Also refer 🡪 <https://www.kaggle.com/sneharth03/bank-marketing>

Bank Marketing Analysis for Term Deposit

1. Background

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) was subscribed or not. Data set has 20 predictor varaibles (features) and around 41K rows. Top 6 rows of the dataset is shown below.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import sklearn.linear\_model as lm

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split, KFold

from sklearn.preprocessing import StandardScaler, label\_binarize

from sklearn.metrics import accuracy\_score,confusion\_matrix,roc\_curve, auc, f1\_score, precision\_score, recall\_score

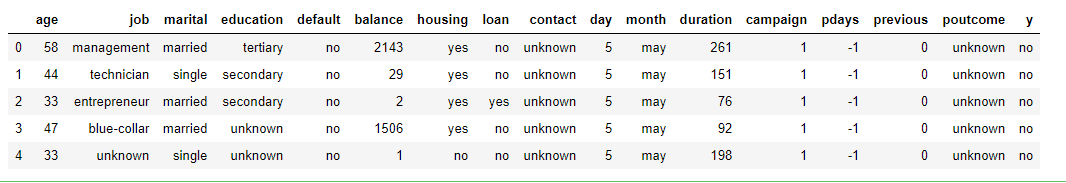
from sklearn.svm import SVC

from imblearn.over\_sampling import RandomOverSampler, SMOTE

from imblearn.under\_sampling import RandomUnderSampler

data = pd.read\_csv("C:/Users/Adinath/Desktop/Data\_Science/Assignments/Logistic Regression/bank-full.csv",sep=';')

data.head()



Customers who received phone calls may not be unique and a same customer might have received multiple calls. The last column in the dataset 'y' is our response. This report tries to get the best model to predict the subscription of term deposit. This report is divided into five sections. Graphical analysis is explored in Section 2, various methodologies to treat imbalance in data is discussed in Section 3, different models are fit in Section 4 and conclusive remarks are drawn in Section 5.

## 2. Graphical Analysis

data1 = data[data['y'] == 'yes']

data2 = data[data['y'] == 'no']

In the given dataset, there are 10 categorical variables and 10 continous variables. All categorical variables are visualized using stacked barplots (color coded using OSU colors :) for 'yes' and 'no' outcome of term deposit subscription. Below are the 10 barplots.

fig, ax = plt.subplots(2, 2, figsize=(12,10))

b1 = ax[0, 0].bar(data1['day\_of\_week'].unique(),height = data1['day\_of\_week'].value\_counts(),color='#000000')

b2 = ax[0, 0].bar(data2['day\_of\_week'].unique(),height = data2['day\_of\_week'].value\_counts(),bottom = data1['day\_of\_week'].value\_counts(),color = '#DC4405')

ax[0, 0].title.set\_text('Day of week')

*#ax[0, 0].legend((b1[0], b2[0]), ('Yes', 'No'))*

ax[0, 1].bar(data1['month'].unique(),height = data1['month'].value\_counts(),color='#000000')

ax[0, 1].bar(data2['month'].unique(),height = data2['month'].value\_counts(),bottom = data1['month'].value\_counts(),color = '#DC4405')

ax[0, 1].title.set\_text('Month')

ax[1, 0].bar(data1['job'].unique(),height = data1['job'].value\_counts(),color='#000000')

ax[1, 0].bar(data1['job'].unique(),height = data2['job'].value\_counts()[data1['job'].value\_counts().index],bottom = data1['job'].value\_counts(),color = '#DC4405')

ax[1, 0].title.set\_text('Type of Job')

ax[1, 0].tick\_params(axis='x',rotation=90)

ax[1, 1].bar(data1['education'].unique(),height = data1['education'].value\_counts(),color='#000000') *#row=0, col=1*

ax[1, 1].bar(data1['education'].unique(),height = data2['education'].value\_counts()[data1['education'].value\_counts().index],bottom = data1['education'].value\_counts(),color = '#DC4405')

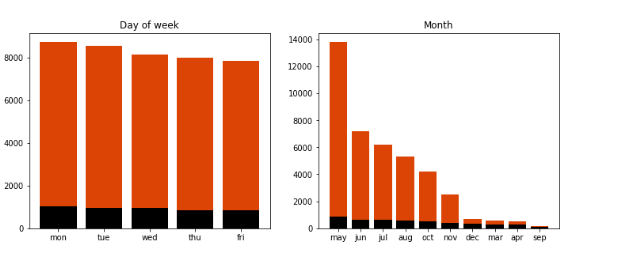
ax[1, 1].title.set\_text('Education')

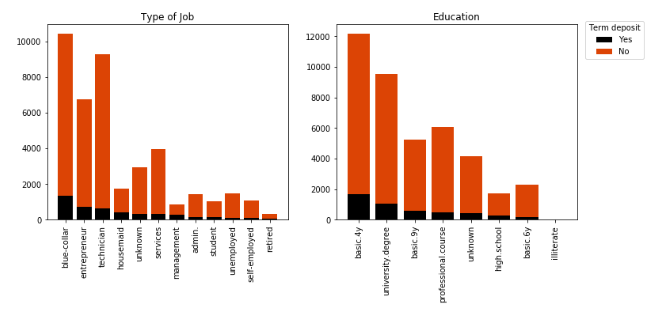
ax[1, 1].tick\_params(axis='x',rotation=90)

*#ax[0, 1].xticks(rotation=90)*

plt.figlegend((b1[0], b2[0]), ('Yes', 'No'),loc="right",title = "Term deposit")

plt.show()





fig, ax = plt.subplots(2, 3, figsize=(15,10))

b1 = ax[0, 0].bar(data1['marital'].unique(),height = data1['marital'].value\_counts(),color='#000000')

b2 = ax[0, 0].bar(data1['marital'].unique(),height = data2['marital'].value\_counts()[data1['marital'].value\_counts().index],bottom = data1['marital'].value\_counts(),color = '#DC4405')

ax[0, 0].title.set\_text('Marital Status')

*#ax[0, 0].legend((b1[0], b2[0]), ('Yes', 'No'))*

ax[0, 1].bar(data1['housing'].unique(),height = data1['housing'].value\_counts(),color='#000000')

ax[0, 1].bar(data1['housing'].unique(),height = data2['housing'].value\_counts()[data1['housing'].value\_counts().index],bottom = data1['housing'].value\_counts(),color = '#DC4405')

ax[0, 1].title.set\_text('Has housing loan')

ax[0, 2].bar(data1['loan'].unique(),height = data1['loan'].value\_counts(),color='#000000')

ax[0, 2].bar(data1['loan'].unique(),height = data2['loan'].value\_counts()[data1['loan'].value\_counts().index],bottom = data1['loan'].value\_counts(),color = '#DC4405')

ax[0, 2].title.set\_text('Has personal loan')

ax[1, 0].bar(data1['contact'].unique(),height = data1['contact'].value\_counts(),color='#000000')

ax[1, 0].bar(data1['contact'].unique(),height = data2['contact'].value\_counts()[data1['contact'].value\_counts().index],bottom = data1['contact'].value\_counts(),color = '#DC4405')

ax[1, 0].title.set\_text('Type of Contact')

ax[1, 1].bar(data1['default'].unique(),height = data1['default'].value\_counts(),color='#000000')

ax[1, 1].bar(data1['default'].unique(),height = data2['default'].value\_counts()[data1['default'].value\_counts().index],bottom = data1['default'].value\_counts(),color = '#DC4405')

ax[1, 1].title.set\_text('Has credit in default')

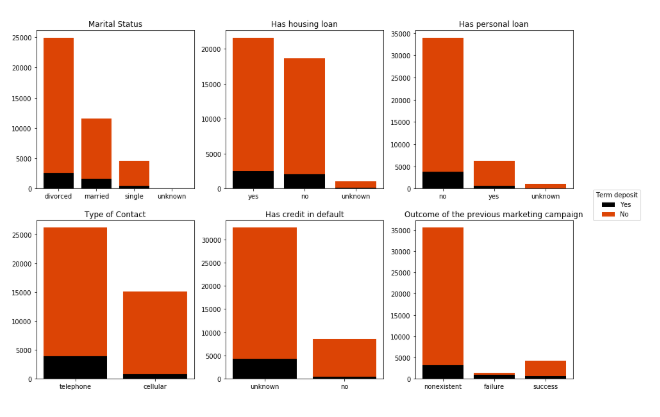
ax[1, 2].bar(data1['poutcome'].unique(),height = data1['poutcome'].value\_counts(),color='#000000')

ax[1, 2].bar(data1['poutcome'].unique(),height = data2['poutcome'].value\_counts()[data1['poutcome'].value\_counts().index],bottom = data1['poutcome'].value\_counts(),color = '#DC4405')

ax[1, 2].title.set\_text('Outcome of the previous marketing campaign')

plt.figlegend((b1[0], b2[0]), ('Yes', 'No'),loc="right",title = "Term deposit")

plt.show()



All the categories in each categorical variable has disproportinate 'yes' compared to 'no' for term deposit subscription. Only noteworthy mention is that in 'Outcome of the previous marketing campaign' variable, failure category has more 'yes' compared to 'no'. This means, if the previous campaign call failed to get the customers to subscribe, repeated calls got customers to subscribe to term deposit. For 10 continous variables, unstacked histograms are plotted below.

fig, ax = plt.subplots(2, 2, figsize=(12,10))

ax[0, 0].hist(data2['age'],color = '#DC4405',alpha=0.7,bins=20, edgecolor='white')

ax[0, 0].hist(data1['age'],color='#000000',alpha=0.5,bins=20, edgecolor='white')

ax[0, 0].title.set\_text('Age')

ax[0, 1].hist(data2['duration'],color = '#DC4405',alpha=0.7, edgecolor='white')

ax[0, 1].hist(data1['duration'],color='#000000',alpha=0.5, edgecolor='white')

ax[0, 1].title.set\_text('Contact duration')

ax[1, 0].hist(data2['campaign'],color = '#DC4405',alpha=0.7, edgecolor='white')

ax[1, 0].hist(data1['campaign'],color='#000000',alpha=0.5, edgecolor='white')

ax[1, 0].title.set\_text('Number of contacts performed')

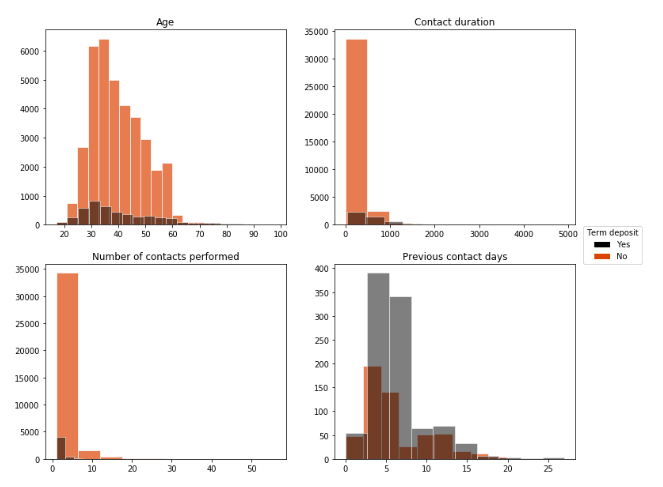
ax[1, 1].hist(data2[data2['pdays'] != 999]['pdays'],color = '#DC4405',alpha=0.7, edgecolor='white')

ax[1, 1].hist(data1[data1['pdays'] != 999]['pdays'],color='#000000',alpha=0.5, edgecolor='white')

ax[1, 1].title.set\_text('Previous contact days')

plt.figlegend((b1[0], b2[0]), ('Yes', 'No'),loc="right",title = "Term deposit")

plt.show()



fig, ax = plt.subplots(2, 3, figsize=(15,10))

ax[0, 0].hist(data2['previous'],color = '#DC4405',alpha=0.7, edgecolor='white')

ax[0, 0].hist(data1['previous'],color='#000000',alpha=0.5, edgecolor='white')

ax[0, 0].title.set\_text('Number of contacts performed previously')

ax[0, 1].hist(data2['emp.var.rate'],color = '#DC4405',alpha=0.7, edgecolor='white')

ax[0, 1].hist(data1['emp.var.rate'],color='#000000',alpha=0.5, edgecolor='white')

ax[0, 1].title.set\_text('Employment variation rate')

ax[0, 2].hist(data2['cons.price.idx'],color = '#DC4405',alpha=0.7, edgecolor='white')

ax[0, 2].hist(data1['cons.price.idx'],color='#000000',alpha=0.5, edgecolor='white')

ax[0, 2].title.set\_text('Consumer price index')

ax[1, 0].hist(data2['cons.conf.idx'],color = '#DC4405',alpha=0.7, edgecolor='white')

ax[1, 0].hist(data1['cons.conf.idx'],color='#000000',alpha=0.5, edgecolor='white')

ax[1, 0].title.set\_text('Consumer confidence index')

ax[1, 1].hist(data2['euribor3m'],color = '#DC4405',alpha=0.7, edgecolor='white')

ax[1, 1].hist(data1['euribor3m'],color='#000000',alpha=0.5, edgecolor='white')

ax[1, 1].title.set\_text('Euribor 3 month rate')

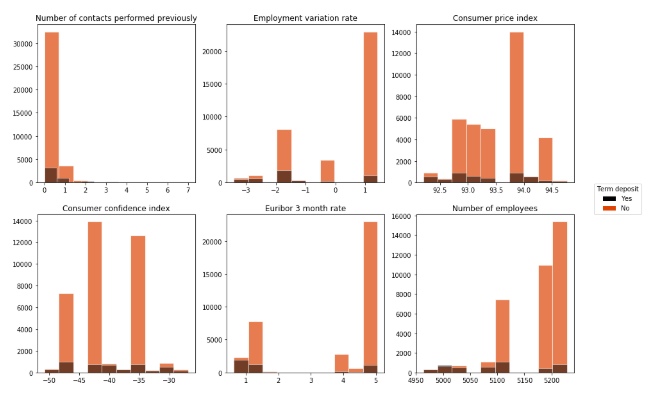
ax[1, 2].hist(data2['nr.employed'],color = '#DC4405',alpha=0.7, edgecolor='white')

ax[1, 2].hist(data1['nr.employed'],color='#000000',alpha=0.5, edgecolor='white')

ax[1, 2].title.set\_text('Number of employees')

plt.figlegend((b1[0], b2[0]), ('Yes', 'No'),loc="right",title = "Term deposit")

plt.show()



All the histograms show similar behavior (proportion of 'no' is higher than 'yes') except 'Previous contact days'. Proportion of 'yes' is higher compared to 'no' (only if contacted more than once).

predictors = data.iloc[:,0:20]

predictors = predictors.drop(['pdays'],axis=1)

y = data.iloc[:,20]

X = pd.get\_dummies(predictors)

## 3. Treating Imbalanced Data

Given data set is highly imbalanced, i.e. number of data belonging to 'no' category is way higher than 'yes' category.

y.value\_counts()

no 36548

yes 4640

Name: y, dtype: int64

This imbalance has to treated so as to make sure that there is no bias in modeling. Imbalance is generally treated in three ways.

### 3.1 Random Undersampling

In this method, the majority category, in this case 'no' category is randomly sampled to match the size of the minority 'yes' category. Remaining data of majority category is discarded.

rus = RandomUnderSampler(random\_state=0)

X\_Usampled, y\_Usampled = rus.fit\_resample(X, y)

pd.Series(y\_Usampled).value\_counts()

yes 4640

no 4640

dtype: int64

### 3.2 Random Oversampling

In this method, the minority category 'no' is randomly sampled with replacement to match the size of the majority 'no' category. Minority category entries will be repeated many times.

ros = RandomOverSampler(random\_state=0)

X\_Osampled, y\_Osampled = ros.fit\_resample(X, y)

pd.Series(y\_Osampled).value\_counts()

yes 36548

no 36548

dtype: int64

3.3 SMOTE - Synthetic Minority Oversampling Technique

This is an oversampling technique in which instead of randomly repeating minority 'yes' category, new entires are sythetically created maintaining the convexity of minority entry space. Minority category will again match the majority category samples.

sm = SMOTE(random\_state=0)

X\_SMOTE, y\_SMOTE = sm.fit\_resample(X, y)

pd.Series(y\_SMOTE).value\_counts()

yes 36548

no 36548

dtype: int64

## 4. Model Fitting

Simple linear classifier is fitted on the imbalanced data. Model is highly biased towards 'no' class. Accuracy is high for this model which is expected for imbalanced data.

Simple linear classifier is fitted on the imbalanced data. Model is highly biased towards 'no' class. Accuracy is high for this model which is expected for imbalanced data.

In [13]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

sc = StandardScaler()

sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train)

X\_test\_std = sc.transform(X\_test)

perp\_model = lm.Perceptron().fit(X\_train\_std,y\_train)

y\_pred = perp\_model.predict(X\_test\_std)

print("Accuracy: ",round(accuracy\_score(y\_test, y\_pred),2))

Accuracy: 0.86

Confusion matrix gives a better idea. Categorization of minority 'yes' class is far from perfect which is indicated by poor precision score and recall score.

mat = confusion\_matrix(y\_test,y\_pred,labels=['no','yes'])

print(mat)

y\_test = label\_binarize(y\_test,classes=['no','yes'])

y\_pred = label\_binarize(y\_pred,classes=['no','yes'])

print("Precision: ",round(precision\_score(y\_test,y\_pred),2),"Recall: ",round(recall\_score(y\_test,y\_pred),2))

[[10006 993]

[ 723 635]]

Precision: 0.39 Recall: 0.47

When percptron model is fit for SMOTE data, bias is shifted towards 'yes' class. This results in better recall score and worse precision score than before.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

sm = SMOTE(random\_state=0)

X\_SMOTE, y\_SMOTE = sm.fit\_resample(X\_train, y\_train)

sc = StandardScaler()

sc.fit(X\_SMOTE)

X\_train\_std = sc.transform(X\_SMOTE)

X\_test\_std = sc.transform(X\_test)

perp\_model = lm.Perceptron().fit(X\_train\_std,y\_SMOTE)

y\_pred = perp\_model.predict(X\_test\_std)

print("Accuracy: ",round(accuracy\_score(y\_test, y\_pred),2))

mat = confusion\_matrix(y\_test,y\_pred)

print("Confusion Matrix: **\n**",mat)

y\_test = label\_binarize(y\_test,classes=['no','yes'])

y\_pred = label\_binarize(y\_pred,classes=['no','yes'])

print("Precision: ",round(precision\_score(y\_test,y\_pred),2),"Recall: ",round(recall\_score(y\_test,y\_pred),2))

Accuracy: 0.79

Confusion Matrix:

[[8728 2242]

[ 315 1072]]

Precision: 0.32 Recall: 0.77

### 4.2 Decision Tree

Decision tree is another simple yet powerful classification tool. But this model faces the same problem of bad precision score and bad recall score for imbalanced data.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

tree = DecisionTreeClassifier(criterion="entropy", max\_depth=7)

model = tree.fit(X\_train,y\_train)

y\_pred = model.predict(X\_test)

y\_test = label\_binarize(y\_test,classes=['no','yes'])

y\_pred = label\_binarize(y\_pred,classes=['no','yes'])

print("Precision: ",round(precision\_score(y\_test,y\_pred),2),"Recall: ",round(recall\_score(y\_test,y\_pred),2))

Precision: 0.66 Recall: 0.51

When decision tree is fit on SMOTE data, recall score improves but precision score deteriorates.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

tree = DecisionTreeClassifier(criterion="entropy", max\_depth=7)

X\_SMOTE, y\_SMOTE = sm.fit\_resample(X\_train, y\_train)

model = tree.fit(X\_SMOTE,y\_SMOTE)

y\_pred = model.predict(X\_test)

y\_test = label\_binarize(y\_test,classes=['no','yes'])

y\_pred = label\_binarize(y\_pred,classes=['no','yes'])

print("Precision: ",round(precision\_score(y\_test,y\_pred),2),"Recall: ",round(recall\_score(y\_test,y\_pred),2))

Precision: 0.47 Recall: 0.77

### 4.3 Random Forest

Random forest is an ensemble technique which reduces the variance in the classification technique. But this classification worsens the bias that is already in the data. Random forest performs badly in terms of precision and recall when applied on imbalanced data.

chnique. But this classification worsens the bias that is already in the data. Random forest performs badly in terms of precision and recall when applied on imbalanced data.

In [18]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

forest = RandomForestClassifier(n\_estimators= 1000,criterion="gini", max\_depth=5,min\_samples\_split = 0.4,min\_samples\_leaf=1, class\_weight="balanced")

model = forest.fit(X\_train,y\_train)

y\_pred = model.predict(X\_test)

pd.Series(y\_pred).value\_counts()

y\_test = label\_binarize(y\_test,classes=['no','yes'])

y\_pred = label\_binarize(y\_pred,classes=['no','yes'])

print("Precision: ",round(precision\_score(y\_test,y\_pred),2),"Recall: ",round(recall\_score(y\_test,y\_pred),2))

Precision: 0.3 Recall: 0.73

Random forest classification hardly performs better when applied on SMOTE data in terms of precision and recall.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

forest = RandomForestClassifier(n\_estimators= 1000,criterion="gini", max\_depth=5,min\_samples\_split = 0.4,min\_samples\_leaf=1, class\_weight="balanced")

X\_SMOTE, y\_SMOTE = sm.fit\_resample(X\_train, y\_train)

model = forest.fit(X\_SMOTE,y\_SMOTE)

y\_pred = model.predict(X\_test)

pd.Series(y\_pred).value\_counts()

y\_test = label\_binarize(y\_test,classes=['no','yes'])

y\_pred = label\_binarize(y\_pred,classes=['no','yes'])

print("Precision: ",round(precision\_score(y\_test,y\_pred),2),"Recall: ",round(recall\_score(y\_test,y\_pred),2))

Precision: 0.32 Recall: 0.68

### 4.4 Logistic Regression

Logisitic regression was fit on imbalanced, random undersampled, random oversampled and SMOTE data. Last three models (logistic regression on treated data) performs fairly better than previous models especially in terms of recall, but still there is room for improvement on precision.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

model = lm.LogisticRegression(random\_state=0, solver='lbfgs',multi\_class='auto',max\_iter=1000).fit(X\_train,y\_train)

y\_pred = model.predict\_proba(X\_test)

y\_pred = y\_pred[:,1]

y\_test = label\_binarize(y\_test,classes=['no','yes'])

fpr\_imb, tpr\_imb, \_ = roc\_curve(y\_test, y\_pred)

roc\_auc\_imb = auc(fpr\_imb, tpr\_imb)

y\_pred = model.predict(X\_test)

y\_pred = label\_binarize(y\_pred,classes=['no','yes'])

print("Imbalanced -")

print("Precision: ",round(precision\_score(y\_test,y\_pred),2),"Recall: ",round(recall\_score(y\_test,y\_pred),2))

*# Undersampled*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

rus = RandomUnderSampler(random\_state=0)

X\_Usampled, y\_Usampled = rus.fit\_resample(X\_train, y\_train)

model = lm.LogisticRegression(random\_state=0, solver='lbfgs',multi\_class='auto',max\_iter=5000).fit(X\_Usampled,y\_Usampled)

y\_pred = model.predict\_proba(X\_test)

y\_pred = y\_pred[:,1]

y\_test = label\_binarize(y\_test,classes=['no','yes'])

fpr\_us, tpr\_us, \_ = roc\_curve(y\_test, y\_pred)

roc\_auc\_us = auc(fpr\_us, tpr\_us)

y\_pred = model.predict(X\_test)

y\_pred = label\_binarize(y\_pred,classes=['no','yes'])

print("Random undersampled -")

print("Precision: ",round(precision\_score(y\_test,y\_pred),2),"Recall: ",round(recall\_score(y\_test,y\_pred),2))

*# Oversampled*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

ros = RandomOverSampler(random\_state=0)

X\_Osampled, y\_Osampled = ros.fit\_resample(X\_train, y\_train)

model = lm.LogisticRegression(random\_state=0, solver='lbfgs',multi\_class='auto',max\_iter=5000).fit(X\_Osampled, y\_Osampled)

y\_pred = model.predict\_proba(X\_test)

y\_pred = y\_pred[:,1]

y\_test = label\_binarize(y\_test,classes=['no','yes'])

fpr\_os, tpr\_os, \_ = roc\_curve(y\_test, y\_pred)

roc\_auc\_os = auc(fpr\_os, tpr\_os)

y\_pred = model.predict(X\_test)

y\_pred = label\_binarize(y\_pred,classes=['no','yes'])

print("Random oversampled -")

print("Precision: ",round(precision\_score(y\_test,y\_pred),2),"Recall: ",round(recall\_score(y\_test,y\_pred),2))

*# SMOTE*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

sm = SMOTE(random\_state=0)

X\_SMOTE, y\_SMOTE = sm.fit\_resample(X\_train, y\_train)

model = lm.LogisticRegression(random\_state=0, solver='lbfgs',multi\_class='auto',max\_iter=5000).fit(X\_SMOTE,y\_SMOTE)

y\_pred = model.predict\_proba(X\_test)

y\_pred = y\_pred[:,1]

y\_test = label\_binarize(y\_test,classes=['no','yes'])

fpr\_smote, tpr\_smote, \_ = roc\_curve(y\_test, y\_pred)

roc\_auc\_smote = auc(fpr\_smote, tpr\_smote)

y\_pred = model.predict(X\_test)

y\_pred = label\_binarize(y\_pred,classes=['no','yes'])

print("SMOTE -")

print("Precision: ",round(precision\_score(y\_test,y\_pred),2),"Recall: ",round(recall\_score(y\_test,y\_pred),2))

Imbalanced -

Precision: 0.69 Recall: 0.39

Random undersampled -

Precision: 0.44 Recall: 0.88

Random oversampled -

Precision: 0.45 Recall: 0.88

SMOTE -

Precision: 0.44 Recall: 0.89

ROC curve depicts the variation of True Positive Rate to False Positive Rate. Area under ROC curve is slightly better for over sampled data compared to imbalanced and undersampled data.

plt.figure()

lw = 2

plt.plot(fpr\_imb, tpr\_imb,

label='Imbalanced data ROC curve (area = **{0:0.4f}**)'

''.format(roc\_auc\_imb),

color='deeppink', linestyle=':', linewidth=2)

plt.plot(fpr\_us, tpr\_us,

label='Undersampled data ROC curve (area = **{0:0.4f}**)'

''.format(roc\_auc\_us),

color='blue', linestyle='--', linewidth=2)

plt.plot(fpr\_os, tpr\_os,

label='Random Oversampled data ROC curve (area = **{0:0.4f}**)'

''.format(roc\_auc\_os),

color='darkred', linestyle='--', linewidth=2)

plt.plot(fpr\_smote, tpr\_smote,

label='SMOTE data ROC curve (area = **{0:0.4f}**)'

''.format(roc\_auc\_smote),

color='darkgreen', linestyle='--', linewidth=2)

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.00])

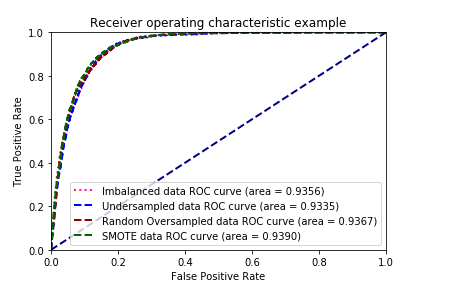
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic example')

plt.legend(loc="lower right")

plt.show()



### 4.5 SVM

Support Vector Machine is employed for SMOTE data using two kernels: linear and guassian. Gaussian kernel performs the best in terms of both precision and recall.

sm = SMOTE(random\_state=0)

X\_SMOTE, y\_SMOTE = sm.fit\_resample(X, y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_SMOTE, y\_SMOTE, test\_size=0.3)

svm = SVC(kernel='linear')

model = svm.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

y\_test = label\_binarize(y\_test,classes=['no','yes'])

y\_pred = label\_binarize(y\_pred,classes=['no','yes'])

print("Linear kernel- ","Precision: ",round(precision\_score(y\_test,y\_pred),2),"Recall: ",round(recall\_score(y\_test,y\_pred),2))

fpr\_linear, tpr\_linear, \_ = roc\_curve(y\_test, y\_pred)

roc\_auc\_linear = auc(fpr\_linear, tpr\_linear)

sm = SMOTE(random\_state=0)

X\_SMOTE, y\_SMOTE = sm.fit\_resample(X, y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_SMOTE, y\_SMOTE, test\_size=0.3)

svm = SVC(kernel='rbf')

model = svm.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

y\_test = label\_binarize(y\_test,classes=['no','yes'])

y\_pred = label\_binarize(y\_pred,classes=['no','yes'])

print("Guassian kernel- ","Precision: ",round(precision\_score(y\_test,y\_pred),2),"Recall: ",round(recall\_score(y\_test,y\_pred),2))

fpr\_rbf, tpr\_rbf, \_ = roc\_curve(y\_test, y\_pred)

roc\_auc\_rbf = auc(fpr\_rbf, tpr\_rbf)

Linear kernel- Precision: 0.9 Recall: 0.73

/opt/conda/lib/python3.6/site-packages/sklearn/svm/base.py:193: FutureWarning: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.

"avoid this warning.", FutureWarning)

Guassian kernel- Precision: 0.89 Recall: 0.98

ROC curves for two different kernels.

plt.figure()

lw = 2

plt.plot(fpr\_linear, tpr\_linear,

label='Linear Kernel ROC curve (area = **{0:0.4f}**)'

''.format(roc\_auc\_linear),

color='darkred', linestyle='--', linewidth=2)

plt.plot(fpr\_rbf, tpr\_rbf,

label='Gaussian Kernel ROC curve (area = **{0:0.4f}**)'

''.format(roc\_auc\_rbf),

color='darkgreen', linestyle='--', linewidth=2)

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.00])

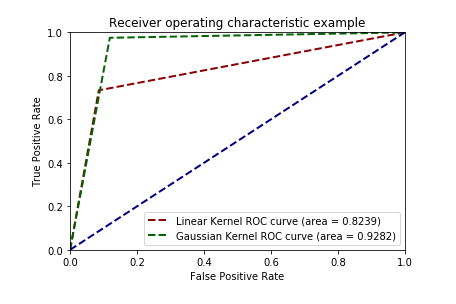
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic example')

plt.legend(loc="lower right")

plt.show()



## 5. Conclusion[¶](https://www.kaggle.com/suhasshastry/bank-marketing-analysis-for-term-deposit#5.-Conclusion)

For the given data, visualization of data, ways to treat imbalance in the data and best predictive model to determine the term deposit subscription was explored. From visualization, it can be derived that repeated campaign calls to customers within 20 days of previous call increases the subscription. After treating the imbalance in data using SMOTE, SVM with Gaussian kernel performs the best in terms of precision and recall.