print(minmax)
# Normalize columns
normalize_dataset(dataset, minmax)
print(dataset)
```

Running this example prints the output below, including the normalized dataset.

- **Output-**

```
[[50, 30], [20, 90]]
[[20, 50], [30, 90]]
[[1, 0], [0, 1]]
```

4. Understanding of the Dataset-

a. Use bar chart for plotting number of missing values for each attribute.

Example: Visualizing missing data for all columns

- **Source code--**

```
age = c(12,34,NA,7,15,NA)

name = c('rob',NA,"arya","jon",NA,NA)

grade = c("A","A","D","B","C","B")

df <- data.frame(age,name,grade)

# function convert dataframe to binary TRUE/FALSE matrix

toBinaryMatrix <- function(df){

m<-c()
```

```
for(i in colnames(df)){

   x<-sum(is.na(df[,i])

   )

   # missing value

   count

   m<-append(m,x)

   # non-missing value count

   m<-append(m,nrow(df)-x)
}

# adding column and row names to

matrix a<-matrix(m,nrow=2)

rownames(a)<-c("TRUE","FALSE"

) colnames(a)<-colnames(df)


return(a)
}

# function call

binMat = toBinaryMatrix(df)
binMat
```

- **Output-**

```
      age   name grade
TRUE    2      3        0
FALSE   4     3        6
```

Stacked barplot-

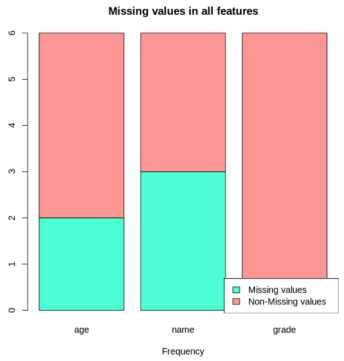
- **Source code--**

```
age = c(12,34,NA,7,15,NA)

name = c('rob',NA,"arya","jon",NA,NA)

grade = c("A","A","D","B","C","B")

df <- data.frame(age,name,grade)
```

stacked barplot for missing data in all columns

```
barplot(binMat,

main = "Missing values in all features",xlab = "Frequency",

col = c("#4dffd2","#ff9999"))

# legend for barplot

legend("bottomright",

c("Missing values","Non-Missing values"),

fill = c("#4dffd2","#ff9999"))
```

- **Output-**



b.Use pie chart for plotting percentage of survivals.

- **Source code--**

```
# Import libraries

import numpy as

np

import matplotlib.pyplot as plt


# Creating dataset

cars = ['AUDI', 'BMW', 'FORD',

        'TESLA', 'JAGUAR', 'MERCEDES']


data = [23, 17, 35, 29, 12, 41]


# Creating explode data
```

```
explode = (0.1, 0.0, 0.2, 0.3, 0.0, 0.0)

# Creating color parameters

colors = ( "orange", "cyan", "brown",

        "grey", "indigo", "beige")

# Wedge properties

wp = { 'linewidth' : 1, 'edgecolor' : "green" }

# Creating autocpt arguments

def func(pct, allvalues):

    absolute = int(pct / 100.*np.sum(allvalues))

    return "{:.1f}%\n({:d} g)".format(pct, absolute)

# Creating plot

fig, ax = plt.subplots(figsize =(10, 7))

wedges, texts, autotexts = ax.pie(data,

                                  autopct = lambda pct: func(pct, data),

                                  explode = explode,

                                  labels = cars,

                                  shadow = True,

                                  colors = colors,

                                  startangle = 90,

                                  wedgeprops =

                                  wp,

                                  textprops = dict(color ="magenta"))

# Adding legend
ax.legend(wedges, cars,

        title ="Cars",
```

loc ="center left",

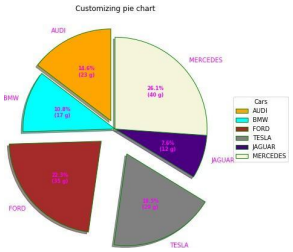
bbox_to_anchor =(1, 0, 0.5, 1))

plt.setp(autotexts, size = 8, weight ="bold")

ax.set_title("Customizing pie chart")

# show plot
plt.show()

- **Output-**



5. Retrieve Train and Test Dataset-

a.Partition the original dataset into train and test dataset with the ratio of 8:2.

- **Source code--**

```
# split imbalanced dataset into train and test sets without stratification
from collections import Counter
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
# create dataset
X, y = make_classification(n_samples=100, weights=[0.80], flip_y=0, random_state=1)
print(Counter(y))
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.50, random_state=1)
print(Counter(y_train))
print(Counter(y_test))
```

Running the example first reports the composition of the dataset by class label, showing the expected 80 percent vs. 20 percent.

- **Output-**

Counter({0: 80, 1:
20}) Counter({0: 45,
1: 5}) Counter({0: 49,
1: 1})

b.Partition the original dataset into train and test dataset with the ratio of 7:3.

- **Source code--**

```
# split imbalanced dataset into train and test sets without stratification
from collections import Counter
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
# create dataset
X, y = make_classification(n_samples=100, weights=[0.70], flip_y=0, random_state=1)
```

```
print(Counter(y))
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.50, random_state=1)
print(Counter(y_train))
print(Counter(y_test))
```

- **Output-**

```
Counter({0: 70, 1:
30}) Counter({0: 45,
1: 5}) Counter({0: 49,
1: 1})
```

6. Predict the chance of survival-
a. Apply Bayesian classification methods
[Use Multinomial and Gaussian Naïve Bayes for classification].

Gaussian Naïve Bayes for classification
Now, we look at an implementation of Gaussian Naive Bayes classifier using scikit-learn.

- **Source code--**

```
# load the iris dataset

from sklearn.datasets import load_iris

iris = load_iris()

# store the feature matrix (X) and response vector (y)

X = iris.data

y =

iris.target

# splitting X and y into training and testing sets

from sklearn.model_selection import

train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)

# training the model on training set

from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(X_train, y_train)

# making predictions on the testing set
```

```
y_pred = gnb.predict(X_test)
```

```
# comparing actual response values (y_test) with predicted response values (y_pred)

from sklearn import metrics

print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test,
y_pred)*100)
```

- **Output-**

```
Gaussian Naive Bayes model accuracy(in %): 95.0
```

Multinomial Naive Bayes Classifier-

Combining probability distribution of P with fraction of documents belonging to each class.
For class j, word i at a word frequency of f,
Even though the stop words have already been set to 0 for this specific use case, the IDF implementation is being added to generalize the function.

- **Source code--**

```
#Calculate IDF
tot = len(df['docIdx'].unique())
pb_ij = df.groupby(['wordIdx'])
IDF = np.log(tot/pb_ij['docIdx'].count())
IDF_dict = IDF.to_dict()
def MNB(df, smooth = False, IDF = False):
    '''
    Multinomial Naive Bayes classifier
    :param df [Pandas Dataframe]: Dataframe of data
    :param smooth [bool]: Apply Smoothing if True
    :param IDF [bool]: Apply Inverse Document Frequency if True
    :return predict [list]: Predicted class ID
    '''
    #Using dictionaries for greater speed
    df_dict = df.to_dict()
    new_dict = {}
    prediction =
    []

    #new_dict = {docIdx : {wordIdx: count},    }
    for idx in range(len(df_dict['docIdx'])):
        docIdx = df_dict['docIdx'][idx]
        wordIdx = df_dict['wordIdx'][idx]
        count = df_dict['count'][idx]
        try:
            new_dict[docIdx][wordIdx] = count
        except:
            new_dict[df_dict['docIdx'][idx]] = {}
            new_dict[docIdx][wordIdx] = count

    #Calculating the scores for each doc
    for docIdx in range(1,
    len(new_dict)+1):
        score_dict = {}
```

```
        #Creating a probability row for each class
        for classIdx in range(1,21):
            score_dict[classIdx] = 1
            #For each word:
            for wordIdx in new_dict[docIdx]:
                #Check for frequency smoothing
                #log(1+f)*log(Pr(i|j))
                if smooth:
                    try:
                        probability=Pr_dict[wordIdx][classIdx]
                        power = np.log(1+ new_dict[docIdx][wordIdx])
                        #Check for IDF
                        if IDF:
                            score_dict[classIdx]+=(
                              power*np.log(
                              probability*IDF_dict[wordIdx]))
                        else:
                            score_dict[classIdx]+=power*np.log(
                                            probability)
                    except:
                        #Missing V will have log(1+0)*log(a/16689)=0
                        score_dict[classIdx] += 0
                #f*log(Pr(i|j))
                else:
                    try:
                        probability = Pr_dict[wordIdx][classIdx]
                        power = new_dict[docIdx][wordIdx]
                        score_dict[classIdx]+=power*np.log(
                                        probability
                        ) #Check for IDF
                        if IDF:
                            score_dict[classIdx]+= power*np.log(
                                 probability*IDF_dict[wordIdx])
                    except:
                        #Missing V will have 0*log(a/16689) = 0
                        score_dict[classIdx] += 0
            #Multiply final with pi
            score_dict[classIdx] += np.log(pi[classIdx])

        #Get class with max probabilty for the given docIdx
        max_score = max(score_dict, key=score_dict.get)
        prediction.append(max_score)

    return prediction
```

Comparing the effects of smoothing and IDF:

- **Source code--**

```
regular_predict = MNB(df, smooth=False, IDF=False)
```

```
smooth_predict = MNB(df, smooth=True, IDF=False)
tfidf_predict = MNB(df, smooth=False, IDF=True)
all_predict     = MNB(df, smooth=True, IDF=True)
#Get list of labels
train_label = pd.read_csv('20news-bydate/matlab/train.label',
                      names=['t'])
train_label= train_label['t'].tolist()
total = len(train_label)
models = [regular_predict, smooth_predict,
         tfidf_predict, all_predict]
strings = ['Regular', 'Smooth', 'IDF', 'Both']

for m,s in zip(models,strings):
   val = 0
   for i,j in zip(m, train_label):
      if i != j:
         val
      +=1 else:
         pass
   print(s,"Error:\t\t",val/total * 100, "%")
```

- **Output-**

```
Regular Error:.        1.6771674505279 %
Smooth Error:.         1.2867157689235 %
IDF Error:.            1.6949152542372 %
Both Error:.           1.2867157689235 %
```

c.Apply Decision Tree classifier-
Try this classifier model using different criterions and splitter.

Decision Tree Classifier Building in Scikit-learn-

- **Source code--**

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

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

- **Output-**

Accuracy:
0.7705627705627706

Visualizing Decision Trees-

```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names =
feature_cols,class_names=['0','1']) graph =
pydotplus.graph_from_dot_data(dot_data.getvalue()) graph.write_png('diabetes.png')
Image(graph.create_png())
```

d.Apply Logistic Regression on the Titanic Dataset to predict the probability of survival of the passengers.

Importing the Libraries-
Open in
app Get
started

Towards Data
Science Published in
Towards Data Science

.
Follo
w
Predicting the Survival of Titanic Passengers

In this blog-post, I will go through the whole process of creating a machine learning model on the famous Titanic dataset, which is used by many people all over the world. It provides information on the fate of passengers on the Titanic, summarized according to economic status (class), sex, age and survival.
I initially wrote this post on kaggle.com, as part of the “Titanic: Machine Learning from Disaster” Competition. In this challenge, we are asked to predict whether a passenger on the titanic would have been survived or not.
RMS Titanic
The RMS Titanic was a British passenger liner that sank in the North Atlantic Ocean in the early morning hours of 15 April 1912, after it collided with an iceberg during its maiden voyage from Southampton to New York City. There were an estimated 2,224 passengers and crew aboard the ship, and more than 1,500 died, making it one of the deadliest commercial peacetime maritime disasters in modern history. The RMS Titanic was the largest ship afloat at the time it entered service and was the second of three Olympic-class ocean liners operated by the White Star Line. The Titanic was built by the Harland and Wolff shipyard in Belfast. Thomas Andrews, her architect, died in the disaster.

Importing the Libraries

```
# linear algebra
import numpy as
np

# data processing
import pandas as
pd

# data visualization
import seaborn as
sns
%matplotlib inline
from matplotlib import pyplot as plt
from matplotlib import style

# Algorithms
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import
KNeighborsClassifier from sklearn.svm import
SVC, LinearSVC
from sklearn.naive_bayes import GaussianNB
```

Getting the Data-

```
test_df = pd.read_csv("test.csv")
train_df = pd.read_csv("train.csv")
```

Data Exploration/Analysis-

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

The training-set has 891 examples and 11 features + the target variable (survived). 2 of the features are floats, 5 are integers and 5 are objects. Below I have listed the features with a short description:

survival: Survival
PassengerId: Unique Id of a
passenger. pclass: Ticket class
sex: Sex
Age: Age in years
sibsp: # of siblings / spouses aboard the Titanic
parch: # of parents / children aboard the Titanic
ticket: Ticket number
fare: Passenger fare
cabin: Cabin
number
embarked: Port of
Embarkation train_df.describe()

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Above we can see that **38% out of the training-set survived the Titanic.** We can also see that the passenger ages range from 0.4 to 80.

9. Import the Dataset "StudentPerformance" and then: a.Apply pre-processing techniques (if applicable)-

- **In[1]:**

```
import pandas as
pd import numpy as
np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import norm
from scipy.stats import kurtosis, skew
%matplotlib inline
from pylab import rcParams
import statsmodels.api as sm
from scipy.stats import linregress
import pylab
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from sklearn import preprocessing

plt.rcParams["figure.figsize"] = 8,4
import warnings
warnings.filterwarnings("ignore")
```

Using TensorFlow backend.

- **In[2]:**

```
df = pd.read_csv("../input/StudentsPerformance.csv")
#df = pd.read_csv("StudentsPerformance.csv")
```

- **In[3]:**

```
df.head(5)
```

  - **Out[3]:**

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

- **In[4]:**

```
# Check for possible null values in the dataset as missing values potentially falsify the
statistical models
df.isnull().sum()
# the dataset does not contain missing values
```

  - **Out[4]:**

```
gender                       0
race/ethnicity               0
parental level of education  0
lunch                        0
test preparation course      0
math score                   0
reading score                0
writing score                0
dtype: int64
```
