cs188_project2-Final

February 23, 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

[90]:
```
#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)
```

[35]:
```
# 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)
```

[36]:
```
# 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 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.)

[37]:
```
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.

[38]:
```
data.head(5)
```

[38]:
```
   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
```

[39]:
```
data.describe()
```

[39]:
```
              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
```

[7]:
```
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
```

[8]:
```
data.isnull().sum()
```

[8]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
sick        0
dtype: int64
```

1.3.2 Question 1.3 Sometimes data will be stored in different formats (e.g., string, date, boolean), but many learning methods work strictly on numeric inputs. Call the info method to determine the datafield type for each column. Are there any that are problemmatic and why?

[Use this area to describe any fields you believe will be problemmatic and why] E.g., All the columns in our dataframe are numeric (either int or float), however our target variable 'sick' is a boolean and may need to be modified.

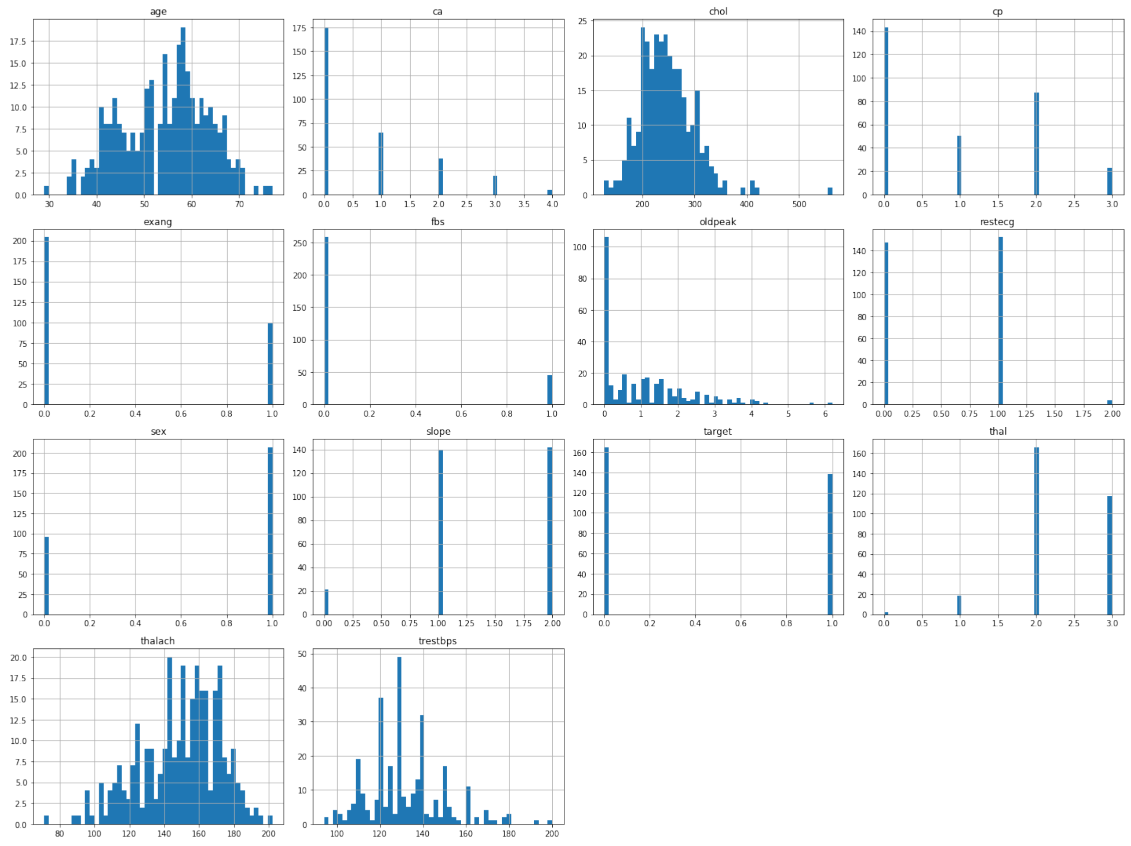
1.3.3 Question 1.4 Before we begin our analysis we need to fix the field(s) that will be problematic. Specifically convert our boolean sick variable into a binary numeric target variable (values of either '0' or '1'), and then drop the original sick datafield from the dataframe.

[40]:
```
data['target'] = (data['sick']).astype(int)
del data['sick']
```

1.3.4 Question 1.5 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?

[41]:
```
data.hist(bins=50, figsize=(20,15))
save_fig("attribute_histogram_plots")
plt.show()
```

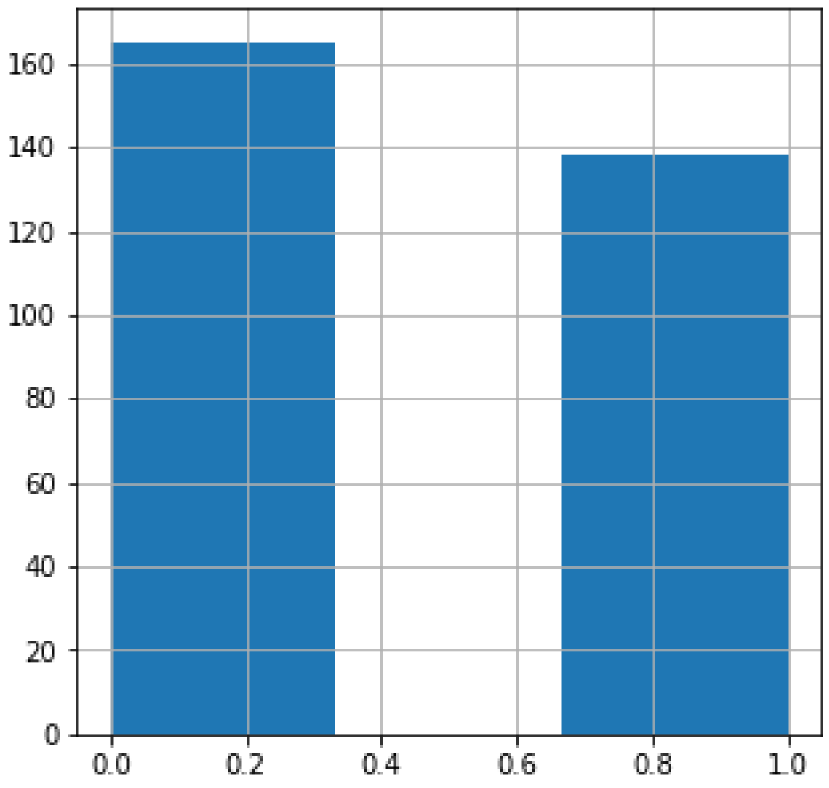
```
Saving figure attribute_histogram_plots
```



1.3.5 Question 1.6 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:

[42]:
```
data['target'].hist(bins=3, figsize=(5,5))
data['target'].value_counts()
```

[42]:
```
0    165
1    138
Name: target, dtype: int64
```



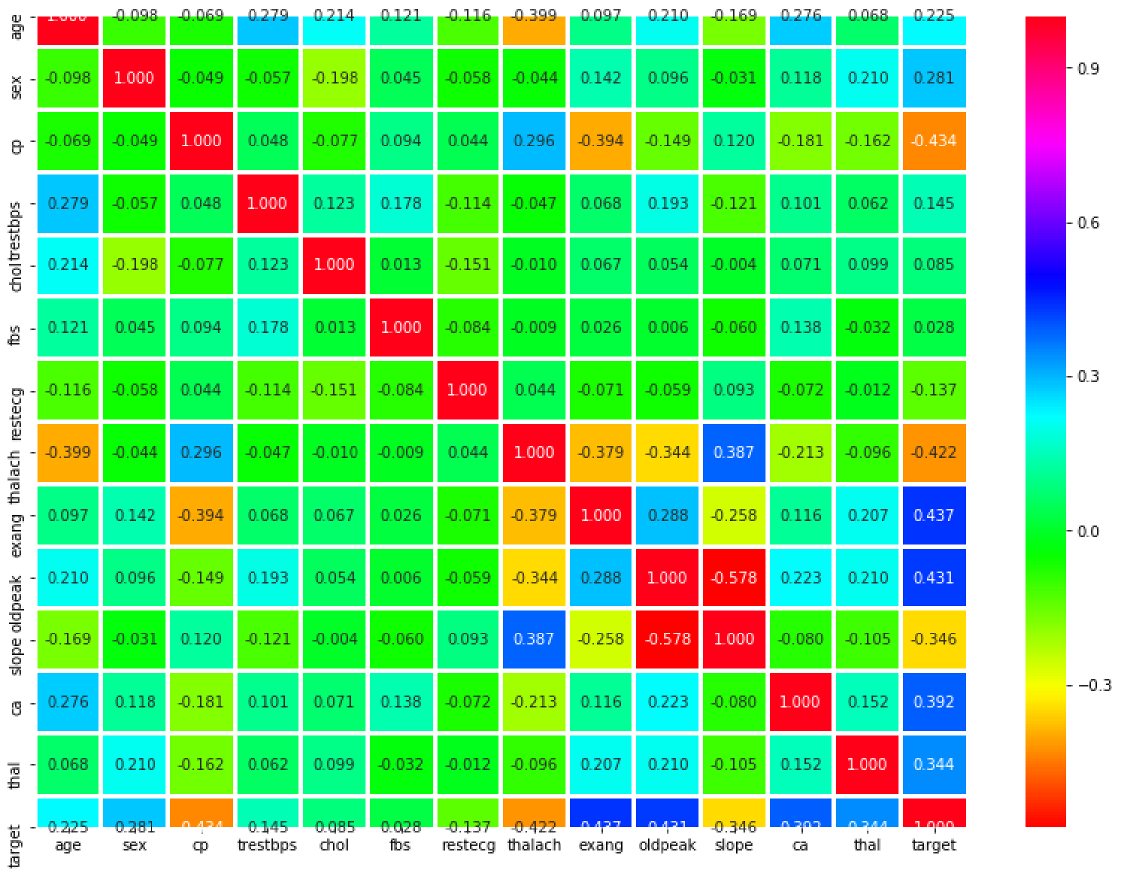
[Include description of findings here] E.g., As we can see, our sample contains 165 healthy individuals and 138 sick individuals, which reflects a sufficiently large and unbiased sampling

1.3.6 Question 1.7 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.

[Discuss prompt here] E.g., artificially inflating numbers to balance a dataset may result in overfitting. Concurrently showing statistically uncommon events as likely may result in a classifier overpredicting it in real life.

1.3.7 Question 1.8 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)?

```
[43]: plt.figure(figsize=(14,10))
      sns.heatmap(data.corr(),annot=True,cmap='hsv',fmt='.3f',linewidths=2)
      plt.show()
```



[Discuss correlations here] E.g., We find the the strongest direct correlation between the presence of exercise induced angina (also a binary), and depression induced by exercise relative to rest indicates a strong direct correlation. Both of these are understandable as heart failure under conditions of duress is a clear indication of heart disease. Conversely, maximum heart rate achieved is inversely correlated, likely as a healthy heart is unable to achieve a high heart rate.

1.4 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.

```
[44]: target_name = 'target'
      data_target = data[target_name]
      data = data.drop([target_name], 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.

```
[45]: train_raw, test_raw, target_raw, target_raw_test = train_test_split(data,␣
       ↪data_target, test_size=0.3, random_state=0)
```

```
[46]: train_raw
```

```
[46]:      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
      137   62    1   1       128   208    1        0      140      0      0.0
      106   69    1   3       160   234    1        0      131      0      0.1
      284   61    1   0       140   207    0        0      138      1      1.9
      44    39    1   2       140   321    0        0      182      0      0.0
      139   64    1   0       128   263    0        1      105      1      0.2
      ..   …  …  ..     …   …  …     …     …      …
      251   43    1   0       132   247    1        0      143      1      0.1
      192   54    1   0       120   188    0        1      113      0      1.4
      117   56    1   3       120   193    0        0      162      0      1.9
      47    47    1   2       138   257    0        0      156      0      0.0
      172   58    1   1       120   284    0        0      160      0      1.8

           slope  ca  thal
      137      2   0     2
      106      1   1     2
      284      2   1     3
```

```
44       2   0     2
139      1   1     3
..     ...  ..   ...
251      1   4     3
192      1   1     3
117      1   0     3
47       2   0     2
172      1   0     2

[212 rows x 13 columns]
```

[48]:
```python
print (train_raw.shape, target_raw.shape)
print (test_raw.shape, target_raw_test.shape)
```

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

1.4.3 Question 2.3 pipeline

[49]:
```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
heart_num = data.drop(['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca',␣
 ↪'thal'], axis=1)



    # remove categorical features from the numeric values

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

heart_num_tr = num_pipeline.fit_transform(heart_num)
numerical_features = list(heart_num)
categorical_features = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca',␣
 ↪'thal']

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

heart_prepared = full_pipeline.fit_transform(data)
```

[50]:
```python
heart_prepared
```

```
[50]: 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.        ]])
```

```
[53]: train, test, target, target_test = train_test_split(heart_prepared,␣
       ↪data_target, test_size=0.3, random_state=0)
```

```
[54]: train
```

```
[54]: array([[ 0.84190778, -0.20696359, -0.73947076, …,  0.        ,
          1.        ,  0.        ],
        [ 1.61392956,  1.62064933, -0.23700823, …,  0.        ,
          1.        ,  0.        ],
        [ 0.73161895,  0.47839125, -0.75879625, …,  0.        ,
          0.        ,  1.        ],
        …,
        [ 0.18017482, -0.66386682, -1.029353  , …,  0.        ,
          0.        ,  1.        ],
        [-0.81242462,  0.36416545,  0.20747787, …,  0.        ,
          1.        ,  0.        ],
        [ 0.40075247, -0.66386682,  0.72926589, …,  0.        ,
          1.        ,  0.        ]])
```

```
[55]: test
```

```
[55]: array([[ 1.72421839,  0.76395577, -1.39653716, …,  0.        ,
          0.        ,  1.        ],
        [ 1.06248543,  2.19177836, -0.3722866 , …,  0.        ,
          0.        ,  1.        ],
        [ 0.5110413 ,  2.19177836,  0.80656782, …,  0.        ,
          0.        ,  1.        ],
        …,
        [-0.37126932, -0.3783023 , -0.02442792, …,  0.        ,
          1.        ,  0.        ],
        [ 1.39335191, -0.66386682, -0.17903178, …,  0.        ,
          1.        ,  0.        ],
        [ 2.49624017, -0.3783023 ,  1.11577554, …,  0.        ,
          1.        ,  0.        ]])
```

[56]:
```
print (train.shape, target.shape)
print (test.shape, target_test.shape)
```

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

1.5 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 dataframe. Review the SVM Documentation for how to implement a model. For this implementation you can simply use the default settings, but set probability = True.

[59]:
```
# SVM

svm = SVC(probability=True)
svm.fit(train_raw, target_raw)
testing_result = svm.predict(test_raw)
predicted = svm.predict(test_raw)
score = svm.predict_proba(test_raw)
```

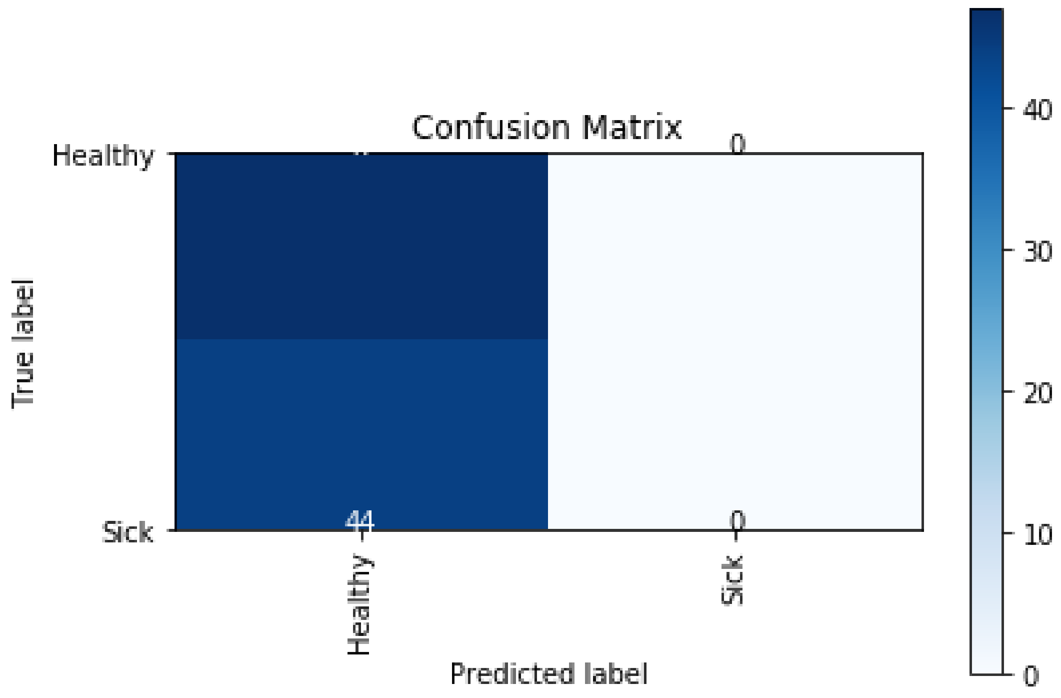
```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly to
'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

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

[60]:
```
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test,␣
 ↪predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(target_test,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
```

```
print("%-12s %f" % ('Recall:', metrics.recall_score(target_test, predicted,␣
↪labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(target_test, predicted,␣
↪labels=None, pos_label=1, average='binary', sample_weight=None)))
print("Confusion Matrix: \n")
draw_confusion_matrix(target_test, predicted, ['Healthy', 'Sick'])
```

```
Accuracy:    0.516484
Precision:   0.000000
Recall:      0.000000
F1 Score:    0.000000
Confusion Matrix:


/opt/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
/opt/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: F-score
is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
```



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.

[Provide explanation for each measure here] E.g., Accuracy is one metric for evaluating classification models. Accuracy is the fraction of predictions our model got right, or number of correct predictions/total number of predictions

Precision explores what proportion of positive identifications was actually correct, and is a ratio of the number of correct possitives over the total number of positive guesses (correct and otherwise).

Recall explores what proportion of actual positives was identified correctly, and is the proportion of correct positives over correct positives and false negatives (i.e., the total number of positives that should have been classified.

F1 Score is a measure that combines precision and recall and is the harmonic mean of both precision and recall. It is a more balanced measure that factors both precision and recall together.

Confusion Matrix is a table used to describe the performance of a classifier model showing how often data was labeled both correctly and incorrectly, and the number of each type of correct and incorrect classification.

Each score tells us important information. While accuracy gives you an indication of the overall performance, and if you're dealing with a well-balanced dataset (like ours is), it can be pretty reflective, however it can be misleading about performance. Here our confusion matrix sheds light on the problem. Our classifier ended up classifying everyone as healthy regardless of their actual state. The resulting accuracy was simply because about half our sample is healthy.

Precision is a measure more focused on 'false positives' and how often our model incorrectly predicts something. For cases like using facial recognition to ID a criminal, a false positive could be very harmful to a person who is misidentified, and we may want build a model that is highly sensitive to avoiding false positives (i.e., it may miss a lot of correct guesses, but when it does guess, it virtually always gets it right.

Recall is the opposite problem. Here we're worried about false negatives, and overlooking potential positives. Disease diagnosis is a great example of this. Here we'd rather flag a person as potentially worrisome, and through subsequent testing rule it out, then disregard it entirely.

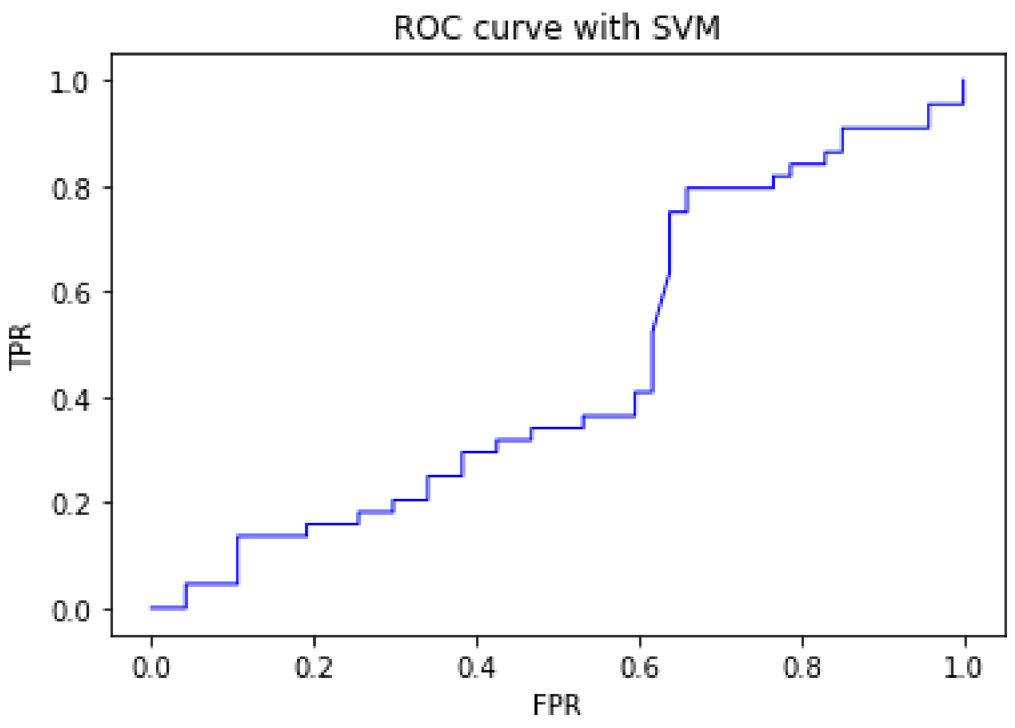
And F1 Score is a balance of the two. When you want generally high performance across the board minimizing both error-types, an F1 Score is the best generalized measure for it. Here we see that balancing the precision and recall results we arrive at a medicore F1 score.

Finally a confusion matrix is incredibly useful as it shows us not only the magnitude of errors, but the precise composition of those errors. It allows us to further refine our model to avoid certain errors or outcomes. In this case we realize that our classifier uniformly classified everything as healthy and that's why we got the results we did.

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

[61]:
```
print("SVM Model Performance Results:\n")

fpr_svm, tpr_svm, thresholds = metrics.roc_curve(target_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:
```



[Describe what an ROC Curve is and what the results mean here] The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. 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, so the areas under ROC curves are used to compare the usefulness of tests. Here we see a relatively low area under the curve indicating a poorly performing model.

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

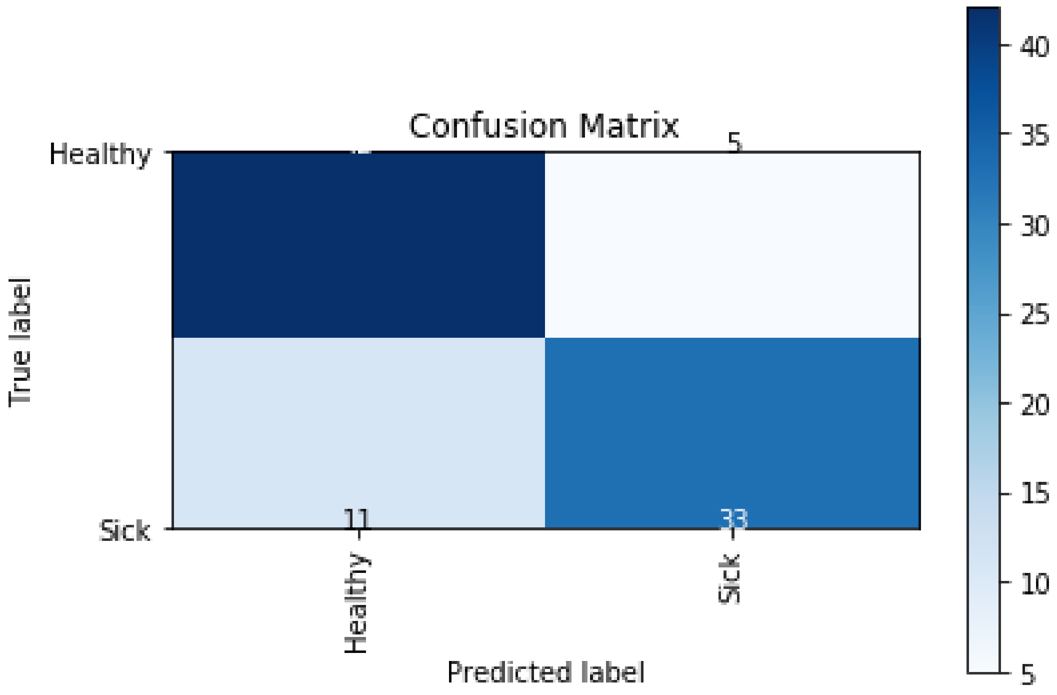
[62]:
```python
# SVM

svm = SVC(probability=True)
svm.fit(train, target)
testing_result = svm.predict(test)
predicted = svm.predict(test)
score = svm.predict_proba(test)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly to
'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

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.

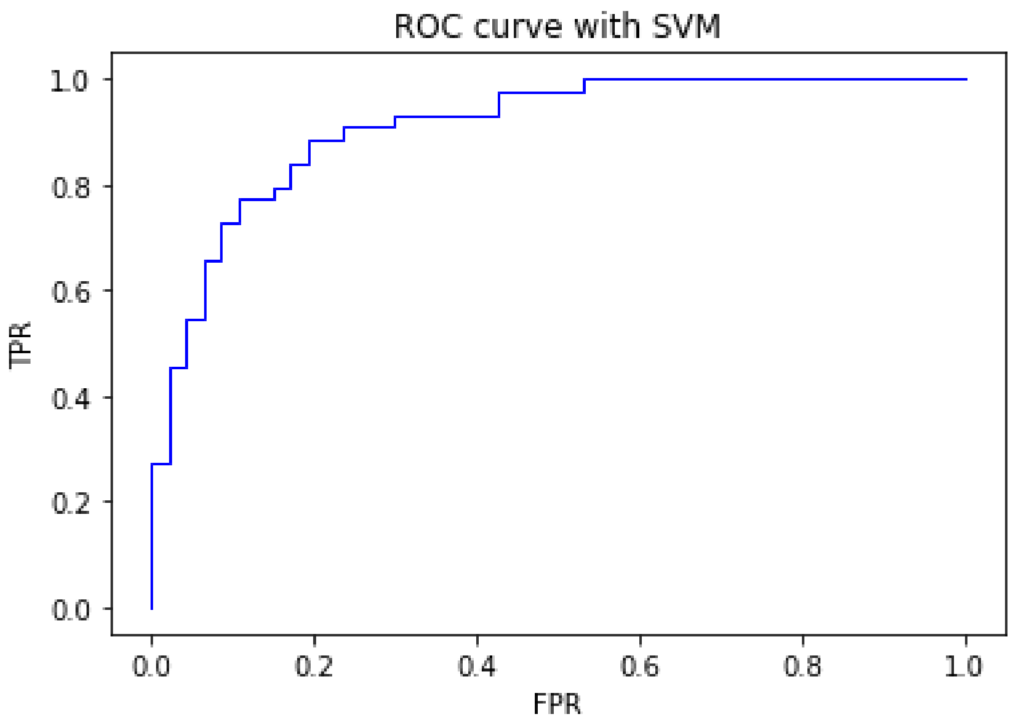
[63]:
```python
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test,␣
 ↪predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(target_test,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(target_test, predicted,␣
 ↪labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(target_test, predicted,␣
 ↪labels=None, pos_label=1, average='binary', sample_weight=None)))
print("Confusion Matrix: \n")
draw_confusion_matrix(target_test, predicted, ['Healthy', 'Sick'])
```

```
Accuracy:    0.824176
Precision:   0.868421
Recall:      0.750000
F1 Score:    0.804878
Confusion Matrix:
```



[64]:
```
print("SVM Model Performance Results:\n")

fpr_svm, tpr_svm, thresholds = metrics.roc_curve(target_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:
```

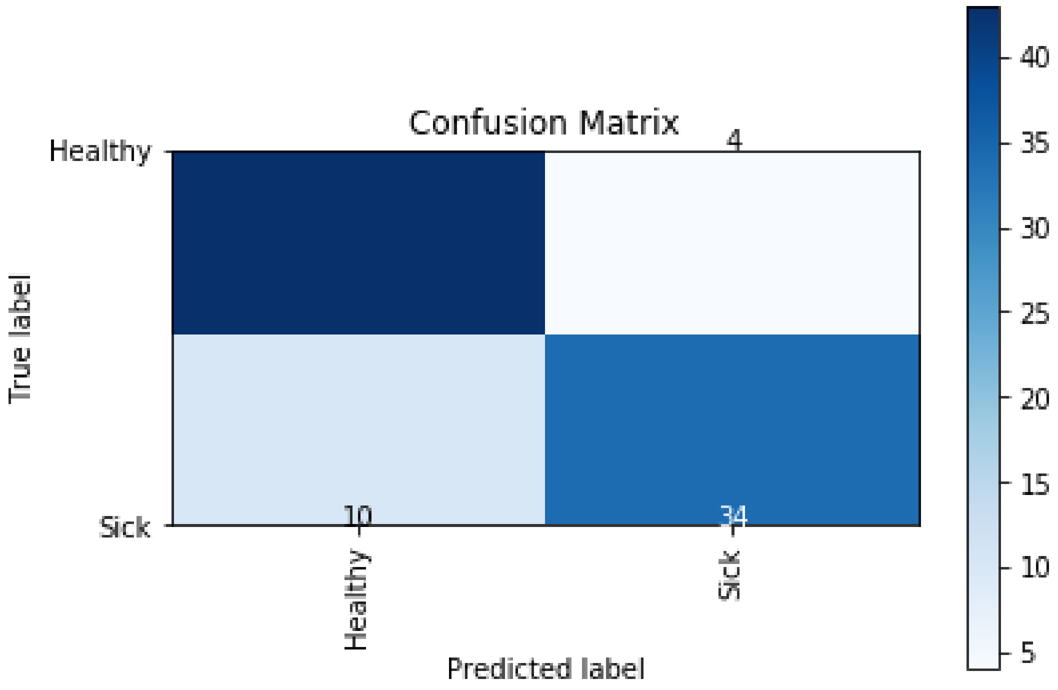


1.5.9 Question 3.1.5 Rerun your SVM, but now modify your model parameter kernel to equal 'linear'. Again report your Accuracy, Precision, Recall, F1 scores, and Confusion matrix and plot the new ROC curve.

[65]:
```python
# SVM

svm = SVC(kernel='linear', probability=True)
svm.fit(train, target)
testing_result = svm.predict(test)
predicted = svm.predict(test)
score = svm.predict_proba(test)
```

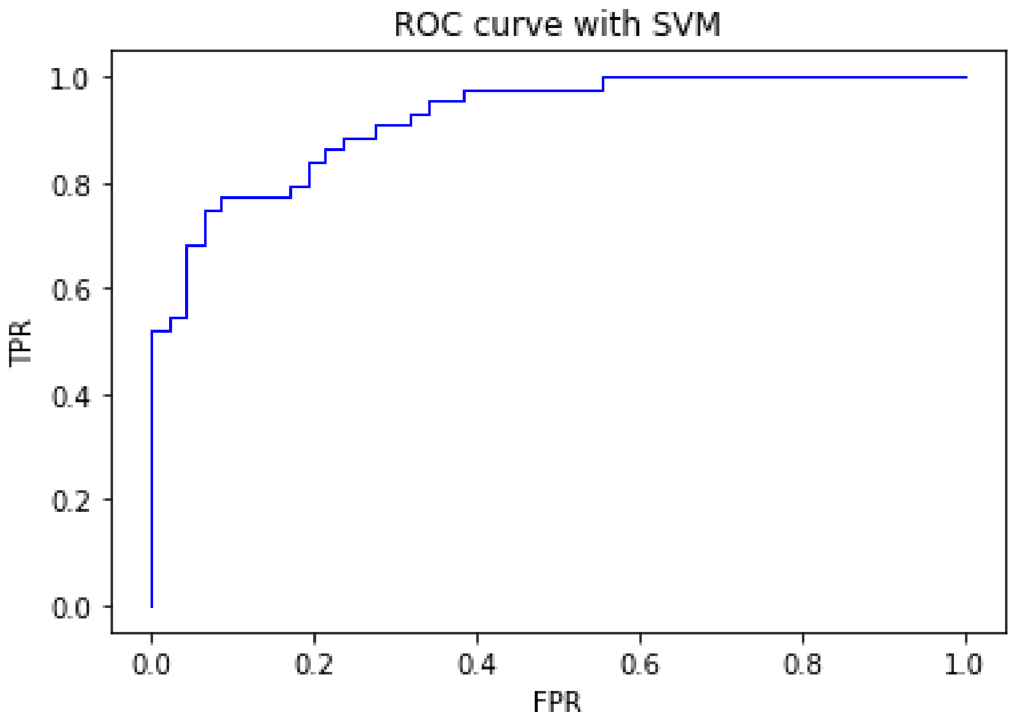
[66]:
```python
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test,␣
 ↪predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(target_test,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(target_test, predicted,␣
 ↪labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(target_test, predicted,␣
 ↪labels=None, pos_label=1, average='binary', sample_weight=None)))
print("Confusion Matrix: \n")
draw_confusion_matrix(target_test, predicted, ['Healthy', 'Sick'])
```

```
Accuracy:     0.846154
Precision:    0.894737
Recall:       0.772727
F1 Score:     0.829268
Confusion Matrix:
```



[67]:
```
print("SVM Model Performance Results:\n")

fpr_svm, tpr_svm, thresholds = metrics.roc_curve(target_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:
```



1.5.10 Question 3.1.6 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.

All scores are much higher indicating that the model has achieved much better success classifying the data. The ROC curve now demonstrates the a significant area under the curve indicating that it is an optimally performing model.

By changing the kernel value, we've changed the kernel type to be used in the algorithm to a linear kernel. The default value is a Radial Basis Function or RBF. RBF is a nonlinear kernels which is optimal for non-linear problems, while linear SVMs (or logistic regression) are optimal for linear problems. This indicates that our data is linearly configured and generally linear classifiers will perform well on them.

1.5.11 Logistic Regression

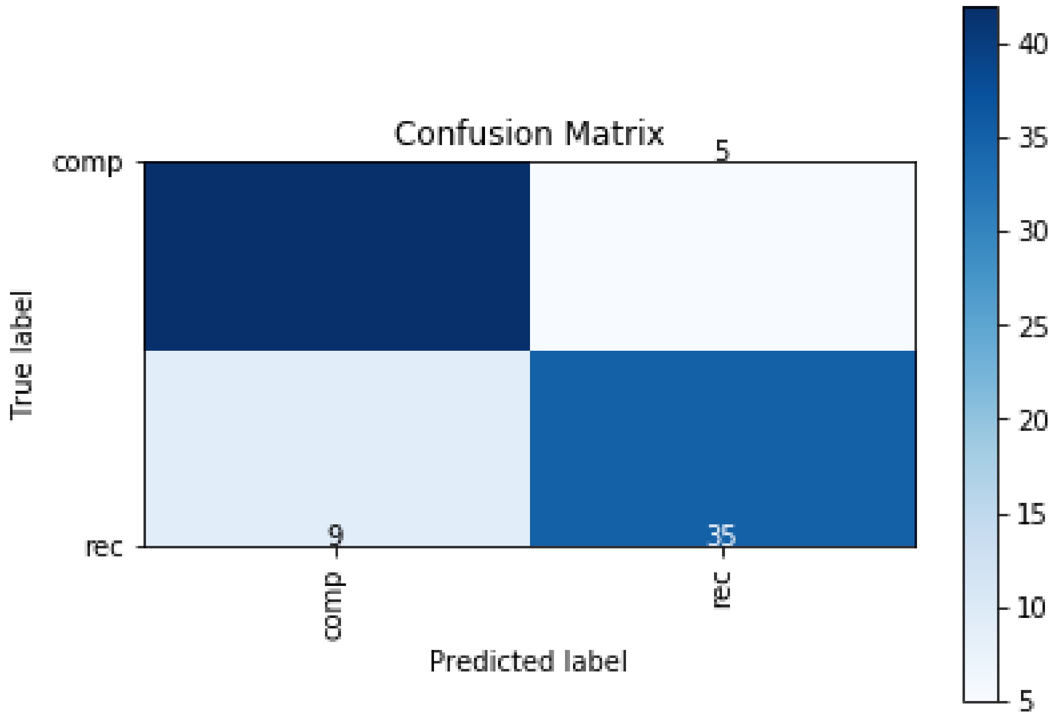
Knowing that we're dealing with a linearly configured dataset, let's now try another classifier that's well known for handling linear models: Logistic Regression. Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable.

1.5.12 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'. Report on the same four metrics as the SVM and graph the resulting ROC curve.

[80]:
```
# Logistic Regression
train_data_category = target
test_data_category = target_test

log_reg = LogisticRegression(solver='sag', max_iter= 10)
log_reg.fit(train, target)
predicted = log_reg.predict(test)
score = log_reg.predict_proba(test)[:,1]

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(test_data_category,␣
 ↪predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("Confusion Matrix: \n", metrics.confusion_matrix(test_data_category,␣
 ↪predicted))
draw_confusion_matrix(target_test, predicted, ['comp', 'rec'])

fpr_log_reg, tpr_log_reg, thresholds = metrics.roc_curve(test_data_category,␣
 ↪score)

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

pyplot.figure(1)
pyplot.plot(fpr_log_reg, tpr_log_reg, color='orange', lw=1)
pyplot.title("ROC curve with Logistic Regression")
pyplot.xlabel('FPR')
pyplot.ylabel('TPR')
```

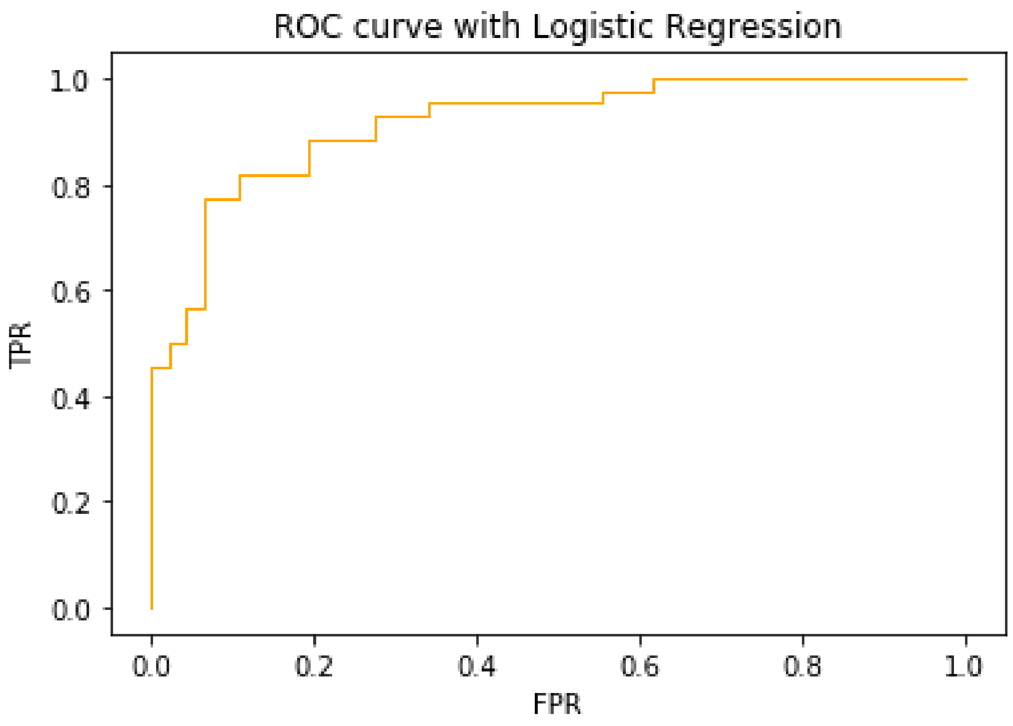
```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/sag.py:337:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
Accuracy:    0.846154
Precision:   0.875000
Recall:      0.795455
F1 Score:    0.833333
Confusion Matrix:
 [[42  5]
[ 9 35]]
```



```
Logistic Model Performance Results:
```

```
[80]: Text(0, 0.5, 'TPR')
```

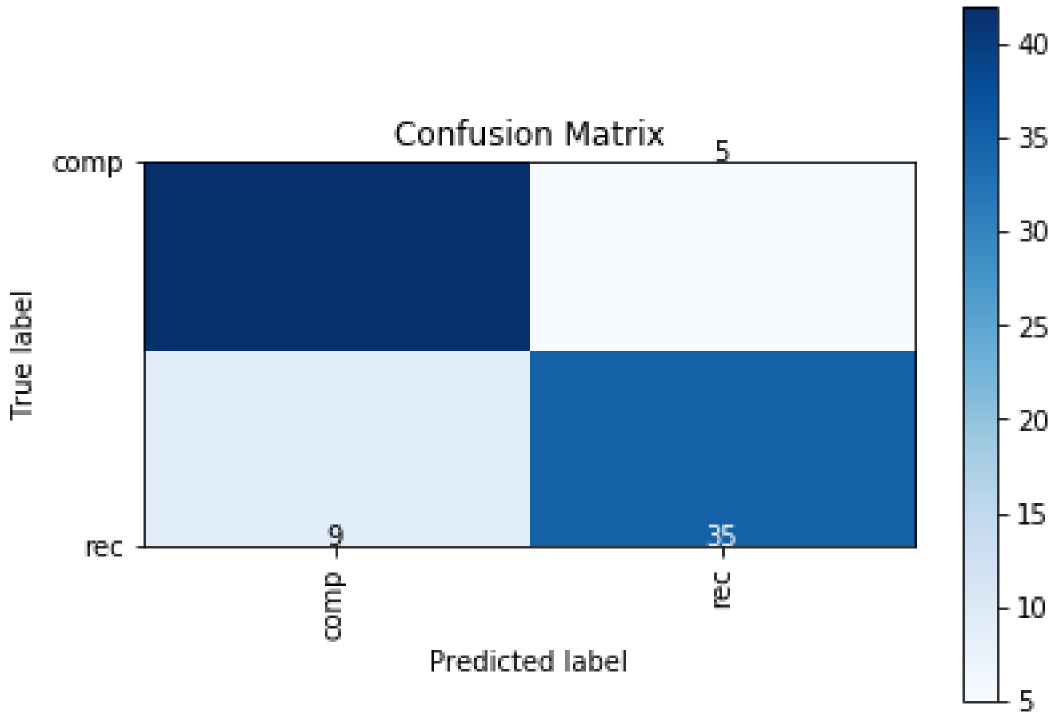


1.5.13 Question 3.2.2 Did you notice that when you ran the previous model you got the following warning: "ConvergenceWarning: The max_iter was reached which means the coef_ did not converge". Check the documentation and see if you can implement a fix for this problem, and again report your results.

```
[73]: # Logistic Regression
train_data_category = target
test_data_category = target_test

log_reg = LogisticRegression(solver='sag', max_iter= 10000)
log_reg.fit(train, target)
predicted = log_reg.predict(test)
score = log_reg.predict_proba(test)[:,1]

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(test_data_category,␣
 ↪predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
```

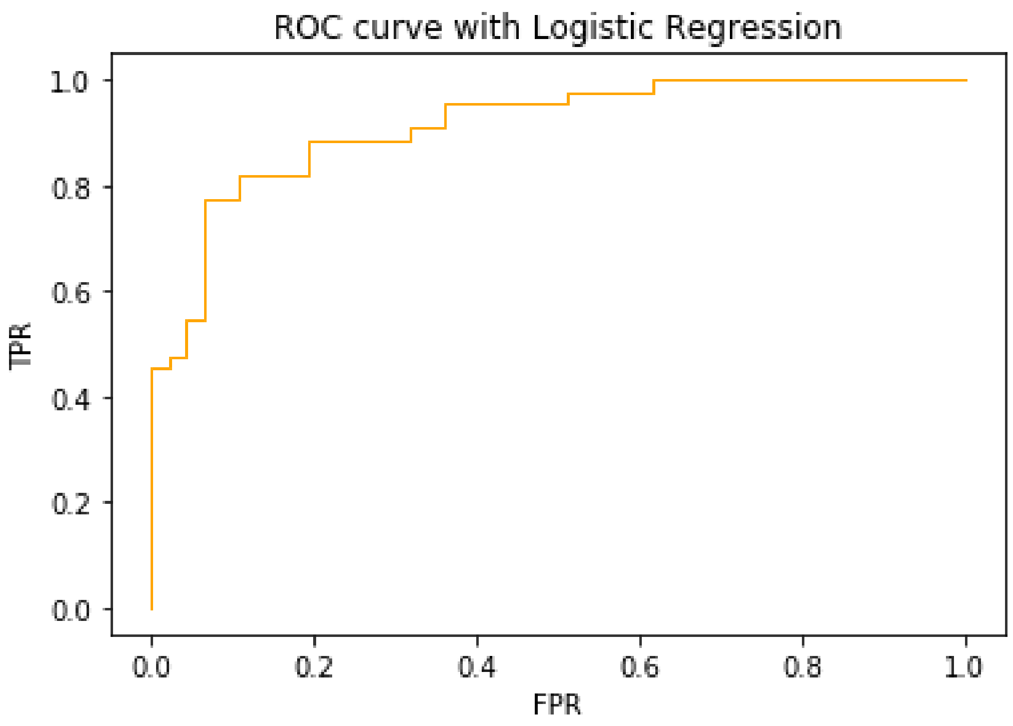
```
print("Confusion Matrix: \n", metrics.confusion_matrix(test_data_category,␣
↪predicted))
draw_confusion_matrix(target_test, predicted, ['comp', 'rec'])

fpr_log_reg, tpr_log_reg, thresholds = metrics.roc_curve(test_data_category,␣
↪score)

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

pyplot.figure(1)
pyplot.plot(fpr_log_reg, tpr_log_reg, color='orange', lw=1)
pyplot.title("ROC curve with Logistic Regression")
pyplot.xlabel('FPR')
pyplot.ylabel('TPR')
```

```
Accuracy:     0.846154
Precision:    0.875000
Recall:       0.795455
F1 Score:     0.833333
Confusion Matrix:
 [[42  5]
 [ 9 35]]
```



```
Logistic Model Performance Results:
```

```
[73]: Text(0, 0.5, 'TPR')
```



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

[Provide explanation here] By updating the maximum number of iterations, it allowed the model to converge on an optimal solution.

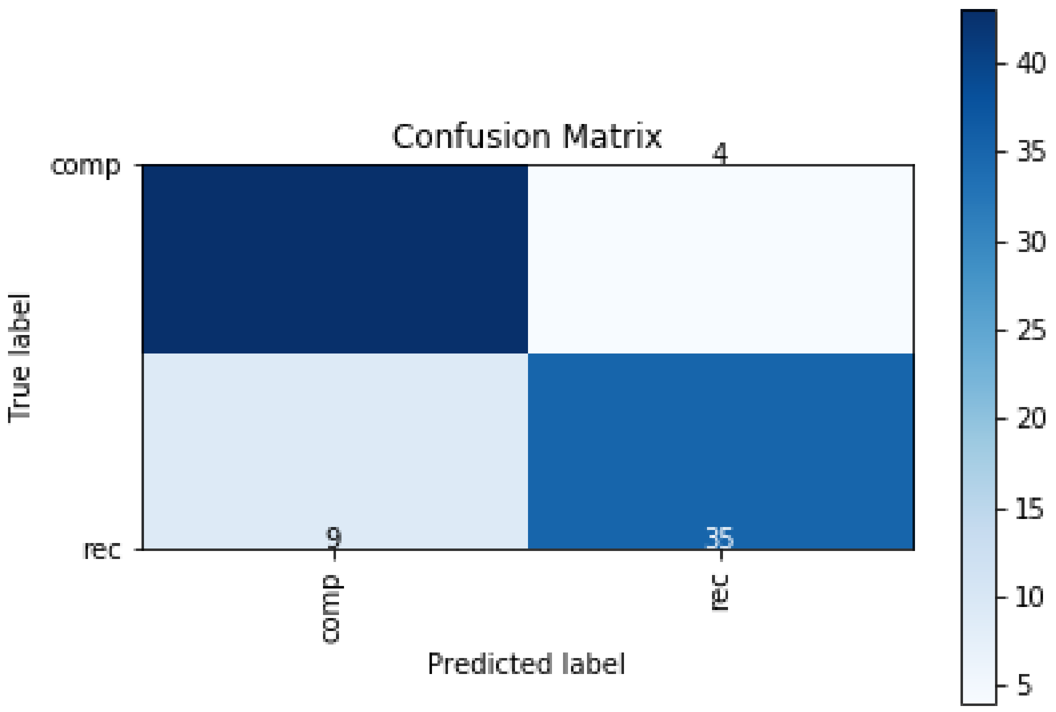
1.5.15 Question 3.2.4 Rerun your logistic classifier, but modify the penalty = 'none', solver='sag' and again report the results.

```
[85]: # Logistic Regression
train_data_category = target
test_data_category = target_test

log_reg = LogisticRegression(penalty = 'none', solver='sag')
log_reg.fit(train, target)
predicted = log_reg.predict(test)
score = log_reg.predict_proba(test)[:,1]

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(test_data_category,␣
 ↪predicted)))
```

```
print("%-12s %f" % ('Precision:', metrics.precision_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("Confusion Matrix: \n", metrics.confusion_matrix(test_data_category,␣
 ↪predicted))
draw_confusion_matrix(target_test, predicted, ['comp', 'rec'])

fpr_log_reg, tpr_log_reg, thresholds = metrics.roc_curve(test_data_category,␣
 ↪score)

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

pyplot.figure(1)
pyplot.plot(fpr_log_reg, tpr_log_reg, color='orange', lw=1)
pyplot.title("ROC curve with Logistic Regression")
pyplot.xlabel('FPR')
pyplot.ylabel('TPR')
```

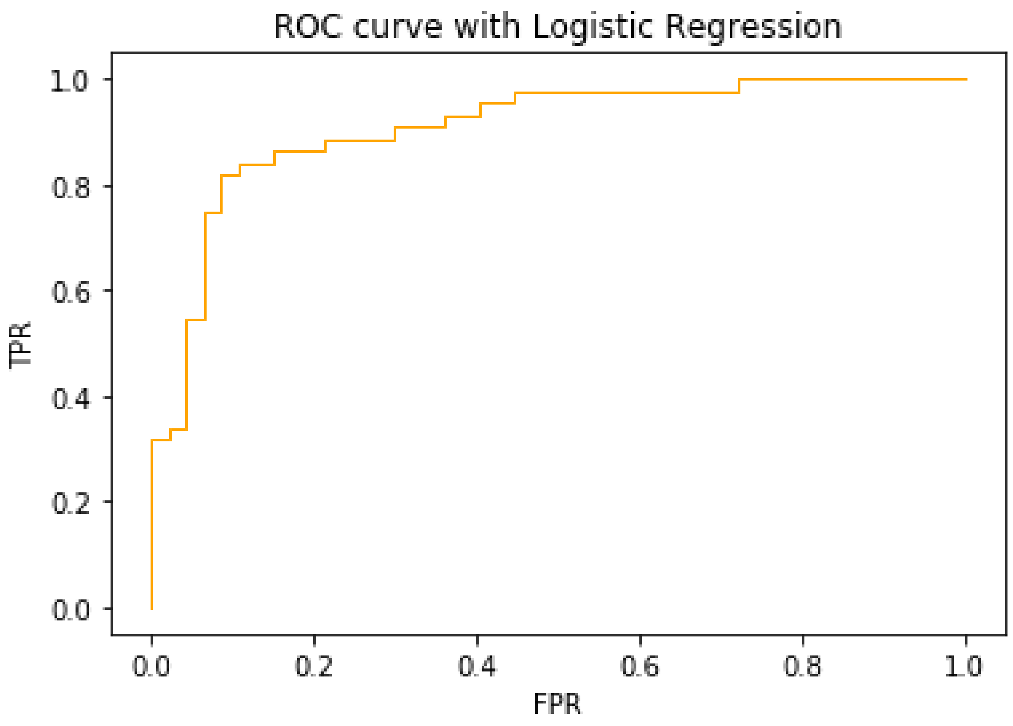
```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/sag.py:337:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
Accuracy:    0.857143
Precision:   0.897436
Recall:      0.795455
F1 Score:    0.843373
Confusion Matrix:
 [[43  4]
 [ 9 35]]
```



```
Logistic Model Performance Results:
```

```
[85]: Text(0, 0.5, 'TPR')
```



1.5.16 Question 3.2.5 Explain what what the two solver approaches are, and why the liblinear likely produced the optimal outcome.

[Provide explanation here] E.g., Library for Large Linear Classification:

It's a linear classification that supports logistic regression and linear support vector machines (A linear classifier achieves this by making a classification decision based on the value of a linear combination of the characteristics i.e feature value).

The solver uses a coordinate descent (CD) algorithm that solves optimization problems by successively performing approximate minimization along coordinate directions or coordinate hyperplanes.

Stochastic Average Gradient: (SAG) method optimizes the sum of a finite number of smooth convex functions. Like stochastic gradient (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 black-box SG methods.

It is faster than other solvers for large datasets, when both the number of samples and the number of features are large.

Per the documentation we see that liblinear is the default and recommended approach, whereas Sag is recommended for larger datasets where convergence is more difficult.

1.5.17 Question 3.2.5 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?

[Provide Answer here:] E.g., Logistic regression maximizes a likelihood function, whereas SVM classifies by finding a separating hyperplane that is able to divide the space into two half spaces which contain the two classes.

Why their performance differ? Logistic regression would work well with noisy data that can be represented as a sample from a bernoulli distribution, whereas SVM works on data that is linearly separable. Their working principles are different, and Logistic regression is almost always used with regularization to ensure that there is no overfitting.

Linear kernel SVMs work well when it is possible to linearly separate data, and do well with high dimension data, considering only the supporting vectors of the separating hyperplane On the other hand, Logistic regression uses every point in the data-set to estimate a function that maximizes the likelihood of the dataset being generated.

In short, their performance differs as the working principle is different.

1.5.18 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.

1.5.19 K-Nearest Neighbor

1.5.20 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.

[86]:
```python
# k-Nearest Neighbors algorithm

# Perform k-means on chosen news documents using 2 clusters, i.e. k =2
# Using workflow from documentation at https://scikit-learn.org/stable/modules/
 ↪generated/sklearn.cluster.KMeans.html
knn = KNeighborsClassifier()
knn.fit(train, target)

testing_result = knn.predict(test)
predicted = knn.predict(test)
score = knn.predict_proba(test)


print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test,␣
 ↪predicted)))
```

```
Accuracy:    0.824176
```

1.5.21 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).

[87]:
```
print("%-12s %f" % ('Homogeneity Score:', smc.homogeneity_score(target_test,␣
↪predicted)))
print("%-12s %f" % ('Completeness:', smc.completeness_score(target_test,␣
↪predicted)))
print("%-12s %f" % ('V-Measure:', smc.v_measure_score(target_test, predicted)))
print("%-12s %f" % ('Adjusted Rand Index:', smc.
↪adjusted_rand_score(target_test, predicted)))
print("%-12s %f" % ('Adjusted Mutual Information:', smc.
↪adjusted_mutual_info_score(target_test, predicted)))
```

```
Homogeneity Score: 0.335795
Completeness: 0.342270
V-Measure:   0.339001
Adjusted Rand Index: 0.414015
Adjusted Mutual Information: 0.330389
```

```
/opt/anaconda3/lib/python3.7/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)
```

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

[Input answer here] E.g., Homogeneity score proportion between 0 and 1, with 1 indicating that clusters contain data points only from a single class

Completeness score Proportion between 0 and 1, 1 indicating that all data points in a given class are in the same cluster.

V-measure score is the harmonic average of homogeneity and completeness scores and is equivalent to the F1 score for clustering algorithms.

Adjusted Rand Index is a score between -1 and 1 that, like accuracy, measures similarity between assigned and ground truth labels

Adjusted mutual information score indicates the mutual information between assigned and ground truth label distributions.

As we can see from these measures, the results are significantly low indicating that our clustering algorithm can be improved significantly.

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.

1.5.23 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).

[88]:
```
r = [1,2,3,5,10,20,50,100]

for i in r:


    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(train, target)

    testing_result = knn.predict(test)
    predicted = knn.predict(test)
    score = knn.predict_proba(test)

    print("-" * 40)
    print("For an r value of = ", i, "\n")
    print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test,␣
 ↪predicted)))
    print("%-12s %f" % ('Homogeneity Score:', smc.
 ↪homogeneity_score(target_test, predicted)))
    print("%-12s %f" % ('Completeness:', smc.completeness_score(target_test,␣
 ↪predicted)))
    print("%-12s %f" % ('V-Measure:', smc.v_measure_score(target_test,␣
 ↪predicted)))
    print("%-12s %f" % ('Adjusted Rand Index:', smc.
 ↪adjusted_rand_score(target_test, predicted)))
    print("%-12s %f" % ('Adjusted Mutual Information:', smc.
 ↪adjusted_mutual_info_score(target_test, predicted)))
```

```
----------------------------------------
For an r value of =  1

Accuracy:    0.758242
Homogeneity Score: 0.202894
Completeness: 0.202753
```

```
V-Measure:   0.202823
Adjusted Rand Index: 0.258526
Adjusted Mutual Information: 0.196274
----------------------------------------
For an r value of =  2

Accuracy:    0.736264
Homogeneity Score: 0.203608
Completeness: 0.235712
V-Measure:   0.218487
Adjusted Rand Index: 0.216023
Adjusted Mutual Information: 0.197096
----------------------------------------
For an r value of =  3

Accuracy:    0.857143
Homogeneity Score: 0.407921
Completeness: 0.408491
V-Measure:   0.408206
Adjusted Rand Index: 0.504722
Adjusted Mutual Information: 0.403106
----------------------------------------
For an r value of =  5

Accuracy:    0.824176
Homogeneity Score: 0.335795
Completeness: 0.342270
V-Measure:   0.339001
Adjusted Rand Index: 0.414015
Adjusted Mutual Information: 0.330389
----------------------------------------
For an r value of =  10

Accuracy:    0.802198
Homogeneity Score: 0.292640
Completeness: 0.301977
V-Measure:   0.297235
Adjusted Rand Index: 0.358454
Adjusted Mutual Information: 0.286881
----------------------------------------
For an r value of =  20

Accuracy:    0.747253
Homogeneity Score: 0.201171
Completeness: 0.217206
V-Measure:   0.208881
Adjusted Rand Index: 0.236841
Adjusted Mutual Information: 0.194657
```

```
----------------------------------------
For an r value of =  50

Accuracy:    0.769231
Homogeneity Score: 0.254342
Completeness: 0.281461
V-Measure:   0.267215
Adjusted Rand Index: 0.282926
Adjusted Mutual Information: 0.248256
----------------------------------------
For an r value of =  100

Accuracy:    0.769231
Homogeneity Score: 0.269300
Completeness: 0.306751
V-Measure:   0.286808
Adjusted Rand Index: 0.283171
Adjusted Mutual Information: 0.263330
```

```
/opt/anaconda3/lib/python3.7/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)
/opt/anaconda3/lib/python3.7/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)
/opt/anaconda3/lib/python3.7/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)
/opt/anaconda3/lib/python3.7/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)
/opt/anaconda3/lib/python3.7/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)
/opt/anaconda3/lib/python3.7/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)
/opt/anaconda3/lib/python3.7/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)
/opt/anaconda3/lib/python3.7/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)
```

An r value of 5 produced the optimal accuracy (0.857143), and a V-Measure of 0.34

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

[Input answer here] E.g., Unlike most other methods of classification, kNN falls under lazy learning, which means that there is no explicit training phase before classification. Instead, any attempts to generalize or abstract the data is made upon classification. While this does mean that we can immediately begin classifying once we have our data, there are some inherent problems with this type of algorithm. We must be able to keep the entire training set in memory unless we apply some type of reduction to the data-set, and performing classifications can be computationally expensive as the algorithm parse through all data points for each classification. For these reasons, kNN tends to work best on smaller data-sets that do not have many features.

In this case, because of its linear nature, clusters may be poorly able to classify compared to linear approaches.