Report

February 6, 2020

1 THE TELCO CHURN CHALLENGE FOR REXEL

1.1 1. Introduction

The objective of this challenge is to prevent customer to stop using TELCO Inc phoning services.

There are many reasons why customers may churn. It's crucial to detect those customers before they leave.

One of the most effective way to achive that goal is to use the data.

Based on historical data, we are going to detect customers who may leave and suggest actions that can avoid the leaving.

1.2 2. The data

We have 2 datasets to achieve the challenge. *The training dataset*, will be use to train, test and evaluate machine learning models. The validation dataset is for the final submission of the challenge.

[1]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

data = pd.read_csv('data/training.csv', na_values=[" "])
```

[2]:
```
print("The dataset shape: ", data.shape)
```

```
The dataset shape:  (11981, 19)
```

The data we have for this challenge has **11981** rows and **19** columns (or variables). Let's visualize the first 6 rows of the data:

[3]:
```
data.head().T
```

[3]:
```
                                  0         1         2  \
CUSTOMER_ID                 C100000   C100001   C100006
COLLEGE                        zero       one      zero
DATA                            660   317.647   208.696
```

```
INCOME                             19995        31477        66742
OVERCHARGE                             0          155            0
LEFTOVER                               0           15           13
HOUSE                             897338       393396       937197
LESSTHAN600k                       False         True        False
CHILD                                  4            0            4
JOB_CLASS                              3            1            2
REVENUE                              160          100          127
HANDSET_PRICE                        155          245          493
OVER_15MINS_CALLS_PER_MONTH            1           27           20
TIME_CLIENT                          1.2          2.7          2.6
AVERAGE_CALL_DURATION                 15            4            4
REPORTED_SATISFACTION        very_unsat        unsat          avg
REPORTED_USAGE_LEVEL              little       little  very_little
CONSIDERING_CHANGE_OF_PLAN   considering  considering  considering
CHURNED                             STAY        LEAVE         STAY

                                       3                         4
CUSTOMER_ID                      C100008                   C100010
COLLEGE                             zero                       one
DATA                             265.018                       440
INCOME                             40864                   43321.5
OVERCHARGE                           183                       200
LEFTOVER                               0                         0
HOUSE                             986430                    394622
LESSTHAN600k                       False                      True
CHILD                                  3                         2
JOB_CLASS                              3                         3
REVENUE                               86                        77
HANDSET_PRICE                        390                       175
OVER_15MINS_CALLS_PER_MONTH           13                        18
TIME_CLIENT                          2.5                       2.4
AVERAGE_CALL_DURATION                 12                        10
REPORTED_SATISFACTION              unsat                very_unsat
REPORTED_USAGE_LEVEL           very_high                    little
CONSIDERING_CHANGE_OF_PLAN   considering  actively_looking_into_it
CHURNED                            LEAVE                     LEAVE
```

[4]:
```
data.dtypes
```

[4]:
```
CUSTOMER_ID                     object
COLLEGE                         object
DATA                           float64
INCOME                         float64
OVERCHARGE                       int64
LEFTOVER                         int64
HOUSE                          float64
```

```
LESSTHAN600k                      object
CHILD                              int64
JOB_CLASS                          int64
REVENUE                          float64
HANDSET_PRICE                      int64
OVER_15MINS_CALLS_PER_MONTH        int64
TIME_CLIENT                      float64
AVERAGE_CALL_DURATION              int64
REPORTED_SATISFACTION             object
REPORTED_USAGE_LEVEL              object
CONSIDERING_CHANGE_OF_PLAN        object
CHURNED                           object
dtype: object
```

1.2.1 *2.1 The data description*

- CUSTOMER_ID: A unique customer identifier (categorical)
- COLLEGE: (one or zero), is the customer college educated ? (categorical)
- DATA: Monthly data consumption in Mo (numerical)
- INCOME: Annual salary of the client (numerical)
- OVERCHARGE:Average overcharge per year (numerical)
- LEFTOVER: Average number of leftover minutes per month (numerical)
- HOUSE: Estimated value of the house (numerical)
- LESSTHAN600k: Is the value of the house smaller than 600K ? (catagorical)
- CHILD: The number of children (numerical)
- JOB_CLASS: Self reported type of job (categorical)
- REVENUE: Annual phone bill (numerical)
- HANDSET_PRICE: The price of the handset (phone) (numerical)
- OVER_15MINS_CALLS_PER_MONTH: Average number of long calls (more than 15 minutes) (numerical)
- TIME_CLIENT: The tenure in year (numerical)
- AVERAGE_CALL_DURATION: The average duration of a call (numerical)
- REPORTED_SATISFACTION: The reported level of satisfaction (categorical)
- REPORTED_USAGE_LEVEL: The self reported usage level (categorical)
- CONSIDERING_CHANGE_OF_PLAN: Self reported consideration whether to change operator (categorical)
- CHURNED: Did the customer stay or leave. This is the class (categorical)

1.2.2 *2.2 The data quality*

Before starting any analysis, it's important to guarantee the quality of the data. Especially, we are going to check if there are missing values:

[5]:
```
data.isna().sum()
```

[5]:

```
CUSTOMER_ID                    0
COLLEGE                        0
DATA                           0
INCOME                         0
OVERCHARGE                     0
LEFTOVER                       0
HOUSE                        635
LESSTHAN600k                 635
CHILD                          0
JOB_CLASS                      0
REVENUE                        0
HANDSET_PRICE                  0
OVER_15MINS_CALLS_PER_MONTH    0
TIME_CLIENT                    0
AVERAGE_CALL_DURATION          0
REPORTED_SATISFACTION          0
REPORTED_USAGE_LEVEL           0
CONSIDERING_CHANGE_OF_PLAN     0
CHURNED                        0
dtype: int64
```

There are 2 variables with missing values: HOUSE (the house value) and the LESSTHAN600K (is the house value smaller or higher tha 600K ?).

What is the type of the missing values ?

[6]:

```
# We retain only rows with missing values for the variable HOUSE
dataNa = data[data['HOUSE'].isna()]
lessthan600k = dataNa['LESSTHAN600k']

# The percentage of missing values in the column LESSTHAN600K
100*lessthan600k.isna().sum()/lessthan600k.shape[0]
```

[6]:

```
100.0
```

The variable **LESSTHAN600K** has missing values because the house were not evaluated (The value of **HOUSE** is missing).

Let's evaluate the percentage of missing values in the whole data set:

[7]:

```
100*lessthan600k.shape[0]/data.shape[0]
```

[7]:

```
5.3000584258409145
```

There are only **5.3%** of rows with a missing value.

Instead of deleting the rows with missing values, I choose to impute the missing values of the variable **HOUSE** with the **median**.

The best solution to handle these missing values is to order an evaluation of that dwelings.

```
[8]: data['HOUSE'] = data['HOUSE'].fillna(data['HOUSE'].median())
     data['LESSTHAN600k'] = np.where(data['HOUSE'] < 600000, 'True', 'False')
```

```
[9]: data.isna().sum()
```

```
[9]: CUSTOMER_ID                    0
     COLLEGE                        0
     DATA                           0
     INCOME                         0
     OVERCHARGE                     0
     LEFTOVER                       0
     HOUSE                          0
     LESSTHAN600k                   0
     CHILD                          0
     JOB_CLASS                      0
     REVENUE                        0
     HANDSET_PRICE                  0
     OVER_15MINS_CALLS_PER_MONTH    0
     TIME_CLIENT                    0
     AVERAGE_CALL_DURATION          0
     REPORTED_SATISFACTION          0
     REPORTED_USAGE_LEVEL           0
     CONSIDERING_CHANGE_OF_PLAN     0
     CHURNED                        0
     dtype: int64
```

Let's convert each variable in the appropriate data type:

```
[10]: data['CUSTOMER_ID'] = pd.Categorical(data['CUSTOMER_ID'])
      data['COLLEGE'] = pd.Categorical(data['COLLEGE'])
      data['LESSTHAN600k'] = pd.Categorical(data['LESSTHAN600k'])
      data['JOB_CLASS'] = pd.Categorical(data['JOB_CLASS'])
      data['REPORTED_SATISFACTION'] = pd.Categorical(data['REPORTED_SATISFACTION'])
      data['REPORTED_USAGE_LEVEL'] = pd.Categorical(data['REPORTED_USAGE_LEVEL'])
      data['CONSIDERING_CHANGE_OF_PLAN'] = pd.
       ↪Categorical(data['CONSIDERING_CHANGE_OF_PLAN'])
      data['CHURNED'] = pd.Categorical(data['CHURNED'])
```

1.3 3. Data exploratory analysis

Before the modeling, it's important to dive deep inside the data, to highlith the link or correlation between varaiables.

1.3.1 *3.1 The class variable exploration (CHURNED)*

First, we check if the data is unbalanced:

[11]:
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.catplot(x="CHURNED", kind="count", palette="ch:.25", data=data);
```

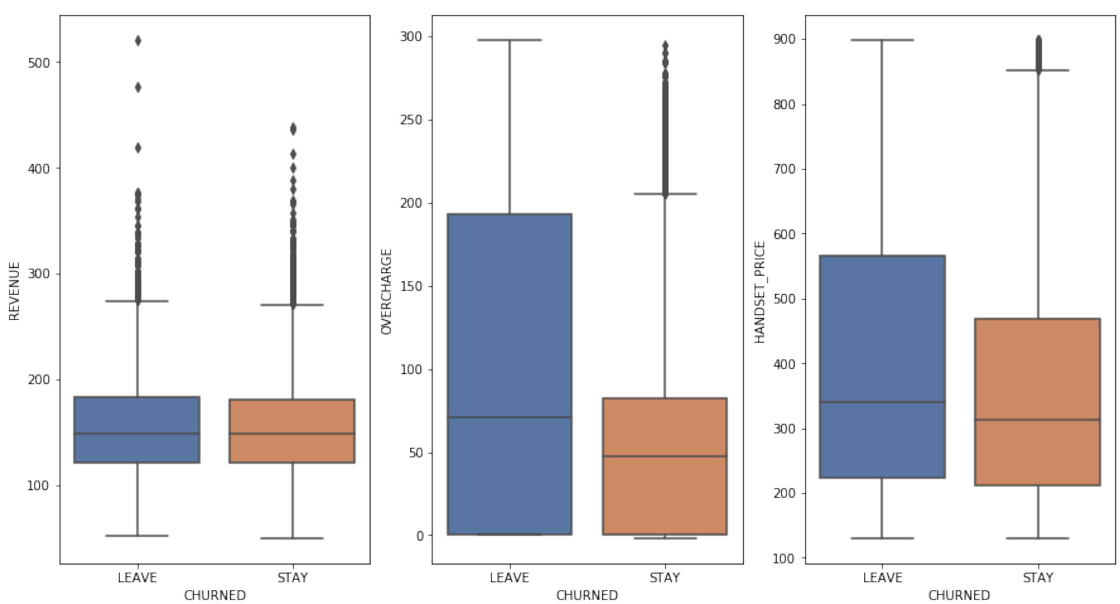
[12]:
```
stay = data[data['CHURNED'] == 'STAY'].shape

percent  = stay[0]*100/data.shape[0]
print("STAY percentage: ", round(percent,2),'%')
print("LEAVE percentage: ", round(100-percent,2),'%')
```

```
STAY percentage:  63.53 %
LEAVE percentage:  36.47 %
```

The data is **unbalanced**. The main risk is to build an overfitted model. To avoid this, the training data will be created with a **stratify partionning**.

1.3.2 *3.2 Numeric variables*

[13]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1,3,figsize=(15,8))
sns.set(style="ticks", color_codes=True)

sns.boxplot(x="CHURNED", y="REVENUE", data=data,  ax=ax1)
sns.boxplot(x="CHURNED", y="OVERCHARGE", data=data, ax=ax2);
sns.boxplot(x="CHURNED", y="HANDSET_PRICE",  data=data, ax=ax3);
```

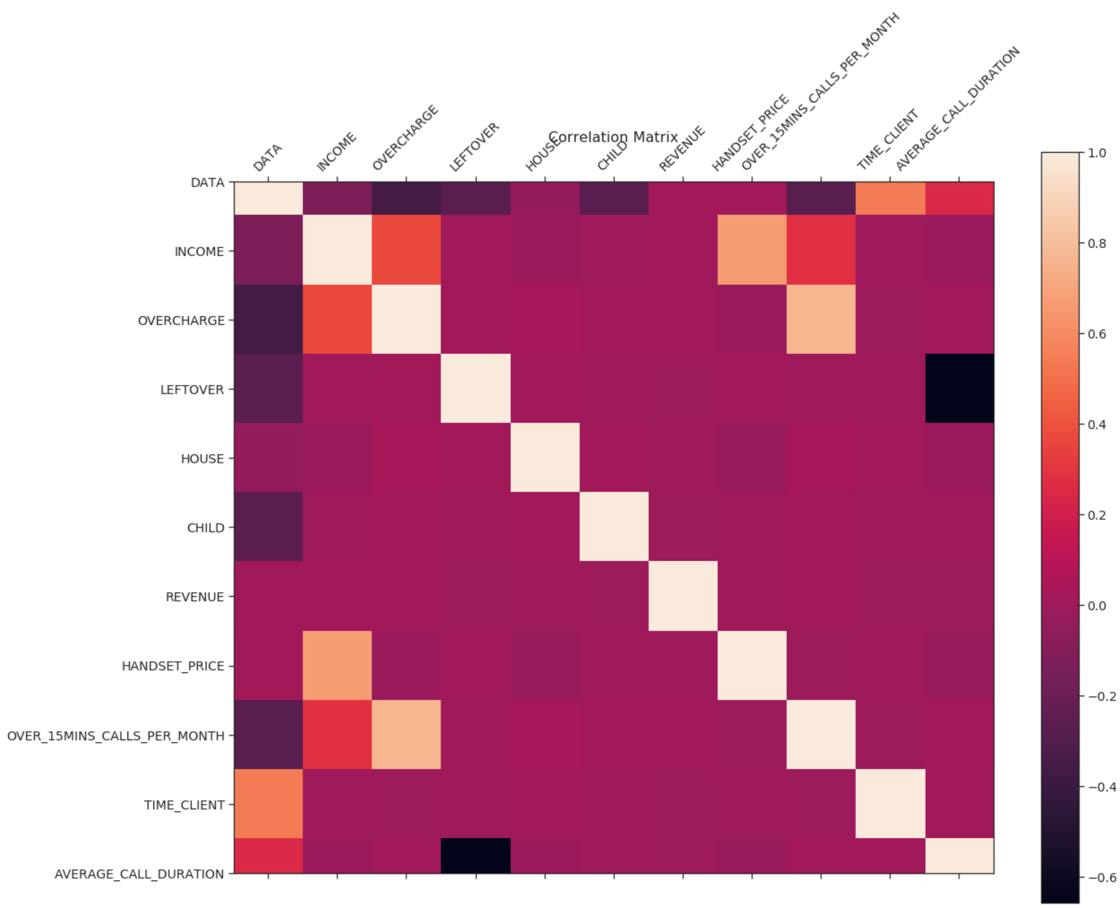


The first boxplot shows that, there is not a difference for the revenue (the median), customers who leave or stay are similar (in termes of phone bill).

The second shows that customers who leave are more overcharged (the median) than customers who stays.

The handset of customers who leave are more expensive than the ones of people who stay.

[14]:
```
dataNumeric = data.select_dtypes(exclude='category')
numericCorr = dataNumeric.corr(method='pearson')
f= plt.figure(figsize=(19, 15))
plt.matshow(numericCorr, fignum=f.number)
plt.xticks(range(dataNumeric.shape[1]), dataNumeric.columns, fontsize=14,␣
 ↪rotation=45)
plt.yticks(range(dataNumeric.shape[1]), dataNumeric.columns, fontsize=14)
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
plt.title('Correlation Matrix', fontsize=16);
plt.show()
```



We can draw some observations according to the table above: There is a strong correlation between:

- **HANDSET_PRICE** and **INCOME**
- **OVERCHARGE** and **OVER_15MINS_CALLS_PER_MONTH** (average number of long calls)

1.3.3 *3.3 Categorical variables*

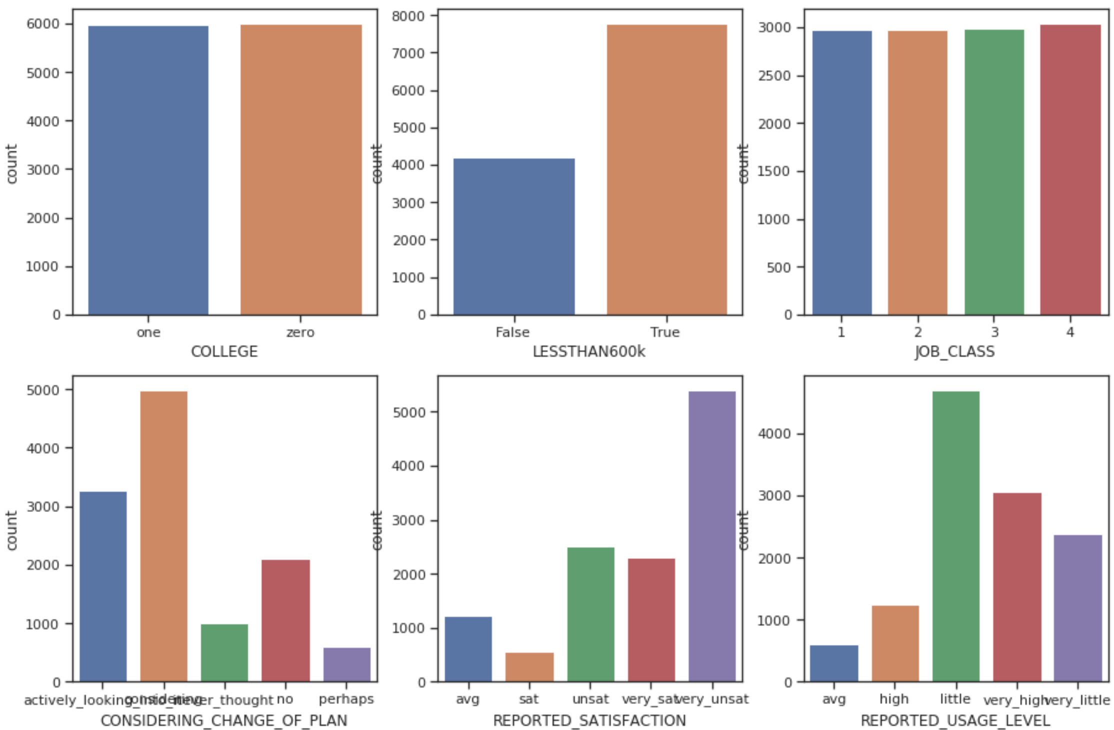
[15]:
```
dataCat = data.select_dtypes(include='category')
del dataCat['CUSTOMER_ID']
dataCat.describe().T
```

[15]:

	count	unique	top	freq
COLLEGE	11981	2	zero	6012
LESSTHAN600k	11981	2	True	7788
JOB_CLASS	11981	4	4	3045
REPORTED_SATISFACTION	11981	5	very_unsat	5397
REPORTED_USAGE_LEVEL	11981	5	little	4693
CONSIDERING_CHANGE_OF_PLAN	11981	5	considering	4981
CHURNED	11981	2	STAY	7612

[16]:
```
fig = plt.figure(figsize = (15,10))

ax1 = fig.add_subplot(2,3,1)
sns.countplot(data = dataCat, x = 'COLLEGE', ax=ax1)

ax2 = fig.add_subplot(2,3,2)
sns.countplot(data = dataCat, x = 'LESSTHAN600k', ax=ax2)

ax3 = fig.add_subplot(2,3,3)
sns.countplot(data = dataCat, x = 'JOB_CLASS', ax=ax3)

ax4 = fig.add_subplot(2,3,4)
sns.countplot(data = dataCat, x = 'CONSIDERING_CHANGE_OF_PLAN' , ax=ax4)

ax5 = fig.add_subplot(2,3,5)
sns.countplot(data = dataCat, x = 'REPORTED_SATISFACTION', ax=ax5)

ax6 = fig.add_subplot(2,3,6)
sns.countplot(data = dataCat, x = 'REPORTED_USAGE_LEVEL', ax=ax6)
```

[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff7a9e6fd10>
```



There are the same number of customers who went to college and those who were not. Is there any link between the customers who leaves and those who went or not to college ?

- Independance test (or **Chi Square test**) between **CONSIDERING_CHANGE_OF_PLAN** and **LESSTHAN600k**:

Ho: The variables **CONSIDERING_CHANGE_OF_PLAN** and **LESSTHAN600k** are independant

Ha: The variables are linked

[17]:
```
from scipy.stats import chi2_contingency

def chi2Test(contengencyTable):
    chi2, pvalue, dof, expected = chi2_contingency(contengencyTable)
    print("X-squered:", chi2, "p-Value:",pvalue, "degrees of freedom:", dof)
```

[18]:
```
observedData = pd.crosstab(dataCat['CONSIDERING_CHANGE_OF_PLAN'],␣
 ↪dataCat['LESSTHAN600k'], margins=False)
chi2Test(observedData)
```

```
X-squered: 11.937731802723693 p-Value: 0.017820364524039840 degrees of freedom: 4
```

As the **p-Value** is less than 0.05, then we rejet the null hypothesis (Ho). There is a link (a correlation) between **CONSIDERING_CHANGE_OF_PLAN** and **LESSTHAN600k**.

- What about the link between **REPORTED_SATISFACTION** and **REPORTED_SATISFACTION** ?

[19]:
```
observedData = pd.crosstab(dataCat['REPORTED_USAGE_LEVEL'],
    dataCat['REPORTED_SATISFACTION'], margins=False)
chi2Test(observedData)
```

```
X-squered: 31.628793930123376 p-Value: 0.011172192519596967 degrees of freedom:
16
```

The **p-Value** is less than 0.05. We can rejet the hypothesis that, **REPORTED_USAGE_LEVEL** and **REPORTED_SATISFACTION** are independant. That means there is a link between **REPORTED_USAGE_LEVEL** and **REPORTED_SATISFACTION**

1.3.4 *3.4 Data exploratory analysis conclusion*

The data exploratory analysis highlights the main features of the data. We are aware of that:

- The class (CHURNED) is unbalanced: We need to use stratified partinning and stratified cross-validation
- The are correlations between certain numerical variables and there links between certain categorical variables

1.4 4. The Modeling

In this part, we are going to train many machine learning models. Here are the steps we are going to follow:

- Data partioning: We are going split data in 3 parts. The training data (for model training), the test data (for model testing) and the validation data (for model comparison)
- The training and testing of machine learning models: We will use the cross-validation with k-folds
- The validation and selection of the best model based on the Air Under Curve of the ROC curve

1.4.1 *4.1 The data sampling*

We mentionned that, the class (CHURNED) was unbalanced. We use the **stratify sampling** to maintain the two subgroups (LEAVE and STAY) in all partion.

[20]:
```
from sklearn.model_selection import train_test_split

dataset = data.copy()
del dataset['CUSTOMER_ID']

predictors = dataset.loc[:, dataset.columns != 'CHURNED']
```

```
#categorical data encoding (one hot)
predictors = pd.get_dummies(predictors)

#Retain only values
X    = predictors.values

outcome      = dataset.loc[:, dataset.columns == 'CHURNED']
outcome  = pd.get_dummies(outcome)

# 1=LEAVE and 0 = STAY
Y         = outcome["CHURNED_LEAVE"]

# The training sample is 60% of the dataset
X_train, XVal_test, y_train, yval_test = train_test_split(X, Y, test_size=0.4,␣
 ↪stratify=Y)

#The test sample is 70% of the remaining 40% (= 0.7*0.4=28% of the initial␣
 ↪dataset)
X_test, X_val, y_test, y_val = train_test_split(XVal_test, yval_test,␣
 ↪test_size=0.3, stratify=yval_test)

print("The training sample: ", X_train.shape)
print("The test sample: ",X_test.shape)
print("The validation sample: ",X_val.shape)
```

```
The training sample:  (7188, 34)
The test sample:  (3355, 34)
The validation sample:  (1438, 34)
```

1.4.2 *4.2 The models training*

We are going to use many families of models. When there correlations or links between predictors some methods will underperform (linear methods especially). These are the methods we will be training:

- Logistic regression: Linear method, easy to train, will be used to compare others methods
- Gradient boosting
- The decision tree
- Random Forest
- Multi layer perceptron

For this methods, we will perform the hyper parameters searching with a **Grid search** and training the models using a cross validation.

The models will training using a repeated cross-validation. For each cross-validation, the fold size will be different. This is intended to minimise the variance of the Cross validation parameters:

[21]:

```python
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

CrossValidationFolds = np.array([5, 7, 10, 13, 16, 20])
```

[23]:

```python
def myGridSearch(model, grid_params, kfolds, trainX, trainY, testX, testY):
    if model is None or kfolds is None:
        return

    BestTrainScore = list()
    TestScores = list()
    TestScoresStd = list()
    auc     = list()
    BestParams  = list()
    maxAuc     = 0
    bestModel = model

    # We have many draws:
    for k in kfolds:
        # Model building with a k number of Folds
        gridModel = GridSearchCV(model, grid_params,error_score='raise', cv=k)
        gridModel.fit(trainX, trainY)


        BestTrainScore.append(gridModel.best_score_)
        TestScores.append(gridModel.score(testX, testY))

        # ROC curve and AUC
        predictions = gridModel.predict(testX)
        thisAuc = roc_auc_score(testY,predictions)
        auc.append(thisAuc)

        #We retain the best model = the highest AUC
        if thisAuc > maxAuc:
            bestModel = gridModel
            maxAuc = thisAuc

    #Display the results in a graphic
    fig, ax = plt.subplots()
    ax.plot(kfolds, BestTrainScore, 'b--', label='Train score')
    ax.plot(kfolds, TestScores, 'r--', label='Test scores')
    ax.plot(kfolds, auc, 'g--', label='AUC')
    leg = ax.legend();
```

```
    print("Mean train score: %0.2f"%np.mean(BestTrainScore),
          " / STD train score: %0.2f"% np.std(BestTrainScore),
          " / Max train score: %0.2f"% np.max(BestTrainScore),
          " / Min train score: %0.2f"% np.min(BestTrainScore))
    print("Mean test score: %0.2f"%np.mean(TestScores),
          " / STD test score: %0.2f"% np.std(TestScores),
          " / Max test score: %0.2f"% np.max(TestScores),
          " / Min test score: %0.2f"% np.min(TestScores))
    print("Mean AUC: %0.2f"% np.mean(auc),
          " / Std AUC: %0.2f"% np.std(auc),
          " / Max AUC: %0.2f"% maxAuc,
          " / Min AUC: %0.2f"% np.min(auc))

    return bestModel
```

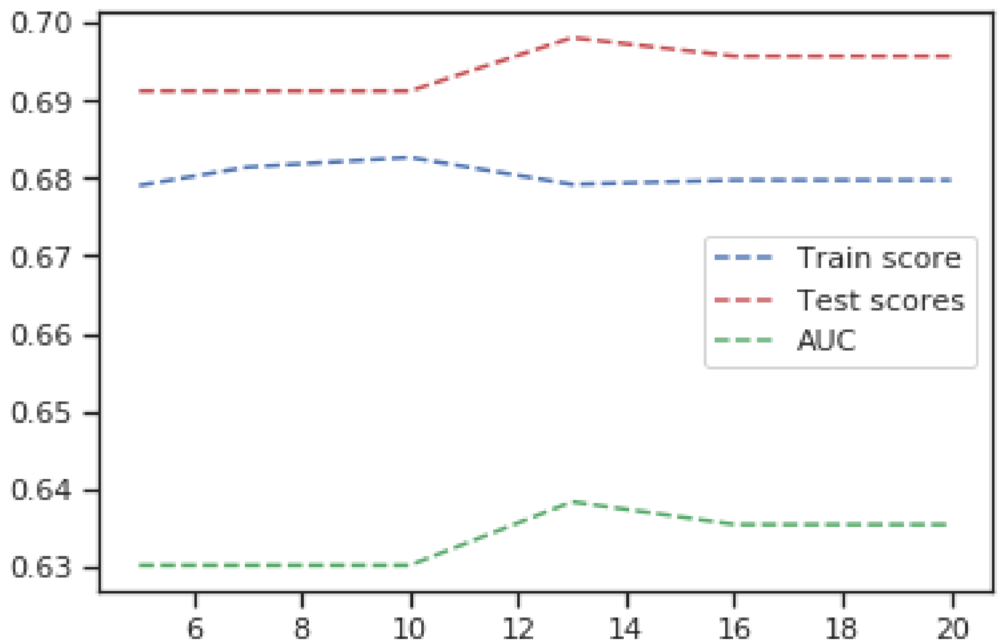
- **The Logistic regression**

[24]:
```
from sklearn.linear_model import LogisticRegression
import pickle

grid_params = {
    'C': [0.001, 0.005, 0.01, 0.05, 1,1.5],
    'tol': [1e-5,1e-4,1e-3,1e-2,1e-1],
    "max_iter": [50,100,200,300]
}

lr = LogisticRegression(solver = 'liblinear',verbose=0)
modelLR = myGridSearch(lr, grid_params, CrossValidationFolds, X_train, y_train,␣
 ↪X_test, y_test)
print("The best hyper parameters values: ", modelLR.best_params_)

fpickle = open('modelLR.pkl', 'wb')
pickle.dump(modelLR, fpickle)
```

```
/home/yefangon/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
/home/yefangon/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
/home/yefangon/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
/home/yefangon/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
```

```
iterations.
  "the number of iterations.", ConvergenceWarning)
/home/yefangon/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
/home/yefangon/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
Mean train score: 0.68  / STD train score: 0.00  / Max train score: 0.68  / Min
train score: 0.68
Mean test score: 0.69  / STD test score: 0.00  / Max test score: 0.70  / Min
test score: 0.69
Mean AUC: 0.63  / Std AUC: 0.00  / Max AUC: 0.64  / Min AUC: 0.63
The best hyper parameters values:  {'C': 0.001, 'max_iter': 100, 'tol': 1e-05}
```

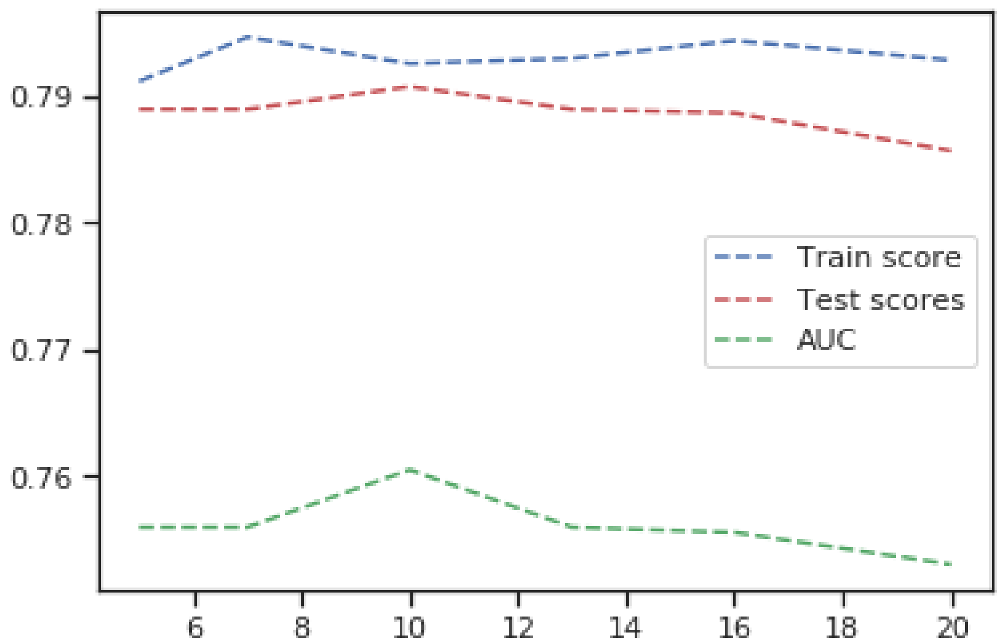


- **The gradient boosting**

[25]:
```
from sklearn.ensemble import GradientBoostingClassifier

grid_params = {
    'loss': ['deviance', 'exponential'],
    'learning_rate': [0.001,0.1],
    "n_estimators": [100,200,300],
    "max_depth": [3,5,10]
```

```
}

gb = GradientBoostingClassifier(verbose=0)
modelGB = myGridSearch(gb, grid_params, CrossValidationFolds, X_train, y_train,␣
↪X_test, y_test)
print("The best hyper parameters values: ", modelGB.best_params_)

fpickle = open('modelGB.pkl', 'wb')
pickle.dump(modelGB, fpickle)
```

```
Mean train score: 0.79  / STD train score: 0.00  / Max train score: 0.79  / Min
train score: 0.79
Mean test score: 0.79  / STD test score: 0.00  / Max test score: 0.79  / Min
test score: 0.79
Mean AUC: 0.76  / Std AUC: 0.00  / Max AUC: 0.76  / Min AUC: 0.75
The best hyper parameters values:  {'learning_rate': 0.1, 'loss': 'exponential',
'max_depth': 5, 'n_estimators': 300}
```

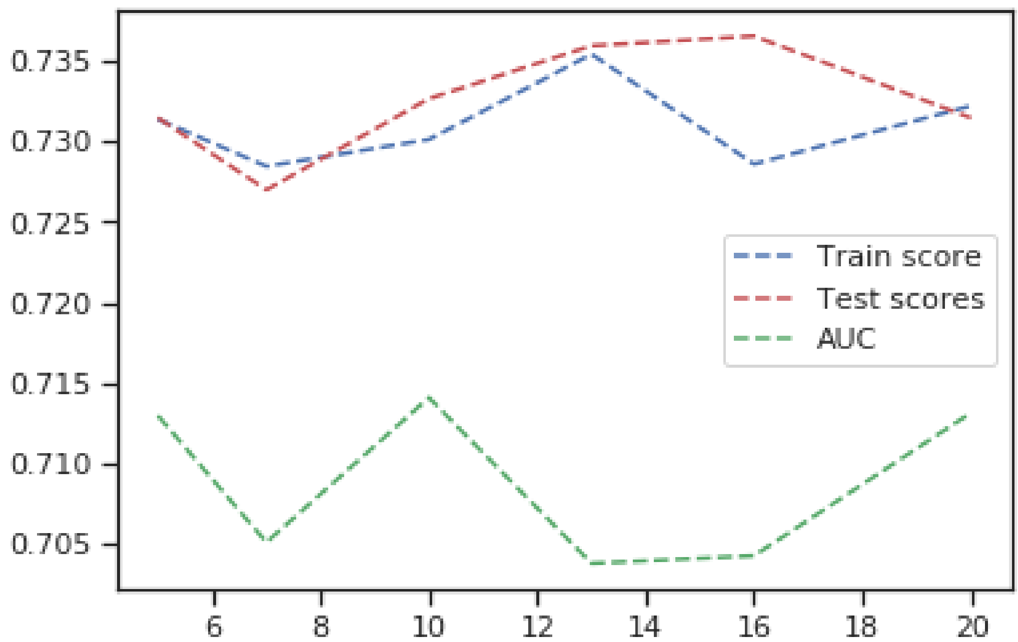


- **The decision tree**

[26]:
```
from sklearn import tree

tr = tree.DecisionTreeClassifier()
grid_params = {'criterion': ['gini', 'entropy'],"max_depth": [ 2, 5,7,8,10],␣
↪"min_samples_split": [2, 3, 5, 10]}
```

```
modelTree = myGridSearch(tr, grid_params, CrossValidationFolds, X_train,␣
 ↪y_train, X_test, y_test)

print("The best hyper parameters values: ",modelTree.best_params_)

fpickle = open('modelTree.pkl', 'wb')
pickle.dump(modelTree, fpickle)
```

```
Mean train score: 0.73  / STD train score: 0.00  / Max train score: 0.74  / Min
train score: 0.73
Mean test score: 0.73  / STD test score: 0.00  / Max test score: 0.74  / Min
test score: 0.73
Mean AUC: 0.71  / Std AUC: 0.00  / Max AUC: 0.71  / Min AUC: 0.70
The best hyper parameters values:  {'criterion': 'gini', 'max_depth': 7,
'min_samples_split': 3}
```

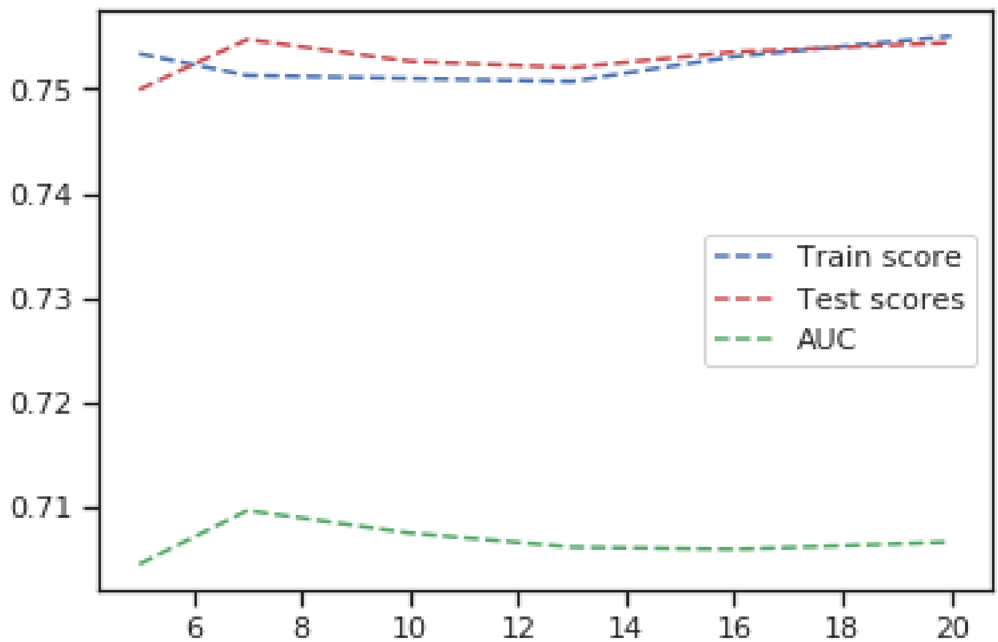


- **The random Forest**

[27]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(verbose = 0)

grid_params = {"n_estimators": [10,20,50, 100,200],
               "criterion": ['gini', 'entropy'],
               'max_depth': [2,3,5,10],
               "class_weight": ['balanced', 'balanced_subsample', None]
```

```
                }
modelRF = myGridSearch(rf, grid_params, CrossValidationFolds, X_train, y_train,␣
 ↪X_test, y_test)
print("The best hyper parameters values: ",modelRF.best_params_)

fpickle = open('modelRF.pkl', 'wb')
pickle.dump(modelRF, fpickle)
```

```
Mean train score: 0.75  / STD train score: 0.00  / Max train score: 0.76  / Min
train score: 0.75
Mean test score: 0.75  / STD test score: 0.00  / Max test score: 0.75  / Min
test score: 0.75
Mean AUC: 0.71  / Std AUC: 0.00  / Max AUC: 0.71  / Min AUC: 0.70
The best hyper parameters values:  {'class_weight': None, 'criterion': 'gini',
'max_depth': 10, 'n_estimators': 200}
```



- **The Perceptron multi layer**

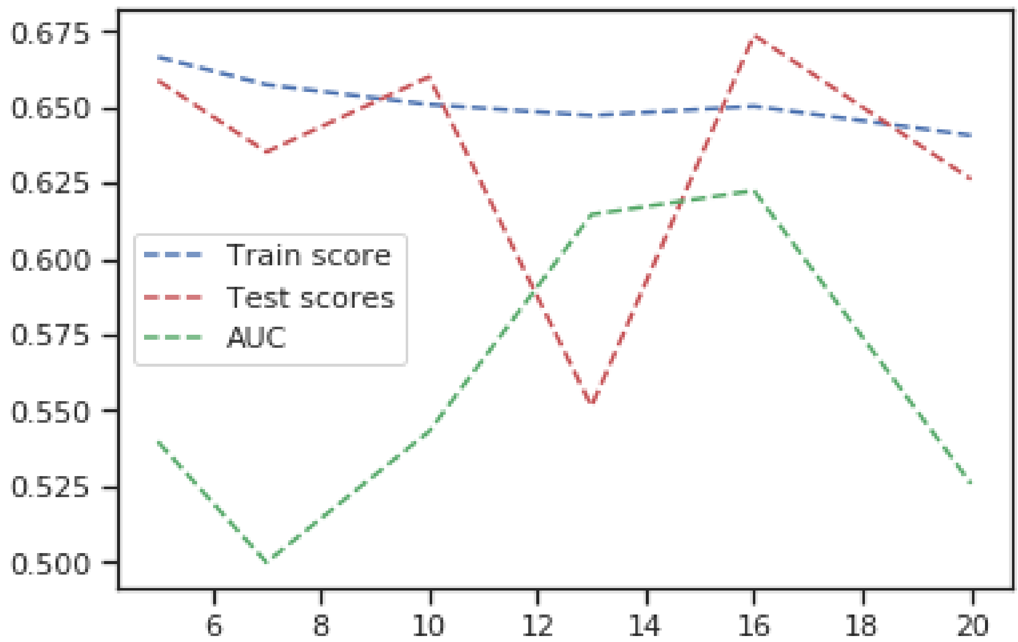
The perceptron multi layer is a fully connected neuronal network with only one hidden layer. The model is trained using backprobagation

[26]:
```
from sklearn.neural_network import MLPClassifier
import pickle
mlp = MLPClassifier()
grid_params = {
    "hidden_layer_sizes": [50, 100, 200, 500],
```

```
    "activation": ['logistic', 'tanh', 'relu'],
    "solver": ['lbfgs', 'sgd', 'adam'],
    "alpha": [0.00001, 0.0001, 0.001, 0.01]
}

modelMlp = myGridSearch(mlp, grid_params, CrossValidationFolds, X_train,␣
↪y_train, X_test, y_test)
print("The best hyper parameters values: ",modelMlp.best_params_)
```

```
Mean train score: 0.65  / STD train score: 0.01  / Max train score: 0.67  / Min
train score: 0.64
Mean test score: 0.63  / STD test score: 0.04  / Max test score: 0.67  / Min
test score: 0.55
Mean AUC: 0.56  / Std AUC: 0.05  / Max AUC: 0.62  / Min AUC: 0.50
The best hyper parameters values:  {'activation': 'relu', 'alpha': 0.01,
'hidden_layer_sizes': 50, 'solver': 'adam'}
```

```
        ␣
↪---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call␣
↪last)

        <ipython-input-26-409eb0a2674f> in <module>
         13
         14 fpickle = open('modelMLP.pkl', 'wb')
    ---> 15 pickle.dump(modelMLP, fpickle)


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



[27]:
```
fpickle = open('modelMLP.pkl', 'wb')
pickle.dump(modelMlp, fpickle)
```

1.5 4.3 The model selection

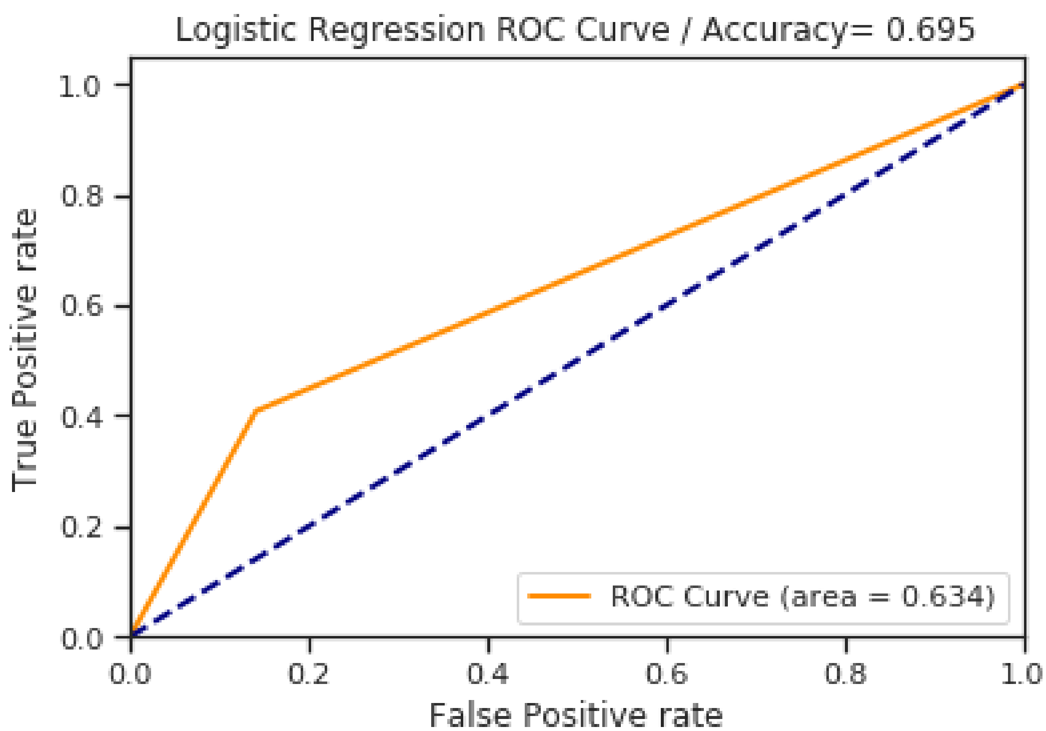
After training the models, we are going to use the validation data set to select the best model. The best model has the highest value of the AUC

[28]:
```
def myROCCurvePlot(xVal, yVal, model, title='ROC Curve'):
    # The score:
    predict = model.predict(xVal)
    score = model.score(xVal, yVal)
    #False positive et True positives
    fp, vp,_ = roc_curve(yVal, predict)
    lw =2
    #A.U.C
    auc = roc_auc_score(y_val,predict)
    plt.figure()
    plt.plot(fp, vp, color='darkorange',lw=lw, label='ROC Curve (area = %0.3f)'␣
 ↪% auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive rate')
```

```
    plt.ylabel('True Positive rate')
    plt.title(title + ' / Accuracy= %0.3f'% score)
    plt.legend(loc="lower right")
    plt.show()

    return auc
```

- Logistic regression

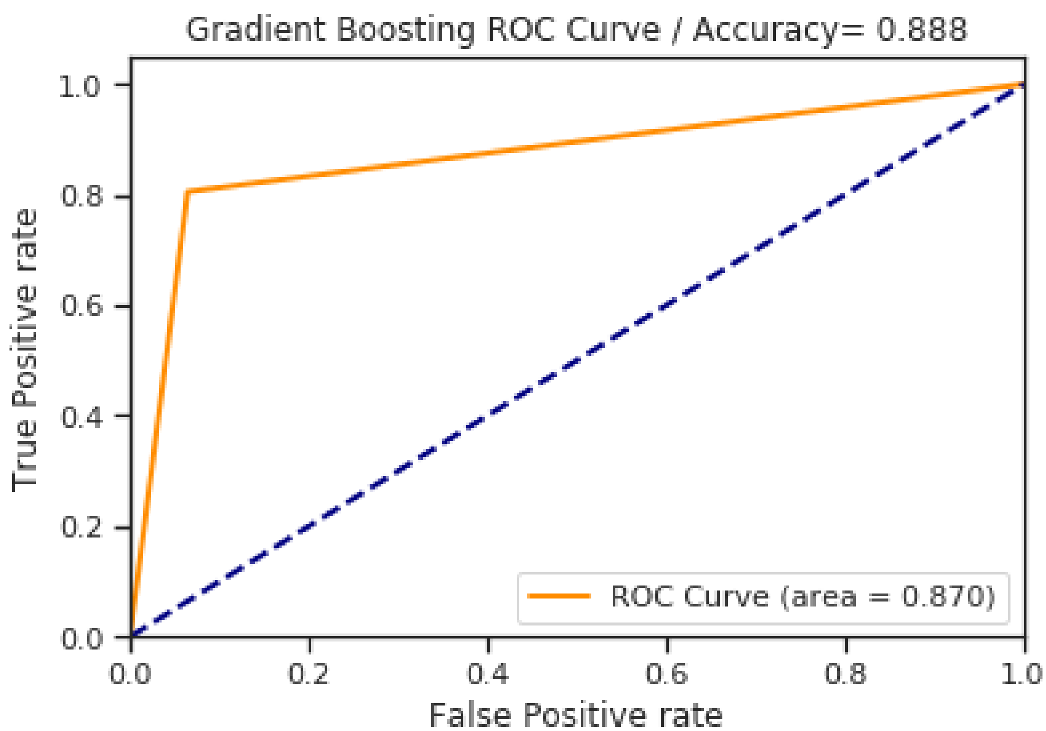
[47]:
```
m = pickle.load(open('modelLR.pkl','rb'))


aucLR = myROCCurvePlot(X_val, y_val, m, 'Logistic Regression ROC Curve')

maxAUC = aucLR
name='Logistic regression'
bestModel=m

print("Air Under Curve: %0.3f"% aucLR)
```




```
Air Under Curve: 0.634
```

- **The gradient boosting**

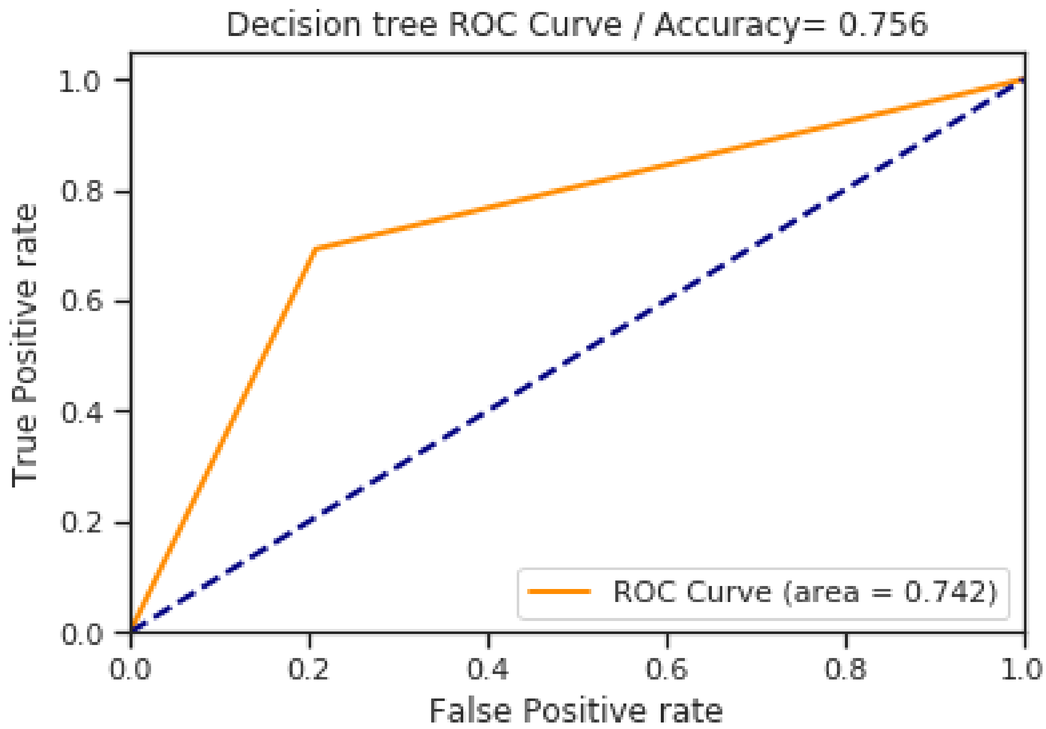
[48]:
```python
m = pickle.load(open('modelGB.pkl','rb'))
aucGB = myROCCurvePlot(X_val, y_val, m, 'Gradient Boosting ROC Curve')
print("Air Under Curve: %0.3f"% aucGB)

if aucGB > maxAUC:
    bestModel=m
    maxAUC=aucGB
    name='Gradient Boosting'
```




```
Air Under Curve: 0.870
```

- **The decision tree (CART)**

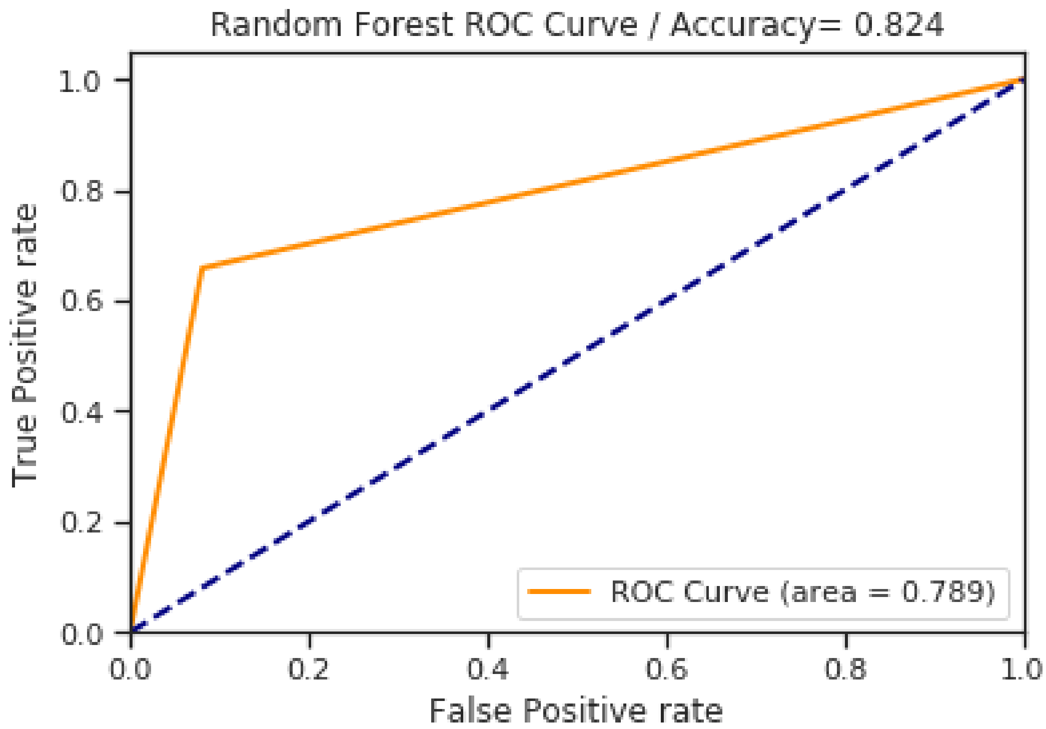
[49]:
```python
m = pickle.load(open('modelTree.pkl','rb'))
aucTR = myROCCurvePlot(X_val, y_val, m, 'Decision tree ROC Curve')
print("Air Under Curve: %0.3f"% aucTR)

if aucTR > maxAUC:
    bestModel=m
    maxAUC=aucTR
    name='The CART'
```




```
Air Under Curve: 0.742
```

- **The random forest**

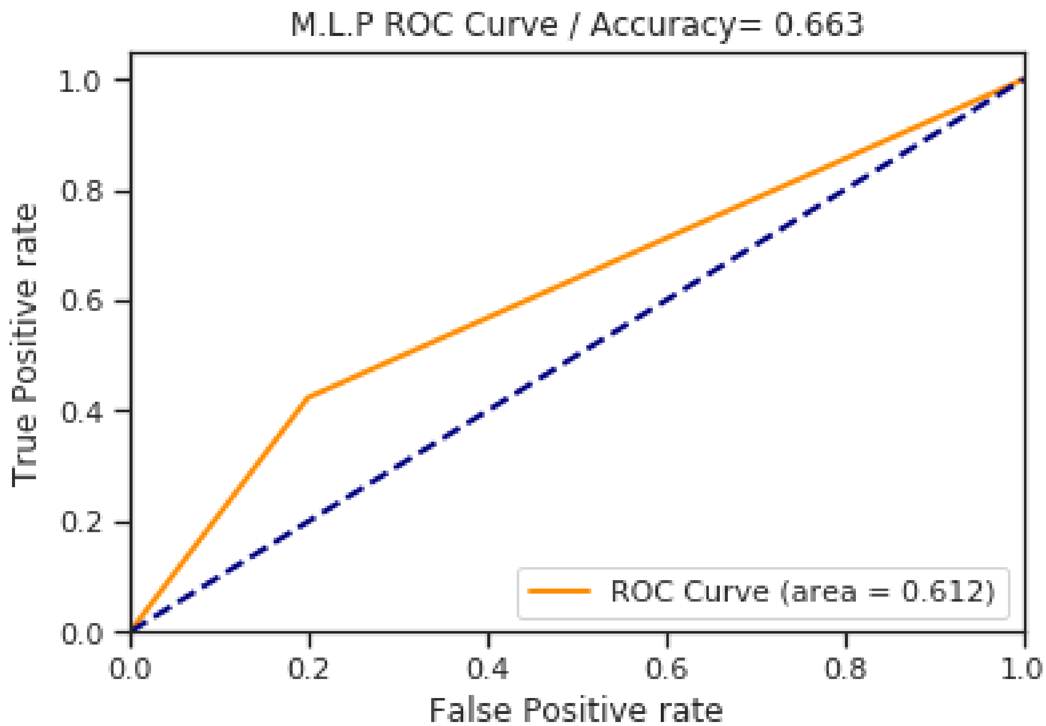
[50]:
```
m = pickle.load(open('modelRF.pkl','rb'))

aucRF = myROCCurvePlot(X_val, y_val, m, 'Random Forest ROC Curve')
print("Air Under Curve: %0.3f"% aucRF)

if aucRF > maxAUC:
    bestModel=m
    maxAUC=aucRF
    name='The random Forest'
```




```
Air Under Curve: 0.789
```

- **The multi layer perceptron**

[51]:
```python
m = pickle.load(open('modelMLP.pkl','rb'))

aucMLP = myROCCurvePlot(X_val, y_val, m, 'M.L.P ROC Curve')
print("Air Under Curve: %0.3f"% aucMLP)

if aucMLP > maxAUC:
    bestModel=m
    maxAUC=aucMLP
    name='Perceptron multilayer'
```




```
Air Under Curve: 0.612
```

The best model is the one which has the highest **AUC**.

Finally, I save the best model found.

[53]:
```
fpickle = open('bestModel.pkl', 'wb')
pickle.dump(bestModel, fpickle)
```

The most performing model according to the AUC is:

[54]:
```
print(name)
```

```
Gradient Boosting
```

2 5. What client to client ?

Contacting a customer has a fixed cost of 10 Euros, it's expensive! TELCO must contact firstly customers who have been classified as to leave.

3 6. What is the maximum discount to proposed ?

The discount will be proposed to customers called by TELCO agents. A call has a fixed cost of 10 Euros.

The discount must maximise the TELCO profit.

I proposed to base the discount amount on the **Overcharge** amount per year. That will allow TELCO to secure their profit because it's still relay on the phone bill

The formula is: **(OVERCHARGE - 10) * LEAVE_PROBABILITY**.

The higher the probability of a customer to leave, the higher the amount of the discount.

If the churn label is LEAVE and the calculated discount amount is 0, then don't call the customer (CLIENT TO CALL = NO).

[ ]:
```

```