Machine Learning Tutorial With Multipler Classifiers

Namani.Vamshi Krishna

SAK Informatics

Address: Plot No. 544, #102, Shiva Sai Murali Ramaneeyam, near Peacock Circle, Pragathi Nagar, Hyderabad, Telangana 500090

Phone: 090001 88676

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
iris=pd.read_csv(r"Iris.csv")
```

[3]:
```
iris
iris.head()
```

[3]:

	150	4	setosa	versicolor	disorder
0	5.1	3.5	1.4	0.2	Normal
1	4.9	3.0	1.4	0.2	Normal
2	4.7	3.2	1.3	0.2	Normal
3	4.6	3.1	1.5	0.2	Normal
4	5.0	3.6	1.4	0.2	Normal

[4]:
```
X = iris.iloc[:, 0:4].values
X
```

[4]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

```
[5.1, 3.8, 1.5, 0.3],
[5.4, 3.4, 1.7, 0.2],
[5.1, 3.7, 1.5, 0.4],
[4.6, 3.6, 1. , 0.2],
[5.1, 3.3, 1.7, 0.5],
[4.8, 3.4, 1.9, 0.2],
[5. , 3. , 1.6, 0.2],
[5. , 3.4, 1.6, 0.4],
[5.2, 3.5, 1.5, 0.2],
[5.2, 3.4, 1.4, 0.2],
[4.7, 3.2, 1.6, 0.2],
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
```

```
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
```

```
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

[5]:
```
y = iris.iloc[:, 4].values
y
```

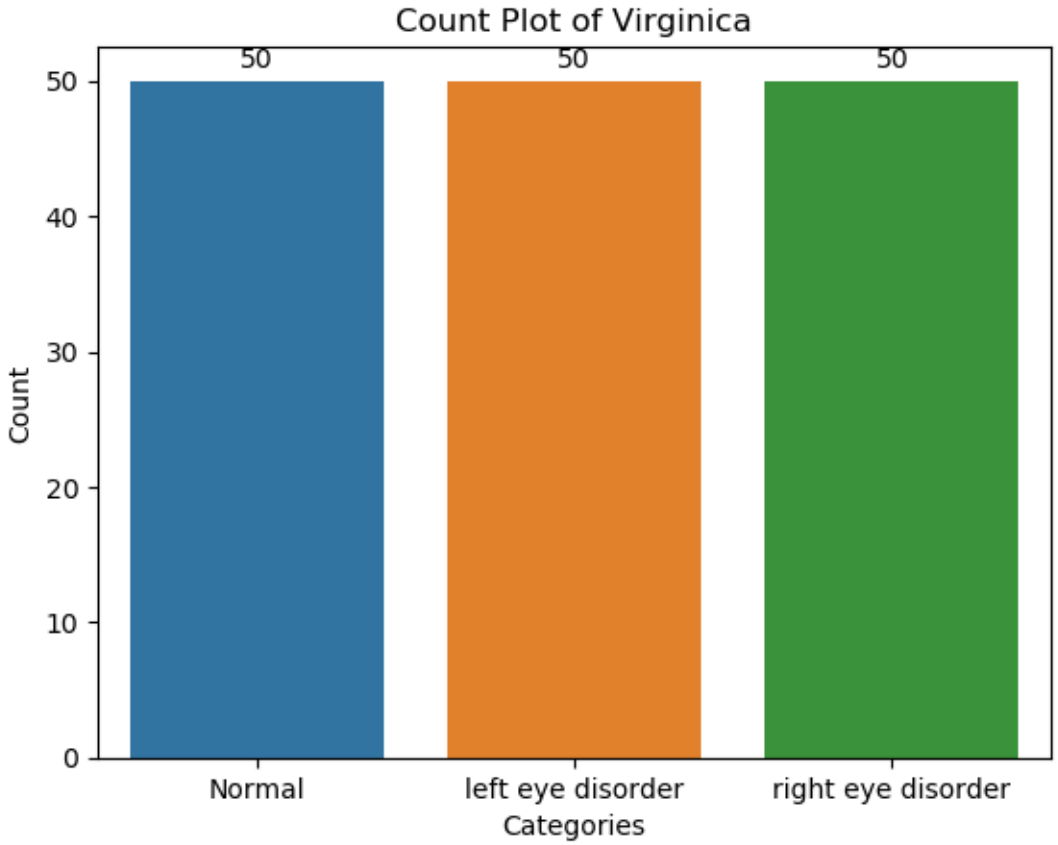
[5]:
```
array(['Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal',
       'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal',
       'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal',
       'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal',
       'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal',
       'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal',
```

```
       'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal',
       'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal',
       'Normal', 'Normal', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'left eye disorder', 'left eye disorder', 'left eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder', 'right eye disorder',
       'right eye disorder', 'right eye disorder'], dtype=object)
```

[6]:

```
import seaborn as sns
import matplotlib.pyplot as plt
plot = sns.countplot(data=iris, x="disorder")

# Get the count of each category
value_counts = iris["disorder"].value_counts()

# Display the count values on top of the bars
for i, count in enumerate(value_counts):
    plot.text(x=i, y=count + 1, s=str(count), ha="center")
```

```
# Customize the plot
plot.set_xlabel("Categories")
plot.set_ylabel("Count")
plot.set_title("Count Plot of Virginica")

# Show the plot
plt.show()
```



[7]:
```
labels = set(y)
labels
```

[7]:
```
{'Normal', 'left eye disorder', 'right eye disorder'}
```

[8]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

[9]:
```
y = le.fit_transform(y)
y
```

[9]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

[ ]:

[10]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)
print("X shape is",X.shape)
print("Y shape is",y.shape)
print("X_test shape is",X_test.shape)
print("X_train shape is",X_train.shape)
print("y_train shape is",y_train.shape)
print("y_test shape is",y_test.shape)
```

```
X shape is (150, 4)
Y shape is (150,)
X_test shape is (30, 4)
X_train shape is (120, 4)
y_train shape is (120,)
y_test shape is (30,)
```

[11]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear')
lr.fit(X_train,y_train)
y_pred = lr.predict(X_test)
print("Original y_test values are ",y_test)
print("predicted y_test values are",y_pred)
```

```
Original y_test values are  [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
predicted y_test values are [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 2 0 0 2 0
0 1 1 0]
```

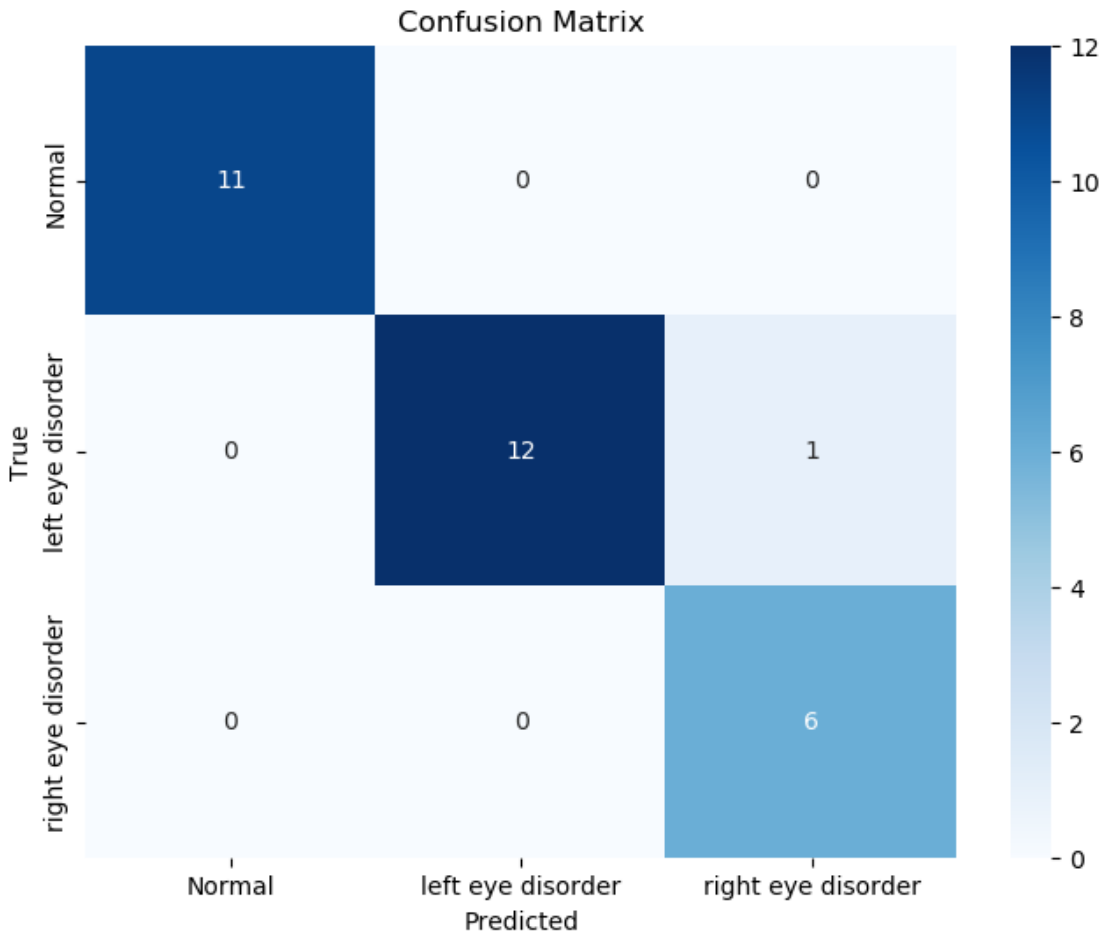
[12]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
# Target is multiclass but average='binary'. Please choose another average␣
 ↪setting, one of [None, 'micro', 'macro', 'weighted'].
accuracy = accuracy_score(y_test, y_pred)
# Precision
```

```
precision = precision_score(y_test, y_pred, average='weighted')
# Recall
recall = recall_score(y_test, y_pred, average='weighted')
# F1 score
f1 = f1_score(y_test, y_pred, average='weighted')
```

[13]:
```
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')
```

```
Accuracy: 0.97
Precision: 0.97
Recall: 0.97
F1 Score: 0.97
```

[14]:
```
from sklearn.metrics import confusion_matrix

# Assuming y_test and y_pred are your true labels and predicted labels,␣
 ↪respectively
conf_matrix = confusion_matrix(y_test, y_pred)

# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

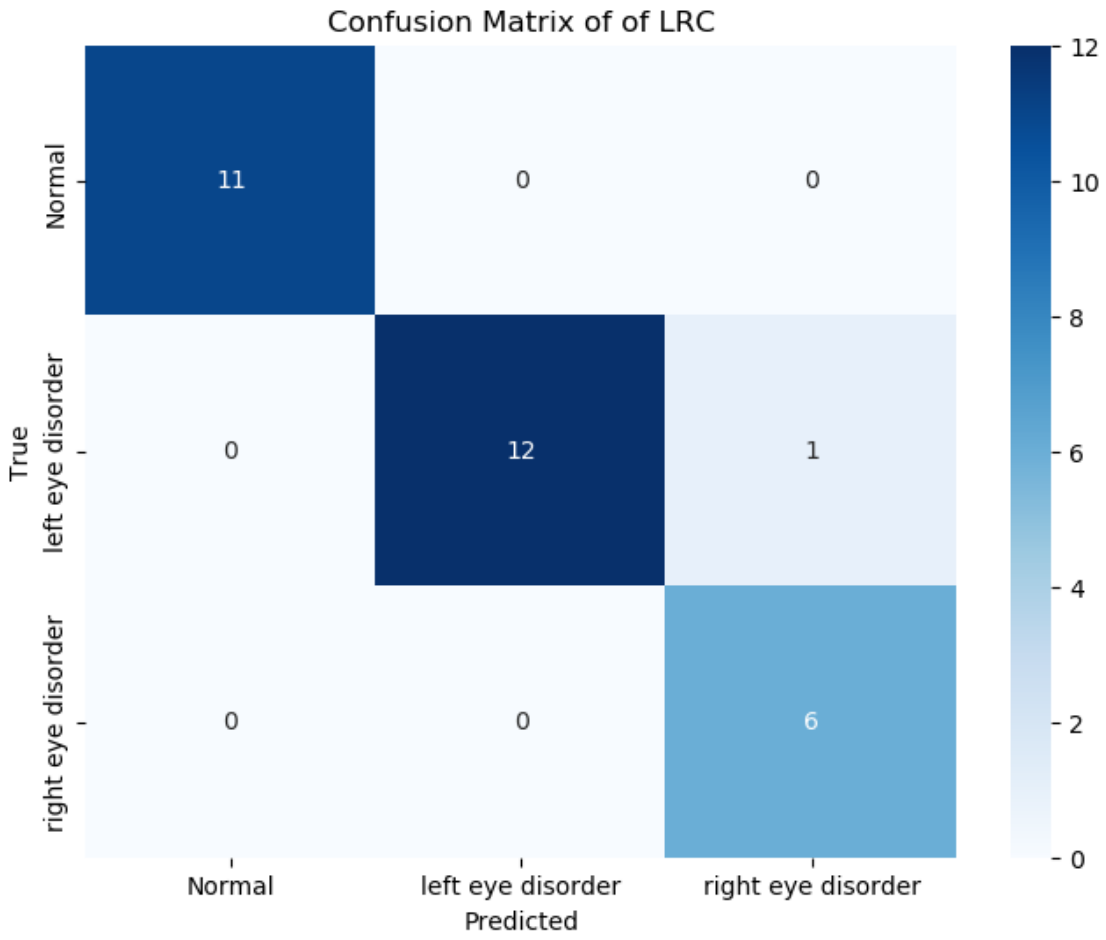


[15]:
```
from sklearn.metrics import classification_report

class_report = classification_report(y_test, y_pred, target_names=labels)

# Print the classification report with labels
print('Classification Report:')
print(class_report)
```

```
Classification Report:
                    precision    recall  f1-score   support

            Normal       1.00      1.00      1.00        11
 left eye disorder       1.00      0.92      0.96        13
right eye disorder       0.86      1.00      0.92         6

          accuracy                           0.97        30
         macro avg       0.95      0.97      0.96        30
      weighted avg       0.97      0.97      0.97        30
```

[16]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,y_train)
y_pred = lr.predict(X_test)

print("Original y_test values are ",y_test)
print("predicted y_test values are",y_pred)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')


conf_matrix = confusion_matrix(y_test, y_pred)
# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix of of LRC')
plt.show()

class_report = classification_report(y_test, y_pred, target_names=labels)

# Print the classification report with labels
print('Classification Report of LRC:')
print(class_report)
```

```
Original y_test values are  [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
predicted y_test values are [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 2 0 0 2 0
0 1 1 0]
Accuracy: 0.97
Precision: 0.97
Recall: 0.97
F1 Score: 0.97
```

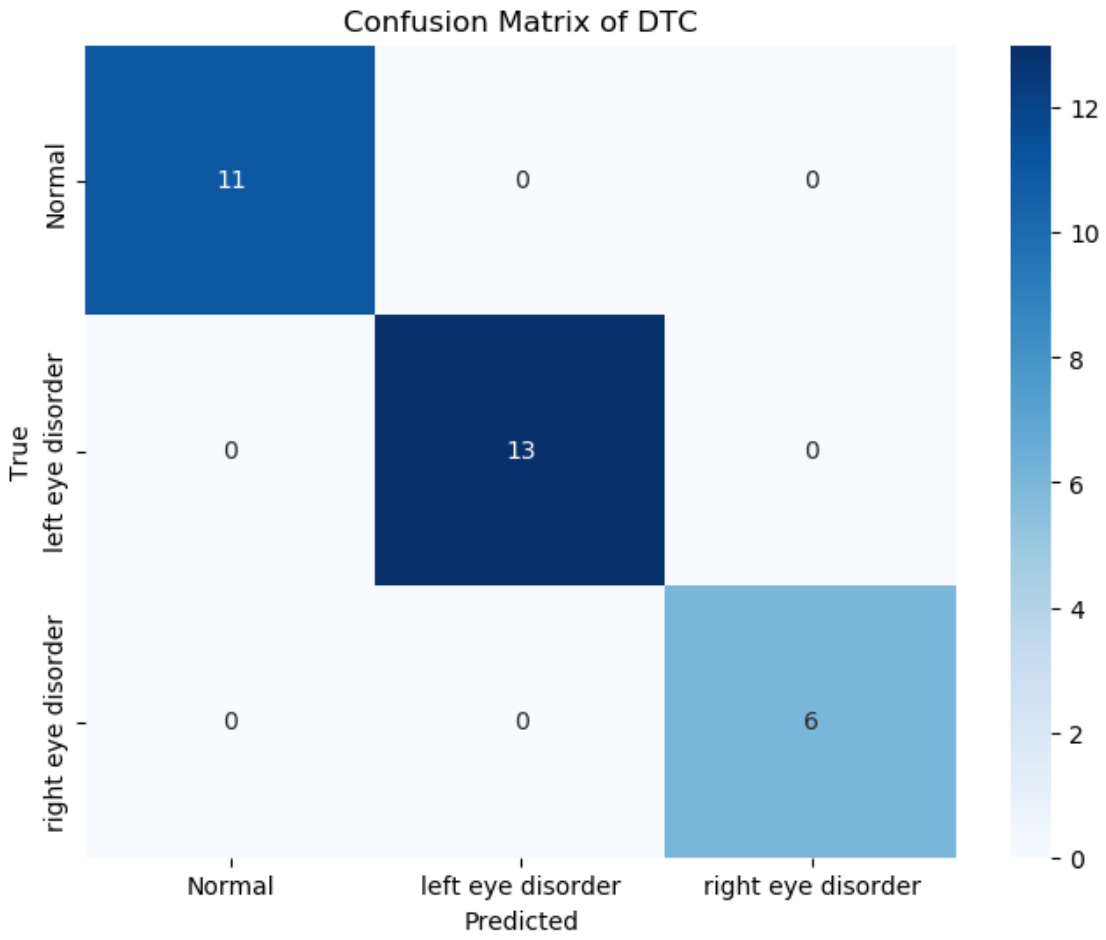



```
Classification Report of LRC:
                    precision    recall  f1-score   support

            Normal       1.00      1.00      1.00        11
 left eye disorder       1.00      0.92      0.96        13
right eye disorder       0.86      1.00      0.92         6

          accuracy                           0.97        30
         macro avg       0.95      0.97      0.96        30
      weighted avg       0.97      0.97      0.97        30
```

[17]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,y_train)
y_pred = dt.predict(X_test)

print("Original y_test values are ",y_test)
```

```
print("predicted y_test values are",y_pred)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')


conf_matrix = confusion_matrix(y_test, y_pred)
# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix of DTC')
plt.show()

class_report = classification_report(y_test, y_pred, target_names=labels)

# Print the classification report with labels
print('Classification Report of DTC:')
print(class_report)
```

```
Original y_test values are  [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
predicted y_test values are [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```

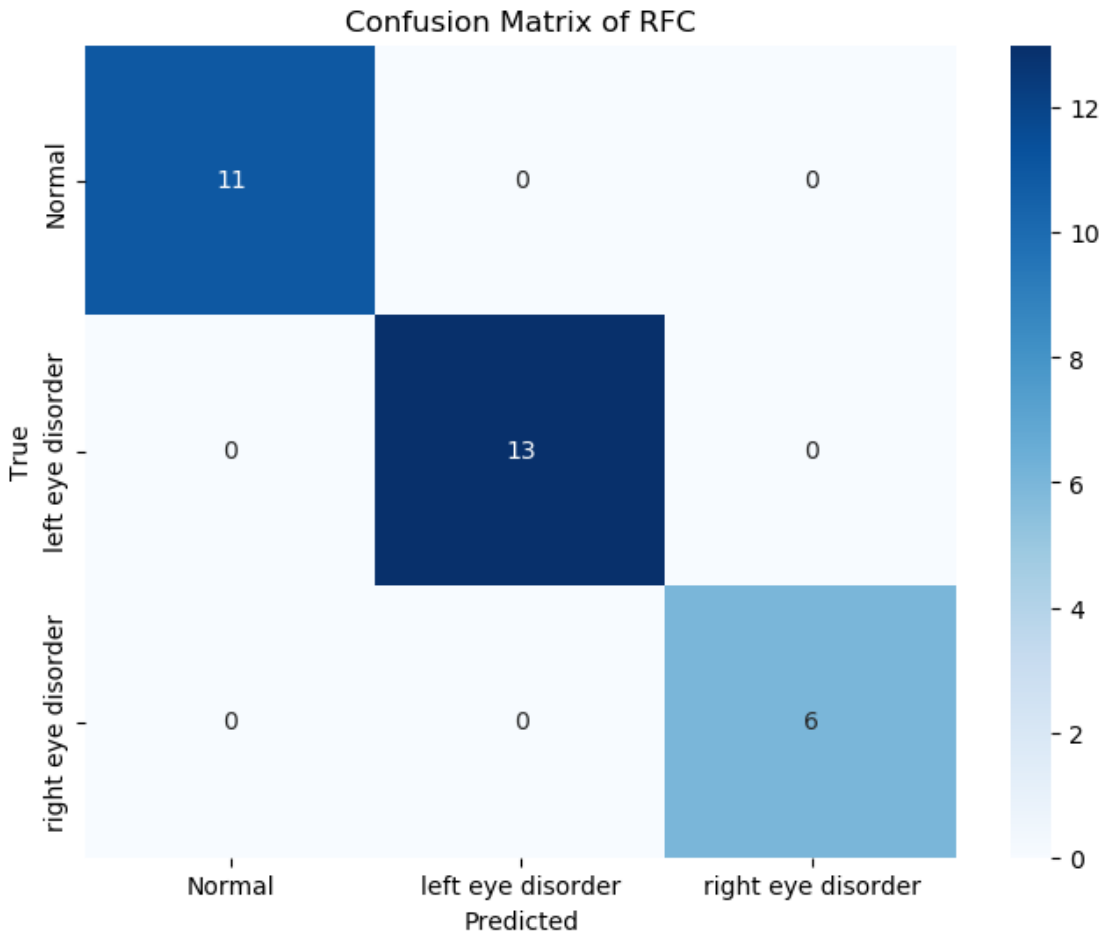



```
Classification Report of DTC:
                    precision    recall  f1-score   support

            Normal       1.00      1.00      1.00        11
 left eye disorder       1.00      1.00      1.00        13
right eye disorder       1.00      1.00      1.00         6

          accuracy                           1.00        30
         macro avg       1.00      1.00      1.00        30
      weighted avg       1.00      1.00      1.00        30
```

[18]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)

print("Original y_test values are ",y_test)
```

```
print("predicted y_test values are",y_pred)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')


conf_matrix = confusion_matrix(y_test, y_pred)
# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix of RFC')
plt.show()

class_report = classification_report(y_test, y_pred, target_names=labels)

# Print the classification report with labels
print('Classification Report of RFC:')
print(class_report)
```

```
Original y_test values are  [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
predicted y_test values are [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```

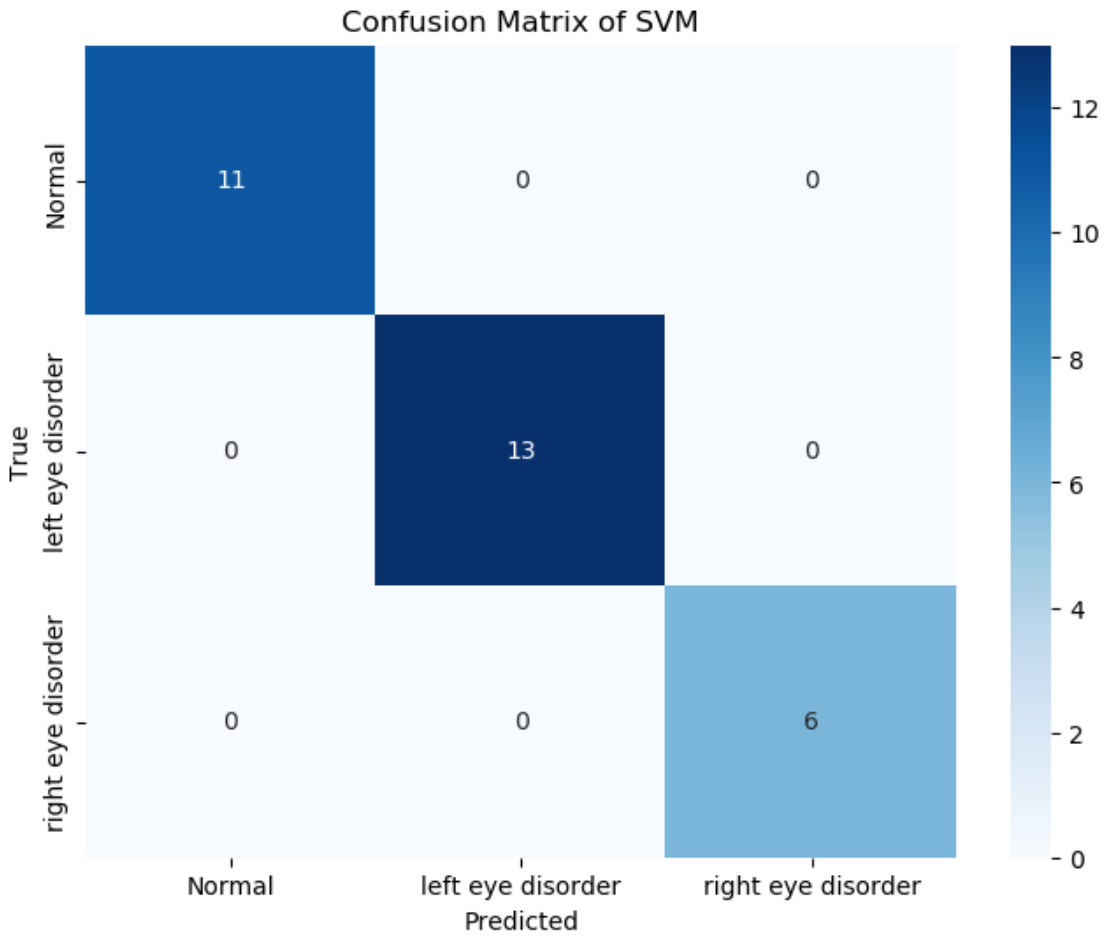



```
Classification Report of RFC:
                    precision    recall  f1-score   support

            Normal       1.00      1.00      1.00        11
 left eye disorder       1.00      1.00      1.00        13
right eye disorder       1.00      1.00      1.00         6

          accuracy                           1.00        30
         macro avg       1.00      1.00      1.00        30
      weighted avg       1.00      1.00      1.00        30
```

[19]:
```
from sklearn.svm import SVC
svm = SVC()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)

print("Original y_test values are ",y_test)
```

```
print("predicted y_test values are",y_pred)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')


conf_matrix = confusion_matrix(y_test, y_pred)
# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix of SVM')
plt.show()

class_report = classification_report(y_test, y_pred, target_names=labels)

# Print the classification report with labels
print('Classification Report of SVM:')
print(class_report)
```

```
Original y_test values are  [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
predicted y_test values are [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```

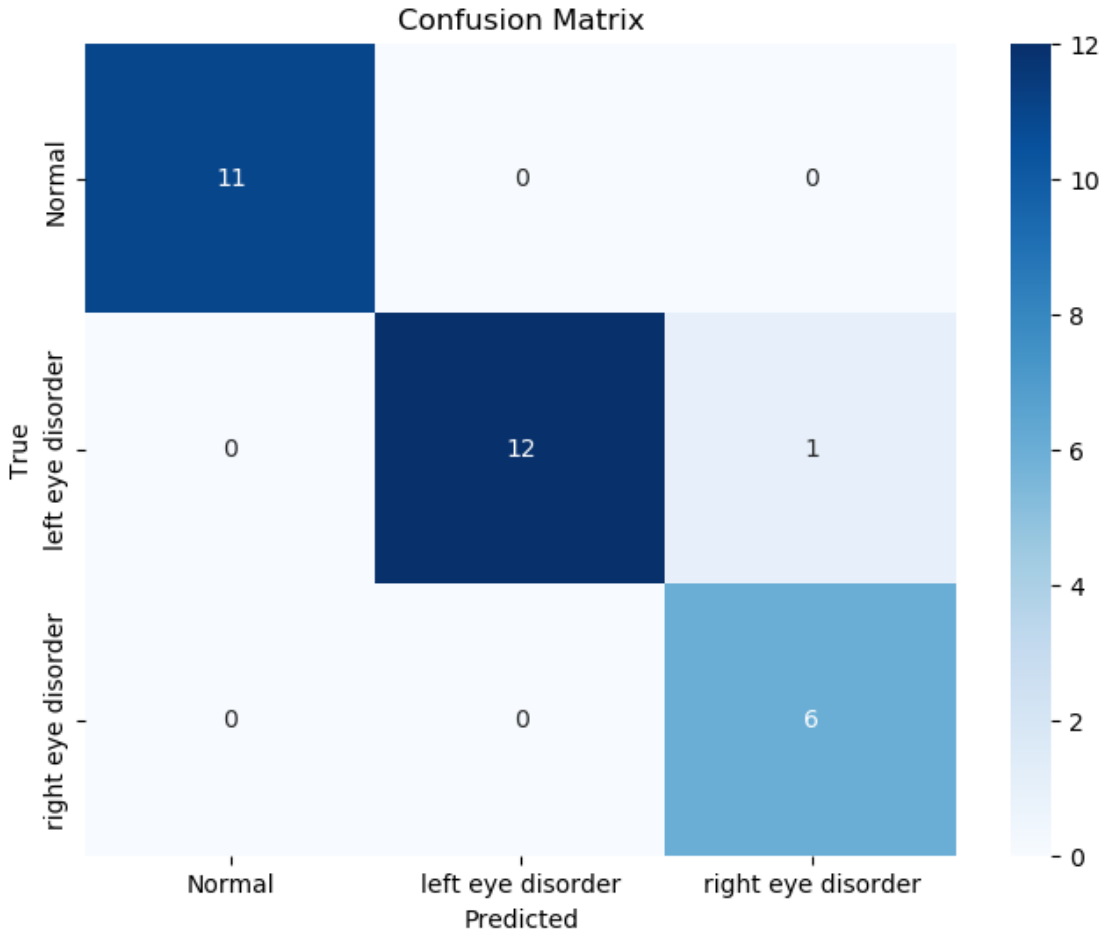



```
Classification Report of SVM:
                    precision    recall  f1-score   support

            Normal       1.00      1.00      1.00        11
 left eye disorder       1.00      1.00      1.00        13
right eye disorder       1.00      1.00      1.00         6

          accuracy                           1.00        30
         macro avg       1.00      1.00      1.00        30
      weighted avg       1.00      1.00      1.00        30
```

[20]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

```
# The rest of your code remains the same
print("Original y_test values are ",y_test)
print("predicted y_test values are",y_pred)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')


conf_matrix = confusion_matrix(y_test, y_pred)
# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

class_report = classification_report(y_test, y_pred, target_names=labels)

# Print the classification report with labels
print('Classification Report:')
print(class_report)
```

```
Original y_test values are  [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
predicted y_test values are [2 1 0 2 0 2 0 1 1 1 2 1 1 1 2 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
Accuracy: 0.97
Precision: 0.97
Recall: 0.97
F1 Score: 0.97
```



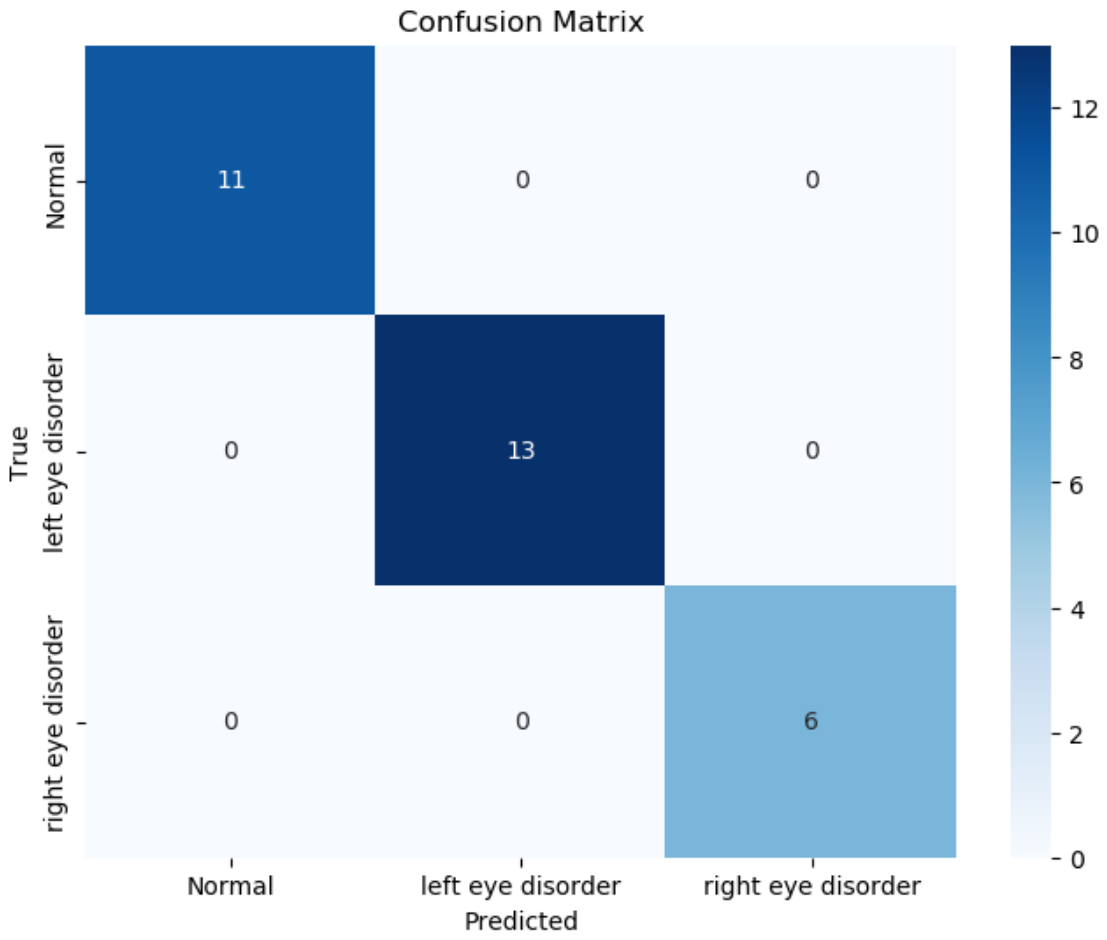

```
Classification Report:
                    precision    recall  f1-score   support

            Normal       1.00      1.00      1.00        11
 left eye disorder       1.00      0.92      0.96        13
right eye disorder       0.86      1.00      0.92         6

          accuracy                           0.97        30
         macro avg       0.95      0.97      0.96        30
      weighted avg       0.97      0.97      0.97        30
```

[21]:
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier()
mlp.fit(X_train, y_train)
y_pred = mlp.predict(X_test)
```

```
# The rest of your code remains the same



# The rest of your code remains the same
print("Original y_test values are ",y_test)
print("predicted y_test values are",y_pred)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')


conf_matrix = confusion_matrix(y_test, y_pred)
# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

class_report = classification_report(y_test, y_pred, target_names=labels)

# Print the classification report with labels
print('Classification Report:')
print(class_report)
```

```
c:\users\asus\appdata\local\programs\python\python37\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:585:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Original y_test values are  [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
predicted y_test values are [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```

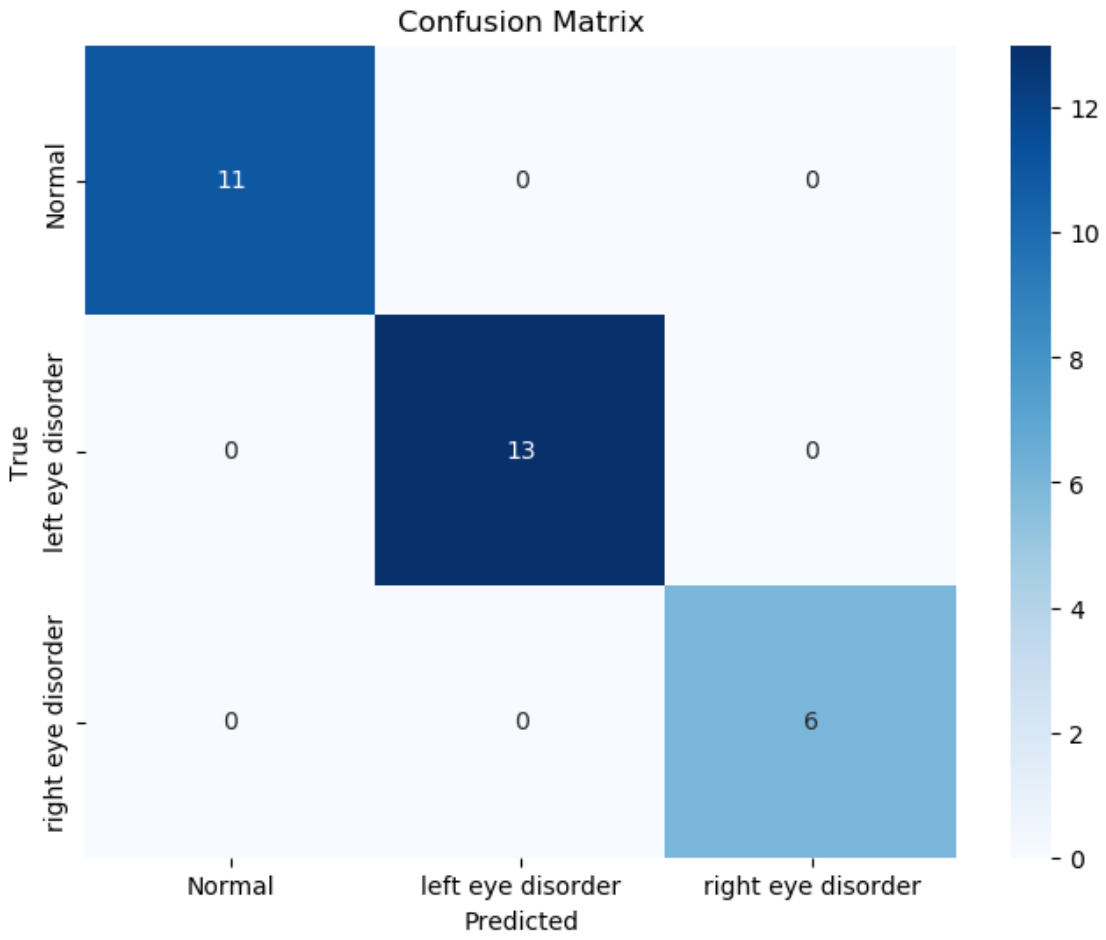



```
Classification Report:
                    precision    recall  f1-score   support

            Normal       1.00      1.00      1.00        11
 left eye disorder       1.00      1.00      1.00        13
right eye disorder       1.00      1.00      1.00         6

          accuracy                           1.00        30
         macro avg       1.00      1.00      1.00        30
      weighted avg       1.00      1.00      1.00        30
```

[22]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred = lin_reg.predict(X_test)
y_pred=np.round(y_pred)
```

```
print("Original y_test values are ",y_test)
print("predicted y_test values are",y_pred)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')


conf_matrix = confusion_matrix(y_test, y_pred)
# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

class_report = classification_report(y_test, y_pred, target_names=labels)

# Print the classification report with labels
print('Classification Report:')
print(class_report)
```

```
Original y_test values are  [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
predicted y_test values are [ 2.  1. -0.  2. -0.  2. -0.  1.  1.  1.  2.  1.  1.
1.  1. -0.  1.  1.
  0. -0.  2.  1.  0.  0.  2. -0.  0.  1.  1.  0.]
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```

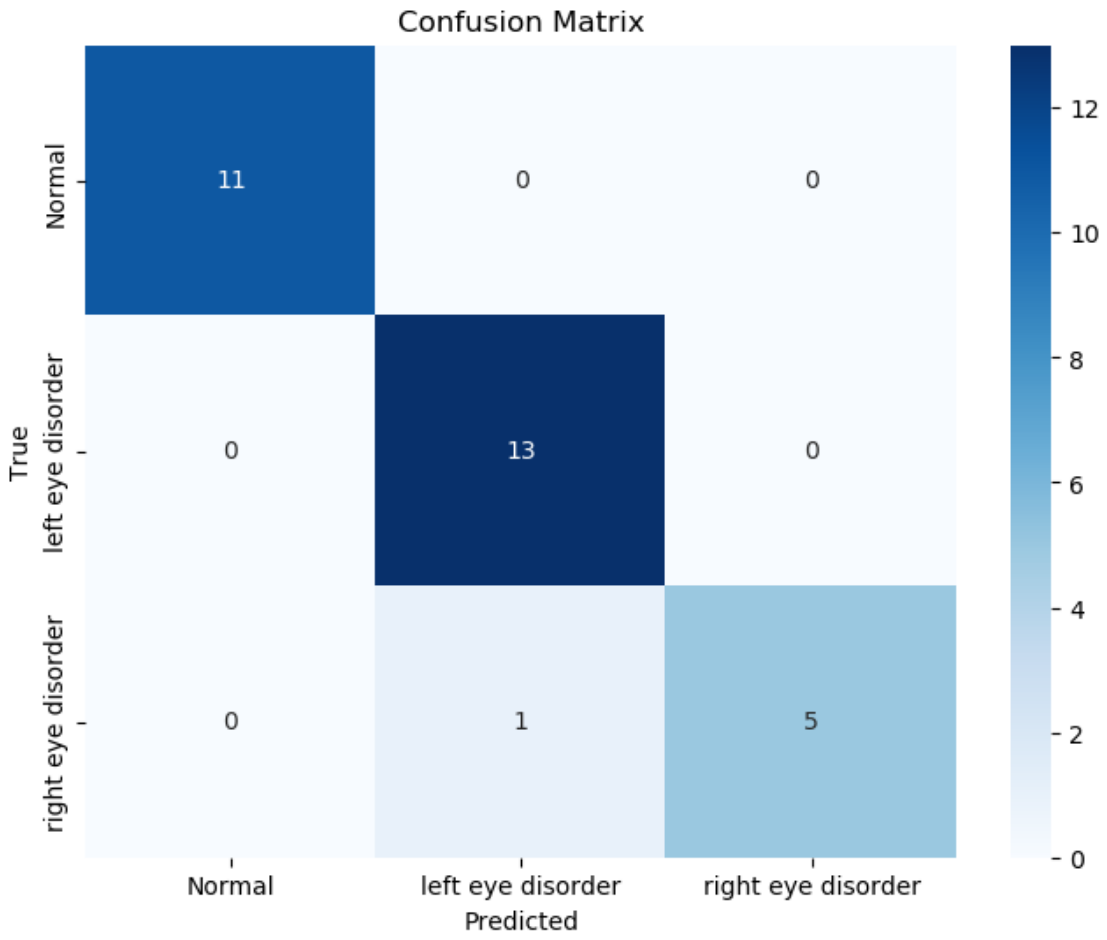



```
Classification Report:
                    precision    recall  f1-score   support

            Normal       1.00      1.00      1.00        11
 left eye disorder       1.00      1.00      1.00        13
right eye disorder       1.00      1.00      1.00         6

          accuracy                           1.00        30
         macro avg       1.00      1.00      1.00        30
      weighted avg       1.00      1.00      1.00        30
```

[23]:
```
from sklearn.ensemble import AdaBoostClassifier

ada_boost = AdaBoostClassifier()
ada_boost.fit(X_train, y_train)
y_pred = ada_boost.predict(X_test)
print("Original y_test values are ",y_test)
```

```
print("predicted y_test values are",y_pred)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')


conf_matrix = confusion_matrix(y_test, y_pred)
# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

class_report = classification_report(y_test, y_pred, target_names=labels)

# Print the classification report with labels
print('Classification Report:')
print(class_report)
```

```
Original y_test values are  [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
predicted y_test values are [2 1 0 2 0 2 0 1 1 1 1 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
Accuracy: 0.97
Precision: 0.97
Recall: 0.97
F1 Score: 0.97
```

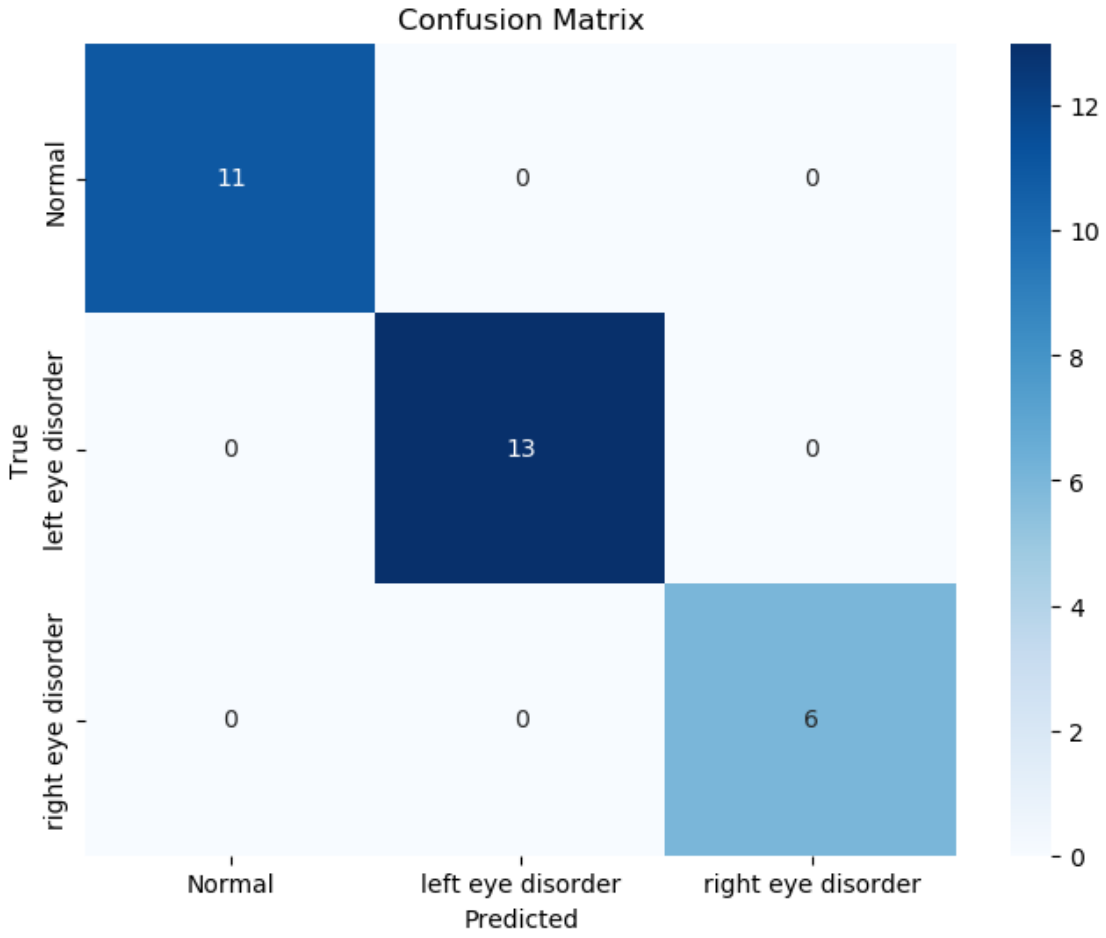



```
Classification Report:
                    precision    recall  f1-score   support

            Normal       1.00      1.00      1.00        11
 left eye disorder       0.93      1.00      0.96        13
right eye disorder       1.00      0.83      0.91         6

          accuracy                           0.97        30
         macro avg       0.98      0.94      0.96        30
      weighted avg       0.97      0.97      0.97        30
```

[24]:
```
from sklearn.ensemble import ExtraTreesClassifier

extra_tree = ExtraTreesClassifier()
extra_tree.fit(X_train, y_train)
y_pred = extra_tree.predict(X_test)
print("Original y_test values are ",y_test)
```

```
print("predicted y_test values are",y_pred)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')


conf_matrix = confusion_matrix(y_test, y_pred)
# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

class_report = classification_report(y_test, y_pred, target_names=labels)

# Print the classification report with labels
print('Classification Report:')
print(class_report)
```

```
Original y_test values are  [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
predicted y_test values are [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0
0 1 1 0]
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```




```
Classification Report:
                    precision    recall  f1-score   support

            Normal       1.00      1.00      1.00        11
 left eye disorder       1.00      1.00      1.00        13
right eye disorder       1.00      1.00      1.00         6

          accuracy                           1.00        30
         macro avg       1.00      1.00      1.00        30
      weighted avg       1.00      1.00      1.00        30
```

[25]:
```python
# all classifiers avilable in SKlearn
# Linear Models
from sklearn.linear_model import LogisticRegression

# Support Vector Machines
from sklearn.svm import SVC
```

```
# Decision Trees
from sklearn.tree import DecisionTreeClassifier

# Ensemble Methods
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier,␣
↪GradientBoostingClassifier, AdaBoostClassifier, ExtraTreesClassifier

# Naive Bayes
from sklearn.naive_bayes import GaussianNB, MultinomialNB, ComplementNB,␣
↪BernoulliNB

# Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier
```

[26]:
```
# all regressors avilable in SKlearn

# Linear Models
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet

# Support Vector Machines
from sklearn.svm import SVR

# Decision Trees
from sklearn.tree import DecisionTreeRegressor

# Ensemble Methods
from sklearn.ensemble import RandomForestRegressor, BaggingRegressor,␣
↪GradientBoostingRegressor, AdaBoostRegressor, ExtraTreesRegressor

# Nearest Neighbors
from sklearn.neighbors import KNeighborsRegressor
```

[27]:
```
# all unsuprvised machine learning models avilable in SKlearn

# Clustering
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering

# Support Vector Machines for Outliers Detection
from sklearn.svm import OneClassSVM
```

[28]:
```
# all feature reduction models avilable in SKlearn

# Principal Component Analysis (PCA)
from sklearn.decomposition import PCA

# Dimensionality Reduction
```

```
from sklearn.decomposition import TruncatedSVD
```