In [42]:
```
import numpy as np
import pandas as pd      # built on top of numpy
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns    # built on top of matplotlib
from pandas.api.types import CategoricalDtype # enables specifying categorical agetype

from sklearn.preprocessing import Imputer, MinMaxScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, r
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
```

In [43]:

In [44]:

Out[44]:

	preg	plas	pres	skin	insu	mass	pedi	age	cla
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.3489
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.4769
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.0000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.0000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.0000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.0000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.0000

In [45]:
```
diabetes2 = diabetes.copy(deep=True)
# use replace as pure function:
diabetes2['plas'] = diabetes['plas'].replace(0,np.NaN)
diabetes2['pres'] = diabetes['pres'].replace(0,np.NaN)
diabetes2['skin'] = diabetes['skin'].replace(0,np.NaN)
diabetes2['insu'] = diabetes['insu'].replace(0,np.NaN)
diabetes2['mass'] = diabetes['mass'].replace(0,np.NaN)

# use replace as mutator by setting arg inplace=True
#diabetes2['pedi'].replace(0,np.NaN, inplace=True)
#diabetes.describe()
#diabetes2.describe()
#len(diabetes2)
#nullGest = df2.gestation.isnull()# nullGest is array of boolean
#countNullGest = len(nullGest[nullGest==True])#take slice to count how many True
#print countNullGest
#print df2.isnull()
#df2.isnull().sum(axis=0)# sum along columns
#diabetes2=diabetes2.dropna()
diabetes2.isnull().sum(axis=0)
#len(diabetes2)
#class1 =diabetes2['class']==0
```

Out[45]:
```
preg       0
plas       5
pres      35
skin     227
insu     374
mass      11
pedi       0
age        0
class      0
dtype: int64
```

In [46]:
```
print diabetes2.apply(np.nanmedian, axis = 0) #calculate medians to compare with means
```

```
preg       3.0000
plas     117.0000
pres      72.0000
skin      29.0000
insu     125.0000
mass      32.3000
pedi       0.3725
age       29.0000
class      0.0000
dtype: float64
```

Out[46]:

	preg	plas	pres	skin	insu	mass	pedi	age	cla
count	768.000000	763.000000	733.000000	541.000000	394.000000	757.000000	768.000000	768.000000	768.0000
mean	3.845052	121.686763	72.405184	29.153420	155.548223	32.457464	0.471876	33.240885	0.3489
std	3.369578	30.535641	12.382158	10.476982	118.775855	6.924988	0.331329	11.760232	0.4769
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.0000
25%	1.000000	99.000000	64.000000	22.000000	76.250000	27.500000	0.243750	24.000000	0.0000
50%	3.000000	117.000000	72.000000	29.000000	125.000000	32.300000	0.372500	29.000000	0.0000
75%	6.000000	141.000000	80.000000	36.000000	190.000000	36.600000	0.626250	41.000000	1.0000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.0000

In [47]:
```
X = diabetes2.drop(labels=['class','insu','skin'], axis=1)
#print X
y = diabetes2.loc[:,'class'] # alt: use iloc for index based data selection
#print y
print y.unique()
X_col_names = X.columns.values
```

```
[1 0]
```

In [48]:
```
imp_x = Imputer(missing_values='NaN', strategy='mean', axis=0)
X_train = imp_x.fit_transform(X_train)# # fit AND transform training set
X_test = imp_x.transform(X_test) # transform test set on scale fitted to training set
```

```
614
```

In [49]:
```
(_, idx, counts) = np.unique(y_train, return_index=True, return_counts=True)
```

```
[0 1] [2 0] [400 214]
```

In [50]:
```

def svc_call(C_val,gamma_val,kernel_val,class_weight_val,degree_val,cache_size_val,prc

    return SVC(C=C_val,gamma=gamma_val,kernel=kernel_val,class_weight=class_weight_val

def predict_accuracy(y_test,y_pred):

    pTot = accuracy_score(y_test, y_pred)

    return pTot

def plot_auc(svc,X_train,y_train,y_test):

    probas_ = svc.fit(X_train_minmax, y_train).predict_proba(X_test_minmax)
    fpr, tpr, thresholds = roc_curve(y_test, probas_[:, 1])  # use the probs of (class
    roc_auc = auc(fpr, tpr)
    #print "thresholds", thresholds
    #print "probas_", probas_
    print "AUC using predict_proba", roc_auc
    plt.figure()
    #plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc, lw=4 ) # plot ROC
    plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc, lw=3, color ="#0000
    plt.plot([0, 1], [0, 1], 'k--') # also plot black dashed line (k=black) from (0,0)

    # Set x and y ranges, labels, title and legend
    plt.xlim([-0.005, 1.0])  #x range basically from 0 to 1: start range a bit to left
    plt.ylim([0.0, 1.005])   #0 range basically from 0 to 1: extend range a bit above
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

In [51]:
```
C_val=100
gamma_val='auto'
kernel_val = 'rbf'
class_weight_val='balanced'
cache_size_val=1000
probability_val=True
degree_val=3

svc = svc_call(C_val,gamma_val,kernel_val,class_weight_val,degree_val,cache_size_val,p

print svc
print

clf = svc.fit(X_train, y_train) # trains the classifier on the training set
y_pred = svc.predict(X_test)    # tests the classifier on the test set


pTot = predict_accuracy(y_test,y_pred)

print "Prediction accuracy: ",pTot
```

```
SVC(C=100, cache_size=1000, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=True, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Prediction accuracy:  0.642857142857
```

In [52]:
```
cm = confusion_matrix(y_test, y_pred)
```

Out[52]:
```
array([[99,  1],
       [54,  0]], dtype=int64)
```

In [53]:
```
report = classification_report(y_test, y_pred)
```

```
             precision    recall  f1-score   support

          0       0.65      0.99      0.78       100
          1       0.00      0.00      0.00        54

avg / total       0.42      0.64      0.51       154
```

In [54]:
```
print X_test #compare before/after scaling
min_max_scaler = MinMaxScaler()
X_train_minmax = min_max_scaler.fit_transform(X_train)# fit AND transform training set
X_test_minmax = min_max_scaler.transform(X_test)# test set transform only, no fit
```

```
[[  4.00000000e+00   9.90000000e+01   7.20000000e+01   2.56000000e+01
    2.94000000e-01   2.80000000e+01]
 [  1.00000000e+00   1.43000000e+02   7.40000000e+01   2.62000000e+01
    2.56000000e-01   2.10000000e+01]
 [  1.00000000e+00   8.90000000e+01   7.60000000e+01   3.12000000e+01
    1.92000000e-01   2.30000000e+01]
 [  7.00000000e+00   1.19000000e+02   7.23118644e+01   2.52000000e+01
    2.09000000e-01   3.70000000e+01]
 [  5.00000000e+00   1.26000000e+02   7.80000000e+01   2.96000000e+01
    4.39000000e-01   4.00000000e+01]
 [  6.00000000e+00   1.14000000e+02   8.80000000e+01   2.78000000e+01
    2.47000000e-01   6.60000000e+01]
 [  1.00000000e+00   9.20000000e+01   6.20000000e+01   1.95000000e+01
    4.82000000e-01   2.50000000e+01]
 [  1.00000000e+00   1.09000000e+02   3.80000000e+01   2.31000000e+01
    4.07000000e-01   2.60000000e+01]
 [  1.00000000e+00   1.72000000e+02   6.80000000e+01   4.24000000e+01
    7.02000000e-01   2.80000000e+01]
 [  0.00000000e+00   1.02000000e+02   5.20000000e+01   2.51000000e+01
```

In [55]:
```
svc = svc_call(C_val,gamma_val,kernel_val,class_weight_val,degree_val,cache_size_val,p

print svc
print

clf = svc.fit(X_train_minmax, y_train) # trains the classifier on the training set
y_pred_minmax = svc.predict(X_test_minmax)    # tests the classifier on the test set

pTot = predict_accuracy(y_test,y_pred_minmax)
```

```
SVC(C=100, cache_size=1000, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=True, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Prediction accuracy:  0.75974025974
```

In [56]:
```
cm = confusion_matrix(y_test, y_pred_minmax)
print cm
report = classification_report(y_test, y_pred_minmax)
```

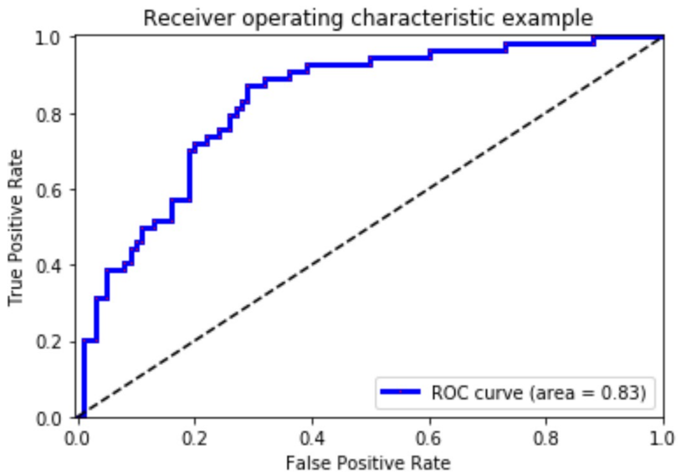
```
[[76 24]
 [13 41]]
             precision    recall  f1-score   support

          0       0.85      0.76      0.80       100
          1       0.63      0.76      0.69        54

avg / total       0.78      0.76      0.76       154
```

In [57]:
```
%matplotlib inline
```

```
AUC using predict_proba 0.831481481481
```



In [58]:
```
C_range = 10.0 ** np.arange(-2, 4)
#gamma_range = 10.0 ** np.arange(-3, 3)
gamma_range = [.01, .1, 1, 'auto', 10, 100]
print gamma_range
param_grid = dict(gamma=gamma_range, C=C_range)
```

```
[0.01, 0.1, 1, 'auto', 10, 100]
```

Out[58]:
```
{'C': array([  1.00000000e-02,   1.00000000e-01,   1.00000000e+00,
          1.00000000e+01,   1.00000000e+02,   1.00000000e+03]),
 'gamma': [0.01, 0.1, 1, 'auto', 10, 100]}
```

In [59]:
```
# Default is 3-fold cross validation
grid = GridSearchCV(SVC(kernel='rbf',cache_size=1000, probability=True), param_grid=pa
#grid = GridSearchCV(SVC(kernel='rbf', class_weight='balanced', cache_size=1000, proba
grid.fit(X_train_minmax, y_train)# run the grid search on the training data only
best_C = grid.best_estimator_.C
best_gamma = grid.best_estimator_.gamma
#print best_C
#print best_gamma
print "The best C and gamma for rbf is: %.5s, %.5s " % (best_C, best_gamma)
grid.best_estimator_
```

```
The best C and gamma for rbf is: 1.0, auto
```

Out[59]:
```
SVC(C=1.0, cache_size=1000, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=True, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [60]:
```
best_predict_minmax = grid.best_estimator_.predict(X_test_minmax)
pTot = accuracy_score(y_test, best_predict_minmax)
print "Prediction accuracy: ",pTot
cm = confusion_matrix(y_test, best_predict_minmax)
print cm
report = classification_report(y_test, best_predict_minmax)
```

```
Prediction accuracy:  0.75974025974
[[92  8]
 [29 25]]
             precision    recall  f1-score   support

          0       0.76      0.92      0.83       100
          1       0.76      0.46      0.57        54

avg / total       0.76      0.76      0.74       154
```

In [61]:
```
C_test = 10
gamma_test = 'auto'
class_weight_val=None

#test_svc = SVC(C=10, gamma='auto',kernel='rbf', cache_size=1000, probability=True)
test_svc = svc_call(C_test,gamma_test,kernel_val,class_weight_val,degree_val,cache_siz
print test_svc
print

test_svc.fit(X_train_minmax, y_train)#

y_pred_test = test_svc.predict(X_test_minmax)

print "Prediction accuracy: ",predict_accuracy(y_test,y_pred_test)
```

```
SVC(C=10, cache_size=1000, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=True, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Prediction accuracy:  0.772727272727
```

The above test SVC gives the best accuracy which takes Soft-margin value(C) as 10, Gamma as 'auto' and the kernel used is 'Radial basis function'

In [62]:
```
cm = confusion_matrix(y_test, y_pred_test)
print cm
report = classification_report(y_test, y_pred_test)
```

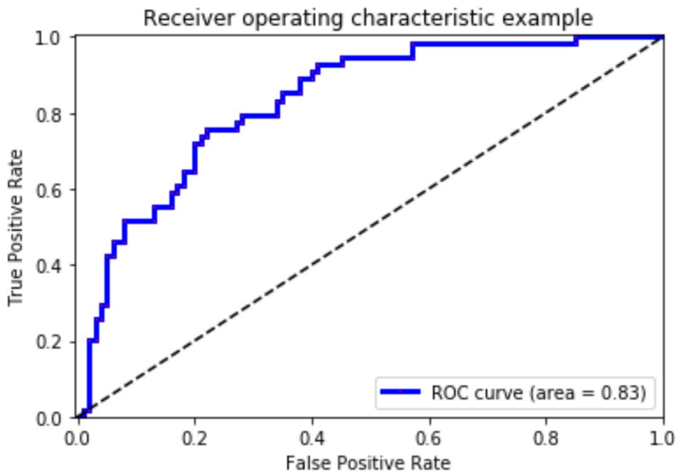
```
[[92  8]
 [27 27]]
             precision    recall  f1-score   support

          0       0.77      0.92      0.84       100
          1       0.77      0.50      0.61        54

avg / total       0.77      0.77      0.76       154
```

In [63]:

```
AUC using predict_proba 0.832962962963
```



In [64]:

```
kernel_val='poly'
degree_val=1

svc = svc_call(C_val,gamma_val,kernel_val,class_weight_val,degree_val,cache_size_val,p
#svc = SVC(kernel='rbf', cache_size=1000, probability=True)
print svc
print

svc.fit(X_train_minmax, y_train) # trains the classifier on the training set
y_pred_minmax = svc.predict(X_test_minmax)     # tests the classifier on the test set
```

```
SVC(C=100, cache_size=1000, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=1, gamma='auto', kernel='poly',
  max_iter=-1, probability=True, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Prediction accuracy:  0.75974025974
```

In [65]:

```
cm = confusion_matrix(y_test, y_pred_minmax)
print cm
report = classification_report(y_test, y_pred_minmax)
```

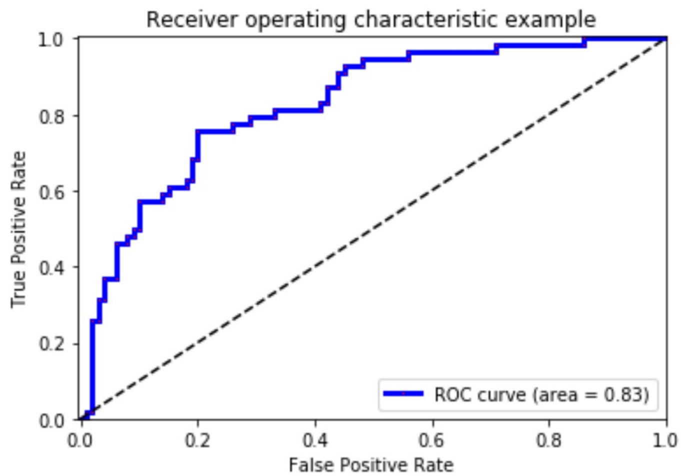
```
[[91  9]
 [28 26]]
             precision    recall  f1-score   support

          0       0.76      0.91      0.83       100
          1       0.74      0.48      0.58        54

avg / total       0.76      0.76      0.74       154
```

In [66]:

```
AUC using predict_proba 0.828518518519
```



In [67]:

```
# Default is 3-fold cross validation
grid = GridSearchCV(SVC(kernel='poly',degree =1,cache_size=1000, probability=True), pa
#grid = GridSearchCV(SVC(kernel='rbf', class_weight='balanced', cache_size=1000, proba
grid.fit(X_train_minmax, y_train)# run the grid search on the training data only
best_C = grid.best_estimator_.C
best_gamma = grid.best_estimator_.gamma
#print best_C
#print best_gamma
print "The best C and gamma for poly is: %.5s, %.5s " % (best_C, best_gamma)
grid.best_estimator_
```

```
The best C and gamma for poly is: 0.01, 100
```

Out[67]:

```
SVC(C=0.01, cache_size=1000, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=1, gamma=100, kernel='poly',
  max_iter=-1, probability=True, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [68]:

```
best_predict_minmax = grid.best_estimator_.predict(X_test_minmax)
pTot = accuracy_score(y_test, best_predict_minmax)
print "Prediction accuracy: ",pTot
cm = confusion_matrix(y_test, best_predict_minmax)
print cm
report = classification_report(y_test, best_predict_minmax)
```

```
Prediction accuracy:  0.753246753247
[[91  9]
 [29 25]]
             precision    recall  f1-score   support

          0       0.76      0.91      0.83       100
          1       0.74      0.46      0.57        54

avg / total       0.75      0.75      0.74       154
```

In [69]:
```
C_test = 10
gamma_test = 'auto'
class_weight_val=None

#test_svc = SVC(C=10, gamma='auto',kernel='rbf', cache_size=1000, probability=True)
test_svc = svc_call(C_test,gamma_test,kernel_val,class_weight_val,degree_val,cache_siz
print test_svc
print

test_svc.fit(X_train_minmax, y_train) # trains the classifier on the training set
y_pred_minmax_test = test_svc.predict(X_test_minmax)    # tests the classifier on the

print "Prediction accuracy: ",predict_accuracy(y_test,y_pred_minmax_test)
```

```
SVC(C=10, cache_size=1000, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=1, gamma='auto', kernel='poly',
  max_iter=-1, probability=True, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Prediction accuracy:  0.753246753247
```

In [70]:
```
cm = confusion_matrix(y_test, y_pred_minmax_test)
print cm
report = classification_report(y_test, y_pred_minmax_test)
```

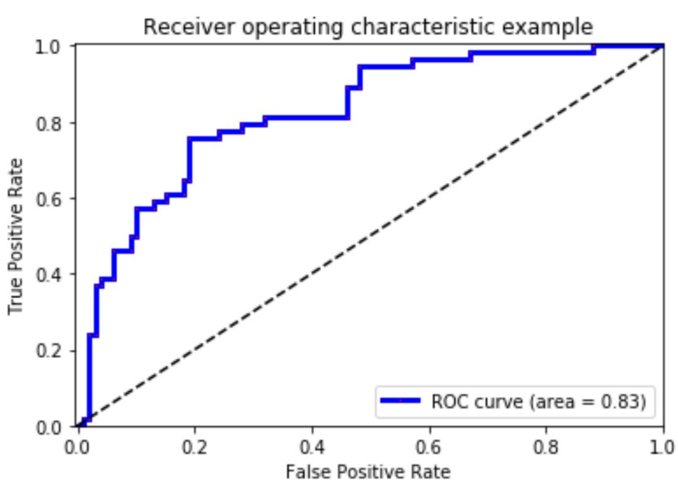
```
[[91  9]
 [29 25]]
             precision    recall  f1-score   support

          0       0.76      0.91      0.83       100
          1       0.74      0.46      0.57        54

avg / total       0.75      0.75      0.74       154
```

In [71]:

```
AUC using predict_proba 0.827037037037
```



In [ ]: