Wanat_Assignment2_final

July 7, 2019

In [1]:
```
# Jump-Start for the Bank Marketing Study
# as described in Marketing Data Science: Modeling Techniques
# for Predictive Analytics with R and Python (Miller 2015)

# jump-start code revised by Thomas W. Milller (2018/10/07)

# Scikit Learn documentation for this assignment:
# http://scikit-learn.org/stable/auto_examples/classification/
#   plot_classifier_comparison.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.BernoulliNB.score
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.linear_model.LogisticRegression.html
# http://scikit-learn.org/stable/modules/model_evaluation.html
# http://scikit-learn.org/stable/modules/generated/
#  sklearn.model_selection.KFold.html

# prepare for Python version 3x features and functions
# comment out for Python 3.x execution
# from __future__ import division, print_function
# from future_builtins import ascii, filter, hex, map, oct, zip

# seed value for random number generators to obtain reproducible results
RANDOM_SEED = 1

# import base packages into the namespace for this program
import numpy as np
import pandas as pd
import pandas_profiling
import matplotlib
import matplotlib.pyplot as plt  # static plotting
import seaborn as sns  # pretty plotting, including heat map
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
```

```
from sklearn.preprocessing import binarize
from sklearn.metrics import precision_recall_curve
from sklearn.naive_bayes import BernoulliNB
#from sklearn.cross_validation import cross_val_score
from sklearn.model_selection import cross_val_score
import math
```

0.1 Defined functions

```
In [2]: # correlation heat map setup for seaborn
        def corr_chart(df_corr):
            corr=df_corr.corr()
            #screen top half to get a triangle
            top = np.zeros_like(corr, dtype=np.bool)
            top[np.triu_indices_from(top)] = True
            fig=plt.figure()
            fig, ax = plt.subplots(figsize=(12,12))
            sns.heatmap(corr, mask=top, cmap='coolwarm',
                center = 0, square=True,
                linewidths=.5, cbar_kws={'shrink':.5},
                annot = True, annot_kws={'size': 9}, fmt = '.3f')
            plt.xticks(rotation=45) # rotate variable labels on columns (x axis)
            plt.yticks(rotation=0) # use horizontal variable labels on rows (y axis)
            plt.title('Correlation Heat Map')
            plt.savefig('plot-corr-map.pdf',
                bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
                orientation='portrait', papertype=None, format=None,
                transparent=True, pad_inches=0.25, frameon=None)

        np.set_printoptions(precision=3)
```

```
In [3]: # define a function to return model metrics for evaluation

        def model_metrics(y_known, y_pred):
            y_test = y_known
            y_pred_class = y_pred
            confusion = metrics.confusion_matrix(y_test, y_pred_class)
            TP = confusion[1, 1]
            TN = confusion[0, 0]
            FP = confusion[0, 1]
            FN = confusion[1, 0]
            accuracy = metrics.accuracy_score(y_test, y_pred_class)
            class_error = 1 - metrics.accuracy_score(y_test, y_pred_class)
            sensitivity = metrics.recall_score(y_test, y_pred_class)
            specificity = TN / (TN + FP)
            false_positive_rate = FP / float(TN + FP)
            precision = TP / float(TP + FP)
            f1_score = metrics.f1_score(y_test, y_pred_class)
```

```
#     F1 = 2*precision*sensitivity/(precision + sensitivity)
    print('The model metrics are:',
          '\naccuracy:', accuracy,
          '\nclassification error:', class_error,
          '\nsensitivity:', sensitivity,
          '\nspecificity:', specificity,
          '\nfalse positive rate:', false_positive_rate,
          '\nprecision:', precision,
           '\nF1 score:', f1_score,
#            '\nF1 by hand:', F1,
          '\nconfusion matrix:')
    return(confusion)
```

In [4]:
```
# define a function that accepts a threshold and
# prints sensitivity and specificity

def evaluate_threshold(threshold):
    print('Sensitivity:', tpr[thresholds > threshold][-1])
    print('Specificity:', 1 - fpr[thresholds > threshold][-1])
```

In [5]:
```
# define a function that accepts the fpr and tpr values
# from the roc_curve function and plot the ROC curve

def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC curve for response')
```

In [6]:
```
# define a function that accepts the recall and precision values
# from the precision_recall_curve function and plot the PR curve

def plot_pr_curve(recall, precision, label=None):
    plt.plot(recall, precision, linewidth=2, label=label)
#     plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('PR curve for response')
```

In [7]:
```
# define a function that accepts the scores from the cross validation
# and print the scores, mean, and standard deviation

def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

0.2 Import data set and prepare for analysis

In [8]:
```
# initial work with the smaller data set
bank = pd.read_csv('bank.csv', sep = ';')  # start with smaller data set
# examine the shape of original input data
print(bank.shape)
```

```
(4521, 17)
```

In [9]:
```
#total number of NaN values in each column
bank.isnull().sum()
```

Out[9]:
```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
response     0
dtype: int64
```

In [10]:
```
print(bank.response.head())
```

```
0    no
1    no
2    no
3    no
4    no
Name: response, dtype: object
```

In [11]:
```
display(bank.head())
```

```
   age         job  marital  education default  balance housing loan  \
0   30  unemployed  married    primary      no     1787      no   no
1   33    services  married  secondary      no     4789     yes  yes
2   35  management   single   tertiary      no     1350     yes   no
3   30  management  married   tertiary      no     1476     yes  yes
```

```
4   59  blue-collar  married  secondary      no        0     yes   no

    contact  day month  duration  campaign  pdays  previous poutcome response
0  cellular   19   oct        79         1     -1         0  unknown       no
1  cellular   11   may       220         1    339         4  failure       no
2  cellular   16   apr       185         1    330         1  failure       no
3   unknown    3   jun       199         4     -1         0  unknown       no
4   unknown    5   may       226         1     -1         0  unknown       no
```

```
In [12]: bank.describe()
```

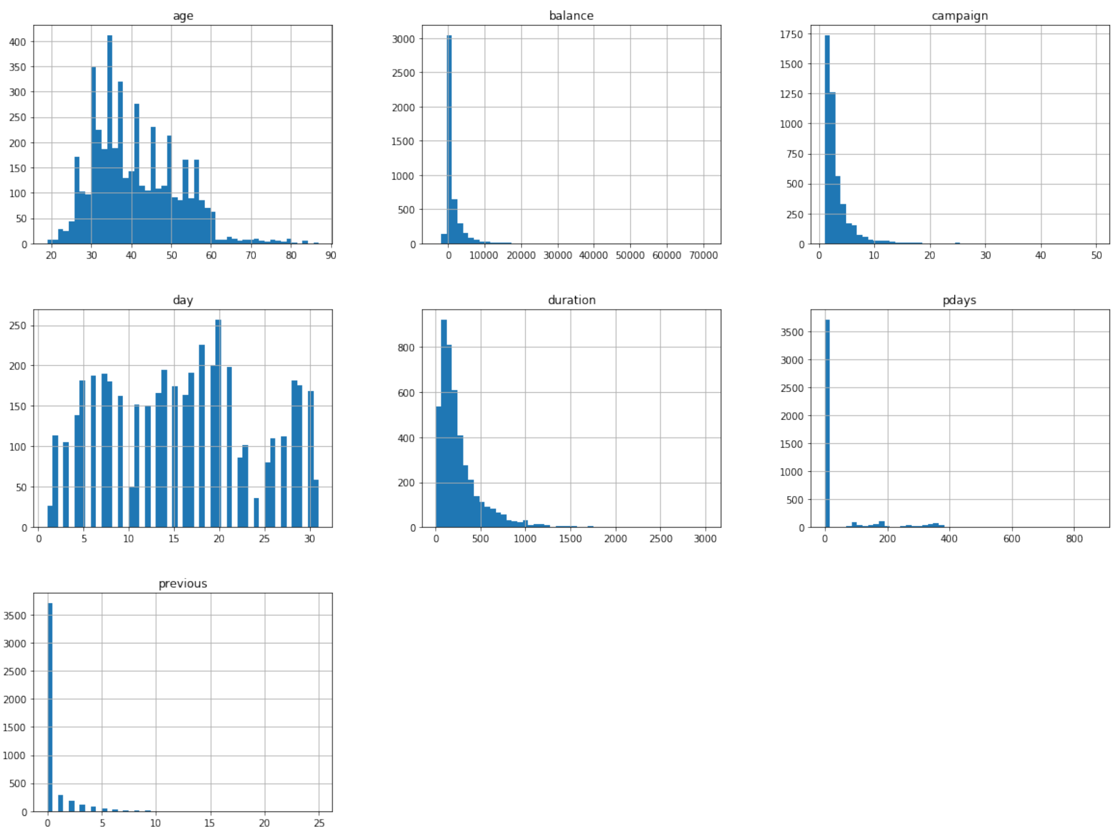
```
Out[12]:                age       balance          day     duration     campaign  \
         count  4521.000000   4521.000000  4521.000000  4521.000000  4521.000000
         mean     41.170095   1422.657819    15.915284   263.961292     2.793630
         std      10.576211   3009.638142     8.247667   259.856633     3.109807
         min      19.000000  -3313.000000     1.000000     4.000000     1.000000
         25%      33.000000     69.000000     9.000000   104.000000     1.000000
         50%      39.000000    444.000000    16.000000   185.000000     2.000000
         75%      49.000000   1480.000000    21.000000   329.000000     3.000000
         max      87.000000  71188.000000    31.000000  3025.000000    50.000000

                      pdays     previous
         count  4521.000000  4521.000000
         mean     39.766645     0.542579
         std     100.121124     1.693562
         min      -1.000000     0.000000
         25%      -1.000000     0.000000
         50%      -1.000000     0.000000
         75%      -1.000000     0.000000
         max     871.000000    25.000000
```

```
In [13]: bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 17 columns):
age          4521 non-null int64
job          4521 non-null object
marital      4521 non-null object
education    4521 non-null object
default      4521 non-null object
balance      4521 non-null int64
housing      4521 non-null object
loan         4521 non-null object
contact      4521 non-null object
day          4521 non-null int64
month        4521 non-null object
duration     4521 non-null int64
```

```
campaign      4521 non-null int64
pdays         4521 non-null int64
previous      4521 non-null int64
poutcome      4521 non-null object
response      4521 non-null object
dtypes: int64(7), object(10)
memory usage: 600.5+ KB
```

```
In [14]: %matplotlib inline
         bank.hist(bins=50, figsize=(20,15))
         plt.savefig('bank_hist.pdf')
         plt.show()
```




```
In [15]: # drop observations with missing data, if any
         bank.dropna()
         # examine the shape of input data after dropping missing data
         print(bank.shape)
```

```
(4521, 17)
```

```
In [16]: # look at the list of column names, note that y is the response
         list(bank.columns.values)
```

```
Out[16]: ['age',
          'job',
          'marital',
          'education',
          'default',
          'balance',
          'housing',
          'loan',
          'contact',
          'day',
          'month',
          'duration',
          'campaign',
          'pdays',
          'previous',
          'poutcome',
          'response']
```

```
In [17]: # look at the beginning of the DataFrame
         bank.head()
```

```
Out[17]:    age          job  marital  education default  balance housing loan  \
         0   30   unemployed  married    primary      no     1787      no   no
         1   33     services  married  secondary      no     4789     yes  yes
         2   35   management   single   tertiary      no     1350     yes   no
         3   30   management  married   tertiary      no     1476     yes  yes
         4   59  blue-collar  married  secondary      no        0     yes   no

             contact  day month  duration  campaign  pdays  previous poutcome response
         0  cellular   19   oct        79         1     -1         0  unknown       no
         1  cellular   11   may       220         1    339         4  failure       no
         2  cellular   16   apr       185         1    330         1  failure       no
         3   unknown    3   jun       199         4     -1         0  unknown       no
         4   unknown    5   may       226         1     -1         0  unknown       no
```

```
In [18]: # mapping function to convert text no/yes to integer 0/1
         convert_to_binary = {'no' : 0, 'yes' : 1}

         # define binary variable for having credit in default
         default = bank['default'].map(convert_to_binary)

         # define binary variable for having a mortgage or housing loan
         housing = bank['housing'].map(convert_to_binary)

         # define binary variable for having a personal loan
         loan = bank['loan'].map(convert_to_binary)
```

```
# define response variable to use in the model
response = bank['response'].map(convert_to_binary)
```

In [19]:
```
# gather three explanatory variables and response into a numpy array
# here we use .T to obtain the transpose for the structure we want
model_data = np.array([np.array(default), np.array(housing), np.array(loan),
    np.array(response)]).T

# examine the shape of model_data, which we will use in subsequent modeling
print(model_data.shape)

# the rest of the program should set up the modeling methods
# and evaluation within a cross-validation design
```

```
(4521, 4)
```

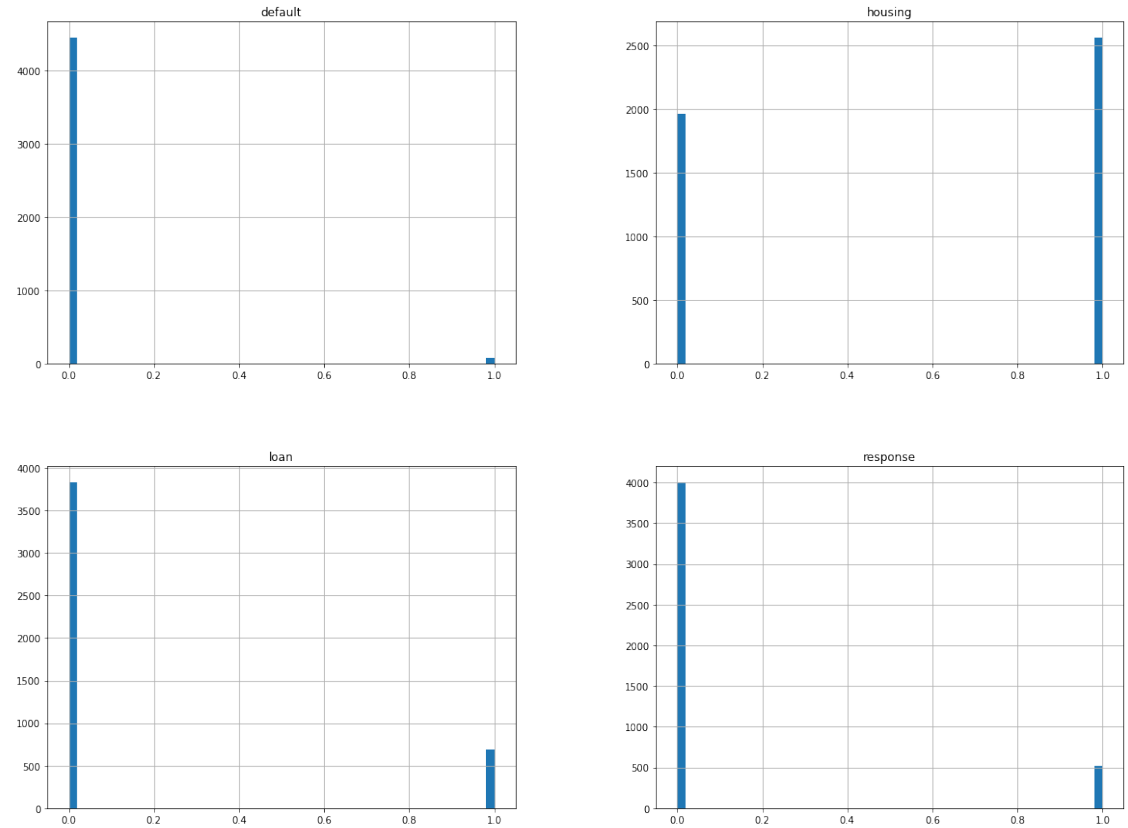
In [20]:
```
model_data_df = pd.DataFrame(model_data)
model_data_df.columns = ['default', 'housing', 'loan', 'response']
```

In [21]:
```
model_data_df.head()
```

Out[21]:

	default	housing	loan	response
0	0	0	0	0
1	0	1	1	0
2	0	1	0	0
3	0	1	1	0
4	0	1	0	0

In [22]:
```
model_data_df.hist(bins=50, figsize=(20,15))
plt.savefig('model_data_df_hist.pdf')
plt.show()
```




```
In [23]: # value counts of the response variable
         # there is a low rate of subscription to a term deposit
         # only 521 clients have a term deposit
         model_data_df.response.value_counts()
```

```
Out[23]: 0    4000
         1     521
         Name: response, dtype: int64
```

```
In [24]: #split data and response

         model_data_df_X = model_data_df.drop('response', axis=1)
         model_data_df_y = model_data_df.response.copy()
```

```
In [25]: model_data_df_X.head()
```

Out[25]:

	default	housing	loan
0	0	0	0
1	0	1	1
2	0	1	0
3	0	1	1
4	0	1	0

```
In [26]: model_data_df_y.head()
```

```
Out[26]: 0    0
         1    0
         2    0
         3    0
         4    0
         Name: response, dtype: int64
```

0.3 Split the data into train and test sets

```
In [27]: #from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(
             model_data_df_X, model_data_df_y, test_size=0.2, random_state=42)
```

0.4 Logistic Regression Model, C = 100

```
In [28]: # Create a logistic regression model on the data
         # C is the hyperparameter controlling the regularization
         # strength of a Scikit-Learn LogisticRegression model.
         # The higher the value of C, the less the model is regularized.

         # Instantiate model
         log_reg100 = LogisticRegression(C=100)

         # Fit the model
         log_reg100.fit(X_train, y_train)
```

```
Out[28]: LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

```
In [29]: # make predictions for the testing set
         y_predictions100 = log_reg100.predict(X_test)
```

```
In [30]: # look at the first 10 entries for predictions
         y_predictions100[0:10]
```

```
Out[30]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
In [31]: # look at the first 10 entries for the true values
         y_test[0:10]
```

```
Out[31]: 2398    0
         800     0
         2288    0
         2344    0
```

```
3615    0
3548    0
1115    0
4053    0
838     0
4141    0
Name: response, dtype: int64
```

In [32]:
```python
# make predicted probabilities for the predictions
y_predict_prob100 = log_reg100.predict_proba(X_test)
```

In [33]:
```python
# Look at the first ten rows of predicted probabilities
# of response class membership.
# The first column is the predicted probability that the
# observation is a member of class 0.
# The second column is the predicted probability that the
# observation is a member of class 1.

y_predict_prob100[0:10]
```

Out[33]:
```
array([[0.871, 0.129],
       [0.831, 0.169],
       [0.831, 0.169],
       [0.909, 0.091],
       [0.831, 0.169],
       [0.831, 0.169],
       [0.909, 0.091],
       [0.831, 0.169],
       [0.831, 0.169],
       [0.831, 0.169]])
```

In [34]:
```python
# the first argument is true values,
# the second argument is predicted values
# this produces a 2x2 numpy array (matrix)

confusion100 = metrics.confusion_matrix(y_test, y_predictions100)
print(confusion100)
```

```
[[807   0]
 [ 98   0]]
```

In [35]:
```python
# precision, recall, F1 score, and count of response variable
# of logistic regression model, C =100 when threshold set to 0.5

print(classification_report(y_test, y_predictions100))
```

```
    precision    recall  f1-score   support
```

```
             0       0.89      1.00      0.94       807
             1       0.00      0.00      0.00        98

avg / total       0.80      0.89      0.84       905
```

```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: U
  'precision', 'predicted', average, warn_for)
```

0.5 Adjusting the classification threshold

In [36]:
```python
# store the predicted probabilities for class 1 of the response
# for the logistic regression, C = 100 model
y_pred_prob100_class1 = log_reg100.predict_proba(X_test)[:,1]
```

In [37]:
```python
# look at the first ten entries for predicted
# probabilities for class 1
y_pred_prob100_class1[0:10]
```

Out[37]:
```
array([0.129, 0.169, 0.169, 0.091, 0.169, 0.169, 0.091, 0.169, 0.169,
       0.169])
```

In [38]:
```python
# the default threshold for predicted probabilities to be classified
# as 0 or 1 is 0.5
# let's see what would happen if the default is set to 0.1
# predict response if the predicted probability is greater than 0.1
# it will return 1 for all values above 0.1 and 0 otherwise
# results are 2D so we slice out the first column

y_pred_class100 = binarize(y_pred_prob100_class1.reshape(-1, 1), 0.1)
```

In [39]:
```python
# print the first 10 predicted classes with the lower threshold
y_pred_class100[0:10]
```

Out[39]:
```
array([[1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.]])
```

In [40]:
```python
# previous confusion matrix (default threshold of 0.5)
print(confusion100)
```

```
[[807   0]
 [ 98   0]]
```

```
In [41]: # new confusion matrix (threshold set to 0.1)

         confusion100_threshold1 = metrics.confusion_matrix(y_test, y_pred_class100)
         print(confusion100_threshold1)
```

```
[[528 279]
 [ 50  48]]
```

```
In [42]: # precision, recall, F1 score, and count of response variable
         # of logistic regression model, C =100 when threshold set to 0.1

         print(classification_report(y_test, y_pred_class100))
```

```
             precision    recall  f1-score   support

          0       0.91      0.65      0.76       807
          1       0.15      0.49      0.23        98

avg / total       0.83      0.64      0.70       905
```

```
In [43]: # evaluate logistic regression model, C = 100
         # when threshold set to 0.1

         model_metrics(y_test, y_pred_class100)
```

```
The model metrics are:
accuracy: 0.636464088397779
classification error: 0.36353591160221
sensitivity: 0.4897959183673469
specificity: 0.654275092936803
false positive rate: 0.34572490706319703
precision: 0.14678899082568808
F1 score: 0.2258823529411765
confusion matrix:
```

```
Out[43]: array([[528, 279],
                [ 50,  48]])
```

```
In [44]: # evaluate logistic regression model, C = 100
         # when threshold set to default 0.5
```

```
model_metrics(y_test, y_predictions100)

#y_predictions100 is the predicted response value
```

```
The model metrics are:
accuracy: 0.8917127071823204
classification error: 0.10828729281767957
sensitivity: 0.0
specificity: 1.0
false positive rate: 0.0
precision: nan
F1 score: 0.0
confusion matrix:


/Users/jmwanat/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:16: RuntimeWarning:
  app.launch_new_instance()
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: U
  'precision', 'predicted', average, warn_for)
```

Out[44]:
```
array([[807,   0],
       [ 98,   0]])
```

0.6 ROC Curve

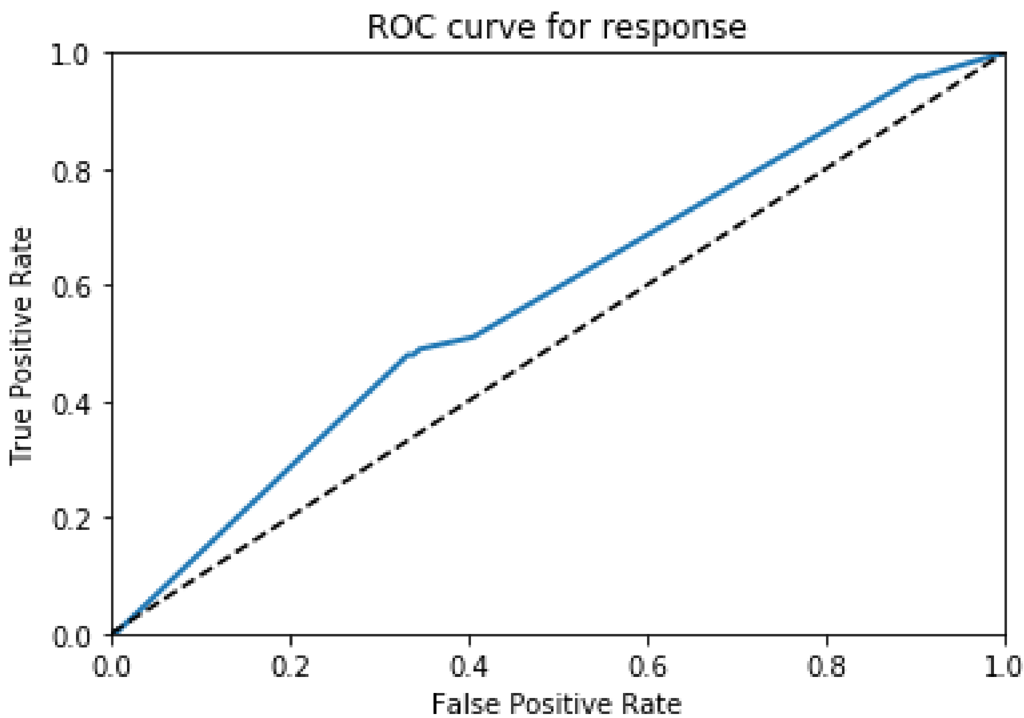
In [45]:
```
# make predicted probabilities for the predictions
# logistic regression, C = 100
# store the predicted probabilities for class 1 of response

y_pred_prob100_class1 = log_reg100.predict_proba(X_test)[:, 1]
```

In [46]:
```
# the first argument is true values,
# the second argument is predicted probabilities

# pass y_test and y_pred_prob
# do not use y_pred_class, because it will
# give incorrect results without generating an error
# roc_curve returns 3 objects fpr, tpr, thresholds
# fpr: false positive rate
# tpr: true positive rate

fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob100_class1)
```

In [47]:
```
# a check of thresholds
thresholds
```

Out[47]:
```
array([1.229, 0.229, 0.169, 0.129, 0.128, 0.092, 0.091, 0.068, 0.048])
```

In [48]:
```
# plot the ROC curve for the logistic regression, C = 100

plot_roc_curve(fpr, tpr)
plt.savefig('ROC_logistic_C100_plot.pdf')
plt.show()
```



In [49]:
```
# evaluate the logistic regression model, C = 100
# sensitivity and specificity when the threshold is set to 0.5

print('When the threshold is set to 0.5')
evaluate_threshold(0.5)
```

```
When the threshold is set to 0.5
Sensitivity: 0.0
Specificity: 1.0
```

In [50]:
```
# evaluate the logistic regression model, C = 100
# sensitivity and specificity when the threshold is set to 0.1

print('When the threshold is set to 0.1')
evaluate_threshold(0.1)
```

```
When the threshold is set to 0.1
Sensitivity: 0.4897959183673469
```

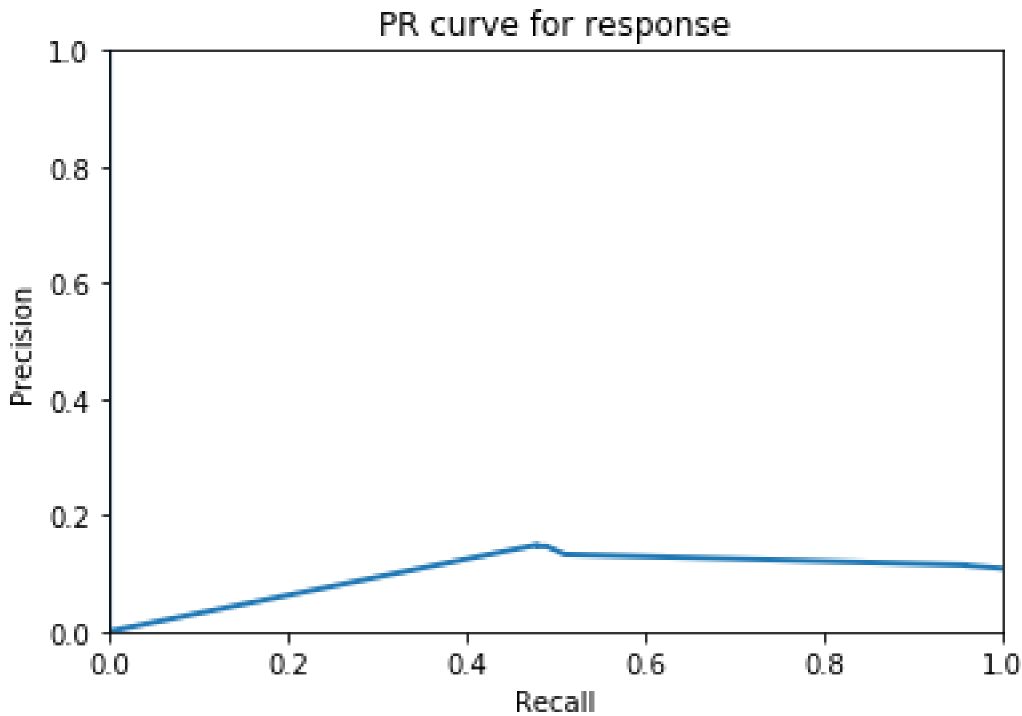
```
Specificity: 0.654275092936803
```

```
In [51]: # AUC is the percentage of the ROC plot that is underneath the curve
         # first argument is true values, second argument is predicted probabilities
         # AUC for logistic regression, C = 100

         roc_auc_y100_class1 = metrics.roc_auc_score(y_test, y_pred_prob100_class1)

         print('\nThe AUC is:', roc_auc_y100_class1)
```

```
The AUC is: 0.5752977771034065
```

0.7 PR Curve

```
In [52]: # Given the imbalance of 0 to 1 in the response category
         # let's see what the Precision-Recall curve looks like
         # for the logistic regression model, C = 100

         precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob100_class1)

In [53]: y_test[:10]

Out[53]: 2398    0
         800     0
         2288    0
         2344    0
         3615    0
         3548    0
         1115    0
         4053    0
         838     0
         4141    0
         Name: response, dtype: int64

In [54]: type(y_test)

Out[54]: pandas.core.series.Series

In [55]: y_pred_prob100_class1[:10]

Out[55]: array([0.129, 0.169, 0.169, 0.091, 0.169, 0.169, 0.091, 0.169, 0.169,
                0.169])

In [56]: # plot the PR curve for logistic regression, C = 100

         plot_pr_curve(recall, precision)
         plt.savefig('PR_logistic_C100_plot.pdf')
         plt.show()
```



0.8 Logistic Regression Model, C = 1000

In [57]:
```
# Create a logistic regression model on the data
# C is the hyperparameter controlling the regularization
# strength of a Scikit-Learn LogisticRegression model.
# The higher the value of C, the less the model is regularized.
# Let's see if there is a difference with C = 1000

# Instantiate model
log_reg1000 = LogisticRegression(C=1000)

# Fit the model
log_reg1000.fit(X_train, y_train)
```

Out[57]:
```
LogisticRegression(C=1000, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [58]:
```
# make predictions for the testing set
y_predictions1000 = log_reg1000.predict(X_test)
```

In [59]:
```
# make predicted probabilities for the predictions
y_predict_prob1000 = log_reg1000.predict_proba(X_test)
```

In [60]:
```
# store the predicted probabilities for class 1 of the response
# for the logistic regression, C = 1000 model

y_pred_prob1000_class1 = log_reg1000.predict_proba(X_test)[:,1]
```

In [61]:
```
# predict response if the predicted probability is greater than 0.1
# it will return 1 for all values above 0.1 and 0 otherwise
# results are 2D so we slice out the first column
y_pred_class1000 = binarize(y_pred_prob1000_class1.reshape(-1, 1), 0.1)
```

In [62]:
```
# precision, recall, F1 score, and count of response variable
# of logistic regression model, C =1000 when threshold set to 0.5

print(classification_report(y_test, y_predictions1000))
```

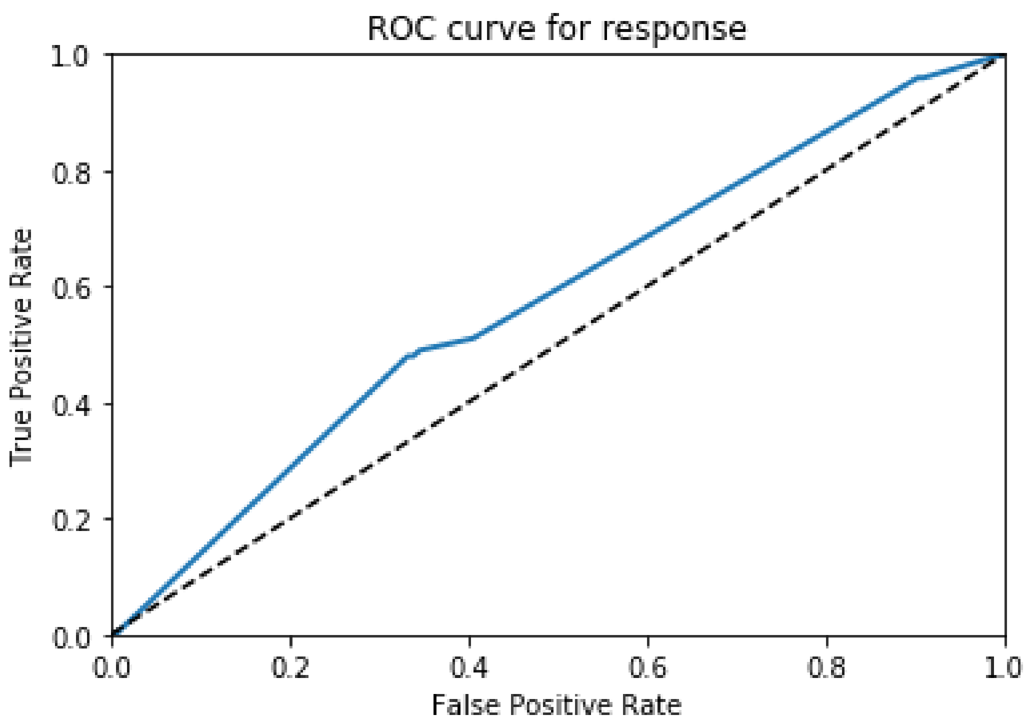
```
             precision    recall  f1-score   support

          0       0.89      1.00      0.94       807
          1       0.00      0.00      0.00        98

avg / total       0.80      0.89      0.84       905



/Users/jmwanat/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Ur
  'precision', 'predicted', average, warn_for)
```

In [63]:
```
# evaluate logistic regression model, C = 1000
# when threshold set to default 0.5
# model_metrics(y_known, y_pred):

model_metrics(y_test, y_predictions1000)
```

```
The model metrics are:
accuracy: 0.8917127071823204
classification error: 0.10828729281767957
sensitivity: 0.0
specificity: 1.0
false positive rate: 0.0
precision: nan
F1 score: 0.0
confusion matrix:


/Users/jmwanat/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:16: RuntimeWarning:
  app.launch_new_instance()
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Ur
  'precision', 'predicted', average, warn_for)
```

Out[63]:
```
array([[807,   0],
       [ 98,   0]])
```

In [64]:
```
# the first argument is true values,
# the second argument is predicted probabilities

# pass y_test and y_pred_prob
# do not use y_pred_class, because it will give
# incorrect results without generating an error
# roc_curve returns 3 objects fpr, tpr, thresholds
# fpr: false positive rate
# tpr: true positive rate

fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob1000_class1)
plot_roc_curve(fpr, tpr)
plt.savefig('ROC_logistic_C1000_plot.pdf')
plt.show()
```



In [65]:
```
# evaluate the logistic regression model, C = 1000
# sensitivity and specificity when the threshold is set to 0.5

print('When the threshold is set to 0.5')
evaluate_threshold(0.5)
```

```
When the threshold is set to 0.5
Sensitivity: 0.0
Specificity: 1.0
```

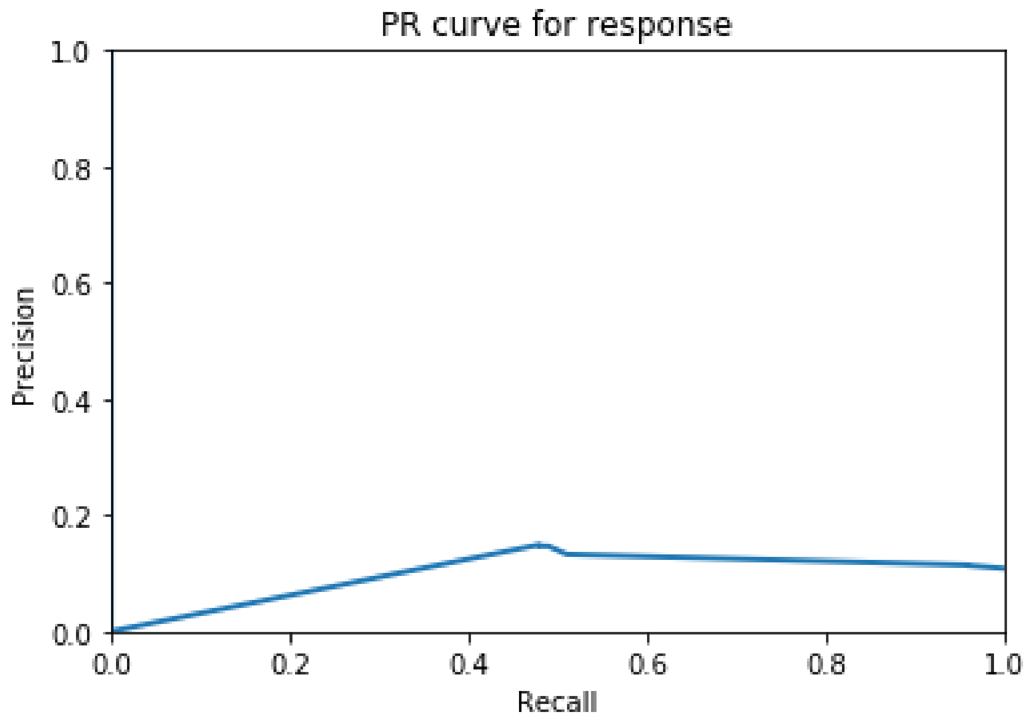
```
In [66]: # evaluate the logistic regression model, C = 1000
         # sensitivity and specificity when the threshold is set to 0.1

         print('When the threshold is set to 0.1')
         evaluate_threshold(0.1)
```

```
When the threshold is set to 0.1
Sensitivity: 0.4897959183673469
Specificity: 0.654275092936803
```

```
In [67]: # AUC is the percentage of the ROC plot that is underneath the curve
         # first argument is true values, second argument is predicted probabilities
         # AUC for logistic regression, C = 1000

         roc_auc_y100_class1 = metrics.roc_auc_score(y_test, y_pred_prob1000_class1)
         print('\nThe AUC is:', roc_auc_y100_class1)
```

```
The AUC is: 0.5752977771034065
```

```
In [68]: # Given the imbalance of 0 to 1 in the response category
         # let's see what the Precision-Recall curve looks like
         # for the logistic regression model, C = 1000

         precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob1000_class1)

         # plot the PR curve for logistic regression, C = 1000
         plot_pr_curve(recall, precision)
         plt.savefig('PR_logistic_C1000_plot.pdf')
         plt.show()
```



0.9 Naive Bayes Classifier

```
In [69]: # Create a Naive Bayes Classifier on the data


         # Instantiate model
         clf = BernoulliNB()

         # Fit the model
         clf.fit(X_train, y_train)

Out[69]: BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)

In [70]: # make predictions for the testing set
         clf_y_predictions = clf.predict(X_test)

In [71]: # make predicted probabilities for the predictions
         clf_y_predict_prob = clf.predict_proba(X_test)

In [72]: # store the predicted probabilities for class 1 of the response
         # for the naive bayes classifier

         clf_y_pred_prob_class1 = clf.predict_proba(X_test)[:,1]
```

In [73]:
```
# precision, recall, F1 score, and count of response variable
# of naive bayes classifier

print(classification_report(y_test, clf_y_predictions))
```

```
             precision    recall  f1-score   support

          0       0.89      1.00      0.94       807
          1       0.00      0.00      0.00        98

avg / total       0.80      0.89      0.84       905
```

```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Un
  'precision', 'predicted', average, warn_for)
```

In [74]:
```
# evaluate naive bayes classifier
# when threshold set to default 0.5
# model_metrics(y_known, y_pred):

model_metrics(y_test, clf_y_predictions)
```

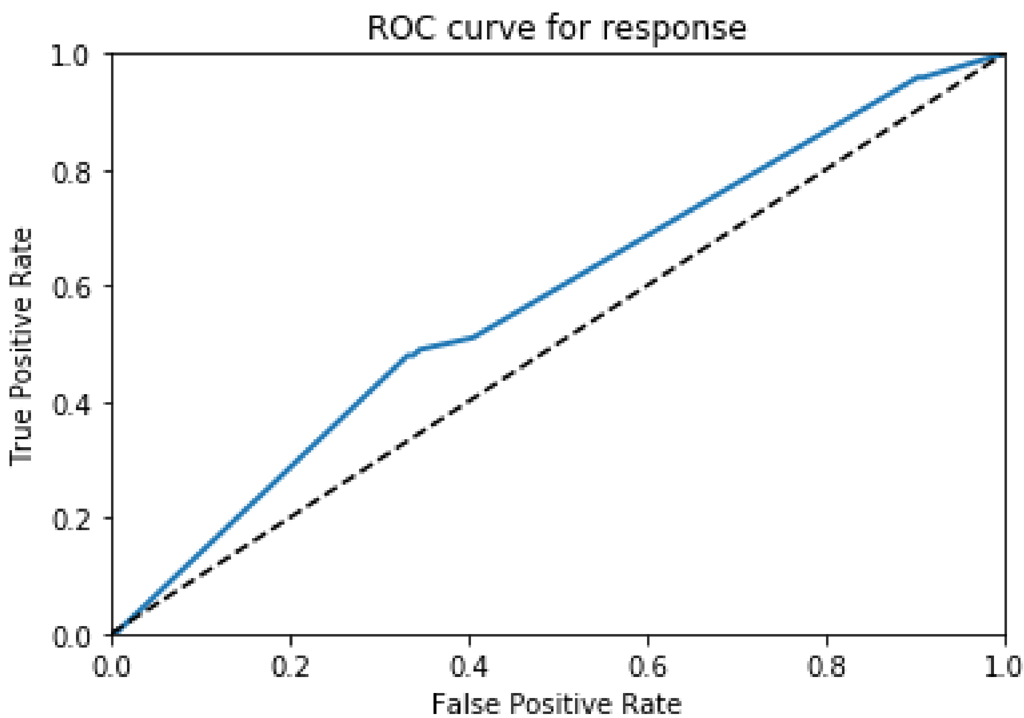
```
The model metrics are:
accuracy: 0.8917127071823204
classification error: 0.10828729281767957
sensitivity: 0.0
specificity: 1.0
false positive rate: 0.0
precision: nan
F1 score: 0.0
confusion matrix:
```

```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:16: RuntimeWarning:
  app.launch_new_instance()
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Un
  'precision', 'predicted', average, warn_for)
```

Out[74]:
```
array([[807,   0],
       [ 98,   0]])
```

In [75]:
```
# first argument is true values,
# second argument is predicted probabilities

# pass y_test and y_pred_prob
# do not use y_pred_class, because it will give
```

```
# incorrect results without generating an error
# roc_curve returns 3 objects fpr, tpr, thresholds
# fpr: false positive rate
# tpr: true positive rate

fpr, tpr, thresholds = roc_curve(y_test, clf_y_pred_prob_class1)
plot_roc_curve(fpr, tpr)
plt.savefig('ROC_bernoulli_plot.pdf')
plt.show()
```




```
In [76]: # evaluate the naive bayes classifier
         # sensitivity and specificity when the threshold is set to 0.5

         print('When the threshold is set to 0.5')
         evaluate_threshold(0.5)
```

```
When the threshold is set to 0.5
Sensitivity: 0.0
Specificity: 1.0
```

```
In [77]: # evaluate the naive bayes classifier
         # sensitivity and specificity when the threshold is set to 0.1
```

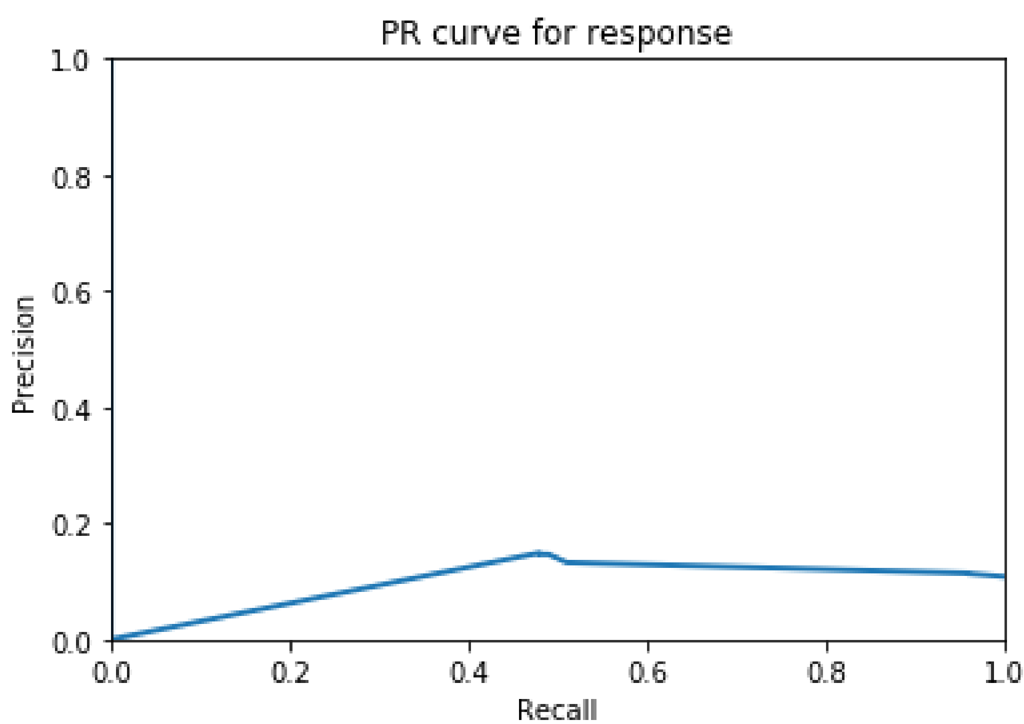
```
print('When the threshold is set to 0.1')
evaluate_threshold(0.1)
```

```
When the threshold is set to 0.1
Sensitivity: 0.4897959183673469
Specificity: 0.654275092936803
```

In [78]:
```
# AUC is the percentage of the ROC plot that is underneath the curve
# first argument is true values, second argument is predicted probabilities
# AUC for naive bayes classifier

roc_auc_y100_class1 = metrics.roc_auc_score(y_test, clf_y_pred_prob_class1)
print('\nThe AUC is:', roc_auc_y100_class1)
```

```
The AUC is: 0.5752977771034065
```

In [79]:
```
# Given the imbalance of 0 to 1 in the response category
# let's see what the Precision-Recall curve looks like
# for the naive bayes classifier

precision, recall, thresholds = precision_recall_curve(y_test, clf_y_pred_prob_class1)
plot_pr_curve(recall, precision)
plt.savefig('PR_bernoulli_plot.pdf')
plt.show()
```



0.10 Cross Validation of Logistic Regression Model, C = 100

```
In [80]: # calculate cross-validated AUC for logistic regression model

         #log_reg100 = LogisticRegression(C=100)
         print('Cross validated AUC for Logistic Regression:')
         cross_log_auc = cross_val_score(log_reg100, X_train, y_train, cv=10, scoring='roc_auc

         display_scores(cross_log_auc)
```

```
Cross validated AUC for Logistic Regression:
Scores: [0.627 0.606 0.555 0.68  0.632 0.592 0.554 0.599 0.577 0.622]
Mean: 0.6045377265827597
Standard deviation: 0.03632462069250767
```

0.11 Cross Validation of Naive Bayes Classification

```
In [81]: # calculate cross-validated AUC for naive Bayes classification

         print('Cross validated AUC for Naive Bayes Classification:')
         cross_clf_auc = cross_val_score(clf, X_train, y_train, cv=10, scoring='roc_auc')

         display_scores(cross_clf_auc)
```

```
Cross validated AUC for Naive Bayes Classification:
Scores: [0.627 0.606 0.547 0.68  0.632 0.592 0.554 0.599 0.577 0.622]
Mean: 0.6036947033269457
Standard deviation: 0.03753998048084224
```

0.12 Interpreting the Logistic Regression Model, C = 100

```
In [82]: log_reg100.coef_

Out[82]: array([[ 0.382, -0.708, -0.695]])

In [83]: model_data_df_X.head()

Out[83]:    default  housing  loan
         0        0        0     0
         1        0        1     1
         2        0        1     0
         3        0        1     1
         4        0        1     0
```

In [84]:
```
#yes = 1, no = 0
#response = has the client subscribed to a term deposit?
#for credit in default = yes, the log of the odds of response increase by 0.382
#for housing loan = yes, the log of the odds of response decrease by -0.708
#for personal loan = yes, the log of the odds of response decrease by -0.695
```

In [85]:
```
clf.coef_
```

Out[85]:
```
array([[-3.855, -0.905, -2.415]])
```

In [86]:
```
math.exp(-3.855)

#for client with credit in default, there is a 2% increase in having a term deposit
```

Out[86]:
```
0.02117360331011653
```

In [87]:
```
math.exp(-0.905)
#for client with housing loan, there is a 40% decrease in having a term deposit
```

Out[87]:
```
0.4045418851030188
```

In [88]:
```
math.exp(-2.415)
#for client with personal loan, there is a 8.9% decrease in having a term deposit
```

Out[88]:
```
0.08936733892175319
```