Experiment No 5

* 1. **Aim/Purpose of the Experiment**

To familiarize the students with Model building using Logistic Regression.

* 1. **Learning Outcomes**

Knowledge of the Data cleaning, modelling using logistic regression, and different libraries in python.

* 1. **Prerequisites**

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

* 1. **Materials/Equipment/Apparatus / Devices/Software required**

Jupyter Notebook.

* 1. **Introduction and Theory**

Logistic regression is a statistical method used for modeling the probability of a binary outcome based on one or more predictor variables. It's widely used for classification tasks where the dependent variable is categorical and has two possible outcomes.

Here's an overview of the key concepts and components of logistic regression:

* Binary outcome: Logistic regression is specifically designed for situations where the dependent variable (also known as the response variable or target variable) is binary, meaning it has only two possible outcomes. These outcomes are typically represented as 0 and 1, or as "success" and "failure", "yes" and "no", etc.
* Logistic function (sigmoid function): In logistic regression, the relationship between the predictor variables and the probability of the binary outcome is modeled using the logistic function, also known as the sigmoid function. The logistic function is an S-shaped curve that maps any real-valued number to a value between 0 and 1, representing probabilities.
* Probability prediction: Unlike linear regression, where the output is continuous, logistic regression predicts the probability that a given observation belongs to a particular category (e.g., the probability of a customer buying a product). The predicted probabilities are then used to make classifications.
* Logit transformation: The logistic function is expressed in terms of the log-odds, also known as the logit function. The logit of the probability of the event occurring (p) is defined as the logarithm of the odds ratio (p / (1 - p)). Mathematically, it can be represented as log(p / (1 - p)).
* Model parameters: Similar to linear regression, logistic regression estimates parameters (coefficients) that define the relationship between the predictor variables and the log-odds of the binary outcome. These parameters are estimated using maximum likelihood estimation or other optimization techniques.
* Interpretation of coefficients: The coefficients obtained from logistic regression represent the change in the log-odds of the outcome associated with a one-unit change in the corresponding predictor variable, holding other variables constant.
* Decision boundary: In logistic regression, a decision boundary is used to classify observations into different categories based on their predicted probabilities. The decision boundary is typically set at 0.5, meaning that observations with predicted probabilities greater than 0.5 are classified into one category, while those with predicted probabilities less than or equal to 0.5 are classified into the other category.

**Logistic Regression**

Step 1: Importing and Merging Data

# Suppressing Warnings

import warnings

warnings.filterwarnings

# Importing Pandas and NumPy

import pandas as pd, numpy as np

# Importing all datasets

churn\_data = pd.read\_csv("churn\_data.csv")

churn\_data.head()

customer\_data = pd.read\_csv("customer\_data.csv")

customer\_data.head()

internet\_data = pd.read\_csv("internet\_data.csv")

internet\_data.head()

#Combining all data files into one consolidated dataframe

# Merging on 'customerID'

df\_1 = pd.merge(churn\_data, customer\_data, how='inner', on='customerID')

# Final dataframe with all predictor variables

telecom = pd.merge(df\_1, internet\_data, how='inner', on='customerID')

Step 2: Inspecting the Dataframe

# Let's see the head of our master dataset

telecom.head()

# Let's check the dimensions of the dataframe

telecom.shape

# let's look at the statistical aspects of the dataframe

telecom.describe()

# Let's see the type of each column

telecom.info()

Step 3: Data Preparation

#Converting some binary variables (Yes/No) to 0/1

# List of variables to map

varlist = ['PhoneService', 'PaperlessBilling', 'Churn', 'Partner', 'Dependents']

# Defining the map function

def binary\_map(x):

return x.map({'Yes': 1, "No": 0})

# Applying the function to the housing list

telecom[varlist] = telecom[varlist].apply(binary\_map)

telecom.head()

#For categorical variables with multiple levels, create dummy features (one-hot encoded)

# Creating a dummy variable for some of the categorical variables and dropping the first one.

dummy1 = pd.get\_dummies(telecom[['Contract', 'PaymentMethod', 'gender', 'InternetService']], drop\_first=True)

# Adding the results to the master dataframe

telecom = pd.concat([telecom, dummy1], axis=1)

telecom.head()

# Creating dummy variables for the remaining categorical variables and dropping the level with big names.

# Creating dummy variables for the variable 'MultipleLines'

ml = pd.get\_dummies(telecom['MultipleLines'], prefix='MultipleLines')

# Dropping MultipleLines\_No phone service column

ml1 = ml.drop(['MultipleLines\_No phone service'], 1)

#Adding the results to the master dataframe

telecom = pd.concat([telecom,ml1], axis=1)

# Creating dummy variables for the variable 'OnlineSecurity'.

os = pd.get\_dummies(telecom['OnlineSecurity'], prefix='OnlineSecurity')

os1 = os.drop(['OnlineSecurity\_No internet service'], 1)

# Adding the results to the master dataframe

telecom = pd.concat([telecom,os1], axis=1)

# Creating dummy variables for the variable 'OnlineBackup'.

ob = pd.get\_dummies(telecom['OnlineBackup'], prefix='OnlineBackup')

ob1 = ob.drop(['OnlineBackup\_No internet service'], 1)

# Adding the results to the master dataframe

telecom = pd.concat([telecom,ob1], axis=1)

# Creating dummy variables for the variable 'DeviceProtection'.

dp = pd.get\_dummies(telecom['DeviceProtection'], prefix='DeviceProtection')

dp1 = dp.drop(['DeviceProtection\_No internet service'], 1)

# Adding the results to the master dataframe

telecom = pd.concat([telecom,dp1], axis=1)

# Creating dummy variables for the variable 'TechSupport'.

ts = pd.get\_dummies(telecom['TechSupport'], prefix='TechSupport')

ts1 = ts.drop(['TechSupport\_No internet service'], 1)

# Adding the results to the master dataframe

telecom = pd.concat([telecom,ts1], axis=1)

# Creating dummy variables for the variable 'StreamingTV'.

st =pd.get\_dummies(telecom['StreamingTV'], prefix='StreamingTV')

st1 = st.drop(['StreamingTV\_No internet service'], 1)

# Adding the results to the master dataframe

telecom = pd.concat([telecom,st1], axis=1)

# Creating dummy variables for the variable 'StreamingMovies'.

sm = pd.get\_dummies(telecom['StreamingMovies'], prefix='StreamingMovies')

sm1 = sm.drop(['StreamingMovies\_No internet service'], 1)

# Adding the results to the master dataframe

telecom = pd.concat([telecom,sm1], axis=1)

telecom.head()

Dropping the repeated variables

# We have created dummies for the below variables, so we can drop them

telecom = telecom.drop(['Contract','PaymentMethod','gender','MultipleLines','InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',

'TechSupport', 'StreamingTV', 'StreamingMovies'], 1)

#The varaible was imported as a string we need to convert it to float

telecom['TotalCharges'] = telecom['TotalCharges'].convert\_objects(convert\_numeric=True)

telecom.info()

#Checking for Outliers

# Checking for outliers in the continuous variables

num\_telecom = telecom[['tenure','MonthlyCharges','SeniorCitizen','TotalCharges']]

#Checking for Missing Values and Inputing Them

# Adding up the missing values (column-wise)

telecom.isnull().sum()

# Checking the percentage of missing values

round(100\*(telecom.isnull().sum()/len(telecom.index)), 2)

# Removing NaN TotalCharges rows

telecom = telecom[~np.isnan(telecom['TotalCharges'])]

# Checking percentage of missing values after removing the missing values

round(100\*(telecom.isnull().sum()/len(telecom.index)), 2)

Step 4: Test-Train Split

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

# Putting feature variable to X

X = telecom.drop(['Churn','customerID'], axis=1)

X.head()

# Putting response variable to y

y = telecom['Churn']

y.head()

# Splitting the data into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.7, test\_size=0.3, random\_state=100)

Step 5: Feature Scaling

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train[['tenure','MonthlyCharges','TotalCharges']] = scaler.fit\_transform(X\_train[['tenure','MonthlyCharges','TotalCharges']])

X\_train.head()

### Checking the Churn Rate

churn = (sum(telecom['Churn'])/len(telecom['Churn'].index))\*100

churn

Step 6: Looking at Correlations

# Importing matplotlib and seaborn

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

# Let's see the correlation matrix

plt.figure(figsize = (20,10)) # Size of the figure

sns.heatmap(telecom.corr(),annot = True)

plt.show()

#Dropping highly correlated dummy variables

X\_test = X\_test.drop(['MultipleLines\_No','OnlineSecurity\_No','OnlineBackup\_No','DeviceProtection\_No','TechSupport\_No','StreamingTV\_No','StreamingMovies\_No'], 1)

X\_train = X\_train.drop(['MultipleLines\_No','OnlineSecurity\_No','OnlineBackup\_No','DeviceProtection\_No','TechSupport\_No', 'StreamingTV\_No','StreamingMovies\_No'], 1)

#Checking the Correlation Matrix

#After dropping highly correlated variables now let's check the correlation matrix again.

plt.figure(figsize = (20,10))

sns.heatmap(X\_train.corr(),annot = True)

plt.show()

Step 7: Model Building

import statsmodels.api as sm

# Logistic regression model

logm1 = sm.GLM(y\_train,(sm.add\_constant(X\_train)), family = sm.families.Binomial())

logm1.fit().summary()

* 1. **Operating Procedure**
* Open Jupyter note book
* Take a new python file
* Type the code
* Run it
* Take inputs from user
* Observe the results
* Verify the results manually
* Store the note book file
  1. **Precautions and/or Troubleshooting**

**Precautions:**

* Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).
* Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."
* Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.
* Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.
* Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.
* Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.
* Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

**Troubleshooting:**

* Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.
* Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.
* Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.
* Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.
* Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.
* Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.
  1. **Observations**

Observe the results obtained in each operation.

* 1. **Calculations & Analysis**

Calculations should be given for each operation.

* 1. **Result & Interpretation**

Result should be printed and pasted in laboratory copy found from Jupyter note book.

* 1. **Follow-up Questions**
* What do you mean by the Logistic Regression?
* What are the different types of Logistic Regression?
* What are the odds?
* What is the Impact of Outliers on Logistic Regression?
* What is the difference between the outputs of the Logistic model and the Logistic function?
* How do we handle categorical variables in Logistic Regression?
* What are the assumptions made in Logistic Regression?
* Why is Logistic Regression termed as Regression and not classification?
* What are the advantages of Logistic Regression?
* What are the disadvantages of Logistic Regression?
  1. **Extension and Follow-up Activities (if applicable)**

NA

* 1. **Assessments**
  2. **Suggested reading**

NA