Projeto04

July 3, 2019

1 Data Science Academy

2 Formacao Cientista de Dados

2.1 Projeto 04 - Prevendo Customer Churn em Operadoras de Telecom

Este notebook contém as seguintes fases para a análise dos modelos de Machine Learning - Pré-Processamento - Criação do Modelo de Machine Learning - Validação e Otimização do Modelo - Previsão e Relatorios

2.2 Definição do Problema de Negócio

Prever o Customer Churn em uma Operadora de Telecom.

Os datasets de treino e de teste estão no diretório /data em anexo a este projeto. O objetivo e criar um modelo de aprendizagem de maquina que possa prever se um cliente pode ou nao cancelar seu plano e qual a probabilidade de isso ocorrer. O cabecalho do dataset e uma descricao do tipo de informacao em cada coluna.

2.2.1 Informações sobre os atributos:

Unnamed
state
account_length
area_code
international_plan
voice_mail_plan
number_vmail_messages
total_day_minutes
total_day_calls
total_day_charge
total_eve_minutes
total_eve_calls
total_eve_charge
total_night_minutes
total_night_calls
total_night_charge
total_intl_minutes

total_intl_calls
total_intl_charge
number_customer_service_calls
churn

2.2.2 GOAL

- Os arquivos projeto4_telecom_treino.csv e projeto4_telecom_teste.csv estão anexos no diretório /data
- Objetivo é prever o Customer Churn em uma Operadora de Telecom (coluna CHURN)
- O modelo é avalidado pelo SCORE e ROC Curve / AUC

2.3 Extraindo e Carregando os Dados

In [1]:
```
# Importando bibliotecas que serao utilizadas neste projeto
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import seaborn as sns
import itertools
import matplotlib.pyplot as plt
%matplotlib inline

# Models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from xgboost import XGBClassifier

# Stats
from scipy import stats
from scipy.stats import skew, norm

#Misc
from sklearn.preprocessing import scale, StandardScaler, MinMaxScaler
from sklearn.model_selection import KFold, cross_val_score, train_test_split, GridSear
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, r
from sklearn.metrics import precision_recall_curve, f1_score, auc, average_precision_s
from imblearn.over_sampling import SMOTE

# Ignore useless warnings
import warnings
```

```
warnings.filterwarnings(action="ignore")
pd.set_option('display.max_columns', None)
```

```
In [2]: # Carregando arquivo csv usando Pandas
        train = pd.read_csv("data/dataset_treino.csv", header = 0)
        test = pd.read_csv("data/dataset_teste.csv", header = 0)
```

2.4 Análise Exploratória de Dados

```
In [3]: # Visualizando o shape do dataset
        # Dataset de Treino (3.333 linhas x 21 colunas)
        # Dataset de Teste (1.667 linhas x 21 colunas)
        train.shape, test.shape
```

```
Out[3]: ((3333, 21), (1667, 21))
```

```
In [4]: # Visualizando as 5 primeiras linhas do dataset de treino
        train.head()
```

```
Out[4]:    Unnamed: 0 state  account_length      area_code international_plan  \
        0           1    KS             128  area_code_415                no
        1           2    OH             107  area_code_415                no
        2           3    NJ             137  area_code_415                no
        3           4    OH              84  area_code_408               yes
        4           5    OK              75  area_code_415               yes

          voice_mail_plan  number_vmail_messages  total_day_minutes  total_day_calls  \
        0             yes                     25              265.1              110
        1             yes                     26              161.6              123
        2              no                      0              243.4              114
        3              no                      0              299.4               71
        4              no                      0              166.7              113

           total_day_charge  total_eve_minutes  total_eve_calls  total_eve_charge  \
        0             45.07              197.4               99             16.78
        1             27.47              195.5              103             16.62
        2             41.38              121.2              110             10.30
        3             50.90               61.9               88              5.26
        4             28.34              148.3              122             12.61

           total_night_minutes  total_night_calls  total_night_charge  \
        0                244.7                 91               11.01
        1                254.4                103               11.45
        2                162.6                104                7.32
        3                196.9                 89                8.86
        4                186.9                121                8.41

           total_intl_minutes  total_intl_calls  total_intl_charge  \
        0                10.0                 3               2.70
```

```
1                13.7                3              3.70
2                12.2                5              3.29
3                 6.6                7              1.78
4                10.1                3              2.73

   number_customer_service_calls churn
0                              1    no
1                              1    no
2                              0    no
3                              2    no
4                              3    no
```

In [5]:
```
# Visualizando as 5 primeiras linhas do dataset de teste
test.head()
```

Out[5]:
```
   Unnamed: 0 state  account_length      area_code international_plan  \
0           1    HI             101  area_code_510                 no
1           2    MT             137  area_code_510                 no
2           3    OH             103  area_code_408                 no
3           4    NM              99  area_code_415                 no
4           5    SC             108  area_code_415                 no

  voice_mail_plan  number_vmail_messages  total_day_minutes  total_day_calls  \
0              no                      0               70.9              123
1              no                      0              223.6               86
2             yes                     29              294.7               95
3              no                      0              216.8              123
4              no                      0              197.4               78

   total_day_charge  total_eve_minutes  total_eve_calls  total_eve_charge  \
0             12.05              211.9               73             18.01
1             38.01              244.8              139             20.81
2             50.10              237.3              105             20.17
3             36.86              126.4               88             10.74
4             33.56              124.0              101             10.54

   total_night_minutes  total_night_calls  total_night_charge  \
0                236.0                 73               10.62
1                 94.2                 81                4.24
2                300.3                127               13.51
3                220.6                 82                9.93
4                204.5                107                9.20

   total_intl_minutes  total_intl_calls  total_intl_charge  \
0                10.6                 3               2.86
1                 9.5                 7               2.57
2                13.7                 6               3.70
3                15.7                 2               4.24
```

```
4                 7.7                 4               2.08

   number_customer_service_calls churn
0                              3    no
1                              0    no
2                              1    no
3                              1    no
4                              2    no
```

```
In [6]: # Sumario geral das features do dataset de treino
        train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
Unnamed: 0                      3333 non-null int64
state                           3333 non-null object
account_length                  3333 non-null int64
area_code                       3333 non-null object
international_plan              3333 non-null object
voice_mail_plan                 3333 non-null object
number_vmail_messages           3333 non-null int64
total_day_minutes               3333 non-null float64
total_day_calls                 3333 non-null int64
total_day_charge                3333 non-null float64
total_eve_minutes               3333 non-null float64
total_eve_calls                 3333 non-null int64
total_eve_charge                3333 non-null float64
total_night_minutes             3333 non-null float64
total_night_calls               3333 non-null int64
total_night_charge              3333 non-null float64
total_intl_minutes              3333 non-null float64
total_intl_calls                3333 non-null int64
total_intl_charge               3333 non-null float64
number_customer_service_calls   3333 non-null int64
churn                           3333 non-null object
dtypes: float64(8), int64(8), object(5)
memory usage: 546.9+ KB
```

```
In [7]: # Sumario geral das features do dataset de treino
        test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1667 entries, 0 to 1666
Data columns (total 21 columns):
Unnamed: 0                      1667 non-null int64
state                           1667 non-null object
account_length                  1667 non-null int64
```

```
area_code                       1667 non-null object
international_plan              1667 non-null object
voice_mail_plan                 1667 non-null object
number_vmail_messages           1667 non-null int64
total_day_minutes               1667 non-null float64
total_day_calls                 1667 non-null int64
total_day_charge                1667 non-null float64
total_eve_minutes               1667 non-null float64
total_eve_calls                 1667 non-null int64
total_eve_charge                1667 non-null float64
total_night_minutes             1667 non-null float64
total_night_calls               1667 non-null int64
total_night_charge              1667 non-null float64
total_intl_minutes              1667 non-null float64
total_intl_calls                1667 non-null int64
total_intl_charge               1667 non-null float64
number_customer_service_calls   1667 non-null int64
churn                           1667 non-null object
dtypes: float64(8), int64(8), object(5)
memory usage: 273.6+ KB
```

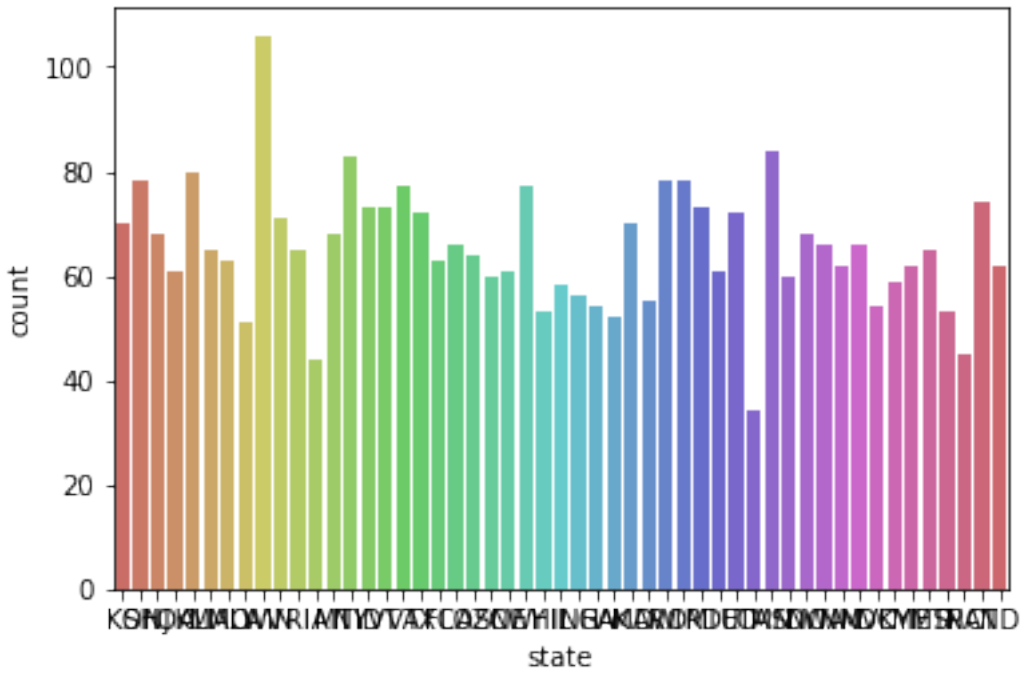
Vamos remover a coluna Unnamed: 0 pois se trata de um ID

```
In [8]: train.drop(columns = "Unnamed: 0", inplace = True)
        test.drop(columns = "Unnamed: 0", inplace = True)
```

2.4.1 Analise Exploratoria das variaveis do tipo object

Como está bem distribuido os registros por cada estado, esta variavel sera removida

```
In [9]: sns.countplot(x='state', data=train, palette='hls')
        plt.show()
```



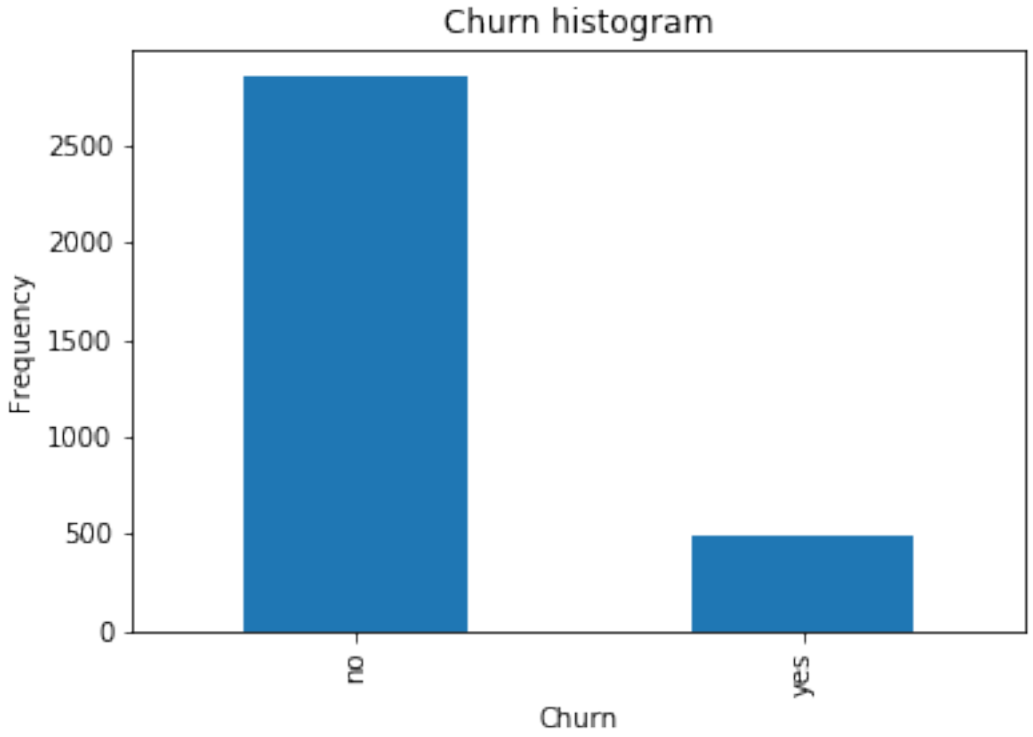

```
In [10]: train.drop(columns = "state", inplace = True)
         test.drop(columns = "state", inplace = True)

In [11]: train['area_code'].value_counts()

Out[11]: area_code_415    1655
         area_code_510     840
         area_code_408     838
         Name: area_code, dtype: int64

In [12]: #train.drop(columns = "area_code", inplace = True)
         #test.drop(columns = "area_code", inplace = True)

In [13]: train['international_plan'].value_counts()

Out[13]: no     3010
         yes     323
         Name: international_plan, dtype: int64

In [14]: train['voice_mail_plan'].value_counts()

Out[14]: no     2411
         yes     922
         Name: voice_mail_plan, dtype: int64
```

Verificando as caracteristicas estatisticas da variavel target “churn”

```
In [15]: # Visualizando a distribuição das classes (variavel TARGET)
         pd.value_counts(train['churn']).plot.bar()
         plt.title('Churn histogram')
         plt.xlabel('Churn')
         plt.ylabel('Frequency')

         # Visualizando um df com quantidade e percentual da variavel TARGET
         df = pd.DataFrame(train['churn'].value_counts())
         df['%'] = 100*df['churn']/train.shape[0]
         df
```

Out[15]:

	churn	%
no	2850	85.508551
yes	483	14.491449



Vamos transformar algumas variaveis categoricas em numericas e fazer analises estatisticas

```
In [16]: # Transformando algumas variaveis string para inteiro
         cleanup_nums = {"area_code":                {"area_code_415": 415, "area_code_510"
                         "international_plan":       {"no": 0, "yes": 1},
                         "voice_mail_plan":          {"no": 0, "yes": 1},
                         "churn":                    {"no": 0, "yes": 1}} # 0 para ativo e

         train.replace(cleanup_nums, inplace=True)
```

```
test.replace(cleanup_nums, inplace=True)
train.head()
```

Out[16]:

	account_length	area_code	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minutes	total_eve_calls	total_eve_charge	total_night_minutes	total_night_calls	total_night_charge	total_intl_minutes	total_intl_calls	total_intl_charge	number_customer_service_calls	churn
0	128	415	0	1	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1	0
1	107	415	0	1	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1	0
2	137	415	0	0	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0	0
3	84	408	1	0	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	0
4	75	415	1	0	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	0

Verificando valores missing nos datasets de treino e teste

```
In [17]: def percent_missing(df):
             data = pd.DataFrame(df)
```

```
            df_cols = list(pd.DataFrame(data))
            dict_x = {}
            for i in range(0, len(df_cols)):
                dict_x.update({df_cols[i]: round(data[df_cols[i]].isnull().mean()*100,2)})

            return dict_x
```

In [18]:
```
# Dados missing no dataset de treino
missing = percent_missing(train)
df_miss = sorted(missing.items(), key=lambda x: x[1], reverse=True)
print('Percent of missing data')
df_miss[0:50]
```

```
Percent of missing data
```

Out[18]:
```
[('account_length', 0.0),
 ('area_code', 0.0),
 ('international_plan', 0.0),
 ('voice_mail_plan', 0.0),
 ('number_vmail_messages', 0.0),
 ('total_day_minutes', 0.0),
 ('total_day_calls', 0.0),
 ('total_day_charge', 0.0),
 ('total_eve_minutes', 0.0),
 ('total_eve_calls', 0.0),
 ('total_eve_charge', 0.0),
 ('total_night_minutes', 0.0),
 ('total_night_calls', 0.0),
 ('total_night_charge', 0.0),
 ('total_intl_minutes', 0.0),
 ('total_intl_calls', 0.0),
 ('total_intl_charge', 0.0),
 ('number_customer_service_calls', 0.0),
 ('churn', 0.0)]
```

In [19]:
```
train.describe().T
```

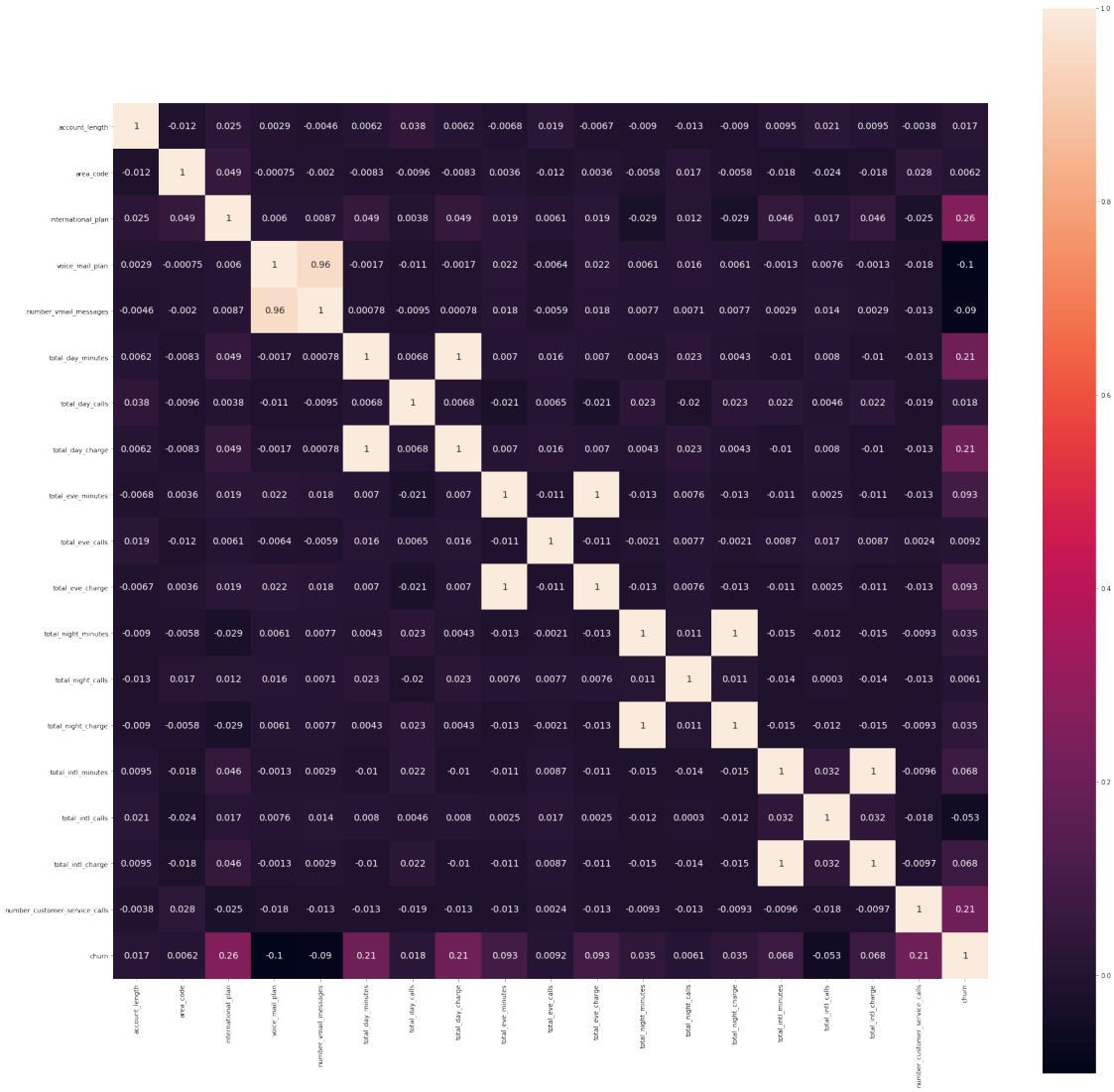
Out[19]:

	count	mean	std	min	25%	\
account_length	3333.0	101.064806	39.822106	1.00	74.00	
area_code	3333.0	437.182418	42.371290	408.00	408.00	
international_plan	3333.0	0.096910	0.295879	0.00	0.00	
voice_mail_plan	3333.0	0.276628	0.447398	0.00	0.00	
number_vmail_messages	3333.0	8.099010	13.688365	0.00	0.00	
total_day_minutes	3333.0	179.775098	54.467389	0.00	143.70	
total_day_calls	3333.0	100.435644	20.069084	0.00	87.00	
total_day_charge	3333.0	30.562307	9.259435	0.00	24.43	
total_eve_minutes	3333.0	200.980348	50.713844	0.00	166.60	
total_eve_calls	3333.0	100.114311	19.922625	0.00	87.00	

```
total_eve_charge              3333.0   17.083540   4.310668    0.00   14.16
total_night_minutes           3333.0  200.872037  50.573847   23.20  167.00
total_night_calls             3333.0  100.107711  19.568609   33.00   87.00
total_night_charge            3333.0    9.039325   2.275873    1.04    7.52
total_intl_minutes            3333.0   10.237294   2.791840    0.00    8.50
total_intl_calls              3333.0    4.479448   2.461214    0.00    3.00
total_intl_charge             3333.0    2.764581   0.753773    0.00    2.30
number_customer_service_calls 3333.0    1.562856   1.315491    0.00    1.00
churn                         3333.0    0.144914   0.352067    0.00    0.00

                                50%     75%     max
account_length                101.00  127.00  243.00
area_code                     415.00  510.00  510.00
international_plan              0.00    0.00    1.00
voice_mail_plan                 0.00    1.00    1.00
number_vmail_messages           0.00   20.00   51.00
total_day_minutes             179.40  216.40  350.80
total_day_calls               101.00  114.00  165.00
total_day_charge               30.50   36.79   59.64
total_eve_minutes             201.40  235.30  363.70
total_eve_calls               100.00  114.00  170.00
total_eve_charge               17.12   20.00   30.91
total_night_minutes           201.20  235.30  395.00
total_night_calls             100.00  113.00  175.00
total_night_charge              9.05   10.59   17.77
total_intl_minutes             10.30   12.10   20.00
total_intl_calls                4.00    6.00   20.00
total_intl_charge               2.78    3.27    5.40
number_customer_service_calls   1.00    2.00    9.00
churn                           0.00    0.00    1.00
```

Vamos plotar um heatmap para verificar a correlacao das features do dataset de treino

```
In [20]: fig = plt.subplots(figsize = (30,30))
         sns.heatmap(train.corr(),square = True,cbar=True,annot=True,annot_kws={'size': 14})
         plt.show()
```



Vamos verificar a media de cada feature relacionando com a variavel target

```
In [21]: train.groupby('churn').mean().T
```

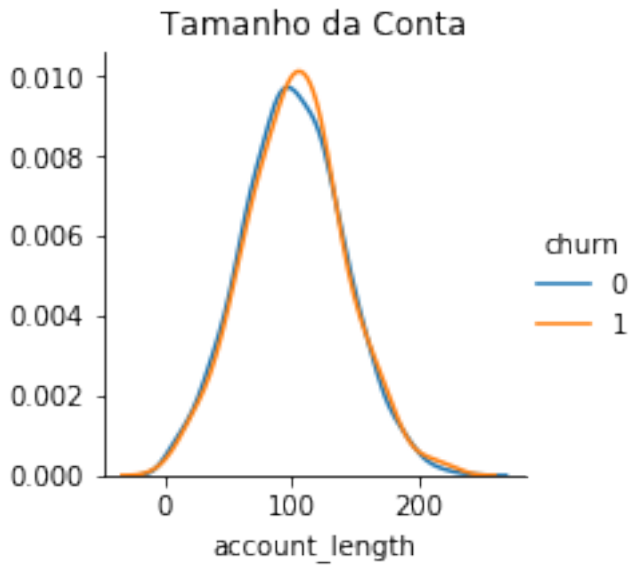
```
Out[21]: churn                          0           1
         account_length          100.793684  102.664596
         area_code               437.074737  437.817805
         international_plan        0.065263    0.283644
         voice_mail_plan           0.295439    0.165631
         number_vmail_messages     8.604561    5.115942
         total_day_minutes       175.175754  206.914079
         total_day_calls         100.283158  101.335404
         total_day_charge         29.780421   35.175921
         total_eve_minutes       199.043298  212.410145
         total_eve_calls         100.038596  100.561077
```

```
total_eve_charge                16.918909   18.054969
total_night_minutes            200.133193  205.231677
total_night_calls              100.058246  100.399586
total_night_charge               9.006074    9.235528
total_intl_minutes              10.158877   10.700000
total_intl_calls                 4.532982    4.163561
total_intl_charge                2.743404    2.889545
number_customer_service_calls    1.449825    2.229814
```

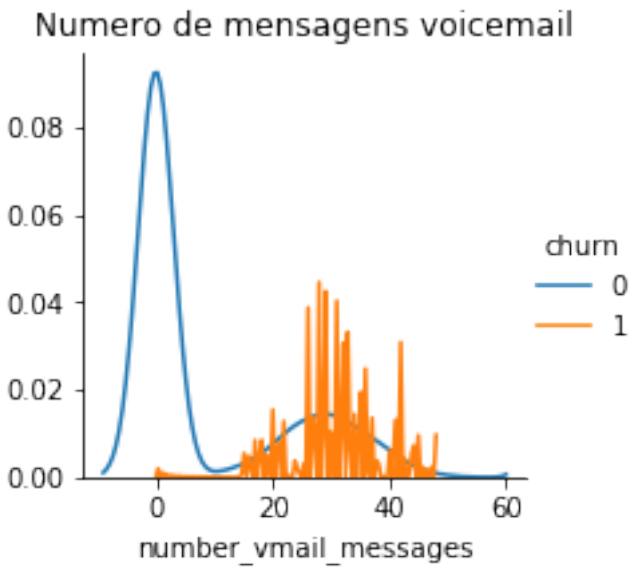
Obsevações: - a media de plano internacional é muito maior para clientes inativos - a media de plano de voz e mail é mais baixo para clientes inativos - a media de numero de mensagens em clientes inativos é menor - a media de ligacoes, tarifas e minutos utilizados, tanto de dia, tarde ou noite é maior em clientes inativos - number_customer_service_calls é praticamente o dobro para clientes inativos

Vamos olhar mais de perto algumas features e a relacao com a variavel target

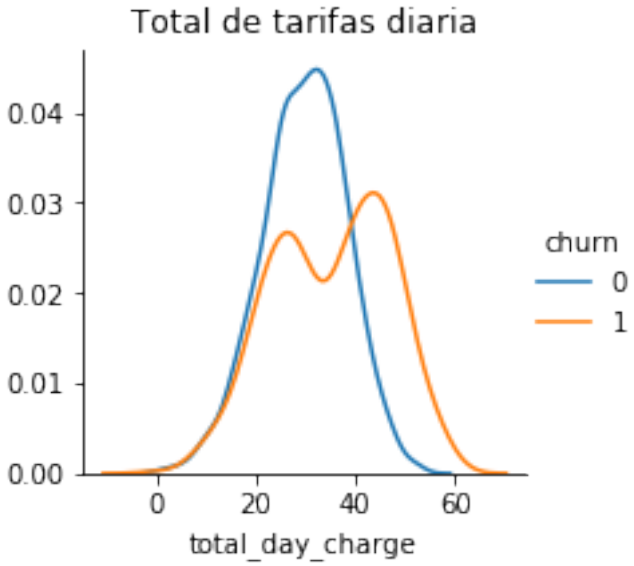
In [22]:
```
sns.FacetGrid(train, hue="churn") \
   .map(sns.kdeplot, "account_length") \
   .add_legend()
plt.title('Tamanho da Conta');
```



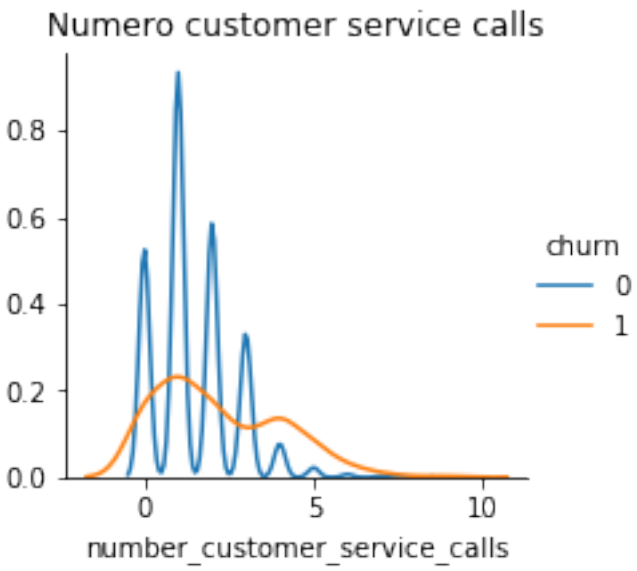
In [23]:
```
sns.FacetGrid(train, hue="churn") \
   .map(sns.kdeplot, "number_vmail_messages") \
   .add_legend()
plt.title('Numero de mensagens voicemail');
```




```
In [24]: sns.FacetGrid(train, hue="churn") \
             .map(sns.kdeplot, "total_day_charge") \
             .add_legend()
         plt.title('Total de tarifas diaria');
```




```
In [25]: sns.FacetGrid(train, hue="churn") \
             .map(sns.kdeplot, "number_customer_service_calls") \
             .add_legend()
         plt.title('Numero customer service calls');
```



Analisando o grafico acima percebemos que o numero de chamadas ao servico de atendimento ao consumidor é relativamente alto em clientes que estao inativos (churn). Isso indica que os clientes que mudaram de fornecedor tentaram entrar em contato com o atendimento ao cliente, mas podem não ter recebido uma resolução satisfatória para o problema.

2.5 Feature Selection - Método Ensemble

```
In [26]: # Importância do Atributo com o Extra Trees Classifier

         # Separando o array em componentes de input e output
         X = train.iloc[:,:-1]
         Y = train.churn

         # Criação do Modelo - Feature Selection
         modelo = ExtraTreesClassifier()
         modelo.fit(X, Y)

         # Convertendo o resultado em um dataframe
         df = pd.DataFrame(X.columns,columns=['Coluna'])
         df['Importancia'] = pd.DataFrame(modelo.feature_importances_.astype(float))

         # Realizando a ordenacao por Importancia (Maior para Menor)
         result = df.sort_values('Importancia',ascending=False)

         # Imprimindo as 10 variaveis mais importantes
         cols_of_interest = result[1:10]['Coluna']
         print(cols_of_interest)

         # Deixando somente as colunas de interesse no df de treino
```

```
        cols_of_interest = cols_of_interest.append(pd.Series(['churn']))
        new_train = train[cols_of_interest]
```

```
17    number_customer_service_calls
7                  total_day_charge
2                 international_plan
10                  total_eve_charge
8                  total_eve_minutes
15                  total_intl_calls
14                 total_intl_minutes
16                  total_intl_charge
11                total_night_minutes
Name: Coluna, dtype: object
```

2.6 Feature Engineering

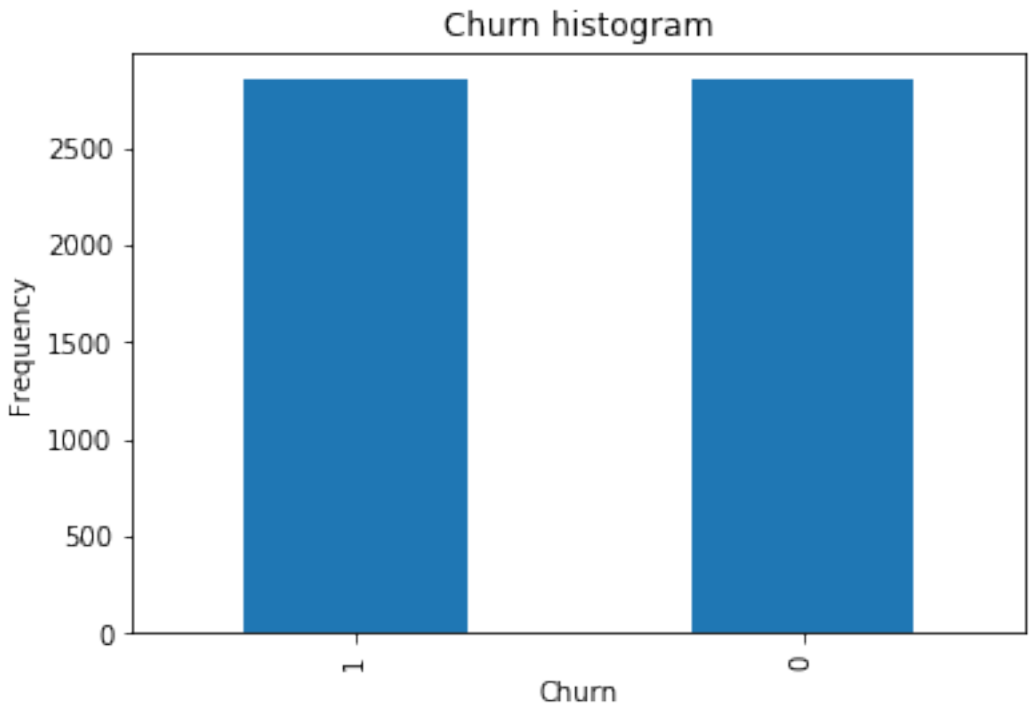
Resolvendo o problema de Overfitting da variavel TARGET utilizando o OverSampling ou Resample

```
In [27]: # Resolvendo problema de Overfitting utilizando o OverSampling

         # Separando o array em componentes de input e output
         X = new_train.iloc[:,:-1]
         Y = new_train.churn

         # Aplicando a funcao SMOTE
         # SMOTE eh um metodo de oversampling. Ele cria exemplos sinteticos da classe minorita
         sm = SMOTE(random_state=0)
         X_treino_res, Y_treino_res = sm.fit_sample(X, Y)

         np.bincount(Y_treino_res)
```

```
Out[27]: array([2850, 2850])
```

```
In [28]: # Distribuição das classes (variavel TARGET) apos aplicar OverSampling
         pd.value_counts(Y_treino_res).plot.bar()
         plt.title('Churn histogram')
         plt.xlabel('Churn')
         plt.ylabel('Frequency')

         # Visualizando um df com quantidade e percentual da variavel Churn
         df = pd.DataFrame(pd.value_counts(Y_treino_res), columns=['Churn'])
         df['%'] = 100*df['Churn']/Y_treino_res.shape[0]
         df
```

```
Out[28]:    Churn     %
         1   2850  50.0
         0   2850  50.0
```

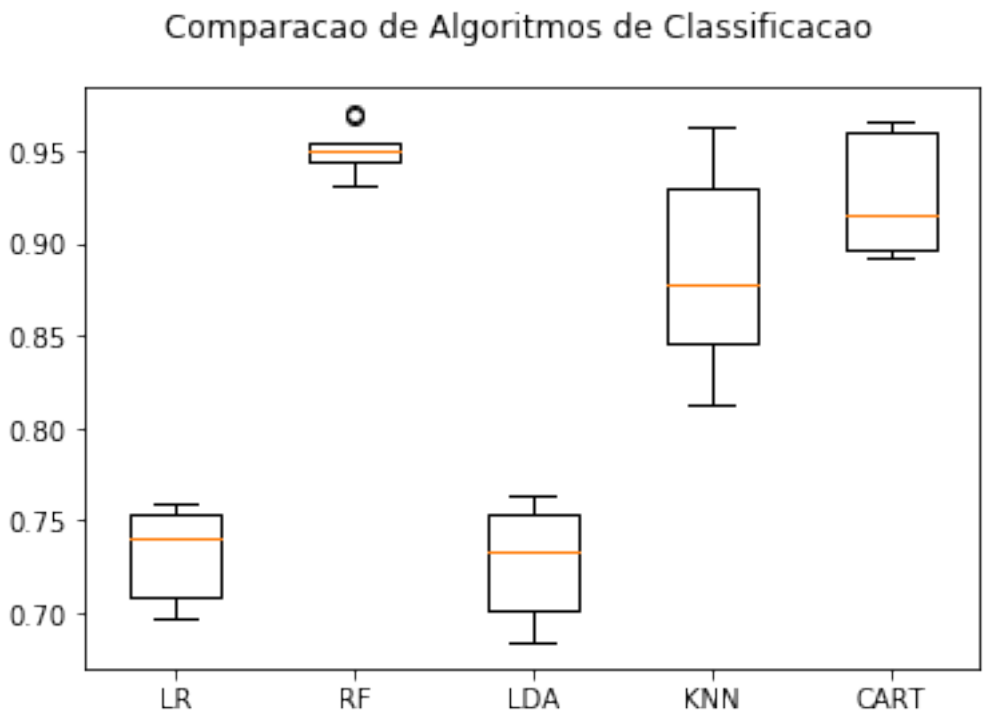


2.7 Criação e Validação dos Modelos de Machine Learning

```
In [29]: # Criacao dos modelos de Classificacao

         # Separando o array em componentes de input e output (dados com oversampling)
         X = X_treino_res
         Y = Y_treino_res

         # Aplicando a mesma escala nos dados
         X = MinMaxScaler().fit_transform(X)

         # Padronizando os dados (0 para a média, 1 para o desvio padrão)
         X = StandardScaler().fit_transform(X)

         # Definindo os valores para o número de folds
         num_folds = 10
         seed = 10

         # Preparando a lista de modelos
         modelos = []
         modelos.append(('LR', LogisticRegression()))
         modelos.append(('RF', RandomForestClassifier()))
         modelos.append(('LDA', LinearDiscriminantAnalysis()))
         modelos.append(('KNN', KNeighborsClassifier()))
```

```
modelos.append(('CART', DecisionTreeClassifier()))
#modelos.append(('SVM', SVC())) # comentei este modelo pois demora bastante e nao eh

# Avaliando cada modelo em um loop
resultados = []
nomes = []

for nome, modelo in modelos:
    kfold = KFold(n_splits = num_folds, random_state = seed)
    cv_results = cross_val_score(modelo, X, Y, cv = kfold, scoring = 'accuracy')
    resultados.append(cv_results)
    nomes.append(nome)
    msg = "%s: %f (%f)" % (nome, cv_results.mean(), cv_results.std())
    print(msg)

# Boxplot para comparar os algoritmos
fig = plt.figure()
fig.suptitle('Comparacao de Algoritmos de Classificacao')
ax = fig.add_subplot(111)
plt.boxplot(resultados)
ax.set_xticklabels(nomes)
plt.show()
```

```
LR: 0.732456 (0.023917)
RF: 0.950702 (0.011328)
LDA: 0.727544 (0.028860)
KNN: 0.886316 (0.051698)
CART: 0.926316 (0.031197)
```



3 Otimizando Performance do modelo

3.1 Grid Search Parameter Tuning

Este método realiza metodicamente combinações entre todos os parâmetros do algoritmo, criando um grid. Vamos usar este método utilizando o algoritmo de Regressão Logística, conforme recomendado na especificacao

```
In [30]: # Definindo os valores que serão testados
         valores_grid = {'penalty': ['l1','l2'], 'C': [0.001,0.01,0.1,1,10,100,1000]}

         # Criando o modelo
         modelo = LogisticRegression()

         # Criando o grid
         grid = GridSearchCV(estimator = modelo, param_grid = valores_grid)
         grid.fit(X, Y)

         # Print do resultado
         print("Acurácia: %.3f" % (grid.best_score_ * 100))
         print("Melhores Parâmetros do Modelo:\n", grid.best_estimator_)
```

```
Acurácia: 77.246
Melhores Parâmetros do Modelo:
```

```
LogisticRegression(C=1000, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

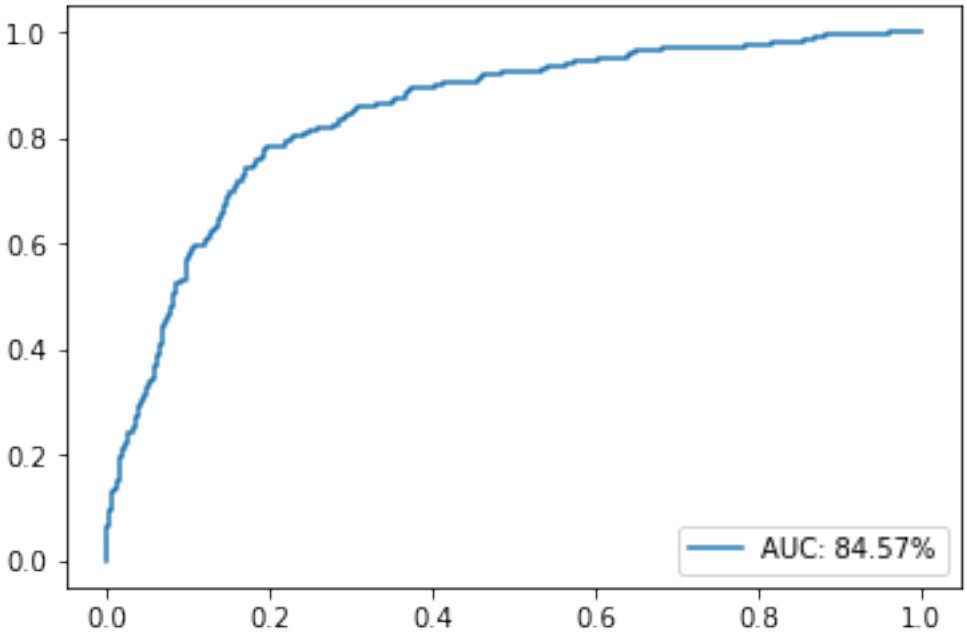
3.2 Utilizando o modelo Logistic Regression para fazer previsoes no dataset de teste

```
In [31]: # Fazendo previsões com o modelo Logistic Regression

         # Separando o array em componentes de input e output
         X_teste = test.iloc[:,:-1]
         Y_teste = test.churn

         # Aplicando a mesma escala nos dados
         X_teste = MinMaxScaler().fit_transform(X_teste)

         # Padronizando os dados (0 para a média, 1 para o desvio padrão)
         X_teste = StandardScaler().fit_transform(X_teste)


         # Fit do modelo com melhores parametros
         modelo_otm = grid.estimator
         modelo_otm.fit(X_teste, Y_teste)

         # Fazendo previsões no modelo de treino
         y_pred = modelo_otm.predict(X_teste)
         previsoes = [round(value) for value in y_pred]

         # Avaliando as previsões
         accuracy = accuracy_score(Y_teste, previsoes)
         print("Acurácia: %.2f%%" % (accuracy * 100.0))
```

```
Acurácia: 87.64%
```

Realizando previsoes no dataset de teste a acurácia do modelo de Regressao Logistica chegou a 87%

```
In [32]: y_pred_proba = modelo_otm.predict_proba(X_teste)[::,1]
         fpr, tpr, _ = roc_curve(Y_teste,  y_pred_proba)
         auc = roc_auc_score(Y_teste, y_pred_proba)
         plt.plot(fpr,tpr,label="AUC: %.2f%%" % (auc * 100.0))
         plt.legend(loc=4)
         plt.show()
```



A pontuação da AUC para este caso é de 84%. AUC 1 representa um classificador perfeito e 0,5 representa um classificador sem valor. Sendo assim, é um ótimo resultado do modelo

4 Verificando o relatório de classificação

```
In [33]: # Relatório de Classificação
         report = classification_report(Y_teste, previsoes)
         print(report)
```

```
              precision    recall  f1-score   support

           0       0.89      0.98      0.93      1443
           1       0.61      0.22      0.32       224

   micro avg       0.88      0.88      0.88      1667
   macro avg       0.75      0.60      0.63      1667
weighted avg       0.85      0.88      0.85      1667
```

```
In [34]: # Confusion Matrix
         # Permite verificar a acurácia em um formato de tabela
         matrix = confusion_matrix(Y_teste, previsoes)
         print("Confusion Matrix")
         print(matrix)
```

```
Confusion Matrix
[[1412   31]
 [ 175   49]]
```

```
In [35]: # Criacao de função para criar um plot para a confusion matrix
         # http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.
```

```
         def plot_confusion_matrix(cm,
                                   target_names,
                                   title='Confusion matrix',
                                   cmap=None,
                                   normalize=True):



             accuracy = np.trace(cm) / float(np.sum(cm))
             misclass = 1 - accuracy

             if cmap is None:
                 cmap = plt.get_cmap('Blues')

             plt.figure(figsize=(8, 6))
             plt.imshow(cm, interpolation='nearest', cmap=cmap)
             plt.title(title)
             plt.colorbar()

             if target_names is not None:
                 tick_marks = np.arange(len(target_names))
                 plt.xticks(tick_marks, target_names, rotation=45)
                 plt.yticks(tick_marks, target_names)

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


             thresh = cm.max() / 1.5 if normalize else cm.max() / 2
             for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                 if normalize:
                     plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                              horizontalalignment="center",
                              color="white" if cm[i, j] > thresh else "black")
                 else:
                     plt.text(j, i, "{:,}".format(cm[i, j]),
                              horizontalalignment="center",
                              color="white" if cm[i, j] > thresh else "black")
```

```
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy,
    plt.show()
```

In [36]:
```
# Chamando a função para visualizar a confusion matrix
plot_confusion_matrix(matrix,
                      normalize    = False,
                      target_names = ['0', '1'],
                      title        = "Confusion Matrix")
```