```
!pip install -q keras
import keras
```

```
import glob
import tensorflow as tf
import io
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
import matplotlib.pyplot as plt
import keras as k
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
from google.colab import files
```

```
uploaded = files.upload()
```

Choose Files chronic_kid…disease.csv

- **chronic_kidney_disease.csv**(text/csv) - 29325 bytes, last modified: 10/24/2020 - 100% done

```
Saving chronic_kidney_disease.csv to chronic_kidney_disease.csv
```

```
from sklearn.model_selection import train_test_split
from sklearn.impute import KNNImputer
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

#Create function for checking missing values which accepts a dataframe as its parameter
```

```
def null_values_check(df):
    #Error handling to prevent abnormal termination of operation
    try:
        #if-else statement for null value check
        if(df.isnull().values.any() == True):
            #if there are null values present, print a column-wise summary of records with null values
            print('Number of null records within each column:\n' + str(df.isnull().sum()))
        else:
            print('There is no missing values in the dataset.')

    except Exception as e:
        logging.error(e)
dataset_name = 'kidney_disease.csv'

#error-handling to prevent abnormal termination of code
try:
    #import and load weather dataset into pandas dataframe
    chronic_kidney_disease_dataframe = pd.read_csv(io.BytesIO(uploaded['chronic_kidney_disease.csv']))

    #Description of Datasets
    #Print number of records and attributes of whole kidney dataset
    print('Shape of dataset: ' + str(chronic_kidney_disease_dataframe.shape))
    print('Total number of records in dataset = ' + str(chronic_kidney_disease_dataframe.shape[0]))
    print('Total number of attributes in dataset = ' + str(chronic_kidney_disease_dataframe.shape[1]))
    print('')
    #call function created to check for null values
    null_values_check(chronic_kidney_disease_dataframe)
    #Missing value imputation
    #replace ? to nan values
    chronic_kidney_disease_dataframe = chronic_kidney_disease_dataframe.replace('?', np.nan)

    #set the features and the target variables
    target_class = chronic_kidney_disease_dataframe['class']
    print('\nAre there missing values in Target Class? ' + str(target_class.isna().any()))
    feature_classes = chronic_kidney_disease_dataframe.iloc[:, 0:24]
    print('\nAre there missing values in the Features? \n' + str(feature_classes.isna().any()))

    #KNN imputation (n_neighbour = 5 means that the missing values will be replaced by the mean value of 5 nearest neighbors
    knn_missing_values_imputer = KNNImputer(n_neighbors=5)
    feature_classes = pd.DataFrame(knn_missing_values_imputer.fit_transform(feature_classes),
                                   columns = feature_classes.columns)
    print('\nNow, Are there any missing values in Features? ' + str(feature_classes.isna().any()))


    #Scaling and normalization of features
    standard_feature_scaler = StandardScaler()
    feature_classes = standard_feature_scaler.fit_transform(feature_classes)
    feature_classes = pd.DataFrame(feature_classes, columns=['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc',
                                                             'pcc', 'ba', 'bgr', 'bu', 'sc', 'sod', 'pot',
                                                             'hemo', 'pcv', 'wbcc', 'rbcc', 'htn', 'dm',
                                                             'cad', 'appet', 'pe', 'ane'])

    #Encoding target class using label encoding
    target_label_encoder = preprocessing.LabelEncoder()
    target_class = target_label_encoder.fit_transform(target_class)
    target_class1 = pd.DataFrame(target_class, columns=['class'])

    #split the dataset into training and testing data
    train_features, test_features, train_target, test_target = train_test_split(feature_classes, target_class,
                                                                                train_size = 0.8, test_size = 0.2)
    print('\nAfter Pre-processing:')
    print('Size of train dataset: ' + str(train_target.shape[0]))
    print('Size of test dataset: ' + str(test_target.shape[0]))

except FileNotFoundError as e:
    logging.error(e)
```

```
Are there missing values in the Features?
age       True

bp        True
sg        True
al        True
su        True
rbc      False
pc       False
pcc      False
ba       False
```

```
bgr       True
bu        True
sc        True
sod       True
pot       True
hemo      True
pcv      False
wbcc     False
rbcc     False
htn      False
dm       False
cad      False
appet    False
pe       False
ane      False
dtype: bool

Now, Are there any missing values in Features? age      False
bp       False
sg       False
al       False
su       False
rbc      False
pc       False
pcc      False
ba       False
bgr      False
bu       False
sc       False
sod      False
pot      False
hemo     False
pcv      False
wbcc     False
rbcc     False
htn      False
dm       False
cad      False
appet    False
pe       False
ane      False
dtype: bool
```

```
After Pre-processing:
Size of train dataset: 320
Size of test dataset: 80
```

```
from collections import Counter
from imblearn.over_sampling import SMOTE
counter=Counter(train_target)
print('before',counter)
smt=SMOTE()
x_train_sm,y_train_sm=smt.fit_resample(train_features,train_target)
counter=Counter(y_train_sm)
print('after',counter)
```

```
before Counter({0: 206, 1: 114})
after Counter({0: 206, 1: 206})
```

```
model = Sequential()

#first layer
model.add(Dense(32,input_dim= 24,kernel_initializer= k.initializers.random_normal(seed= 13),activation= 'relu'))

#second layer
model.add(Dense(32,input_dim= 24,kernel_initializer= k.initializers.random_normal(seed= 13),activation= 'relu'))
#final layer
model.add(Dense (1, activation = 'hard_sigmoid'))
```

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

```
history = model.fit(x_train_sm, y_train_sm, epochs =150, batch_size =x_train_sm.shape[0])
```

```
Epoch 122/150
1/1 [==============================] - 0s 11ms/step - loss: 0.0283 - accuracy: 0.9879
Epoch 123/150
1/1 [==============================] - 0s 10ms/step - loss: 0.0278 - accuracy: 0.9879
Epoch 124/150
1/1 [==============================] - 0s 13ms/step - loss: 0.0274 - accuracy: 0.9879
Epoch 125/150
1/1 [==============================] - 0s 12ms/step - loss: 0.0270 - accuracy: 0.9879
```

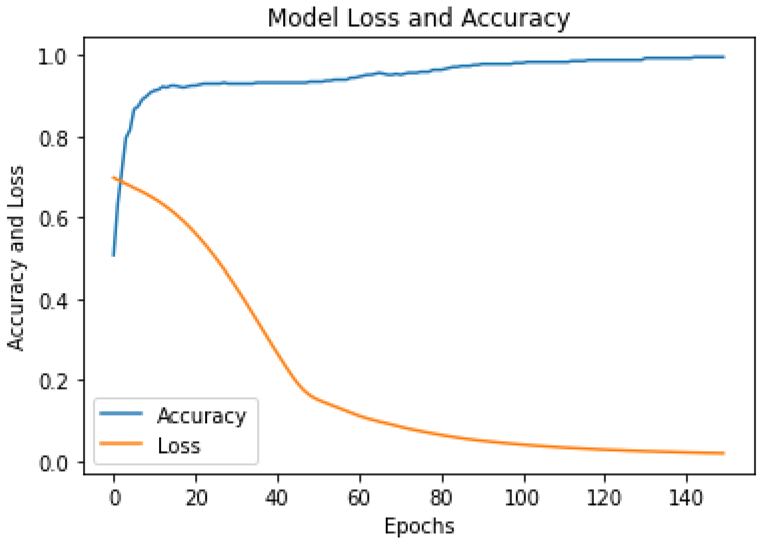
```
1/1 [==============================] - 0s 12ms/step - loss: 0.0270 - accuracy: 0.9879
Epoch 126/150
1/1 [==============================] - 0s 10ms/step - loss: 0.0267 - accuracy: 0.9879
Epoch 127/150
1/1 [==============================] - 0s 9ms/step - loss: 0.0263 - accuracy: 0.9879
Epoch 128/150
1/1 [==============================] - 0s 12ms/step - loss: 0.0259 - accuracy: 0.9879
Epoch 129/150
1/1 [==============================] - 0s 6ms/step - loss: 0.0255 - accuracy: 0.9879

Epoch 130/150
1/1 [==============================] - 0s 9ms/step - loss: 0.0251 - accuracy: 0.9879
Epoch 131/150
1/1 [==============================] - 0s 6ms/step - loss: 0.0248 - accuracy: 0.9927
Epoch 132/150
1/1 [==============================] - 0s 6ms/step - loss: 0.0245 - accuracy: 0.9927
Epoch 133/150
1/1 [==============================] - 0s 8ms/step - loss: 0.0242 - accuracy: 0.9927
Epoch 134/150
1/1 [==============================] - 0s 7ms/step - loss: 0.0238 - accuracy: 0.9927
Epoch 135/150
1/1 [==============================] - 0s 10ms/step - loss: 0.0235 - accuracy: 0.9927
Epoch 136/150
1/1 [==============================] - 0s 6ms/step - loss: 0.0232 - accuracy: 0.9927
Epoch 137/150
1/1 [==============================] - 0s 6ms/step - loss: 0.0229 - accuracy: 0.9927
Epoch 138/150
1/1 [==============================] - 0s 8ms/step - loss: 0.0227 - accuracy: 0.9927
Epoch 139/150
1/1 [==============================] - 0s 7ms/step - loss: 0.0224 - accuracy: 0.9927
Epoch 140/150
1/1 [==============================] - 0s 10ms/step - loss: 0.0222 - accuracy: 0.9927
Epoch 141/150
1/1 [==============================] - 0s 12ms/step - loss: 0.0219 - accuracy: 0.9927
Epoch 142/150
1/1 [==============================] - 0s 7ms/step - loss: 0.0217 - accuracy: 0.9927
Epoch 143/150
1/1 [==============================] - 0s 18ms/step - loss: 0.0214 - accuracy: 0.9951
Epoch 144/150
1/1 [==============================] - 0s 12ms/step - loss: 0.0212 - accuracy: 0.9951
Epoch 145/150
1/1 [==============================] - 0s 15ms/step - loss: 0.0209 - accuracy: 0.9951
Epoch 146/150
1/1 [==============================] - 0s 10ms/step - loss: 0.0207 - accuracy: 0.9951
```

```
1/1 [==============================] - 0s 10ms/step - loss: 0.0207 - accuracy: 0.9951
Epoch 147/150
1/1 [==============================] - 0s 10ms/step - loss: 0.0205 - accuracy: 0.9951
Epoch 148/150
1/1 [==============================] - 0s 12ms/step - loss: 0.0203 - accuracy: 0.9951
Epoch 149/150
1/1 [==============================] - 0s 11ms/step - loss: 0.0200 - accuracy: 0.9951
Epoch 150/150
1/1 [==============================] - 0s 13ms/step - loss: 0.0198 - accuracy: 0.9951
```

```
model.save('ckd.model');
```

```
INFO:tensorflow:Assets written to: ckd.model/assets
```

```
plt.plot(history.history['accuracy'],label='Accuracy')
plt.plot(history.history['loss'],label='Loss')
plt.title('Model Loss and Accuracy')
plt.ylabel('Accuracy and Loss')
plt.xlabel('Epochs')
plt.legend();
```



```
pred = model.predict(test_features)
pred = [1 if y>=0.5 else 0 for y in pred]

print('Original:  {0}'.format(",".join(str(x) for x in test_target)))
print('Predicted: {0}'.format(",".join(str(x) for x in pred)))
```

```
Original:  1,1,1,1,0,0,0,0,0,1,0,1,1,0,0,1,0,1,0,0,0,0,0,1,0,1,0,1,0,0,1,1,0,0,0,0,1,1,1,0,0,0,1,1,0,1,0,1,0,1,0,1,0,
Predicted: 1,1,1,1,0,0,0,0,0,1,0,1,1,0,0,1,0,1,0,0,0,0,0,1,0,1,0,1,0,0,1,1,0,0,0,0,1,1,1,0,0,0,1,1,0,1,0,1,0,1,0,1,0,
```

```
report = classification_report(test_target, pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        44
           1       1.00      0.97      0.99        36

    accuracy                           0.99        80
   macro avg       0.99      0.99      0.99        80
weighted avg       0.99      0.99      0.99        80
```

+ Code + Text

✓ 0s completed at 09:57 ● ✕