main

September 23, 2021

[1]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

#Tahmin Kütüphaneleri
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

0.1 Clean Data and Create Your Functions

[3]:
```
dataSet = pd.read_csv("Iris.csv")
del dataSet["Id"]

labelEncoder = preprocessing.LabelEncoder()

X = dataSet.iloc[:,:4].values
Y = dataSet.iloc[:,4:].values

Y = labelEncoder.fit_transform(Y)
standardScaler = StandardScaler()

x_train , x_test , y_train , y_test = train_test_split(X , Y , test_size= 0.33␣
↪, random_state=0)
```

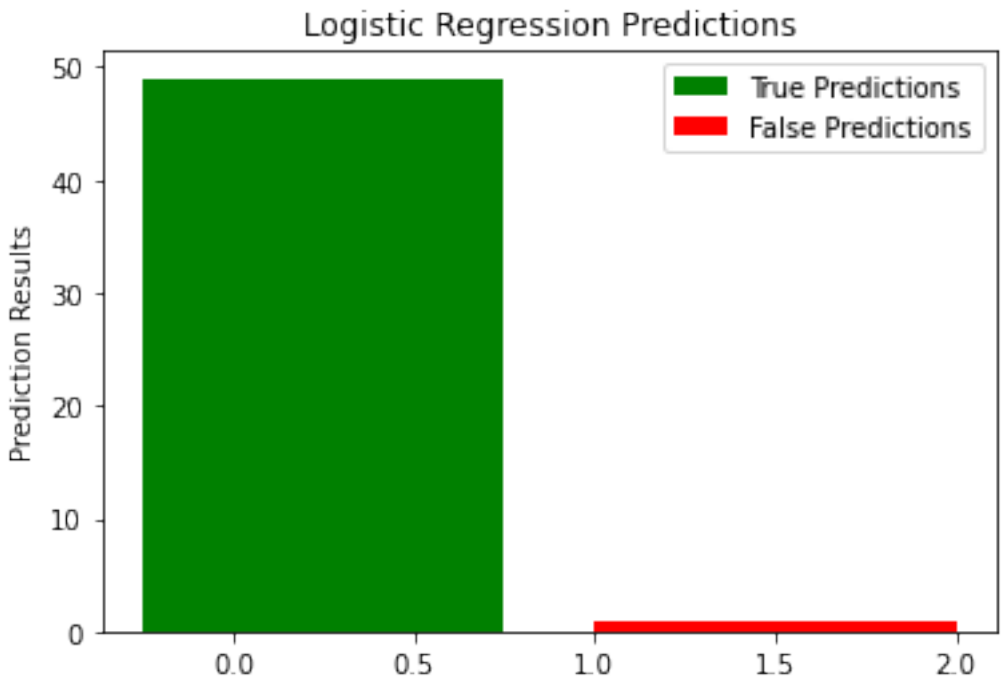
[21]:
```
def succesRateCalculate(matrix):

    truePred = matrix[0][0] + matrix[1][1] + matrix[2][2]
    falsePred = matrix[0][1] + matrix[1][0] + matrix[2][0] + + matrix[2][1] + +␣
↪matrix[1][2] + + matrix[0][2]
```

```
    succesRate = (truePred / (truePred + falsePred)) * 100

    return succesRate


def confPlot(matrix , name):
    truePred = matrix[0][0] + matrix[1][1] + matrix[2][2]
    falsePred = matrix[0][1] + matrix[1][0] + matrix[2][0] + + matrix[2][1] + +␣
 ↪matrix[1][2] + + matrix[0][2]

    plt.bar([0.25] , [truePred] , label = "True Predictions" , width=1 , color␣
 ↪= "green")
    plt.bar([1.50] , [falsePred] , label = "False Predictions" , width=1 ,␣
 ↪color = "red")
    plt.legend()
    plt.ylabel('Prediction Results')
    title = name + " Predictions"
    plt.title(title)
    plt.show()
```

[22]:
```
Succes_Rates = []

labels = ["Logistic Regression" , "KNN Algorithm" , "Support Vector Machine" ,␣
 ↪"Naive Bayes" , "Decision Tree" , "Rassal Forest" , "XGBoost" , "Deep␣
 ↪Learning"   ]
```

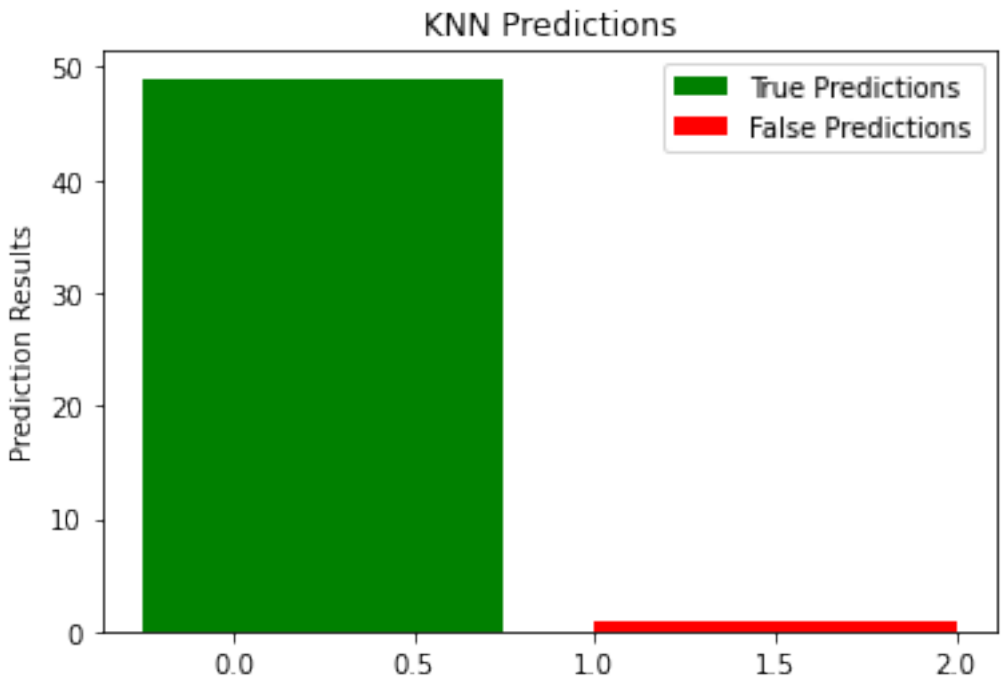
0.1.1 Logistic Regression

[23]:
```
logisticRegression = LogisticRegression(random_state = 0 )
logisticRegression.fit(x_train ,y_train)
logisticRegression_prediction = logisticRegression.predict(x_test)

logisticRegression_confmatrix = confusion_matrix(y_test ,␣
 ↪logisticRegression_prediction)

Succes_Rates.append(succesRateCalculate(logisticRegression_confmatrix).
 ↪astype(int))

confPlot(logisticRegression_confmatrix , "Logistic Regression")
```



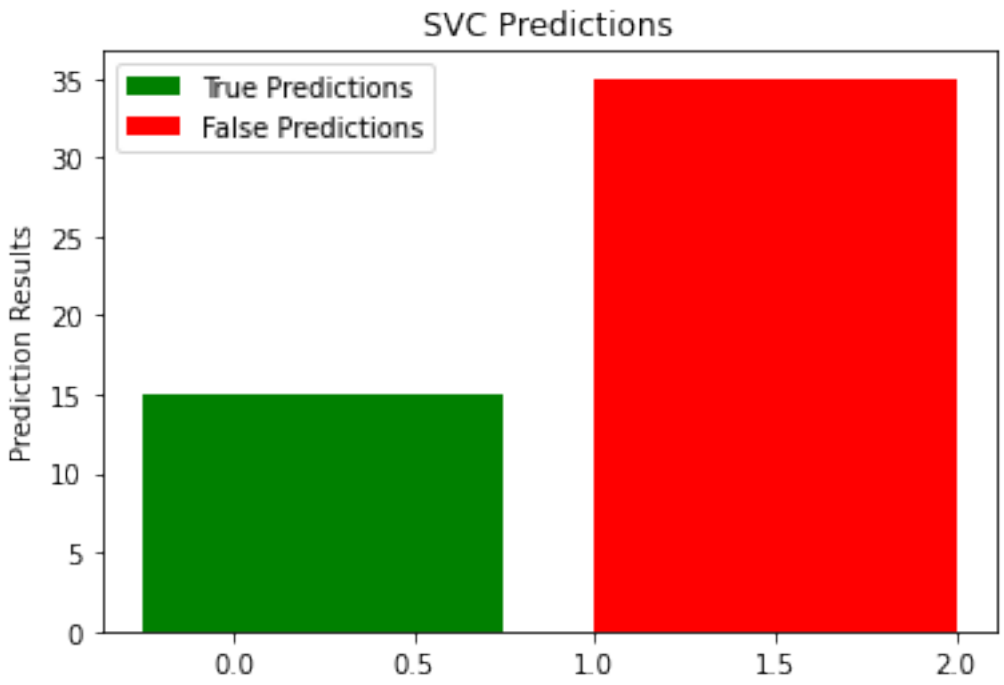
0.1.2 KNN Algorithm

[24]:
```
knn = KNeighborsClassifier(n_neighbors=5 , metric="minkowski")
knn.fit(x_train , y_train)
knn_predictions = knn.predict(x_test)

knn_confmatrix = confusion_matrix(knn_predictions , y_test)
Succes_Rates.append(succesRateCalculate(knn_confmatrix).astype(int))
confPlot(knn_confmatrix, "KNN")
```



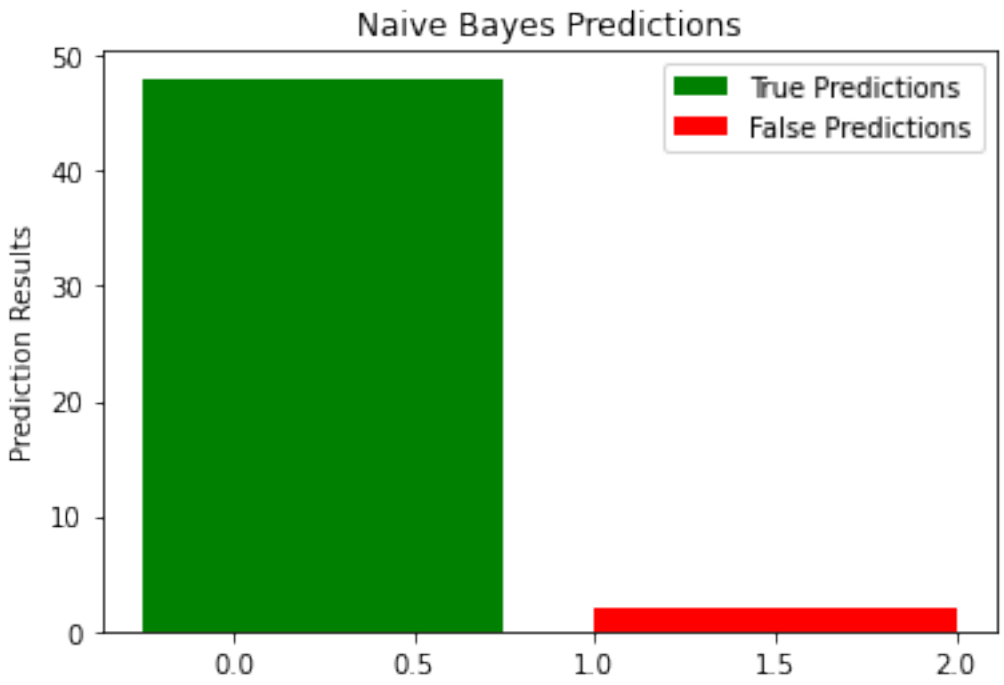
0.1.3 Support Vector Machine

[25]:
```
svc = SVC(kernel = "linear")
x_train_ss = standardScaler.fit_transform(x_train)
x_test_ss = standardScaler.fit_transform(x_test)

svc.fit(x_train_ss , y_train)
svc_predictions = svc.predict(x_test)

svc_confmatrix = confusion_matrix(y_test , svc_predictions)
Succes_Rates.append(succesRateCalculate(svc_confmatrix).astype(int))
confPlot(svc_confmatrix , "SVC")
```



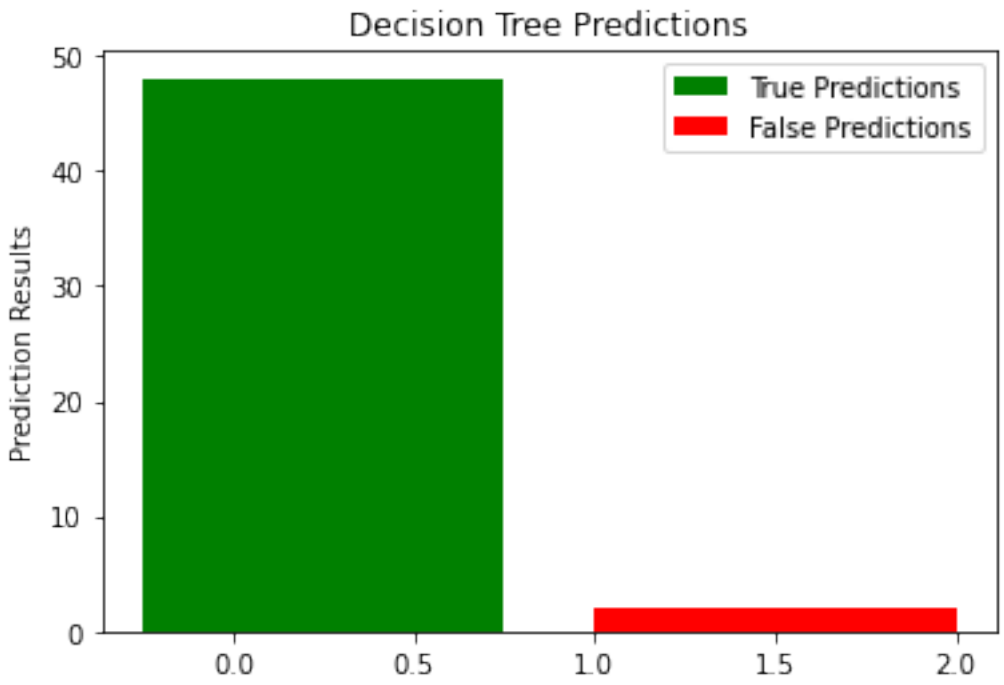
0.1.4 Naive Bayes

[26]:
```
gnb = GaussianNB()
gnb.fit(x_train , y_train)
gnb_predictions = gnb.predict(x_test)

gnb_confmatrix = confusion_matrix(y_test , gnb_predictions)
Succes_Rates.append(succesRateCalculate(gnb_confmatrix).astype(int))
confPlot(gnb_confmatrix , "Naive Bayes")
```



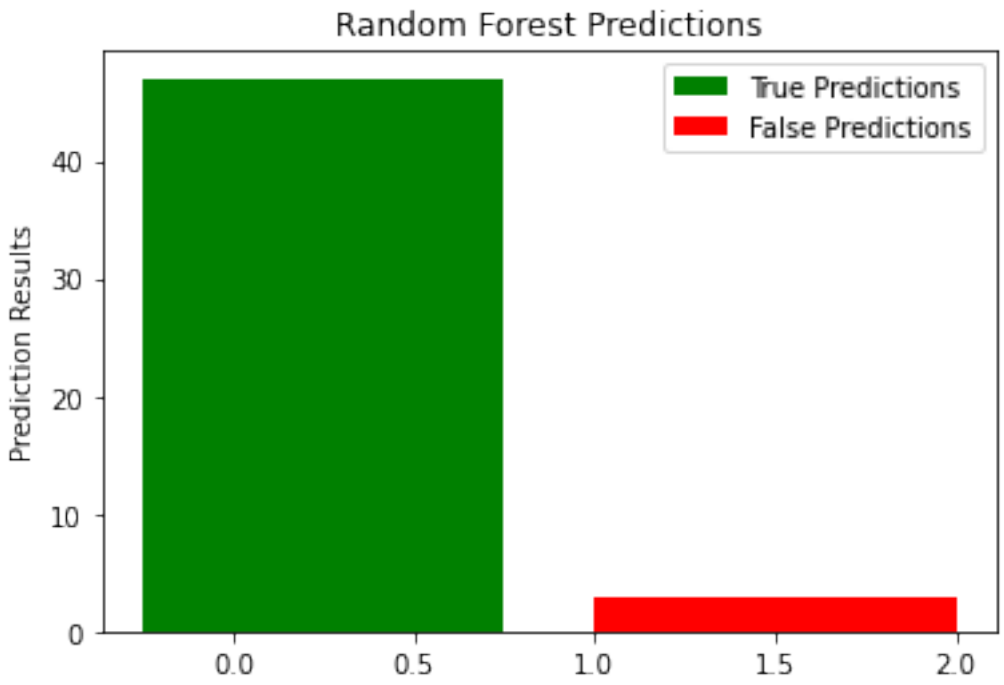
0.1.5 Decision Tree

[27]:
```
decisionTree = DecisionTreeClassifier(criterion="entropy")
decisionTree.fit(x_train , y_train)
decisionTree_predictions = decisionTree.predict(x_test)

decisionTree_confmatrix = confusion_matrix(y_test , decisionTree_predictions)
Succes_Rates.append(succesRateCalculate(decisionTree_confmatrix).astype(int))
confPlot(decisionTree_confmatrix , "Decision Tree")
```



0.1.6 Rassal Forest

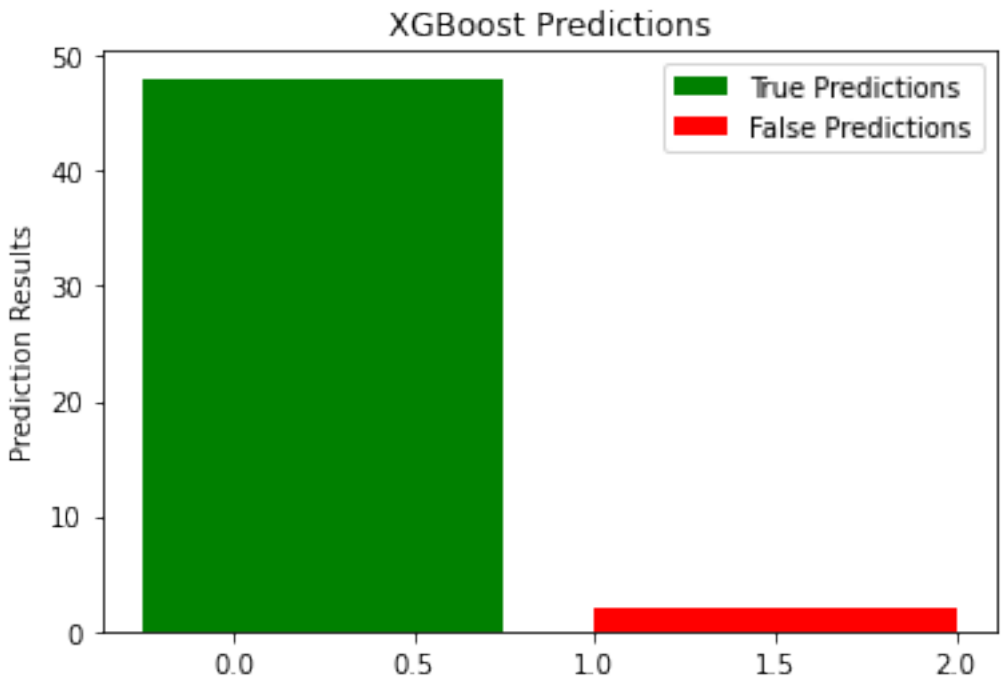
[28]:
```
randomForest = RandomForestClassifier(n_estimators=5)
randomForest.fit(x_train , y_train)
randomForest_predictions = randomForest.predict(x_test)

randomForest_confmatrix = confusion_matrix(y_test , randomForest_predictions)
Succes_Rates.append(succesRateCalculate(randomForest_confmatrix).astype(int))
confPlot(randomForest_confmatrix , "Random Forest")
```



0.1.7 XGBoost

[29]:
```
xgbClassifier = XGBClassifier()
xgbClassifier.fit(x_train , y_train)
xgb_predictions = xgbClassifier.predict(x_test)

xgb_confmatrix = confusion_matrix(y_test , xgb_predictions)
Succes_Rates.append(succesRateCalculate(xgb_confmatrix).astype(int))
confPlot(xgb_confmatrix , "XGBoost")
```

```
[17:37:50] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'multi:softprob' was changed
from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore
the old behavior.

C:\Users\samit\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in a
future release. To remove this warning, do the following: 1) Pass option
use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, …, [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

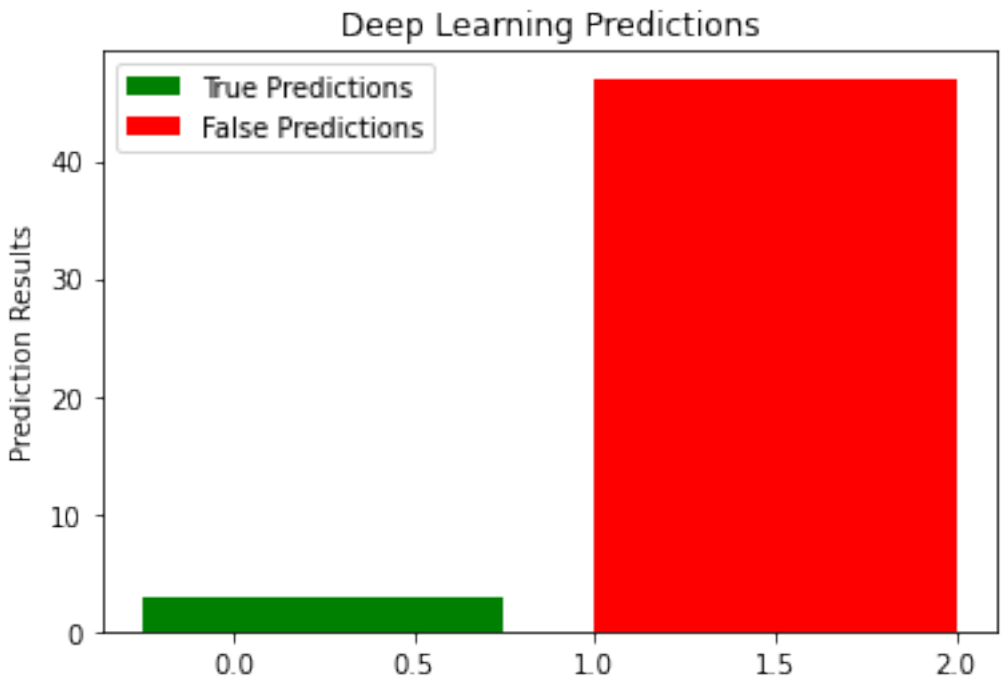


0.1.8 Deep Learning

[32]:
```
from keras.models import Sequential
from keras.layers import Dense

EPOCHS = 10

classifier = Sequential()

classifier.add(Dense(25 , activation = "relu" , input_dim = 4 , name =␣
↪"input-layer" ))
classifier.add(Dense(25 , activation = "relu" , name = "hidden-layer"))
classifier.add(Dense(1 , activation = "sigmoid" ,name = "output-layer"))

classifier.compile(optimizer = "adam", loss = "categorical_crossentropy" ,␣
↪metrics=['accuracy'])

classifier.fit(x_train_ss , y_train , epochs=EPOCHS)

dl_predictions = classifier.predict(x_test_ss)
dl_predictions = (dl_predictions > 0.5)

dl_confmatrix = confusion_matrix(y_test  , dl_predictions)
Succes_Rates.append(succesRateCalculate(dl_confmatrix).astype(int))
```

```
confPlot(dl_confmatrix , "Deep Learning")
```

```
Epoch 1/10
4/4 [==============================] - 0s 748us/step - loss: 1.2040e-07 -
accuracy: 0.0800
Epoch 2/10
4/4 [==============================] - 0s 997us/step - loss: 1.2040e-07 -
accuracy: 0.0800
Epoch 3/10
4/4 [==============================] - 0s 499us/step - loss: 1.2040e-07 -
accuracy: 0.0800
Epoch 4/10
4/4 [==============================] - 0s 748us/step - loss: 1.2040e-07 -
accuracy: 0.0800
Epoch 5/10
4/4 [==============================] - 0s 498us/step - loss: 1.2040e-07 -
accuracy: 0.0800
Epoch 6/10
4/4 [==============================] - 0s 499us/step - loss: 1.2040e-07 -
accuracy: 0.0800
Epoch 7/10
4/4 [==============================] - 0s 499us/step - loss: 1.2040e-07 -
accuracy: 0.0800
Epoch 8/10
4/4 [==============================] - 0s 748us/step - loss: 1.2040e-07 -
accuracy: 0.0800
Epoch 9/10
4/4 [==============================] - 0s 499us/step - loss: 1.2040e-07 -
accuracy: 0.0800
Epoch 10/10
4/4 [==============================] - 0s 748us/step - loss: 1.2040e-07 -
accuracy: 0.0800
WARNING:tensorflow:5 out of the last 9 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x00000207BA8F28B0>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```


Deep Learning Predictions
True Predictions
False Predictions
Prediction Results
0
10
20
30
40
0.0
0.5
1.0
1.5
2.0