Model 2 - Credit Card Fraud detection

The dataset used in this model is contributed by "Machine Learning Group - ULB" on kaggle names as "Credit Card Fraud Detection". it is important that the credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they didnot purchase. The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occured in two days, where we have 492 frauds out of 284,807 transactions. The data is highly unbalanced, the positive class(frauds) account for 0.172% of all transactions. It contains only numerical input variables which are the result of a PCA transfromation. Due to confidentiality issues, the dataset doesn't contain original features and more background information about the data. Features V1, V2, ...V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are "Time" and "Amount". Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature "Amount" is the transaction amount, this feature can be used for example-dependent cost-sensitive learning. Feature "Class" is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Data preprocessing

Steps include in data preprocessing are:

1. Download dataset.
2. Create training, validation and test sets.
3. Create inputs and targets.
4. Identify numeric and categorical columns.
5. Imputer missing numerical columns.
6. Scale numeric features.

7. One-hot encode categorical features.
8. Save processed data to disk.
9. Load processed data from disk.

Pass the link of kaggle dataset in the data_dir variable and download it using opendatasets package.

```
data_dir2 = "https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud"
```

Download the dataset using opendatasets download function.

```
od.download(data_dir2)
```

```
Skipping, found downloaded files in "./creditcardfraud" (use force=True to force download)
```

```
os.listdir("./creditcardfraud")
```

```
['creditcard.csv']
```

Read the csv file from the directory and convert it into dataframe using pandas.

```
card_df = pd.read_csv("./creditcardfraud/creditcard.csv")
```

Let's check the shape of the dataframe named card_df.

```
card_df.shape
```

```
(284807, 31)
```

So,the dataframe contains 284807 rows and 31 columns.

Let's analyze the top 10 rows of the dataframe.

```
card_df.head(10)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0

10 rows × 31 columns

So, the dataframe contains 31 columns named Time, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20, V21, V22, V23, V24, V25, V26, V27, V28, Amount and Class column all are of float type except Class column which is of int type containing some float values.

```
card_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
```

```
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

After watching the dataset it seems like that the dataframe doesn't contain any empty fields.

Let's create training, validation and test datasets.Training set is used to train model i.e., compute the loss and adjust the models weights using an optimization technique. Validation set is used to evaluate the model during training, it tests the accuracy of the model and changes the model to provide accuracy while building the model. Test set is used to compare different models or approaches and report the model's final accuracy. The dataset is splitted in 80:20 ratio in which 80 percent dataset is training set and 20% is test set. And then training set is splitted in 75:25 ration in which 75% set is going to train and 25% set is going to use for validation.

```
train_val_df1, test_df1 = train_test_split(card_df, test_size=0.25, random_state=42)
train_df1, val_df1 = train_test_split(train_val_df1, test_size=0.25, random_state=42)
```

Let's create inputs and targets.

```
inputs1 = list(card_df.columns)[:-1]
target1 = card_df.columns[-1]
```

Copy the dataset to another dataframe to avoid making changes in the original dataset.

```
train_inputs1=train_df1[inputs1].copy()
train_target1 = train_df1[target1].copy()
```

```
val_inputs1 = val_df1[inputs1].copy()
val_target1 = val_df1[target1].copy()
```

```
test_inputs1 = test_df1[inputs1].copy()
test_target1 = test_df1[target1].copy()
```

Identify numeric and categorical columns. Numeric columns are of numeric type and categorical columns are of object type.

```
numeric_cols1 = train_inputs1.select_dtypes(include = np.number).columns.tolist()
categorical_cols1 = train_inputs1.select_dtypes('object').columns.tolist()
```

Run Imputer on numeric columns to fill missing values. Basic technique of replacing missing values with the average value in the column using SimpleImputer class from sklearn.impute . Here, we have used the mean strategy to fill missing values and fit it in the dataset. The missing values can be filled using the transform method on them in training set, validation set and test set.

```
imputer = SimpleImputer(strategy="mean").fit(card_df[numeric_cols1])
```

```
train_inputs1[numeric_cols1] = imputer.transform(train_inputs1[numeric_cols1])
val_inputs1[numeric_cols1] = imputer.transform(val_inputs1[numeric_cols1])
test_inputs1[numeric_cols1] = imputer.transform(test_inputs1[numeric_cols1])
```

Let's check the statistics of the columns.

```
list(imputer.statistics_)
```

```
[94813.85957508067,
 1.1683749838001528e-15,
 3.416908049651284e-16,
 -1.379536707896593e-15,
 2.0740951198584196e-15,
 9.604066317127324e-16,
 1.4873130132010145e-15,
 -5.556467295694611e-16,
 1.2134813634275587e-16,
 -2.4063305498905906e-15,
 2.2390527426993533e-15,
 1.673326932726423e-15,
 -1.2470117695222676e-15,
 8.190001274383203e-16,
 1.2072942051600827e-15,
 4.887455859804944e-15,
 1.4377159541859243e-15,
 -3.7721706856547467e-16,
 9.564149167014576e-16,
 1.0399166050935636e-15,
 6.406203628719748e-16,
 1.654066907797022e-16,
 -3.568593220079729e-16,
 2.5786478972835623e-16,
 4.473265530947536e-15,
 5.340914685085768e-16,
 1.6834371984034178e-15,
 -3.6600908126037946e-16,
 -1.2273899954199695e-16,
 88.34961925093133]
```

Let's scale the features of the dataset in a specific range using each feature's minimum and maximum value. MinMaxScaler method scale the range between 0 and 1. Unlike standard scaling, where data are scaled based on the standard normal distribution(with mean = 0 and standard deviation = 1), the min-max scaler uses each column's minimum and maximum value to scale the data series.

```
scaler = MinMaxScaler().fit(card_df[inputs1])
```

Transform the dataframe into the scaler.

```
train_inputs1[numeric_cols1] = scaler.transform(train_inputs1[numeric_cols1])
val_inputs1[numeric_cols1] = scaler.transform(val_inputs1[numeric_cols1])
test_inputs1[numeric_cols1] = scaler.transform(test_inputs1[numeric_cols1])
```

```
train_inputs1.describe().loc[['min', 'max']]
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26
min	0.000006	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.003924	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
max	0.999977	1.0	1.0	0.909982	1.0	1.0	1.0	1.0	1.0	0.820797	...	1.0	1.0	1.0	1.0	1.0	1.0	1.0

2 rows × 30 columns

As we can see, the min and max values of all the columns are scaled between 1s and 0s.

To use categorical columns, we simply need to convert them to numbers. Here, we have categorical columns with more than 2 categories. So we can perform one-hot encoding i.e., create a new column for each category with 1s and 0s.

```
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore').fit(card_df[categorical_
```

```
encoded_cols = list(encoder.get_feature_names_out(categorical_cols1))
```

```
train_inputs1[encoded_cols] = encoder.transform(train_inputs[categorical_cols1])
val_inputs1[encoded_cols] = encoder.transform(val_input[categorical_cols1])
test_inputs1[encoded_cols] = encoder.transform(test_input[categorical_cols1])
```

```
val_inputs1
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
187926	0.739328	0.988063	0.761380	0.804677	0.276348	0.767367	0.255660	0.268833	0.782429	0.469066
25579	0.194876	0.944556	0.780025	0.855943	0.311913	0.764968	0.262843	0.266819	0.791198	0.447299
200728	0.772906	0.947050	0.778190	0.803585	0.255984	0.784734	0.300322	0.269994	0.798475	0.422364
94458	0.375573	0.975323	0.764091	0.848400	0.285922	0.764913	0.275533	0.261948	0.791309	0.489346
264751	0.935107	0.993112	0.768150	0.806544	0.263042	0.769482	0.258654	0.265708	0.784752	0.472831
...	...	...	...	...	...	...	...	...	...	...
32462	0.213048	0.976175	0.767997	0.844513	0.295816	0.764378	0.262113	0.265249	0.786392	0.456277
279438	0.977279	0.992753	0.763002	0.823677	0.253537	0.766493	0.270700	0.260987	0.787419	0.501159
34568	0.218343	0.902221	0.732768	0.867593	0.286094	0.788923	0.230000	0.254593	0.788638	0.438452
37902	0.226637	0.932756	0.772409	0.853528	0.245155	0.769904	0.265456	0.268575	0.786613	0.469391
232363	0.851550	0.994025	0.758998	0.809714	0.199720	0.768267	0.268979	0.261999	0.786084	0.439084

53402 rows × 30 columns

You can see that there is no effect on dataframe columns after performing the one-hot encoder because there are no categorical columns exist. Therefore it is unaffected by one-hot encode.

Let's save the processed data to disk. To save data, we will use parquet file format.

```
train_inputs1.to_parquet('train_inputs_model2.parquet')
```

```
pd.DataFrame(train_target1).to_parquet('train_target_model2.parquet')
```

Reload the saved files.

```
train_inputs1 = pd.read_parquet('train_inputs_model2.parquet')
```

```
train_target = pd.read_parquet('train_target_model2.parquet')
```

Model Training and Evaluation

Steps include in model training and evaluation :

1. Select the columns to be used for training and prediction.
2. Create and train the model.
3. Generate predictions and probabilities
4. Helper function to predict, compute accuracy and plot confusion matrix.
5. Evaluate on validation and test set.
6. Save the trained model and load it back.

We are training our model using Decision Tree Classifier method and fit it on training inputs and targets. A decision tree is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represent the outcome.A decision tree has essentially two nodes; decision node and leaf node. It uses an iterative approach, we take all possible splits take all possible splits and calculate gini index and decide splits accordingly.To best split gini index must be 0. And then we apply an loss function to calculate the loss of the model.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree, export_text
```

We will use decision tree classifier model with two hyperparameters random_state=42 and criterion='entropy'. random_state hyperparameter controls the randomness of the estimator. The features are randomly permuted at each split, even if splitter is set to "best", the value 42 returns the same permuted vales every time. And the criterion hyperparameter is the function to measure the quality of a split, the value entropy is for Shannon information gain.

```
model = DecisionTreeClassifier(random_state=42, criterion='entropy')
```

```
model.fit(train_inputs1[numeric_cols1], train_target1)
```

Make prediction from the model using predict function on training dataset.

```
pred_train=model.predict(train_inputs1)
```

```
pd.value_counts(pred_train)
```

```
0    159920
1       283
dtype: int64
```

It returns the count of predictions which made by the model. The prediction count of 0's is 159920 and the prediction count of 1's is 283. It seems like the model is predicting 0 more than 1.

```
pred_prob_train = model.predict_proba(train_inputs1)
pred_prob_train
```

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

predict_proba gives the predicted output along with their probabilities.

Let's check the accuracy on training, validation and test dataset.

```
score = accuracy_score(train_target1, pred_train)
score
```

```
1.0
```

The accuracy score of the model in the train inputs dataset if 1.0 which is 100% and it's wonderful. It means that our model is learning well from the dataset.

```
model.score(val_inputs1, val_target1)
```

```
0.9993820456162691
```

It will make predictions on the validation inputs and then compare them with the targets and displays the probability that how well our model is predicting and the accuracy of prediction of model on validation input set is about 99.9% and it is pretty good in comparison of training input dataset.

```
model.score(test_inputs1, test_target1)
```

```
0.9990309260975815
```

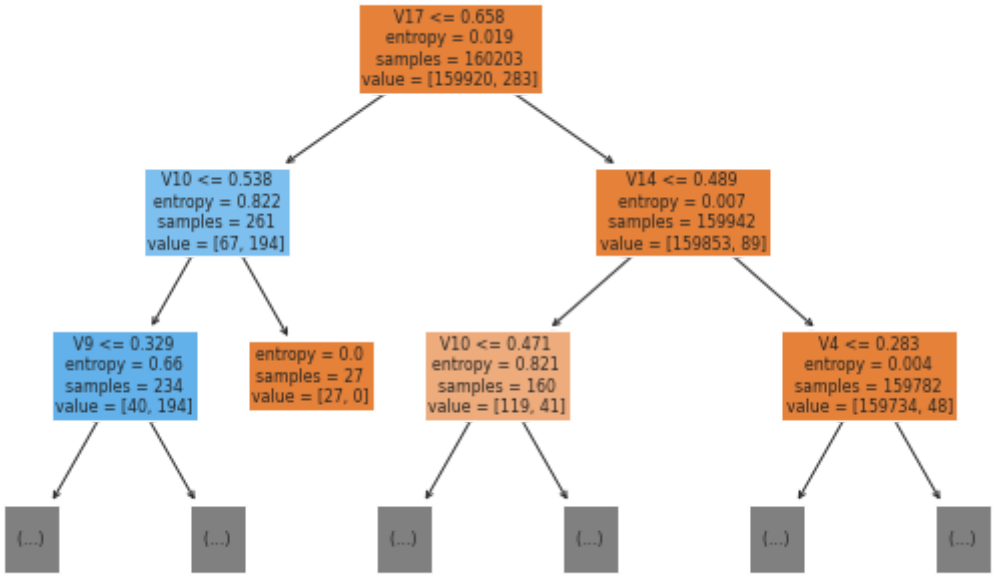
The test input dataset also giving 99.9% accuracy which is not more differentiable from the training and validation dataset. It means that our performed well on all three datasets and it learned very well.

Let's visualize the model tree to see the feature importance and how the data is splitted.

```
plot_tree(model, feature_names=train_inputs1.columns, max_depth=2, filled=True)
```

```
[Text(255.75, 285.39000000000004, 'V17 <= 0.658\nentropy = 0.019\nsamples =
160203\nvalue = [159920, 283]'),
```

```
 Text(139.5, 203.85000000000002, 'V10 <= 0.538\nentropy = 0.822\nsamples = 261\nvalue =
[67, 194]'),
 Text(93.0, 122.31, 'V9 <= 0.329\nentropy = 0.66\nsamples = 234\nvalue = [40, 194]'),
 Text(46.5, 40.76999999999998, '\n  (...)  \n'),
 Text(139.5, 40.76999999999998, '\n  (...)  \n'),
 Text(186.0, 122.31, 'entropy = 0.0\nsamples = 27\nvalue = [27, 0]'),
 Text(372.0, 203.85000000000002, 'V14 <= 0.489\nentropy = 0.007\nsamples =
159942\nvalue = [159853, 89]'),
 Text(279.0, 122.31, 'V10 <= 0.471\nentropy = 0.821\nsamples = 160\nvalue = [119,
41]'),
 Text(232.5, 40.76999999999998, '\n  (...)  \n'),
 Text(325.5, 40.76999999999998, '\n  (...)  \n'),
 Text(465.0, 122.31, 'V4 <= 0.283\nentropy = 0.004\nsamples = 159782\nvalue = [159734,
48]'),
 Text(418.5, 40.76999999999998, '\n  (...)  \n'),
 Text(511.5, 40.76999999999998, '\n  (...)  \n')]
```



You can analyze from the tree that the V17 feature is the most important amongst all, it has entropy value of 0.019. After that V10 and V14 features are given importance with entropy value 0.0822 and 0.007 respectively. It goes deep until we got the output but here we visualized only three levels of the tree to analyze it better.

Let's check the classes of the model.

```
model.classes_
```

```
array([0, 1])
```

Model contains two classes 0 and 1. First class is 0 and second class is 1.

Let's check the importance of the features given by model to them.

```
model.feature_importances_
```

```
array([0.01484227, 0.00732797, 0.00301423, 0.00255991, 0.02805082,
```

```
       0.02163403, 0.00228452, 0.0160368 , 0.00518668, 0.00911806,
       0.05240211, 0.0071663 , 0.01133216, 0.01877821, 0.12212851,
       0.00564893, 0.00925369, 0.58237515, 0.00482718, 0.01203113,
       0.01336296, 0.01615171, 0.00557921, 0.        , 0.00401464,
       0.00138245, 0.00932871, 0.00714819, 0.00066757, 0.00636588])
```

According to the feature importance, the highest importance is given to V17 feature about 58.2%, followed by V10 and V14.

Let's calculate the loss of the model using root mean squared error loss .

```
rmse=np.sqrt(mean_squared_error(train_target1, pred_train))
rmse
```

```
0.0
```

The loss of the model is 0.0 which is very good.

Let's plot the confusion matrix of the training, validation and test dataset.

```
from sklearn.metrics import plot_confusion_matrix
```

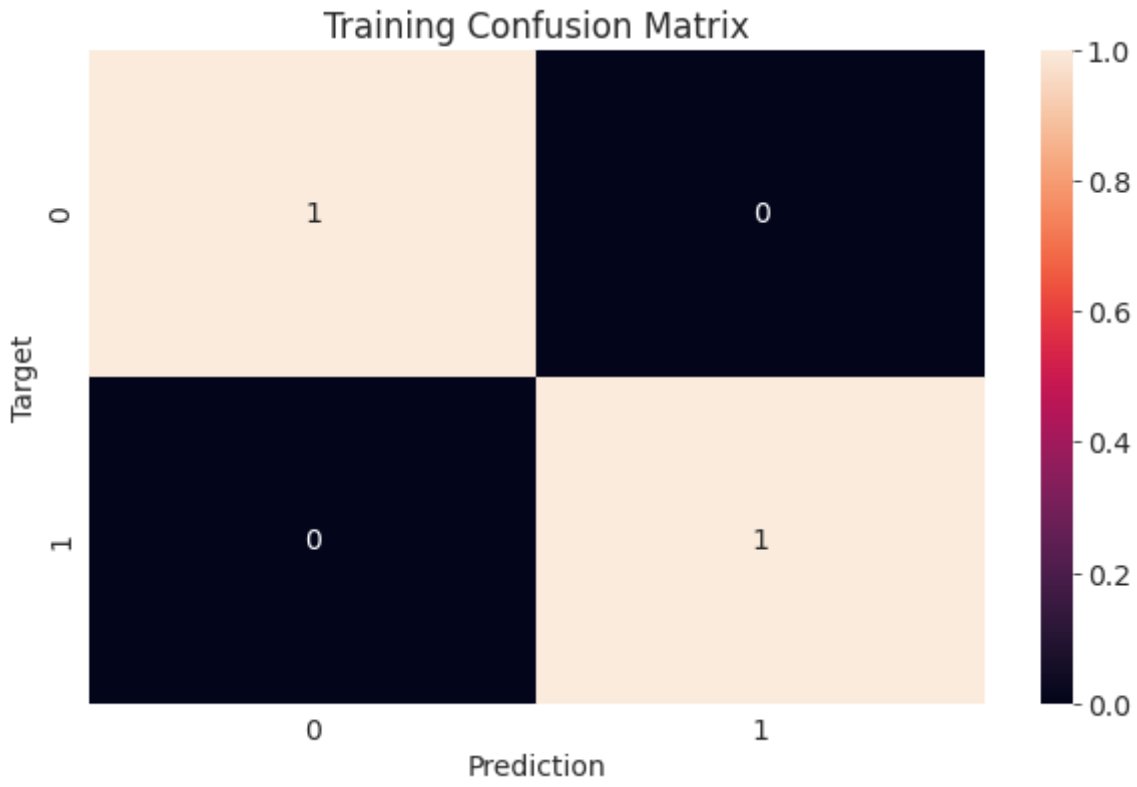
```
confusion_matrix(train_target1, pred_train, normalize='true')
```

```
array([[1., 0.],
       [0., 1.]])
```

```
def predict_and_plot(inputs, targets, name=''):
    preds = model.predict(inputs)
    accuracy = accuracy_score(targets, preds)
    print("Accuracy: {:.2f}%".format(accuracy * 100))
    cf = confusion_matrix(targets, preds, normalize='true')
    plt.figure()
    sns.heatmap(cf, annot=True)
    plt.xlabel('Prediction')
    plt.ylabel('Target')
    plt.title('{} Confusion Matrix'.format(name));
    return preds
```

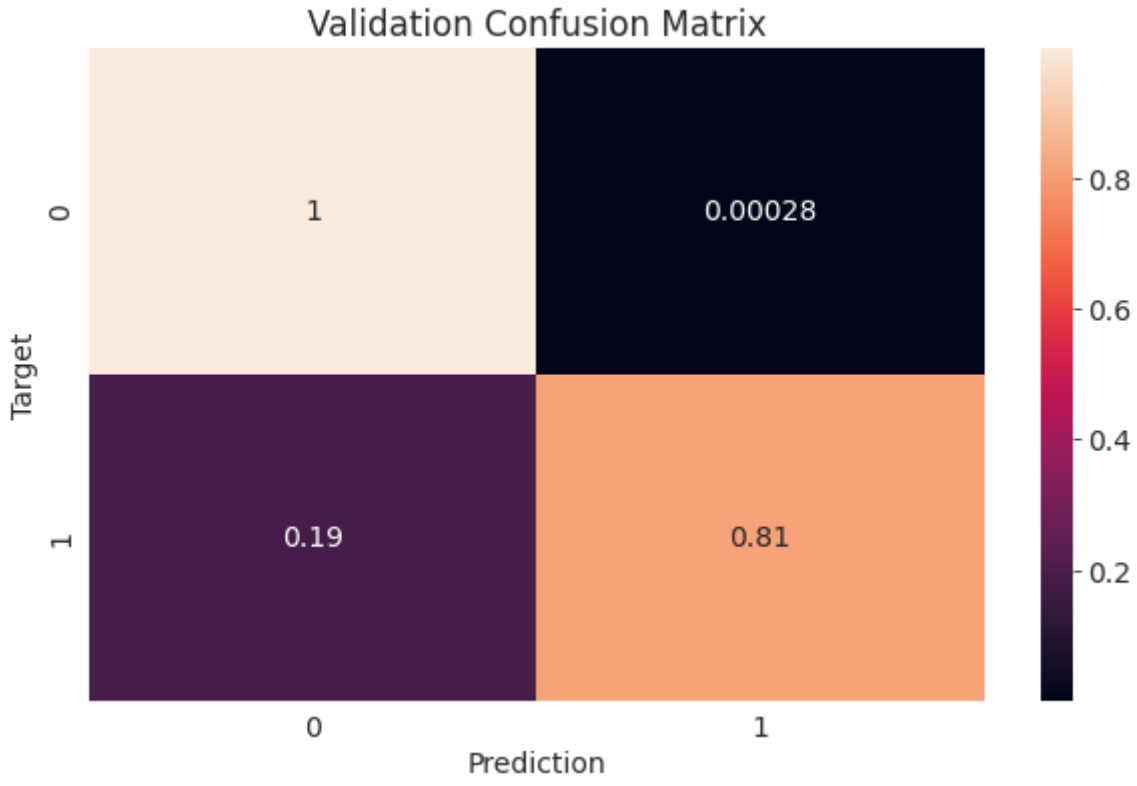
```
pred = predict_and_plot(train_inputs1, train_target1, 'Training')
```

```
Accuracy: 100.00%
```



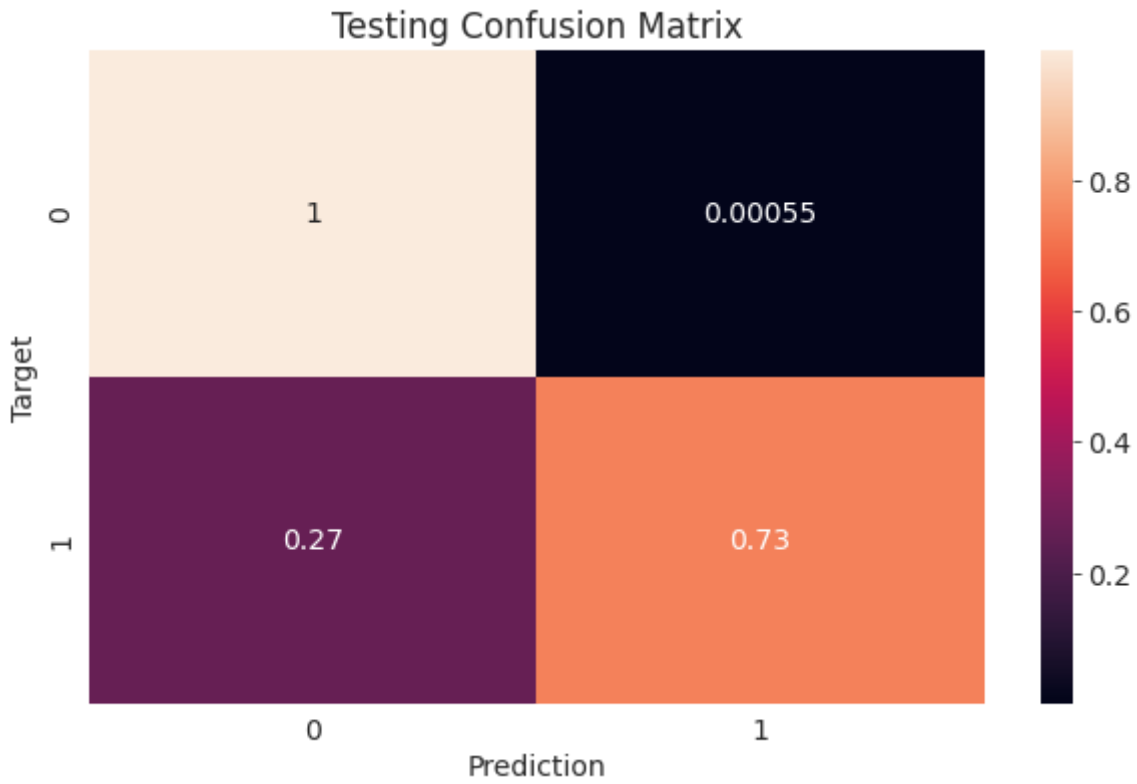

```
val_preds = predict_and_plot(val_inputs1, val_target1, "Validation")
```

```
Accuracy: 99.94%
```




```
test_preds = predict_and_plot(test_inputs1, test_target1, "Testing")
```

```
Accuracy: 99.90%
```



Prediction on single input

Let's define a helper function to predict output on single input.

```
def predict_single(single_input):
    input_df = pd.DataFrame([single_input])
    input_df[numeric_cols1] = imputer.transform(input_df[numeric_cols1])
    input_df[numeric_cols1] = scaler.transform(input_df[numeric_cols1])
    preds = model.predict(input_df)[0]
    preds_proba = model.predict_proba(input_df)[0][list(model.classes_).index(preds)]
    return preds, preds_proba
```

```
new_input = {'Time':0.640000,
             'V1':1.658354,
             'V2':-1.860163,
             'V3':1.373209,
             'V4':0.979780,
             'V5':-0.403198,
             'V6':1.200499,
             'V7':0.491461,
             'V8':0.647676,
             'V9':-1.714654,
             'V10':0.907643,
             'V11':0.324501,
             'V12':0.966084,
             'V13':0.517293,
             'V14':-0.465946,
             'V15':2.545865,
             'V16':-2.690083,
             'V17':1.609969,
```

```
            'V18':-0.421359,
            'V19':-2.661857,
            'V20':0.824980,
            'V21':0.347998,
            'V22':0.871679,
            'V23':0.209412,
            'V24':-0.989281,
            'V25':-0.727642,
            'V26':-0.639097,
            'V27':-0.855353,
            'V28':-0.659752,
            'Amount':358.660000}
```

```
predicts, predict_prob = predict_single(new_input)
predicts, predict_prob
```

```
(0, 1.0)
```

Our model predicted output 0 with the 100% probability for the new input which is unknown for the model.

```
jovian.commit()
```

```
[jovian] Updating notebook "tannu945/personal-key-indicators-of-heart-disease" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/tannu945/personal-key-indicators-of-heart-disease
```

```
'https://jovian.ai/tannu945/personal-key-indicators-of-heart-disease'
```

```
```