A BRIEF PROJECT REPORT ON THE TITLE

CUSTOMER CHURN PREDICTION

Done by – Syed Rayan

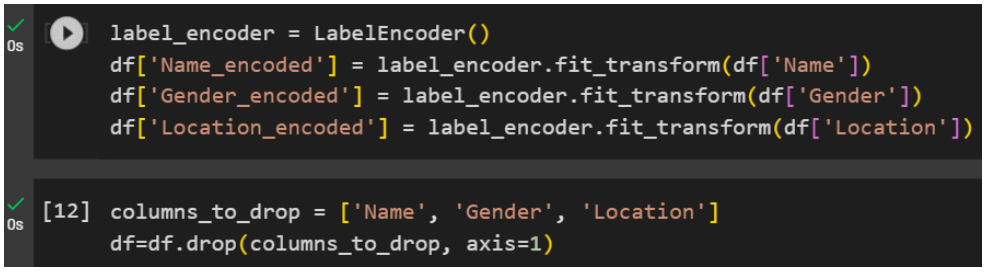
Email Id – 160420aid032@mjcollege.ac.in

Dataset used – Customer Churn Large Dataset (CSV File)

This project report is divided into stages and explained simultaneously.

Stage – 1: Data Preprocessing

1. Firstly, Numpy and Pandas libraries are imported for mathematical operations and creating data frames respectively.
2. Then I have created a data frame "df" and loaded the given dataset into it.
3. Basic data exploratory information was then produced like df.head(), df.dtypes, df.isnull().sum(), df.describe() etc., to get the over-view of the dataset.
4. By using label encoder, I have then encoded the attributes that were of "string" type ('Name', 'Gender', 'Location') into numerical type and gave new attribute names to them.
5. As we now have new numerical attributes, the older attributes are dropped by using the 'df.drop()' function.

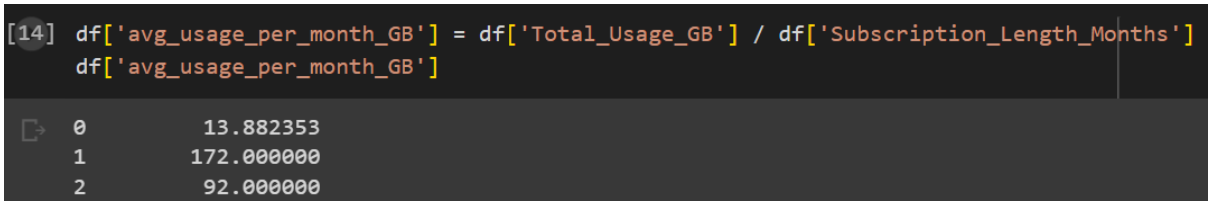


```
label_encoder = LabelEncoder()
df['Name_encoded'] = label_encoder.fit_transform(df['Name'])
df['Gender_encoded'] = label_encoder.fit_transform(df['Gender'])
df['Location_encoded'] = label_encoder.fit_transform(df['Location'])

[12] columns_to_drop = ['Name', 'Gender', 'Location']
df=df.drop(columns_to_drop, axis=1)
```

Stage – 2: Feature Engineering

1. I have calculated 'avg_usage_per_month_GB' by dividing 'Total_Usage_GB' by 'Subscription_Length_Months' so that we may have an attribute the shows us how much GB of data we are using per month (this feature is not present in the given dataset).



```
[14] df['avg_usage_per_month_GB'] = df['Total_Usage_GB'] / df['Subscription_Length_Months']
df['avg_usage_per_month_GB']

0         13.882353
1        172.000000
2         92.000000
```

2. Next feature that I have derived is 'high_usage' which is calculated by 'usage_threshold' that consists of 'avg_usage_per_month_GB' and taking its quartile(0.75) and by using lambda function for obtaining binary output. This means

that if the user used more than 75% (exceeds the threshold) of its 'avg_usage_per_month_GB' then it will fall under 'high_usage'.

```
[15] usage_threshold = df['avg_usage_per_month_GB'].quantile(0.75)
     df['high_usage'] = df['avg_usage_per_month_GB'].apply(lambda x: 1 if x > usage_threshold else 0)
     df['high_usage']

     0       0
     1       1
     2       1
```

3. I have also calculated 'user_ratio' as to how much internet (in GB) a single user is using on average when compared to others. The feature "user_ratio' is obtained by dividing 'avg_usage_per_month_GB' by its own mean.

```
[16] avg_monthly_usage = df['avg_usage_per_month_GB'].mean()
     df['usage_ratio'] = df['avg_usage_per_month_GB'] / avg_monthly_usage
     df['usage_ratio']

     0        0.320241
     1        3.967734
     2        2.122276
```

4. Finally, we have the feature of 'avg_bill_per_location' where we are mapping 'average_bill_per_location' to the mean of 'Location_encoded' and 'Monthly_Bill'.

```
[17] average_bill_per_location = df.groupby('Location_encoded')['Monthly_Bill'].mean()
     df['avg_bill_per_location'] = df['Location_encoded'].map(average_bill_per_location)
```

5. We finally print all the datatypes to find that there newly made features are all of 'float64' type.

```
[19] print(df.dtypes)

     CustomerID                    int64
     Age                           int64
     Subscription_Length_Months    int64
     Monthly_Bill                float64
     Total_Usage_GB                int64
     Churn                         int64
     Name_encoded                  int64
     Gender_encoded                int64
     Location_encoded              int64
     avg_usage_per_month_GB      float64
     high_usage                    int64
     usage_ratio                 float64
     avg_bill_per_location       float64
     dtype: object
```

Stage – 3: Model Building

```
[20] X = df.drop('Churn', axis=1)
     y = df['Churn']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Here, we first defined X and y where we have dropped 'Churn' from X and assigned it into y. We have also split our dataset into training dataset and testing dataset by the use of 'train_test_split' function where test_size=0.2 and random_state=4.

Different algorithms were used to train the Machine Learning model. Overview of them are given below –

Model – 1: Neural Networks

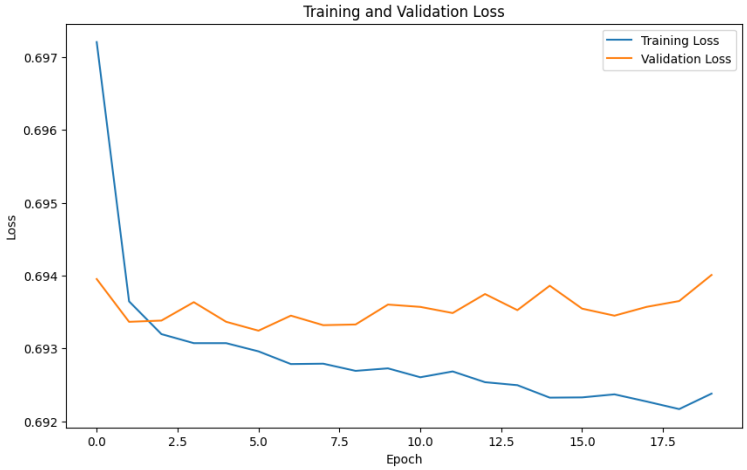
1. I have imported Keras and StandardScaler here.
2. The values for 'Dense' are given as 64 and 32 with the Non-Linear Activation Functions of 'relu' and 'sigmoid'.

```
model = keras.Sequential([
    layers.Input(shape=(X_train.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(32, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(1, activation='sigmoid')
])
```

3. I also have tried the values as 128 and 64, still there is no significant change in the result.
4. I kept changing the number of neurons to see if there is any point where I might get better results.
5. The value of epochs is 20 (and I have also tried 80 and100).
6. By this algorithm, we got Test Loss = 0.6939 and Test Accuracy = 0.5002

```
Epoch 19/20
1000/1000 [==============================] - 3s 3ms/step - loss: 0.6924 - accuracy: 0.5102 -
Epoch 20/20
1000/1000 [==============================] - 3s 3ms/step - loss: 0.6924 - accuracy: 0.5128 -
625/625 [==============================] - 2s 2ms/step - loss: 0.6939 - accuracy: 0.5002
Test Loss: 0.6939, Test Accuracy: 0.5002
```

7. The plot of Training Loss and Validation Loss upon Epoch and Loss was generated as below



8. We can see that Validation Loss is being between 0.693 and 0.694 only, so there was also no point in applying 'Early stopping' here.
9. The values of Test Loss and Test Accuracy is not satisfactory as the model is undergoing over-fitting.

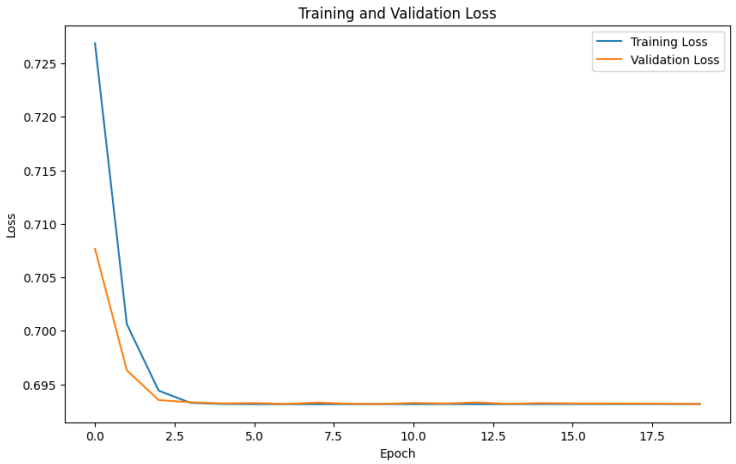
10. Over-fitting will give us good metrices only on trained dataset but will perform poorly on new dataset, so I have tried to model the same with the help of Regularizer.

Model – 2: Neural Networks with Regularizer

1. L1 and L2 Regularizers penalizes large weights and encourages the model to use smaller weights. I have also imported Keras and StandardScaler here.
2. The difference here when compared with the previous approach is that I have added "kernel_regularizer" here within the dense layers.
3. The values of dense and rest of the code remains the same.
4. Upon execution, we have opted Test Loss: 0.6931 and Test Accuracy: 0.5039 which is slightly better than the previous model.

```
Epoch 19/20
1000/1000 [==============================] - 4s 4ms/step - loss: 0.6932 - accuracy: 0.5029 -
Epoch 20/20
1000/1000 [==============================] - 4s 4ms/step - loss: 0.6931 - accuracy: 0.5028 -
625/625 [==============================] - 2s 2ms/step - loss: 0.6931 - accuracy: 0.5039
Test Loss: 0.6931, Test Accuracy: 0.5039
```

5. With Regularizer, we can see that there is a slight increase in the Accuracy value.
6. The plot of Training Loss and Validation Loss upon Epoch and Loss was generated as below



Model – 3: Logistic Regression

1. I have imported Logistic Regression from scikit-learn.
2. We have then fitted 'X_train' and 'y_train' into the Logistic Regression model.
3. We have used StandardScaler for transforming and fitting 'X_train' and 'X_test'.
4. We have then used classifier for classification and y_pred for making predictions.
5. Finally, I got the Accuracy = 0.5002 and Recall = 0.3950

```
[30] from sklearn.metrics import recall_score
     recall = recall_score(y_test, y_pred)
     print('Recall: ', recall)

     Recall:  0.3950206632395928

[31] from sklearn.metrics import accuracy_score
     print ("Accuracy : ", accuracy_score(y_test, y_pred))

     Accuracy :  0.5002
```

6. We can notice that Logistic Regression Model has got the Accuracy lesser than "Neural Networks with Regularizer" and same as "Neural Networks (Model – 1)".

Note: I have also tried other algorithms like Support Vector Machine, K-Means, KNN, and also these algorithms with and without "Standard Scaler", and also with and without "One Hot Encoder" but even they do not have any significant difference in their results.

Model – 4: Random Forest Classifier

1. We have imported RandomForestClassifier and different sk.learn metrices alongside StandardScaler for transforming and fitting the data.
2. 'y_pred' is used to make predictions of the model based on 'X_test'.
3. Finally, we got the Accuracy = 0.4979 and Recall = 0.4652

```
[35] accuracy = accuracy_score(y_test, y_pred)
     classification_rep = classification_report(y_test, y_pred)
     print("Accuracy:", accuracy)

     Accuracy: 0.4979

[36] from sklearn.metrics import recall_score
     recall = recall_score(y_test, y_pred)
     print('Recall: ', recall)

     Recall:  0.4652756778550549
```

4. We can notice that Random Forest Classifier has got Accuracy that is lesser than Logistic Regression.

Model – 5: XGBoost (Gradient Boosting Algorithm)

1. We have imported XGBClassifier along with StandardScaler and various sk.learn metrices.
2. This model was made to fit on 'X_train' and 'y_train'.
3. 'y_pred' was to be make predictions of the model upon 'X_test'.
4. We have got the value of Accuracy = 0.4968 and Recall = 0.47989

```
[50] # Print evaluation metrics
     print("Accuracy:", accuracy)

     Accuracy: 0.4968

[75] from sklearn.metrics import recall_score
     recall = recall_score(y_test, y_pred)
     print('Recall: ', recall)

     Recall:  0.47989114000604777
```

5. The Accuracy obtained by this algorithm can be seen to be even lesser than the Random Forest Classifier.

CONCLUSION – The over-all Accuracy among all the above-mentioned models is lying somewhere very near to 0.5, hence I can conclude from that the Accuracy that I can obtain from this dataset can be somewhere around 0.5 only.

Result Interpretation (Model Comparison) –

Model name	Accuracy score	Recall score
Neural Networks	0.5002	0.6939 (Test Loss)
Neural Networks with Regularizer	0.5039	0.6931 (Test Loss)
Logistic Regression	0.5002	0.3950
Random Forest Classifier	0.4979	0.4652
XGBoost (Gradient Boosting Algorithm)	0.4968	0.47989

Stage – 4: Model Optimization

1. I have tried to use GridSearchCV alongside Logistic Regression as this was giving better accuracy when compared with others.
2. Hyperparameter tuning is done here to achieve better performance and generalization on unseen data.

```
[96] model = LogisticRegression(random_state=42, max_iter=1000)
     grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
     grid_search.fit(X_train_scaled, y_train)
     best_model = grid_search.best_estimator_
     y_pred = best_model.predict(X_test_scaled)
```

3. Still there was no much difference, and the accuracy remains unsatisfied.

```
[89] from sklearn.metrics import recall_score
     recall = recall_score(y_test, y_pred)
     print('Recall: ', recall)

     Recall:  0.3950206632395928

[90] accuracy = accuracy_score(y_test, y_pred)
     print("Accuracy:", accuracy)

     Accuracy: 0.5002
```

4. Next, I had decided to use Decision Tree Classifier as it has a top-down approach.
5. It uses the values in each feature to split the dataset to a point where all data points that have the same class are grouped together.
6. From this algorithm, I got Accuracy = 0.505 and Recall = 0.5345

```
[95] print(f'Accuracy: {round(accuracy_score(y_test, y_test_preds),3)}')
     print(f'Recall score: {round(recall_score(y_test, y_test_preds),4)}')

     Accuracy: 0.505
     Recall score: 0.5345
```

Here, we can say that the model was optimized which in-turn has increased the Accuracy as well as the Recall Score.

Stage – 5: Model Development

1. In this stage, I have tried to take real-time inputs from the user.
2. The user shall input the values to the attributes that are used in this dataset and the model trained shall try to predict the value of 'Churn' to be 0 or 1.
3. We have used Random Forest Classifier here along with StandardScaler.
4. The inputs were given in the following way

```
    'usage_ratio': float(input("Enter Usage Ratio: ")),
    'avg_bill_per_location': float(input("Enter Average Bill per Location: "))
}
```

```
Enter Random CustomerID: 7860
Enter Age: 22
Enter Subscription Length (Months): 14
Enter Monthly Bill: 67
Enter Total Usage (GB): 145
Enter Name Encoded: 44445
Enter Gender Encoded (1 for male, 0 for female): 0
Enter Location Encoded: 2
Enter High Usage (0 for not high usage, 1 for high usage): 1
Enter Average Usage per Month (GB): 107
Enter Usage Ratio: 2.122276
Enter Average Bill per Location: 65.174980
```

5. The result was shown in this way

```
if churn_prediction[0] == 1:
    print("Churn: Yes")
else:
    print("Churn: No")
```

```
Churn: Yes
/usr/local/lib/python3.10/dist-package
  warnings.warn(
```