Cold Calls

Import Libraries

In [1]:

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassif
ier,GradientBoostingClassifier, VotingClassifier
from sklearn.naive_bayes import GaussianNB
```

Import Data

In [2]:

```
train=pd.read_csv('../input/carInsurance_train.csv')
test=pd.read_csv('../input/carInsurance_test.csv')
```

In [3]:

```
print('The train dataset has %d observations and %d features' % (train
.shape[0], train.shape[1]))
print('The test dataset has %d observations and %d features' % (test.s
hape[0], test.shape[1]))
```

```
The train dataset has 4000 observations and 19 features
The test dataset has 1000 observations and 19 features
```

Data Exploration & Visualization¶

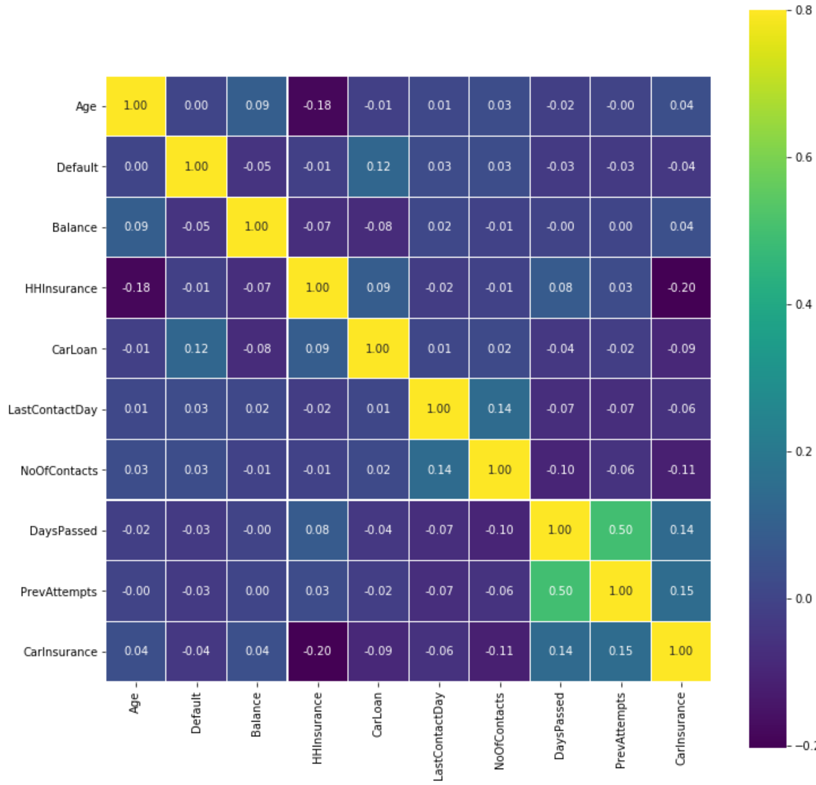
In [4]:

```
# First check out correlations among numeric features
# Heatmap is a useful tool to get a quick understanding of which variabl
es are important
colormap = plt.cm.viridis
cor = train.corr()
cor = cor.drop(['Id'],axis=1).drop(['Id'],axis=0)
plt.figure(figsize=(12,12))
sns.heatmap(cor,vmax=0.8,cmap=colormap,annot=True,fmt='.2f',square=Tru
```

```
e,annot_kws={'size':10},linecolor='white',linewidths=0.1)
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe2f78f6a20>
```



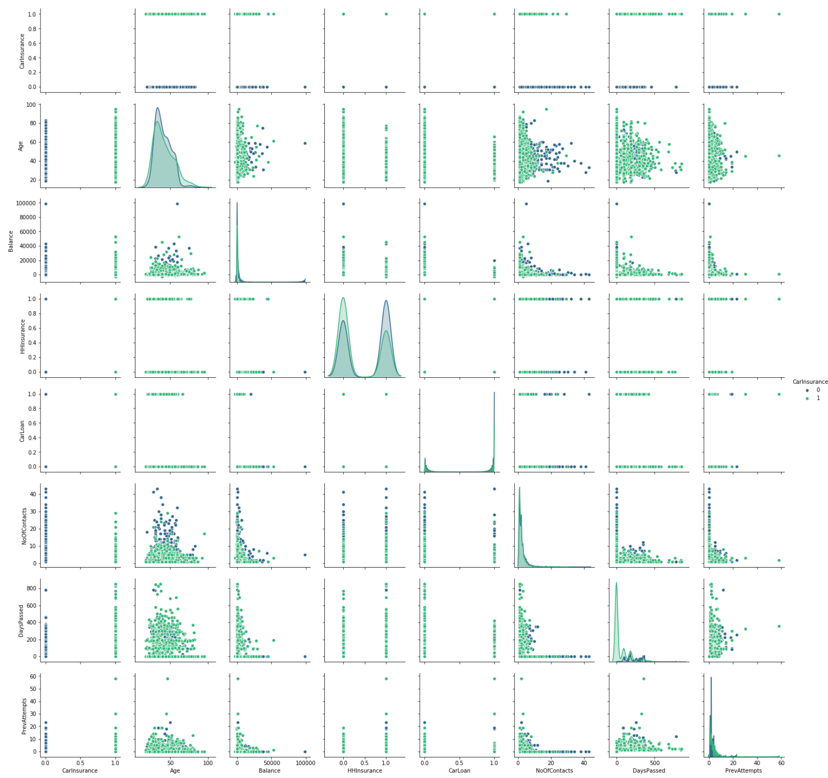
Features are fairly independent, except DaysPassed and PreAttempts. Cold call success is positively correlated with PreAttemps,DaysPassed,Age and Balance, and negatively correlated with default, HHInsurance, CarLoan, LastContactDay and NoOfContacts.

In [5]:

```
imp_feats = ['CarInsurance','Age','Balance','HHInsurance', 'CarLoan',
'NoOfContacts','DaysPassed','PrevAttempts']
sns.pairplot(train[imp_feats],hue='CarInsurance',palette='viridis',siz
e=2.5)
plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/seaborn/axisgrid.py:2065: UserW
arning: The `size` parameter has been renamed to `height`; pleaes upda
te your code.
  warnings.warn(msg, UserWarning)
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: Futu
reWarning: Using a non-tuple sequence for multidimensional indexing is
deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future
this will be interpreted as an array index, `arr[np.array(seq)]`, whic
h will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
/opt/conda/lib/python3.6/site-packages/statsmodels/nonparametric/kde.p
y:488: RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/opt/conda/lib/python3.6/site-packages/statsmodels/nonparametric/kdeto
ols.py:34: RuntimeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
/opt/conda/lib/python3.6/site-packages/numpy/core/fromnumeric.py:83: R
untimeWarning: invalid value encountered in reduce
```

```
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
```



Age: It's interesting to see that seniors are more likely to buy car insurance.

Balance: For balance, the data point at the upper right corner might be an outlier

HHInsurance: Households insured are less likely to buy car insurance

CarLoan: People with car loan are less likely to buy

NoOfContacts: Too many contacts causes customer attrition

DaysPassed: It looks like the more day passed since the last contact, the better

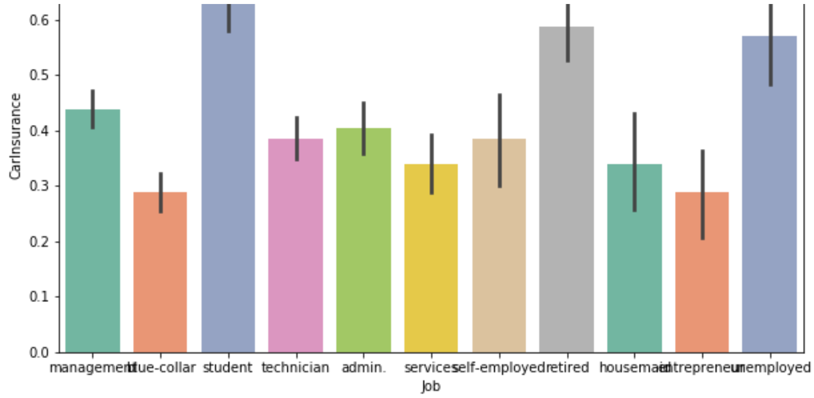
PrevAttempts: Also, more previous attempts, less likely to buy. There is a potential outlier here

In [6]:

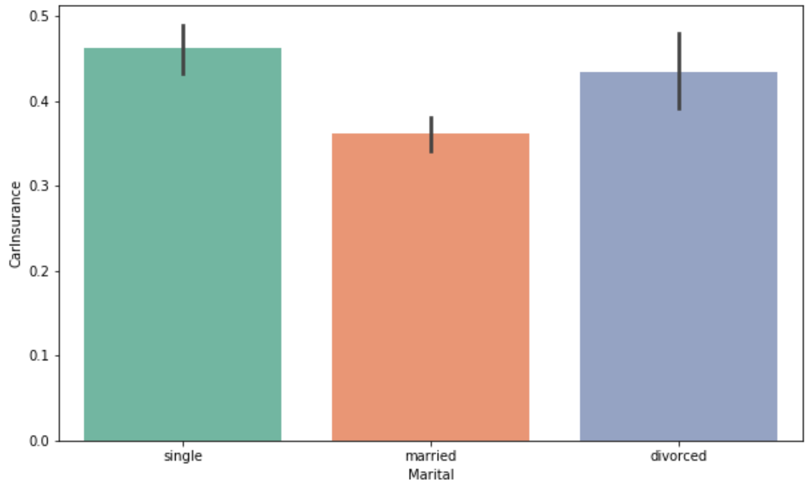
```
# Next check out categorical features
cat_feats = train.select_dtypes(include=['object']).columns
plt_feats = cat_feats[(cat_feats!= 'CallStart') & (cat_feats!='CallEn
d')]

for feature in plt_feats:
    plt.figure(figsize=(10,6))
    sns.barplot(feature,'CarInsurance', data=train,palette='Set2')
```

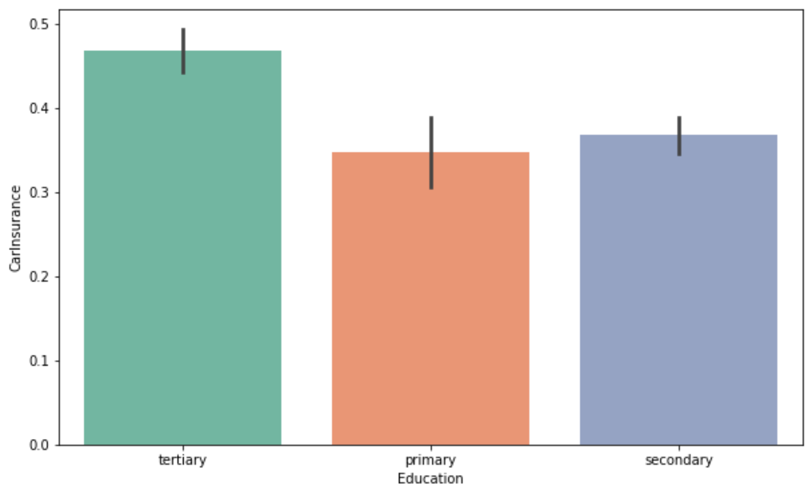
```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: Futu
reWarning: Using a non-tuple sequence for multidimensional indexing is
deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future
this will be interpreted as an array index, `arr[np.array(seq)]`, whic
h will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



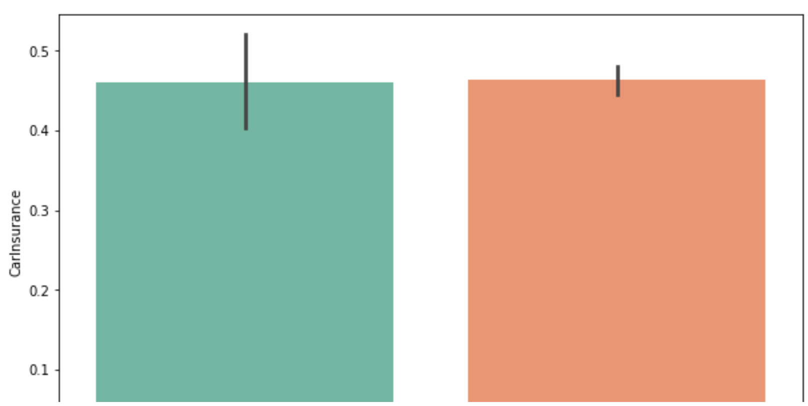
CarInsurance
0.0
0.1
0.2
0.3
0.4
0.5
0.6
management
blue-collar
student
technician
admin.
services
self-employed
retired
housemaid
entrepreneur
unemployed
Job



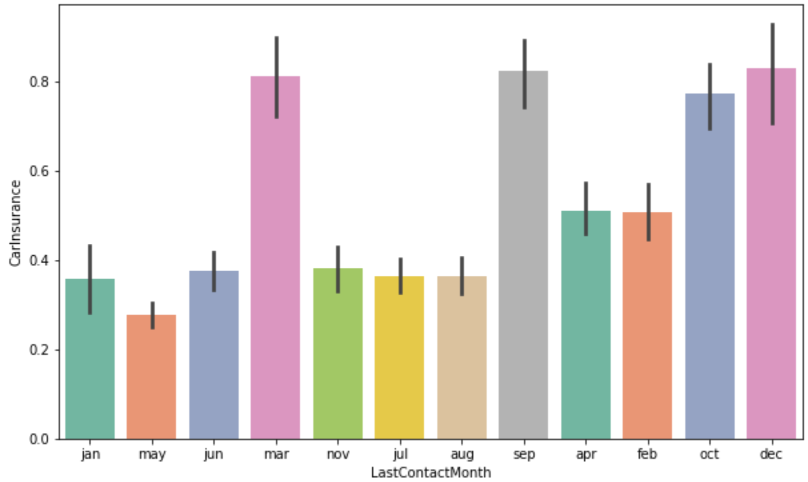
CarInsurance
0.0
0.1
0.2
0.3
0.4
0.5
single
married
divorced
Marital

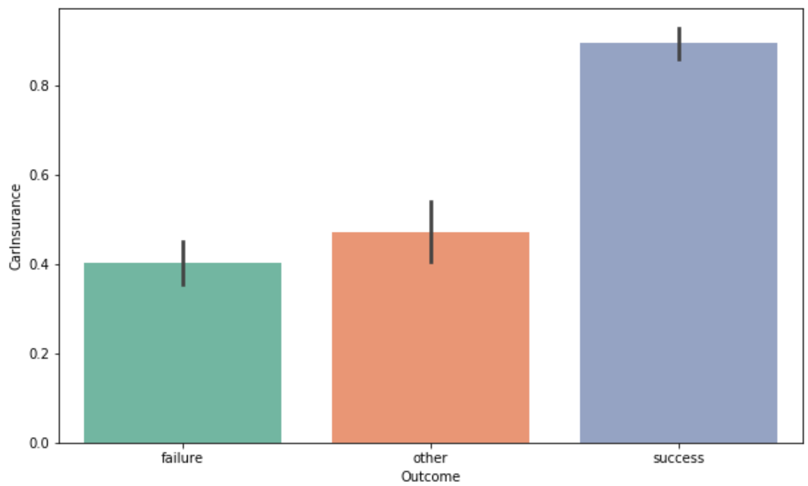


CarInsurance
0.0
0.1
0.2
0.3
0.4
0.5
tertiary
primary
secondary
Education



CarInsurance
0.1
0.2
0.3
0.4
0.5





Job: Student are most likely to buy insurance, followed by retired and unemployed folks.This is aligned with the age distribution. There might be some promotion targeting students?

Marital status: Married people are least likely to buy car insurance. Opportunities for developing family insurance business

Education: People with higher education are more likely to buy

Communication: No big difference between cellular and telephone

Outcome in previous campaign: Success in previous marketing campaign is largely associated with success in this campaign

Contact Month: Mar, Sep, Oct, and Dec are the hot months. It might be associated with school season?

In [7]:

```
# Check outliers
# From the pairplot, we can see there is an outlier with extreme high ba
lance. Drop that obs here.
train[train['Balance']>80000]
train = train.drop(train[train.index==1742].index)
```

Handling Miss Data

In [8]:

```
# merge train and test data here in order to impute missing values all a
t once
all=pd.concat([train,test],keys=('train','test'))
all.drop(['CarInsurance','Id'],axis=1,inplace=True)
print(all.shape)
```

```
(4999, 17)
```

In [9]:

```
total = all.isnull().sum()
pct = total/all.isnull().count()
NAs = pd.concat([total,pct],axis=1,keys=('Total','Pct'))
NAs[NAs.Total>0].sort_values(by='Total',ascending=False)
```

Out[9]:

	Total	Pct
Outcome	3798	0.759752
Communication	1123	0.224645
Education	216	0.043209
Job	24	0.004801

In [10]:

```
all_df = all.copy()

# Fill missing outcome as not in previous campaign
all_df[all_df['DaysPassed']==-1].count()
all_df.loc[all_df['DaysPassed']==-1,'Outcome']='NoPrev'

# Fill missing communication with none
all_df['Communication'].value_counts()
all_df['Communication'].fillna('None',inplace=True)

all_df['Education'].value_counts()
```

Out[10]:

```
secondary    2489
tertiary     1600
primary       694
Name: Education, dtype: int64
```

In [11]:

```
# Create job-education level mode mapping
edu_mode=[]
job_types = all_df.Job.value_counts().index
for job in job_types:
    mode = all_df[all_df.Job==job]['Education'].value_counts().nlarges
t(1).index
    edu_mode = np.append(edu_mode,mode)
edu_map=pd.Series(edu_mode,index=all_df.Job.value_counts().index)
edu_map
```

Out[11]:

```
management        tertiary
blue-collar      secondary
technician       secondary
```

```
admin.           secondary
services         secondary
retired          secondary
self-employed     tertiary
unemployed       secondary
student          secondary
entrepreneur      tertiary
housemaid          primary
dtype: object
```

In [12]:

```
#  Apply the mapping to missing eductaion obs
for j in job_types:
    all_df.loc[(all_df['Education'].isnull()) & (all_df['Job']==j),'Ed
ucation'] = edu_map.loc[edu_map.index==j][0]
all_df['Education'].fillna('None',inplace=True)

# Fill missing job with none
all_df['Job'].fillna('None',inplace=True)

# Double check if there is still any missing value
all_df.isnull().sum().sum()
```

Out[12]:

```
0
```

Feature Engineering

There are three types of features:

Client features: Age, Job, Marital, Education, Default, Balance, HHInsurance, CarLoan

Communication features: LastContactDay, LastContactMonth, CallStart, CallEnd, Communication, NoOfContacts, DaysPassed

Previous campaign features: PrevAttempts, Outcome

In [13]:

```
# First simplify some client features

# Create age group based on age bands
all_df['AgeBand']=pd.cut(all_df['Age'],5)
print(all_df['AgeBand'].value_counts())
```

```
(33.4, 48.8]      2184
(17.923, 33.4]    1508
(48.8, 64.2]      1147
(64.2, 79.6]       133
(79.6, 95.0]        27
Name: AgeBand, dtype: int64
```

In [14]:

```
all_df.loc[(all_df['Age']>=17) & (all_df['Age']<34),'AgeBin'] = 1
all_df.loc[(all_df['Age']>=34) & (all_df['Age']<49),'AgeBin'] = 2
all_df.loc[(all_df['Age']>=49) & (all_df['Age']<65),'AgeBin'] = 3
all_df.loc[(all_df['Age']>=65) & (all_df['Age']<80),'AgeBin'] = 4
all_df.loc[(all_df['Age']>=80) & (all_df['Age']<96),'AgeBin'] = 5
```

```
all_df['AgeBin'] = all_df['AgeBin'].astype(int)
```

In [15]:

```
# Create balance groups
all_df['BalanceBand']=pd.cut(all_df['Balance'],5)
print(all_df['BalanceBand'].value_counts())
all_df.loc[(all_df['Balance']>=-3200) & (all_df['Balance']<17237),'BalanceBin'] = 1
all_df.loc[(all_df['Balance']>=17237) & (all_df['Balance']<37532),'BalanceBin'] = 2
all_df.loc[(all_df['Balance']>=37532) & (all_df['Balance']<57827),'BalanceBin'] = 3
all_df.loc[(all_df['Balance']>=57827) & (all_df['Balance']<78122),'BalanceBin'] = 4
all_df.loc[(all_df['Balance']>=78122) & (all_df['Balance']<98418),'BalanceBin'] = 5
all_df['BalanceBin'] = all_df['BalanceBin'].astype(int)
```

```
(-3113.645, 8071.0]    4847
(8071.0, 19200.0]       123
(19200.0, 30329.0]       20
(30329.0, 41458.0]        5
(41458.0, 52587.0]        4
Name: BalanceBand, dtype: int64
```

In [16]:

```
all_df = all_df.drop(['AgeBand','BalanceBand','Age','Balance'],axis=1)
```

In [17]:

```
#  Convert education level to numeric
all_df['Education'] = all_df['Education'].replace({'None':0,'primary':1,'secondary':2,'tertiary':3})
```

In [18]:

```
# Next create some new communication Features. This is the place feature engineering coming into play

# Get call length
all_df['CallEnd'] = pd.to_datetime(all_df['CallEnd'])
all_df['CallStart'] = pd.to_datetime(all_df['CallStart'])
all_df['CallLength'] = ((all_df['CallEnd'] - all_df['CallStart'])/np.timedelta64(1,'m')).astype(float)
```

In [19]:

```
all_df['CallLenBand']=pd.cut(all_df['CallLength'],5)
print(all_df['CallLenBand'].value_counts())

# Create call length bins
all_df.loc[(all_df['CallLength']>= 0) & (all_df['CallLength']<11),'CallLengthBin'] = 1
all_df.loc[(all_df['CallLength']>=11) & (all_df['CallLength']<22),'CallLengthBin'] = 2
all_df.loc[(all_df['CallLength']>=22) & (all_df['CallLength']<33),'CallLengthBin'] = 3
all_df.loc[(all_df['CallLength']>=33) & (all_df['CallLength']<44),'CallLengthBin'] = 4
all_df.loc[(all_df['CallLength']>=44) & (all_df['CallLength']<55),'CallLengthBin'] = 5
all_df['CallLengthBin'] = all_df['CallLengthBin'].astype(int)
all_df = all_df.drop('CallLenBand',axis=1)
```

```
(0.0292, 10.91]     4274
(10.91, 21.737]      601
(21.737, 32.563]     104
(32.563, 43.39]       15
(43.39, 54.217]        5
Name: CallLenBand, dtype: int64
```

In [20]:

```
# Get call start hour
all_df['CallStartHour'] = all_df['CallStart'].dt.hour
all_df[['CallStart','CallEnd','CallLength','CallStartHour']].head()
```

Out[20]:

		CallStart	CallEnd	CallLength	CallStartHour
train	0	2018-12-20 13:45:20	2018-12-20 13:46:30	1.166667	13
	1	2018-12-20 14:49:03	2018-12-20 14:52:08	3.083333	14
	2	2018-12-20 16:30:24	2018-12-20 16:36:04	5.666667	16
	3	2018-12-20 12:06:43	2018-12-20 12:20:22	13.650000	12
	4	2018-12-20 14:35:44	2018-12-20 14:38:56	3.200000	14

In [21]:

```
# Get workday of last contact based on call day and month, assuming the
 year is 2016
all_df['LastContactDate'] = all_df.apply(lambda x:datetime.datetime.st
rptime("%s %s %s" %(2018,x['LastContactMonth'],x['LastContactDay']),"%
Y %b %d"),axis=1)
all_df['LastContactWkd'] = all_df['LastContactDate'].dt.weekday
all_df['LastContactWkd'].value_counts()
all_df['LastContactMon'] = all_df['LastContactDate'].dt.month
all_df = all_df.drop('LastContactMonth',axis=1)
```

In [22]:

```
# Get week of last contact
all_df['LastContactWk'] = all_df['LastContactDate'].dt.week
MonWk = all_df.groupby(['LastContactWk','LastContactMon'])['Education'
].count().reset_index()
```

In [23]:

```
MonWk = MonWk.drop('Education',axis=1)
MonWk['LastContactWkNum']=0
for m in range(1,13):
    k=0
    for i,row in MonWk.iterrows():
        if row['LastContactMon']== m:
            k=k+1
            row['LastContactWkNum']=k
```

In [24]:

```
def get_num_of_week(df):
    for i,row in MonWk.iterrows():
        if (df['LastContactWk']== row['LastContactWk']) & (df['LastCon
tactMon']== row['LastContactMon']):
            return row['LastContactWkNum']

all_df['LastContactWkNum'] = all_df.apply(lambda x: get_num_of_week(x
),axis=1)
all_df[['LastContactWkNum','LastContactWk','LastContactMon']].head(10)
```

Out[24]:

		LastContactWkNum	LastContactWk	LastContactMon
train	0	4	4	1
	1	4	21	5
	2	1	22	6
	3	2	19	5
	4	1	22	6
	5	4	21	5
	6	3	11	3
	7	2	19	5
	8	3	46	11
	9	2	19	5

In [25]:

```
# Spilt numeric and categorical features
cat_feats = all_df.select_dtypes(include=['object']).columns
num_feats = all_df.select_dtypes(include=['float64','int64']).columns
num_df = all_df[num_feats]
cat_df = all_df[cat_feats]
print('There are %d numeric features and %d categorical features\n' %(
len(num_feats),len(cat_feats)))
print('Numeric features:\n',num_feats.values)
print('Categorical features:\n',cat_feats.values)
```

```
There are 17 numeric features and 4 categorical features

Numeric features:
 ['Education' 'Default' 'HHInsurance' 'CarLoan' 'LastContactDay'
 'NoOfContacts' 'DaysPassed' 'PrevAttempts' 'AgeBin' 'BalanceBin'
 'CallLength' 'CallLengthBin' 'CallStartHour' 'LastContactWkd'
 'LastContactMon' 'LastContactWk' 'LastContactWkNum']
Categorical features:
 ['Job' 'Marital' 'Communication' 'Outcome']
```

In [26]:

```
cat_df = pd.get_dummies(cat_df)
all_data = pd.concat([num_df,cat_df],axis=1)
```

In [27]:

```
# Split train and test
idx=pd.IndexSlice
train_df=all_data.loc[idx[['train',],:]]
test_df=all_data.loc[idx[['test',],:]]
train_label=train['CarInsurance']
print(train_df.shape)
print(len(train_label))
print(test_df.shape)
```

```
(3999, 39)
3999
(1000, 39)
```

In [28]:

```
# Train test split
x_train, x_test, y_train, y_test = train_test_split(train_df,train_lab
el,test_size = 0.005,random_state=3)
```

In [29]:

```
x_train.shape
```

Out[29]:

```
(3979, 39)
```

Modeling

In [30]:

```
# Create a cross validation function
def get_best_model(estimator, params_grid={}):

    model = GridSearchCV(estimator = estimator,param_grid = params_gri
d,cv=3, scoring="accuracy", n_jobs= -1)
    model.fit(x_train,y_train)
    print('\n--- Best Parameters -----------------------------')
    print(model.best_params_)
    print('\n--- Best Model -----------------------------')
    best_model = model.best_estimator_
    print(best_model)
    return best_model
```

In [31]:

```
# Create a model fitting function
def model_fit(model,feature_imp=True,cv=5):

    # model fit
    clf = model.fit(x_train,y_train)

    # model prediction
    y_pred = clf.predict(x_test)

    # model report
    cm = confusion_matrix(y_test,y_pred)
    plot_confusion_matrix(cm, classes=class_names, title='Confusion ma
trix')

    print('\n--- Train Set -----------------------------')
    print('Accuracy: %.5f +/- %.4f' % (np.mean(cross_val_score(clf,x_t
rain,y_train,cv=cv)),np.std(cross_val_score(clf,x_train,y_train,cv=cv
))))
    print('AUC: %.5f +/- %.4f' % (np.mean(cross_val_score(clf,x_train,
y_train,cv=cv,scoring='roc_auc')),np.std(cross_val_score(clf,x_train,y
_train,cv=cv,scoring='roc_auc'))))
    print('\n--- Validation Set -----------------------------')
    print('Accuracy: %.5f +/- %.4f' % (np.mean(cross_val_score(clf,x_t
est,y_test,cv=cv)),np.std(cross_val_score(clf,x_test,y_test,cv=cv))))
    print('AUC: %.5f +/- %.4f' % (np.mean(cross_val_score(clf,x_test,y
_test,cv=cv,scoring='roc_auc')),np.std(cross_val_score(clf,x_test,y_te
st,cv=cv,scoring='roc_auc'))))
    print('-----------------------------------------------')

    # feature importance
    if feature_imp:
        feat_imp = pd.Series(clf.feature_importances_,index=all_data.c
olumns)
        feat_imp = feat_imp.nlargest(15).sort_values()
        plt.figure()
        feat_imp.plot(kind="barh",figsize=(6,8),title="Most Important
```

```
 Features")
```

In [32]:

```
# The confusion matrix plotting function is from the sklearn documentati
on below:
# http://scikit-learn.org/stable/auto_examples/model_selection/plot_conf
usion_matrix.html
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    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)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    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 else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

class_names = ['Success','Failure']
```

k-Nearest Neighbors (KNN)

In [33]:

```
# Let's start with KNN. An accuracy of 0.76 is not very impressive. I wi
ll just take this as the model benchmark.
knn = KNeighborsClassifier()
parameters = {'n_neighbors':[5,6,7],
              'p':[1,2],
              'weights':['uniform','distance']}
clf_knn = get_best_model(knn,parameters)
model_fit(model=clf_knn, feature_imp=False)
```

```
----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
<ipython-input-33-7c32227b23a0> in <module>()
      4               'p':[1,2],
      5               'weights':['uniform','distance']}
----> 6 clf_knn = get_best_model(knn,parameters)
```

```
      7 model_fit(model=clf_knn, feature_imp=False)

<ipython-input-30-66c017a1c7f8> in get_best_model(estimator, params_grid)
      2 def get_best_model(estimator, params_grid={}):
      3
----> 4     model = GridSearchCV(estimator = estimator,param_grid = params_grid,cv=3, scoring="accuracy", n_jobs= -1)
      5     model.fit(x_train,y_train)
      6     print('\n--- Best Parameters -----------------------------')

NameError: name 'GridSearchCV' is not defined
```

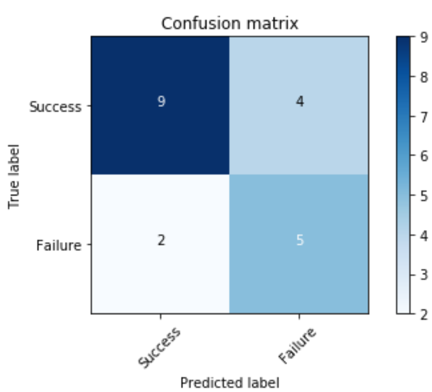
Naive Bayes Classifier

In [34]:

```
# As expected, Naive Bayes classifier doesn't perform well here.
# There are multiple reasons. Some of the numeric features are not normally distributed, which is a strong assemption hold by Naive Bayes.
# Also, features are definitely not independent.
clf_nb = GaussianNB()
model_fit(model=clf_nb,feature_imp=False)
```

```
--- Train Set -----------------------------
Accuracy: 0.75297 +/- 0.0149
AUC: 0.81061 +/- 0.0097

--- Validation Set -----------------------------
Accuracy: 0.55333 +/- 0.2696
AUC: 0.45000 +/- 0.3712
-----------------------------------------------
```



Logistic Regression

In [35]:

```
# We're making progress here. Logistic regression performs better than KNN.
lg = LogisticRegression(random_state=3)
parameters = {'C':[0.8,0.9,1],
              'penalty':['l1','l2']}
clf_lg = get_best_model(lg,parameters)
model_fit(model=clf_lg, feature_imp=False)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-35-04c7d843c9d6> in <module>()
      3 parameters = {'C':[0.8,0.9,1],
      4               'penalty':['l1','l2']}
----> 5 clf_lg = get_best_model(lg,parameters)
      6 model_fit(model=clf_lg, feature_imp=False)

<ipython-input-30-66c017a1c7f8> in get_best_model(estimator, params_grid)
      2 def get_best_model(estimator, params_grid={}):
      3 
----> 4     model = GridSearchCV(estimator = estimator,param_grid = params_grid,cv=3, scoring="accuracy", n_jobs= -1)
      5     model.fit(x_train,y_train)
      6     print('\n--- Best Parameters -----------------------------')

NameError: name 'GridSearchCV' is not defined
```

Random Forest

In [36]:

```
# I did some manual parameter tuning here. This is the best model so far.
# Based on the feature importance report, call length, last contact week, and previous success are strong predictors of cold call success
rf = RandomForestClassifier(random_state=3)
parameters={'n_estimators':[100],
            'max_depth':[10],
            'max_features':[13,14],
            'min_samples_split':[11]}
clf_rf= get_best_model(rf,parameters)
model_fit(model=clf_rf, feature_imp=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-36-dcfb22d67508> in <module>()
      6             'max_features':[13,14],
      7             'min_samples_split':[11]}
----> 8 clf_rf= get_best_model(rf,parameters)
      9 model_fit(model=clf_rf, feature_imp=True)

<ipython-input-30-66c017a1c7f8> in get_best_model(estimator, params_grid)
      2 def get_best_model(estimator, params_grid={}):
      3 
----> 4     model = GridSearchCV(estimator = estimator,param_grid = params_grid,cv=3, scoring="accuracy", n_jobs= -1)
      5     model.fit(x_train,y_train)
      6     print('\n--- Best Parameters -----------------------------')

NameError: name 'GridSearchCV' is not defined
```

Support Vector Machines

In [37]:

```
# try a SVM RBF model
svc = svm.SVC(kernel='rbf', probability=True, random_state=3)
parameters = {'gamma': [0.005,0.01,0.02],
              'C': [0.5,1,5]}
clf_svc = get_best_model(svc, parameters)
model_fit(model=clf_svc,feature_imp=False)
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
<ipython-input-37-2d804e425a8a> in <module>()
      3 parameters = {'gamma': [0.005,0.01,0.02],
      4               'C': [0.5,1,5]}
----> 5 clf_svc = get_best_model(svc, parameters)
      6 model_fit(model=clf_svc,feature_imp=False)

<ipython-input-30-66c017a1c7f8> in get_best_model(estimator, params_gr
id)
      2 def get_best_model(estimator, params_grid={}):
      3
----> 4     model = GridSearchCV(estimator = estimator,param_grid = pa
rams_grid,cv=3, scoring="accuracy", n_jobs= -1)
      5     model.fit(x_train,y_train)
      6     print('\n--- Best Parameters -----------------------------
')

NameError: name 'GridSearchCV' is not defined
```

XGBoost

In [38]:

```
# Finally let's try out XBGoost. As expected, it outperforms all other a
lgorithms.
# Also, based on feature importances, some of the newly created features
such as call start hour, last contact week and weekday
# have been picked as top features.

import xgboost as xgb
xgb = xgb.XGBClassifier()
parameters={'n_estimators':[900,1000,1100],
            'learning_rate':[0.01],
            'max_depth':[8],
            'min_child_weight':[1],
            'subsample':[0.8],
            'colsample_bytree':[0.3,0.4,0.5]}
clf_xgb= get_best_model(xgb,parameters)
model_fit(model=clf_xgb, feature_imp=True)
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
```

```
<ipython-input-38-d6ccb2ea9a90> in <module>()
     11             'subsample':[0.8],
     12             'colsample_bytree':[0.3,0.4,0.5]}
---> 13 clf_xgb= get_best_model(xgb,parameters)
     14 model_fit(model=clf_xgb, feature_imp=True)

<ipython-input-30-66c017a1c7f8> in get_best_model(estimator, params_gr
id)
      2 def get_best_model(estimator, params_grid={}):
      3
----> 4     model = GridSearchCV(estimator = estimator,param_grid = pa
rams_grid,cv=3, scoring="accuracy", n_jobs= -1)
      5     model.fit(x_train,y_train)
      6     print('\n--- Best Parameters -----------------------------
')

NameError: name 'GridSearchCV' is not defined
```

Model Evaluation¶

In [39]:

```
# Compare model performance
clfs= [clf_knn, clf_nb, clf_lg, clf_rf, clf_svc, clf_xgb]
index =['K-Nearest Neighbors','Naive Bayes','Logistic Regression','Ran
dom Forest','Support Vector Machines','XGBoost']
scores=[]
for clf in clfs:
    score = np.mean(cross_val_score(clf,x_test,y_test,cv=5,scoring =
'accuracy'))
    scores = np.append(scores,score)
models = pd.Series(scores,index=index)
models.sort_values(ascending=False)
```

```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
<ipython-input-39-b8c152d6cf76> in <module>()
      1 # Compare model performance
----> 2 clfs= [clf_knn, clf_nb, clf_lg, clf_rf, clf_svc, clf_xgb]
      3 index =['K-Nearest Neighbors','Naive Bayes','Logistic Regressi
on','Random Forest','Support Vector Machines','XGBoost']
      4 scores=[]
      5 for clf in clfs:

NameError: name 'clf_knn' is not defined
```

Ensemble Voting

In [40]:

```
#XGBoost and Random Forest show different important features, implying t
hat those models are capturing different aspects of the data
# To get the final model, I ensembled different classifiers based on maj
```

```
ority voting.
# XGBoost and Random Forest are given larger weights due to their better
performance.

clf_vc = VotingClassifier(estimators=[('xgb', clf_xgb),
                                      ('rf', clf_rf),
                                      ('lg', clf_lg),
                                      ('svc', clf_svc)],
                          voting='hard',
                          weights=[4,4,1,1])
clf_vc = clf_vc.fit(x_train, y_train)
```

```
----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
<ipython-input-40-8cc60d60934e> in <module>()
      3 # XGBoost and Random Forest are given larger weights due to th
eir better performance.
      4
----> 5 clf_vc = VotingClassifier(estimators=[('xgb', clf_xgb),
      6                                       ('rf', clf_rf),
```

Cold Call Haythem notebook

Python notebook using data from Car Insurance Cold Calls · 85 views · 10mo ago

0 Copy and Edit 1 ...

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

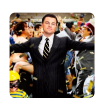
0

Version 3
3 commits

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 (0)

Please sign in to leave a comment.

Notebook Data Comments

Our Team Terms Privacy Contact/Support