

# bank\_customer\_churn\_modeling

January 28, 2019

## 1 Data Science

### 1.1 Bank Customer Churn Modeling

nhan.cao@beesightsoft.com

```
In [0]: # Basic Libraries
import numpy as np
import pandas as pd
import operator
import re
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter("ignore")

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Preprocessing
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.pipeline import _name_estimators
from sklearn.base import BaseEstimator
from sklearn.base import ClassifierMixin
from sklearn.base import clone
from sklearn.externals import six

# Evaluation
from sklearn import metrics
from sklearn import linear_model, datasets
from sklearn.metrics import accuracy_score, log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import LocalOutlierFactor
```

```

# Classifier (machine learning algorithm)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier,
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.neural_network import MLPClassifier

from sklearn.base import BaseEstimator
from sklearn.base import ClassifierMixin
from sklearn.externals import six
from sklearn.base import clone
from sklearn.pipeline import _name_estimators

```

## 2 Read data

<https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling>

```

In [2]: from google.colab import drive
        drive.mount('/content/gdrive')
        dataset = pd.read_csv("gdrive/My Drive/Colab Notebooks/Churn_Modelling.csv", header = 0)

```

Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive")

```

In [0]: # dataset = pd.read_csv('../input/Churn_Modelling.csv', header = 0)

```

```

In [4]: # Tmp data
        dataset_tmp = dataset.copy()
        dataset_tmp.head()

```

```

Out[4]:
   RowNumber  CustomerId  Surname  CreditScore  Geography  Gender  Age  \
0          1    15634602  Hargrave         619     France  Female  42
1          2    15647311    Hill         608      Spain  Female  41
2          3    15619304    Onio         502     France  Female  42
3          4    15701354    Boni         699     France  Female  39
4          5    15737888  Mitchell         850      Spain  Female  43

   Tenure  Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0        2      0.00              1          1              1
1        1  83807.86              1          0              1
2        8 159660.80              3          1              0

```

3	1	0.00	2	0	0
4	2	125510.82	1	1	1

	EstimatedSalary	Exited
0	101348.88	1
1	112542.58	0
2	113931.57	1
3	93826.63	0
4	79084.10	0

### 3 Functions

```
In [0]: class MajorityVoteClassifier(BaseEstimator, ClassifierMixin):
        """ A majority vote ensemble classifier
        Parameters
        classifiers : array-like, shape = [n_classifiers] Different classifiers for the ensemble
        vote : str, {'classlabel', 'probability'} (default='label')
            If 'classlabel' the prediction is based on the argmax of class labels. Else if 'probability'
        weights : array-like, shape = [n_classifiers], optional (default=None)
            If a list of `int` or `float` values are provided, the classifiers are weighted
        """
        def __init__(self, classifiers, vote='classlabel', weights=None):
            self.classifiers = classifiers
            self.named_classifiers = {key: value for key, value in _name_estimators(classifiers)}
            self.vote = vote
            self.weights = weights
        def fit(self, X, y):
            """ Fit classifiers. Parameters
            X : {array-like, sparse matrix}, shape = [n_samples, n_features] Matrix of training data
            y : array-like, shape = [n_samples] Vector of target class labels.
            Returns self : object
            """
            if self.vote not in ('probability', 'classlabel'):
                raise ValueError("vote must be 'probability' or 'classlabel' "; got (vote))
            if self.weights and len(self.weights) != len(self.classifiers):
                raise ValueError('Number of classifiers and weights must be equal'; got %s
                                  (len(self.weights), len(self.classifiers)))
            # Use LabelEncoder to ensure class labels start with 0, which is important for
            self.lablenc_ = LabelEncoder()
            self.lablenc_.fit(y)
            self.classes_ = self.lablenc_.classes_
            self.classifiers_ = []
            for clf in self.classifiers:
                fitted_clf = clone(clf).fit(X, self.lablenc_.transform(y))
                self.classifiers_.append(fitted_clf)
            return self
        def predict(self, X):
```

```

        """ Predict class labels for X.
        Parameters
        -----
        X : {array-like, sparse matrix}, shape = [n_samples, n_features] Matrix of tra
        Returns -----
        maj_vote : array-like, shape = [n_samples] Predicted class labels.
        """
        if self.vote == 'probability':
            maj_vote = np.argmax(self.predict_proba(X), axis=1)
        else: # 'classlabel' vote
            # Collect results from clf.predict calls
            predictions = np.asarray([clf.predict(X) for clf in self.classifiers_]).T
            maj_vote = np.apply_along_axis(lambda x: np.argmax(np.bincount(x, weights=
                                     axis=1,
                                     arr=predictions)
            maj_vote = self.labelenc_.inverse_transform(maj_vote)
        return maj_vote
    def predict_proba(self, X):
        """ Predict class probabilities for X.
        X : {array-like, sparse matrix}, shape = [n_samples, n_features]
            Training vectors, where n_samples is the number of samples and n_features
        Returns
        avg_proba : array-like, shape = [n_samples, n_classes] Weighted average probab
        """
        probas = np.asarray([clf.predict_proba(X) for clf in self.classifiers_])
        avg_proba = np.average(probas, axis=0, weights=self.weights)
        return avg_proba
    def get_params(self, deep=True):
        """ Get classifier parameter names for GridSearch"""
        if not deep:
            return super(MajorityVoteClassifier, self).get_params(deep=False)
        else:
            out = self.named_classifiers.copy()
            for name, step in six.iteritems(self.named_classifiers):
                for key, value in six.iteritems(step.get_params(deep=True)):
                    out['%s_%s' % (name, key)] = value
            return out

# Split Train and Test and check shape
def SplitDataFrameToTrainAndTest(DataFrame, TrainDataRate, TargetAtt):
    # gets a random TrainDataRate % of the entire set
    training = DataFrame.sample(frac=TrainDataRate, random_state=1)
    # gets the left out portion of the dataset
    testing = DataFrame.loc[~DataFrame.index.isin(training.index)]

    X_train = training.drop(TargetAtt, 1)
    y_train = training[[TargetAtt]]
    X_test = testing.drop(TargetAtt, 1)

```

```

y_test = testing[[TargetAtt]]
return X_train, y_train, X_test, y_test

def PrintTrainTestInformation(X_train, y_train, X_test, y_test):
    print("Train rows and columns : ", X_train.shape)
    print("Test rows and columns : ", X_test.shape)

def DrawJointPlot(DataFrame, XAtt, yAtt, bins = 20):
    sns.set(color_codes=True)
    sns.distplot(data[XAtt], bins=bins);
    df = pd.DataFrame(DataFrame, columns=[XAtt,yAtt])
    df = df.reset_index(drop=True)
    sns.jointplot(x=XAtt, y=yAtt, data=df)

def DrawBoxplot2(DataFrame, xAtt, yAtt, hAtt="N/A"):
    plt.figure()
    if(hAtt == "N/A"):
        sns.boxplot(x=xAtt, y=yAtt, data=DataFrame)
    else:
        sns.boxplot(x=xAtt, y=yAtt, hue=hAtt, data=DataFrame)
    plt.show()

def DrawBarplot(DataFrame, att):
    Distribution = DataFrame[att].value_counts()
    Distribution = pd.DataFrame({att:Distribution.index, 'Freq':Distribution.values})
    Distribution = Distribution.sort_values(by=att, ascending=True)
    plt.bar(Distribution[att], Distribution["Freq"])
    plt.xticks(Distribution[att])
    plt.ylabel('Frequency')
    plt.title('Barplot of ' + att)
    plt.show()

def DrawCountplot(DataFrame, att, hatt="N/A"):
    if(hatt == "N/A"):
        sns.countplot(x=att, data=DataFrame)
    else:
        sns.countplot(x=att, hue=hatt, data=DataFrame)
    plt.show()

def DrawHistogram(DataFrame, att):
    plt.figure()
    DataFrame[att].hist(edgecolor='black', bins=20)
    plt.title(att)
    plt.show()

# Detect outlier in each feature
def DetectOutlierByIQR(DataFrame, AttList, Rate = 3.0):
    OutlierIdx = []

```

```

for att in AttList:
    AttData = DataFrame.loc[:, att]
    lowerq = AttData.quantile(0.25)
    upperq = AttData.quantile(0.75)
    IQR = upperq - lowerq
    threshold_upper = (IQR * Rate) + upperq
    threshold_lower = lowerq - (IQR * Rate)
    AttOutlierIdx = set(AttData[AttData.apply(lambda x: x > threshold_upper
                                                or x < threshold_lower)].index.get

    OutlierIdx = set(OutlierIdx) | AttOutlierIdx
    # print("Min, Max and IQR : %f, %f, and %f" % (AttData.min(), AttData.max(), I
    # print("Upper Fence and Lower Fence : %f and %f" % (threshold_lower, threshol
    # print("OutlierIdx : " + str(OutlierIdx))
    # print(att + " " + str(len(AttOutlierIdx)) + " Outlier Idx : " + str(AttOutl

OutlierIdx = list(OutlierIdx)
OutlierIdx = sorted(OutlierIdx)
return OutlierIdx

# Detect outlier in group features
def DetectOutlierByLOF(DataFrame, AttList, LOFThresh=3.0, neighbors = 10):
    clf = LocalOutlierFactor(n_neighbors=neighbors)
    AttData = DataFrame.loc[:, AttList].values
    y_pred = clf.fit_predict(AttData)
    AttData_scores = -1 * clf.negative_outlier_factor_
    LOFFactorData = pd.DataFrame(AttData_scores, columns=['LOF'])
    LOFFactorData = LOFFactorData.sort_values('LOF', ascending=False)
    LOFFactorData = LOFFactorData.reset_index(drop=False)
    # print(LOFFactorData.loc[0:10, :])
    OutlierThreshold = LOFThresh
    SuspectOutlierData = LOFFactorData[LOFFactorData['LOF'].apply(lambda x: x > OutlierTh
    OutlierIdx = SuspectOutlierData.loc[:, 'index'].tolist()
    # print("OutlierIdx : " + str(OutlierIdx))
    return OutlierIdx, LOFFactorData

def RemoveRowsFromDataFrame(DataFrame, RowIdxList = []):
    DataFrame = DataFrame.drop(RowIdxList)
    DataFrame = DataFrame.reset_index(drop=True)
    return DataFrame

def NaiveBayesLearning(DataTrain, TargetTrain):
    NBModel = GaussianNB()
    NBModel.fit(DataTrain, TargetTrain.values.ravel())
    return NBModel

def NaiveBayesTesting(NBModel, DataTest, TargetTest):
    PredictTest = NBModel.predict(DataTest)
    Accuracy = accuracy_score(TargetTest, PredictTest)

```

```

    return Accuracy, PredictTest

def LogisticRegressionLearning(DataTrain, TargetTrain):
    logreg = LogisticRegression()
    # Training by Logistic Regression
    logreg.fit(DataTrain, TargetTrain.values.ravel())
    return logreg

def LogisticRegressionTesting(LRModel, DataTest, TargetTest):
    logreg = LRModel
    PredictTest = logreg.predict(DataTest)
    Accuracy = accuracy_score(TargetTest, PredictTest)
    # print('Logistic regression accuracy: {:.3f}'.format(Accuracy))
    return Accuracy, PredictTest

def RandomForestLearning(DataTrain, TargetTrain):
    rf = RandomForestClassifier()
    rf.fit(DataTrain, TargetTrain.values.ravel())
    return rf

def RandomForestTesting(RFModel, DataTest, TargetTest):
    PredictTest = RFModel.predict(DataTest)
    Accuracy = accuracy_score(TargetTest, PredictTest)
    # print('Random Forest Accuracy: {:.3f}'.format(accuracy_score(TargetTest, PredictTest)))
    return Accuracy, PredictTest

def SVMLearning(DataTrain, TargetTrain, ClassifierType = " "):
    if(ClassifierType == 'Linear'):
        svc = SVC(kernel="linear", C=0.025)
        # print('SVM Linear processing')
        # Radial basis function kernel
    elif (ClassifierType == 'RBF'):
        svc = SVC(gamma=2, C=1)
        # print('SVM RBF processing')
    else:
        svc = SVC()
        # print('SVM Default processing')
    svc.fit(DataTrain, TargetTrain.values.ravel())
    return svc

def SVMTesting(SVMModel, DataTest, TargetTest):
    PredictTest = SVMModel.predict(DataTest)
    Accuracy = accuracy_score(TargetTest, PredictTest)
    # print('Support Vector Machine Accuracy: {:.3f}'.format(accuracy_score(TargetTest, PredictTest)))
    return Accuracy, PredictTest

def KNNLearning(DataTrain, TargetTrain, K = 3):
    neigh = KNeighborsClassifier(n_neighbors=K)

```

```

    neigh.fit(DataTrain, TargetTrain.values.ravel())
    return neigh

def KNNTesting(KNNModel, DataTest, TargetTest):
    PredictTest = KNNModel.predict(DataTest)
    Accuracy = accuracy_score(TargetTest, PredictTest)
    # print('KNN Accuracy: {:.3f}'.format(accuracy_score(TargetTest, PredictTest)))
    return Accuracy, PredictTest

def ANNLearning(DataTrain, TargetTrain):
    ANNModel = MLPClassifier(alpha=1)
    ANNModel.fit(DataTrain, TargetTrain.values.ravel())
    return ANNModel

def ANNTesting (ANNModel, DataTest, TargetTest):
    PredictTest = ANNModel.predict(DataTest)
    Accuracy = accuracy_score(TargetTest, PredictTest)
    # print('Neural Net Accuracy: {:.3f}'.format(Accuracy))
    return Accuracy, PredictTest

# Continuous Data Plot
def ContPlot(df, feature_name, target_name, palettemap, hue_order, feature_scale):
    df['Counts'] = "" # A trick to skip using an axis (either x or y) on splitting via
    fig, [axis0,axis1] = plt.subplots(1,2,figsize=(10,5))
    sns.distplot(df[feature_name], ax=axis0);
    sns.violinplot(x=feature_name, y="Counts", hue=target_name, hue_order=hue_order, data=df,
                  palette=palettemap, split=True, orient='h', ax=axis1)
    axis1.set_xticks(feature_scale)
    plt.show()

# Categorical/Ordinal Data Plot
def CatPlot(df, feature_name, target_name, palettemap):
    fig, [axis0,axis1] = plt.subplots(1,2,figsize=(10,5))
    df[feature_name].value_counts().plot.pie(autopct='%1.1f%%',ax=axis0)
    sns.countplot(x=feature_name, hue=target_name, data=df,
                  palette=palettemap,ax=axis1)
    plt.show()

def MachineLearningModelEvaluate(X_train, y_train, X_test, y_test):
    NBModel = NaiveBayesLearning(X_train, y_train)
    NBAccuracy,NBPredictTest = NaiveBayesTesting(NBModel,X_test, y_test)
    print('Naive Bayes accuracy: {:.3f}'.format(NBAccuracy))

    LRModel = LogisticRegressionLearning(X_train, y_train)
    LRAccuracy,LRPredictTest = LogisticRegressionTesting(LRModel,X_test, y_test)
    print('Logistic Regression accuracy: {:.3f}'.format(LRAccuracy))

    RFModel = RandomForestLearning(X_train, y_train)

```



```

RFAccuracy, RFPredictTest = RandomForestTesting(RFModel, X_test, y_test)
print('Random Forest accuracy: {:.6f}'.format(RFAccuracy))

LiSVMModel = SVMLearning(X_train, y_train)
LiSVMAccuracy, LiSVMPredictTest = SVMTesting(LiSVMModel, X_test, y_test)
print('Linear SVM accuracy: {:.6f}'.format(LiSVMAccuracy))

RBFSVMModel = SVMLearning(X_train, y_train, 'RBF')
RBFSVMAccuracy, RBFSVMPredictTest = SVMTesting(RBFSVMModel, X_test, y_test)
print('RBF SVM accuracy: {:.6f}'.format(RBFSVMAccuracy))

KNNModel = KNNLearning(X_train, y_train)
KNNAccuracy, KNNPredictTest = KNNTesting(KNNModel, X_test, y_test)
print('K Nearest Neighbor accuracy: {:.6f}'.format(KNNAccuracy))

ANNModel = ANNLearning(X_train, y_train)
ANNAccuracy, ANNPredictTest = ANNTTesting(ANNModel, X_test, y_test)
print('ANN accuracy: {:.6f}'.format(ANNAccuracy))

```

## 4 Checking missing values

- Fill missing value: Median / Mode, Label Encode / Dummies

```

In [6]: # Checking the percentage of missing values in each variable
        (dataset.isnull().sum()/len(dataset)*100)

```

```

Out [6]: RowNumber      0.0
        CustomerId      0.0
        Surname         0.0
        CreditScore     0.0
        Geography       0.0
        Gender          0.0
        Age             0.0
        Tenure          0.0
        Balance         0.0
        NumOfProducts   0.0
        HasCrCard       0.0
        IsActiveMember  0.0
        EstimatedSalary 0.0
        Exited          0.0
        dtype: float64

```

### 4.1 Preparation and EDA

```

In [7]: # Split Train and Test and check shape
        data_train, target_train, data_test, target_test = SplitDataFrameToTrainAndTest(dataset)
        PrintTrainTestInformation(data_train, target_train, data_test, target_test)

```

```
Train rows and columns : (6000, 13)
Test rows and columns : (4000, 13)
```

```
In [8]: # Check column types
        data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6000 entries, 9953 to 2374
Data columns (total 13 columns):
RowNumber          6000 non-null int64
CustomerId          6000 non-null int64
Surname            6000 non-null object
CreditScore        6000 non-null int64
Geography          6000 non-null object
Gender             6000 non-null object
Age                6000 non-null int64
Tenure             6000 non-null int64
Balance            6000 non-null float64
NumOfProducts      6000 non-null int64
HasCrCard          6000 non-null int64
IsActiveMember     6000 non-null int64
EstimatedSalary    6000 non-null float64
dtypes: float64(2), int64(8), object(3)
memory usage: 656.2+ KB
```

```
In [9]: print(" List of unique values in Surname : ")
        print(dataset['Surname'].unique())
        print(" List of unique values in Geography : ")
        print(dataset['Geography'].unique())
        print(" List of unique values in Gender : ")
        print(dataset['Gender'].unique())

        #Special Field
        print(" List of unique values in NumOfProducts : ")
        print(dataset['NumOfProducts'].unique())
```

```
List of unique values in Surname :
['Hargrave' 'Hill' 'Onio' ... 'Kashiwagi' 'Aldridge' 'Burbidge']
List of unique values in Geography :
['France' 'Spain' 'Germany']
List of unique values in Gender :
['Female' 'Male']
List of unique values in NumOfProducts :
[1 3 2 4]
```

```
In [10]: # Numerical data distribution
         data_train.describe()
```

```
Out[10]:
```

	RowNumber	CustomerId	CreditScore	Age	Tenure \
count	6000.000000	6.000000e+03	6000.000000	6000.000000	6000.000000
mean	5000.534667	1.569090e+07	652.017833	38.801333	5.021667
std	2877.924946	7.201902e+04	96.171969	10.409335	2.888469
min	4.000000	1.556570e+07	350.000000	18.000000	0.000000
25%	2501.250000	1.562812e+07	586.000000	32.000000	3.000000
50%	4995.500000	1.569189e+07	655.000000	37.000000	5.000000
75%	7483.500000	1.575351e+07	718.000000	44.000000	8.000000
max	9999.000000	1.581569e+07	850.000000	92.000000	10.000000

	Balance	NumOfProducts	HasCrCard	IsActiveMember \
count	6000.000000	6000.000000	6000.000000	6000.000000
mean	76069.556590	1.524667	0.702333	0.521833
std	62709.267925	0.576582	0.457270	0.499565
min	0.000000	1.000000	0.000000	0.000000
25%	0.000000	1.000000	0.000000	0.000000
50%	96598.420000	1.000000	1.000000	1.000000
75%	127671.927500	2.000000	1.000000	1.000000
max	238387.560000	4.000000	1.000000	1.000000

	EstimatedSalary
count	6000.000000
mean	99470.172248
std	57622.657250
min	11.580000
25%	50343.395000
50%	99482.980000
75%	149170.417500
max	199992.480000

```
In [11]: data_train.describe(include=['O'])
```

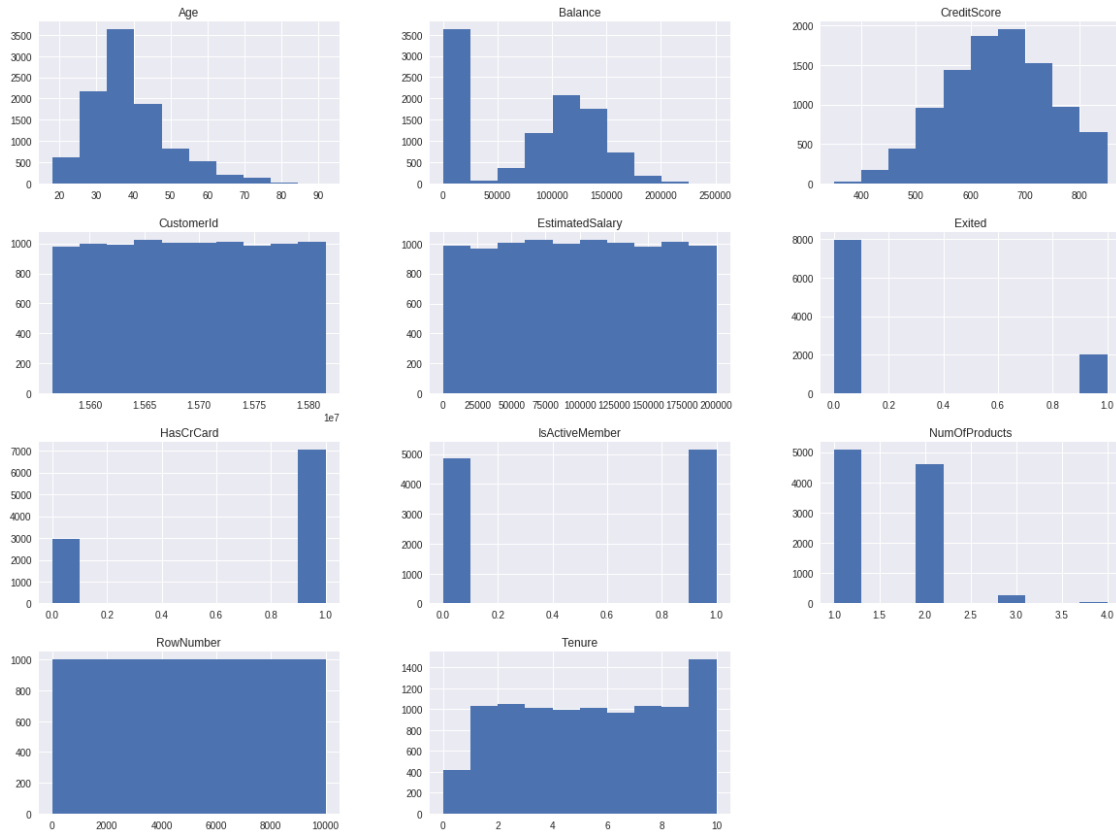
```
Out[11]:
```

	Surname	Geography	Gender
count	6000	6000	6000
unique	2246	3	2
top	Smith	France	Male
freq	23	3026	3259

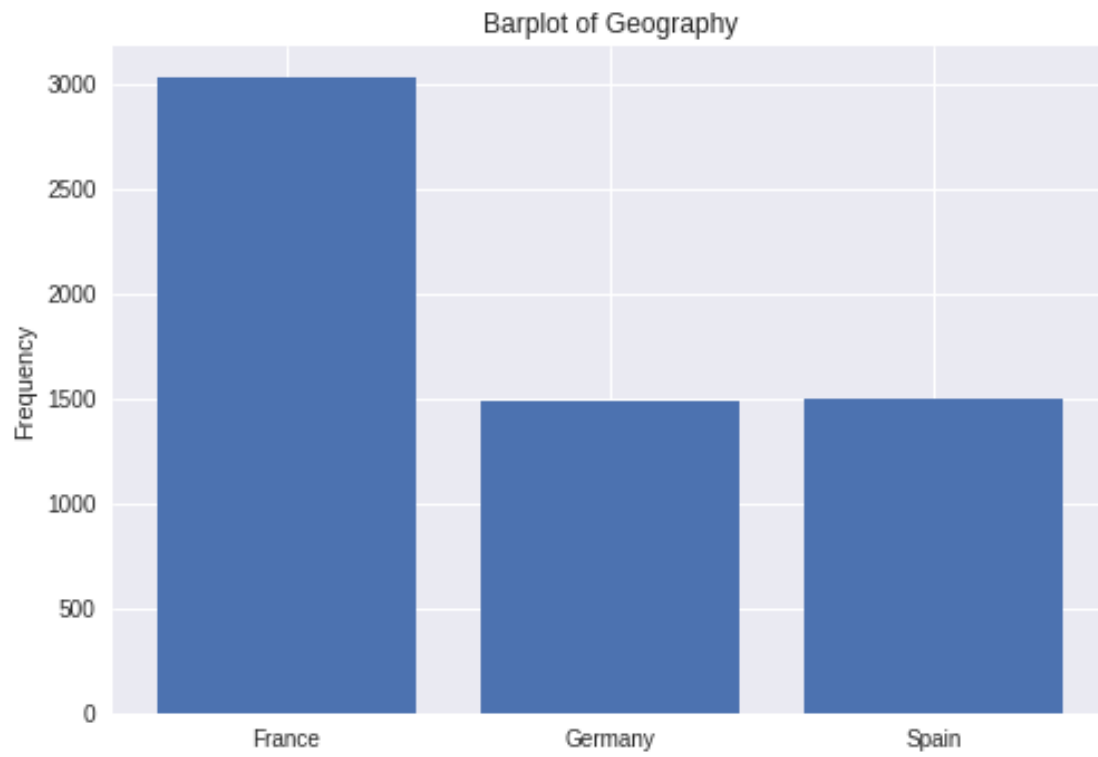
```
In [12]: dataset.hist(bins=10, figsize=(20,15))
```

```
Out[12]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f97fdf0af98>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f97fd490400>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f97fd438a58>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f97fd3ea0f0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f97fd40d748>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f97fd3b4da0>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f97fd364400>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f97fd388a90>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f97fd388ac8>],
```

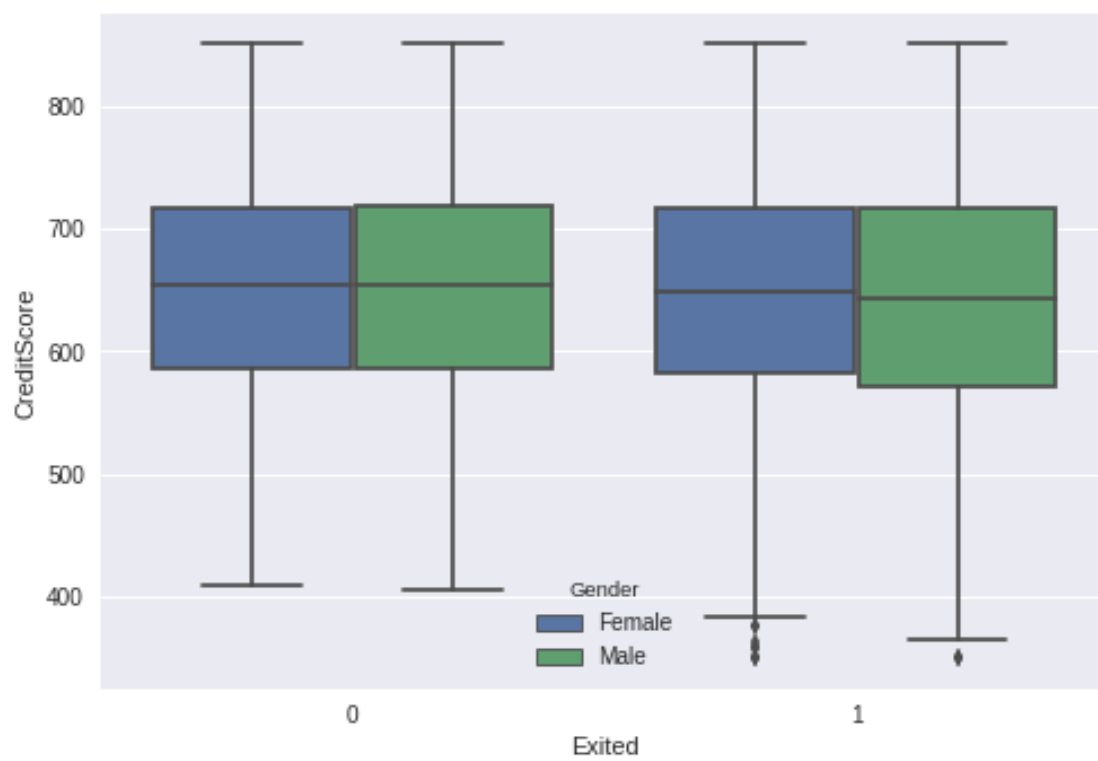
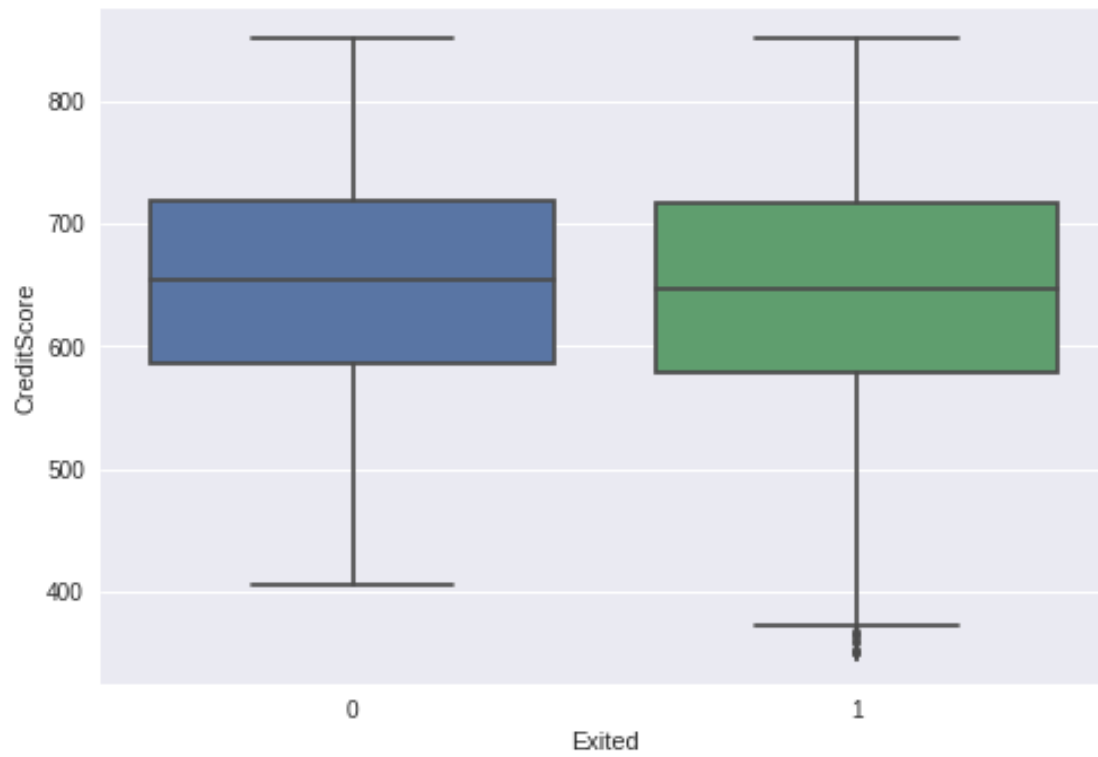
```
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f97fd2de748>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f97fd306da0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f97fd2b5438>]],
 dtype=object)
```



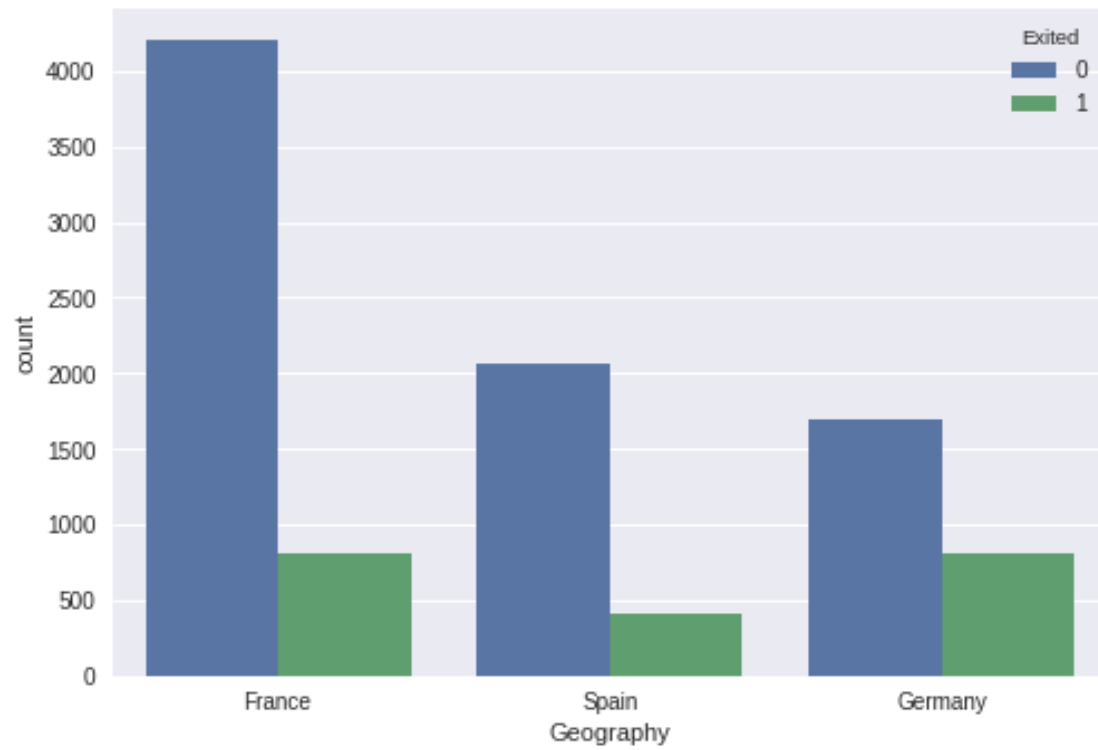
```
In [13]: DrawBarplot(data_train, 'Geography')
```

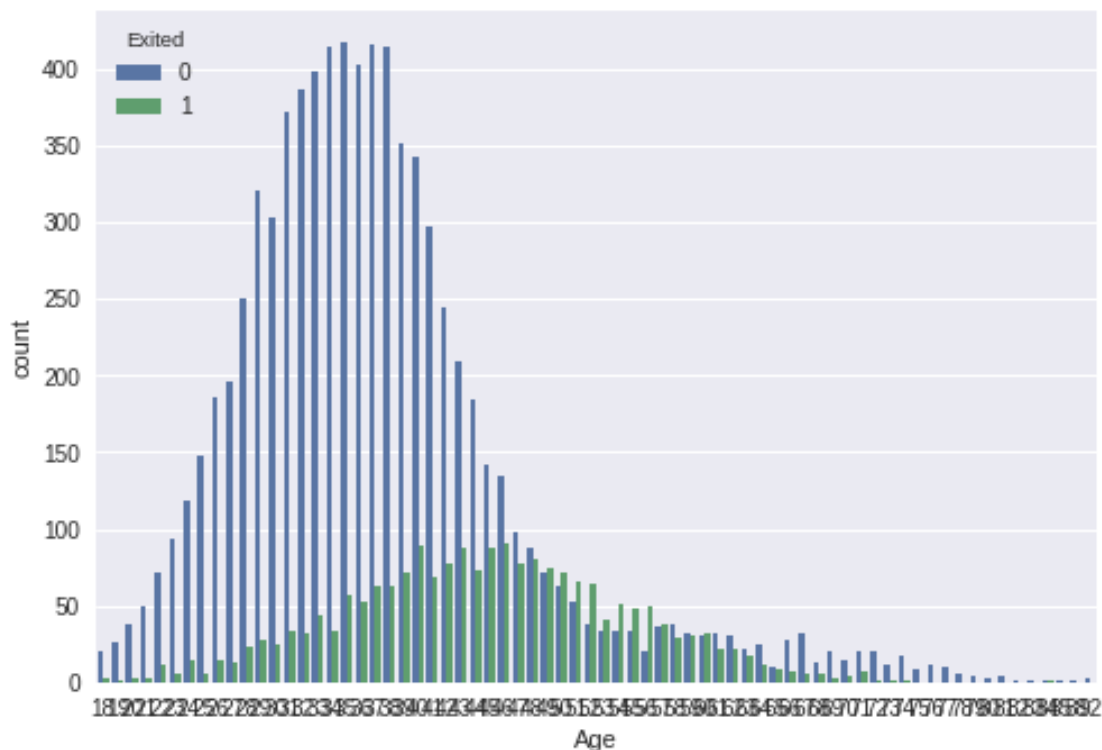


```
In [14]: DrawBoxplot2(dataset, xAtt = 'Exited', yAtt='CreditScore')
         DrawBoxplot2(dataset, xAtt = 'Exited', yAtt='CreditScore', hAtt='Gender')
```



```
In [15]: DrawCountplot(dataset, 'Geography', 'Exited')
         DrawCountplot(dataset, 'Age', 'Exited')
```





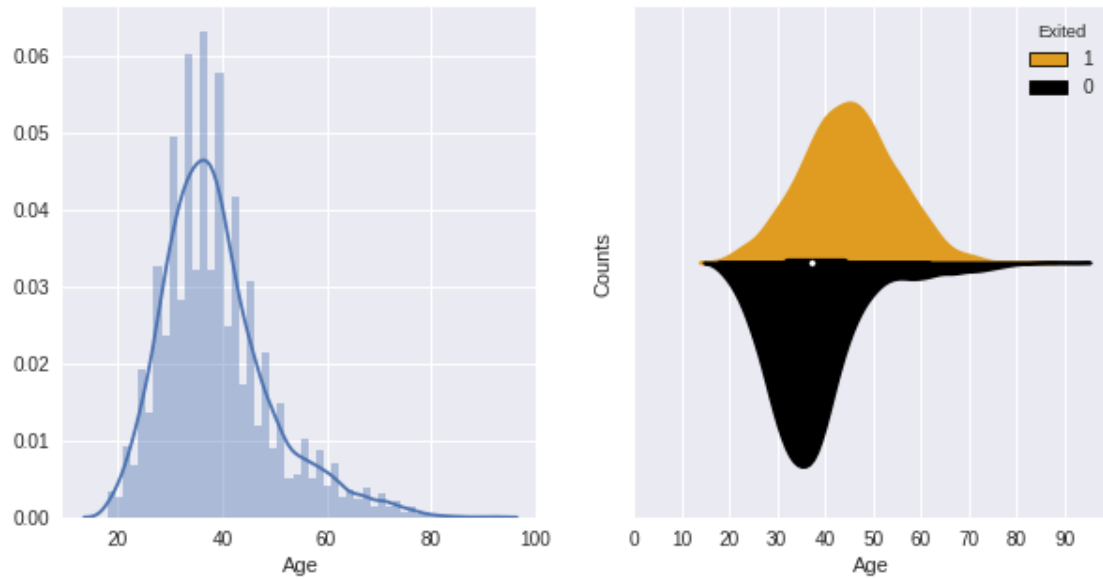
```
In [16]: dataset['CategoricalCreditScore'] = pd.qcut(dataset['CreditScore'], 3)
         print (dataset[['CategoricalCreditScore', 'Exited']].groupby(['CategoricalCreditScore
```

	CategoricalCreditScore	Exited
0	(349.999, 608.0]	0.215284
1	(608.0, 695.0]	0.197660
2	(695.0, 850.0]	0.198002

```
In [17]: ContPlot(dataset[['Age', 'Exited']].copy().dropna(axis=0),
                'Age', 'Exited', {0: "black", 1: "orange"} , [1, 0], range(0,100,10))

dataset['CategoricalAge'] = pd.qcut(dataset['Age'], 5, duplicates='drop')
print (dataset[['CategoricalAge', 'Exited']].groupby(['CategoricalAge'], as_index=False
```

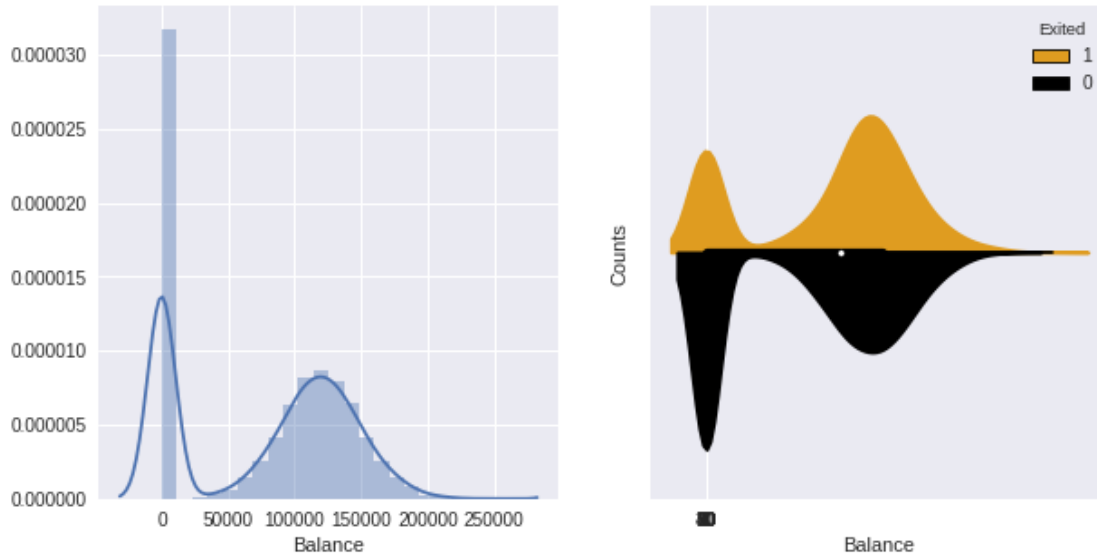




	CategoricalAge	Exited
0	(17.999, 31.0]	0.076307
1	(31.0, 35.0]	0.093206
2	(35.0, 40.0]	0.149603
3	(40.0, 46.0]	0.285967
4	(46.0, 92.0]	0.459416

```
In [18]: ContPlot(dataset[['Balance', 'Exited']].copy().dropna(axis=0),
                'Balance', 'Exited', {0: "black", 1: "orange"} , [1, 0], range(0,100,10))

dataset['CategoricalBalance'] = pd.qcut(dataset['Balance'], 3, duplicates='drop')
print (dataset[['CategoricalBalance', 'Exited']].groupby(['CategoricalBalance'], as_i
```



	CategoricalBalance	Exited
0	(-0.001, 118100.59]	0.183441
1	(118100.59, 250898.09]	0.244224

## 5 Encoder

```
In [19]: data_encoder = dataset.copy()
data_encoder['Geography'] = LabelEncoder().fit_transform(data_encoder['Geography'])
# data_encoder['Surname'] = LabelEncoder().fit_transform(data_encoder['Surname'])
# data_encoder['Gender'] = LabelEncoder().fit_transform(data_encoder['Gender'])
data_encoder = data_encoder.join(pd.get_dummies(data_encoder['Gender'], prefix='Gender'))
data_encoder = data_encoder.drop('Gender', axis=1)

data_encoder.loc[ data_encoder['Balance'] <= 118100.59, 'Balance'] = 0
data_encoder.loc[ data_encoder['Balance'] > 118100.59, 'Balance'] = 1

data_encoder.head(10)
```

```
Out[19]:
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Age	Tenure	\
0	1	15634602	Hargrave	619	0	42	2	
1	2	15647311	Hill	608	2	41	1	
2	3	15619304	Onio	502	0	42	8	
3	4	15701354	Boni	699	0	39	1	
4	5	15737888	Mitchell	850	2	43	2	
5	6	15574012	Chu	645	2	44	8	
6	7	15592531	Bartlett	822	0	50	7	
7	8	15656148	Obinna	376	1	29	4	

8	9	15792365	He	501	0	44	4
9	10	15592389	H?	684	0	27	2

	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	\
0	0.0	1	1	1	101348.88	1	
1	0.0	1	0	1	112542.58	0	
2	1.0	3	1	0	113931.57	1	
3	0.0	2	0	0	93826.63	0	
4	1.0	1	1	1	79084.10	0	
5	0.0	2	1	0	149756.71	1	
6	0.0	2	1	1	10062.80	0	
7	0.0	4	1	0	119346.88	1	
8	1.0	2	0	1	74940.50	0	
9	1.0	1	1	1	71725.73	0	

	CategoricalCreditScore	CategoricalAge	CategoricalBalance	\
0	(608.0, 695.0]	(40.0, 46.0]	(-0.001, 118100.59]	
1	(349.999, 608.0]	(40.0, 46.0]	(-0.001, 118100.59]	
2	(349.999, 608.0]	(40.0, 46.0]	(118100.59, 250898.09]	
3	(695.0, 850.0]	(35.0, 40.0]	(-0.001, 118100.59]	
4	(695.0, 850.0]	(40.0, 46.0]	(118100.59, 250898.09]	
5	(608.0, 695.0]	(40.0, 46.0]	(-0.001, 118100.59]	
6	(695.0, 850.0]	(46.0, 92.0]	(-0.001, 118100.59]	
7	(349.999, 608.0]	(17.999, 31.0]	(-0.001, 118100.59]	
8	(349.999, 608.0]	(40.0, 46.0]	(118100.59, 250898.09]	
9	(608.0, 695.0]	(17.999, 31.0]	(118100.59, 250898.09]	

	Gender_Female	Gender_Male
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0
5	0	1
6	0	1
7	1	0
8	0	1
9	0	1

```
In [20]: AttList = ["RowNumber", "CustomerId", "Surname", "CategoricalCreditScore", "CategoricalAge", "CategoricalBalance"]
data_encoder = data_encoder.drop(AttList, axis=1)
data_encoder.head()
```

```
Out[20]:
```

	CreditScore	Geography	Age	Tenure	Balance	NumOfProducts	HasCrCard	\
0	619	0	42	2	0.0	1	1	
1	608	2	41	1	0.0	1	0	
2	502	0	42	8	1.0	3	1	
3	699	0	39	1	0.0	2	0	

4	850	2	43	2	1.0	1	1
	IsActiveMember	EstimatedSalary	Exited	Gender_Female	Gender_Male		
0	1	101348.88	1	1	0		
1	1	112542.58	0	1	0		
2	0	113931.57	1	1	0		
3	0	93826.63	0	1	0		
4	1	79084.10	0	1	0		

```
In [21]: # Split Train and Test and check shape
         data_train_encoder, target_train_encoder, data_test_encoder, target_test_encoder = Sp
         PrintTrainTestInformation(data_train_encoder, target_train_encoder, data_test_encoder
```

```
Train rows and columns : (6000, 11)
Test rows and columns : (4000, 11)
```

## 5.1 Classification by trairditional models

```
In [0]: X_train = data_train_encoder
        y_train = target_train_encoder
        X_test = data_test_encoder
        y_test = target_test_encoder
```

```
In [23]: MachineLearningModelEvaluate(X_train, y_train, X_test, y_test)
```

```
Naive Bayes accuracy: 0.770
Logistic Regression accuracy: 0.785
Random Forest accuracy: 0.839500
Linear SVM accuracy: 0.798250
RBF SVM accuracy: 0.798250
K Nearest Neighbor accuracy: 0.736250
ANN accuracy: 0.797750
```

## 6 Approach 1 (Feature Selection)

## 7 Correlation

```
In [24]: ## get the most important variables.
         corr = dataset.corr()**2
         corr.Exited.sort_values(ascending=False)
```

```
Out[24]: Exited          1.000000
         Age            0.081409
         IsActiveMember  0.024376
         Balance         0.014050
         NumOfProducts   0.002287
```

```

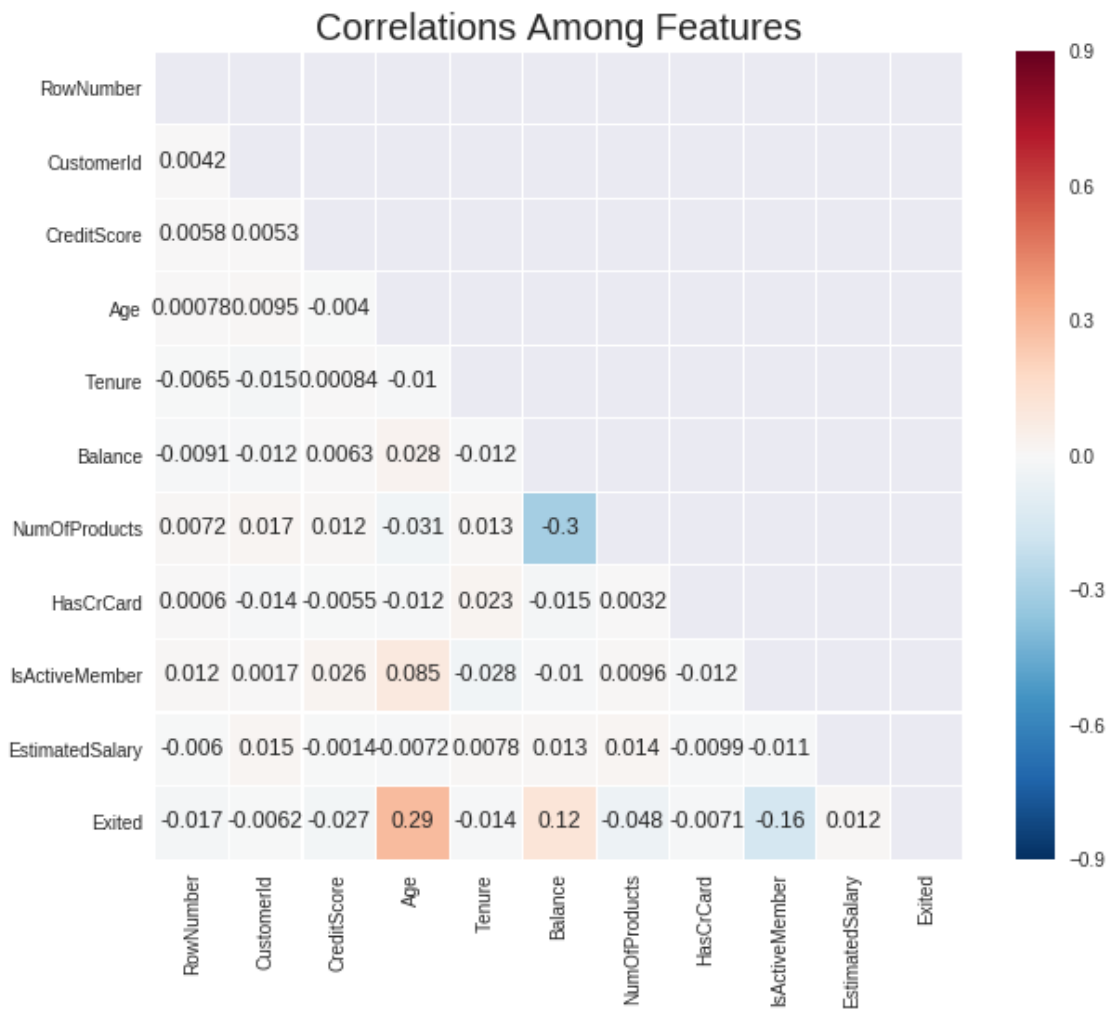
CreditScore      0.000734
RowNumber        0.000275
Tenure           0.000196
EstimatedSalary  0.000146
HasCrCard        0.000051
CustomerId       0.000039
Name: Exited, dtype: float64

```

```

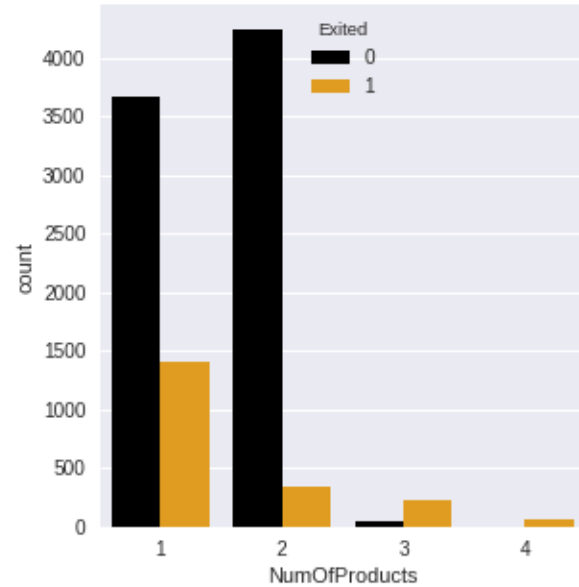
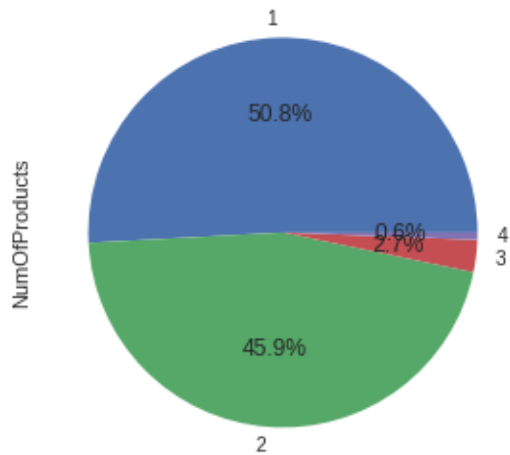
In [25]: # Heatmap to see the correlation between features.
# Generate a mask for the upper triangle (taken from seaborn example gallery)
mask = np.zeros_like(dataset.corr(), dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
# plot
plt.subplots(figsize = (10,8))
sns.heatmap(dataset.corr(), annot=True, mask = mask, cmap = 'RdBu_r', linewidths=0.1,
plt.title("Correlations Among Features", y = 1.03,fontsize = 20);

```



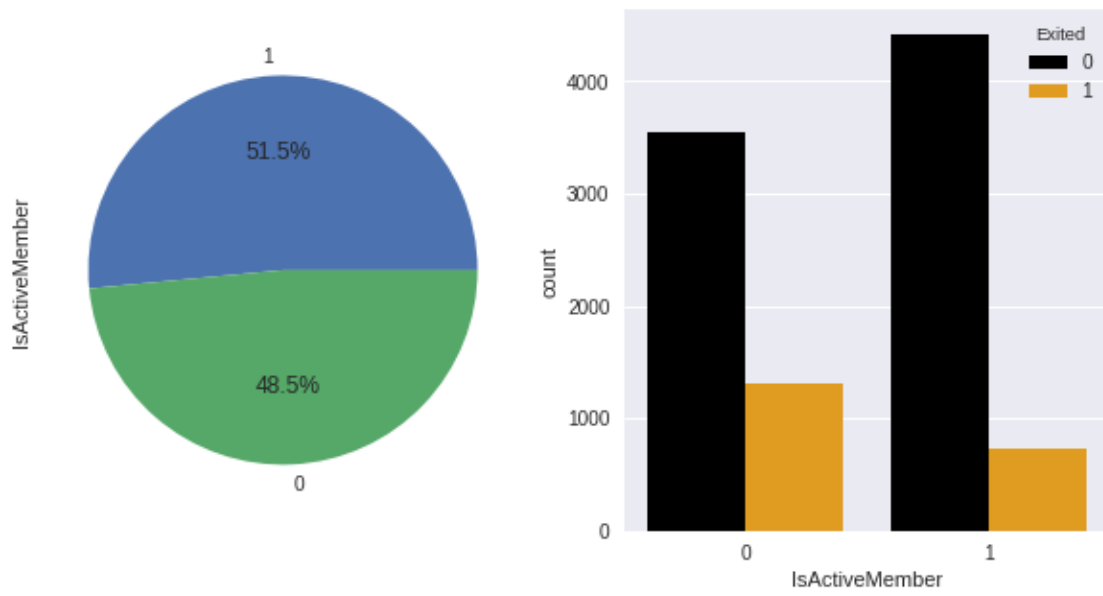
```
In [26]: print(dataset[['NumOfProducts', 'Exited']].groupby(['NumOfProducts'], as_index=False)
          CatPlot(dataset, 'NumOfProducts', 'Exited', {0: "black", 1: "orange"} )
```

	NumOfProducts	Exited
3	4	1.000000
2	3	0.827068
0	1	0.277144
1	2	0.075817



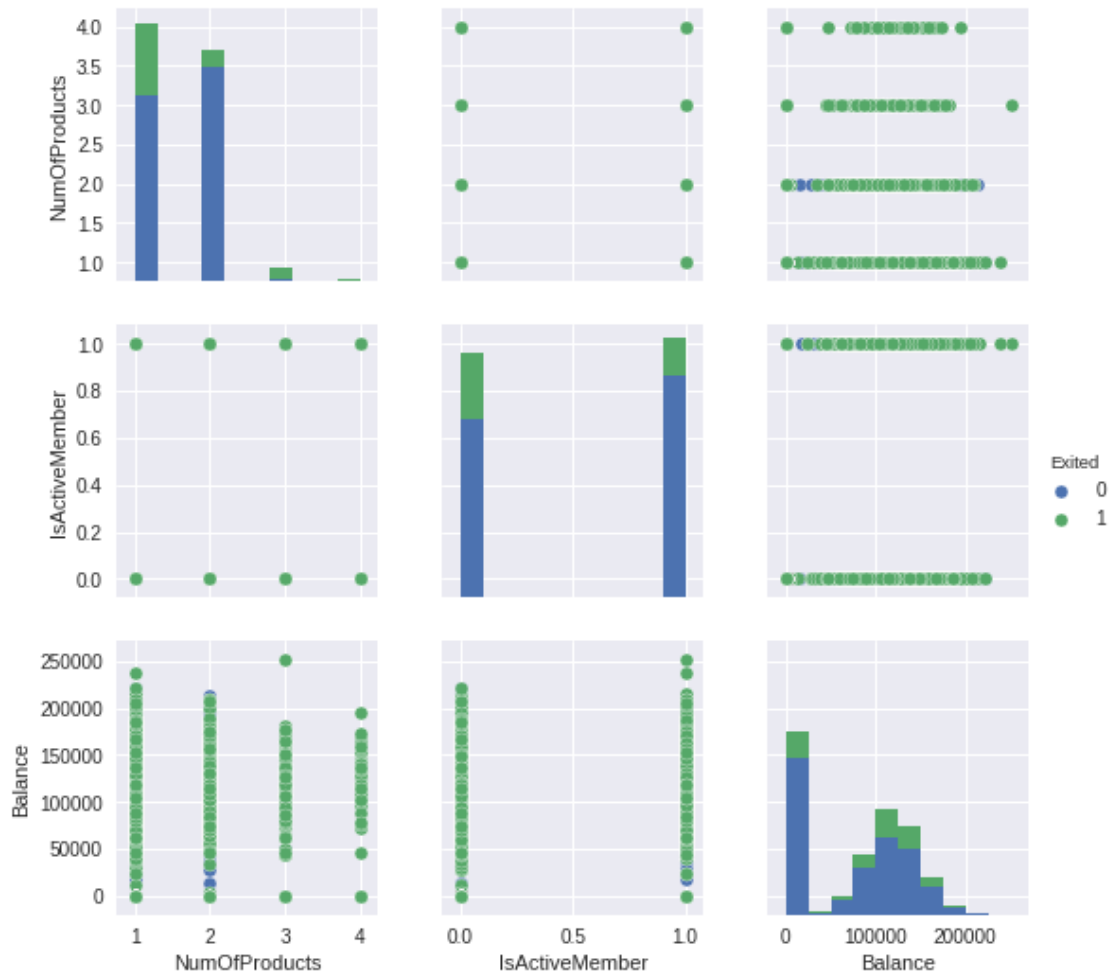
```
In [27]: print(dataset[['IsActiveMember', 'Exited']].groupby(['IsActiveMember'], as_index=False)
          CatPlot(dataset, 'IsActiveMember', 'Exited', {0: "black", 1: "orange"} )
```

	IsActiveMember	Exited
0	0	0.268509
1	1	0.142691



```
In [28]: # https://seaborn.pydata.org/generated/seaborn.pairplot.html
sns.pairplot(dataset, vars=["NumOfProducts", "IsActiveMember", "Balance"], hue="Exited")
```

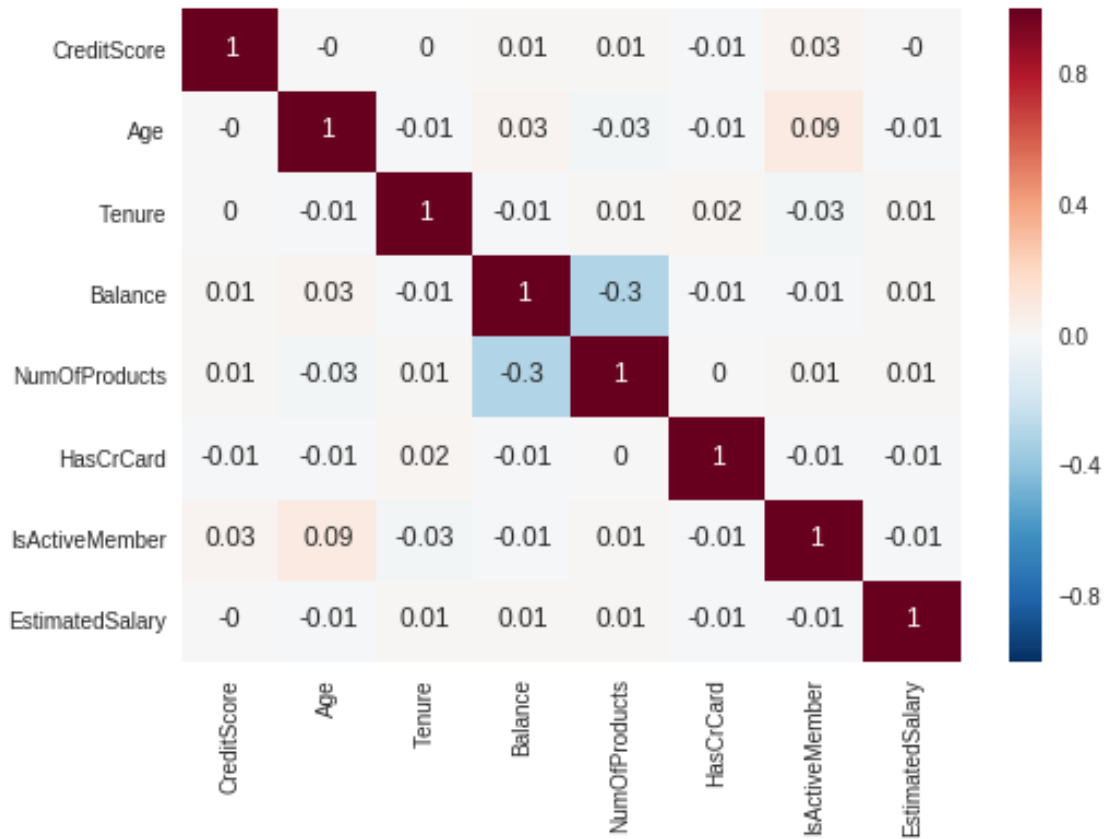
```
Out[28]: <seaborn.axisgrid.PairGrid at 0x7f97f8b54198>
```



```
In [29]: AttList = ["CreditScore", "Age", "Tenure", "Balance", "NumOfProducts"]
correlation_matrix = dataset[AttList].corr().round(2)
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
```

```
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x7f97f8ef3dd8>
```





```
In [30]: data_encoder_feselection = data_encoder.copy()
# AttList = ["Surname", "RowNumber", "CustomerId"]
# data_encoder_feselection = data_encoder_feselection.drop(AttList, axis=1)
print(data_encoder_feselection.shape)
data_encoder_feselection.head()
```

(10000, 12)

```
Out [30]:
```

	CreditScore	Geography	Age	Tenure	Balance	NumOfProducts	HasCrCard	\
0	619	0	42	2	0.0	1	1	
1	608	2	41	1	0.0	1	0	
2	502	0	42	8	1.0	3	1	
3	699	0	39	1	0.0	2	0	
4	850	2	43	2	1.0	1	1	

	IsActiveMember	EstimatedSalary	Exited	Gender_Female	Gender_Male
0	1	101348.88	1	1	0
1	1	112542.58	0	1	0
2	0	113931.57	1	1	0
3	0	93826.63	0	1	0
4	1	79084.10	0	1	0

```
In [31]: # Split Train and Test and check shape
        data_train_encoder_feselection, target_train_encoder_feselection, data_test_encoder_feselection =
        PrintTrainTestInformation(data_train_encoder_feselection, target_train_encoder_feselection, data_test_encoder_feselection)
```

```
Train rows and columns : (6000, 11)
```

```
Test rows and columns : (4000, 11)
```

```
In [32]: # Retest all traditional classification approaches
```

```
X_train = data_train_encoder
y_train = target_train_encoder
X_test = data_test_encoder
y_test = target_test_encoder
```

```
MachineLearningModelEvaluate(X_train, y_train, X_test, y_test)
```

```
Naive Bayes accuracy: 0.770
```

```
Logistic Regression accuracy: 0.785
```

```
Random Forest accuracy: 0.846000
```

```
Linear SVM accuracy: 0.798250
```

```
RBF SVM accuracy: 0.798250
```

```
K Nearest Neighbor accuracy: 0.736250
```

```
ANN accuracy: 0.798000
```

```
In [33]: # Retest all traditional classification approaches
```

```
X_train = data_train_encoder_feselection
y_train = target_train_encoder_feselection
X_test = data_test_encoder_feselection
y_test = target_test_encoder_feselection
```

```
MachineLearningModelEvaluate(X_train, y_train, X_test, y_test)
```

```
Naive Bayes accuracy: 0.770
```

```
Logistic Regression accuracy: 0.785
```

```
Random Forest accuracy: 0.840000
```

```
Linear SVM accuracy: 0.798250
```

```
RBF SVM accuracy: 0.798250
```

```
K Nearest Neighbor accuracy: 0.736250
```

```
ANN accuracy: 0.797500
```

## 7.1 Feature Importances

```
In [71]: model = RandomForestRegressor(random_state=1, max_depth=10)
        model.fit(data_train_encoder, target_train_encoder.values.ravel())

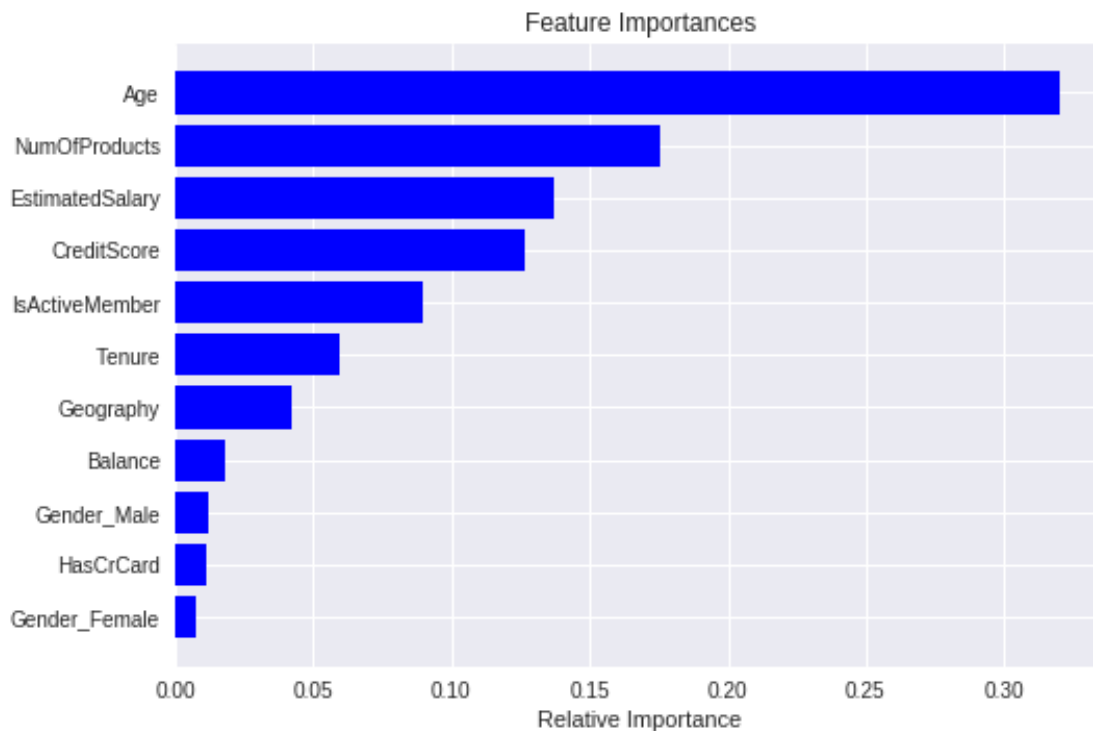
        print(data_train_encoder.shape)
        features = data_train_encoder.columns
```

```

importances = model.feature_importances_
indices = np.argsort(importances)[-len(features):] # top features
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()

```

(6000, 11)



```

In [72]: # Get numerical feature importances
feature_list = list(data_train_encoder.columns)
importances = list(model.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(feature_list, importances)]
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances]

```

```

Variable: Age                Importance: 0.32
Variable: NumOfProducts      Importance: 0.18
Variable: EstimatedSalary    Importance: 0.14

```

Variable: CreditScore	Importance: 0.13
Variable: IsActiveMember	Importance: 0.09
Variable: Tenure	Importance: 0.06
Variable: Geography	Importance: 0.04
Variable: Balance	Importance: 0.02
Variable: HasCrCard	Importance: 0.01
Variable: Gender_Female	Importance: 0.01
Variable: Gender_Male	Importance: 0.01

In [74]: *# Split Train and Test and check shape*

```
AttSelection = ["Age", "NumOfProducts", "EstimatedSalary", "CreditScore", "Tenure", "Exited"]
```

```
data_train_encoder_feselection02, target_train_encoder_feselection02, data_test_encoder_feselection02, target_test_encoder_feselection02 = train_test_split(data_train_encoder_feselection02, target_train_encoder_feselection02, data_test_encoder_feselection02, target_test_encoder_feselection02)
```

Train rows and columns : (6000, 7)

Test rows and columns : (4000, 7)

In [75]: *# Retest all traditional classification approaches*

```
X_train = data_train_encoder_feselection02
y_train = target_train_encoder_feselection02
X_test = data_test_encoder_feselection02
y_test = target_test_encoder_feselection02
```

```
MachineLearningModelEvaluate(X_train, y_train, X_test, y_test)
```

Naive Bayes accuracy: 0.767

Logistic Regression accuracy: 0.784

Random Forest accuracy: 0.830750

Linear SVM accuracy: 0.798250

RBF SVM accuracy: 0.798250

K Nearest Neighbor accuracy: 0.736000

ANN accuracy: 0.798750

```
In [38]: from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
import pandas as pd
from sklearn.svm import SVR
```

```
# Retest all traditional classification approaches
```

```
X_train = data_train_encoder
y_train = target_train_encoder
X_test = data_test_encoder
y_test = target_test_encoder
```

```

LRModel = LogisticRegressionLearning(X_train, y_train)
model = LRModel
rfe = RFE(model, 10)
rfe = rfe.fit(X_train, y_train.values.ravel())

feature_list = list(X_train.columns)
RankStatistics = pd.DataFrame(columns=['Attributes', 'Ranking', 'Support'])
for i, att, rank, support in zip(range(len(feature_list)), feature_list, rfe.ranking_):
    RankStatistics.loc[i] = [att, rank, support]
RankStatistics = RankStatistics.sort_values('Ranking')

RankStatistics

```

```

Out [38]:
      Attributes Ranking Support
0      CreditScore      1     True
1       Geography      1     True
2           Age      1     True
3        Tenure      1     True
4        Balance      1     True
5  NumOfProducts      1     True
6      HasCrCard      1     True
7  IsActiveMember      1     True
9   Gender_Female      1     True
10  Gender_Male      1     True
8  EstimatedSalary      2    False

```

```

In [39]: # Split Train and Test and check shape
AttSelection = RankStatistics[(RankStatistics["Support"] == True)]
AttSelection = list(filter(lambda a: a not in ["CustomerId", "Surname"], AttSelection))
AttSelection = AttSelection + ['Exited']

data_train_encoder_feselection03, target_train_encoder_feselection03, data_test_encoder_feselection03, target_test_encoder_feselection03 = train_test_split(
    RankStatistics, RankStatistics['Support'], test_size=0.2, random_state=42)
PrintTrainTestInformation(data_train_encoder_feselection03, target_train_encoder_feselection03, data_test_encoder_feselection03, target_test_encoder_feselection03)

Train rows and columns : (6000, 10)
Test rows and columns : (4000, 10)

```

```

In [40]: # Retest all traditional classification approaches
X_train = data_train_encoder_feselection03
y_train = target_train_encoder_feselection03
X_test = data_test_encoder_feselection03
y_test = target_test_encoder_feselection03

MachineLearningModelEvaluate(X_train, y_train, X_test, y_test)

Naive Bayes accuracy: 0.818
Logistic Regression accuracy: 0.803
Random Forest accuracy: 0.841000

```

Linear SVM accuracy: 0.795500  
RBF SVM accuracy: 0.798250  
K Nearest Neighbor accuracy: 0.772750  
ANN accuracy: 0.806750

## 8 Approach 2 (Feature Reduction)

In [41]: *# Feature Reduction: Dimensionality Reduction with PCA.*

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

AttRemoved = ["RowNumber", "CustomerId", "Surname", "HasCrCard", "Gender_Male", "Gender_Female"]
DataFrame = data_encoder
hr_vars = DataFrame.columns.values.tolist()
hr_vars = list(filter(lambda a: a not in AttRemoved, hr_vars))
targets = ['Exited']
features = [i for i in hr_vars if i not in targets]

# Separating out the features
x = DataFrame.loc[:, features].values
# Separating out the target
y = DataFrame.loc[:, ['Exited']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)

nSelectedFeature = len(hr_vars) - 1
SelectedAttList = []
for i in range(1, nSelectedFeature + 1):
    SelectedAttList.append("principal component" + str(i))

pca = PCA(n_components=nSelectedFeature)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data=principalComponents, columns=SelectedAttList)
PCAdf = pd.concat([principalDf, DataFrame[targets]], axis=1)
PCAdf = PCAdf.dropna()
PCAdata = PCAdf

PCAdata.head(10)
```

Out [41]:

	principal component1	principal component2	principal component3	\
0	-0.210853	0.884002	1.446764	
1	-0.569838	1.195424	0.047422	
2	0.869274	-1.103759	0.016253	
3	1.128893	-0.151478	1.081788	

4	-1.923635	1.451394	-1.077486
5	0.847114	-0.370761	-1.909737
6	0.937177	1.788649	0.042209
7	3.409745	-1.006174	0.435076
8	-0.565255	0.566436	1.276217
9	-1.303973	-0.089951	1.253032

	principal component4	principal component5	principal component6 \
0	0.278925	-0.634165	0.096706
1	0.380679	0.028651	-1.688115
2	1.523084	0.787701	0.617678
3	-0.534498	-0.756419	-0.618177
4	-1.795071	-0.735991	-0.963920
5	0.815389	0.614184	-0.434943
6	-1.637323	0.347318	1.693827
7	2.204620	1.414215	-1.814186
8	1.057400	0.478299	0.262575
9	-1.072287	-0.924581	-0.059585

	principal component7	principal component8	Exited
0	-0.253957	-0.944001	1
1	-0.447760	-1.349368	0
2	0.319225	2.904623	1
3	1.185436	0.322298	0
4	-0.146821	0.122698	0
5	0.884744	-0.210759	1
6	0.486942	0.326536	0
7	-0.327603	2.408140	1
8	-0.764268	1.755141	0
9	-1.356978	0.445756	0

```
In [42]: PCAdat_train, PCAtarget_train, PCAdat_test, PCAtarget_test = SplitDataFrameToTrainAndTest(PCAdat, PCAtarget)
         PrintTrainTestInformation(PCAdat_train, PCAtarget_train, PCAdat_test, PCAtarget_test)
```

```
Train rows and columns : (6000, 8)
Test rows and columns : (4000, 8)
```

```
In [43]: # Retest all traditional classification approaches
```

```
X_train = PCAdat_train
y_train = PCAtarget_train
X_test = PCAdat_test
y_test = PCAtarget_test
```

```
MachineLearningModelEvaluate(X_train, y_train, X_test, y_test)
```

```
Naive Bayes accuracy: 0.827
Logistic Regression accuracy: 0.802
Random Forest accuracy: 0.839000
```

Linear SVM accuracy: 0.853500  
RBF SVM accuracy: 0.810500  
K Nearest Neighbor accuracy: 0.817500  
ANN accuracy: 0.850500

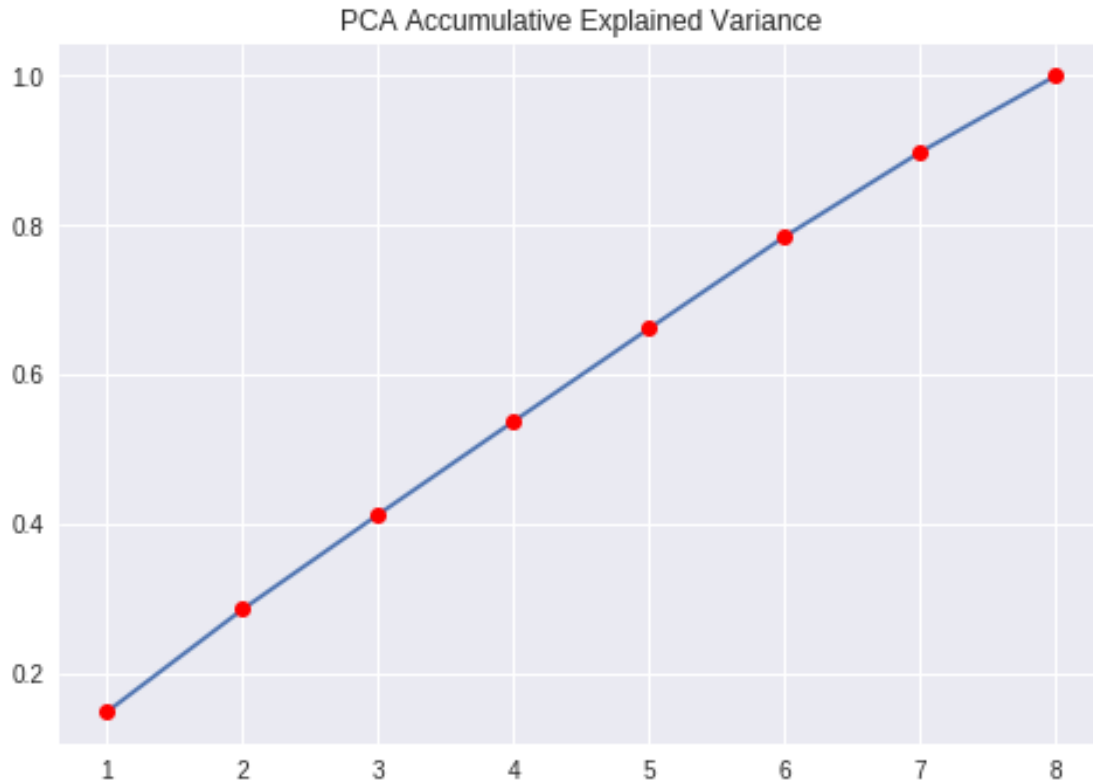
```
In [44]: import matplotlib.pyplot as plt

cum_explained_var = []
for i in range(0, len(pca.explained_variance_ratio_)):
    if i == 0:
        cum_explained_var.append(pca.explained_variance_ratio_[i])
    else:
        cum_explained_var.append(pca.explained_variance_ratio_[i] +
                                cum_explained_var[i - 1])

x_val = range(1, len(cum_explained_var) + 1)
y_val = cum_explained_var

fig = plt.figure()
plt.plot(x_val, y_val)
plt.plot(x_val, y_val, 'or')
plt.title("PCA Accumulative Explained Variance")
plt.xticks(range(1, len(cum_explained_var) + 1))
plt.grid(True)
plt.show()
```





```
In [45]: AttSelection = PCAdat.columns.values.tolist()
AttSelection = AttSelection[:15]
if AttSelection[len(AttSelection)-1] != 'Exited' :
    AttSelection = AttSelection + ['Exited']
print(AttSelection)
```

```
PCAdat_train_feReduction, PCAtarget_train_feReduction, PCAdat_test_feReduction, PCAtarget_test_feReduction
PrintTrainTestInformation(PCAdat_train_feReduction, PCAtarget_train_feReduction, PCAdat_test_feReduction, PCAtarget_test_feReduction)
```

```
['principal component1', 'principal component2', 'principal component3', 'principal component4']
Train rows and columns : (6000, 8)
Test rows and columns : (4000, 8)
```

```
In [46]: # Retest all traditional classification approaches
X_train = PCAdat_train_feReduction
y_train = PCAtarget_train_feReduction
X_test = PCAdat_test_feReduction
y_test = PCAtarget_test_feReduction

MachineLearningModelEvaluate(X_train, y_train, X_test, y_test)
```

Naive Bayes accuracy: 0.827  
 Logistic Regression accuracy: 0.802  
 Random Forest accuracy: 0.839750  
 Linear SVM accuracy: 0.853500  
 RBF SVM accuracy: 0.810500  
 K Nearest Neighbor accuracy: 0.817500  
 ANN accuracy: 0.850750

## 9 Outlier Removal Approach

In [47]: data\_encoder.head()

```
Out[47]:
```

	CreditScore	Geography	Age	Tenure	Balance	NumOfProducts	HasCrCard	\
0	619	0	42	2	0.0	1	1	
1	608	2	41	1	0.0	1	0	
2	502	0	42	8	1.0	3	1	
3	699	0	39	1	0.0	2	0	
4	850	2	43	2	1.0	1	1	

	IsActiveMember	EstimatedSalary	Exited	Gender_Female	Gender_Male
0	1	101348.88	1	1	0
1	1	112542.58	0	1	0
2	0	113931.57	1	1	0
3	0	93826.63	0	1	0
4	1	79084.10	0	1	0

In [48]: data\_encoder.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
CreditScore      10000 non-null int64
Geography        10000 non-null int64
Age              10000 non-null int64
Tenure           10000 non-null int64
Balance          10000 non-null float64
NumOfProducts    10000 non-null int64
HasCrCard        10000 non-null int64
IsActiveMember   10000 non-null int64
EstimatedSalary  10000 non-null float64
Exited           10000 non-null int64
Gender_Female    10000 non-null uint8
Gender_Male      10000 non-null uint8
dtypes: float64(2), int64(8), uint8(2)
memory usage: 800.9 KB
```

```
In [49]: CheckOutlierAtt = ['CreditScore', 'Geography']
        LOFOutlierIdx01, LOFFactorData01 = DetectOutlierByLOF(data_encoder, AttList=CheckOutlierAtt)

        print("Size of LOFOutlierIdx : " + str(len(LOFOutlierIdx01)))
        print(LOFFactorData01.head())
```

Size of LOFOutlierIdx : 1656

	index	LOF
0	3871	9.000000e+09
1	9409	9.000000e+09
2	656	8.000000e+09
3	17	8.000000e+09
4	268	8.000000e+09

```
In [50]: CheckOutlierAtt = ['Age', 'Tenure', 'Balance']
        LOFOutlierIdx02, LOFFactorData02 = DetectOutlierByLOF(data_encoder, AttList=CheckOutlierAtt)

        print("Size of LOFOutlierIdx : " + str(len(LOFOutlierIdx02)))
        print(LOFFactorData02.head())
```

Size of LOFOutlierIdx : 1020

	index	LOF
0	8174	8.000000e+09
1	5881	8.000000e+09
2	4402	8.000000e+09
3	7122	8.000000e+09
4	5228	8.000000e+09

```
In [51]: CheckOutlierAtt = ['HasCrCard', 'IsActiveMember', 'EstimatedSalary']
        LOFOutlierIdx03, LOFFactorData03 = DetectOutlierByLOF(data_encoder, AttList=CheckOutlierAtt)

        print("Size of LOFOutlierIdx : " + str(len(LOFOutlierIdx03)))
        print(LOFFactorData03.head())
```

Size of LOFOutlierIdx : 0

	index	LOF
0	2813	2.275599
1	8397	2.226634
2	3470	2.079527
3	4132	2.013270
4	7478	2.001296

```
In [52]: print('LOFOutlierIdx01 :' + str(LOFOutlierIdx01))
        print('LOFOutlierIdx02 :' + str(LOFOutlierIdx02))
        print('LOFOutlierIdx03 :' + str(LOFOutlierIdx03))
```

```

LOFOutlierIdx01 :[3871, 9409, 656, 17, 268, 4792, 6836, 5589, 2233, 6095, 7587, 1053, 8489, 22
LOFOutlierIdx02 :[8174, 5881, 4402, 7122, 5228, 6198, 3042, 2595, 4520, 5049, 7500, 8476, 3454
LOFOutlierIdx03 :[]

```

```

In [53]: OutlierIndex = set(LOFOutlierIdx01 + LOFOutlierIdx02 + LOFOutlierIdx03)
        OutlierIndex = list(OutlierIndex)
        print(len(OutlierIndex))
        print('OutlierIdx : ' + str(OutlierIndex))

```

2485

```

OutlierIdx : [2, 8195, 8194, 6, 8, 13, 15, 16, 17, 18, 19, 8213, 8217, 8220, 8222, 30, 32, 822

```

```

In [54]: data_encoder_mining = data_encoder.copy()
        print(data_encoder_mining.shape)
        data_encoder_mining = RemoveRowsFromDataFrame(data_encoder_mining, OutlierIndex)
        print(data_encoder_mining.shape)

```

```

        # feature selection
        # AttList = ["Surname", "RowNumber", "CustomerId"]
        # data_encoder_mining = data_encoder_mining.drop(AttList, axis=1)
        # print(data_encoder_mining.shape)

```

(10000, 12)

(7515, 12)

```

In [55]: # Split Train and Test and check shape
        data_train_encoder_mining, target_train_encoder_mining, data_test_encoder_mining, tar
        PrintTrainTestInformation(data_train_encoder_mining, target_train_encoder_mining, data

```

Train rows and columns : (4509, 11)

Test rows and columns : (3006, 11)

```

In [56]: # Retest all traditional classification approaches

```

```

        X_train = data_train_encoder_mining
        y_train = target_train_encoder_mining
        X_test = data_test_encoder_mining
        y_test = target_test_encoder_mining

```

```

        MachineLearningModelEvaluate(X_train, y_train, X_test, y_test)

```

Naive Bayes accuracy: 0.791

Logistic Regression accuracy: 0.807

Random Forest accuracy: 0.858949

Linear SVM accuracy: 0.811045

RBF SVM accuracy: 0.811045

K Nearest Neighbor accuracy: 0.767132

ANN accuracy: 0.207917

## 10 Neural Network Approach

```
In [57]: # Retest all traditional classification approaches
# X_train = data_train_encoder_mining
# y_train = target_train_encoder_mining
# X_test = data_test_encoder_mining
# y_test = target_test_encoder_mining

X_train = PCAdata_train_feReduction
y_train = PCAtarget_train_feReduction
X_test = PCAdata_test_feReduction
y_test = PCAtarget_test_feReduction

from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint

seed = 42
np.random.seed(seed)

## Create our model
model = Sequential()

# 1st layer: 23 nodes, input shape[1] nodes, RELU
model.add(Dense(23, input_dim=X_train.shape[1], kernel_initializer='uniform', activation='relu'))
# 2nd layer: 17 nodes, RELU
model.add(Dense(17, kernel_initializer='uniform', activation='relu'))
# 3rd layer: 15 nodes, RELU
model.add(Dense(15, kernel_initializer='uniform', activation='relu'))
# 4th layer: 11 nodes, RELU
model.add(Dense(11, kernel_initializer='uniform', activation='relu'))
# 5th layer: 9 nodes, RELU
model.add(Dense(9, kernel_initializer='uniform', activation='relu'))
# 6th layer: 7 nodes, RELU
model.add(Dense(7, kernel_initializer='uniform', activation='relu'))
# 7th layer: 5 nodes, RELU
model.add(Dense(5, kernel_initializer='uniform', activation='relu'))
# 8th layer: 2 nodes, RELU
model.add(Dense(2, kernel_initializer='uniform', activation='relu'))
# output layer: dim=1, activation sigmoid
model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid' ))
# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

NB_EPOCHS = 100
BATCH_SIZE = 23

# checkpoint: store the best model
```

```

ckpt_model = 'pima-weights.best.hdf5'
checkpoint = ModelCheckpoint(ckpt_model, monitor='val_acc', verbose=1, save_best_only=True,
callbacks_list = [checkpoint])

print('Starting training...')
# train the model, store the results for plotting
history = model.fit(X_train,
                    y_train,
                    validation_data=(X_test, y_test),
                    epochs=NB_EPOCHS,
                    batch_size=BATCH_SIZE,
                    callbacks=callbacks_list,
                    verbose=0)

```

Using TensorFlow backend.

Starting training...

```

Epoch 00001: val_acc improved from -inf to 0.79825, saving model to pima-weights.best.hdf5
Epoch 00002: val_acc did not improve from 0.79825
Epoch 00003: val_acc did not improve from 0.79825
Epoch 00004: val_acc did not improve from 0.79825
Epoch 00005: val_acc improved from 0.79825 to 0.82400, saving model to pima-weights.best.hdf5
Epoch 00006: val_acc improved from 0.82400 to 0.82750, saving model to pima-weights.best.hdf5
Epoch 00007: val_acc did not improve from 0.82750
Epoch 00008: val_acc improved from 0.82750 to 0.84075, saving model to pima-weights.best.hdf5
Epoch 00009: val_acc improved from 0.84075 to 0.84325, saving model to pima-weights.best.hdf5
Epoch 00010: val_acc improved from 0.84325 to 0.84400, saving model to pima-weights.best.hdf5
Epoch 00011: val_acc improved from 0.84400 to 0.84625, saving model to pima-weights.best.hdf5
Epoch 00012: val_acc did not improve from 0.84625
Epoch 00013: val_acc improved from 0.84625 to 0.84875, saving model to pima-weights.best.hdf5
Epoch 00014: val_acc did not improve from 0.84875
Epoch 00015: val_acc did not improve from 0.84875

```

Epoch 00016: val\_acc did not improve from 0.84875

Epoch 00017: val\_acc improved from 0.84875 to 0.84975, saving model to pima-weights.best.hdf5

Epoch 00018: val\_acc improved from 0.84975 to 0.85100, saving model to pima-weights.best.hdf5

Epoch 00019: val\_acc did not improve from 0.85100

Epoch 00020: val\_acc improved from 0.85100 to 0.85275, saving model to pima-weights.best.hdf5

Epoch 00021: val\_acc did not improve from 0.85275

Epoch 00022: val\_acc did not improve from 0.85275

Epoch 00023: val\_acc did not improve from 0.85275

Epoch 00024: val\_acc did not improve from 0.85275

Epoch 00025: val\_acc did not improve from 0.85275

Epoch 00026: val\_acc did not improve from 0.85275

Epoch 00027: val\_acc did not improve from 0.85275

Epoch 00028: val\_acc did not improve from 0.85275

Epoch 00029: val\_acc did not improve from 0.85275

Epoch 00030: val\_acc did not improve from 0.85275

Epoch 00031: val\_acc did not improve from 0.85275

Epoch 00032: val\_acc did not improve from 0.85275

Epoch 00033: val\_acc did not improve from 0.85275

Epoch 00034: val\_acc did not improve from 0.85275

Epoch 00035: val\_acc did not improve from 0.85275

Epoch 00036: val\_acc did not improve from 0.85275

Epoch 00037: val\_acc did not improve from 0.85275

Epoch 00038: val\_acc did not improve from 0.85275

Epoch 00039: val\_acc did not improve from 0.85275

Epoch 00040: val\_acc did not improve from 0.85275  
Epoch 00041: val\_acc did not improve from 0.85275  
Epoch 00042: val\_acc did not improve from 0.85275  
Epoch 00043: val\_acc did not improve from 0.85275  
Epoch 00044: val\_acc did not improve from 0.85275  
Epoch 00045: val\_acc did not improve from 0.85275  
Epoch 00046: val\_acc did not improve from 0.85275  
Epoch 00047: val\_acc did not improve from 0.85275  
Epoch 00048: val\_acc did not improve from 0.85275  
Epoch 00049: val\_acc did not improve from 0.85275  
Epoch 00050: val\_acc did not improve from 0.85275  
Epoch 00051: val\_acc did not improve from 0.85275  
Epoch 00052: val\_acc did not improve from 0.85275  
Epoch 00053: val\_acc did not improve from 0.85275  
Epoch 00054: val\_acc did not improve from 0.85275  
Epoch 00055: val\_acc did not improve from 0.85275  
Epoch 00056: val\_acc did not improve from 0.85275  
Epoch 00057: val\_acc did not improve from 0.85275  
Epoch 00058: val\_acc did not improve from 0.85275  
Epoch 00059: val\_acc did not improve from 0.85275  
Epoch 00060: val\_acc did not improve from 0.85275  
Epoch 00061: val\_acc did not improve from 0.85275  
Epoch 00062: val\_acc did not improve from 0.85275  
Epoch 00063: val\_acc did not improve from 0.85275



Epoch 00064: val\_acc did not improve from 0.85275  
Epoch 00065: val\_acc did not improve from 0.85275  
Epoch 00066: val\_acc did not improve from 0.85275  
Epoch 00067: val\_acc did not improve from 0.85275  
Epoch 00068: val\_acc did not improve from 0.85275  
Epoch 00069: val\_acc did not improve from 0.85275  
Epoch 00070: val\_acc did not improve from 0.85275  
Epoch 00071: val\_acc did not improve from 0.85275  
Epoch 00072: val\_acc did not improve from 0.85275  
Epoch 00073: val\_acc did not improve from 0.85275  
Epoch 00074: val\_acc did not improve from 0.85275  
Epoch 00075: val\_acc did not improve from 0.85275  
Epoch 00076: val\_acc did not improve from 0.85275  
Epoch 00077: val\_acc did not improve from 0.85275  
Epoch 00078: val\_acc did not improve from 0.85275  
Epoch 00079: val\_acc did not improve from 0.85275  
Epoch 00080: val\_acc did not improve from 0.85275  
Epoch 00081: val\_acc did not improve from 0.85275  
Epoch 00082: val\_acc did not improve from 0.85275  
Epoch 00083: val\_acc did not improve from 0.85275  
Epoch 00084: val\_acc did not improve from 0.85275  
Epoch 00085: val\_acc did not improve from 0.85275  
Epoch 00086: val\_acc did not improve from 0.85275  
Epoch 00087: val\_acc did not improve from 0.85275

Epoch 00088: val\_acc did not improve from 0.85275

Epoch 00089: val\_acc did not improve from 0.85275

Epoch 00090: val\_acc did not improve from 0.85275

Epoch 00091: val\_acc did not improve from 0.85275

Epoch 00092: val\_acc did not improve from 0.85275

Epoch 00093: val\_acc did not improve from 0.85275

Epoch 00094: val\_acc did not improve from 0.85275

Epoch 00095: val\_acc did not improve from 0.85275

Epoch 00096: val\_acc did not improve from 0.85275

Epoch 00097: val\_acc did not improve from 0.85275

Epoch 00098: val\_acc did not improve from 0.85275

Epoch 00099: val\_acc did not improve from 0.85275

Epoch 00100: val\_acc did not improve from 0.85275

## 11 *Bagging Boosting and Stacking*

```
In [58]: X = data_encoder_mining.copy()
        X = X.drop('Exited', 1)
        y = data_encoder_mining[['Exited']]
        X.head()
```

```
Out [58]:
```

	CreditScore	Geography	Age	Tenure	Balance	NumOfProducts	HasCrCard	\
0	619	0	42	2	0.0	1	1	
1	608	2	41	1	0.0	1	0	
2	699	0	39	1	0.0	2	0	
3	850	2	43	2	1.0	1	1	
4	645	2	44	8	0.0	2	1	

	IsActiveMember	EstimatedSalary	Gender_Female	Gender_Male
0	1	101348.88	1	0
1	1	112542.58	1	0
2	0	93826.63	1	0
3	1	79084.10	1	0
4	0	149756.71	0	1

```

In [0]: X = PCAdata.copy()
        X = X.drop('Exited', 1)
        y = PCAdata[['Exited']]
        X.head()

        X_train = PCAdata_train_feReduction
        y_train = PCAtarget_train_feReduction
        X_test = PCAdata_test_feReduction
        y_test = PCAtarget_test_feReduction

In [0]: NBModel = NaiveBayesLearning(X_train, y_train)
        LRModel = LogisticRegressionLearning(X_train, y_train)
        RFModel = RandomForestLearning(X_train, y_train)
        LiSVMModel = SVMLearning(X_train, y_train)
        RBFSVMModel = SVMLearning(X_train, y_train, 'RBF')
        KNNModel = KNNLearning(X_train, y_train)
        ANNModel = ANNLearning(X_train, y_train)

In [61]: from sklearn import model_selection
        print('5-fold cross validation:\n')
        labels = ['NaiveBayesLearning', 'LogisticRegressionLearning', 'RandomForestLearning',
                  'SVMLearningLinear', 'SVMLearningRBF', 'KNNLearning', 'ANNLearning']
        for clf, label in zip([NBModel, LRModel, RFModel, LiSVMModel, RBFSVMModel, KNNModel, ANNModel], labels):
            scores = model_selection.cross_val_score(clf, X, y.values.ravel(), cv=5, scoring='accuracy')
            print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))

5-fold cross validation:

Accuracy: 0.82 (+/- 0.00) [NaiveBayesLearning]
Accuracy: 0.81 (+/- 0.00) [LogisticRegressionLearning]
Accuracy: 0.84 (+/- 0.00) [RandomForestLearning]
Accuracy: 0.86 (+/- 0.00) [SVMLearningLinear]
Accuracy: 0.82 (+/- 0.00) [SVMLearningRBF]
Accuracy: 0.83 (+/- 0.01) [KNNLearning]
Accuracy: 0.85 (+/- 0.01) [ANNLearning]

In [62]: from mlxtend.classifier import EnsembleVoteClassifier
        eclf = EnsembleVoteClassifier(clfs=[RFModel,
                                           LiSVMModel,
                                           ANNModel], weights=[1,1,1])

        labels = ['RandomForestLearning', 'SVMLearningLinear', 'ANNModel', 'Ensemble']
        for clf, label in zip([RFModel, LiSVMModel, ANNModel, eclf], labels):
            scores = model_selection.cross_val_score(clf, X, y.values.ravel(), cv=5, scoring='accuracy')
            print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))

Accuracy: 0.84 (+/- 0.01) [RandomForestLearning]
Accuracy: 0.86 (+/- 0.00) [SVMLearningLinear]

```

```
Accuracy: 0.84 (+/- 0.02) [ANNModel]
Accuracy: 0.86 (+/- 0.01) [Ensemble]
```

```
In [63]: # Majority Rule (hard) Voting
```

```
mv_clf = MajorityVoteClassifier(classifiers=[RFModel, LiSVMModel, ANNModel])

labels = ['RandomForestLearning', 'SVMLearningLinear', 'ANN', 'Majority voting']
all_clf = [RFModel, LiSVMModel, ANNModel, mv_clf]

for clf, label in zip(all_clf, labels):
    scores = cross_val_score(estimator=clf, X=X, y=y.values.ravel(), cv=5, scoring='a
    print("ROC AUC: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
ROC AUC: 0.84 (+/- 0.01) [RandomForestLearning]
ROC AUC: 0.86 (+/- 0.00) [SVMLearningLinear]
ROC AUC: 0.84 (+/- 0.02) [ANN]
ROC AUC: 0.85 (+/- 0.00) [Majority voting]
```

```
In [64]: # Split Train and Test and check shape
```

```
data_train_encoder_mining, target_train_encoder_mining, data_test_encoder_mining, targ
PrintTrainTestInformation(data_train_encoder_mining, target_train_encoder_mining, data

# Retest all traditional classification approaches
X_train = data_train_encoder_mining
y_train = target_train_encoder_mining
X_test = data_test_encoder_mining
y_test = target_test_encoder_mining
```

```
Train rows and columns : (4509, 11)
Test rows and columns : (3006, 11)
```

```
In [65]: tree = DecisionTreeClassifier(criterion='entropy', max_depth=None, random_state=1)
bag = BaggingClassifier(base_estimator=RFModel,
                        n_estimators=1000,
                        max_samples=1.0,
                        max_features=1.0,
                        bootstrap=True,
                        bootstrap_features=False,
                        n_jobs=1,
                        random_state=1)

tree = tree.fit(X_train, y_train.values.ravel())
y_train_pred = tree.predict(X_train)
y_test_pred = tree.predict(X_test)
```

```

tree_train = accuracy_score(y_train, y_train_pred)
tree_test = accuracy_score(y_test, y_test_pred)
print('Decision tree train/test accuracies %.3f/%.3f'
      % (tree_train, tree_test))

```

```

bag = bag.fit(X_train, y_train.values.ravel())
y_train_pred = bag.predict(X_train)
y_test_pred = bag.predict(X_test)

```

```

bag_train = accuracy_score(y_train, y_train_pred)
bag_test = accuracy_score(y_test, y_test_pred)
print('Bagging train/test accuracies %.3f/%.3f'
      % (bag_train, bag_test))

```

Decision tree train/test accuracies 1.000/0.788

Bagging train/test accuracies 0.957/0.874

In [66]: `from sklearn.ensemble import AdaBoostClassifier`

```

tree = DecisionTreeClassifier(criterion='entropy', max_depth=None, random_state=1)
ada = AdaBoostClassifier(base_estimator=tree, n_estimators=500, learning_rate=0.1, ra
tree = tree.fit(X_train, y_train.values.ravel())
y_train_pred = tree.predict(X_train)
y_test_pred = tree.predict(X_test)

```

```

tree_train = accuracy_score(y_train, y_train_pred)
tree_test = accuracy_score(y_test, y_test_pred)
print('Decision tree train/test accuracies %.3f/%.3f'% (tree_train, tree_test))

```

```

ada = ada.fit(X_train, y_train.values.ravel())
y_train_pred = ada.predict(X_train)
y_test_pred = ada.predict(X_test)

```

```

ada_train = accuracy_score(y_train, y_train_pred)
ada_test = accuracy_score(y_test, y_test_pred)
print('AdaBoost train/test accuracies %.3f/%.3f'
      % (ada_train, ada_test))

```

Decision tree train/test accuracies 1.000/0.788

AdaBoost train/test accuracies 1.000/0.789

In [67]: `from mlxtend.classifier import StackingClassifier`  
`import matplotlib.gridspec as gridspec`  
`import itertools`  
`from mlxtend.plotting import plot_learning_curves`  
`from mlxtend.plotting import plot_decision_regions`

```

lr = LogisticRegression()
sclf = StackingClassifier(classifiers=[RFModel, LiSVMModel, ANNModel], meta_classifier=lr)

label = ['RandomForestLearning', 'SVMLearningLinear', 'ANN', 'Stacking Classifier']
clf_list = [RFModel, LiSVMModel, ANNModel, sclf]

clf_cv_mean = []
clf_cv_std = []
for clf, label in zip(clf_list, label):
    scores = cross_val_score(clf, X, y.values.ravel(), cv=5, scoring='accuracy')
    print("Accuracy: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))
    clf_cv_mean.append(scores.mean())
    clf_cv_std.append(scores.std())
    clf.fit(X, y.values.ravel())

```

```

Accuracy: 0.84 (+/- 0.01) [RandomForestLearning]
Accuracy: 0.86 (+/- 0.00) [SVMLearningLinear]
Accuracy: 0.84 (+/- 0.02) [ANN]
Accuracy: 0.84 (+/- 0.01) [Stacking Classifier]

```

## 12 Summaries

### 12.0.1 Using Bagging on RandomForest can make up to 87.4%