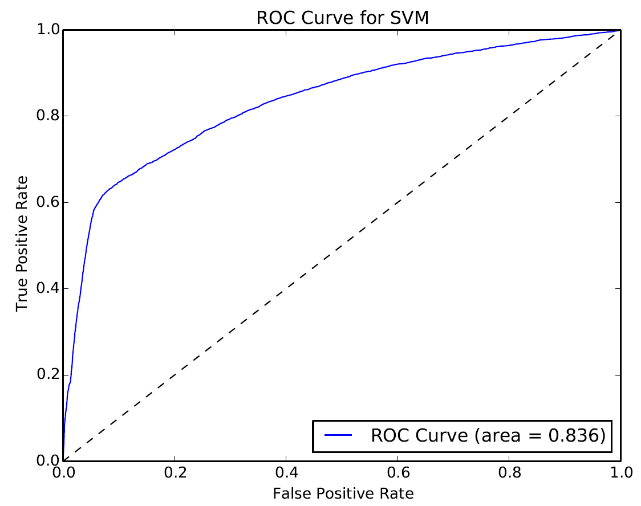
**Introduction**

The purpose of this assignment is to make a recommendation to bank management whether to use a machine learning model to offer loans to specific customers through direct marketing. A Support Vector Machine (SVM) model was developed using 16 variables from the “Using Data Mining for Bank Direct Marketing: An Application of the CRISP-DM Methodology” study. This paper recommends using the model to optimize return on investment (ROI) and highlights the risks and costs associated with its use.

**Results**

The area under the receiver operating characteristic (ROC) curve characterizes the classification performance and the 0.836 index provides a strong lift over a 0.5 random chance. Given that there is a relatively low chance that people will accept the loan offer, model accuracy even at 0.893 is not a viable measure for evaluating its performance.

The confusion matrix is an effective method for articulating the recommendations and understanding the risks of using this model (based on the “test” data).

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Actual Binary Response | |
|  |  | YES | NO |
| Predicted Binary Response | YES | **True Positive**  **19686** | **False Positive**  **260** |
| NO | **False Negative**  **2156** | **True Negative**  **504** |
|  |  | **True Positive Rate = .9013** | **False Positive Rate = .3403** |

The model should be used to send direct mail to targeted customers to optimize conversion and ROI given that over 98% customers who are targeted accept the offer. The false positive represents consumers that were predicted to accept the offer and actually did not. This represents direct marketing costs that are incurred without conversion. Overall this is a very small percentage (1.3%) of consumers that do not convert and the non-conversion costs can be calculated accordingly to set expectations.

The false negative represents the consumers that were predicted to not accept the offer but actually did. This represents lost revenue for approximately 10% of the consumers that would have accepted the offer but were not targeted. This is money left on the table and the model should be continuously improved as well as evaluating alternative models to minimize the false negatives. Another recommendation would be to target a slightly larger customer base utilizing using low-cost digital marketing capabilities such as email to potentially capture these customers.

**Code**

# James Gray - Northwestern University CIS435 - Assignment #3 (August 2, 2014)

# Bank Marketing Study

# original source data from http://archive.ics.uci.edu/ml/datasets/Bank+Marketing

from \_\_future\_\_ import division, print\_function

from future\_builtins import ascii, filter, hex, map, oct, zip

import sklearn as sk

import sklearn.linear\_model as sklm

import numpy as np # efficient processing of numerical arrays

import pandas as pd # pandas for data frame operations

import matplotlib.pyplot as plt # for plotting ROC curve

import sklearn.svm as svm # support vector machine classifier

import sklearn.naive\_bayes as nb # naive bayes classifier

# use the full data set after development is complete with the smaller data set

# bank = pd.read\_csv('bank-full.csv', sep = ';') # start with smaller data set

# ==================================================================================

# data set predictors

# ==================================================================================

# 1 - age: numeric

# 2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','

# housemaid','management','retired','self-employed','services','student',

# 'technician','unemployed','unknown')

# 3 - marital : marital status (categorical: 'divorced','married','single','unknown';

# note: 'divorced' means divorced or widowed)

# 4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','

# illiterate','professional.course','university.degree','unknown')

# 5 - default: has credit in default? (categorical: 'no','yes','unknown')

# 6 - balance: loan balance

# 7 - housing: has housing loan? (categorical: 'no','yes','unknown')

# 8 - loan: has personal loan? (categorical: 'no','yes','unknown')

# 9 - contact: contact communication type (categorical: 'cellular','telephone')

# 10 - day: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')

# 11 - month: last contact month of year (categorical: 'jan', 'feb', 'mar'....)

# 12- duration: last contact duration, in seconds (numeric). Should be removed

# 13 - campaign: number of contacts performed during this campaign and for this client\

# (numeric, includes last contact)

# 14 - pdays: number of days that passed by after the client was last contacted from

# a previous campaign (numeric; 999 means client was not previously contacted)

# 15 - previous: number of contacts performed before this campaign and for this client (numeric)

# 16 - poutcome: outcome of the previous marketing campaign (categorical: 'failure',

# 'nonexistent','success')

# ==================================================================================

# output variable

# 21 - y - has the client subscribed a term deposit? (binary: 'yes','no')

# ==================================================================================

# initial work with the smaller data set

#bank = pd.read\_csv('bank.csv', sep = ';') # start with smaller data set

# full data set

bank = pd.read\_csv('bank-full.csv', sep = ';')

# drop observations with missing data, if any

bank.dropna()

# examine the shape of the DataFrame

print(bank.shape)

# look at the list of column names, note that y is the response

list(bank.columns.values)

# look at the beginning of the DataFrame

print (bank.head())

# ==================================================================================

# Data Preparation - Transform categorical data into dummy/indicator variables

# ==================================================================================

# We will use the Pandas .getdummies function to convert categorical variables

# into dummy/indicator variables

# job, marital, education, default, housing, loan, contact, month, poutcome are

# categorical vars that needs to be converted

for column in ['job', 'marital', 'education', 'default', 'housing', 'loan',

'contact', 'day', 'month', 'poutcome']:

# create a new dummmies DataFrame that holds the new dummy columns

dummies = pd.get\_dummies(bank[column],prefix=column)

# add new dummy columns to original bank DataFrame

bank[dummies.columns] = dummies

# remove the original categorical column in the bank DataFrame

bank = bank.drop(column, axis=1)

print (bank)

# confirm new dummy vars have been added

print (bank.head)

# ==================================================================================

# Data Preparation - Transform "yes" or "no" categorical data into binary

# ==================================================================================

# helper dict function to convert yes and no into binary

convert\_to\_binary = {'no' : 0, 'yes' : 1}

# define response variable of use in the model

y = bank['y'].map(convert\_to\_binary)

# 5 - define binary variable for having credit in default

#default = bank['default'].map(convert\_to\_binary)

# 6 - define binary variable for having a housing loan

#housing = bank['housing'].map(convert\_to\_binary)

# 7- define binary variable for having a personal loan

#loan = bank['loan'].map(convert\_to\_binary)

# =================================================================================

# Data Preparation - Discretize continuous data into bins

# =================================================================================

# 1 - age array

age = bank['age']

# Discretize age using quantiles - returns a Categorical object (array of strings)

# but numpy array if labels=False

age\_to\_bins = pd.qcut(age,4,labels=False)

# Transform binned data into indicator variables (DataFrame)

dummies\_age\_bins = pd.get\_dummies(age\_to\_bins, prefix='age\_q')

# append new Age dummy columns to bank DF

bank[dummies\_age\_bins.columns] = dummies\_age\_bins

# use average yearly balance in euros as explanatory variable

balance = bank['balance']

# Discretize Balance using quantiles

balance\_to\_bins = pd.qcut(balance,4,labels=False)

# Transform binned data into indicator variables (DataFrame)

dummies\_balance\_bins = pd.get\_dummies(balance\_to\_bins, prefix='bal\_q')

# append new balance dummy columns to bank DF

bank[dummies\_balance\_bins.columns] = dummies\_balance\_bins

# number of previous contacts

previous = bank['previous']

# Discretize previous contacts using quantiles

#previous\_to\_bins = pd.qcut(previous,4, labels=False)

# Transform binned data into indicator variables (DataFrame)

#dummies\_previous\_bins = pd.get\_dummies(previous\_to\_bins, prefix='prev\_q')

# ================================================================================

# Data Preparation - Construct final x input array

# ================================================================================

# gather these explanatory variables into a numpy array

# here we use .T to obtain the transpose for the structure we want

#x = np.array([np.array(default), np.array(housing), np.array(loan),

# np.array(balance), np.array(previous)]).T

#x = np.array([np.array(previous)]).T

# Drop columns the numerical colums that have been discretized

drop\_columns = ['age','balance','duration', 'campaign', 'pdays', 'previous','y']

bank = bank.drop(drop\_columns, axis=1)

x = np.array(bank)

# =============================================================================

# LOGISTIC REGRESSION

# =============================================================================

"""

# fit a logistic regression model

# note differences with and without class\_weight settings

# by using class\_weight = 'auto' argument in LogisticRegression

logreg = sklm.LogisticRegression(C=1e5)

my\_model\_fit = logreg.fit(x, y)

# predicted class in training data only

y\_pred = my\_model\_fit.predict(x)

print('Logistic Confusion matrix for training set')

print(sk.metrics.confusion\_matrix(y, y\_pred))

print('Logistic Predictive accuracy in training set:',round(sk.metrics.accuracy\_score(y, y\_pred), 3))

# multi-fold cross-validation with 5 folds

cv\_results = sk.cross\_validation.cross\_val\_score(logreg, x, y, cv=5)

print('Logistic Cross-validation average accuracy:', round(cv\_results.mean(),3))

# compute ROC curve and area under the ROC curve

probs = my\_model\_fit.predict\_proba(x)

false\_positive, true\_positive, thresholds = sk.metrics.roc\_curve(y, probs[:, 1])

roc\_auc = sk.metrics.auc(false\_positive, true\_positive)

print('Logistic Area under the ROC curve:', round(roc\_auc,3))

# Plot ROC curve to IPython shell and to external file

plt.clf()

plt.plot(false\_positive, true\_positive, label='ROC Curve (area = %0.3f)' % roc\_auc)

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve for Logistic Regression')

plt.legend(loc="lower right")

plt.savefig('plot\_rocLR.pdf')

"""

# =============================================================================

# SUPPORT VECTOR MACHINE (SVM)

# =============================================================================

from sklearn import metrics

# split data for training and testing regimen

from sklearn.cross\_validation import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.5, random\_state=9999)

# Instantiate the estimator

svmclf = svm.SVC(probability=True)

# Fit the estimator to the training data

svm\_train\_model\_fit = svmclf.fit(x\_train, y\_train)

# training set predictions from the model fit to the training set

y\_svmpred = svm\_train\_model\_fit.predict (x\_test)

print('SVM Confusion Matrix for training set')

# Compare actual y and prediction in a confusion matrix using test data

print(sk.metrics.confusion\_matrix(y\_test, y\_svmpred))

# Show accuracy rate

print('SVM Predictive accuracy in training set:',

round(sk.metrics.accuracy\_score(y\_test, y\_svmpred), 3))

# accuracy = correct labels / total samples

print ("accuracy: ", metrics.accuracy\_score (y\_test, y\_svmpred))

# precision = true positives / (true positives + false positives)

# This represents the % of labeled class that actually the class

print ("precision: ", metrics.precision\_score (y\_test, y\_svmpred))

# recall = true positives / (true positives + false negatives)

# This represents the % of the actual class we are pulling out of the sample

print ("recall: ", metrics.recall\_score (y\_test, y\_svmpred))

# f1 = precision \* recall / (precision + recall)

print ("f1 score: ", metrics.f1\_score (y\_test, y\_svmpred))

# Print Classification report

#print (metrics.classification\_report (y\_test, y\_svmpred,

# target\_names = ['reject', 'accept'] )

# SVM multi-fold cross-validation with 5 folds

#svm\_cv\_results = sk.cross\_validation.cross\_val\_score(svmclf, x, y, cv=5)

#print('SVM Cross-validation average accuracy:', round(svm\_cv\_results.mean(),3))

# run full data set

full\_model\_fit = svmclf.fit(x, y)

# compute ROC curve and area under the ROC curve

svm\_probs = full\_model\_fit.predict\_proba(x)

false\_positive, true\_positive, thresholds = sk.metrics.roc\_curve(y, svm\_probs[:, 1])

svm\_roc\_auc = sk.metrics.auc(false\_positive, true\_positive)

print('SVM Area under the ROC curve:', round(svm\_roc\_auc,3))

# Plot ROC curve to IPython shell and to external file

plt.clf()

plt.plot(false\_positive, true\_positive, label='ROC Curve (area = %0.3f)' % svm\_roc\_auc)

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve for SVM')

plt.legend(loc="lower right")

plt.savefig('plot\_rocSVM.pdf')