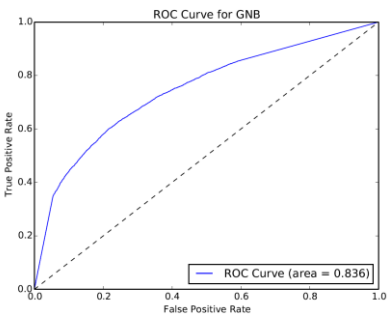
James Gray
graymatter@u.northwestern.edu

Introduction

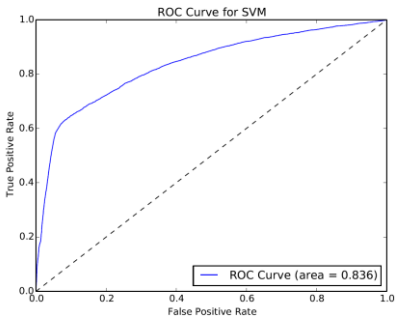
The purpose of this assignment is to make a recommendation to bank management whether to use a machine learning model to offer loans to specific customers through direct marketing. Three machine learning algorithms were evaluated (Classification Tree, Support Vector Machine and Gaussian Naïve Bayes) 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 Gaussian Naïve Bayes model to optimize return on investment (ROI) for direct marketing campaigns.

Results

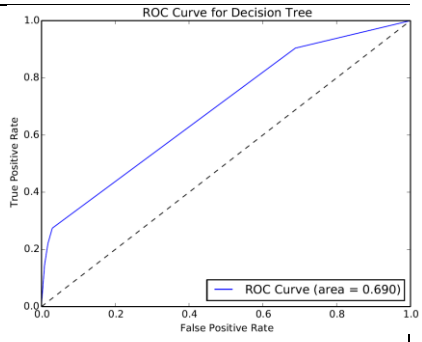
The three models were evaluated using the area under the receiver operating characteristic (ROC) curve and a confusion matrix.



Gaussian NB ROC AUC = 0.836



SVM ROC AUC = 0.836



Decision Tree ROC AUC = 0.690

		Actual Binary Response Gaussian Naïve Bayes		**Actual Binary Response Support Vector Machines**		**Actual Binary Response Classification Tree**	
		YES	NO	YES	NO	YES	NO
Predicted Binary Response	YES	**True Positive 17098**	**False Positive 2848**	**True Positive 19686**	**False Positive 260**	**True Positive 19791**	**False Positive 155**
	NO	**False Negative 1344**	**True Negative 1316**	**False Negative 2156**	**True Negative 504**	**False Negative 2318**	**True Negative 342**
		True Positive Rate = .9271	**False Positive Rate = .68395**	**True Positive Rate = .9013**	**False Positive Rate = .3403**	**True Positive Rate = .8951**	**False Positive Rate = .3119**

Although the Gaussian Naïve Bayes and Support Vector Classification models have identical area under the ROC curve performance, it is recommended to use the Gaussian Naïve Bayes model that has a higher true positive rate. This recommendation assumes that the slightly higher conversion rate will offset the higher false positive rate which represents direct mail that is sent to consumers that do not accept the offer. The false negative represents the consumers that were predicted to not accept the offer but actually did. This represents lost revenue for approximately 8% 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. Using the Gaussian Naïve Bayes model will deliver an estimated 812 additional consumers that accept the offer over the Support Vector Classification while 2588 more people are sent direct mail that do not accept the offer. The exact incremental profit over the Support Vector can be calculated by subtracting the direct mail costs from the revenue of the 812 additional consumers.

Code

```
James Gray - Northwestern University CIS435 - Assignment #4 (August 10 2014)
# Bank Marketing Study
# This code runs multiple ML algorithms to determine the optimal model for
# direct marketing programs

# 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 - Professor Miller
```

```
# ============================================================================

# 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')dt

"""

# ============================================================================
# SUPPORT VECTOR MACHINE (SVM) CLASSIFICATION
# ============================================================================

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')

# ================================================================================
# GAUSSIAN NAIVE BAYES CLASSSIFIER
# ================================================================================

# Instantiate the estimator
gnb = nb.GaussianNB ()

# Fit the estimator to the training data
gnb_train_model = gnb.fit(x_train, y_train)
```

```
# predictions from the model fit to the training set
y_gnb_pred = gnb_train_model.predict (x_test)

print('Gaussian Naive Bayes 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_gnb_pred))

#def plot_confusion_matrix(y_gnb_pred, y_test):
# plt.imshow(metrics.confusion_matrix(y, y_gnb_pred),
# cmap=plt.cm.binary, interpolation='nearest')
# plt.colorbar()
# plt.title ("Gaussian NB Confusion Matrix")
# plt.xlabel('true value')
# plt.ylabel('predicted value')
# plt.savefig('plot_GNB_Confusion.pdf')
print ("classification accuracy:", metrics.accuracy_score(y_test, y_gnb_pred))
#plot_confusion_matrix(y, y_gnb_pred)

# accuracy = correct labels / total samples
print ("accuracy: ", metrics.accuracy_score (y_test, y_gnb_pred))

# 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_gnb_pred))

# 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_gnb_pred))

# f1 = precision * recall / (precision + recall)
print ("f1 score: ", metrics.f1_score (y_test, y_gnb_pred))

# run full data set
gnb_full_model_fit = gnb.fit(x, y)

# compute ROC curve and area under the ROC curve
gnb_probs = gnb_full_model_fit.predict_proba(x)
false_positive, true_positive, thresholds = sk.metrics.roc_curve(y, gnb_probs[:, 1])
gnb_roc_auc = sk.metrics.auc(false_positive, true_positive)
print('Gaussian Naive Bayes Area under the ROC curve:', round(gnb_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 GNB')
plt.legend(loc="lower right")
plt.savefig('plot_rocGNB.pdf')

# ================================================================================
# DECISION TREE CLASSIFICATION
# ================================================================================
```

```
from sklearn import tree

# instantiate the decision tree estimator
dtclf = tree.DecisionTreeClassifier (min_samples_split = 20, min_samples_leaf = 10,
max_depth = 4, random_state = 9999)

# fit the decision tree using the training data
my_dt_model = dtclf.fit(x_train, y_train)

# predictions from the model fit to the training set
y_dtpred = my_dt_model.predict (x_test)

print('Decision Tree 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_dtpred))

# Show accuracy rate
print('Decision Tree Predictive accuracy in training set:',
round(sk.metrics.accuracy_score(y_test, y_dtpred), 3))
# accuracy = correct labels / total samples
print ("accuracy: ", metrics.accuracy_score (y_test, y_dtpred))

# 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_dtpred))

# 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_dtpred))

# f1 = precision * recall / (precision + recall)
print ("f1 score: ", metrics.f1_score (y_test, y_dtpred))

# run full data set
dt_full_model_fit = dtclf.fit(x, y)

# lets see what the fitted tree looks like
from sklearn.externals.six import StringIO
with open('tree.dot', 'w') as f:
f = tree.export_graphviz(dt_full_model_fit, out_file=f)

# compute ROC curve and area under the ROC curve
dt_probs = dt_full_model_fit.predict_proba(x)
false_positive, true_positive, thresholds = sk.metrics.roc_curve(y, dt_probs[:, 1])
dt_roc_auc = sk.metrics.auc(false_positive, true_positive)
print('Decision Tree Area under the ROC curve:', round(dt_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)' % dt_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 Decision Tree')
plt.legend(loc="lower right")
plt.savefig('plot_rocDT.pdf'
```