cs188_project2

February 12, 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

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)

1.2 Loading Essentials and Helper Functions

[2]:

```python
#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)
```

[3]:

```python
# 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)
```

[4]:
```
# 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()
```

1.3 [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.)

[5]:
```
data = pd.read_csv("./heartdisease.csv")
```

1.3.1 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]:
```
data.head()
```

```
[6]:    age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
     0   63    1   3       145   233    1        0      150      0      2.3      0
     1   37    1   2       130   250    0        1      187      0      3.5      0
     2   41    0   1       130   204    0        0      172      0      1.4      2
     3   56    1   1       120   236    0        1      178      0      0.8      2
     4   57    0   0       120   354    0        1      163      1      0.6      2

        ca  thal   sick
     0   0     1  False
     1   0     2  False
     2   0     2  False
     3   0     2  False
     4   0     2  False
```

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

```
[7]:              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
     count  303.000000
     mean     2.313531
     std      0.612277
     min      0.000000
     25%      2.000000
     50%      2.000000
     75%      3.000000
     max      3.000000
```

```
[8]: data.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
```

[9]:

```
null_data = data[data.isnull().any(axis=1)]
null_data
```

[9]:

```
Empty DataFrame
Columns: [age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak,
slope, ca, thal, sick]
Index: []
```

1.3.2 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?

Answer

- In the above cells, I use the basic info() and describe() functions to get a general idea of how my dataset looks and what kinds of values I would be dealing with for each feature.
- I then use the info() function to see the data types for each feature and to check if there are any data types that I would have deal with in order for my model to work effectively. I see that 'sick' is a bool type that I would have to deal with since the model won't be able to understand how to deal with a bool
- I also check for any null values but I find that there are no null values in the dataset. The presence of no null values means that I don't have to do any data imputation that might negetively affect my model.

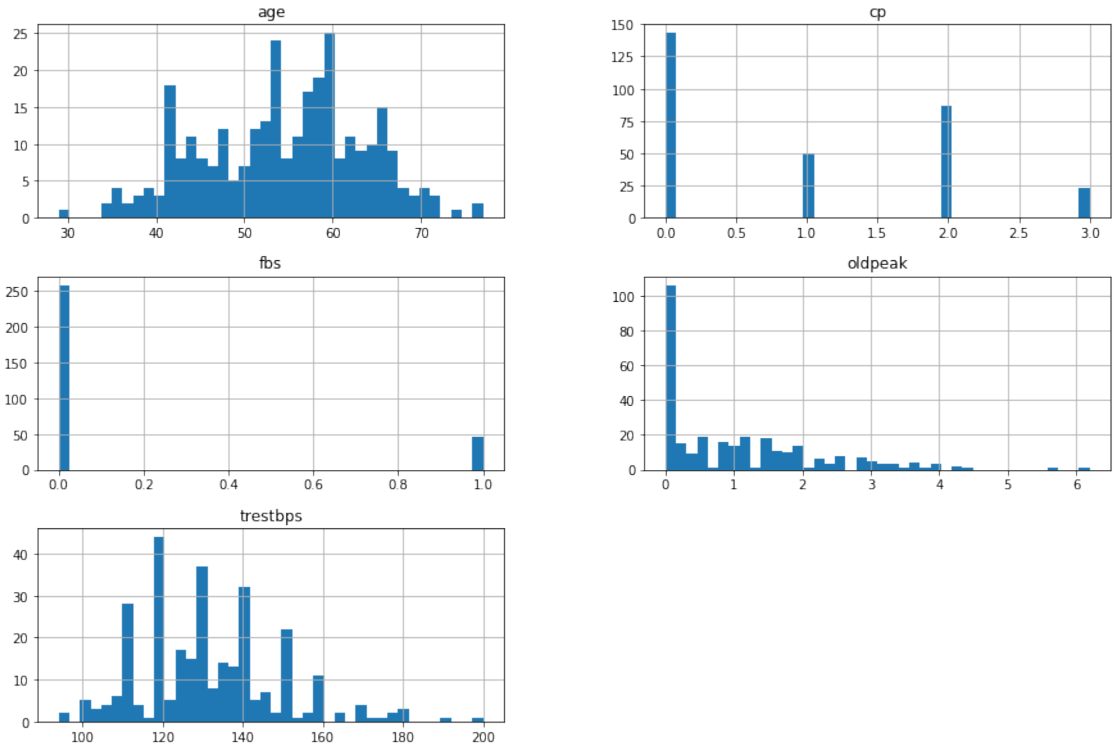
1.3.3 Question 1.3 Before we begin our analysis we need to fix the field(s) that will be problematic. Specifically convert our boolean sick variable into a binary numeric target variable (values of either '0' or '1'), and then drop the original sick datafield from the dataframe.

[10]:
```
data.sick = data.sick.replace({True: 1, False: 0})
#data.info()
```

1.3.4 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]:
```
features = ["age","cp","trestbps","fbs","oldpeak"]
data[features].hist(bins=40, figsize=(15,10))
```

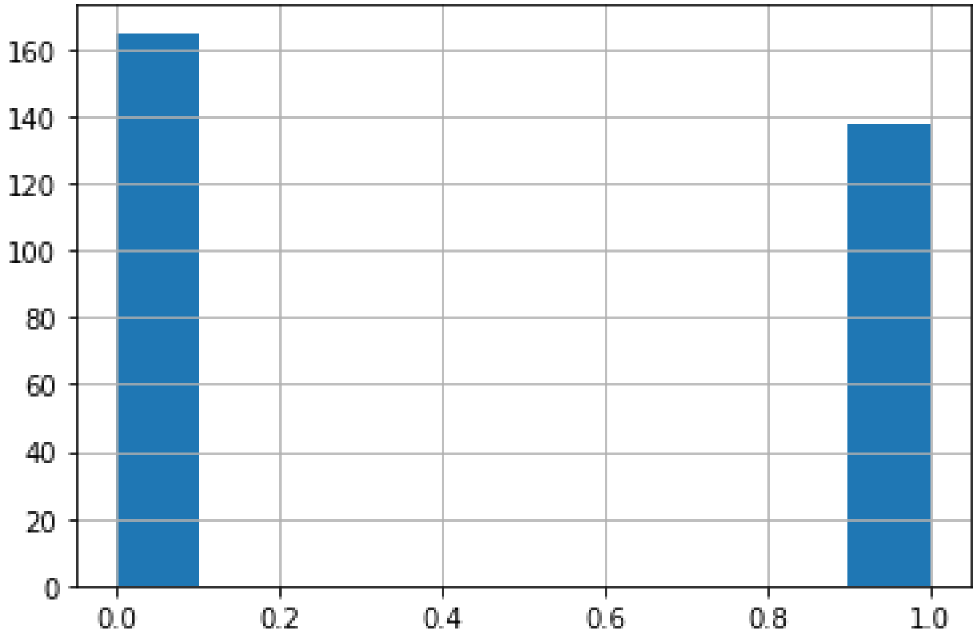
[11]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000271766D3D48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000027176CA3108>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000027176CD5A08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000027176D0B788>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000027176D440C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000027176D7D208>]],
      dtype=object)
```



1.3.5 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]:
```python
data["sick"].hist()
print("Number of sick patients: {}".format(data["sick"].isin([1]).sum()))
print("Number of healthy patients: {}".format(data["sick"].isin([0]).sum()))
```

```
Number of sick patients: 138
Number of healthy patients: 165
```



Findings

- The number of sick patients are 138 while the number of healthy patients are 165.
- This is good for us because our dataset is roughly balanced and therefore we don't need to artificially balance a dataset.

1.3.6 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.

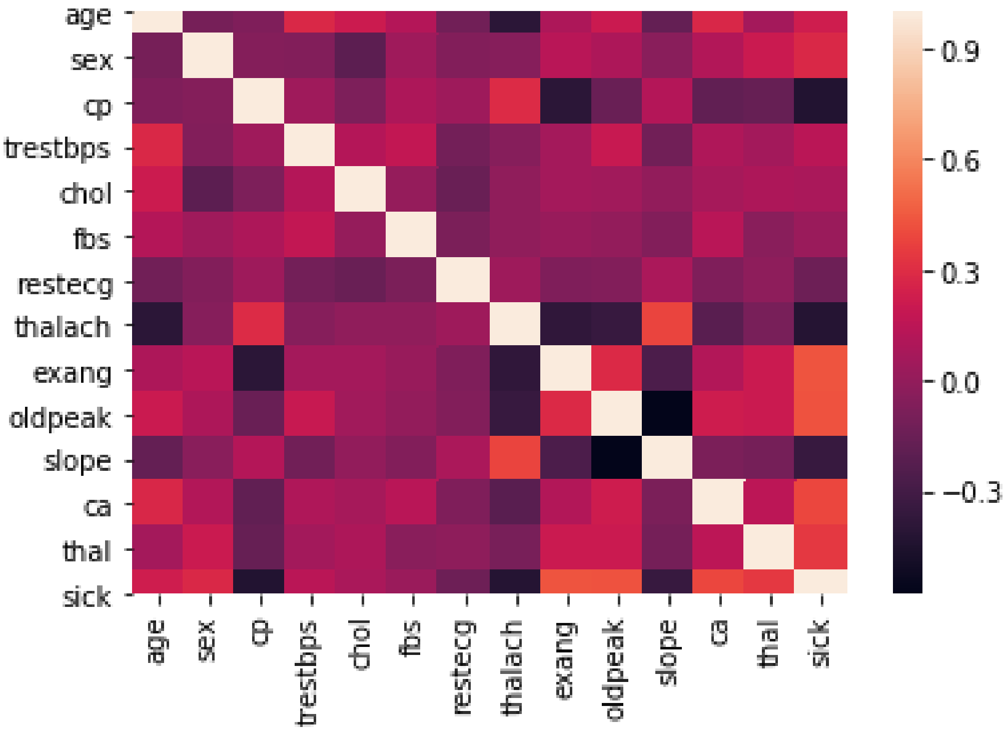
Answer Some problems that can arise while artificially balancing a dataset:

- You might skew the data towards one class or one feature
- You might artificially add data that is around the border of the datasets and hence increase false predictions
- Could change the correlation between 2 features

1.3.7 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)?

[40]:
```
corr_matrix = data.corr()
sns.heatmap(corr_matrix)
```

[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27177213688>
```



[Discuss correlations here]

Positive and comparatively strong correlation: - exang and sick - oldpeak and sick - sick and ca

Negative and comparatively strong correlation: - sick and cp - sick and thalach - slope and old peak - exang and cp - thalach and age

1.4 [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!

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

[14]:
```
target_column = data.sick

new_data = data.drop("sick", axis=1)
```

1.4.2 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]:
```
raw_x_train, raw_x_test, raw_y_train, raw_y_test = train_test_split(new_data,
                                                                             ␣
 ↪target_column,
                                                                      test_size=0.
 ↪3, random_state=42)

print(raw_x_train.shape, raw_x_test.shape, raw_y_train.shape, raw_y_test.shape)
```

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

1.4.3 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.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

data_cat_features = ["cp","fbs","restecg","exang","slope","ca", "thal"]

data_num = new_data.drop(columns=data_cat_features)

data_num_pipeline = Pipeline([
    ('std_scaler', StandardScaler()),
])

data_num_tr = data_num_pipeline.fit_transform(data_num)
data_num_features = list(data_num)

data_full_pipeline = ColumnTransformer([
    ("num", data_num_pipeline, data_num_features),
    ("cat", OneHotEncoder(sparse=False, handle_unknown='ignore'),␣
 ↪data_cat_features),
])

data_prepared = data_full_pipeline.fit_transform(new_data)
data_prepared = pd.DataFrame(data_prepared)

data_prepared.head(10)
```

[16]:

```
          0         1         2         3         4    5    6    7    8    9  \
0  0.952197  0.763956 -0.256334  0.015443  1.087338  0.0  1.0  0.0  0.0  0.0
1 -1.915313 -0.092738  0.072199  1.633471  2.122573  0.0  1.0  0.0  0.0  1.0
2 -1.474158 -0.092738 -0.816773  0.977514  0.310912  1.0  0.0  0.0  1.0  0.0
3  0.180175 -0.663867 -0.198357  1.239897 -0.206705  0.0  1.0  0.0  1.0  0.0
4  0.290464 -0.663867  2.082050  0.583939 -0.379244  1.0  0.0  1.0  0.0  0.0
5  0.290464  0.478391 -1.048678 -0.072018 -0.551783  0.0  1.0  1.0  0.0  0.0
6  0.180175  0.478391  0.922521  0.146634  0.224643  1.0  0.0  0.0  1.0  0.0
7 -1.143291 -0.663867  0.323431  1.021244 -0.896862  0.0  1.0  0.0  1.0  0.0
8 -0.260980  2.306004 -0.913400  0.540209 -0.465514  0.0  1.0  0.0  0.0  1.0
9  0.290464  1.049520 -1.512490  1.064975  0.483451  0.0  1.0  0.0  0.0  1.0

   …   20   21   22   23   24   25   26   27   28   29
0  …  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
1  …  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
2  …  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
3  …  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
4  …  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
5  …  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
6  …  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
7  …  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
8  …  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
9  …  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0

[10 rows x 30 columns]
```

1.4.4 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_train, x_test, y_train, y_test = train_test_split(data_prepared,␣
 ↪target_column,
                                                        test_size=0.
 ↪3,random_state=42)

print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

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

The two datasets primarily difer in two major areas:

1. The raw dataset is not scaled appropriately whereas the pipelined data is. This is important because our model will weigh features with a greater value more even though they should not be. For example, cholestrol is in the range 126 to 564, which is much greater than the other features and therefore will be weighed much more. However, from our correlation heatmap, we see that cholestrol is not highly correlated to sickness. Therefore, this results in incorrect results in our model.

2. A similar problem occurs with categorical features like sex, cp, fbs, exang, slope, ca, thalach, restecg which account for the majority of the features in our dataset. This means that the model will weigh a greater label more and therefore lead to incorrect results.

- We will see this result later when we test our datasets. You can see that the raw data performs poorly against the pipelined data.

1.5 [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.

1.5.1 Linear Decision Boundary Methods

1.5.2 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.

1.5.3 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]:

```
# SVM
raw_svm = SVC(probability=True, gamma='auto')
raw_svm.fit(raw_x_train, raw_y_train)
raw_svm_predicted = raw_svm.predict(raw_x_test)
raw_score = raw_svm.predict_proba(raw_x_test)
```

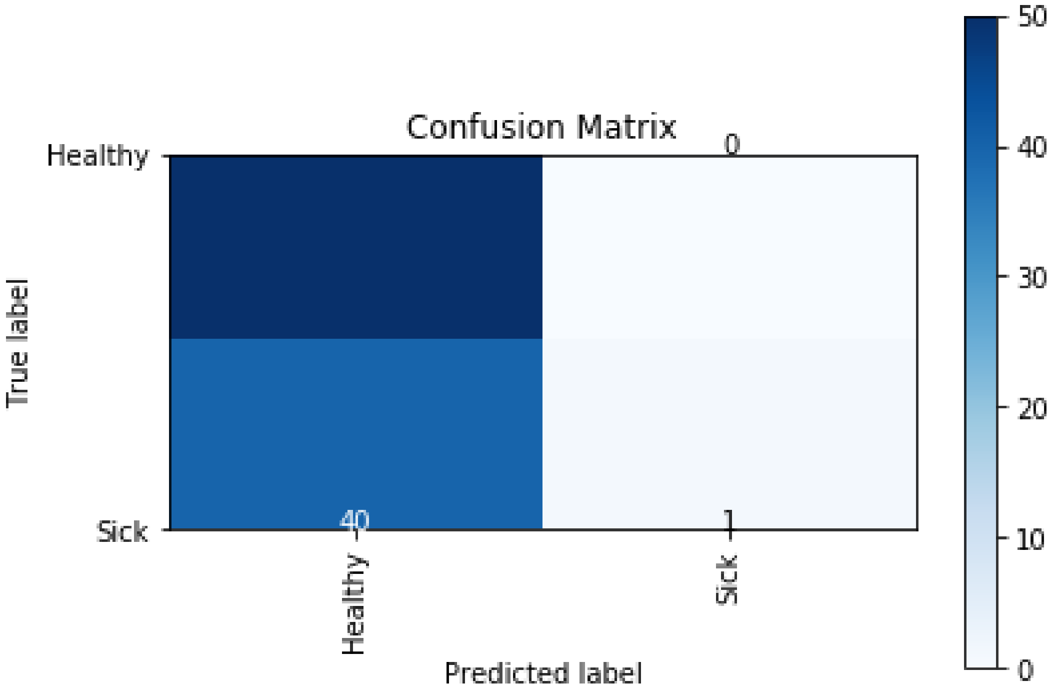
1.5.4 Question 3.1.2 Report the accuracy, precision, recall, F1 Score, and confusion matrix of the resulting model.

[19]:

```
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(raw_y_test,␣
 ↪raw_svm_predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(raw_y_test,␣
 ↪raw_svm_predicted)))
print("%-12s %f" % ('Recall:', metrics.recall_score(raw_y_test,␣
 ↪raw_svm_predicted)))
print("%-12s %f" % ('F1 score:', metrics.f1_score(raw_y_test,␣
 ↪raw_svm_predicted)))

print("Confusion Matrix for Raw Data: \n")
draw_confusion_matrix(raw_y_test, raw_svm_predicted, ["Healthy","Sick"])
```

```
Accuracy:    0.560440
Precision:   1.000000
Recall:      0.024390
F1 score:    0.047619
Confusion Matrix for Raw Data:
```



1.5.5 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.

Answer

Accuracy

- Accuracy is the fraction of predictions our model got right, or number of correct predictions/total number of predictions.
- It is important as it tells us how many false negatives, or false positive values that our model is classifying. Optimistically we would want our model to classify as less false values as possible and therefore this score is really important in determining that.
- Value this over anything else when we want to see the overall effectiveness of our model

Precision

- Precision explores what proportion of positive classifications were actually correct.
- It is the ratio of the number of correct positives over the total number of positive classifications (correct and otherwise).
- This relates directly to False Positives as a higher Precision score corresponds to a low rate of false positives.

- Again the importance of this score is similar to that of accuracy as it tells us how many false positives our model has guessed.
- Value this over other models when we care a lot about false positives. For example, in cancer detection we don't want the patient to go through the whole process of chemotherapy or other procedures when they don't have cancer.

Recall:

- In recall, we are worried about false negatives, and overlooking potential positives.
- Recall is the ratio of correctly predicted positive observations to the all observations in actual class
- This again has a similar importance as that of precision and accuracy
- Value this over other models when false negatives are extreemely harmful. For example in a fraud detection model we don't want to classify a potential fraud to be safe when it has a potential to be harmful.

F1 Score:

- F1 Score is a balance. It is the weighted average of Precision and Recall and therefore takes both false positives and false negatives into account.
- This is important when we want to minimize the false predictions our model makes.
- Value this over other models when we absolutely don't want our model to classify things incorrectly. For example in the case of cancer, we don't want to classify a sick person as healthy and vice-versa.

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

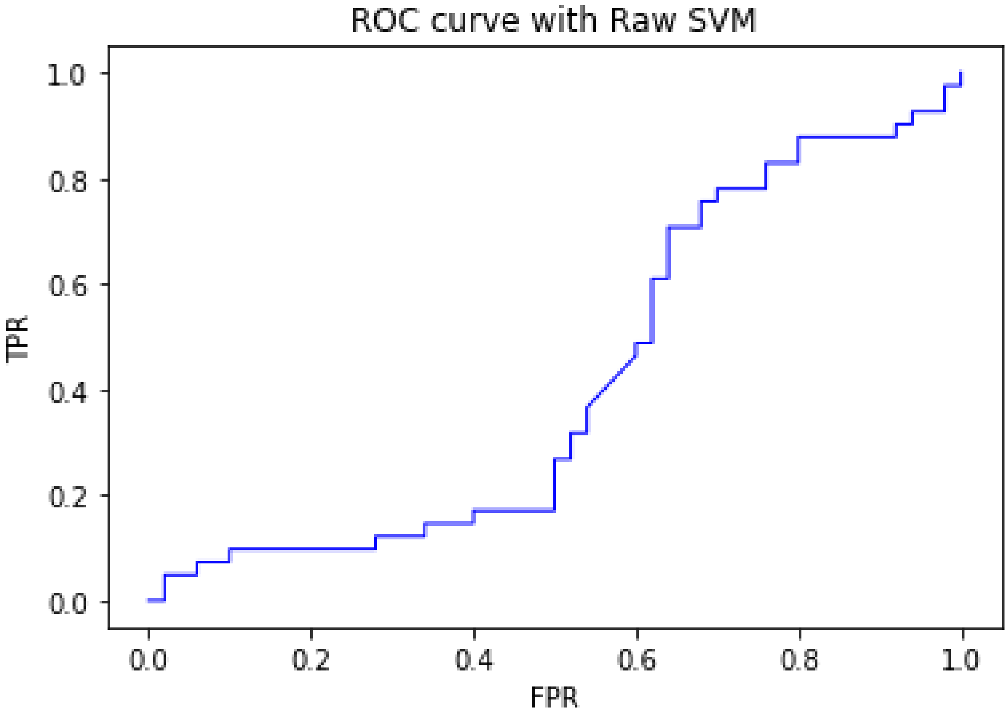
[20]:

```
print("Raw SVM Model Performance Results: \n")

fpr_svm, tpr_svm, thresholds = metrics.roc_curve(raw_y_test, raw_score[:, 1],␣
 ↪pos_label=1)

pyplot.figure(1)
pyplot.plot(fpr_svm, tpr_svm, color='blue', lw=1)
pyplot.title("ROC curve with Raw SVM")
pyplot.xlabel("FPR")
pyplot.ylabel("TPR")
pyplot.show()
```

```
Raw SVM Model Performance Results:
```



ROC Curve:

- The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.
- An ROC curve plots TPR vs. FPR at different classification thresholds.
- Lowering the classification threshold classifies more items as positive, increasing both False Positives and True Positives.
- The area under an ROC curve is a measure of the usefulness of a test in general, where a greater area means a more useful test. Closer the area under the curve is to 1, the better the model is.

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

[21]:
```
svm = SVC(probability=True, gamma='auto')
svm.fit(x_train, y_train)
svm_predicted = svm.predict(x_test)
score = svm.predict_proba(x_test)
```

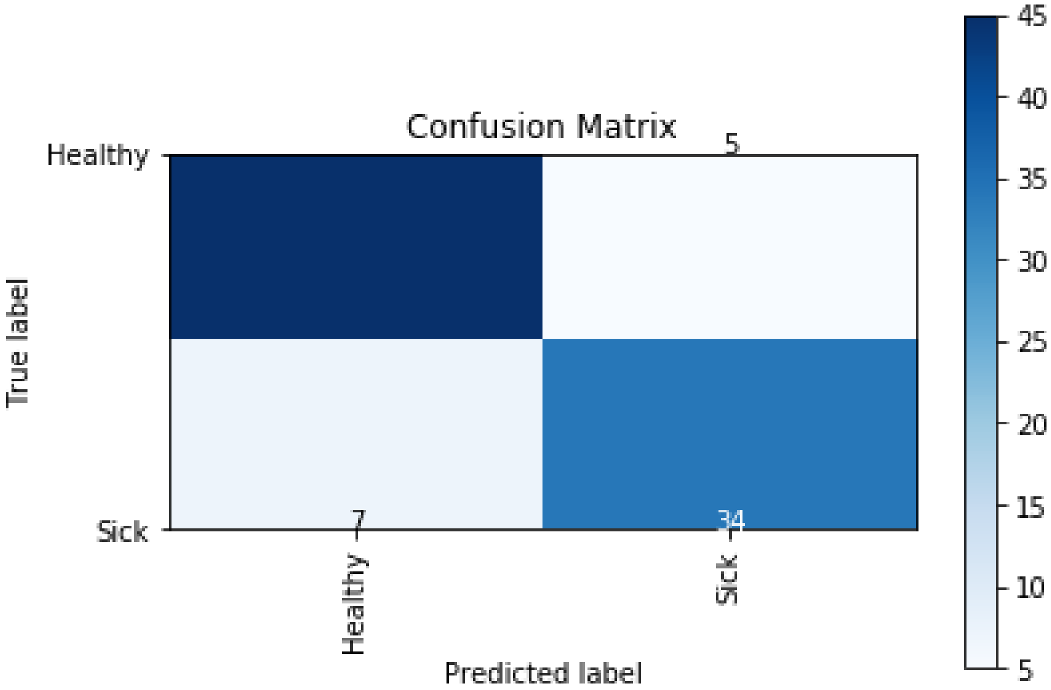
1.5.8 Question 3.1.6 Report the accuracy, precision, recall, F1 Score, confusion matrix, and plot the ROC Curve of the resulting model.

[22]:

```
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(y_test, svm_predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(y_test,␣
 ↪svm_predicted)))
print("%-12s %f" % ('Recall:', metrics.recall_score(y_test, svm_predicted)))
print("%-12s %f" % ('F1 score:', metrics.f1_score(y_test, svm_predicted)))

print("Confusion Matrix for Data: \n")
draw_confusion_matrix(y_test, svm_predicted, ["Healthy","Sick"])
```

```
Accuracy:    0.868132
Precision:   0.871795
Recall:      0.829268
F1 score:    0.850000
Confusion Matrix for Data:
```



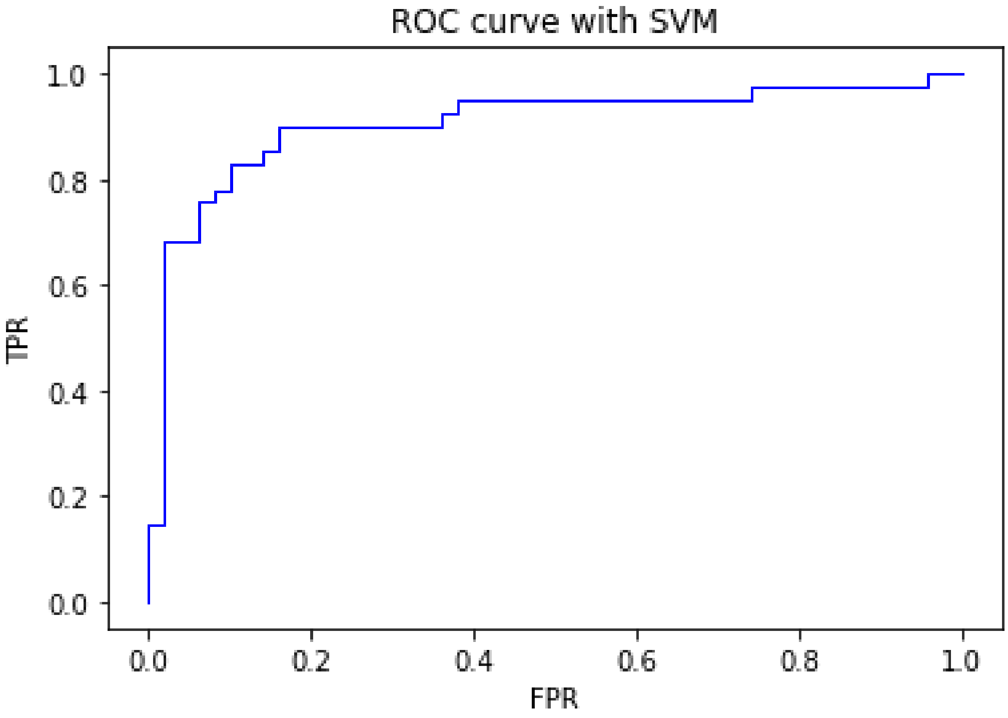
[23]:

```
print("SVM Model Performance Results: \n")

fpr_svm, tpr_svm, thresholds = metrics.roc_curve(y_test, score[:, 1],␣
 ↪pos_label=1)
```

```
pyplot.figure(1)
pyplot.plot(fpr_svm, tpr_svm, color='blue', lw=1)
pyplot.title("ROC curve with SVM")
pyplot.xlabel("FPR")
pyplot.ylabel("TPR")
pyplot.show()
```

```
SVM Model Performance Results:
```



1.5.9 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.

Answer

- Since I pipelined my data and made sure all my features were appropriately scaled or encoded, my model correctly weighed the data and did not unnecessarily weigh some category or feature more than others.
- This resulted in correct predictions and greatly reduced the classification of false positives or false negatives.

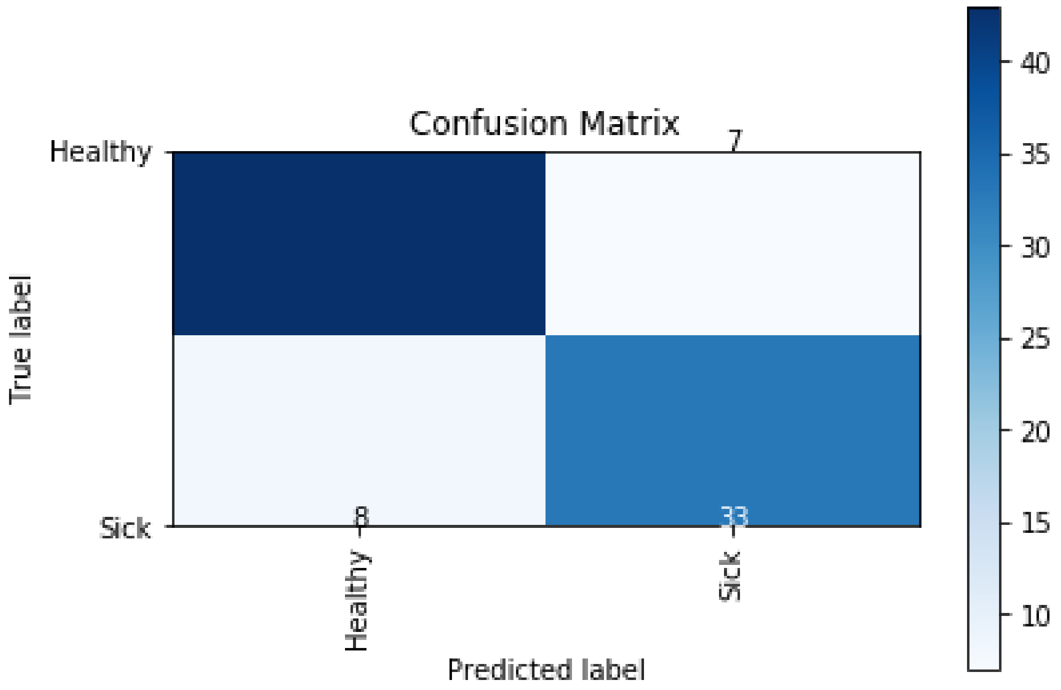
1.5.10 Question 3.1.8 Rerun your SVM, but now modify your model parameter kernel to equal 'linear'. Again report your Accuracy, Precision, Recall, F1 scores, and Confusion matrix and plot the new ROC curve.

[24]:
```
# SVM
svm = SVC(probability=True, gamma='auto', kernel='linear')
svm.fit(x_train, y_train)
svm_predicted = svm.predict(x_test)
score = svm.predict_proba(x_test)
```

[25]:
```
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(y_test, svm_predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(y_test,␣
 ↪svm_predicted)))
print("%-12s %f" % ('Recall:', metrics.recall_score(y_test, svm_predicted)))
print("%-12s %f" % ('F1 score:', metrics.f1_score(y_test, svm_predicted)))

print("Confusion Matrix for Data (Linear): \n")
draw_confusion_matrix(y_test, svm_predicted, ["Healthy","Sick"])
```

```
Accuracy:    0.835165
Precision:   0.825000
Recall:      0.804878
F1 score:    0.814815
Confusion Matrix for Data (Linear):
```



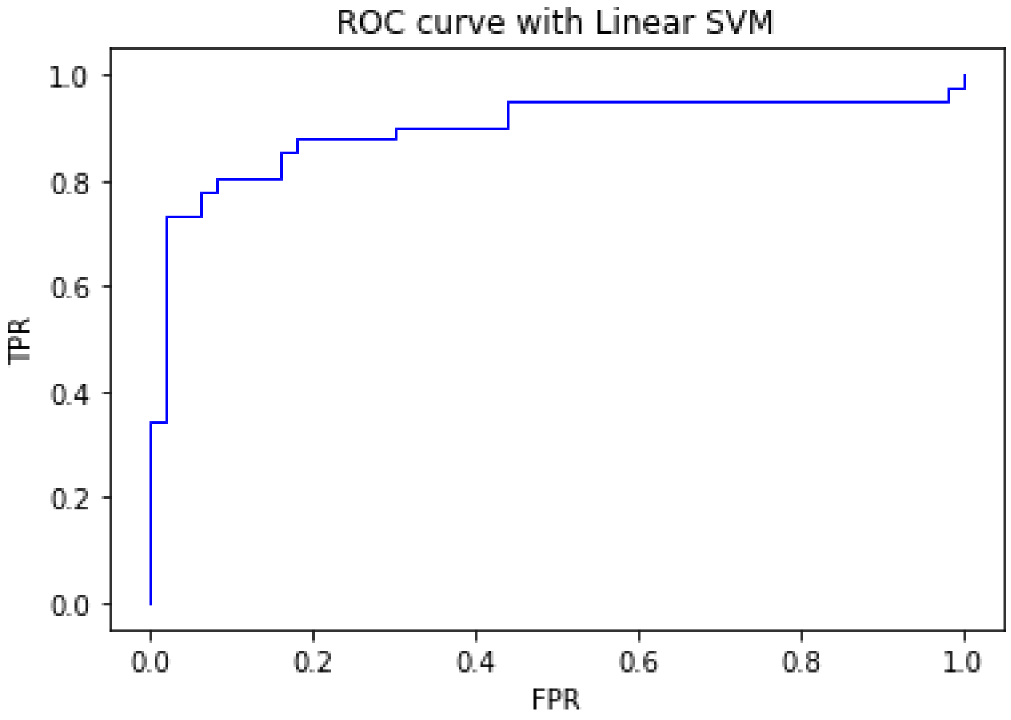
[26]:

```
print("Linear SVM Model Performance Results: \n")

fpr_svm, tpr_svm, thresholds = metrics.roc_curve(y_test, score[:, 1],␣
 ↪pos_label=1)

pyplot.figure(1)
pyplot.plot(fpr_svm, tpr_svm, color='blue', lw=1)
pyplot.title("ROC curve with Linear SVM")
pyplot.xlabel("FPR")
pyplot.ylabel("TPR")
pyplot.show()
```

```
Linear SVM Model Performance Results:
```



1.5.11 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.

Answer

- The new results show that my model performs slightly worse when the SVM model is forced to be linear.

- This might occur due to the fact that our data is not linearly separable and hence a linear model might not produce the best results when using a linear model.
- When a non-linear model is used, it can easily be fit into the data as now is can produce a non-linear “boundary” between the 2 categories of data better. Therefore, it performs better.

1.5.12 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.

1.5.13 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.

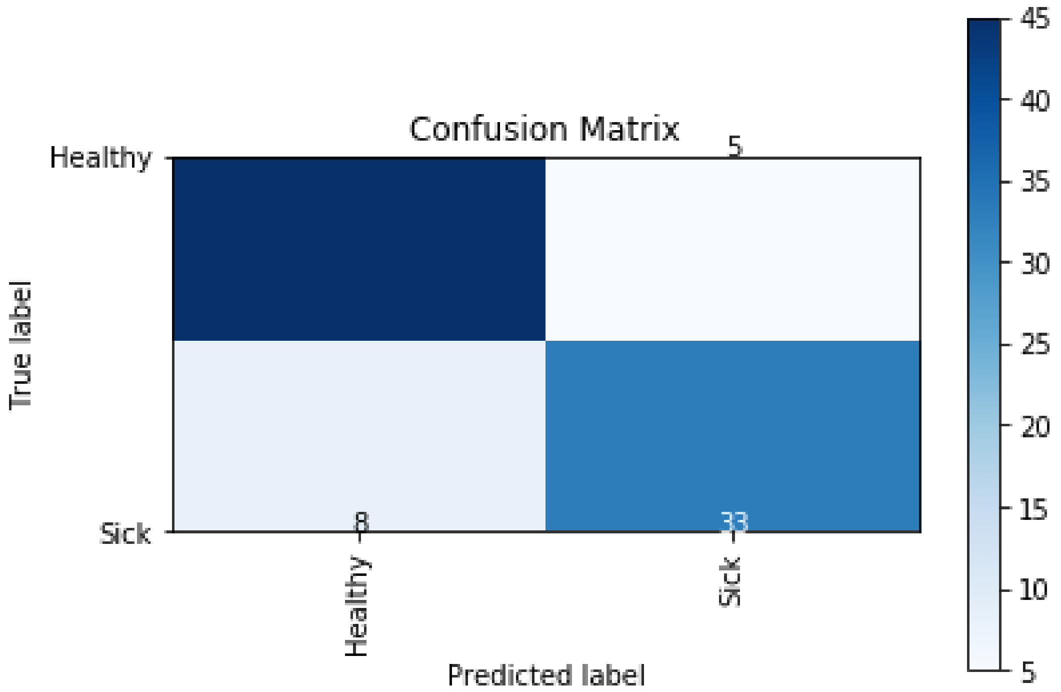
[27]:
```
# Logistic Regression
logistic = LogisticRegression(solver='sag', max_iter=10)
logistic.fit(x_train, y_train)
logistic_predicted = logistic.predict(x_test)
logistic_score = logistic.predict_proba(x_test)
```

```
C:\Users\param\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)
```

[28]:
```
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(y_test,␣
 ↪logistic_predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(y_test,␣
 ↪logistic_predicted)))
print("%-12s %f" % ('Recall:', metrics.recall_score(y_test,␣
 ↪logistic_predicted)))
print("%-12s %f" % ('F1 score:', metrics.f1_score(y_test, logistic_predicted)))

print("Confusion Matrix for Data (Logistic): \n")
draw_confusion_matrix(y_test, logistic_predicted, ["Healthy","Sick"])
```

```
Accuracy:    0.857143
Precision:   0.868421
Recall:      0.804878
F1 score:    0.835443
Confusion Matrix for Data (Logistic):
```



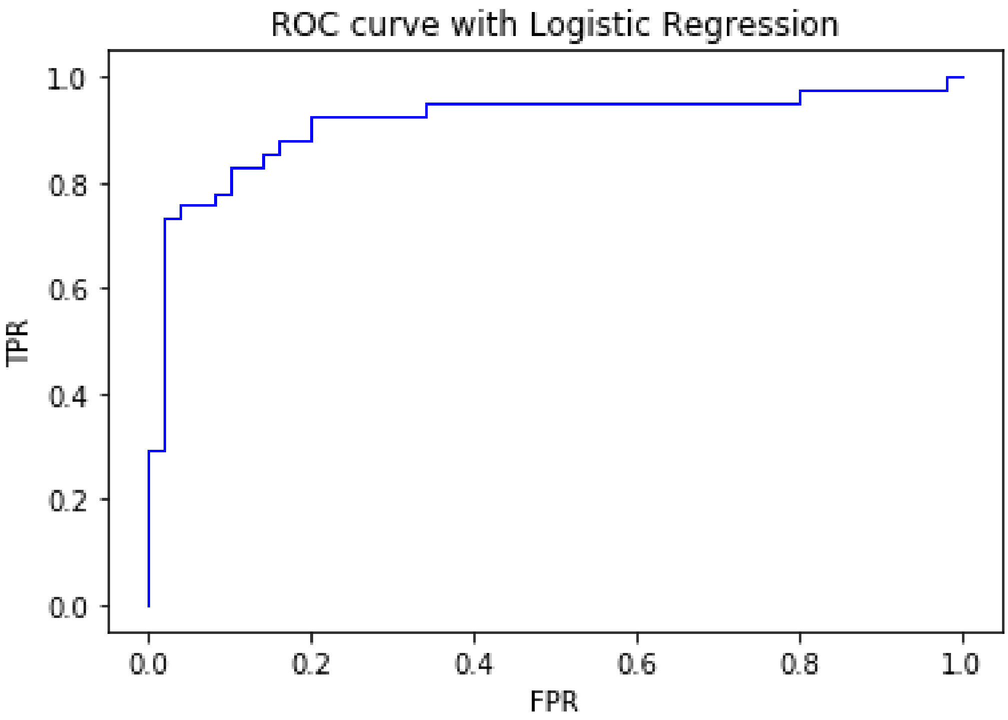
[29]:

```
print("Logistic Regression Model Performance Results: \n")

fpr_svm, tpr_svm, thresholds = metrics.roc_curve(y_test, logistic_score[:, 1],␣
 ↪pos_label=1)

pyplot.figure(1)
pyplot.plot(fpr_svm, tpr_svm, color='blue', lw=1)
pyplot.title("ROC curve with Logistic Regression")
pyplot.xlabel("FPR")
pyplot.ylabel("TPR")
pyplot.show()
```

```
Logistic Regression Model Performance Results:
```



1.5.14 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.

[30]:

```
# Logistic Regression
logistic = LogisticRegression(solver='sag', max_iter=1000)
logistic.fit(x_train, y_train)
logistic_predicted = logistic.predict(x_test)
logistic_score = logistic.predict_proba(x_test)
```

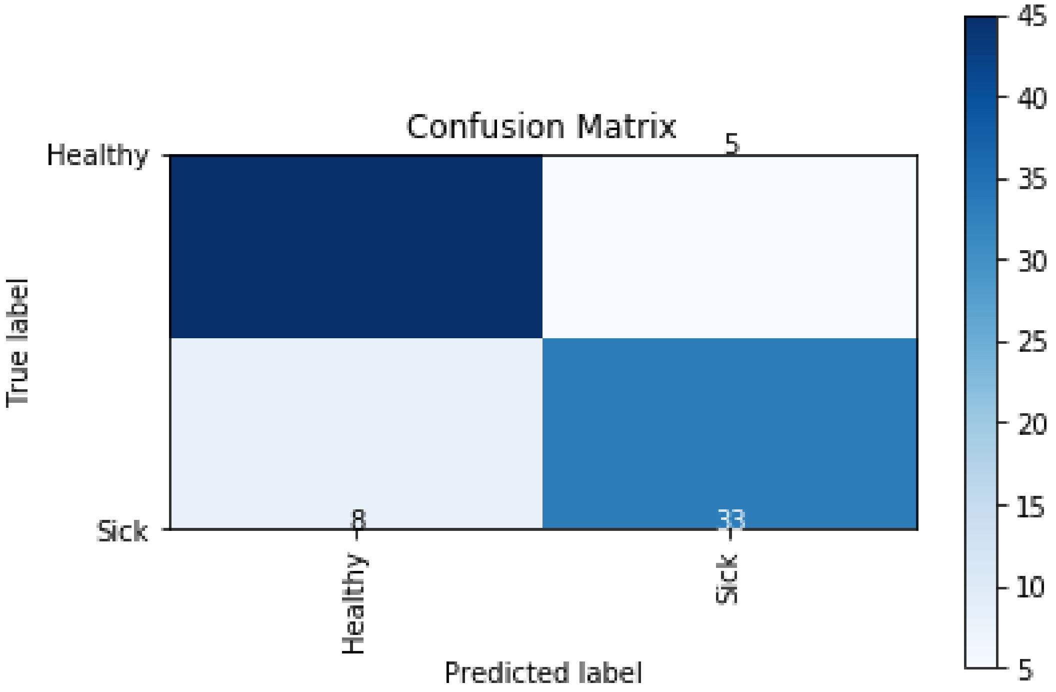
[31]:

```
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(y_test,␣
 ↪logistic_predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(y_test,␣
 ↪logistic_predicted)))
print("%-12s %f" % ('Recall:', metrics.recall_score(y_test,␣
 ↪logistic_predicted)))
print("%-12s %f" % ('F1 score:', metrics.f1_score(y_test, logistic_predicted)))

print("Confusion Matrix for Data (Logistic): \n")
draw_confusion_matrix(y_test, logistic_predicted, ["Healthy","Sick"])
```

```
Accuracy:    0.857143
```

```
Precision:    0.868421
Recall:       0.804878
F1 score:     0.835443
Confusion Matrix for Data (Logistic):
```

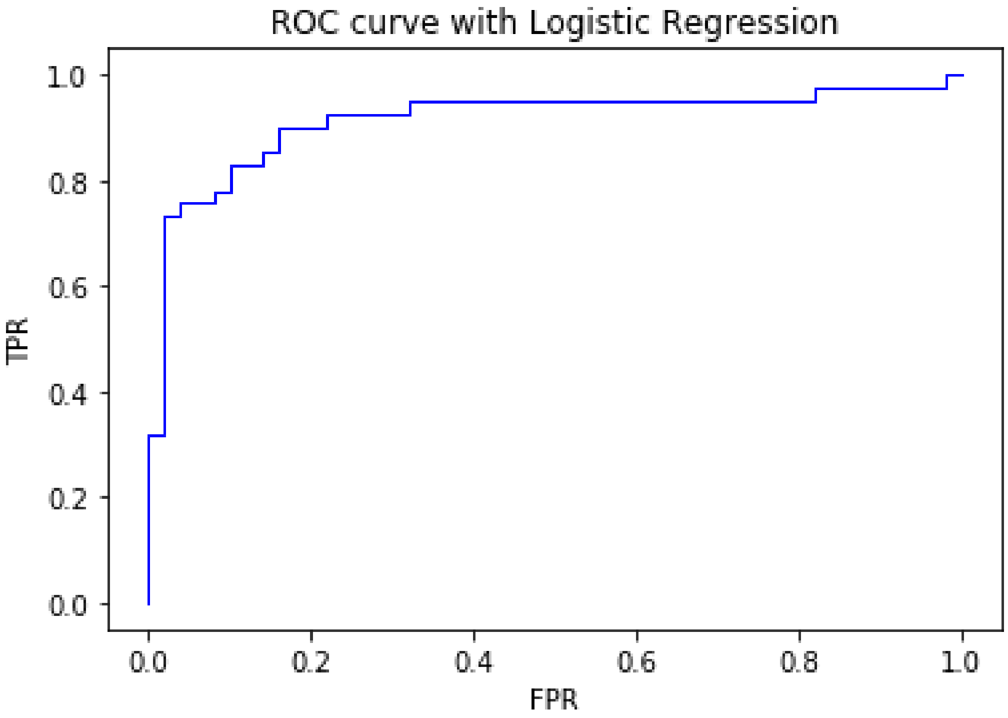


[32]:
```python
print("Logistic Regression Model Performance Results: \n")

fpr_svm, tpr_svm, thresholds = metrics.roc_curve(y_test, logistic_score[:, 1],␣
 ↪pos_label=1)

pyplot.figure(1)
pyplot.plot(fpr_svm, tpr_svm, color='blue', lw=1)
pyplot.title("ROC curve with Logistic Regression")
pyplot.xlabel("FPR")
pyplot.ylabel("TPR")
pyplot.show()
```

```
Logistic Regression Model Performance Results:
```



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

Answer

- In this model we get the error message "the coef_ did not converge" because a max_iter of value 10 is extremely small for a model to converge to its optimal value. Therefore, changing the max_iter value to be 1000 (or greater than that) makes the warning go away as now the model has sufficient number of iterations to converge.
- In our specific example however, changing the max_iter from 10 to a 1000 does not increase the performance of the model as this model gets a really good performance really fast. Therefore, it only takes a few iterations for it to converge and hence produces the same result.

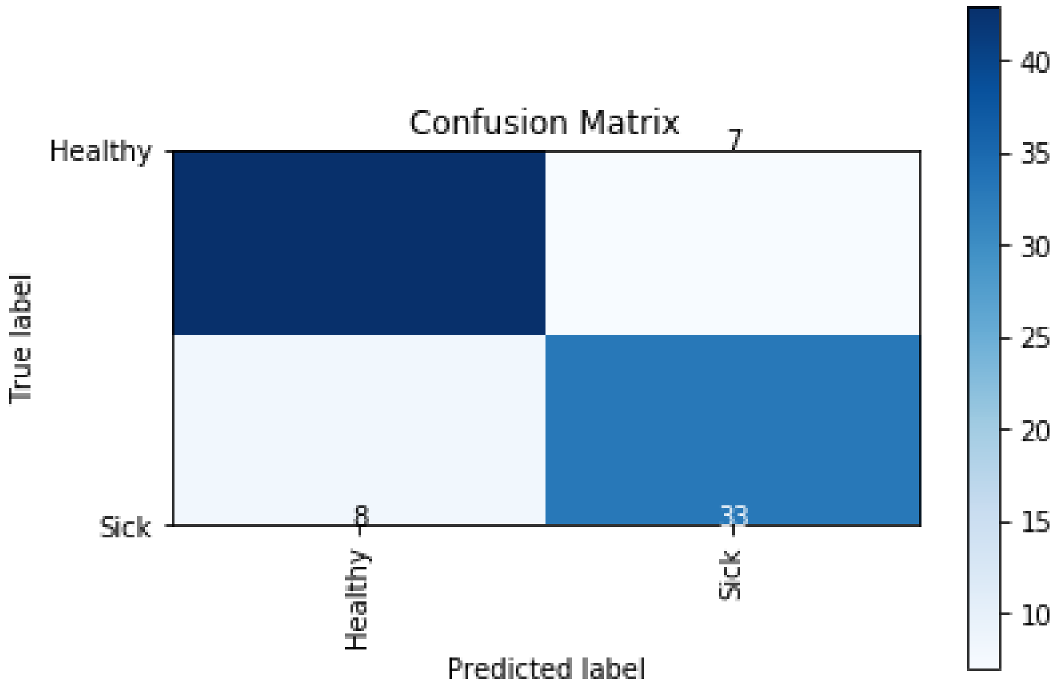
1.5.16 Question 3.2.4 Rerun your logistic classifier, but modify the penalty = 'none', solver='sag' and again report the results.

[33]:
```
# Logistic Regression
logistic = LogisticRegression(penalty='none', solver='sag', max_iter=100000)
logistic.fit(x_train, y_train)
logistic_predicted = logistic.predict(x_test)
logistic_score = logistic.predict_proba(x_test)
```

[34]:
```
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(y_test,␣
 ↪logistic_predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(y_test,␣
 ↪logistic_predicted)))
print("%-12s %f" % ('Recall:', metrics.recall_score(y_test,␣
 ↪logistic_predicted)))
print("%-12s %f" % ('F1 score:', metrics.f1_score(y_test, logistic_predicted)))

print("Confusion Matrix for Data (Logistic): \n")
draw_confusion_matrix(y_test, logistic_predicted, ["Healthy","Sick"])
```

```
Accuracy:    0.835165
Precision:   0.825000
Recall:      0.804878
F1 score:    0.814815
Confusion Matrix for Data (Logistic):
```

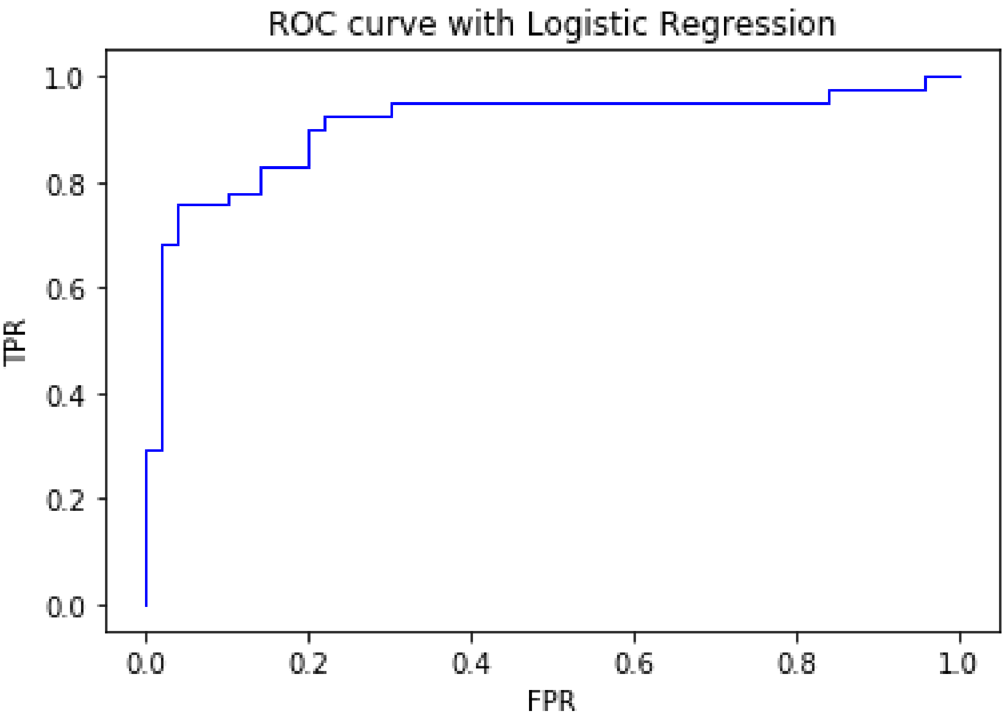


[35]:
```
print("Logistic Regression Model Performance Results: \n")

fpr_svm, tpr_svm, thresholds = metrics.roc_curve(y_test, logistic_score[:, 1],␣
 ↪pos_label=1)

pyplot.figure(1)
pyplot.plot(fpr_svm, tpr_svm, color='blue', lw=1)
pyplot.title("ROC curve with Logistic Regression")
pyplot.xlabel("FPR")
pyplot.ylabel("TPR")
pyplot.show()
```

```
Logistic Regression Model Performance Results:
```



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

Answer Our goal with the model is to minimize the loss function as much as possible. But, having an extremely low loss function means that we potentially overfit the data and therefore we regularize it to penalize high values and try to set a lower bound on our loss function. Therefore, what the penalty parameter essentially does is that it removes regularization (which could potential overfit).

The solver method, SAG or stochastic average gradient descent, is an optimisation/variation to the stochastic gradient descent (SG) alogrithm that makes it faster for cases where you have many datapoints + features. Like SG methods, the SAG method's iteration cost is independent of the number of terms in the sum. However, by incorporating a memory of previous gradient values the SAG method achieves a faster convergence rate than SG methods.

A combination of both produces an optimal outcome becuase both produce an optimisation to the existing model. Penalty parameter makes the model converge faster by having a lower bound and prevents overfitting while SAG also produces an optimization to the existing model and enables it to converge faster as well.

1.5.18 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?

Answer

Logistic Regression calculates the probabilities and models them on the sigmoid function. Depending on where the point falls on the function, the model classifies it accordingly. Whereas, SVM does a linear split in the data and depending on where the point lies, it classifies it accordingly

1.5.19 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.

1.5.20 K-Nearest Neighbor

1.5.21 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.

[36]:
```
# k-Nearest Neighbors algorithm
KNN = KNeighborsClassifier()
KNN.fit(x_train, y_train)
KNN_predicted = KNN.predict(x_test)
KNN_score = KNN.predict_proba(x_test)

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(y_test, KNN_predicted)))
```

```
Accuracy:    0.868132
```

1.5.22 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).

[37]:
```
print("%-12s %f" % ('Homogeneity Score:', metrics.homogeneity_score(y_test,␣
 ↪KNN_predicted)))
print("%-12s %f" % ('Completeness Score:', metrics.completeness_score(y_test,␣
 ↪KNN_predicted)))
print("%-12s %f" % ('V-Measure Score:', metrics.v_measure_score(y_test,␣
 ↪KNN_predicted)))
```

```
print("%-12s %f" % ('Adjusted Rand Score:', metrics.adjusted_rand_score(y_test,␣
 ↪KNN_predicted)))
print("%-12s %f" % ('Adjusted Mutual Information:',
                    metrics.adjusted_mutual_info_score(y_test, KNN_predicted,␣
 ↪average_method='arithmetic')))
```

```
Homogeneity Score: 0.434542
Completeness Score: 0.434542
V-Measure Score: 0.434542
Adjusted Rand Score: 0.536996
Adjusted Mutual Information: 0.429912
```

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

Answer

Homogeneity Score:

- This score is useful to check whether the clustering algorithm meets an important requirement: a cluster should contain only samples belonging to a single class. It's bounded between 0 and 1, with low values indicating a low homogeneity.
- According to me, the homogeneity score of ~0.43 means that our data has low homogeneity and therefore the clusters are not completely separate but kind of mixed together.

Completeness Score:

- Its purpose is to provide a piece of information about the assignment of samples belonging to the same class. More precisely, a good clustering algorithm should assign all samples with the same true label to the same cluster.
- Therefore, a completeness score of 0.43 means that the model is not complete in the sense that it has not clustered classes that completely correlates to it true labels

V-measure Score:

- The V-measure is the harmonic mean between homogeneity and completeness
- This metric is independent of the absolute values of the labels: a permutation of the class or cluster label values won't change the score value in any way.

Adjusted Rand Score:

- The Rand score is a measure of the similarity between two data clusterings. A form of the Rand index that is adjusted for the chance grouping of elements, is called the adjusted Rand score. From a mathematical standpoint, Rand score is related to the accuracy, but is applicable even when class labels are not used.

- The adjusted Rand index is ensured to have a value close to 0.0 for random labeling independently of the number of clusters and samples and exactly 1.0 when the clusterings are identical.
- The score of 0.53 means that it is alsmost adjusted and perfectly matching but it has incorrect elements and thus penalized

Adjusted Mutual Information:

- Adjusted mutual information is be used for comparing clusterings. It corrects the effect of agreement solely due to chance between clusterings, similar to the way the adjusted rand score corrects the Rand score.
- Adjusted Mutual Information (AMI) is an adjustment of the Mutual Information (MI) score to account for chance. It accounts for the fact that the MI is generally higher for two clusterings with a larger number of clusters, regardless of whether there is actually more information shared.

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.

1.5.24 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).

[38]:

```
l = [1, 2, 3, 5, 10, 20, 50, 100]

for i in l:
    KNN = KNeighborsClassifier(n_neighbors=i)
    KNN.fit(x_train, y_train)
    KNN_predicted = KNN.predict(x_test)
    KNN_score = KNN.predict_proba(x_test)

    print("Number of neighbors:", i)
    print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(y_test,␣
 ↪KNN_predicted)))
    print("%-12s %f" % ('Homogeneity Score:', metrics.homogeneity_score(y_test,␣
 ↪KNN_predicted)))
    print("%-12s %f" % ('Completeness Score:', metrics.
 ↪completeness_score(y_test, KNN_predicted)))
```

```
    print("%-12s %f" % ('V-Measure Score:', metrics.v_measure_score(y_test,␣
↪KNN_predicted)))
    print("%-12s %f" % ('Adjusted Rand Score:', metrics.
↪adjusted_rand_score(y_test, KNN_predicted)))
    print("%-12s %f" % ('Adjusted Mutual Information:',
                        metrics.adjusted_mutual_info_score(y_test,
                                                           KNN_predicted,␣
↪average_method='arithmetic')))
    print("")
```

```
Number of neighbors: 1
Accuracy:    0.747253
Homogeneity Score: 0.180080
Completeness Score: 0.180717
V-Measure Score: 0.180398
Adjusted Rand Score: 0.236145
Adjusted Mutual Information: 0.173675

Number of neighbors: 2
Accuracy:    0.802198
Homogeneity Score: 0.298268
Completeness Score: 0.327995
V-Measure Score: 0.312426
Adjusted Rand Score: 0.358337
Adjusted Mutual Information: 0.306507

Number of neighbors: 3
Accuracy:    0.835165
Homogeneity Score: 0.364809
Completeness Score: 0.362262
V-Measure Score: 0.363531
Adjusted Rand Score: 0.443216
Adjusted Mutual Information: 0.358339

Number of neighbors: 5
Accuracy:    0.868132
Homogeneity Score: 0.434542
Completeness Score: 0.434542
V-Measure Score: 0.434542
Adjusted Rand Score: 0.536996
Adjusted Mutual Information: 0.429912

Number of neighbors: 10
Accuracy:    0.846154
Homogeneity Score: 0.377564
Completeness Score: 0.384636
V-Measure Score: 0.381067
```

```
Adjusted Rand Score: 0.473520
Adjusted Mutual Information: 0.375949

Number of neighbors: 20
Accuracy:    0.890110
Homogeneity Score: 0.501223
Completeness Score: 0.510611
V-Measure Score: 0.505874
Adjusted Rand Score: 0.604407
Adjusted Mutual Information: 0.501787

Number of neighbors: 50
Accuracy:    0.868132
Homogeneity Score: 0.435961
Completeness Score: 0.444126
V-Measure Score: 0.440005
Adjusted Rand Score: 0.537010
Adjusted Mutual Information: 0.435374

Number of neighbors: 100
Accuracy:    0.857143
Homogeneity Score: 0.415335
Completeness Score: 0.432549
V-Measure Score: 0.423767
Adjusted Rand Score: 0.504794
Adjusted Mutual Information: 0.418946
```

Optimal Results The most optimal result in terms of clustering is with 20 neighbours (out of the tested variables). This makes sense as it kind of balances considering too few neighbours and considering too many neighbours.

I feel that with 1, 5, and 10 number of neighbours, we don't get good results because the points on the border or near the borders of the 2 clusters might be skewed one way or the other. For example, even though a data point might belong to the sick class, since it is near more healthy data points, it will be classified incorrectly.

A similar case occurs with considering 50 or 100 neighbours. I feel that 50 and 100 are too big of a number on our small dataset and therefore we take into consideration points that may be very far away from the actual point. Therefore, this results in low scores.

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

Answer

- Generally clustering algorithms are most effective when the dataset is nicely separated into its classes. Usually, when the data has clusters that kind of merge together or don't have a fine line of separation, they don't perform well in classification/clustering algorithms. We could essentially create a model that perfectly separates our 2 clusters in our test data but this might lead to overfitting and our model won't be generalisable.

- Unlike in SVM and logistic regression, where creating a fine line of separation is difficult, models like KNN perfom relatively better as the algorithm does not involve creating such a boundary. Rather, you just look at the k-nearest neighbours and based on the occurences of different classes, you decide which class to classify the point into. This reduces the false predictions that our model can possibly make around the boundary of different classes.
- Therefore, KNN comparatively performs better than SVM and logistic regression.