cs188_project2-Blank

February 13, 2020

1 CS188 Project 2 - Binary Classification Comparative Methods

For this project we're going to attempt a binary classification of a dataset using multiple methods and compare results.

Our goals for this project will be to introduce you to several of the most common classification techniques, how to perform them and tweek parameters to optimize outcomes, how to produce and interpret results, and compare performance. You will be asked to analyze your findings and provide explanations for observed performance.

Specifically you will be asked to classify whether a patient is suffering from heart disease based on a host of potential medical factors.

DEFINITIONS

Binary Classification: In this case a complex dataset has an added 'target' label with one of two options. Your learning algorithm will try to assign one of these labels to the data.

Supervised Learning: This data is fully supervised, which means it's been fully labeled and we can trust the veracity of the labeling.

1.1 Background: The Dataset

2 For this exercise we will be using a subset of the UCI Heart Disease dataset, leveraging the fourteen most commonly used attributes. All identifying information about the patient has been scrubbed.

The dataset includes 14 columns. The information provided by each column is as follows:

age: Age in years

sex: (1 = male; 0 = female)

cp: Chest pain type (0 = asymptomatic; 1 = atypical angina; 2 = non-anginal pain; 3 = typical angina)

trestbps: Resting blood pressure (in mm Hg on admission to the hospital)

cholserum: Cholestoral in mg/dl

fbs Fasting blood sugar > 120 mg/dl (1 = true; 0 = false)

restecg: Resting electrocardiographic results (0= showing probable or definite left ventricular hypertrophy by Estes' criteria; 1 = normal; 2 = having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV))

thalach: Maximum heart rate achieved

exang: Exercise induced angina (1 = yes; 0 = no)

oldpeakST: Depression induced by exercise relative to rest

slope: The slope of the peak exercise ST segment (0 = downsloping; 1 = flat; 2 = upsloping)

ca: Number of major vessels (0-3) colored by flourosopy

thal: 1 = normal; 2 = fixed defect; 7 = reversable defect

Sick: Indicates the presence of Heart disease (True = Disease; False = No disease)

2.1 Loading Essentials and Helper Functions

[1]:
```
#Here are a set of libraries we imported to complete this assignment.
#Feel free to use these or equivalent libraries for your implementation
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # this is used for the plot the graph
import os
import seaborn as sns # used for plot interactive graph.
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪GridSearchCV
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
import sklearn.metrics.cluster as smc
from sklearn.model_selection import KFold


from matplotlib import pyplot
import itertools

%matplotlib inline
import random

random.seed(42)
```

[2]:
```
# Helper function allowing you to export a graph
def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(fig_id + "." + fig_extension)
```

```
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

[3]:
```
# Helper function that allows you to draw nicely formatted confusion matrices
def draw_confusion_matrix(y, yhat, classes):
    '''
        Draws a confusion matrix for the given target and predictions
        Adapted from scikit-learn and discussion example.
    '''
    plt.cla()
    plt.clf()
    matrix = confusion_matrix(y, yhat)
    plt.imshow(matrix, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.colorbar()
    num_classes = len(classes)
    plt.xticks(np.arange(num_classes), classes, rotation=90)
    plt.yticks(np.arange(num_classes), classes)

    fmt = 'd'
    thresh = matrix.max() / 2.
    for i, j in itertools.product(range(matrix.shape[0]), range(matrix.
↪shape[1])):
        plt.text(j, i, format(matrix[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if matrix[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.show()
```

2.2 [20 Points] Part 1. Load the Data and Analyze

Let's first load our dataset so we'll be able to work with it. (correct the relative path if your notebook is in a different directory than the csv file.)

[4]:
```
#Things I just created for shortcuts
ROOT_DIR = "."
```

[5]:
```
def load_heartdisease_data(heartdisease_path):
        DATASET_PATH = os.path.join(heartdisease_path, "heartdisease.csv")
        return pd.read_csv(DATASET_PATH)
```

2.2.1 Question 1.1 Now that our data is loaded, let's take a closer look at the dataset we're working with. Use the head method to display some of the rows so we can visualize the types of data fields we'll be working with, then use the describe method, along with any additional methods you'd like to call to better help you understand what you're working with and what issues you might face.

[6]:
```
heartdisease = load_heartdisease_data(ROOT_DIR)
heartdisease.head() #head just showed the first few elements of dataframe.
```

[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	sick
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	False
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	False
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	False
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	False
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	False

[7]:
```
heartdisease.describe()
```

[7]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000

```
mean     2.313531
std      0.612277
min      0.000000
25%      2.000000
50%      2.000000
75%      3.000000
max      3.000000
```

[8]:
```
heartdisease.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
age         303 non-null int64
sex         303 non-null int64
cp          303 non-null int64
trestbps    303 non-null int64
chol        303 non-null int64
fbs         303 non-null int64
restecg     303 non-null int64
thalach     303 non-null int64
exang       303 non-null int64
oldpeak     303 non-null float64
slope       303 non-null int64
ca          303 non-null int64
thal        303 non-null int64
sick        303 non-null bool
dtypes: bool(1), float64(1), int64(12)
memory usage: 31.2 KB
```

2.2.2 Question 1.2 Discuss your data preprocessing strategy. Are their any datafield types that are problemmatic and why? Will there be any null values you will have to impute and how do you intend to do so? Finally, for your numeric and categorical features, what if any, additional preprocessing steps will you take on those data elements?

For my data processing strategy, I ended up using three different functions named head(), describe() and info(). Head() showed me only the couple first few elements to familiarize me with the different types of data that I was going to use. Describe() showed me more of the total amount of data that I was going to be working with. Info() ended up showing me that there were no data values that I would have to impute because they were all non-null. The only problematic data field type were the booleans being used in the sick datafield. This was problamatic because our model can not process boolean data types. For numeric and categorial features, we will need to do data cleaning and data reduction to have our data fit our model.

2.2.3 Question 1.3 Before we begin our analysis we need to fix the field(s) that will be problematic. Specifically convert our boolean sick variable into a binary numeric target variable (values of either '0' or '1'), and then drop the original sick datafield from the dataframe.

[9]:
```
heartdisease['sick'] = heartdisease['sick'].astype(int)
```

[10]:
```
heartdisease.describe()
```

[10]:

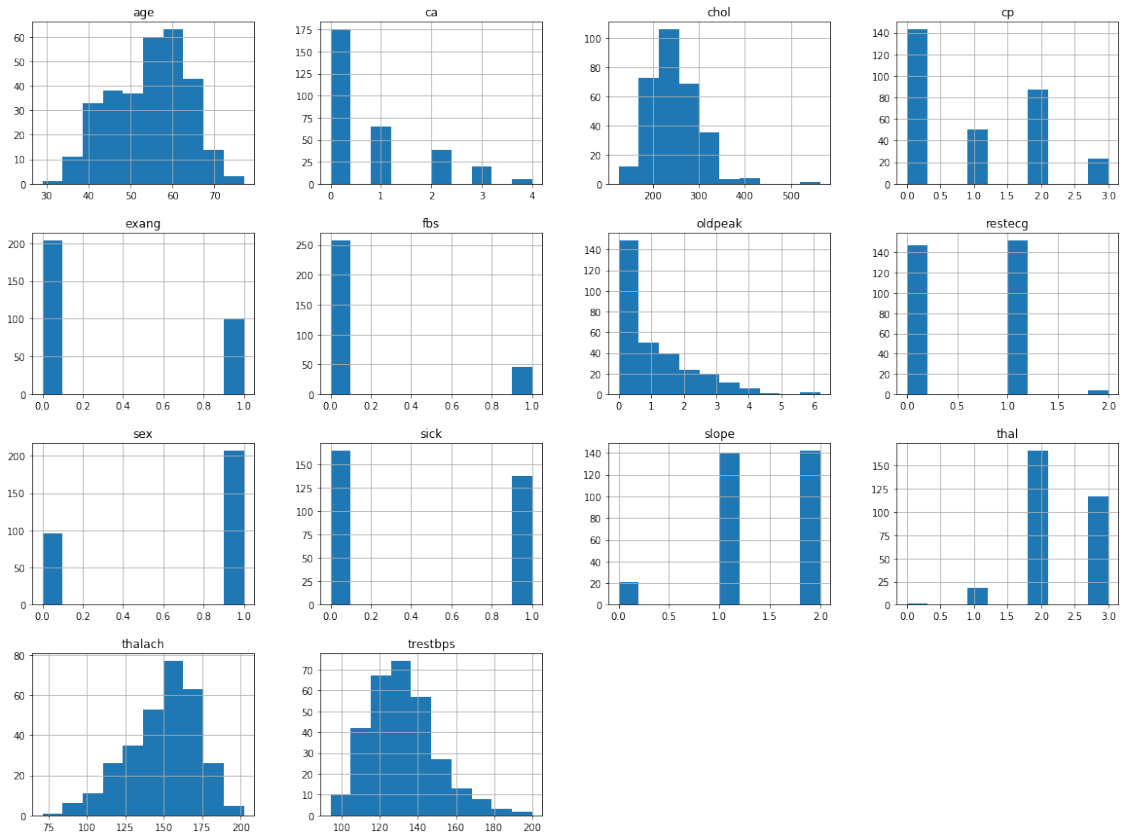
	age	sex	cp	trestbps	chol	fbs	\
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	

	restecg	thalach	exang	oldpeak	slope	ca	\
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	
mean	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	
std	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	
min	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	
50%	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	
75%	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	
max	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	

	thal	sick
count	303.000000	303.000000
mean	2.313531	0.455446
std	0.612277	0.498835
min	0.000000	0.000000
25%	2.000000	0.000000
50%	2.000000	0.000000
75%	3.000000	1.000000
max	3.000000	1.000000

2.2.4 Question 1.4 Now that we have a feel for the data-types for each of the variables, plot histograms of each field and attempt to ascertain how each variable performs (is it a binary, or limited selection, or does it follow a gradient? (Note: No need to describe each variable, but pick out a few you wish to highlight)

[11]:
```
heartdisease.hist(["chol", "trestbps", "cp", "age", "sex","fbs", "restecg",␣
↪"thalach", "exang", "oldpeak", "slope", "ca" ,"thal" ,"sick"],  figsize =␣
↪(20,15))
plt.show()
```



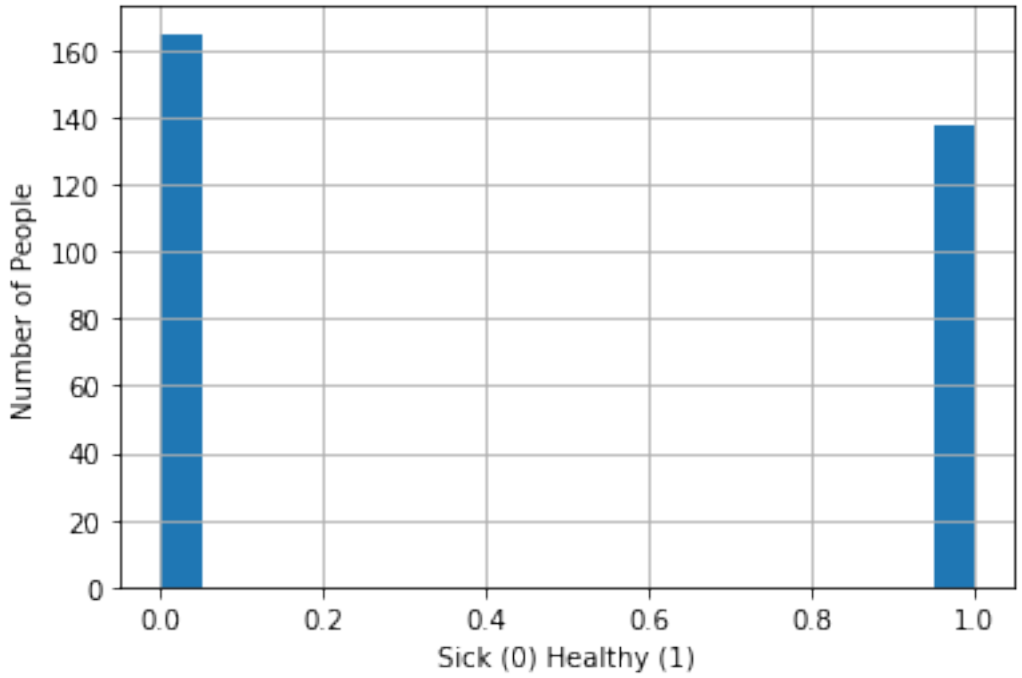
Binary: exang, fbs, sex, sick

Limited Selection: ca, cp, restecg, slope, thal

Gradient: age, chol, oldpeak, thalach, trestbps

2.2.5 Question 1.5 We also want to make sure we are dealing with a balanced dataset. In this case, we want to confirm whether or not we have an equitable number of sick and healthy individuals to ensure that our classifier will have a sufficiently balanced dataset to adequately classify the two. Plot a histogram specifically of the sick target, and conduct a count of the number of sick and healthy individuals and report on the results:

[12]:
```
heartdisease["sick"].hist(bins = 20)
plt.xlabel("Sick (0) Healthy (1)")
plt.ylabel("Number of People")
plt.show()
```



By looking at the bar graph histogram above, we can say that we have an equitable number of sick and healthy individuals. There is almost a one to one ratio between the number of sick people to the number of healthy people which shows that the data is not too biased towards one group.

2.2.6 Question 1.6 Balanced datasets are important to ensure that classifiers train adequately and don't overfit, however arbitrary balancing of a dataset might introduce its own issues. Discuss some of the problems that might arise by artificially balancing a dataset.

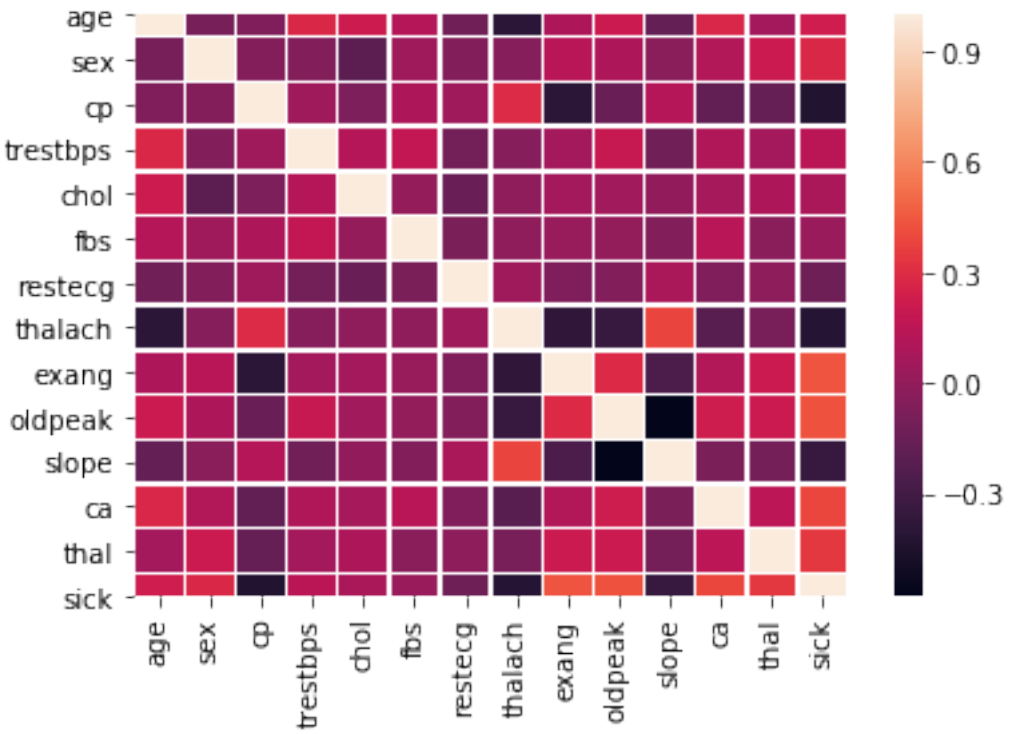
Some of the problems that will arise from artificially balancing a dataset:

-The data can become skewed towards one point depending on the type of category or value that we choose to use to balance the dataset. For example, if you use the median to better fit the our dataset to our model, the dataset would become more biased toward the center. -When we

artificially balance a dataset, the model is not representative of the true data that was originally provided. We will have false representation. -If we end up removing data to artificially balance our dataset, we may end up throwing out data that is crucial to our dataset.

2.2.7 Question 1.9 Now that we have our dataframe prepared let's start analyzing our data. For this next question let's look at the correlations of our variables to our target value. First, map out the correlations between the values, and then discuss the relationships you observe. Do some research on the variables to understand why they may relate to the observed corellations. Intuitively, why do you think some variables correlate more highly than others (hint: one possible approach you can use the sns heatmap function to map the corr() method)?

[13]:
```
sns.heatmap(heartdisease.corr(),
            xticklabels=heartdisease.columns,
            yticklabels=heartdisease.columns,
            linewidth=0.5)
plt.show()
```



A possible reason why some of these variables correlate more highly than others is because there are different variations of heart diseases. If you look at the heat map, two variables such as thalach and slope as well as cp and thalach show that there is a high correlation between these variables and heart disease because a person can

2.3 [30 Points] Part 2. Prepare the Data

Before running our various learning methods, we need to do some additional prep to finalize our data. Specifically you'll have to cut the classification target from the data that will be used to classify, and then you'll have to divide the dataset into training and testing cohorts.

Specifically, we're going to ask you to prepare 2 batches of data: 1. Will simply be the raw numeric data that hasn't gone through any additional pre-processing. The other, will be data that you pipeline using your own selected methods. We will then feed both of these datasets into a classifier to showcase just how important this step can be!

2.3.1 Question 2.1 Save the target column as a separate array and then drop it from the dataframe.

[14]:
```
target = heartdisease['sick']
heartdisease.drop(['sick'], axis=1, inplace = True)
```

2.3.2 Question 2.2 First Create your 'Raw' unprocessed training data by dividing your dataframe into training and testing cohorts, with your training cohort consisting of 70% of your total dataframe (hint: use the train_test_split method) Output the resulting shapes of your training and testing samples to confirm that your split was successful.

[15]:
```
X_train, X_test, y_train, y_test = train_test_split(heartdisease, target,␣
↪test_size = 0.3, random_state = 42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(212, 13)
(91, 13)
(212,)
(91,)
```

2.3.3 Question 2.3 Now create a pipeline to conduct any additional preparation of the data you would like. Output the resulting array to ensure it was processed correctly.

[16]:
```
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.base import BaseEstimator, TransformerMixin
```

```
df_num = heartdisease.
 ↪drop(["sex","cp","fbs","restecg","exang","slope","ca","thal"], axis=1)
num_pipeline = Pipeline([('std_scaler', StandardScaler()),])
#housing_num_tr = num_pipeline.fit_transform(X_train)
numerical_features = list(df_num)
categorical_features = ["sex","cp","fbs","restecg","exang","slope","ca","thal"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])

df_prepared = full_pipeline.fit_transform(heartdisease)
df_prepared
```

```
C:\Users\choie\Anaconda3\lib\site-
packages\sklearn\preprocessing\_encoders.py:415: FutureWarning: The handling of
integer data will change in version 0.22. Currently, the categories are
determined based on the range [0, max(values)], while in the future they will be
determined based on the unique values.
If you want the future behaviour and silence this warning, you can specify
"categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the
categories to integers, then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
```

[16]:
```
array([[ 0.9521966 ,  0.76395577, -0.25633371, …,  1.        ,
         0.        ,  0.        ],
       [-1.91531289, -0.09273778,  0.07219949, …,  0.        ,
         1.        ,  0.        ],
       [-1.47415758, -0.09273778, -0.81677269, …,  0.        ,
         1.        ,  0.        ],
       …,
       [ 1.50364073,  0.70684287, -1.029353  , …,  0.        ,
         0.        ,  1.        ],
       [ 0.29046364, -0.09273778, -2.2275329 , …,  0.        ,
         0.        ,  1.        ],
       [ 0.29046364, -0.09273778, -0.19835726, …,  0.        ,
         1.        ,  0.        ]])
```

2.3.4 Question 2.4 Now create a separate, processed training data set by dividing your processed dataframe into training and testing cohorts, using the same settings as Q2.2 (REMEMBER TO USE DIFFERENT TRAINING AND TESTING VARIABLES SO AS NOT TO OVERWRITE YOUR PREVIOUS DATA). Output the resulting shapes of your training and testing samples to confirm that your split was successful, and describe what differences there are between your two training datasets.

[17]:
```
X_train2, X_test2, y_train2, y_test2 = train_test_split(df_prepared, target,␣
↪test_size = 0.3, random_state = 42)
print(X_train2.shape)
print(X_test2.shape)
print(y_train2.shape)
print(y_test2.shape)
```

```
(212, 30)
(91, 30)
(212,)
(91,)
```

The difference between the two training datasets are that train without the 2 is the train of unprocessed data while the train2 is the train of processed data which means that the shape of our datasets are different. The different shapes between our data types comes from the the data that comes through after being passed through the OneHotEncoder(). The OneHotEncoder() adds extra columns for each category which causes the category shape displayed for train2 to change from train.

2.4 [50 Points] Part 3. Learning Methods

We're finally ready to actually begin classifying our data. To do so we'll employ multiple learning methods and compare result.

2.4.1 Linear Decision Boundary Methods

2.4.2 SVM (Support Vector Machine)

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

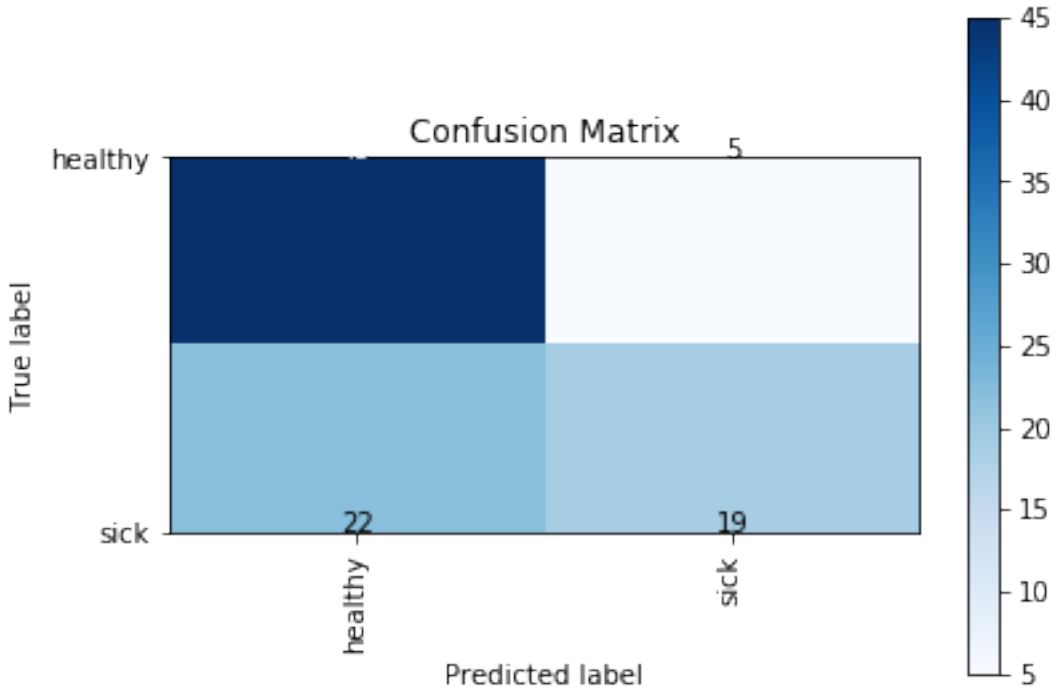
2.4.3 Question 3.1.1 Implement a Support Vector Machine classifier on your RAW dataset. Review the SVM Documentation for how to implement a model. For this implementation you can simply use the default settings, but set probability = True.

[18]:
```
clf = SVC(gamma = 'scale', probability = True, random_state=42).
↪fit(X_train,y_train)
```

2.4.4 Question 3.1.2 Report the accuracy, precision, recall, F1 Score, and confusion matrix of the resulting model.

[19]:
```
predict = clf.predict(X_test)
print("Accuracy: ", metrics.accuracy_score(y_test, predict))
print("Precision: ", metrics.precision_score(y_test, predict))
print("Recall: ", metrics.recall_score(y_test, predict))
print("F1 Score: ", metrics.f1_score(y_test, predict))
print("Confusion Matrix: ", metrics.confusion_matrix(y_test, predict))
draw_confusion_matrix(y_test, predict, ['healthy', 'sick'])
```

```
Accuracy:  0.7032967032967034
Precision:  0.7916666666666666
Recall:  0.4634146341463415
F1 Score:  0.5846153846153846
Confusion Matrix:  [[45  5]
 [22 19]]
```



2.4.5 Question 3.1.3 Discuss what each measure is reporting, why they are different, and why are each of these measures is significant. Explore why we might choose to evaluate the performance of differing models differently based on these factors. Try to give some specific examples of scenarios in which you might value one of these measures over the others.

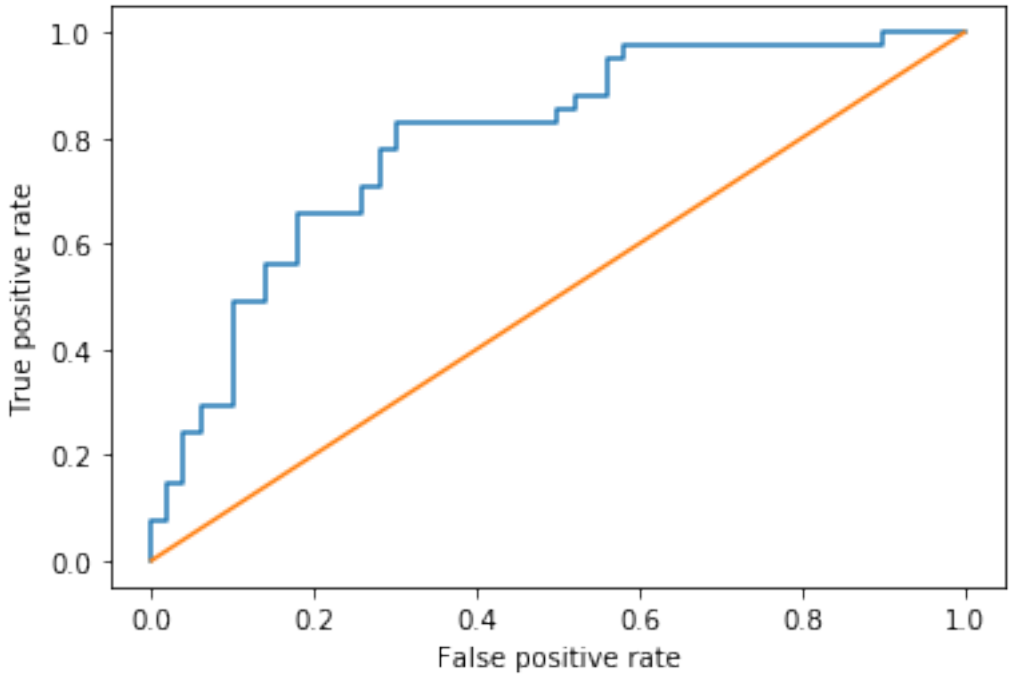
Accuracy represents the accuracy of the classification score. Precision represents the intuivity of the classifier not to label as positive a sample that is negative. Recall represents intuitively the ability of the classifier to find all the positive samples. F1 score is the weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The top left of the confusion matrix represents true positive (number of positive examples classified correctly), the top right represents false negative (number of positive examples classified incorrectly), the bottom left shows false positive (number of negative examples classified incorrectly!), and the bottom right represents true negative (number of negative examples classified correctly).

The accuracy is significant because it is used to check how well our model is performing. If the model is performing well, the accuracy would be close to 1.0.

The precision is significant because it tells us how well our data for true positives is performing. We can determine if correlation really does cause causation based on

2.4.6 Question 3.1.4 Plot a Receiver Operating Characteristic curve, or ROC curve, and describe what it is and what the results indicate

[20]:
```
y_train_score = clf.predict_proba(X_test) #ROC curve rate of true positives in␣
↪proportion to false positives
y_train_score = y_train_score[:,1]
rScore = [0 for _ in range(len(y_test))]
rfpr, rtpr, _ = metrics.roc_curve(y_test, rScore)
tfpr, ttpr, _ = metrics.roc_curve(y_test, y_train_score)
plt.plot(tfpr, ttpr)
plt.plot(rfpr, rtpr, linestyle = '-')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.show()
```



ROC stands for Receiver Operating Characteristic. ROC is away to see how a model can distinguish between true positives and negatives. The results of the ROC indicate the accuracy of a test. When the curve is larger and stretched closer to one, the test much more accurate. When the curve is closer to the 45 degree angle (closer to the middle line), the test is much less accurate.

2.4.7 Question 3.1.5 Rerun, using the exact same settings, only this time use your processed data as inputs.

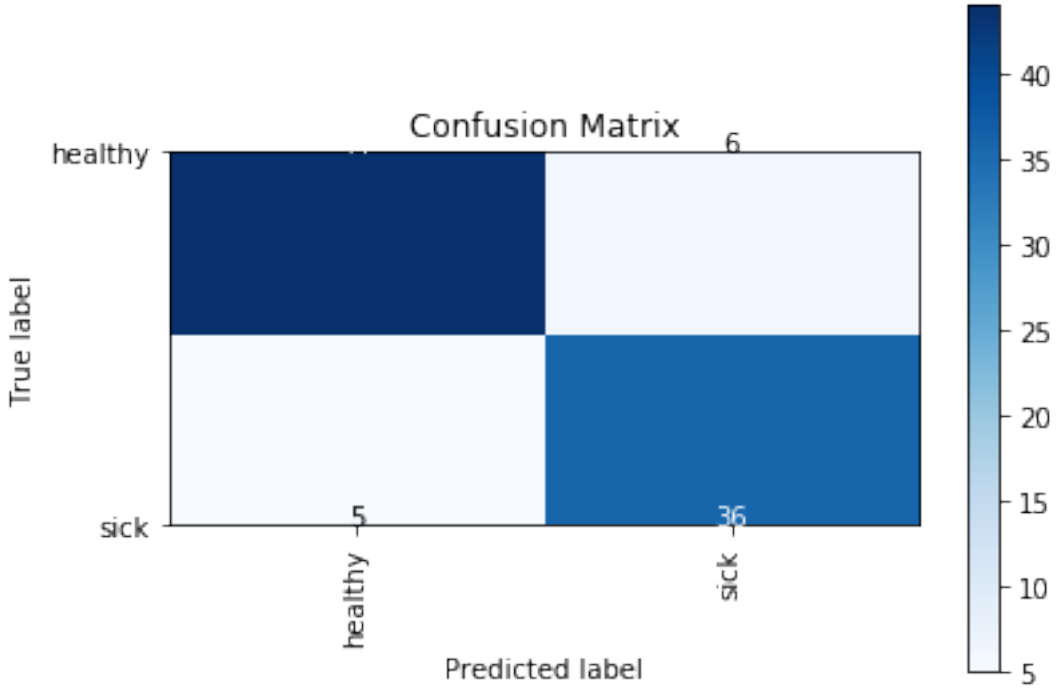
[21]:
```
clf = SVC(gamma = 'scale', probability = True, random_state=42).
↪fit(X_train2,y_train2)
```

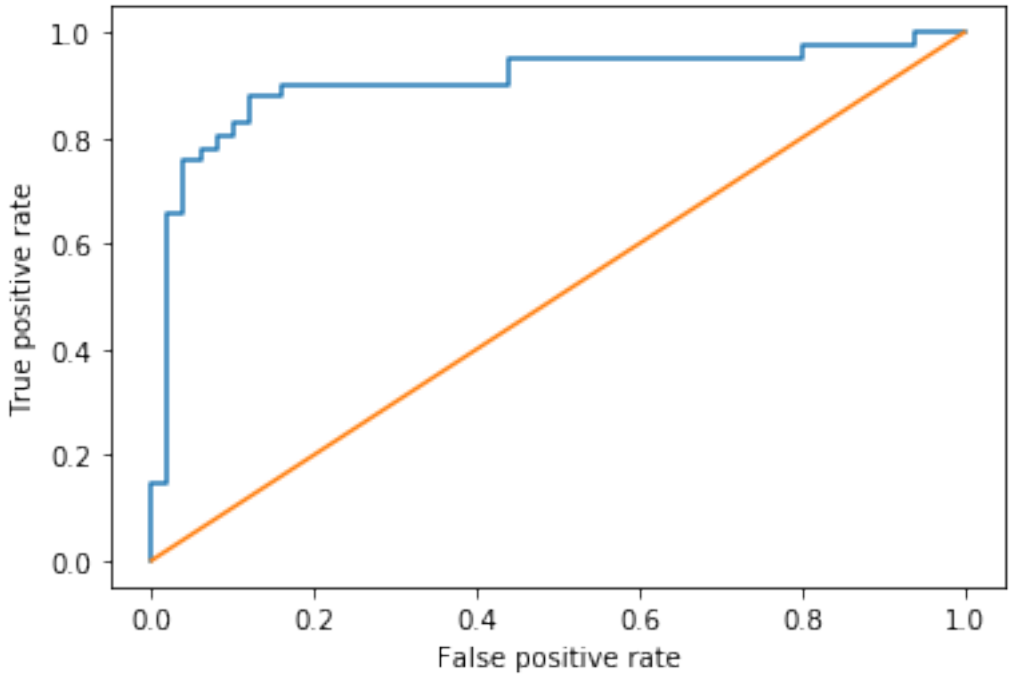
2.4.8 Question 3.1.6 Report the accuracy, precision, recall, F1 Score, confusion matrix, and plot the ROC Curve of the resulting model.

[22]:
```
predict = clf.predict(X_test2)
print("Accuracy: ", metrics.accuracy_score(y_test2, predict))
print("Precision: ", metrics.precision_score(y_test2, predict))
print("Recall: ", metrics.recall_score(y_test2, predict))
print("F1 Score: ", metrics.f1_score(y_test2, predict))
print("Confusion Matrix: ", metrics.confusion_matrix(y_test2, predict))
draw_confusion_matrix(y_test2, predict, ['healthy', 'sick'])
y_train2_score = clf.predict_proba(X_test2) #ROC curve rate of true positives␣
↪in proportion to false positives
y_train2_score = y_train2_score[:,1]
```

```
rScore = [0 for _ in range(len(y_test2))]
rfpr, rtpr, _ = metrics.roc_curve(y_test2, rScore)
tfpr, ttpr, _ = metrics.roc_curve(y_test2, y_train2_score)
plt.plot(tfpr, ttpr)
plt.plot(rfpr, rtpr, linestyle = '-')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.show()
```

```
Accuracy:  0.8791208791208791
Precision:  0.8571428571428571
Recall:  0.8780487804878049
F1 Score:  0.8674698795180722
Confusion Matrix:  [[44  6]
 [ 5 36]]
```





2.4.9 Question 3.1.7 Hopefully you've noticed a dramatic change in performance. Discuss why you think your new data has had such a dramatic impact.

The new data has had a dramatic impact because OneHotEncoder() and standard scalar improve the overall model fit. That is why we can see a dramatic difference between the processed data and the unprocessed data. For the processed data, when you run the preprocessing, it helps us find the separator for data.

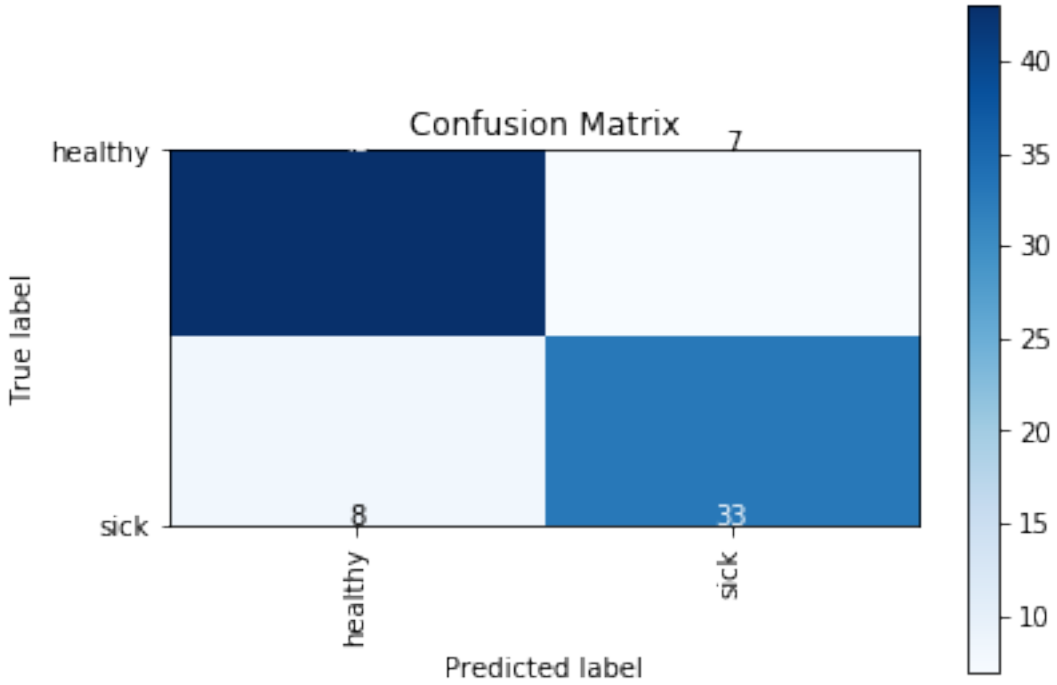
2.4.10 Question 3.1.8 Rerun your SVM, but now modify your model parameter kernel to equal 'linear'. Again report your Accuracy, Precision, Recall, F1 scores, and Confusion matrix and plot the new ROC curve.

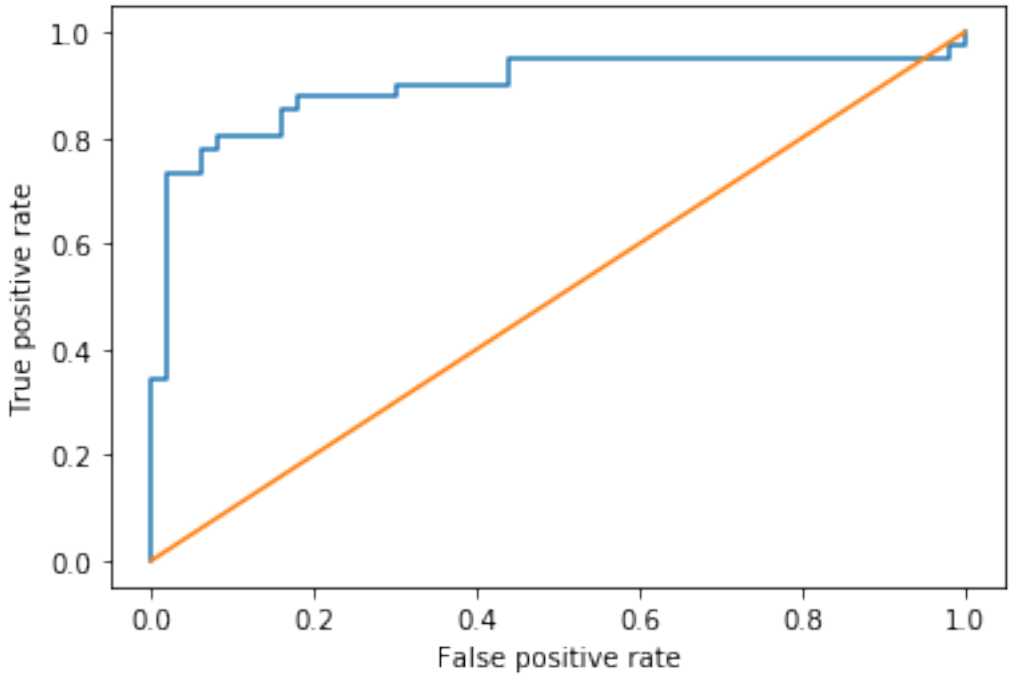
[23]:
```
# SVM
clf = SVC(gamma = 'scale',kernel='linear', probability = True, random_state=42).
 ↪fit(X_train2,y_train2)
```

[24]:
```
predict = clf.predict(X_test2)
print("Accuracy: ", metrics.accuracy_score(y_test2, predict))
print("Precision: ", metrics.precision_score(y_test2, predict))
print("Recall: ", metrics.recall_score(y_test2, predict))
print("F1 Score: ", metrics.f1_score(y_test2, predict))
print("Confusion Matrix: ", metrics.confusion_matrix(y_test2, predict))
draw_confusion_matrix(y_test2, predict, ['healthy', 'sick'])
y_train2_score = clf.predict_proba(X_test2) #ROC curve rate of true positives␣
 ↪in proportion to false positives
```

```
y_train2_score = y_train2_score[:,1]
rScore = [0 for _ in range(len(y_test2))]
rfpr, rtpr, _ = metrics.roc_curve(y_test2, rScore)
tfpr, ttpr, _ = metrics.roc_curve(y_test2, y_train2_score)
plt.plot(tfpr, ttpr)
plt.plot(rfpr, rtpr, linestyle = '-')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.show()
```

```
Accuracy:  0.8351648351648352
Precision:  0.825
Recall:  0.8048780487804879
F1 Score:  0.8148148148148149
Confusion Matrix:  [[43  7]
 [ 8 33]]
```





2.4.11 Question 3.1.9 Explain the what the new results you've achieved mean. Read the documentation to understand what you've changed about your model and explain why changing that input parameter might impact the results in the manner you've observed.

By changing the parameter kernel to linear has caused the results to be worse than before. What this means is that my data is not best linearly seperated. When I have a radially based seperator, it divides the data better which results in data with better accuracy.

2.4.12 Logistic Regression

Knowing that we're dealing with a linearly configured dataset, let's now try another classifier that's well known for handling linear models: Logistic Regression. Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable.

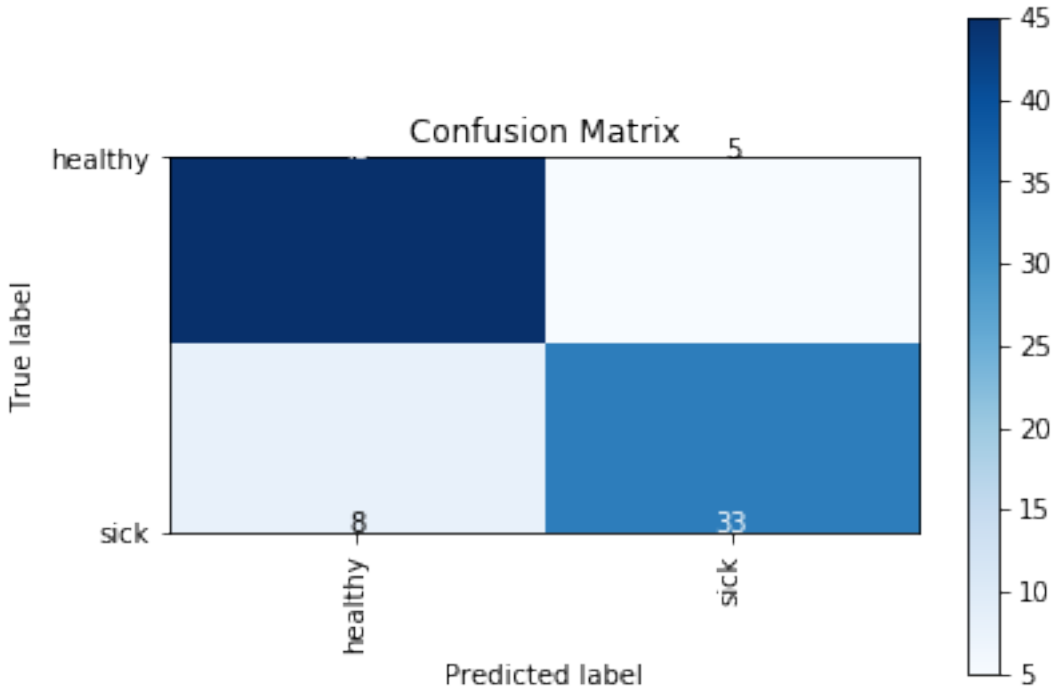
2.4.13 Question 3.2.1 Implement a Logistical Regression Classifier. Review the Logistical Regression Documentation for how to implement the model. For this initial model set the solver = 'sag' and max_iter= 10). Report on the same four metrics as the SVM and graph the resulting ROC curve.

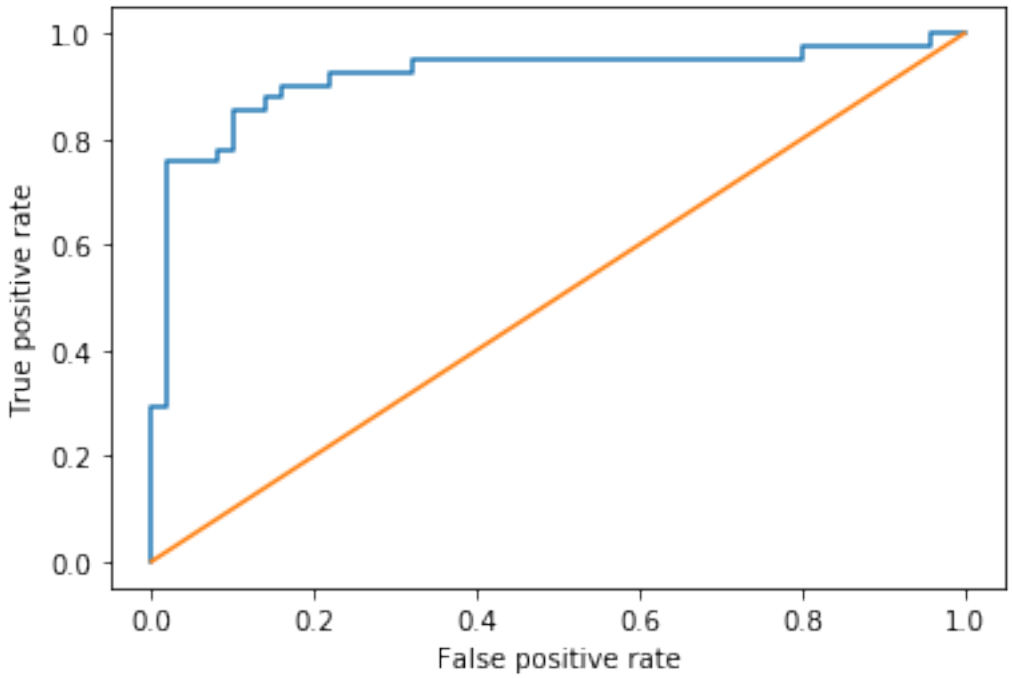
[25]:
```
clf = LogisticRegression(solver = 'sag', max_iter=10).fit(X_train2, y_train2)
predict = clf.predict(X_test2)
print("Accuracy: ", metrics.accuracy_score(y_test2, predict))
print("Precision: ", metrics.precision_score(y_test2, predict))
print("Recall: ", metrics.recall_score(y_test2, predict))
```

```
print("F1 Score: ", metrics.f1_score(y_test2, predict))
print("Confusion Matrix: ", metrics.confusion_matrix(y_test2, predict))
draw_confusion_matrix(y_test2, predict, ['healthy', 'sick'])
y_train2_score = clf.predict_proba(X_test2) #ROC curve rate of true positives␣
 ↪in proportion to false positives
y_train2_score = y_train2_score[:,1]
rScore = [0 for _ in range(len(y_test2))]
rfpr, rtpr, _ = metrics.roc_curve(y_test2, rScore)
tfpr, ttpr, _ = metrics.roc_curve(y_test2, y_train2_score)
plt.plot(tfpr, ttpr)
plt.plot(rfpr, rtpr, linestyle = '-')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.show()
```

```
Accuracy:  0.8571428571428571
Precision:  0.868421052631579
Recall:  0.8048780487804879
F1 Score:  0.8354430379746836
Confusion Matrix:  [[45  5]
 [ 8 33]]
```

```
C:\Users\choie\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
```



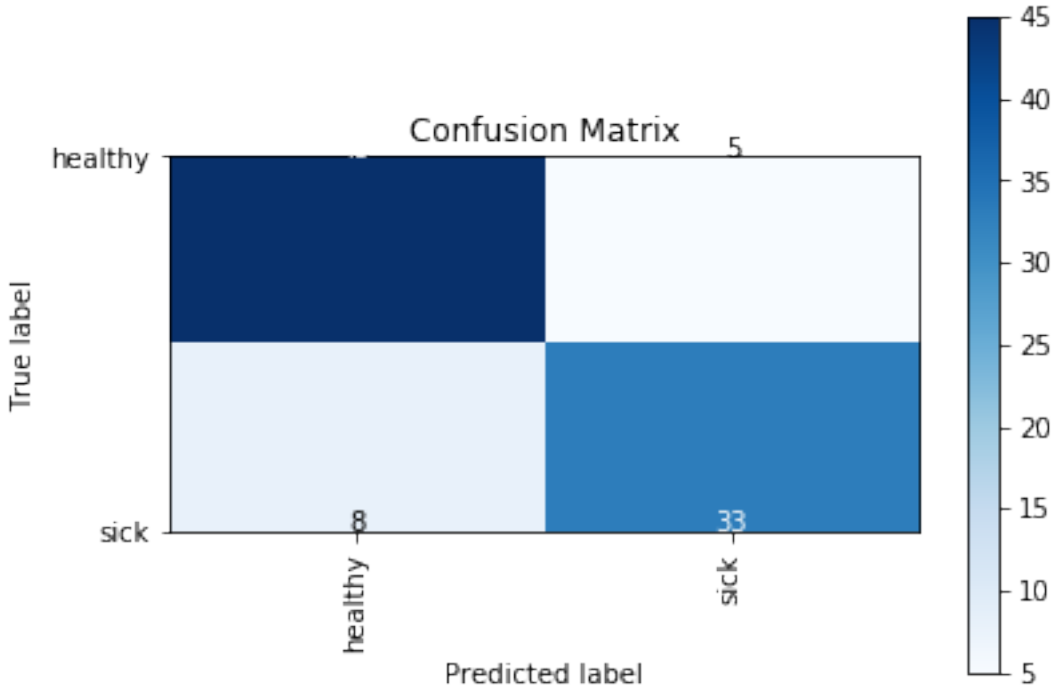


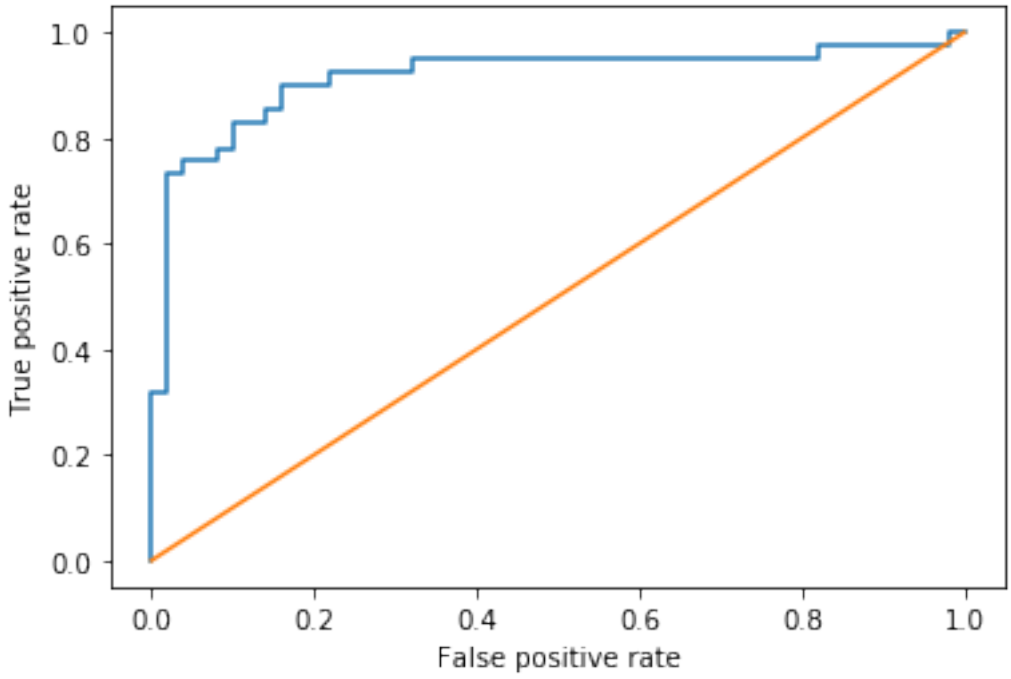
2.4.14 Question 3.2.2 Did you notice that when you ran the previous model you got the following warning: "ConvergenceWarning: The max_iter was reached which means the coef_ did not converge". Check the documentation and see if you can implement a fix for this problem, and again report your results.

[26]:
```
# Logistic Regression
clf = LogisticRegression(solver = 'sag', max_iter=4000).fit(X_train2, y_train2)
predict = clf.predict(X_test2)
print("Accuracy: ", metrics.accuracy_score(y_test2, predict))
print("Precision: ", metrics.precision_score(y_test2, predict))
print("Recall: ", metrics.recall_score(y_test2, predict))
print("F1 Score: ", metrics.f1_score(y_test2, predict))
print("Confusion Matrix: ", metrics.confusion_matrix(y_test2, predict))
draw_confusion_matrix(y_test2, predict, ['healthy', 'sick'])
y_train2_score = clf.predict_proba(X_test2) #ROC curve rate of true positives␣
 ↪in proportion to false positives
y_train2_score = y_train2_score[:,1]
rScore = [0 for _ in range(len(y_test2))]
rfpr, rtpr, _ = metrics.roc_curve(y_test2, rScore)
tfpr, ttpr, _ = metrics.roc_curve(y_test2, y_train2_score)
plt.plot(tfpr, ttpr)
plt.plot(rfpr, rtpr, linestyle = '-')
```

```
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.show()
```

```
Accuracy:  0.8571428571428571
Precision:  0.868421052631579
Recall:  0.8048780487804879
F1 Score:  0.8354430379746836
Confusion Matrix:  [[45  5]
 [ 8 33]]
```





2.4.15 Question 3.2.3 Explain what you changed, and why that produced an improved outcome.

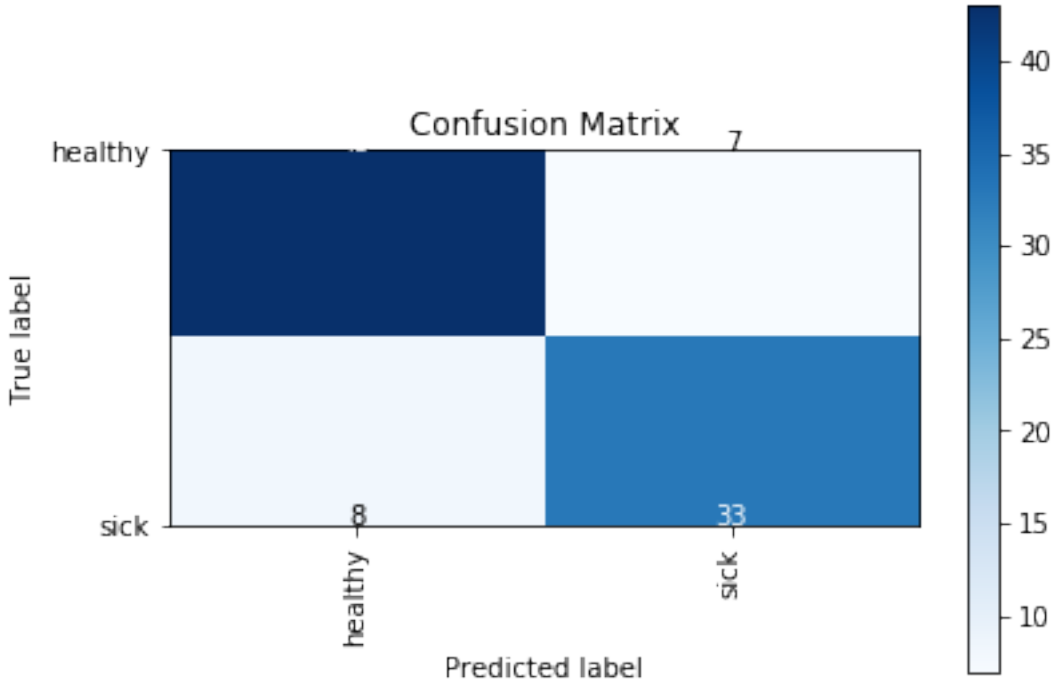
For this model, I changed the max_iter to 4000 when calling LogisticRegression. This produced an improved outcome because by increasing the max_iter, I was allowed to better fit the data to the logistic model.

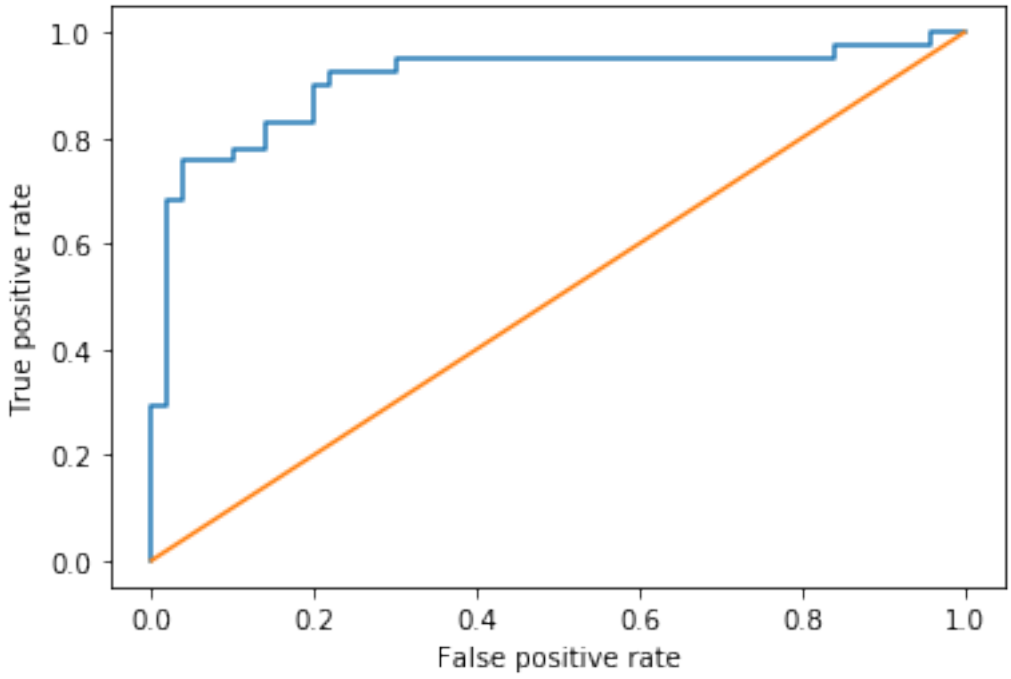
2.4.16 Question 3.2.4 Rerun your logistic classifier, but modify the penalty = 'none', solver='sag' and again report the results.

[27]:
```
# Logistic Regression
clf = LogisticRegression(penalty = 'none', solver = 'sag', max_iter=4000).
 ↪fit(X_train2, y_train2)
predict = clf.predict(X_test2)
print("Accuracy: ", metrics.accuracy_score(y_test2, predict))
print("Precision: ", metrics.precision_score(y_test2, predict))
print("Recall: ", metrics.recall_score(y_test2, predict))
print("F1 Score: ", metrics.f1_score(y_test2, predict))
print("Confusion Matrix: ", metrics.confusion_matrix(y_test2, predict))
draw_confusion_matrix(y_test2, predict, ['healthy', 'sick'])
y_train2_score = clf.predict_proba(X_test2) #ROC curve rate of true positives␣
 ↪in proportion to false positives
y_train2_score = y_train2_score[:,1]
rScore = [0 for _ in range(len(y_test2))]
rfpr, rtpr, _ = metrics.roc_curve(y_test2, rScore)
```

```
tfpr, ttpr, _ = metrics.roc_curve(y_test2, y_train2_score)
plt.plot(tfpr, ttpr)
plt.plot(rfpr, rtpr, linestyle = '-')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.show()
```

```
Accuracy:  0.8351648351648352
Precision:  0.825
Recall:  0.8048780487804879
F1 Score:  0.8148148148148149
Confusion Matrix:  [[43  7]
 [ 8 33]]
```





2.4.17 Question 3.2.5 Explain what what the penalty parameter is doing in this function, what the solver method is, and why this combination likely produced a more optimal outcome.

What the penalty parameter does is apply L2 regularization – the term regulatization being an action where information is added to prevent overfitting for a model. By having the penalty parameter as none, regulatization is not being implemented for the data. Sag is a method of doing gradient descent. By default sag without a penalty parameter uses L2 regularization. The original data was not linearly seperable, but by conducting L2 regularization, you help the logistic regression model find a more optimal hyperplane. This is proven by the fact that without L2 regularization, you're setting the penalty and the loss function to 0 which eliminate regularization. Since our results are better with regularization, the original model may be overfitting.

2.4.18 Question 3.2.6 Both logistic regression and linear SVM are trying to classify data points using a linear decision boundary, then what's the difference between their ways to find this boundary?

Logistic regression takes all the values and squishes them into a sigmoid function and based on a threshold value, it classifies them into either 0 or 1 based on binary classifier. Linear SVM looks for fattest seperator between support vectors and the seperator. It tries to maximize the distance between the classifier.

2.4.19 Clustering Approaches

Let us now try a different approach to classification using a clustering algorithm. Specifically, we're going to be using K-Nearest Neighbor, one of the most popular clustering approaches.

2.4.20 K-Nearest Neighbor

2.4.21 Question 3.3.1 Implement a K-Nearest Neighbor algorithm on our data and report the results. For this initial implementation simply use the default settings. Refer to the KNN Documentation for details on implementation. Report on the accuracy of the resulting model.

[28]:
```
# k-Nearest Neighbors algorithm
neigh = KNeighborsClassifier()
neigh.fit(X_train2, y_train2)
pred = neigh.predict(X_test2)
print("Accuracy: ", metrics.accuracy_score(y_test2, pred))
```

```
Accuracy:  0.8681318681318682
```

2.4.22 Question 3.3.2 For clustering algorithms, we use different measures to determine the effectiveness of the model. Specifically here, we're interested in the Homogeneity Score, Completeness Score, V-Measure, Adjusted Rand Score, and Adjusted Mutual Information. Calculate each score (hint review the SKlearn Metrics Clustering documentation for how to implement).

[29]:
```
print("Homogeneity Score: ", metrics.homogeneity_score(y_test2, pred))
print("Completeness Score: ", metrics.completeness_score(y_test2, pred))
print("V-Measure", metrics.v_measure_score(y_test2, pred))
print("Adjusted Rand Score: ", metrics.adjusted_rand_score(y_test2, pred))
print("Adjusted Mutual Information", metrics.
 ↪adjusted_mutual_info_score(y_test2, pred))
```

```
Homogeneity Score:  0.43454222652531754
Completeness Score:  0.43454222652531754
V-Measure 0.43454222652531754
Adjusted Rand Score:  0.5369956467290834
Adjusted Mutual Information 0.4299123552669263
```

```
C:\Users\choie\Anaconda3\lib\site-
packages\sklearn\metrics\cluster\supervised.py:746: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
```

2.4.23 Question 3.3.3 Explain what each score means and interpret the results for this particular model.

As we're beginning to see, the input parameters for your model can dramatically impact the performance of the model. How do you know which settings to choose? Studying the models and studying your datasets are critical as they can help you anticipate which models and settings are likely to produce optimal results. However sometimes that isn't enough, and a brute force method is necessary to determine which parameters to use. For this next question we'll attempt to optimize a parameter using a brute force approach.

2.4.24 Question 3.3.4 Parameter Optimization. The KNN Algorithm includes an n_neighbors attribute that specifies how many neighbors to use when developing the cluster. (The default value is 5, which is what your previous model used.) Lets now try n values of: 1, 2, 3, 5, 10, 20, 50, and 100. Run your model for each value and report the 6 measures (5 clustering specific plus accuracy) for each. Report on which n value produces the best accuracy and V-Measure. (HINT leverage python's ability to loop to run through the array and generate results without needing to manually code each iteration).

[30]:
```
n = [ 1, 2, 3, 5, 10, 20, 50, 100]
for k in n:
    neigh = KNeighborsClassifier(n_neighbors = k)
    neigh.fit(X_train2, y_train2)
    pred = neigh.predict(X_test2)
    print("Accuracy: ", metrics.accuracy_score(y_test2, pred))
    print("Homogeneity Score: ", metrics.homogeneity_score(y_test2, pred))
    print("Completeness Score: ", metrics.completeness_score(y_test2, pred))
    print("V-Measure", metrics.v_measure_score(y_test2, pred))
    print("Adjusted Rand Score: ", metrics.adjusted_rand_score(y_test2, pred))
    print("Adjusted Mutual Information", metrics.
 ↪adjusted_mutual_info_score(y_test2, pred))
```

```
Accuracy:  0.7472527472527473
Homogeneity Score:  0.18008024677600198
Completeness Score:  0.18071700740060376
V-Measure 0.18039806518811052
Adjusted Rand Score:  0.23614505250729015
Adjusted Mutual Information 0.17336608891216787
Accuracy:  0.8021978021978022
Homogeneity Score:  0.2982684707995045
Completeness Score:  0.32799516339289997
V-Measure 0.3124263025139275
Adjusted Rand Score:  0.3583372288260098
Adjusted Mutual Information 0.29250315084058737
Accuracy:  0.8351648351648352
Homogeneity Score:  0.3648087447659865
Completeness Score:  0.3622620682280168
V-Measure 0.3635309464352902
Adjusted Rand Score:  0.443215972520395
Adjusted Mutual Information 0.3570783373624062
Accuracy:  0.8681318681318682
Homogeneity Score:  0.43454222652531754
Completeness Score:  0.43454222652531754
V-Measure 0.43454222652531754
Adjusted Rand Score:  0.5369956467290834
Adjusted Mutual Information 0.4299123552669263
Accuracy:  0.8461538461538461
Homogeneity Score:  0.3775642686402215
```

```
Completeness Score:  0.3846360011742515
V-Measure 0.38106732896173023
Adjusted Rand Score:  0.4735198145206303
Adjusted Mutual Information 0.3724646601891104
Accuracy:  0.8901098901098901
Homogeneity Score:  0.50122315250762
Completeness Score:  0.5106110007994157
V-Measure 0.5058735261886211
Adjusted Rand Score:  0.6044072817826628
Adjusted Mutual Information 0.4971366798028251
Accuracy:  0.8681318681318682
Homogeneity Score:  0.43596050415976273
Completeness Score:  0.44412599103679684
V-Measure 0.44000536769880233
Adjusted Rand Score:  0.5370100038641535
Adjusted Mutual Information 0.43133933536300423
Accuracy:  0.8571428571428571
Homogeneity Score:  0.4153345842002435
Completeness Score:  0.43254919125846314
V-Measure 0.4237671334147018
Adjusted Rand Score:  0.5047942411861096
Adjusted Mutual Information 0.4105407301203446
```

```
C:\Users\choie\Anaconda3\lib\site-
packages\sklearn\metrics\cluster\supervised.py:746: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
C:\Users\choie\Anaconda3\lib\site-
packages\sklearn\metrics\cluster\supervised.py:746: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
C:\Users\choie\Anaconda3\lib\site-
packages\sklearn\metrics\cluster\supervised.py:746: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
C:\Users\choie\Anaconda3\lib\site-
packages\sklearn\metrics\cluster\supervised.py:746: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
C:\Users\choie\Anaconda3\lib\site-
packages\sklearn\metrics\cluster\supervised.py:746: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
```

```
C:\Users\choie\Anaconda3\lib\site-
packages\sklearn\metrics\cluster\supervised.py:746: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
C:\Users\choie\Anaconda3\lib\site-
packages\sklearn\metrics\cluster\supervised.py:746: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
C:\Users\choie\Anaconda3\lib\site-
packages\sklearn\metrics\cluster\supervised.py:746: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
```

2.4.25 Question 3.3.5 When are clustering algorithms most effective, and what do you think explains the comparative results we achieved?

Results were better with rbf kernel on SVM which indicates that the data is not best linearly seperated. The fact that clustering isn't as good as well means that we are unable to come up with an optimal set of neighbors to group my data points into classes. My clustering algorithm based on the homogeneity completeness and vmeasure scores indicate that our clustering algorithm is inconsistent with finding groups to seperate our data into classes. Therefore, the clustering algorithm is suboptimal compared to the rbf kernel SVM algorithm.