In [ ]:

```
```

In [1]:

```
import numpy as np
import pandas as pd
import os
```

In [2]:

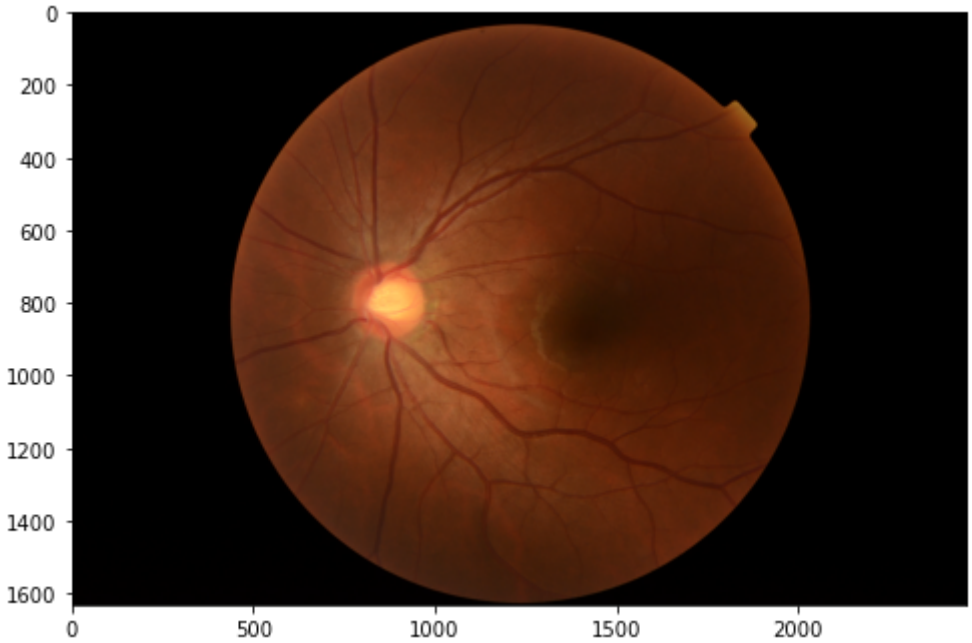
```
import cv2 as cv
import matplotlib.pyplot as plt
from skimage.feature import greycomatrix, greycoprops
```

In [3]:

```
# Method to show image
def show_image(img, cmap='gray'):
    fig = plt.figure(figsize=(8,8))
    axes = fig.add_subplot(111)
    axes.imshow(img, cmap=cmap)
```

In [4]:

```
#Just taken one image vor visualization
path = r'C:\Users\kshit\Desktop\test\Major Project\Dataset\1_normal\NL_001.png'
test_img = cv.imread(path)
test_img = cv.cvtColor(test_img, cv.COLOR_BGR2RGB)
show_image(test_img)
```



In [5]:

```
#RGB image have 3 dimensions
width, height, dimension = test_img.shape
print(f'Width RGB = {width}')
print(f'Height RGB = {height}')
print(f'Dimension RGB = {dimension}')
```

```
Width RGB = 1632
Height RGB = 2464
Dimension RGB = 3
```

In [6]:

```
test_img_gray = cv.cvtColor(test_img, cv.COLOR_RGB2GRAY)
show_image(test_img_gray)
```

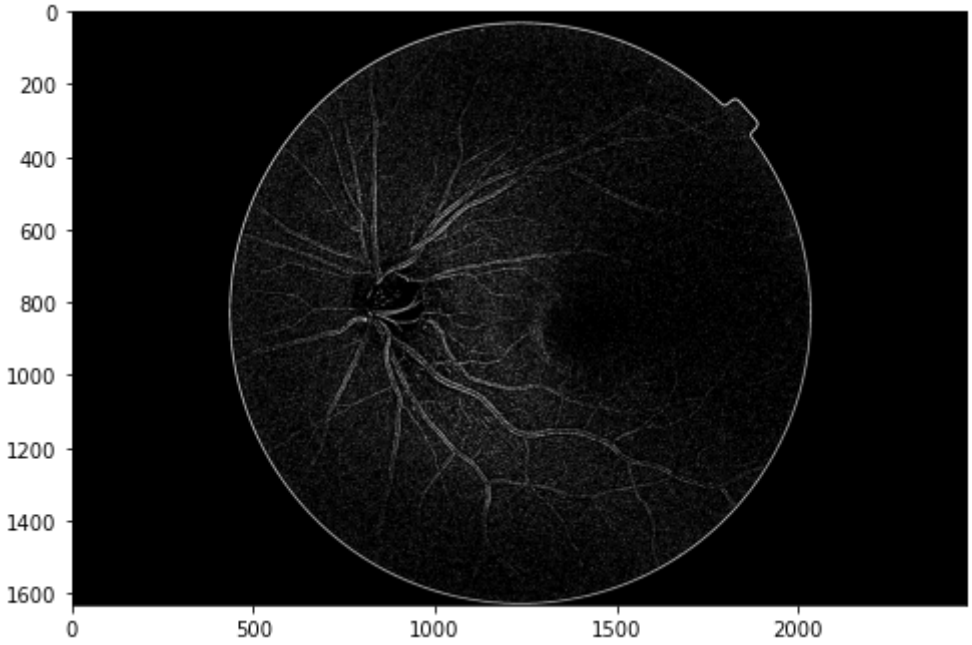
In [7]:

```
#grayscale image doesn't have dimenssion because it have one colour ranges from 0-255
width, height = test_img_gray.shape
print(f'Width Grayscale = {width}')
print(f'Height Grayscale = {height}')
print(f'Image Shape Grayscale {test_img_gray.shape}')
```

```
Width Grayscale = 1632
Height Grayscale = 2464
Image Shape Grayscale (1632, 2464)
```

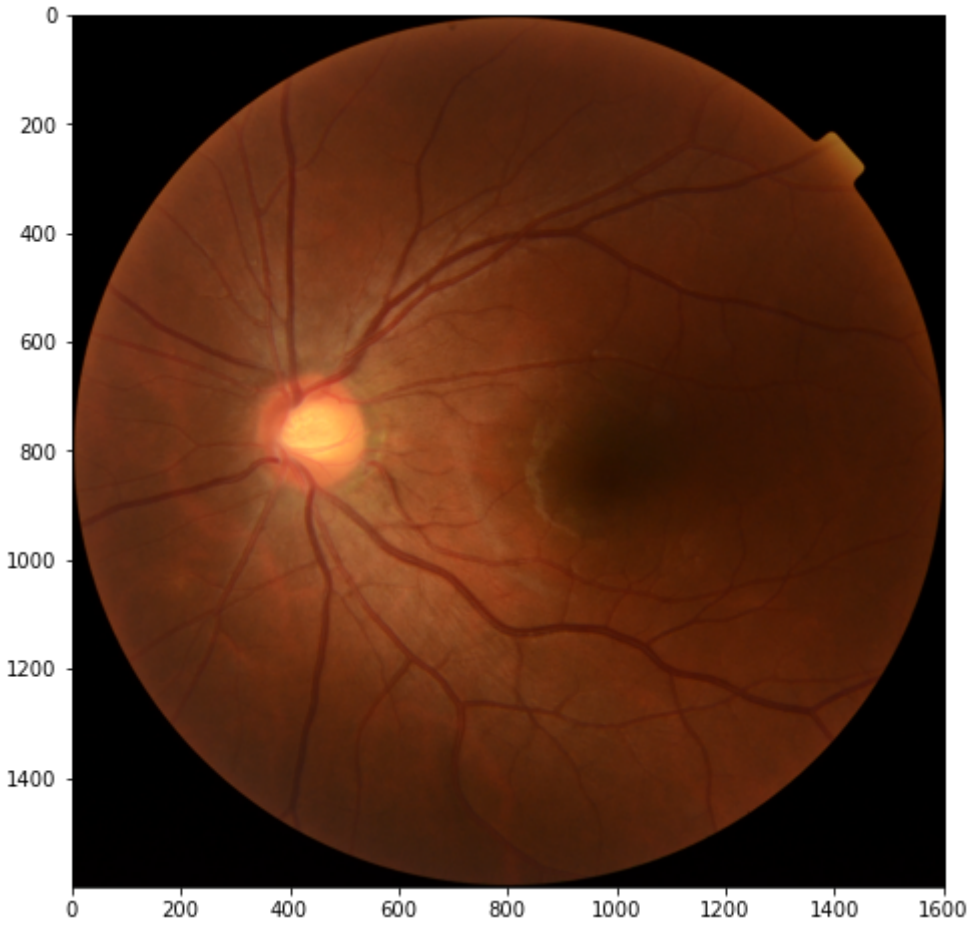
In [8]:

```
#Binary image it have range from 0-1
test_img_thresh = cv.adaptiveThreshold(test_img_gray,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.
show_image(test_img_thresh)
```



In [9]:

```
#cropping image
cnts = cv.findContours(test_img_thresh, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]
cnts = sorted(cnts, key=cv.contourArea, reverse=True)
for c in cnts:
    x,y,w,h = cv.boundingRect(c)
    test_img_ROI = test_img[y:y+h, x:x+w]
    break
show_image(test_img_ROI)
```



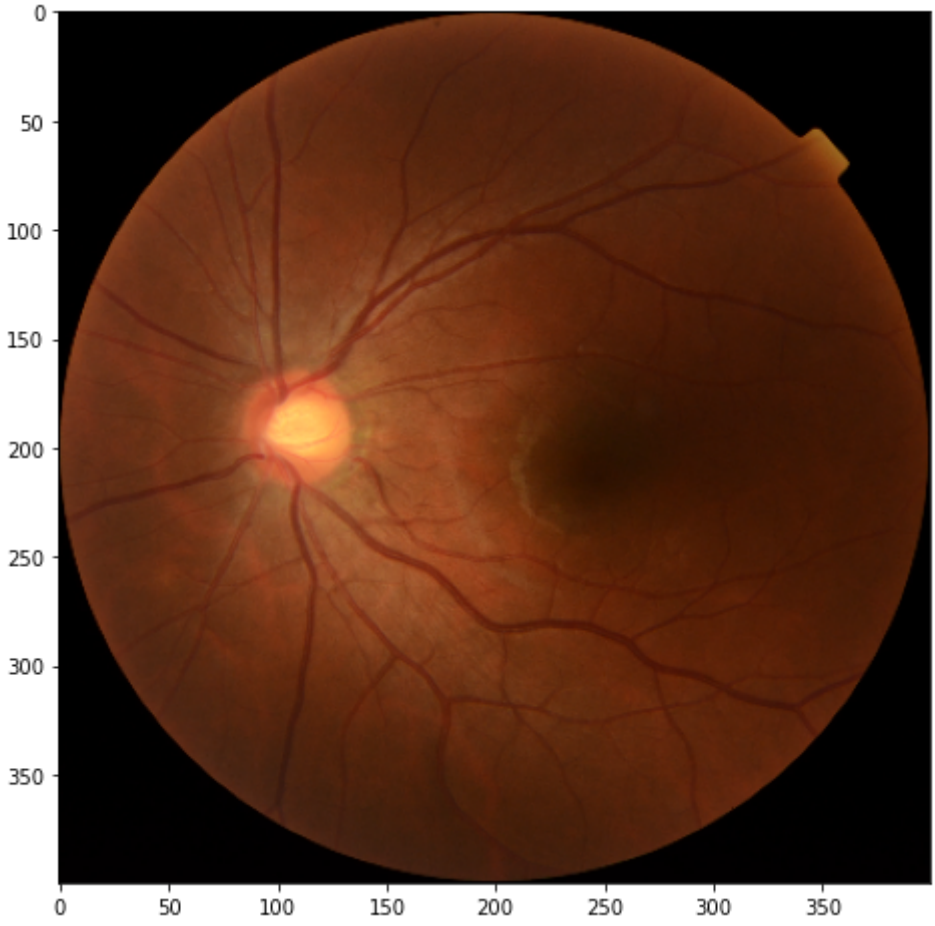
In [10]:

```
width, height, dimension = test_img_ROI.shape
print(f'Width = {width}')
print(f'Height = {height}')
print(f'Dimension = {dimension}')
```

```
Width = 1600
Height = 1601
Dimension = 3
```

In [11]:

```
#reducing dimensions because of costly computation
test_img_ROI_resize = cv.resize(test_img_ROI, (int(width/4), int(height/4)))
show_image(test_img_ROI_resize)
```



In [12]:

```
width, height, dimension = test_img_ROI_resize.shape
print(f'Width = {width}')
print(f'Height = {height}')
print(f'Dimension = {dimension}')
```

```
Width = 400
Height = 400
Dimension = 3
```

In [13]:

```
#function to perform all the preprocessing
def preprocessingImage(image):
    test_img = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    test_img_gray = cv.cvtColor(test_img, cv.COLOR_RGB2GRAY)
    test_img_thresh = cv.adaptiveThreshold(test_img_gray,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,
    cnts = cv.findContours(test_img_thresh, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
    cnts = cnts[0] if len(cnts) == 2 else cnts[1]
    cnts = sorted(cnts, key=cv.contourArea, reverse=True)
    for c in cnts:
        x,y,w,h = cv.boundingRect(c)
        test_img_ROI = test_img[y:y+h, x:x+w]
        break
    test_img_ROI_resize = cv.resize(test_img_ROI, (width, height))
    test_img_ROI_resize_gray = cv.cvtColor(test_img_ROI_resize, cv.COLOR_RGB2GRAY)

    return test_img_ROI_resize_gray
```

In [14]:

```
import pathlib
```

In [15]:

```
data_dir='C:/Users/kshit/Desktop/test/Major Project/Dataset'
data_dir = pathlib.Path(data_dir)
images_count = len(list(data_dir.glob('*/*.png')))
print(images_count)
```

```
600
```

In [16]:

```
#we have 600 images
```

In [17]:

```
images_dict = {
    'normal':list(data_dir.glob('1_normal/*')),
    'cataract':list(data_dir.glob('2_cataract/*'))
}
labels_dict ={
    'normal':0,
    'cataract':1,
}
```

In [18]:

```
str(images_dict['normal'][0])
```

Out[18]:

```
'C:\\Users\\kshit\\Desktop\\test\\Major Project\\Dataset\\1_normal\\NL_001.p
ng'
```

In [19]:

```
X,y = [],[]
for typ_dis,images in images_dict.items(): #typ_dia name gives tittle and images is a list
    for image in images:      #image is the selection of a perticular image from all image
        img = cv.imread(str(image))
        new_img = preprocessingImage(img)
        X.append(new_img)
        y.append(labels_dict[typ_dis])
```

In [20]:

```
X = np.array(X)
y = np.array(y)
```

In [21]:

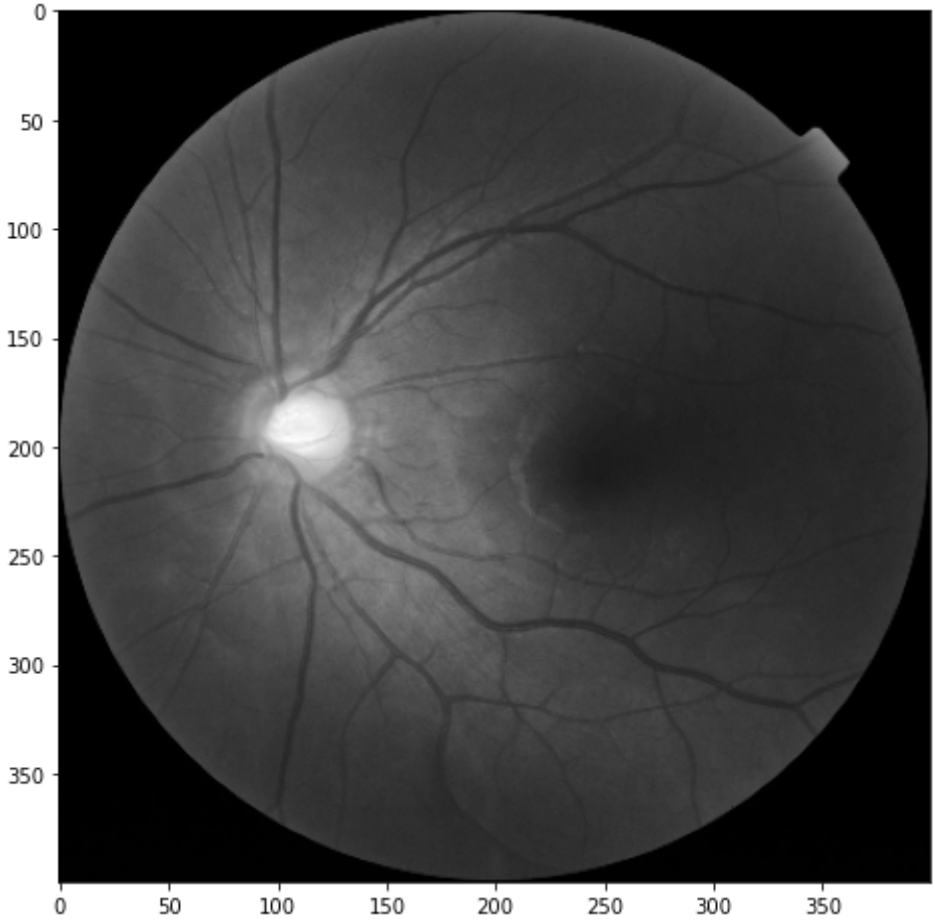
```
(len(X),len(y),X[0])
```

Out[21]:

```
(600,
 600,
 array([[0, 0, 0, ..., 0, 0, 1],
        [0, 1, 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]], dtype=uint8))
```

In [22]:

```
show_image(X[0])
```



In [23]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=2,test_size=0.2)
```

In [24]:

```
print(len(X_train),len(y_train),len(X_test),len(y_test))
```

```
480 480 120 120
```

In [25]:

```
y_train[:100]
```

Out[25]:

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

In [26]:

```
X_train[0]
```

Out[26]:

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 1, 1, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 1, 0],
       [0, 1, 0, ..., 0, 1, 1]], dtype=uint8)
```

In [27]:

```
X_train.shape
```

Out[27]:

```
(480, 400, 400)
```

In [28]:

```
# Machine Learning models Expect 2D data set
nsamples, nx, ny = X_train.shape
X_train2 = X_train.reshape((nsamples,nx*ny))
nsamples, nx, ny = X_test.shape
X_test2 = X_test.reshape((nsamples,nx*ny))
```

In [29]:

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
import seaborn as sns
```

In [42]:

```
def graphplot(y_test, y_pred):
    mat = confusion_matrix(y_test, y_pred)
    sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
                xticklabels=['Normal','Cataract'],
                yticklabels=['Normal','Cataract'])
    plt.xlabel('true label')
    plt.ylabel('predicted label')
    plt.figure(figsize=(50, 50));
```

Machine Learning Models

1. Support Vector Machine

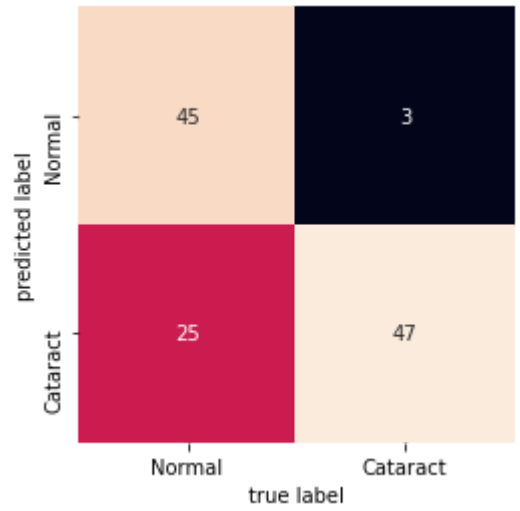
In [43]:

```
#SVM Model
from sklearn.svm import SVC
model_svm = SVC()
model_svm.fit(X_train2, y_train)
model_svm.score(X_test2, y_test)
f1_score(y_test, model_svm.predict(X_test2), average='macro')
y_pred_svm = model_svm.predict(X_test2)
print(classification_report(y_test, y_pred_svm))
graphplot(y_test, y_pred_svm)
```

```
              precision    recall  f1-score   support

           0       0.94      0.64      0.76        70
           1       0.65      0.94      0.77        50

    accuracy                           0.77       120
   macro avg       0.80      0.79      0.77       120
weighted avg       0.82      0.77      0.77       120
```




```
<Figure size 3600x3600 with 0 Axes>
```

2. Random Forest Classifier

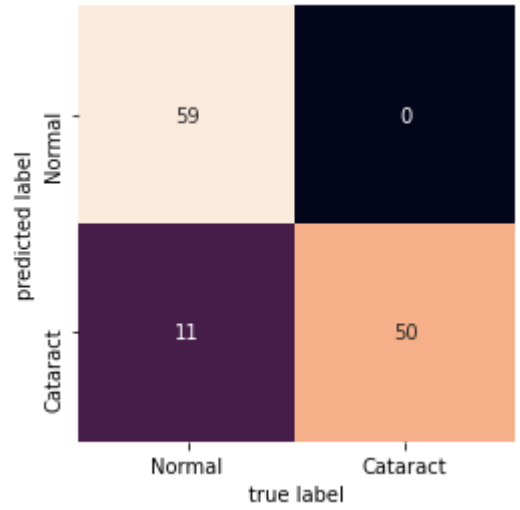
In [44]:

```
#RandomForestClassifier Model
from sklearn.ensemble import RandomForestClassifier
model_rfc = RandomForestClassifier()
model_rfc.fit(X_train2, y_train)
model_rfc.score(X_test2, y_test)
f1_score(y_test, model_rfc.predict(X_test2), average='macro')
y_pred_rfc = model_rfc.predict(X_test2)
print(classification_report(y_test, y_pred_rfc))
graphplot(y_test, y_pred_rfc)
```

```
              precision    recall  f1-score   support

           0       1.00      0.84      0.91        70
           1       0.82      1.00      0.90        50

    accuracy                           0.91       120
   macro avg       0.91      0.92      0.91       120
weighted avg       0.92      0.91      0.91       120
```




```
<Figure size 3600x3600 with 0 Axes>
```

3. Logistic Regression

In [46]:

```
#LogisticRegression Model
from sklearn.linear_model import LogisticRegression
model_lr = LogisticRegression(solver='lbfgs', max_iter=10)
model_lr.fit(X_train2, y_train)
model_lr.score(X_test2, y_test)
f1_score(y_test, model_lr.predict(X_test2), average='macro')
y_pred_lr = model_lr.predict(X_test2)
print(classification_report(y_test, y_pred_lr))
graphplot(y_test, y_pred_lr)
```

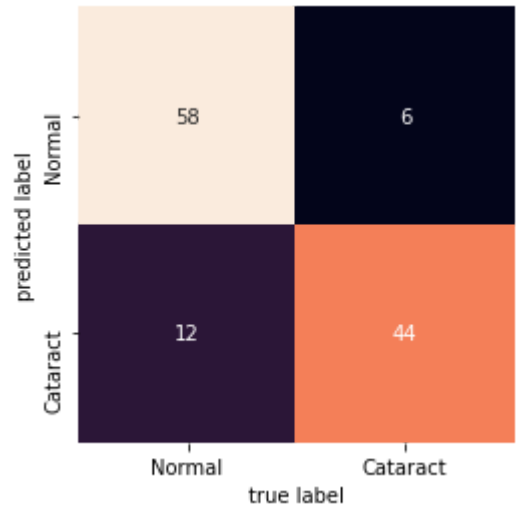
```
C:\Users\kshit\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(

              precision    recall  f1-score   support

           0       0.91      0.83      0.87        70
           1       0.79      0.88      0.83        50

    accuracy                           0.85       120
   macro avg       0.85      0.85      0.85       120
weighted avg       0.86      0.85      0.85       120
```




```
<Figure size 3600x3600 with 0 Axes>
```

4. k-Nearest Neighbors

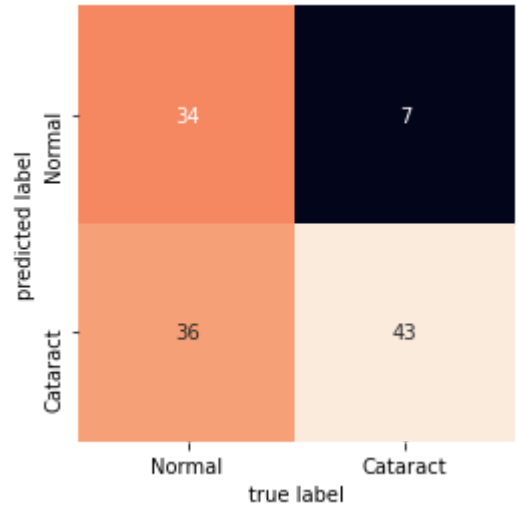
In [47]:

```
#KNeighborsClassifier Model
from sklearn.neighbors import KNeighborsClassifier
model_kn = KNeighborsClassifier()
model_kn.fit(X_train2, y_train)
model_kn.score(X_test2, y_test)
f1_score(y_test, model_kn.predict(X_test2), average='macro')
y_pred_kn = model_kn.predict(X_test2)
print(classification_report(y_test, y_pred_kn))
graphplot(y_test, y_pred_kn)
```

```
              precision    recall  f1-score   support

           0       0.83      0.49      0.61        70
           1       0.54      0.86      0.67        50

    accuracy                           0.64       120
   macro avg       0.69      0.67      0.64       120
weighted avg       0.71      0.64      0.64       120
```




```
<Figure size 3600x3600 with 0 Axes>
```

5. Naive Bayes

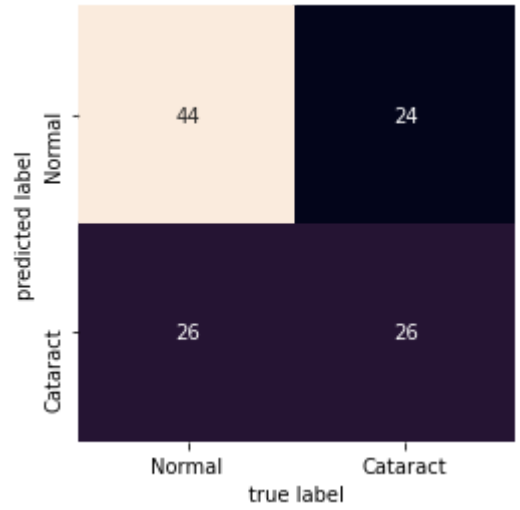
In [48]:

```
#NaiveBayes
from sklearn.naive_bayes import GaussianNB
model_nb = GaussianNB()
model_nb.fit(X_train2, y_train)
model_nb.score(X_test2, y_test)
f1_score(y_test, model_nb.predict(X_test2), average='macro')
y_pred_nb = model_nb.predict(X_test2)
print(classification_report(y_test, y_pred_nb))
graphplot(y_test, y_pred_nb)
```

```
              precision    recall  f1-score   support

           0       0.65      0.63      0.64        70
           1       0.50      0.52      0.51        50

    accuracy                           0.58       120
   macro avg       0.57      0.57      0.57       120
weighted avg       0.59      0.58      0.58       120
```




```
<Figure size 3600x3600 with 0 Axes>
```

DEEP LEARNING MODELS

In [54]:

```
X_train.shape
```

Out[54]:

```
(480, 400, 400)
```

In [55]:

```
import tensorflow as tf
```

1. Neural Network with 0 Hidden Layer

In [56]:

```
#Artificial Neural network with 0 Hidden Layer
def checkbestfit(x):
    model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape=(400,400)),
        tf.keras.layers.Dense(x, activation='relu'),
        tf.keras.layers.Dense(1,activation='sigmoid')
    ])
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    model.fit(X_train, y_train, epochs=10)

    yp = model.predict(X_test)
    print(max(yp),min(yp))

    y_pred = []
    for i in yp:
        if i>0.5:
            y_pred.append(1)
        else:
            y_pred.append(0)

    return model.evaluate(X_test,  y_test),classification_report(y_test,y_pred)
```

In [57]:

```
res = {}
low_lim = 100
upp_lim = 1000
for i in range(low_lim,upp_lim,100):
    x,y = checkbestfit(i)
    res[i] = [x,y]
```

```
Epoch 1/10
15/15 [==============================] - 3s 174ms/step - loss: 9553.0023 -
accuracy: 0.4931
Epoch 2/10
15/15 [==============================] - 2s 163ms/step - loss: 4019.0624 -
accuracy: 0.4593
Epoch 3/10
15/15 [==============================] - 2s 148ms/step - loss: 2261.9377 -
accuracy: 0.5006
Epoch 4/10
15/15 [==============================] - 2s 164ms/step - loss: 3411.6385 -
accuracy: 0.5565
Epoch 5/10
15/15 [==============================] - 2s 154ms/step - loss: 1972.3196 -
accuracy: 0.5273
Epoch 6/10
15/15 [==============================] - 2s 148ms/step - loss: 971.3341 -
accuracy: 0.6580
Epoch 7/10
```

In [59]:

```
m = 0
print("Accuracy report of models")
print("Model    1st_Layer   Accuracy")
for i in range(low_lim,upp_lim,100):
    print("  ",int(i/100),"      ",i,"      ",(res[i][0][1]*100))
    if(m<(res[i][0][1]*100)):
        m = (res[i][0][1]*100)
        ind = i
print("\n\n\nClassifiation Report of",int(ind/100),"Model Because of its maximum Accuracy")
print(res[ind][1])
```

```
Accuracy report of models
Model    1st_Layer   Accuracy
   1        100        67.5000011920929
   2        200        43.33333373069763
   3        300        85.00000238418579
   4        400        60.83333492279053
   5        500        66.66666865348816
   6        600        64.99999761581421
   7        700        82.4999988079071
   8        800        58.33333134651184
   9        900        59.16666388511658



Classifiation Report of 3 Model Because of its maximum Accuracy
              precision    recall  f1-score   support

           0       0.83      0.93      0.88        70
           1       0.88      0.74      0.80        50

    accuracy                           0.85       120
   macro avg       0.86      0.83      0.84       120
weighted avg       0.85      0.85      0.85       120
```

2. Neural Network with 1 Hidden Layer

In [68]:

```
#Artificial Neural Network model
def checkbestfit2(x):
    model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape=(400,400)),
        tf.keras.layers.Dense(x, activation='relu'),
        tf.keras.layers.Dense(2*x, activation='relu'),
        tf.keras.layers.Dense(1,activation='sigmoid')
    ])
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    model.fit(X_train, y_train, epochs=10)

    yp = model.predict(X_test)
    print(max(yp),min(yp))

    y_pred = []
    for i in yp:
        if i>0.5:
            y_pred.append(1)
        else:
            y_pred.append(0)

    return model.evaluate(X_test,  y_test),classification_report(y_test,y_pred)
```

In [69]:

```
res = {}
low_lim = 100
upp_lim = 1000
for i in range(low_lim,upp_lim,100):
    x,y = checkbestfit2(i)
    res[i] = [x,y]
```

```
15/15 [==============================] - 4s 260ms/step - loss: 108.2170 -
accuracy: 0.5927
Epoch 9/10
15/15 [==============================] - 4s 254ms/step - loss: 263.4486 -
accuracy: 0.5058
Epoch 10/10
15/15 [==============================] - 4s 242ms/step - loss: 127.2958 -
accuracy: 0.5041
[1.] [0.]
4/4 [==============================] - 0s 52ms/step - loss: 55.8461 - accu
racy: 0.5000
Epoch 1/10
15/15 [==============================] - 7s 423ms/step - loss: 7021.4861 -
accuracy: 0.4525
Epoch 2/10
15/15 [==============================] - 6s 416ms/step - loss: 2049.8285 -
accuracy: 0.5092
Epoch 3/10
15/15 [==============================] - 6s 388ms/step - loss: 992.1068 -
accuracy: 0.4906
```

In [70]:

```
m = 0
print("Accuracy report of models")
print("Model    1st_Layer   Hidden    Accuracy")
for i in range(low_lim,upp_lim,100):
    print("  ",int(i/100),"      ",i,"    ",i*2,"      ",(res[i][0][1]*100))
    if(m<(res[i][0][1]*100)):
        m = (res[i][0][1]*100)
        ind = i
print("\n\n\nClassifiation Report of",int(ind/100),"Model Because of its maximum Accuracy")
print(res[ind][1])
```

```
Accuracy report of models
Model    1st_Layer   Hidden    Accuracy
   1        100     200        68.33333373069763
   2        200     400        50.0
   3        300     600        72.50000238418579
   4        400     800        41.66666567325592
   5        500     1000        55.83333373069763
   6        600     1200        57.499998807907104
   7        700     1400        70.83333134651184
   8        800     1600        42.500001192092896
   9        900     1800        44.16666626930237


Classifiation Report of 3 Model Because of its maximum Accuracy
              precision    recall  f1-score   support

           0       0.97      0.54      0.70        70
           1       0.60      0.98      0.75        50

    accuracy                           0.73       120
   macro avg       0.79      0.76      0.72       120
weighted avg       0.82      0.72      0.72       120
```

4. Neural Network with 2 Hidden Layer

In [71]:

```
#Artificial Neural Network model
def checkbestfit3(x):
    model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape=(400,400)),
        tf.keras.layers.Dense(x, activation='relu'),
        tf.keras.layers.Dense(2*x, activation='relu'),
        tf.keras.layers.Dense(3*x, activation='relu'),
        tf.keras.layers.Dense(1,activation='sigmoid')
    ])
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    model.fit(X_train, y_train, epochs=10)

    yp = model.predict(X_test)
    print(max(yp),min(yp))

    y_pred = []
    for i in yp:
        if i>0.5:
            y_pred.append(1)
        else:
            y_pred.append(0)

    return model.evaluate(X_test,  y_test),classification_report(y_test,y_pred)
```

In [72]:

```
res = {}
low_lim = 100
upp_lim = 1000
for i in range(low_lim,upp_lim,100):
    x,y = checkbestfit3(i)
    res[i] = [x,y]
```

```
Epoch 1/10
15/15 [==============================] - 3s 146ms/step - loss: 2062.4217 -
accuracy: 0.5071
Epoch 2/10
15/15 [==============================] - 2s 150ms/step - loss: 584.6989 -
accuracy: 0.5084
Epoch 3/10
15/15 [==============================] - 2s 149ms/step - loss: 498.2990 -
accuracy: 0.5608
Epoch 4/10
15/15 [==============================] - 2s 153ms/step - loss: 583.3715 -
accuracy: 0.4889
Epoch 5/10
15/15 [==============================] - 2s 155ms/step - loss: 522.3430 -
accuracy: 0.4480
Epoch 6/10
15/15 [==============================] - 2s 160ms/step - loss: 30.5285 - a
ccuracy: 0.5249
Epoch 7/10
```

In [73]:

```
m = 0
print("Accuracy report of models")
print("Model    1st_Layer   Hidden    2nd Hidden     Accuracy")
for i in range(low_lim,upp_lim,100):
    print("  ",int(i/100),"      ",i,"    ",i*2,"     ",i*3,"        ",(res[i][0][1]*100))
    if(m<(res[i][0][1]*100)):
        m = (res[i][0][1]*100)
        ind = i
print("\n\n\nClassifiation Report of",int(ind/100),"Model Because of its maximum Accuracy")
print(res[ind][1])
```

```
Accuracy report of models
Model    1st_Layer   Hidden    2nd Hidden     Accuracy
   1        100     200       300          41.66666567325592
   2        200     400       600          41.66666567325592
   3        300     600       900          41.66666567325592
   4        400     800       1200          41.66666567325592
   5        500     1000       1500          41.66666567325592
   6        600     1200       1800          44.16666626930237
   7        700     1400       2100          41.66666567325592
   8        800     1600       2400          42.500001192092896
   9        900     1800       2700          43.33333373069763



Classifiation Report of 6 Model Because of its maximum Accuracy
              precision    recall  f1-score   support

           0       1.00      0.04      0.08        70
           1       0.43      1.00      0.60        50

    accuracy                           0.44       120
   macro avg       0.71      0.52      0.34       120
weighted avg       0.76      0.44      0.30       120
```

3. Neural Network with 1 Hidden Layer

In [74]:

```
#Artificial Neural Network model
def checkbestfit4(x):
    model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape=(400,400)),
        tf.keras.layers.Dense(x, activation='relu'),
        tf.keras.layers.Dense(x/2, activation='relu'),
        tf.keras.layers.Dense(1,activation='sigmoid')
    ])
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    model.fit(X_train, y_train, epochs=10)

    yp = model.predict(X_test)
    print(max(yp),min(yp))

    y_pred = []
    for i in yp:
        if i>0.5:
            y_pred.append(1)
        else:
            y_pred.append(0)

    return model.evaluate(X_test,  y_test),classification_report(y_test,y_pred)
```

In [75]:

```
res = {}
low_lim = 100
upp_lim = 1000
for i in range(low_lim,upp_lim,100):
    x,y = checkbestfit4(i)
    res[i] = [x,y]
```

```
Epoch 1/10
15/15 [==============================] - 3s 150ms/step - loss: 3734.3485 -
accuracy: 0.4683
Epoch 2/10
15/15 [==============================] - 2s 151ms/step - loss: 782.7162 -
accuracy: 0.5276
Epoch 3/10
15/15 [==============================] - 2s 151ms/step - loss: 463.0427 -
accuracy: 0.5117
Epoch 4/10
15/15 [==============================] - 2s 160ms/step - loss: 219.4639 -
accuracy: 0.5506
Epoch 5/10
15/15 [==============================] - 2s 164ms/step - loss: 123.6176 -
accuracy: 0.5785
Epoch 6/10
15/15 [==============================] - 2s 166ms/step - loss: 34.6783 - a
ccuracy: 0.6959
Epoch 7/10
```

In [76]:

```
m = 0
print("Accuracy report of models")
print("Model    1st_Layer   Hidden     Accuracy")
for i in range(low_lim,upp_lim,100):
    print("  ",int(i/100),"      ",i,"   ",i/2,"        ",(res[i][0][1]*100))
    if(m<(res[i][0][1]*100)):
        m = (res[i][0][1]*100)
        ind = i
print("\n\n\nClassifiation Report of",int(ind/100),"Model Because of its maximum Accuracy")
print(res[ind][1])
```

```
Accuracy report of models
Model    1st_Layer   Hidden     Accuracy
   1        100     50.0          80.83333373069763
   2        200     100.0          64.99999761581421
   3        300     150.0          80.0000011920929
   4        400     200.0          58.33333134651184
   5        500     250.0          80.0000011920929
   6        600     300.0          77.49999761581421
   7        700     350.0          58.33333134651184
   8        800     400.0          72.50000238418579
   9        900     450.0          82.4999988079071



Classifiation Report of 9 Model Because of its maximum Accuracy
              precision    recall  f1-score   support

           0       0.87      0.83      0.85        70
           1       0.77      0.82      0.80        50

    accuracy                           0.82       120
   macro avg       0.82      0.82      0.82       120
weighted avg       0.83      0.82      0.83       120
```

In [ ]: