Heart Disease Prediction

1. Data description

Dataset: https://www.kaggle.com/ronitf/heart-disease-uci

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date.

Attribute Information:

- age
- sex (1 = male, 0 = female)
- cp: chest pain type (1 = typical angina, 2 = atypical angina, 3 = non-anginal pain, 4 = asymptomatic)
- trestbps: resting blood pressure
- chol: serum cholestoral in mg/dl
- fbs: fasting blood sugar (> 120 mg/dl, 1 = true; 0 = false)
- restecg: resting electrocardiographic results (0 = normal, 1 = having ST-T wave abnormality, 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria)
- thalach: maximum heart rate achieved
- exang: exercise induced angina(1 = yes; 0 = no)
- oldpeak = ST depression induced by exercise relative to rest
- slope: the slope of the peak exercise ST segment(1 = upsloping, 2 = flat, 3 = downsloping)
- ca: number of major vessels (0-3) colored by flourosopy
- thal: thalassemia (1 = normal; 2 = fixed defect; 3 = reversable defect)
- target: Heart disease (0 = no, 1 = yes)

2. Load & Explore the dataset

In [1]:

```
import pandas as pd
data = pd.read_csv("heart.csv")
df = data.copy()
```

In [2]:

```
df.head()
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
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

In [3]:

```
df.shape
```

Out[3]:

```
(303, 14)
```

In [4]:
```
#NO NaN values
df.isna().sum()
```

Out[4]:
```
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
target      0
dtype: int64
```

In [5]:
```
df.dtypes
```

Out[5]:
```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

In [6]:
```
df.describe(include='all')
```

Out[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000

In [7]:
```
#unique values: actually categorical variables
import pandas as pd
import numpy as np
for i in df.columns:
    uniquevalues=pd.unique(df[i])
    if(len(uniquevalues)<20):
```

```
        print(i,np.sort(uniquevalues))
#except for ca
```

```
sex [0 1]
cp [0 1 2 3]
fbs [0 1]
restecg [0 1 2]
exang [0 1]
slope [0 1 2]
ca [0 1 2 3 4]
thal [0 1 2 3]
target [0 1]
```

In [8]:

```
#unique values: actually numerical variables
import pandas as pd
for i in df.columns:
    unique=pd.unique(df[i])
    if(len(unique)>20):
        print(i,[df[i].min(),df[i].max()])
```

```
age [29, 77]
trestbps [94, 200]
chol [126, 564]
thalach [71, 202]
oldpeak [0.0, 6.2]
```

3. Initial Thoughts

- 303 entries with 14 variables - 13 of them can be predictors.
- No missing values. :) Great!
- All the values in the dataframe is "numerical" - however, 7 of the predictors are actually categorical and are encoded to integers.
  categorical variables: sex, cp, fbs, restecg, exang, slope, thal.
  numerical variables: age, trestbps, chol, thalach, oldpeak, ca.
  The categorical ones should be converted into categorical variables (and then make them into dummy variables when needed for the model).

4. Data Wrangling

In [9]:

```
#for sex
df.loc[df.sex==0, 'sex'] = "female"
df.loc[df.sex==1, 'sex'] = "male"
#for cp
df.loc[df.cp==1, 'cp'] = "typical angina"
df.loc[df.cp==2, 'cp'] = "atypical angina"
df.loc[df.cp==3, 'cp'] = "non-anginal pain"
df.loc[df.cp==4, 'cp'] = "asymptomatic chest pain"
#for fbs
df.loc[df.fbs==1,'fbs']="fbs greater than 120"
df.loc[df.fbs==0,'fbs']="fbs lower than 120"
#for restecg
df.loc[df.restecg==0,'restecg']="restecg normal"
df.loc[df.restecg==1,'restecg']="ST-T wave abnormality"
df.loc[df.restecg==2,'restecg']="left ventricular hypertrophy"
#for exang
df.loc[df.exang==0,'exang']="no exercise induced angina"
df.loc[df.exang==1,'exang']="exercise induced angina"
#for slope
df.loc[df.slope==1,'slope']="upsloping"
```

```
df.loc[df.slope==2,'slope']="flat"
df.loc[df.slope==3,'slope']="downsloping"
#thal
df.loc[df.thal==1,'thal']="normal thal"
df.loc[df.thal==2,'thal']="fixed defect"
df.loc[df.thal==3,'thal']="reversable defect"
```

In [10]:

```
#drop the rows that thal not described(2 rows)
#df[df.thal==0]
df=df[df.thal!=0]
```

In [11]:

```
list(df.select_dtypes(include=['object']).columns)
```

Out[11]:

```
['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'thal']
```

5. Modeling

Aim:

For potential disease detecting purpose, we should care most about how many people that have heart disease(true=1) that found out to have heart disease(predict=1) by our prediction. That's recall for the positive(1, in sklearn) class.
We should focus on other performance metrics as well...

In [15]:

```
# get dummy variables
df = pd.get_dummies(df,drop_first=True)
#drop_first: Whether to get k-1 dummies out of k categorical levels by removing the first
#https://stackoverflow.com/questions/50176096/removing-redundant-columns-when-using-get-du
df.head()
```

Out[15]:

	age	trestbps	chol	thalach	oldpeak	ca	target	sex_male	cp_atypical angina	cp_non-anginal pain	cp_typical angina	fbs_fbs lower than 120	restecg_le ventricul hypertrop
0	63	145	233	150	2.3	0	1	1	0	1	0	0	
1	37	130	250	187	3.5	0	1	1	1	0	0	1	
2	41	130	204	172	1.4	0	1	0	0	0	1	1	
3	56	120	236	178	0.8	0	1	1	0	0	1	1	
4	57	120	354	163	0.6	0	1	0	0	0	0	1	

In [16]:

```
#separate predictors and response variable
train_features = df.columns.drop('target').tolist()
X_df = df[train_features]
y_df = df['target']

# Create training and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, random_state = 0)#default.

#scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

function: model performance on the test set

In [17]:
```
from sklearn.metrics import roc_curve, auc, confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

def model_performance_testset(model, X_train, y_train, X_test, y_test, pos_label=1):#posit

    #predicted value
    print("==================predicted values==================")
    y_predicted=model.predict(X_test)
    if len(y_predicted)<20:
        print(y_predicted)
    else:
        print(y_predicted[0:21])

    #4 metrics for the positive class
    print("\n=========================metrics=========================")
    print('Accuracy: {:.2f}'.format(accuracy_score(y_test, y_predicted)))
    print('Precision: {:.2f}'.format(precision_score(y_test, y_predicted,pos_label=pos_lab
    print('Recall: {:.2f}'.format(recall_score(y_test, y_predicted,pos_label=pos_label)))
    print('F1: {:.2f}'.format(f1_score(y_test, y_predicted,pos_label=pos_label)))

    #report(for each class)
    print("\n================classification report for each class: ================")
    print(classification_report(y_test, y_predicted))

    #confusion matrix
    print("\n==================confusion matrix: ===================")
    confusion = confusion_matrix(y_test, y_predicted)
    print(confusion)
    return("=================finished summarization :)==================")

def model_auc(model, X_train, y_train, X_test, y_test, pos_label=1):
    #auc
    print("\n===================auc: ===================")
    y_score_lr=model.fit(X_train, y_train).decision_function(X_test)
    fpr_lr, tpr_lr, _ = roc_curve(y_test, y_score_lr)
    auc_value = auc(fpr_lr, tpr_lr)
    print("auc={}".format(auc_value))

    #auc plot
    plt.figure()
    plt.xlim([-0.01, 1.00])
    plt.ylim([-0.01, 1.01])
    plt.plot(fpr_lr, tpr_lr, lw=3, label='ROC curve (auc = {:0.2f})'.format(auc_value))
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.title('ROC curve', fontsize=16)
    plt.legend(loc='lower right', fontsize=13)
    plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
    plt.show()
    return("=================finished auc :)==================")
```

function: model performance using CV

In [18]:
```
from sklearn.model_selection import cross_val_score
def CV_metrics(model, X_train, y_train, cv=5):
    #the model SHOULD NOT be fitted#
```

```
    #accuracy
    cv_scores_accuracy = np.mean(cross_val_score(model, X_train, y_train, cv=cv))
    print('Mean cross-validation (accuracy) (5-fold): {:.3f}\n'.format(cv_scores_accuracy)

    #auc
    cv_scores_accuracy = np.mean(cross_val_score(model, X_train, y_train, cv=cv, scoring =
    print('Mean cross-validation (auc) (5-fold): {:.3f}\n'.format(cv_scores_accuracy))

    #recall (for 1 class)
    cv_scores_recall = np.mean(cross_val_score(model, X_train, y_train, cv=cv, scoring = '
    print('Mean cross-validation (recall) (5-fold): {:.3f}\n'.format(cv_scores_recall))

    #precision (for 1 class)
    cv_scores_precision = np.mean(cross_val_score(model, X_train, y_train, cv=cv, scoring
    print('Mean cross-validation (precision) (5-fold): {:.3f}\n'.format(cv_scores_precisic

    return("=================finished CV summarization :)==================")
```

(1) Logistic regression

In [12]:

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [126…

```
#logistic: grid search for the best parameter C
clf = LogisticRegression(max_iter=10000)
grid_values = {'C': [0.01, 0.1, 1, 10, 50, 100]}#values
grid_clf_acc = GridSearchCV(clf, param_grid = grid_values, scoring="recall").fit(X_train_s
predict_test=grid_clf_acc.predict(X_test_scaled) #Call predict on the estimator with the b
print('Grid best parameter (max. recall): ', grid_clf_acc.best_params_)
print('Grid best score (recall): ', grid_clf_acc.best_score_)
print('Test set recall: ', recall_score(y_test, predict_test))
```

```
Grid best parameter (max. recall):  {'C': 0.01}
Grid best score (recall):  0.9526153846153846
Test set recall:  0.9473684210526315
```

In [129…

```
logistic_clf = LogisticRegression(C=0.01,max_iter=10000)
logistic_clf.fit(X_train_scaled,y_train)
```

Out[129…

```
LogisticRegression(C=0.01, max_iter=10000)
```

(2) Random Forest

In [156…

```
randomforest_clf= RandomForestClassifier()
grid_values = {'max_depth': [2,3,4,5,6]}
grid_ranfor_recall = GridSearchCV(randomforest_clf, param_grid = grid_values, scoring="rec
predict_test_ranfor=grid_ranfor_recall.predict(X_test_scaled)
print('Grid best parameter (max. recall): ', grid_ranfor_recall.best_params_)
print('Grid best score (recall): ', grid_ranfor_recall.best_score_)
print('Test set recall: ', recall_score(y_test, predict_test_ranfor))
```

```
Grid best parameter (max. recall):  {'max_depth': 3}
Grid best score (recall):  0.8652307692307692
Test set recall:  0.9473684210526315
```

In [157…

```
randomforest_clf= RandomForestClassifier(max_depth=3).fit(X_train_scaled,y_train)
```

(3) Gradient-boosted decision trees

In [165…

```
gbdt_clf = GradientBoostingClassifier()
grid_values = {'learning_rate': [0.005,0.01,0.05,0.1],
               'max_depth':[2,3,4,5,6]}
grid_gbdt_recall = GridSearchCV(gbdt_clf, param_grid = grid_values, scoring="recall").fit
predict_test_gbdt=grid_gbdt_recall.predict(X_test_scaled)
print('Grid best parameter (max. recall): ', grid_gbdt_recall.best_params_)
print('Grid best score (recall): ', grid_gbdt_recall.best_score_)
print('Test set recall: ', recall_score(y_test, predict_test_gbdt))
```

```
Grid best parameter (max. recall):  {'learning_rate': 0.005, 'max_depth': 3}
Grid best score (recall):  0.8649230769230769
Test set recall:  0.9210526315789473
```

In [167…

```
gbdt_clf = GradientBoostingClassifier(learning_rate=0.005, max_depth=3).fit(X_train_scalec
```

(4) SVM

In [22]:

```
svm_clf = SVC()
grid_values = {'kernel': ['rbf', 'poly'],
               'gamma':[0.001, 1, 5, 10],
               'C':[0.1, 1, 15, 100]}
#grid_svm_recall = GridSearchCV(svm_clf, param_grid = grid_values, scoring="accuracy").fi
#A bad one, predict all 1s.
grid_svm_recall = GridSearchCV(svm_clf, param_grid = grid_values, scoring="recall").fit(X_
predict_test_svm=grid_svm_recall.predict(X_test_scaled)
print('Grid best parameter (max. recall): ', grid_svm_recall.best_params_)
print('Grid best score (recall): ', grid_svm_recall.best_score_)
print('Test set recall: ', recall_score(y_test, predict_test_svm))
```

```
Grid best parameter (max. recall):  {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
Grid best score (recall):  0.8266666666666668
Test set recall:  0.8947368421052632
```

In [23]:

```
svm_clf = SVC(kernel='rbf',gamma=0.001,C=100).fit(X_train_scaled,y_train)
```

(5) KNN

In [32]:

```
knn_clf = KNeighborsClassifier()
grid_values = {'n_neighbors': [3, 5, 7, 9, 12]}
grid_knn_recall = GridSearchCV(knn_clf, param_grid = grid_values, scoring="recall").fit(X_
predict_test_knn=grid_knn_recall.predict(X_test_scaled)
print('Grid best parameter (max. recall): ', grid_knn_recall.best_params_)
print('Grid best score (recall): ', grid_knn_recall.best_score_)
print('Test set recall: ', recall_score(y_test, predict_test_knn))
```

```
Grid best parameter (max. recall):  {'n_neighbors': 9}
Grid best score (recall):  0.8329230769230769
Test set recall:  0.8947368421052632
```

In [33]:

```
knn_clf = KNeighborsClassifier(n_neighbors=9).fit(X_train_scaled,y_train)
```

(6) MLPclassifier

In [40]:
```
from sklearn.neural_network import MLPClassifier

mlp_clf = MLPClassifier(hidden_layer_sizes = [500, 5], max_iter = 10000)
grid_values = {'alpha': [0.01, 0.1, 1.0, 5.0]}
grid_mlp_recall = GridSearchCV(mlp_clf, param_grid = grid_values, scoring="recall").fit(X_
predict_test_mlp=grid_mlp_recall.predict(X_test_scaled)
print('Grid best parameter (max. recall): ', grid_mlp_recall.best_params_)
print('Grid best score (recall): ', grid_mlp_recall.best_score_)
print('Test set recall: ', recall_score(y_test, predict_test_mlp))
```

```
Grid best parameter (max. recall):  {'alpha': 5.0}
Grid best score (recall):  0.8886153846153846
Test set recall:  0.9473684210526315
```

In [41]:
```
mlp_clf = MLPClassifier(hidden_layer_sizes = [500, 5], max_iter = 10000, alpha=5)
mlp_clf.fit(X_train_scaled,y_train)
```

Out[41]:
```
MLPClassifier(alpha=5, hidden_layer_sizes=[500, 5], max_iter=10000)
```

7. Model-selection (test set and CV)

(1) Logistic regression

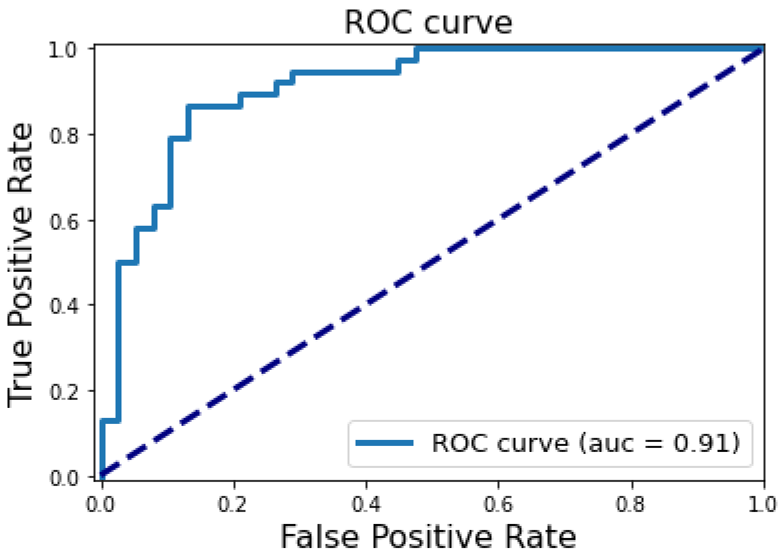
In [145…
```
model_performance_testset(logistic_clf,X_train_scaled, y_train, X_test_scaled, y_test)
```

```
==================predicted values==================
[0 1 0 0 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1]

=========================metrics=========================
Accuracy: 0.78
Precision: 0.71
Recall: 0.95
F1: 0.81

================classification report for each class: ===============
              precision    recall  f1-score   support

           0       0.92      0.61      0.73        38
           1       0.71      0.95      0.81        38

    accuracy                           0.78        76
   macro avg       0.81      0.78      0.77        76
weighted avg       0.81      0.78      0.77        76


==================confusion matrix: ===================
[[23 15]
 [ 2 36]]
```

Out[145…
```
'=================finished summarization :)=================='
```

In [147…
```
model_auc(logistic_clf,X_train_scaled, y_train, X_test_scaled, y_test)
```

```
==================auc: ===================
auc=0.9106648199445984
```



Out[147…

```
'=================finished auc :)=================='
```

In [149…

```
CV_metrics(LogisticRegression(C=0.01,max_iter=10000),X_train_scaled, y_train)
```

```
Mean cross-validation (accuracy) (5-fold): 0.738

Mean cross-validation (auc) (5-fold): 0.884

Mean cross-validation (recall) (5-fold): 0.953

Mean cross-validation (precision) (5-fold): 0.696
```

Out[149…

```
'=================finished CV summarization :)=================='
```

(2) Random Forest

In [158…

```
model_performance_testset(randomforest_clf,X_train_scaled, y_train, X_test_scaled, y_test)
```

```
==================predicted values==================
[0 1 0 0 1 0 0 0 1 0 0 1 0 1 1 1 1 1 1 1 1]

=========================metrics=========================
Accuracy: 0.86
Precision: 0.80
Recall: 0.95
F1: 0.87

================classification report for each class: ===============
              precision    recall  f1-score   support

           0       0.94      0.76      0.84        38
           1       0.80      0.95      0.87        38

    accuracy                           0.86        76
   macro avg       0.87      0.86      0.85        76
weighted avg       0.87      0.86      0.85        76


==================confusion matrix: ===================
[[29  9]
 [ 2 36]]
```

Out[158…

```
'=================finished summarization :)=================='
```

In [161…
```
CV_metrics(RandomForestClassifier(max_depth=3),X_train_scaled, y_train)
```

```
Mean cross-validation (accuracy) (5-fold): 0.804

Mean cross-validation (auc) (5-fold): 0.887

Mean cross-validation (recall) (5-fold): 0.881

Mean cross-validation (precision) (5-fold): 0.806
```

Out[161…
```
'=================finished CV summarization :)=================='
```

(3) Gradient-boosted decision trees

In [168…
```
model_performance_testset(gbdt_clf,X_train_scaled, y_train, X_test_scaled, y_test)
```

```
==================predicted values==================
[0 1 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1]

=========================metrics=========================
Accuracy: 0.83
Precision: 0.78
Recall: 0.92
F1: 0.84

================classification report for each class: ===============
              precision    recall  f1-score   support

           0       0.90      0.74      0.81        38
           1       0.78      0.92      0.84        38

    accuracy                           0.83        76
   macro avg       0.84      0.83      0.83        76
weighted avg       0.84      0.83      0.83        76


==================confusion matrix: ===================
[[28 10]
 [ 3 35]]
```

Out[168…
```
'=================finished summarization :)=================='
```

In [169…
```
CV_metrics(GradientBoostingClassifier(learning_rate=0.005, max_depth=3),X_train_scaled, y_
```

```
Mean cross-validation (accuracy) (5-fold): 0.760

Mean cross-validation (auc) (5-fold): 0.826

Mean cross-validation (recall) (5-fold): 0.865

Mean cross-validation (precision) (5-fold): 0.748
```

Out[169…
```
'=================finished CV summarization :)=================='
```

(4) SVM

In [24]:
```
model_performance_testset(svm_clf,X_train_scaled, y_train, X_test_scaled, y_test)
```

```
==================predicted values==================
[0 1 0 0 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1]
```

```
=========================metrics=========================
Accuracy: 0.83
Precision: 0.79
Recall: 0.89
F1: 0.84

================classification report for each class: ===============
              precision    recall  f1-score   support

           0       0.88      0.76      0.82        38
           1       0.79      0.89      0.84        38

    accuracy                           0.83        76
   macro avg       0.83      0.83      0.83        76
weighted avg       0.83      0.83      0.83        76


==================confusion matrix: ===================
[[29  9]
 [ 4 34]]
```

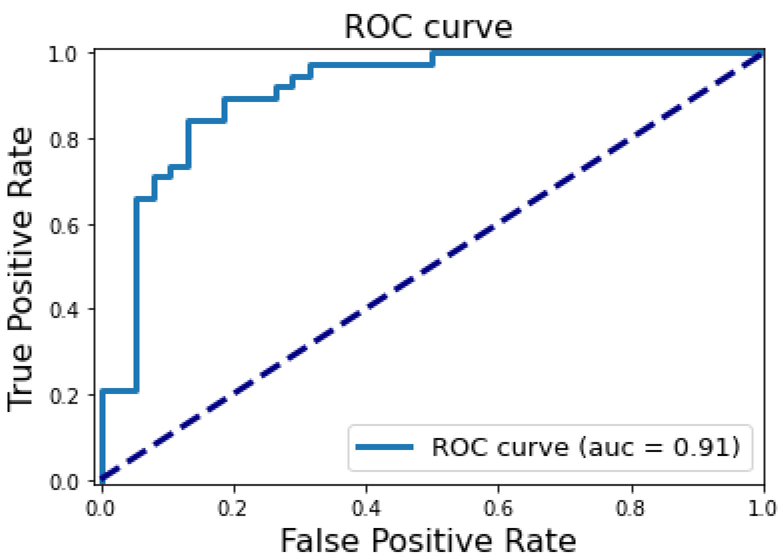
Out[24]:

```
'=================finished summarization :)=================='
```

In [36]:

```
model_auc(svm_clf,X_train_scaled, y_train, X_test_scaled, y_test)
```

```
==================auc: ===================
auc=0.9099722991689752
```



Out[36]:

```
'=================finished auc :)=================='
```

In [25]:

```
CV_metrics(SVC(kernel='rbf',gamma=0.001,C=100),X_train_scaled, y_train)
```

```
Mean cross-validation (accuracy) (5-fold): 0.827

Mean cross-validation (auc) (5-fold): 0.893

Mean cross-validation (recall) (5-fold): 0.873

Mean cross-validation (precision) (5-fold): 0.827
```

Out[25]:

```
'=================finished CV summarization :)=================='
```

(5) KNN

In [34]:

```
model_performance_testset(knn_clf,X_train_scaled, y_train, X_test_scaled, y_test)
```

```
==================predicted values==================
[0 1 0 0 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1]

=========================metrics=========================
Accuracy: 0.86
Precision: 0.83
Recall: 0.89
F1: 0.86

================classification report for each class: ================
              precision    recall  f1-score   support

           0       0.89      0.82      0.85        38
           1       0.83      0.89      0.86        38

    accuracy                           0.86        76
   macro avg       0.86      0.86      0.86        76
weighted avg       0.86      0.86      0.86        76


==================confusion matrix: ====================
[[31  7]
 [ 4 34]]
```

Out[34]:

```
'=================finished summarization :)=================='
```

In [38]:

```python
CV_metrics(KNeighborsClassifier(n_neighbors=9),X_train_scaled, y_train)
```

```
Mean cross-validation (accuracy) (5-fold): 0.818

Mean cross-validation (auc) (5-fold): 0.880

Mean cross-validation (recall) (5-fold): 0.833

Mean cross-validation (precision) (5-fold): 0.840
```

Out[38]:

```
'=================finished CV summarization :)=================='
```

(6) MLP classifier

In [154…

```python
model_performance_testset(mlp_clf,X_train_scaled, y_train, X_test_scaled, y_test)
```

```
==================predicted values==================
[0 1 0 0 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1]

=========================metrics=========================
Accuracy: 0.83
Precision: 0.77
Recall: 0.95
F1: 0.85

================classification report for each class: ================
              precision    recall  f1-score   support

           0       0.93      0.71      0.81        38
           1       0.77      0.95      0.85        38

    accuracy                           0.83        76
   macro avg       0.85      0.83      0.83        76
weighted avg       0.85      0.83      0.83        76
```

```
==================confusion matrix: ====================
[[27 11]
 [ 2 36]]
```

Out[154…

```
'=================finished summarization :)=================='
```

In [43]:

```python
CV_metrics(MLPClassifier(hidden_layer_sizes = [500, 5], max_iter = 10000, alpha=5),X_trair
```

```
Mean cross-validation (accuracy) (5-fold): 0.844

Mean cross-validation (auc) (5-fold): 0.898

Mean cross-validation (recall) (5-fold): 0.873

Mean cross-validation (precision) (5-fold): 0.839
```

Out[43]:

```
'=================finished CV summarization :)=================='
```

Based on the metrics, considering all the metric(and especially the recall), I would choose SVM as the final model.