Department of Information Engineering Technology

The Superior University, Lahore

Artificial Intelligence Lab

Experiment No.11

Executive the Customer Churn Prediction using Artificial Neural Network

Prepared for

_____

By:

Name: _____

ID: _____

Section: _____

Semester: _____

Total Marks: _____

Obtained Marks: _____

Signature: _____

Date: _____

Experiment No. 11

Executive the Customer Churn Prediction using Artificial Neural Network (Rubrics)

Name: _____ **Roll No. : _____**

A. PSYCHOMOTOR

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Follow Procedures	1	0	0.25	0.5	0.75	1	
2	Software and Simulations	2	0	0.5	1	1.5	2	
3	Accuracy in Output Results	3	0	0.75	1.5	2.25	3	
Sub Total		**6**	**Sub Total marks Obtained in Psychomotor(P)**					

B. AFFECTIVE

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Respond to Questions	1	0	0.25	0.5	0.75	1	
2	Lab Report	1	0	0.25	0.5	0.75	1	
3	Assigned Task	2	0	0.5	1	1.5	2	
Sub Total		**4**	**Sub Total marks Obtained in Affective (A)**					

Instructor Name:_____ **Total Marks (P+A): _____**

Instructor Signature:_____ **Obtained Marks (P+A): _____**

Introduction to Customer Churn Prediction

Customer Churn Prediction is a crucial task for businesses, especially in industries where customer retention is vital for sustained growth. Churn refers to the phenomenon where customers stop using the products or services of a business, and predicting it can help companies take proactive measures to retain customers and maintain a stable customer base.

Importance of Customer Churn Prediction

Business Sustainability: Retaining existing customers is often more cost-effective than acquiring new ones. Predicting churn allows businesses to identify and address issues before customers decide to leave.

Revenue Impact: Losing customers means losing revenue. Churn prediction helps in implementing targeted strategies to prevent revenue loss.

Customer Satisfaction: Identifying dissatisfied customers early on enables businesses to address concerns, improve services, and enhance overall customer satisfaction.

Conclusion

Customer Churn Prediction is an essential strategy for businesses aiming to maintain a loyal customer base. By leveraging machine learning and predictive analytics, companies can proactively address issues, enhance customer satisfaction, and ultimately improve their bottom line.

BUILDING A MODEL

Here's how it will work:

Import the libraries

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

Load Dataset

```
df = pd.read_csv("customer_churn.csv")
df.sample(5)
```

Drops the 'customerID' column as it is considered not useful.

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

Checking the data types of the columns.

```
df.dtypes
```

Prints the values in the 'TotalCharges' column.

```
df.TotalCharges.values
```

Attempts to convert the 'TotalCharges' column to numeric values.

```
pd.to_numeric(df.TotalCharges)
```

Identifies rows where the conversion to numeric values resulted in null values.

```
pd.to_numeric(df.TotalCharges,errors='coerce').isnull()
```

This line of code is used to identify and display the rows in the DataFrame df where the conversion of the 'TotalCharges' column to numeric values resulted in null values.

```
df[pd.to_numeric(df.TotalCharges,errors='coerce').isnull()]
```

This line prints the shape of the DataFrame df, indicating the number of rows and columns.

```
df.shape
```

This line retrieves the value in the 'TotalCharges' column for the row with index 488.

```
df.iloc[488].TotalCharges
```

This line filters the DataFrame df to include only the rows where the 'TotalCharges' column is not a blank string, and then prints the shape of this filtered DataFrame. This operation essentially removes rows where 'TotalCharges' is a blank string.

```
df[df.TotalCharges!=' '].shape
```

Remove rows with space in TotalCharges

This line creates a new DataFrame df1 by filtering out the rows from the original DataFrame df where the 'TotalCharges' column is not a blank string.

```
df1 = df[df.TotalCharges!=' ']
df1.shape
```

This line prints the data types of each column in the DataFrame df1.

```
df1.dtypes
```

Converts the 'TotalCharges' column in the DataFrame df1 to numeric values. This is done to ensure that the 'TotalCharges' column contains only numeric data.

```
df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

Prints the values in the 'TotalCharges' column after the conversion to numeric.

```
df1.TotalCharges.values
```

Filters the DataFrame df1 to include only the rows where the 'Churn' column is 'No', indicating customers who did not churn. This is a subset of the DataFrame for customers who did not churn.

```
df1[df1.Churn=='No']
```

Data Visualization

Visualizing Customer Churn based on Tenure

```
tenure_churn_no = df1[df1.Churn=='No'].tenure
tenure_churn_yes = df1[df1.Churn=='Yes'].tenure

plt.xlabel("tenure")
plt.ylabel("Number Of Customers")
plt.title("Customer Churn Prediction Visualiztion")

blood_sugar_men = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82,
129]
blood_sugar_women = [67, 98, 89, 120, 133, 150, 84, 69, 89, 79, 120,
112, 100]

plt.hist([tenure_churn_yes, tenure_churn_no], rwidth=0.95,
color=['green','red'],label=['Churn=Yes','Churn=No'])
plt.legend()
```

Visualizing Customer Churn based on Monthly Charges.

```
mc_churn_no = df1[df1.Churn=='No'].MonthlyCharges
mc_churn_yes = df1[df1.Churn=='Yes'].MonthlyCharges

plt.xlabel("Monthly Charges")
plt.ylabel("Number Of Customers")
plt.title("Customer Churn Prediction Visualiztion")

blood_sugar_men = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82,
129]
blood_sugar_women = [67, 98, 89, 120, 133, 150, 84, 69, 89, 79, 120,
112, 100]

plt.hist([mc_churn_yes, mc_churn_no], rwidth=0.95,
color=['green','red'],label=['Churn=Yes','Churn=No'])
plt.legend()
```

Function to Print Unique Values for Object Columns

This function takes a DataFrame df as an argument and iterates through its columns. If the data type of a column is 'object', it prints the column name and its unique values.

```
def print_unique_col_values(df):
       for column in df:
            if df[column].dtypes=='object':
                print(f'{column}: {df[column].unique()}')
print_unique_col_values(df1)
```

Replacement of 'No internet service' and 'No phone service' with 'No'

These lines replace occurrences of 'No internet service' and 'No phone service' in the DataFrame df1 with 'No'. This is done to simplify the values and make the data more consistent.

```
df1.replace('No internet service','No',inplace=True)
df1.replace('No phone service','No',inplace=True)
```

After the replacement, this line uses the previously defined function to print unique values for object columns in the modified DataFrame df1.

```
print_unique_col_values(df1)
```

Converting Yes and No to 1 or 0

The first loop iterates through a list of columns specified in yes_no_columns.

It uses the replace method to replace 'Yes' with 1 and 'No' with 0 in each of these columns.

The second loop prints the unique values for all columns in the DataFrame df1 after the replacement.

This process essentially converts categorical binary columns into numerical format.

```
yes_no_columns =
['Partner','Dependents','PhoneService','MultipleLines','OnlineSecurity',
'OnlineBackup','DeviceProtection','TechSupport','StreamingTV','Streaming
Movies','PaperlessBilling','Churn']
for col in yes_no_columns:
    df1[col].replace({'Yes': 1,'No': 0},inplace=True)
for col in df1:
    print(f'{col}: {df1[col].unique()}')
```

Converting 'gender' to 1 or 0

This code segment specifically converts the 'gender' column, replacing 'Female' with 1 and 'Male' with 0.

It then prints the unique values in the 'gender' column.

```
df1['gender'].replace({'Female':1,'Male':0},inplace=True)
df1.gender.unique()
```

One-Hot Encoding for Categorical Columns

pd.get_dummies is used to perform one-hot encoding on specified categorical columns ('InternetService', 'Contract', 'PaymentMethod') in the DataFrame df1.

The resulting DataFrame is stored in df2.

df2.columns prints the column names after one-hot encoding.

df2.sample(5) displays a random sample of 5 rows from the one-hot encoded DataFrame.

df2.dtypes prints the data types of each column in the one-hot encoded DataFrame

```
df2 = pd.get_dummies(data=df1,
columns=['InternetService','Contract','PaymentMethod'])
df2.columns
df2.sample(5)
df2.dtypes
```

cols_to_scale is a list of column names containing numerical values that need to be scaled (tenure, MonthlyCharges, TotalCharges).

MinMaxScaler from scikit-learn is imported to perform Min-Max scaling, which scales the values in each column to a range between 0 and 1.

scaler.fit_transform(df2[cols_to_scale]) applies the scaling transformation to the specified columns and updates the corresponding values in the DataFrame df2.

```
cols_to_scale = ['tenure','MonthlyCharges','TotalCharges']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df2[cols_to_scale] = scaler.fit_transform(df2[cols_to_scale])
```

This loop prints the unique values for all columns in the DataFrame df2 after the Min-Max scaling operation.

```
for col in df2:
    print(f'{col}: {df2[col].unique()}')
```

Train-Test Split

This code segment is responsible for splitting the data into training and testing sets using the train_test_split function from scikit-learn.

```
X = df2.drop('Churn',axis='columns')
y = df2['Churn']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.2,random_state=5)
```

Building the Artificial Neural Network (ANN) Model

This code defines a sequential model:

- The first layer has 26 units (neurons), an input shape of 26 (matching the number of features), and a ReLU activation function.
- The second layer has 15 units and a ReLU activation function.
- The third layer has 1 unit with a sigmoid activation function, suitable for binary classification tasks like predicting customer churn.
- This line compiles the model, specifying:
- optimizer='adam': Adam optimizer is used for optimization.

- loss='binary_crossentropy': Binary crossentropy is chosen as the loss function, suitable for binary classification.
- metrics=['accuracy']: The accuracy metric is used to monitor the performance during training.
- This line trains the model using the training data (X_train and y_train) for 100 epochs.

```
import tensorflow as tf
from tensorflow import keras


model = keras.Sequential([
    keras.layers.Dense(26, input_shape=(26,), activation='relu'),
    keras.layers.Dense(15, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# opt = keras.optimizers.Adam(learning_rate=0.01)

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=100)
```

This line evaluates the trained model on the test data (X_test and y_test)

```
model.evaluate(X_test, y_test)
```

This line uses the trained model to predict churn probabilities for the test set (X_test).

```
yp = model.predict(X_test)
yp[:5]
```

This loop converts the predicted probabilities (yp) to binary predictions (y_pred) using a threshold of 0.5.

```
y_pred = []
for element in yp:
    if element > 0.5:
        y_pred.append(1)
    else:
```

```
        y_pred.append(0)
y_pred[:10]
```

This line prints the actual target values from the test set for the first 10 samples.

```
y_test[:10]
```

The classification_report function from scikit-learn is used to generate a text report showing the main classification metrics such as precision, recall, and F1-score for each class (0 and 1).

```
from sklearn.metrics import confusion_matrix , classification_report
print(classification_report(y_test,y_pred))
```

tf.math.confusion_matrix generates a confusion matrix using TensorFlow. This matrix shows the counts of true positive, true negative, false positive, and false negative predictions.

seaborn is used to create a heatmap visualization of the confusion matrix.

```
import seaborn as sn
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_pred)
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

This line prints the accuracy.

```
round((862+229)/(862+229+137+179),2)
```

Calculates precision for the 0 class, i.e., precision for customers who did not churn.

```
round(862/(862+179),2)
```

Calculates recall for the 0 class.

```
round(862/(862+137),2)
```

Task:

Take this dataset for bank customer churn prediction: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling

1) Build a deep learning model to predict churn rate at bank.

2) Once model is built, print classification report and analyze precision, recall and f1-score