Model 1: personal-key-indicators-of-heart-disease

Use the "Run" button to execute the code.

This model contains the dataset of personal key indicators of heart disease which was based on 2020 annual CDC survey data of 400k adults related to their health issues.The dataset was contributed by Kamil Pytlak on Kaggle. According to CDC, about half of all americans have atleast 1 of 3 key risk factors for heart disease: high blood pressure, high cholestrol, and smoking. Other key indicators include diabetic status, obesity(high bmi), not getting enough physical activity or drinking too much alcohol. Originally, the dataset come from the CDC and is a major part of Behavioral Risk Factor Surveillance System(BRFSS), which conducts annual telephone surveys to gather data on the health status of U.S. residents. The most recent dataset(as of February 15, 2022) includes data from 2020. It consists of 401,958 rows and 279 columns. In this dataset, the selected variables are the most relevant variables which directly or indirectly influence heart disease. The original dataset of nearly 300 variables was reduced to just about 18 variables. In addition to classical EDA, this dataset can be used to apply a range of machine learning methods, most notably classifier models (logistic regression, SVM, random forest, etc.). You should treat the variable "HeartDisease" as a binary ("Yes" - respondent had heart disease; "No" - respondent had no heart disease). But note that classes are not balanced, so the classic model application approach is not advisable. Fixing the weights/undersampling should yield significantly betters results. Based on the dataset, I constructed a logistic regression model and embedded it in an application you might be inspired by: https://heart-condition-checker.herokuapp.com/.

About the dataset

The dataset contains 18 variables(9 boolenas, 5 strings and 4 decimals). In machine learning projects, "Heart Disease" can be used as the explonatory variable, but note that the classes are heavily unbalanced. Columns:

1. HeartDisease: Respondents that have ever reported having coronary heart disease(CHD) or myocardial infarction(MI).
2. BMI: Body Mass Index
3. Smoking: Have you smoked at least 100 cigarettes in your entire life?[Note: 5 packs = 100 cigarettes]
4. AlcoholDrinking: Heavy drinkers(adult men having more than 14 drinks per week and adult women having more than 7 drinks per week)
5. Stroke: Ever told you had a stroke?
6. PhysicalHealth: Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30
7. MentalHealth: Thinking about your mental health, for how many days during the past 30 days was your mental health not good?
8. DiffWalking: Do you have serious difficulty walking or climbing stairs?
9. Sex: Are you male or female?
10. AgeCategory: Fourteen-level age category
11. Race: Imputed race/ethnicity value
12. Diabetic: Ever told you had diabetes?
13. PhysicalActivity: Adults who reported doing physical activity or exercise during the past 30 days other than their regular job

14. GenHealth: Would you say that in general your health is...
15. SleepTime: On average, how many hours of sleep do you get in a 24-hour period?
16. Asthma: (Ever told) (you had) asthma?
17. KidneyDisease: Not including kidney stones, bladder infection or incontinence, were you ever told you had kidney disease?
18. SkinCancer: (Ever told) (you had) skin cancer?

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.

Let's start the model training by installing the required libraries first.

```
!pip install jovian --upgrade --quiet
```

```
!pip install sklearn matplotlib pandas --upgrade --quiet
```

```
pip install numpy opendatasets
```

```
Requirement already satisfied: numpy in /opt/conda/lib/python3.9/site-packages (1.20.3)
Collecting opendatasets
  Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
Requirement already satisfied: click in /opt/conda/lib/python3.9/site-packages (from
opendatasets) (8.0.1)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.9/site-packages (from
opendatasets) (4.62.3)
Collecting kaggle
  Downloading kaggle-1.5.12.tar.gz (58 kB)
     |████████████████████████████████| 58 kB 5.7 MB/s eta 0:00:011
Requirement already satisfied: six>=1.10 in /opt/conda/lib/python3.9/site-packages (fron
kaggle->opendatasets) (1.16.0)
Requirement already satisfied: certifi in /opt/conda/lib/python3.9/site-packages (from
kaggle->opendatasets) (2021.5.30)
Requirement already satisfied: python-dateutil in /opt/conda/lib/python3.9/site-packages
(from kaggle->opendatasets) (2.8.2)
Requirement already satisfied: requests in /opt/conda/lib/python3.9/site-packages (from
kaggle->opendatasets) (2.26.0)
Collecting python-slugify
  Downloading python_slugify-6.1.2-py2.py3-none-any.whl (9.4 kB)
Requirement already satisfied: urllib3 in /opt/conda/lib/python3.9/site-packages (from
kaggle->opendatasets) (1.26.7)
Collecting text-unidecode>=1.3
  Downloading text_unidecode-1.3-py2.py3-none-any.whl (78 kB)
     |████████████████████████████████| 78 kB 10.6 MB/s eta 0:00:01
Requirement already satisfied: charset-normalizer~=2.0.0 in /opt/conda/lib/python3.9/sit
packages (from requests->kaggle->opendatasets) (2.0.0)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.9/site-packages
(from requests->kaggle->opendatasets) (3.1)
Building wheels for collected packages: kaggle
  Building wheel for kaggle (setup.py) ... done
  Created wheel for kaggle: filename=kaggle-1.5.12-py3-none-any.whl size=73051
sha256=ab43c7c72dcc7843af7e11584af0b96a9ad3b124f174240c6d2f93a9bb3ac027
  Stored in directory:
/home/jovyan/.cache/pip/wheels/ac/b2/c3/fa4706d469b5879105991d1c8be9a3c2ef329ba9fe2ce508
Successfully built kaggle
Installing collected packages: text-unidecode, python-slugify, kaggle, opendatasets
Successfully installed kaggle-1.5.12 opendatasets-0.1.22 python-slugify-6.1.2 text-
unidecode-1.3
Note: you may need to restart the kernel to use updated packages.
```

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

```
data_dir = "https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-hear
```

```
import opendatasets as od
od.download(data_dir)
```

```
Please provide your Kaggle credentials to download this dataset. Learn more:
http://bit.ly/kaggle-creds
Your Kaggle username: tanishaagrawal945
Your Kaggle Key: ········
Downloading personal-key-indicators-of-heart-disease.zip to ./personal-key-indicators-
of-heart-disease

100%|██████████| 3.13M/3.13M [00:00<00:00, 82.8MB/s]
```

```
import os
os.listdir("./personal-key-indicators-of-heart-disease")
```

```
['heart_2020_cleaned.csv']
```

Read the csv file which you had just downloaded using pandas package and convert it into a dataframe.

```
import pandas as pd
heart_df = pd.read_csv('./personal-key-indicators-of-heart-disease/heart_2020_cleaned.c
```

Let's check the starting 10 rows of the dataset.

```
heart_df.head(10)
```

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCa
0	No	16.60	Yes	No	No	3.0	30.0	No	Female	
1	No	20.34	No	No	Yes	0.0	0.0	No	Female	80 c
2	No	26.58	Yes	No	No	20.0	30.0	No	Male	
3	No	24.21	No	No	No	0.0	0.0	No	Female	
4	No	23.71	No	No	No	28.0	0.0	Yes	Female	
5	Yes	28.87	Yes	No	No	6.0	0.0	Yes	Female	
6	No	21.63	No	No	No	15.0	0.0	No	Female	
7	No	31.64	Yes	No	No	5.0	0.0	Yes	Female	80 c
8	No	26.45	No	No	No	0.0	0.0	No	Female	80 c
9	No	40.69	No	No	No	0.0	0.0	Yes	Male	

Import sklearn package to implement to machine learning models.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
```

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_df, test_df = train_test_split(heart_df, test_size=0.2, random_state=42)
```

```
train_df, val_df = train_test_split(train_val_df, test_size=0.25, random_state=42)
```

Let's create inputs and targets.

```
inputs = list(train_df.columns)[1:]
target = 'HeartDisease'
```

Copy the set to avoid making changes in the original dataset.

```
train_inputs= train_df[inputs].copy()
train_target = train_df[target].copy()
```

```
val_input = val_df[inputs].copy()
val_target = val_df[target].copy()
```

```
test_input = test_df[inputs].copy()
test_target = test_df[target].copy()
```

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

```
numeric_cols = train_inputs.select_dtypes(include = np.number).columns.tolist()
categorical_cols = train_inputs.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(heart_df[numeric_cols])
```

```
train_inputs[numeric_cols]=imputer.transform(train_inputs[numeric_cols])
val_input[numeric_cols]=imputer.transform(val_input[numeric_cols])
test_input[numeric_cols]=imputer.transform(test_input[numeric_cols])
```

Check the statistics of the dataset.

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

```
[28.325398520927465, 3.3717100017198516, 3.898366140808956, 7.097074688472302]
```

```
train_inputs[numeric_cols].isna().sum()
```

```
BMI               0
PhysicalHealth    0
MentalHealth      0
```

```
SleepTime        0
dtype: int64
```

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(heart_df[numeric_cols])
```

```
train_inputs[numeric_cols]=scaler.transform(train_inputs[numeric_cols])
val_input[numeric_cols]=scaler.transform(val_input[numeric_cols])
test_input[numeric_cols]=scaler.transform(test_input[numeric_cols])
```

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

	BMI	PhysicalHealth	MentalHealth	SleepTime
min	0.000000	0.0	0.0	0.0
max	0.960159	1.0	1.0	1.0

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(heart_df[categorical
```

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

```
train_inputs[encoded_cols] = encoder.transform(train_inputs[categorical_cols])
val_input[encoded_cols] = encoder.transform(val_input[categorical_cols])
test_input[encoded_cols] = encoder.transform(test_input[categorical_cols])
```

```
val_input
```

	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategoı
167018	0.183025	No	No	No	0.066667	0.933333	No	Male	18-2
14318	0.201376	No	No	No	0.000000	0.000000	No	Male	60-6
228294	0.414343	Yes	No	No	0.000000	0.000000	Yes	Female	60-6
25428	0.170590	No	No	No	0.000000	0.000000	No	Female	18-2
12295	0.128577	No	No	No	0.000000	0.333333	No	Female	55-5
...	...	...	...	...	...	...	...	...	.
80175	0.144996	Yes	No	No	0.000000	0.000000	No	Female	50-5
36141	0.113968	No	No	No	0.000000	0.333333	No	Female	40-4

	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategoı
145255	0.195943	Yes	Yes	No	0.000000	0.066667	No	Male	25-2
243103	0.179041	No	No	No	0.000000	0.000000	Yes	Male	70-7
309703	0.176385	Yes	No	No	0.000000	0.000000	No	Male	25-2

63959 rows × 63 columns

One-hot encoder creates a new column for every category having values 0 and 1.

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

```
pip install pyarrow --quiet
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
train_inputs.to_parquet('train_inputs.parquet')
```

```
pd.DataFrame(train_target).to_parquet('train_target.parquet')
```

Load the processed data from disk.

```
train_inputs = pd.read_parquet('train_inputs.parquet')
```

```
train_target = pd.read_parquet('train_target.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 Logistic Regression method and fit it on training inputs and targets. In linear regression we take combination or weighted sum of th input features. Then we apply sigmoid function to the result to obtain a number between 0 and 1. This number represents the probability of the input being classified as "YES". And then we apply an loss function to calculate the loss of the model.

```
model = LogisticRegression(solver='liblinear')
```

```
model.fit(train_inputs[numeric_cols+encoded_cols], train_target)
```

```
/opt/conda/lib/python3.9/site-packages/sklearn/utils/validation.py:985:
DataConversionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
LogisticRegression(solver='liblinear')
```

Let's check the weights which are assigned to each feature.

```
model.coef_.tolist()
```

```
[[0.7011747046928634,
  0.09213406653132525,
  0.16709363456028245,
  -0.5378520857632133,
  -0.35124863852398214,
  0.004533268743356414,
  -0.04276680451945443,
  -0.3039485652649767,
  -0.6754922446941583,
  0.32877687490944285,
  -0.28894695968546424,
  -0.05776841009522303,
  -0.5395907745304378,
  0.19287540474933942,
  -1.5830843800121046,
  -1.3572942482314263,
  -1.2508078932696585,
  -1.0461175570538035,
  -0.6519118620658675,
  -0.33890877800024644,
  0.11057073780975754,
  0.30954837114443695,
  0.5872504792472761,
  0.8440596381755354,
  1.0980756743498885,
  1.349052912495495,
  1.5828515356320452,
  0.20382269246886311,
  -0.3796113327498083,
  -0.20419270547966265,
  -0.0704356212471385,
  0.03374175640495389,
  0.06995984081570991,
  -0.2959537632321075,
  -0.1262333652697976,
  0.19954426904706588,
  -0.12407251032819067,
  -0.19038619500339352,
  -0.15632917477454852,
  -1.063443434918191,
  0.47769694039612487,
  -0.008879198371225614,
  0.8239020762790331,
  -0.5759917531639223,
  -0.2980918597329353,
  -0.04862351005262984,
  -0.4660098855457512,
  0.11929451576165297,
  -0.23359926498160943,
  -0.11311610480218882]]
```

Let's check the bias.

```
model.intercept_
```

```
array([-0.34671537])
```

Copy the dataset to other variable.

```
x_train = train_inputs[numeric_cols+encoded_cols]
x_val = val_input[numeric_cols+encoded_cols]
x_test = test_input[numeric_cols+encoded_cols]
```

Generate predictions on training dataset.

```
train_preds = model.predict(x_train)
train_preds
```

```
array(['No', 'No', 'No', ..., 'No', 'No', 'No'], dtype=object)
```

The model predicted the outputs in the form of 'yes' or 'no'.

```
train_preds_proba = model.predict_proba(x_train)
train_preds_proba
```

```
array([[0.66768608, 0.33231392],
       [0.95145953, 0.04854047],
       [0.97706405, 0.02293595],
       ...,
       [0.89069249, 0.10930751],
       [0.99758721, 0.00241279],
       [0.99652458, 0.00347542]])
```

This calculated the probabilities of predictions to be true.

```
model.classes_
```

```
array(['No', 'Yes'], dtype=object)
```

Let's plot the confusion matrix, compute accuracy and calculate loss of the model.

```
from sklearn.metrics import accuracy_score, confusion_matrix, mean_squared_error
```

```
accuracy_score(train_target, train_preds)
```

```
0.9168477722707776
```

Our model is giving 91.6% accuracy and which is quiet good. That means our model is predicting correct outputs for the 91% of the data and might give false outputs for the rest 9% data.

Plot confusion matrix of the accuracy.

```
confusion_matrix(train_target, train_preds, normalize='true')
```

```
array([[0.99181009, 0.00818991],
       [0.89088678, 0.10911322]])
```

Let's create helper function to plot the confusion matrix on train, validation and test sets and compare the accuracies between them.

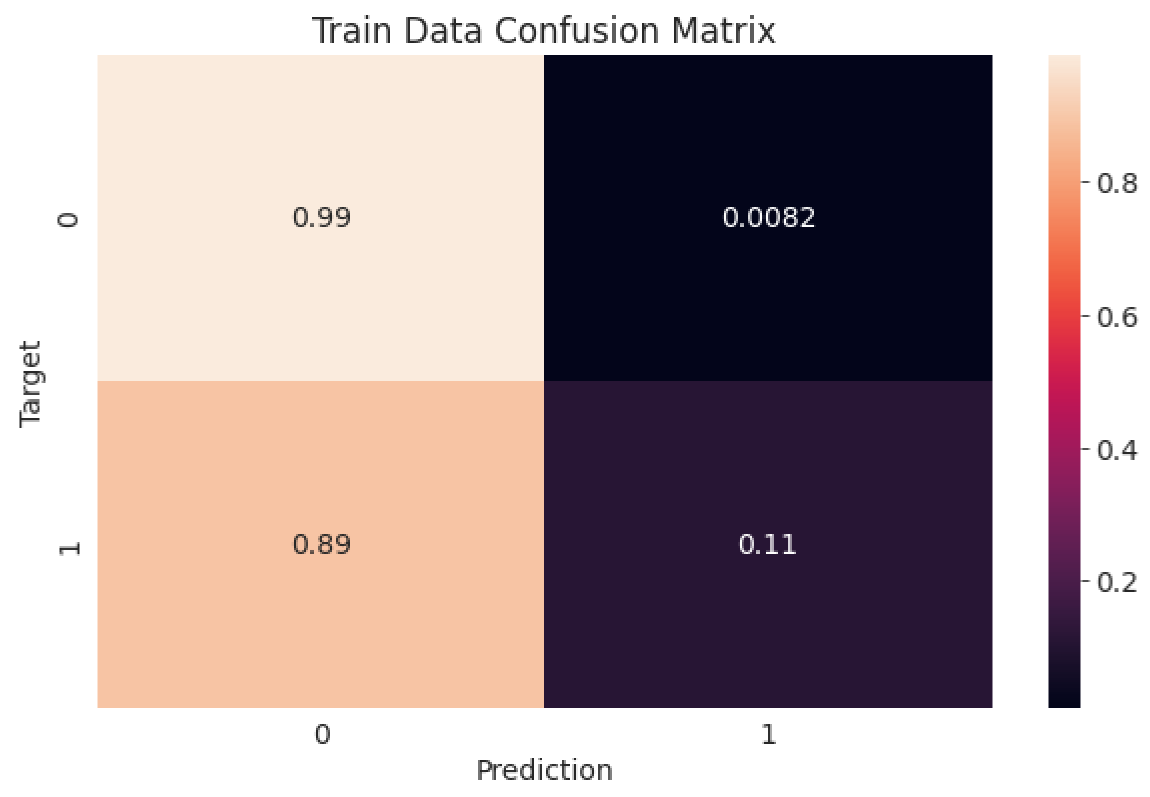
```
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set_style('darkgrid')
matplotlib.rcParams['font.size'] = 14
matplotlib.rcParams['figure.figsize'] = (10, 6)
matplotlib.rcParams['figure.facecolor'] = '#00000000'
```

```
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
```

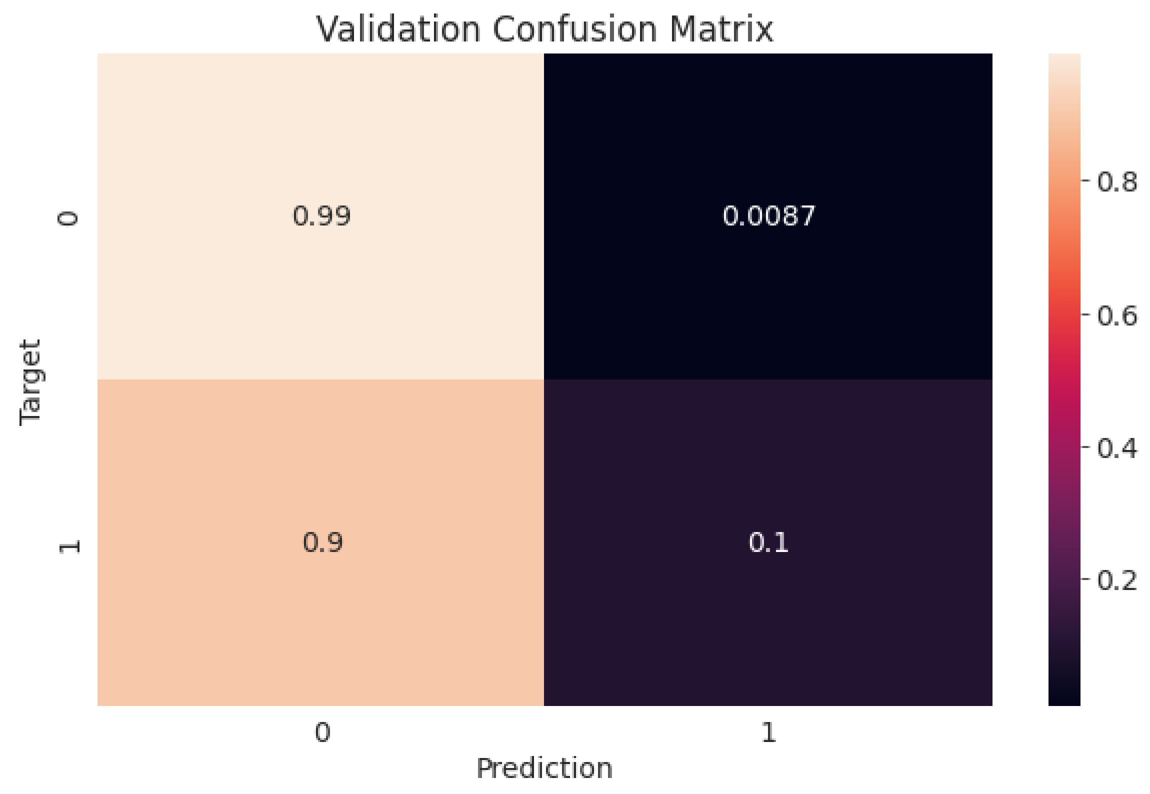
```
preds = predict_and_plot(x_train, train_target,'Train Data')
```

```
Accuracy: 91.68%
```



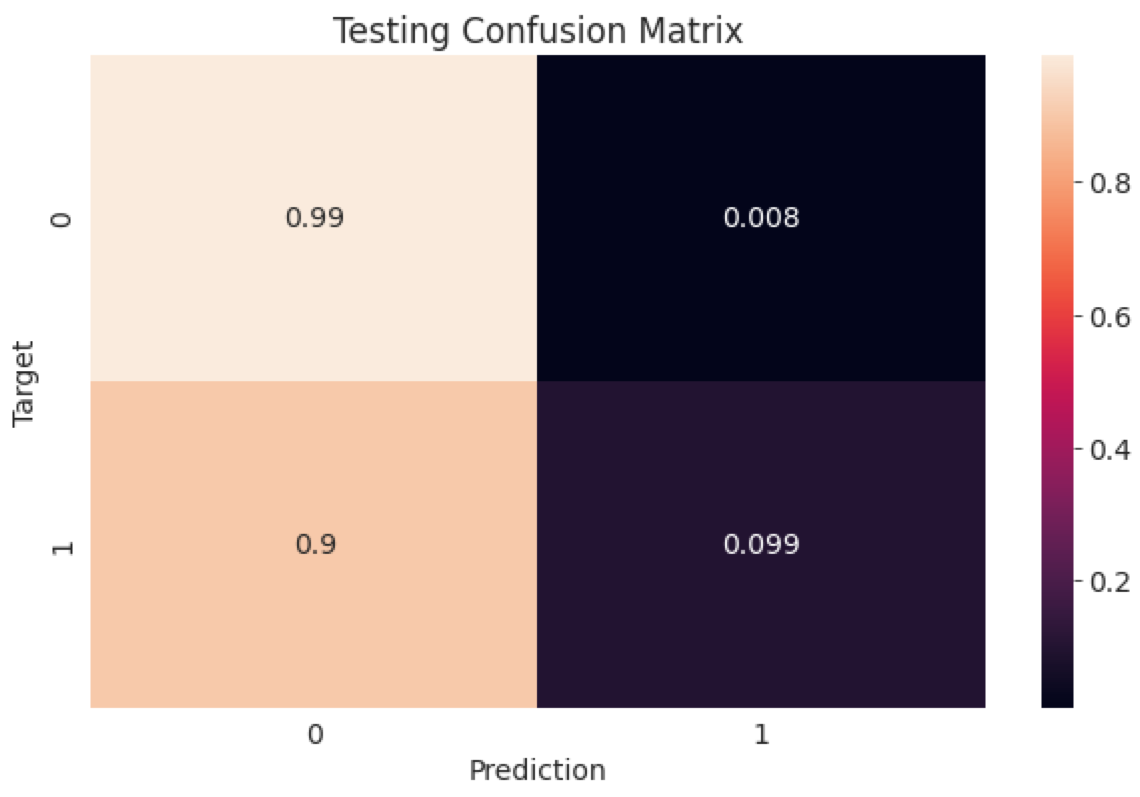
```
vals = predict_and_plot(x_val, val_target, 'Validation')
```

```
Accuracy: 91.49%
```



```
tests = predict_and_plot(x_test, test_target, 'Testing')
```

```
Accuracy: 91.39%
```



Prediction on single input

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

```
def predict_input(single_input):
    input_df = pd.DataFrame([single_input])
    input_df[numeric_cols] = imputer.transform(input_df[numeric_cols])
    input_df[numeric_cols] = scaler.transform(input_df[numeric_cols])
    input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
    x_input = input_df[numeric_cols+encoded_cols]
    pred=model.predict(x_input)[0]
    proba = model.predict_proba(x_input)[0][list(model.classes_).index(pred)]
    return pred, proba
```

```
new_input = {'BMI':20.2,
             'Smoking':'No',
             'AlcoholDrinking':'No',
             'Stroke':'No',
             'PhysicalHealth':5.0,
             'MentalHealth':25.0,
             'DiffWalking':'No',
             'Sex':'Female',
             'AgeCategory':'20-25',
             'Race':'White',
             'Diabetic':'No',
             'PhysicalActivity':'No',
             'GenHealth':'Good',
             'SleepTime':5.0,
             'Asthma':'No',
             'KidneyDisease':'No',
             'SkinCancer':'No'}
```

```
pred_new, pred_proba_new = predict_input(new_input)
pred_new, pred_proba_new
```

```
('No', 0.9717337871320096)
```

Our model predicted output "No" with the 71.1% probability for the new input which is unknown for the model.

```
import jovian
```

```
# Execute this to save new versions of the notebook
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'
```