Import packages

In [27]:
```
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 # for data visualization
import seaborn as sns # for statistical data visualization
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.metrics import roc_curve
```

Upload the dataset

In [2]:
```
cd C:\Users\CHENG\ICE_World\Python\SVM
```

C:\Users\ CHENG\ICE_World\Python\SVM

In [3]:
```
df = pd.read_csv("heart.csv")
```

Exploratory data analysis

In [4]:
```
# view the shape of the dataset
df.shape
```

Out[4]: (303, 14)

We can see that there are 303 instances and 14 attributes in the data set.

In [5]:
```
# view the first 5 rows
df.head()
```

Out[5]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

We can see that there are 14 variables in the dataset.All numeric, and there are 13 discrete variables and 1 continuous variable (oldpeak).You can also check this using df.info()

In [6]:
```
# view summary of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #    Column     Non-Null Count   Dtype
----  ------     --------------   -----
 0    age        303 non-null     int64
 1    sex        303 non-null     int64
 2    cp         303 non-null     int64
 3    trtbps     303 non-null     int64
 4    chol       303 non-null     int64
 5    fbs        303 non-null     int64
 6    restecg    303 non-null     int64
 7    thalachh   303 non-null     int64
 8    exng       303 non-null     int64
 9    oldpeak    303 non-null     float64
 10   slp        303 non-null     int64
 11   caa        303 non-null     int64
 12   thall      303 non-null     int64
 13   output     303 non-null     int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [7]:
```
# Check the distribution for the target class named (output)
df['output'].value_counts()
```

Out[7]:
```
1    165
0    138
Name: output, dtype: int64
```

In [8]:
```
# view the percentage distribution of target_class column (output)
df['output'].value_counts()/np.float(len(df))
```

Out[8]:
```
1    0.544554
0    0.455446
Name: output, dtype: float64
```

We can see that percentage of observations of the class label 1 and 0 is 54.45% and 45.54%.There appers to be no class imbalance problem.

In [9]:
```
# check for missing values in variables
df.isnull().sum()
```

Out[9]:
```
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
output      0
dtype: int64
```

We can see that there are no missing values in the dataset.

In [10]:
```
# Declare feature vector and target variable
x = df.drop(['output'], axis=1)
y = df['output']
```

In [11]:
```
# Split data into separate training and test set
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 42)
```

In [12]:
```
# check the shape of X_train and X_test, y_train, y_test
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[12]:
```
((227, 13), (76, 13), (227,), (76,))
```

Feature Scaling

In [13]:
```
cols = X_train.columns

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [14]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
```

In [15]:
```
X_test = pd.DataFrame(X_test, columns=[cols])
```

In [16]:
```
X_train.describe()
```

Out[16]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	
count	2.270000e+02	2.270000e+02	2.270000e+02	2.270000e+02	2.270000e+02	2.270000e+02	2.270000e+02	2.270000e+02	2.270000e+02	2.270000e+02	2.2
mean	8.558988e-17	7.923178e-17	-1.541841e-16	-6.451032e-16	8.803531e-17	-1.325420e-16	7.629727e-17	8.118812e-17	-1.124896e-16	2.934510e-18	-6.7
std	1.002210e+00	1.002210e+00	1.002210e+00	1.002210e+00	1.002210e+00	1.002210e+00	1.002210e+00	1.002210e+00	1.002210e+00	1.002210e+00	1.0
min	-2.751968e+00	-1.395726e+00	-9.793709e-01	-2.126827e+00	-2.154352e+00	-3.827070e-01	-1.044861e+00	-2.735962e+00	-6.815542e-01	-9.312660e-01	-2.3
25%	-7.030748e-01	-1.395726e+00	-9.793709e-01	-5.969633e-01	-6.576218e-01	-3.827070e-01	-1.044861e+00	-7.188027e-01	-6.815542e-01	-9.312660e-01	-6.6
50%	1.596170e-01	7.164728e-01	-1.277440e-02	-8.553968e-03	-1.337662e-01	-3.827070e-01	8.375472e-01	1.456940e-01	-6.815542e-01	-2.063714e-01	-6.6
75%	6.987993e-01	7.164728e-01	9.538221e-01	5.798553e-01	5.210534e-01	-3.827070e-01	8.375472e-01	7.220251e-01	1.467235e+00	5.185232e-01	9.7
max	2.424183e+00	7.164728e-01	1.920419e+00	3.639584e+00	5.946701e+00	2.612965e+00	2.719955e+00	2.318019e+00	1.467235e+00	4.142996e+00	9.7

SVM with default hyperparameters

Default hyperparameter means C=1.0, kernel=rbf and gamma=auto among other parameters.

In [17]:
```
# instantiate classifier with default hyperparameters
svc=SVC()
# fit classifier to training set
svc.fit(X_train,y_train)
# make predictions on test set
y_pred=svc.predict(X_test)
# compute and print accuracy score
print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with default hyperparameters: 0.8816
```

In [18]:
```
# declare parameters for hyperparameter tuning
parameters = [ {'C':[1, 10, 100, 1000], 'kernel':['linear']},
               {'C':[1, 10, 100, 1000], 'kernel':['rbf'], 'gamma':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]},
               {'C':[1, 10, 100, 1000], 'kernel':['poly'], 'degree': [2,3,4] ,'gamma':[0.01,0.02,0.03,0.04,0.05]}
              ]




grid_search = GridSearchCV(estimator = svc,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5)


grid_search.fit(X_train, y_train)
```

Out[18]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid=[{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
                         {'C': [1, 10, 100, 1000],
                          'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
                                    0.9],
                          'kernel': ['rbf']},
                         {'C': [1, 10, 100, 1000], 'degree': [2, 3, 4],
                          'gamma': [0.01, 0.02, 0.03, 0.04, 0.05],
                          'kernel': ['poly']}],
             scoring='accuracy')
```

In [19]:
```
# gamma is used for non-linear classification problems
```

In [20]:
```python
# examine the best model

# best score achieved during the GridSearchCV
print('GridSearch CV best score : {:.4f}\n\n'.format(grid_search.best_score_))


# print parameters that give the best results
print('Parameters that give the best results :','\n\n', (grid_search.best_params_))


# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :','\n\n', (grid_search.best_estimator_))
```

```
GridSearch CV best score : 0.8064


Parameters that give the best results :

 {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}


Estimator that was chosen by the search :

 SVC(C=1, gamma=0.1)
```

In [21]:
```python
# calculate GridSearch CV score on test set
print('GridSearch CV score on test set: {0:0.4f}'.format(grid_search.score(X_test, y_test)))
```

```
GridSearch CV score on test set: 0.8816
```

In [22]:
```python
svc=SVC(C = 1.0, gamma = 0.1, kernel ='rbf', probability=True)
# fit classifier to training set
svc.fit(X_train,y_train)
# make predictions on test set
y_pred_test=svc.predict(X_test)
```

In [23]:
```python
# Print the Confusion Matrix and slice it into four pieces

cm = confusion_matrix(y_test, y_pred_test)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[31  4]
 [ 5 36]]

True Positives(TP) = 31

True Negatives(TN) = 36

False Positives(FP) = 4

False Negatives(FN) = 5
```

The confusion matrix shows 31 + 36 = 67 correct predictions and 4 + 5 = 9 incorrect predictions.

In this case, we have

- True Positives (Actual Positive:1 and Predict Positive:1) - 31
- True Negatives (Actual Negative:0 and Predict Negative:0) - 36
- False Positives (Actual Negative:0 but Predict Positive:1) - 4 (Type I error)
- False Negatives (Actual Positive:1 but Predict Negative:0) - 5 (Type II error)

Classification Report

Classification report is another way to evaluate the classification model performance. It displays the precision, recall, f1 and support scores for the model. We can print a classification report as follows:

In [24]:
```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.86      0.89      0.87        35
           1       0.90      0.88      0.89        41

    accuracy                           0.88        76
   macro avg       0.88      0.88      0.88        76
weighted avg       0.88      0.88      0.88        76
```

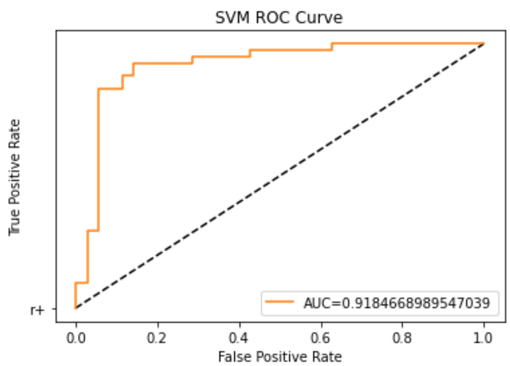
ROC CURVE

In [25]:
```
y_pred_proba = svc.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
auc = metrics.roc_auc_score(y_test,  y_pred_proba)
```

In [28]:
```
# calculating the probabilities
y_pred_prob = svc.predict_proba(X_test)[:,1]

# instantiating the roc_cruve
fpr,tpr,threshols=roc_curve(y_test,y_pred_prob)

# plotting the curve
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot([0,1],[0,1],"k--",'r+')
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("SVM ROC Curve")
plt.legend(loc="lower right")
plt.show()
```



Comments ROC AUC is a single number summary of classifier performance. The higher the value, the better the classifier.

ROC AUC of our model approaches towards 1. So, we can conclude that our classifier does a good job in classifying the pulsar star.

Here, y_test are the true class labels and y_pred are the predicted class labels in the test-set.

Compare the train-set and test-set accuracy Now, I will compare the train-set and test-set accuracy to check for overfitting.

Check for overfitting and underfitting

In [29]:
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(svc.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(svc.score(X_test, y_test)))
```

```
Training set score: 0.9295
Test set score: 0.8816
```

The training-set accuracy score is 0.9295 while the test-set accuracy to be 0.8816. These two values are quite comparable. So, there is no question of overfitting.

Compare model accuracy with null accuracy So, the model accuracy is 0.8816. But, we cannot say that our model is very good based on the above accuracy. We must compare it with the null accuracy. Null accuracy is the accuracy that could be achieved by always predicting the most frequent class.

So, we should first check the class distribution in the test set.

In [30]:
```
# check class distribution in test set
y_test.value_counts()
```

Out[30]:
```
1    41
0    35
Name: output, dtype: int64
```

We can see that the occurences of most frequent class 1 is 41. So, we can calculate null accuracy by dividing 41 by total number of occurences.

In [31]:
```
# check null accuracy score
null_accuracy = (41/(41+35))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.5395
```

We can see that our model accuracy score is 0.8816 but null accuracy score is 0.5395. So, we can conclude that our SVM classifier is doing a very good job in predicting the class labels.

Exercises

In [32]:
```
# Run SVM with rbf kernel and C=100.0
svc=SVC(kernel='rbf', C=100.0)


# fit classifier to training set
svc.fit(X_train,y_train)


# make predictions on test set
y_pred=svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with rbf kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with rbf kernel and C=100.0 : 0.7763
```

In [33]:
```
# Run SVM with rbf kernel and C=1000.0
svc=SVC(kernel='rbf', C=1000.0)


# fit classifier to training set
svc.fit(X_train,y_train)


# make predictions on test set
y_pred=svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with rbf kernel and C=1000.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with rbf kernel and C=1000.0 : 0.7763
```

In [34]:
```
# Run SVM with linear kernel and C=100.0
linear_svc100=SVC(kernel='linear', C=100.0)


# fit classifier to training set
linear_svc100.fit(X_train, y_train)


# make predictions on test set
y_pred=linear_svc100.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with linear kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with linear kernel and C=100.0 : 0.8553
```

In [35]:

```
# Run SVM with linear kernel and C=1000.0
linear_svc1000=SVC(kernel='linear', C=1000.0)


# fit classifier to training set
linear_svc1000.fit(X_train, y_train)


# make predictions on test set
y_pred=linear_svc1000.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with linear kernel and C=1000.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with linear kernel and C=1000.0 : 0.8553
```

In [36]:

```
# Run SVM with polynomial kernel and C=100.0
poly_svc=SVC(kernel='poly', C=100.0)


# fit classifier to training set
poly_svc.fit(X_train,y_train)


# make predictions on test set
y_pred=poly_svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with polynomial kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with polynomial kernel and C=100.0 : 0.7895
```

In [37]:

```
#Run SVM with polynomial kernel and C=1000.0

poly_svc100=SVC(kernel='poly', C=1000.0)


# fit classifier to training set
poly_svc100.fit(X_train, y_train)


# make predictions on test set
y_pred=poly_svc100.predict(X_test)


#  compute  and  print accuracy  score
print('Model accuracy score with polynomial kernel and C=1000.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with polynomial kernel and C=1000.0 : 0.7895
```

In [38]:

```
#Run SVM with sigmoid kernel and C=100.0
sigmoid_svc=SVC(kernel='sigmoid', C=100.0)


#  fit classifier to training set
sigmoid_svc.fit(X_train,y_train)


# make predictions on test set
y_pred=sigmoid_svc.predict(X_test)


#  compute  and  print accuracy  score
print('Model accuracy score with sigmoid kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with sigmoid kernel and C=100.0 : 0.7500
```

In [39]:

```
#Run SVM with sigmoid kernel and C=1000.0
sigmoid_svc100=SVC(kernel='sigmoid', C=1000.0)


# fit classifier to training set
sigmoid_svc100.fit(X_train,y_train)


# make predictions on test set
y_pred=sigmoid_svc100.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with sigmoid kernel and C=1000.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with sigmoid kernel and C=1000.0 : 0.7763
```

In [ ]: