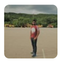
kaggle Search Competitions Datasets Notebooks Discussion Courses Sign in Registe



Cleaning, Visualizing and Modeling Cold Call Data

Python notebook using data from Car Insurance Cold Calls · 2,657 views · 2y ago

8 Copy and Edit 14 ...

Version 9
9 commits

Notebook Data Comments

CAR INSURANCE COLD CALLS - REPORT

ABOUT THE DATASET

This is a dataset from a bank in the United States. Besides usual services, this bank also provides carinsurance services. The bank organizes regular campaigns to attract new clients.The bank has potential customers data, and bank's employees call them for advertising available car insurance options. We are provided with general information about clients (age, job, etc.) as well as more specific information about the current insurance sell campaign (communication, last contact day) and previous campaigns(attributes like previous attempts, outcome).

CLIENT

STAR Bank is our Client, located in United States they operate in almost all states and they try to convert already existing insurance customers from a different agency to STAR by their marketing campaigns mostly ***Cold Call***

PROBLEM(S) TO SOLVE

The client wants to know the most important factor which determines cold call success so that they can work on it and further improve their business using the cold call data.The problem I am trying to solve involves creating predictive models and choosing the best model among them using model validation techniques to gain more insights about the key factors which contributes to cold call success and provide recommendations to improve cold call success as well. Further the model implementation can improve their business and help them on concentarting on the key areas to their success

Lets look at the features of the dataset and understand what each attribute/feature is about.The table below shows a brief description of the dataset and whether the variables are continuous, categorical or binary.

Feature	Description	Example
Id	Unique ID number. Predictions file should contain this feature.	“1” … “5000”
Age	Age of the client	
Job	Job of the client.	"admin.", "blue-collar", etc.
Marital	Marital status of the client	"divorced", "married", "single"
Education	Education level of the client	"primary", "secondary", etc.
Default	Has credit in default?	"yes" - 1,"no" - 0
Balance	Average yearly balance, in USD	
HHInsurance	Is household insured	"yes" - 1,"no" - 0
CarLoan	Has the client a car loan	"yes" - 1,"no" - 0
Communication	Contact communication type	"cellular", "telephone", “NA”
LastContactMonth	Month of the last contact	"jan", "feb", etc.
LastContactDay	Day of the last contact	
CallStart	Start time of the last call (HH:MM:SS)	12:43:15
CallEnd	End time of the last call (HH:MM:SS)	12:43:15
NoOfContacts	Number of contacts performed uring this campaign for this client	
DaysPassed	Number of days that passed by after the client was last contacted	
	from a previous campaign (numeric; -1 means client was not	
	previously contacted)	
PrevAttempts	Number of contacts performed before this campaign and for this client	
Outcome	Outcome of the previous marketing campaign	"failure", "other", "success", “NA”
CarInsurance	Has the client subscribed a CarInsurance?	"yes" - 1,"no" - 0

DATA WRANGLING / DATA MUNGING

Data Wrangling or Data Munging is the process of converting data from one form to another to better understand it. Here in our case our data is availabe to us as a CSV file and lets use our powerful python data science libraries to load it into a dataframe. Well I never thought it would look so easy !!

In [1]:

```
# Importing Data Science Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
%matplotlib inline
from sklearn.model_selection import train_test_split,c
ross_val_score,KFold,cross_val_predict
from sklearn.metrics import accuracy_score, classifica
tion_report, precision_score, recall_score,confusion_m
atrix,precision_recall_curve,roc_curve
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import ExtraTreesClassifier,Rand
omForestClassifier,AdaBoostClassifier,GradientBoosting
Classifier
from sklearn.neighbors  import KNeighborsClassifier
from sklearn import svm,tree
```

Here indexing already existing column saves a lot of time and hassle. Trust me

In [2]:

```
# Reading Csv file
df = pd.read_csv('../input/carInsurance_train.csv',ind
ex_col = 'Id')
```

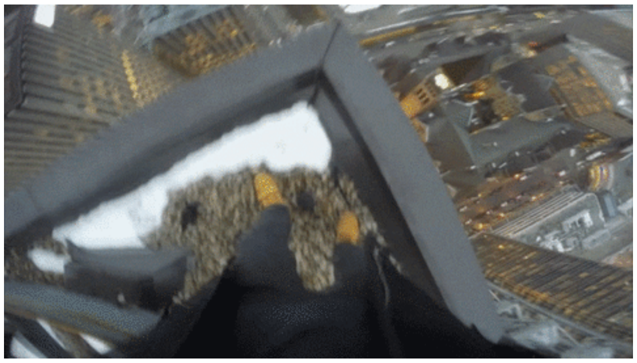
Sneak peek into our top 5 rows of the dataframe. Well, our data looks too good

In [3]:

```
# Top rows
df.head()
```

Out[3]:

	Age	Job	Marital	Education	Default	Balanc
Id						
1	32	management	single	tertiary	0	1218
2	32	blue-collar	married	primary	0	1156
3	29	management	single	tertiary	0	637
4	25	student	single	primary	0	373
5	30	management	married	tertiary	0	2694



EXPLORATORY DATA ANALYSIS

Exploring is always fun and the new insights you can find is always interesting. Starting from the shape of the dataset to knowing about the columns, datatypes and the statistics part of it gives us a lot more of understanding and deep dive into the data. By having a look at the Balance amount in our dataset 75% of the field is 1619 and the max is 98417. Nice, we are exploring !!

In [4]:

```
# Shape of dataframe
df.shape
```

Out[4]:

```
(4000, 18)
```

In [5]:

```
# Columns in dataset
df.columns
```

Out[5]:

```
Index(['Age', 'Job', 'Marital', 'Educatio
n', 'Default', 'Balance',
       'HHInsurance', 'CarLoan', 'Communi
cation', 'LastContactDay',
       'LastContactMonth', 'NoOfContact
s', 'DaysPassed', 'PrevAttempts',
       'Outcome', 'CallStart', 'CallEnd',
'CarInsurance'],
      dtype='object')
```

Looking at our Numerical columns Default, HHInsurance, CarLoan,CarInsurance are binary having 0's' and 1's'

In [6]:

```
# Statistics of numerical columns
df.describe()
```

Out[6]:

	Age	Default	Balance	HHInsuran
count	4000.000000	4000.000000	4000.000000	4000.0000
mean	41.214750	0.014500	1532.937250	0.49275
std	11.550194	0.119555	3511.452489	0.50001
min	18.000000	0.000000	-3058.000000	0.00000
25%	32.000000	0.000000	111.000000	0.00000
50%	39.000000	0.000000	551.500000	0.00000
75%	49.000000	0.000000	1619.000000	1.00000
max	95.000000	1.000000	98417.000000	1.00000

In [7]:

```
# Datatypes of columns in dataset
df.dtypes
```

Out[7]:

```
Age                  int64
Job                 object
Marital             object
Education           object
Default              int64
Balance              int64
HHInsurance          int64
CarLoan              int64
Communication       object
LastContactDay       int64
LastContactMonth    object
NoOfContacts         int64
DaysPassed           int64
PrevAttempts         int64
Outcome             object
CallStart           object
CallEnd             object
CarInsurance         int64
dtype: object
```

In [8]:

```
# Statistics of categorical features
df.describe(include=['O'])
```

Out[8]:

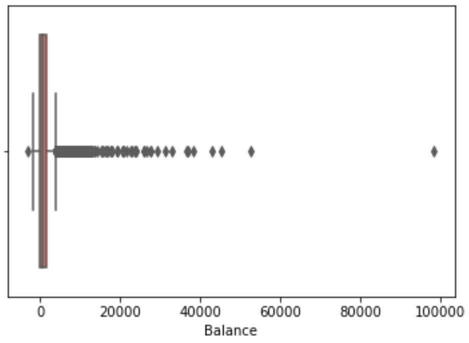
	Job	Marital	Education	Communication
count	3981	4000	3831	3098
unique	11	3	3	2
top	management	married	secondary	cellular
freq	893	2304	1988	2831

OUTLIER ANALYSIS

An Outlier is usually an observation which typically lies farthest from the mean. According to Statistical theory if any observation is 3*IQR(Inter Quartile Range) from the mean then its called an Outlier. Sometimes values are distributed randomly such as a Balance amount - from our boxplot below looks like more values have crossed the whisker. One particular data point is too far when compared to the other points in the data and the outlier is dropped from the dataset.

In [9]:

```
# Plotting Balance field as a Boxplot using Seaborn
sns.boxplot(x='Balance',data=df,palette='hls');
```



In [10]:

```
# Maximum value in Balance field
df.Balance.max()
```

Out[10]:

```
98417
```

In [11]:

```
# Looking at the particular maximum value in the datafr
ame
df[df['Balance'] == 98417]
```

Out[11]:

	Age	Job	Marital	Education	Default	Bal
Id						
1743	59	management	married	tertiary	0	984

In [12]:

```
# Dropping the index value corresponding to the outlier
df_new = df.drop(df.index[1742]);
```

HANDLING MISSING VALUES

Missing values are a major concern with data analysis and dealing them is another hurdle. Python treats missing data as NaN but doesnot include it into calcualtions and visulizations. Also predictive models cannot be built without treating missing values. In our case missing values occurs mostly in Outcome and Communication fields. Job and Education have considerable amount of missing values.

IMPUTING MISSING VALUES The missing values like Job and Education are very few and can imputed using backfill/frontfill pad method in python .Outcome and Communication have quite a lot missing values and hence they are imputed using None for NaN values.

In [13]:

```
#checking for missing values using isnull() method
df_new.isnull().sum()
```

Out[13]:

```
Age                    0
Job                   19
Job                  19
Marital               0
Education           169
Default               0
Balance               0
HHInsurance           0
CarLoan               0
Communication       902
LastContactDay        0
LastContactMonth      0
NoOfContacts          0
DaysPassed            0
PrevAttempts          0
Outcome            3041
CallStart             0
CallEnd               0
CarInsurance          0
dtype: int64
```

In [14]:

```
# Using frontfill to fill the missing values in Job and
Education fields
df_new['Job'] = df_new['Job'].fillna(method ='pad')
df_new['Education'] = df_new['Education'].fillna(metho
d ='pad')
```

In [15]:

```
# Using none to fill Nan values in Communication and Ou
tcome fields
df_new['Communication'] = df_new['Communication'].fill
na('none')
df_new['Outcome'] = df_new['Outcome'].fillna('none')
```

In [16]:

```
#Looks like all missing values have been imputed
df_new.isnull().sum()
```

Out[16]:

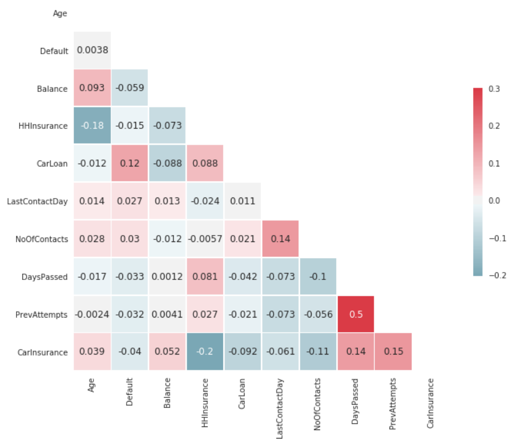
```
Age                 0
Job                 0
Marital             0
Education           0
Default             0
Balance             0
HHInsurance         0
CarLoan             0
Communication       0
LastContactDay      0
LastContactMonth    0
NoOfContacts        0
DaysPassed          0
PrevAttempts        0
Outcome             0
CallStart           0
CallEnd             0
CarInsurance        0
dtype: int64
```

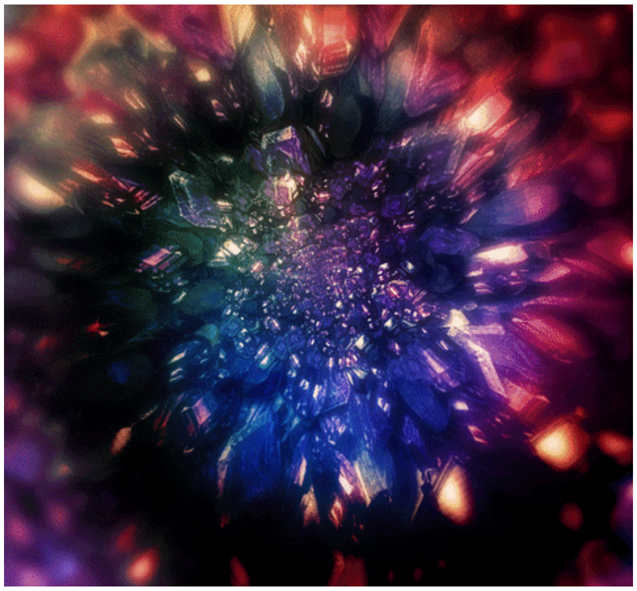
CORRELATION

Correlation is used to determine the relationship between two variables/ fields. Correlation varies from -1 to 1; if the Correlation is 1 then the fields are positively correlated, 0 having no correlation and -1 is negatively correlated. Lets see how each atttribute correlates with one another using Heatmap. Looks like there is not much of a correlation among variables but DaysPassed and PrevAttempts have a positive Correlation with each other.

In [17]:

```
#Setting up correlation for our dataframe and passing i
t to seaborn heatmap function
sns.set(style="white")
corr = df_new.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(corr,annot=True, mask=mask, cmap=cmap, vma
x=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shr
ink": .5});
```





DATA VISUALIZATION

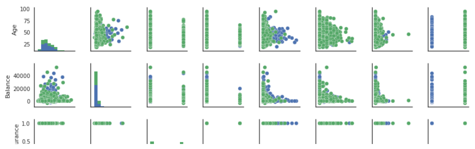
Visualization is an important aspect of Data Science without which its pretty diffcult to arrive at an outcome easily.Eventhough the result is determined in tables there is a painpoint in looking at each values and coming to a conclusion. Charts/Graphs are much helpful to accomplish those tasks with ease even to a non-technical person. Executives and managers love to look at a report with visualization so that they can easily come up with complex decisions.Below is a pairplot which pairs fields of interest and plots them. The variables for the Pairplot are selected from the heatmap which have an impact on the outcome

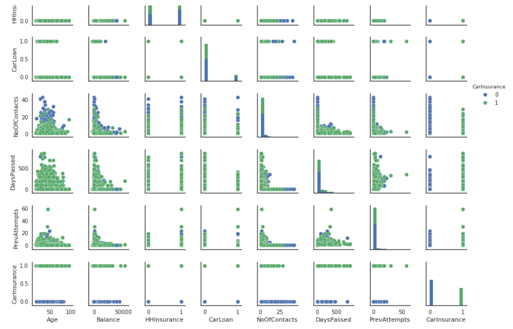
Key takeaways from the Pairplot

```
*Older people are more likely to Buy Car Insurance.
*People having prior Car Insurance and Home Insuranc
e are less likely to purchase.
*People give a positive sign if the days passed (tim
e before they were contacted) increases.
*When you contact persons frequently their buying te
ndency increases after 20+ contacts.
*No.of contacts and PrevAttempts work the same , mor
e the better i.e increases Car Insurance purchase.
</span>
```

In [18]:

```
# Plotting paired fields of intrest using Seaborn pairp
lot
df_sub = ['Age','Balance','HHInsurance', 'CarLoan','No
OfContacts','DaysPassed','PrevAttempts','CarInsurance'
]
sns.pairplot(df_new[df_sub],hue='CarInsurance',size=1.
5);
```

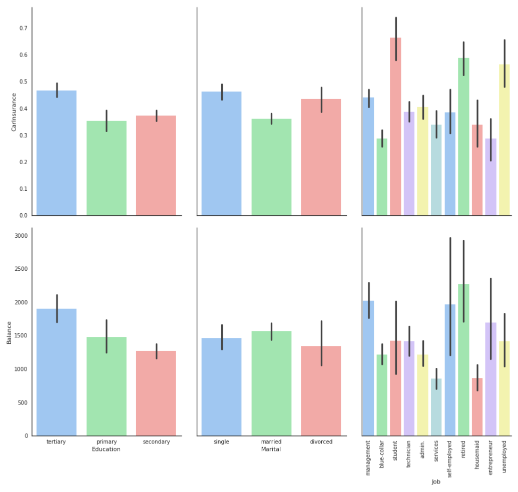




PairGrid helps us to view relationships between CarInsurance,Balance and Categorical variables such as Education,Marital Status and Job.Students and Retired people have purchased the most Car Insurances with Former leading the Latter People with single status and who are highly educated dominate the charts.

In [19]:

```
#Uses multiple x and y variables to form pair grid of c
ategorical values passed
g = sns.PairGrid(df_new,
                 x_vars=["Education","Marital", "Job"
],
                 y_vars=["CarInsurance", "Balance"],
                 aspect=.75, size=6)
plt.xticks(rotation=90)
g.map(sns.barplot, palette="pastel");
```

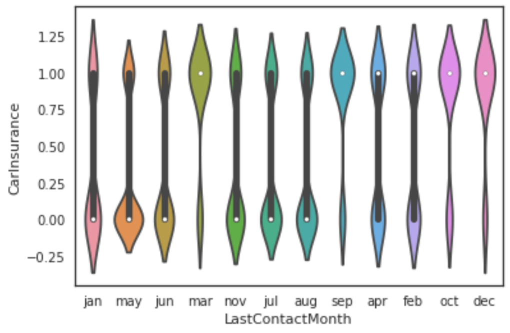


LOOKS INTERESTING

Violin plot has bulges near value 1 in y axis shows that Mar, Sep, Oct and Dec are the desired months for people buying Car Insurance.The Count plot below has more missing previous campaign outcome where majority said **No** to car insurance.

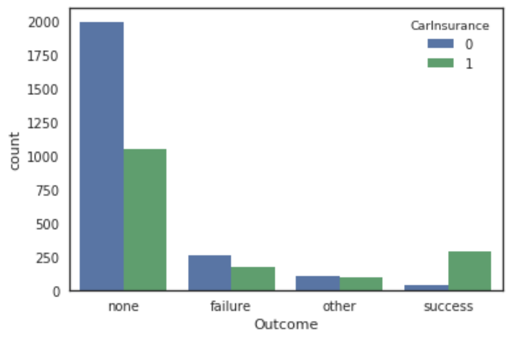
In [20]:

```
#Seaborn violin plot for LastContactMonth and CarInsura
nce fields
sns.violinplot(x="LastContactMonth",y='CarInsurance',d
ata=df_new);
```



In [21]:

```
#Count of CarInsurance against Outcome i.e previous cam
paign outcome
sns.countplot(x="Outcome",hue='CarInsurance',data=df_n
ew);
```



FEATURE ENGINEERING/ FEATURE EXTRACTION

Feature Engineering is an essential element to a Machine Learning Problem. Picking a feature from a set of attribute determines how well the algorithm will work in making predictions, so this part is a crucial one. In our problem there are a list of continuous variables like Age and Balance and they need to be binned. The Age and Balance continuous variables are bucketed using quartile cut function into 5 segments.

In [22]:

```
#Qcut splits both the attribute into 5 buckets
df_new['AgeBinned'] = pd.qcut(df_new['Age'], 5 , label
s = False)
df_new['BalanceBinned'] = pd.qcut(df_new['Balance'], 5
,labels = False)
```

There seems to be a unique problem with respect to the CallStart and CallEnd attributes and are recorded as object variables which can be computed easily using the datetime function, so converting it to datetime function and subtracting them arrives at the actual CallTime which can be further binned as above.

In [23]:

```
#Converting CallStart and CallEnd to datetime datatype
df_new['CallStart'] = pd.to_datetime(df_new['CallStar
t'] )
df_new['CallEnd'] = pd.to_datetime(df_new['CallEnd'] )
#Subtracting both the Start and End times to arrive at
 the actual CallTime
df_new['CallTime'] = (df_new['CallEnd'] - df_new['Call
Start']).dt.total_seconds()
#Binning the CallTime
df_new['CallTimeBinned'] = pd.qcut(df_new['CallTime'],
5,labels = False)
```

In [24]:

```
#Dropping the original columns of the binned, just to m
ake things easy
df_new.drop(['Age','Balance','CallStart','CallEnd','Ca
llTime'],axis = 1,inplace = True)
```

Categorical variables can also paricipate in model building provided that they get their dummy values inorder to be included. Well, we would have more columns included to our dataframe by this procedure.

In [25]:

```
# Using get_dummies function to assign binary values to
each value in the categorical column
Job = pd.get_dummies(data = df_new['Job'],prefix = "Jo
b")
Marital= pd.get_dummies(data = df_new['Marital'],prefi
x = "Marital")
Education= pd.get_dummies(data = df_new['Education'],p
refix="Education")
Communication = pd.get_dummies(data = df_new['Communic
ation'],prefix = "Communication")
LastContactMonth = pd.get_dummies(data = df_new['LastC
ontactMonth'],prefix= "LastContactMonth")
Outcome = pd.get_dummies(data = df_new['Outcome'],pref
ix = "Outcome")
```

In [26]:

```
# Dropping the categorical columns which have been assi
gned dummies
df_new.drop(['Job','Marital','Education','Communicatio
n','LastContactMonth','Outcome'],axis=1,inplace=True)
```

In [27]:

```
#Concatenating the dataframe with the categorical dummy
```

```
columns
df = pd.concat([df_new,Job,Marital,Education,Communica
tion,LastContactMonth,Outcome],axis=1)
```

In [28]:

```
# The dataframe has some new additions resulting from t
he categorical dummies added
df.columns
```

Out[28]:

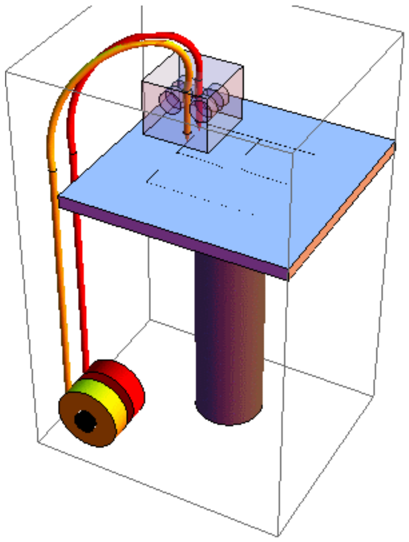
```
Index(['Default', 'HHInsurance', 'CarLoa
n', 'LastContactDay', 'NoOfContacts',
       'DaysPassed', 'PrevAttempts', 'Car
Insurance', 'AgeBinned',
       'BalanceBinned', 'CallTimeBinned',
'Job_admin.', 'Job_blue-collar',
       'Job_entrepreneur', 'Job_housemai
d', 'Job_management', 'Job_retired',
       'Job_self-employed', 'Job_service
s', 'Job_student', 'Job_technician',
       'Job_unemployed', 'Marital_divorce
d', 'Marital_married',
       'Marital_single', 'Education_prima
ry', 'Education_secondary',
       'Education_tertiary', 'Communicati
on_cellular', 'Communication_none',
       'Communication_telephone', 'LastCo
ntactMonth_apr',
       'LastContactMonth_aug', 'LastConta
ctMonth_dec', 'LastContactMonth_feb',
       'LastContactMonth_jan', 'LastConta
ctMonth_jul', 'LastContactMonth_jun',
       'LastContactMonth_mar', 'LastConta
ctMonth_may', 'LastContactMonth_nov',
       'LastContactMonth_oct', 'LastConta
ctMonth_sep', 'Outcome_failure',
       'Outcome_none', 'Outcome_other',
'Outcome_success'],
      dtype='object')
```

TEST TRAIN SPLIT

The Train Test Split is usually done to evaluate our model by Training it on the known output(labeled data) so that the model can learn on it and Testing using unlabeled data so that the predictive accuracy of the model can be determined.

In [29]:

```
# Dropping the Target for X
X= df.drop(['CarInsurance'],axis=1).values
# Including only the Target for y
y=df['CarInsurance'].values
#Splitting the Training and Testing data having 20% of
 Test data
X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.20,random_state=42, stratify = y)
```



PREDICTIVE MODEL BUILDING AND VALIDATION

PREDICTIVE MODELS

Predictive Models are built to correctly classify the unknown label inputs, the models are trained using the labeled outputs so that it can learn from them and correctly classify the non labeled items. There are a lot of Classification Predictor Algorithms incorporated into sklearn and in our case we have utilized most of the classification algorithms related to our problem. Our Classifiers include

1. kNN
2. Logistic Regression
3. SVM
4. Decision Tree
5. Random Forest
6. AdaBoost
7. XGBoost

CROSS VALIDATION

Cross-validation is used to split the data into training and test sets to evaluate how the model performs. In KFold, K determins the number of partitions to be made on the data and from which 1 sample is used for training and 10-1 in our case 9 is used for the validation purposes. Each model's cross validation score is obtained by evaluating the model by splitting it into 10 Folds.

MODEL VALIDATION

Validating our models built is a key component which helps in determining how our model's predictive power. Starting from the most common accuracy score, cross validation score to classification report(precision, recall, f1-score,support), ROC curves and Confusion matrix , the models have gone through extensive validation to choose the best predictor.

BEST MODEL

The best model is a tie between **Random Forest** and **XGBoost** both doing their part well with good accuracy scores, less false positives and true negatives.

In [30]:

```
#The code for the below matrix is taken from sklearn do
cumentation
#Defining the confusion matrix function
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):



    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)




    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh el
se "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
#Using Success and Failure for 0 and 1
class_names = ['Success','Failure']
```

In [31]:

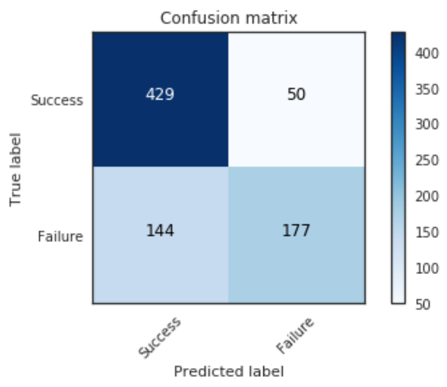
```
# Defining the kNNClassifier with 6 neighbors
knn = KNeighborsClassifier(n_neighbors = 6)
#Fitting the classifier to the training set
knn.fit(X_train,y_train)
print ("kNN Accuracy is %2.2f" % accuracy_score(y_test
, knn.predict(X_test)))
#The cross validation score is obtained for kNN using 1
0 folds
score_knn = cross_val_score(knn, X, y, cv=10).mean()
print("Cross Validation Score = %2.2f" % score_knn)
y_pred= knn.predict(X_test)
print(classification_report(y_test, y_pred))
#Defining the confusion matrix
cm = confusion_matrix(y_test,y_pred)
#Plotting the confusion matrix
plot_confusion_matrix(cm, classes=class_names, title=
'Confusion matrix')
```

```
kNN Accuracy is 0.76
Cross Validation Score = 0.75
             precision    recall  f1-scor
e   support

           0       0.75      0.90      0.8
```

```
2       479
          1       0.78      0.55      0.6
5       321

avg / total       0.76      0.76      0.7
5       800
```



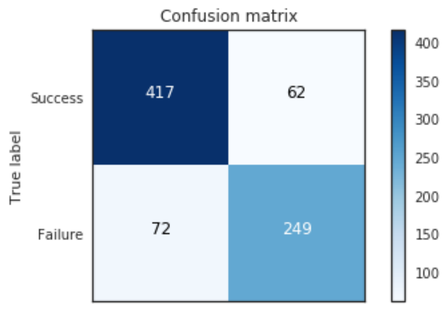
In [32]:

```
#Logistic Regression Classifier
LR = LogisticRegression()
LR.fit(X_train,y_train)
print ("Logistic Accuracy is %2.2f" % accuracy_score(y
_test, LR.predict(X_test)))
score_LR = cross_val_score(LR, X, y, cv=10).mean()
print("Cross Validation Score = %2.2f" % score_LR)
y_pred = LR.predict(X_test)
print(classification_report(y_test, y_pred))
# Confusion matrix for LR
cm = confusion_matrix(y_test,y_pred)
plot_confusion_matrix(cm, classes=class_names, title=
'Confusion matrix')
```

```
Logistic Accuracy is 0.83
Cross Validation Score = 0.81
             precision    recall  f1-scor
e   support

          0       0.85      0.87      0.8
6       479
          1       0.80      0.78      0.7
9       321

avg / total       0.83      0.83      0.8
3       800
```



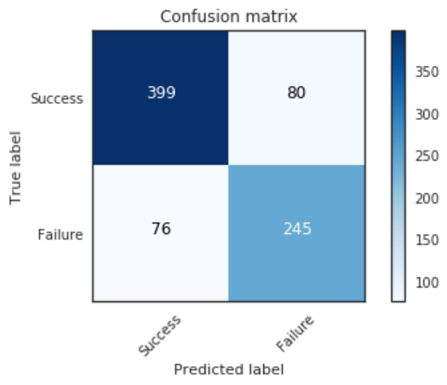
In [33]:

```
#SVM Classifier
SVM = svm.SVC()
SVM.fit(X_train, y_train)
print ("SVM Accuracy is %2.2f" % accuracy_score(y_test
, SVM.predict(X_test)))
score_svm = cross_val_score(SVM, X, y, cv=10).mean()
print("Cross Validation Score = %2.2f" % score_svm)
y_pred = SVM.predict(X_test)
print(classification_report(y_test,y_pred))
#Confusion matrix for SVM
cm = confusion_matrix(y_test,y_pred)
plot_confusion_matrix(cm, classes=class_names, title=
'Confusion matrix')
```

```
SVM Accuracy is 0.81
Cross Validation Score = 0.81
             precision    recall  f1-scor
e   support

          0       0.84      0.83      0.8
4       479
          1       0.75      0.76      0.7
6       321

avg / total       0.81      0.81      0.8
1       800
```



In [34]:

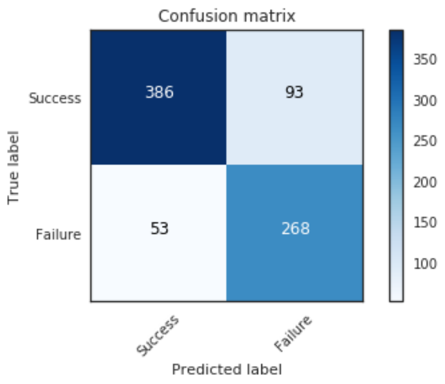
```
# Decision Tree Classifier
DT = tree.DecisionTreeClassifier(random_state = 0,clas
s_weight="balanced",
    min_weight_fraction_leaf=0.01)
DT = DT.fit(X_train,y_train)
print ("Decision Tree Accuracy is %2.2f" % accuracy_sc
ore(y_test, DT.predict(X_test)))
score_DT = cross_val_score(DT, X, y, cv=10).mean()
print("Cross Validation Score = %2.2f" % score_DT)
y_pred = DT.predict(X_test)
print(classification_report(y_test, y_pred))
# Confusion Matrix for Decision Tree
```

```
cm = confusion_matrix(y_test,y_pred)
plot_confusion_matrix(cm, classes=class_names, title=
'Confusion matrix')
```

```
Decision Tree Accuracy is 0.82
Cross Validation Score = 0.81
             precision    recall  f1-scor
e   support

          0       0.88      0.81      0.8
4       479
          1       0.74      0.83      0.7
9       321

avg / total       0.82      0.82      0.8
2       800
```



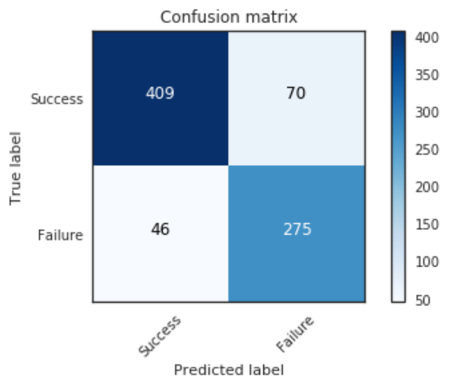
In [35]:

```
#Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=1000, max_de
pth=None, min_samples_split=10,class_weight="balanced"
)
rfc.fit(X_train, y_train)
print ("Random Forest Accuracy is %2.2f" % accuracy_sc
ore(y_test, rfc.predict(X_test)))
score_rfc = cross_val_score(rfc, X, y, cv=10).mean()
print("Cross Validation Score = %2.2f" % score_rfc)
y_pred = rfc.predict(X_test)
print(classification_report(y_test,y_pred ))
#Confusion Matrix for Random Forest
cm = confusion_matrix(y_test,y_pred)
plot_confusion_matrix(cm, classes=class_names, title=
'Confusion matrix')
```

```
Random Forest Accuracy is 0.85
Cross Validation Score = 0.84
             precision    recall  f1-scor
e   support

          0       0.90      0.85      0.8
8       479
          1       0.80      0.86      0.8
3       321

avg / total       0.86      0.85      0.8
6       800
```



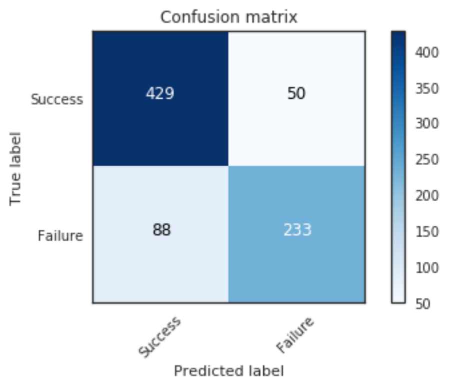
In [36]:

```
#AdaBoost Classifier
ada = AdaBoostClassifier(n_estimators=400, learning_ra
te=0.1)
ada.fit(X_train,y_train)
print ("AdaBoost Accuracy= %2.2f" % accuracy_score(y_t
est,ada.predict(X_test)))
score_ada = cross_val_score(ada, X, y, cv=10).mean()
print("Cross Validation Score = %2.2f" % score_ada)
y_pred = ada.predict(X_test)
print(classification_report(y_test,y_pred ))
#Confusion Marix for AdaBoost
cm = confusion_matrix(y_test,y_pred)
plot_confusion_matrix(cm, classes=class_names, title=
'Confusion matrix')
```

```
AdaBoost Accuracy= 0.83
Cross Validation Score = 0.82
             precision    recall  f1-scor
e   support

          0       0.83      0.90      0.8
6       479
          1       0.82      0.73      0.7
7       321

avg / total       0.83      0.83      0.8
3       800
```



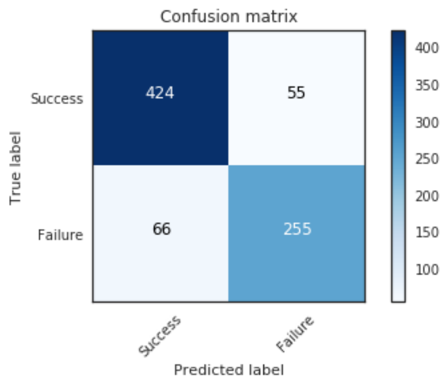
In [37]:

```
#XGBoost Classifier
xgb = GradientBoostingClassifier(n_estimators=1000,lea
rning_rate=0.01)
xgb.fit(X_train,y_train)
print ("GradientBoost Accuracy= %2.2f" % accuracy_scor
e(y_test,xgb.predict(X_test)))
score_xgb = cross_val_score(xgb, X, y, cv=10).mean()
print("Cross Validation Score = %2.2f" % score_ada)
y_pred = xgb.predict(X_test)
print(classification_report(y_test,y_pred))
#Confusion Matrix for XGBoost Classifier
cm_xg = confusion_matrix(y_test,y_pred)
plot_confusion_matrix(cm_xg, classes=class_names, titl
e='Confusion matrix')
```

```
GradientBoost Accuracy= 0.85
Cross Validation Score = 0.82
             precision    recall  f1-scor
e   support

          0       0.87      0.89      0.8
8       479
          1       0.82      0.79      0.8
1       321

avg / total       0.85      0.85      0.8
5       800
```



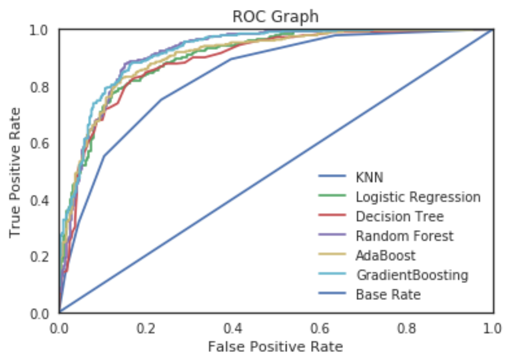
ROC CURVES

Another important visual model validation technique is the Reciever Operating Characteristic(ROC) Curves, which plots the true positive rate and the false postive rates. The curve is a good validator of the models and helps us determine whether our model works best. When the AOC(Area Under Curve) is maximum and when its towards the upper left then the model works best. The ROC has all the models plotted and Gradient Boosting(XGBoost) and Randomforest towards the upper left showing that those predictor models are the best.

In [38]:

```
#Obtaining False Positive Rate, True Positive Rate and
 Threshold for all classifiers
fpr, tpr, thresholds = roc_curve(y_test, knn.predict_p
roba(X_test)[:,1])
LR_fpr, LR_tpr, thresholds = roc_curve(y_test, LR.pred
ict_proba(X_test)[:,1])
#SVM_fpr, SVM_tpr, thresholds = roc_curve(y_test, SVM.p
redict_proba(X_test)[:,1])
DT_fpr, DT_tpr, thresholds = roc_curve(y_test, DT.pred
ict_proba(X_test)[:,1])
rfc_fpr, rfc_tpr, thresholds = roc_curve(y_test, rfc.p
redict_proba(X_test)[:,1])
ada_fpr, ada_tpr, thresholds = roc_curve(y_test, ada.p
redict_proba(X_test)[:,1])
xgb_fpr, xgb_tpr, thresholds = roc_curve(y_test, xgb.p
redict_proba(X_test)[:,1])
#PLotting ROC Curves for all classifiers
plt.plot(fpr, tpr, label='KNN' )
plt.plot(LR_fpr, LR_tpr, label='Logistic Regression')
#plt.plot(SVM_fpr, SVM_tpr, label='SVM')
plt.plot(DT_fpr, DT_tpr, label='Decision Tree')
plt.plot(rfc_fpr, rfc_tpr, label='Random Forest')
plt.plot(ada_fpr, ada_tpr, label='AdaBoost')
plt.plot(xgb_fpr, xgb_tpr, label='GradientBoosting')
# Plot Base Rate ROC
plt.plot([0,1],[0,1],label='Base Rate')

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 Graph')
plt.legend(loc="lower right")
plt.show()
```



FEATURE IMPORTANCES

Knowing which feature has a major part in determining the output can be very useful and help in working on them to increase the output of the solution. Finding the important features can be very handy when making important decisions and conclusions. The Important feature identification is done by using models such as Logistic Regression and Decision trees. Both of them provide very good clarity in identifying the features. The Graph below shows the most important features determined by the ExtraTreesClassifier and the top 10 features are

1. CallTime
2. LastContactDay
3. Balance
4. NoofContacts
5. Outcome_success
6. Age
7. HHInsurance
8. Communication_none
9. Dayspassed
10. Outcome_none

In [39]:

```
# Using Recursive Feature Elimination Function and fitt
ing it in a Logistic Regression Model
modell = LogisticRegression()
rfe = RFE(modell, 5)
rfe = rfe.fit(X_train,y_train)
# Displays the feature rank
rfe.ranking_
```

Out[39]:

```
array([13, 12, 16, 41, 37, 42, 35, 39, 3
6,  3, 22, 15, 17, 26, 31, 18, 20,
       32,  2, 34, 25, 27, 19, 28, 24, 2
3, 33, 30,  1, 29, 38,  7, 14, 40,
        4,  5, 21,  1,  6,  8,  1,  1, 1
0,  9, 11,  1])
```

In [40]:

```
# Using ExtraTreesClassifier model function
model = ExtraTreesClassifier()
model.fit(X_train, y_train)
# Printing important features in the model
print(model.feature_importances_)
importances = model.feature_importances_
```

```
feat_names = df.drop(['CarInsurance'],axis=1).columns

# Displaying the feature importances as a chart by sort
ing it in the order of importances
indices = np.argsort(importances)[::-1]
plt.figure(figsize=(12,6))
plt.title("Feature importances")
plt.bar(range(len(indices)), importances[indices], col
or='lightblue',  align="center")
plt.step(range(len(indices)), np.cumsum(importances[in
dices]), where='mid', label='Cumulative')
plt.xticks(range(len(indices)), feat_names[indices], r
otation='vertical',fontsize=14)
plt.xlim([-1, len(indices)])
plt.show()
```

```
[ 0.00257713  0.02942149  0.01602683  0.0
6692516  0.05389973  0.01550315
```

Did you find this Kernel useful?
Show your appreciation with an upvote

8

Data

Data Sources

- Car Insurance Cold Calls
  - carInsurance_test.csv 19 columns
  - carInsurance_train.csv 19 columns
  - DSS_DMC_Description.pdf

Car Insurance Cold Calls

We help the guys and girls at the front to get out of Cold Call Hell

Last Updated: 2 years ago (Version 1)

About this Dataset

Introduction

Here you find a very simple, beginner-friendly data set. No sparse matrices, no fancy tools needed to understand what's going on. Just a couple of rows and columns. Super simple stuff. As explained below, this data set is used for a competition. As it turns out, this competition tends to reveal a common truth in data science: KISS - Keep It Simple Stupid

What is so special about this data set is, given it's simplicity, it pays off to use "simple" classifiers as well. This year's competition was won by a C5.0 . Can you do better?

Description

We are looking at cold call results. Turns out, same salespeople called existing insurance customers up and tried to sell car insurance. What you have are details about the called customers. Their age, job, marital status, whether the have home insurance, a car loan, etc. As I said, super simple.

What I would love to see is some of you applying some crazy XGBoost classifiers, which we can square off against some logistic regressions. It would be curious to see what comes out on top. Thank you for your time, I hope you enjoy using the data set.

Acknowledgements

Thanks goes to the Decision Science and Systems Chair of
Technical University of Munich (TUM) for getting the data set

Comments (2)

Sort by
All Comments | Hotness

Please sign in to leave a comment.

Aleksey Bilogur • Posted on Latest Version • 2 years ago 0

Come for the analysis, stay for the kooky animations. Nice notebook!

Manikandan Bha... Kernel Author • Posted on Latest Version • 2 years ago 0

Thank you, Aleksey. Glad you liked it !

Our Team Terms Privacy Contact/Support