Final Project

Telecom churn prediction

Table of Contents

Goal

Develop a binary classification model for the telecom operator Interconnect that predicts whether a customer will leave the company soon based on the clientele's personal data, including information about their plans and contracts.

Primary metric: AUC-ROC, it should be more than 0.75, preferably above 0.88.

Additional metric: Accuracy.

Additional Assignment

Client outflow research

1. Compare the monthly payment distribution (*MonthlyCharges*) of all active clients with the clients who have left. Calculate the following statistics for each group: the average, minimum and maximum values, the median, and the values of the 25% and 75% percentiles. Build distribution histograms based on your findings.

2. Compare the behavior of the clients from the two groups below. For each group, build any two graphs which display:

- The share of telephone users
- The share of Internet users

Data description

The data consists of files obtained from different sources:

- `contract.csv` — contract information
- `personal.csv` — the client's personal data
- `internet.csv` — information about Internet services
- `phone.csv` — information about telephone services

In each file, the column `customerID` contains a unique code assigned to each client.

Target feature: the `'EndDate'` column equals `'No'` .

The contract information is valid as of February 1, 2020.

Imports

In [643…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from functools import reduce
import re
from sklearn.utils import shuffle
```

```
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from xgboost import XGBClassifier

from sklearn.preprocessing import StandardScaler as ss
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV

import sys
import warnings
if not sys.warnoptions:
       warnings.simplefilter("ignore")
```

In [644…

```
import matplotlib.patches as mpatches
%matplotlib inline
%config InlineBackend.figure_format = 'png'
# the next line provides graphs of better quality on HiDPI screens
%config InlineBackend.figure_format = 'retina'

plt.style.use('seaborn')
pd.set_option('display.max_rows', None, 'display.max_columns', None)

print("Setup Complete")
```

```
Setup Complete
```

Input data

In [645…

```
try:
    df_contract = pd.read_csv('final_provider/contract.csv')
    df_personal = pd.read_csv('final_provider/personal.csv')
    df_internet = pd.read_csv('final_provider/internet.csv')
    df_phone = pd.read_csv('final_provider/phone.csv')

except:
    df_contract = pd.read_csv('datasets/final_provider/contract.csv')
    df_personal = pd.read_csv('datasets/final_provider/personal.csv')
    df_internet = pd.read_csv('datasets/final_provider/internet.csv')
    df_phone = pd.read_csv('datasets/final_provider/phone.csv')
```

Descriptive statistics

Merge data frames

First of all, let's merge all 4 data frames based on `customerID` .

In [646…

```
# compile the list of dataframes you want to merge
data_frames = [df_contract, df_personal, df_internet, df_phone]

df_merged = reduce(lambda  left,right: pd.merge(left,right,on=['customerID'],
                                            how='outer'), data_frames)
df_merged.head()
```

Out[646…

customerID	BeginDate	EndDate	Type	PaperlessBilling	PaymentMethod	MonthlyCharge

	customerID	BeginDate	EndDate	Type	PaperlessBilling	PaymentMethod	MonthlyCharge
0	7590-VHVEG	2020-01-01	No	Month-to-month	Yes	Electronic check	29.8
1	5575-GNVDE	2017-04-01	No	One year	No	Mailed check	56.9
2	3668-QPYBK	2019-10-01	2019-12-01 00:00:00	Month-to-month	Yes	Mailed check	53.8
3	7795-CFOCW	2016-05-01	No	One year	No	Bank transfer (automatic)	42.3
4	9237-HQITU	2019-09-01	2019-11-01 00:00:00	Month-to-month	Yes	Electronic check	70.7

In [647…

```
df_merged.head()
```

Out[647…

	customerID	BeginDate	EndDate	Type	PaperlessBilling	PaymentMethod	MonthlyCharge
0	7590-VHVEG	2020-01-01	No	Month-to-month	Yes	Electronic check	29.8
1	5575-GNVDE	2017-04-01	No	One year	No	Mailed check	56.9
2	3668-QPYBK	2019-10-01	2019-12-01 00:00:00	Month-to-month	Yes	Mailed check	53.8
3	7795-CFOCW	2016-05-01	No	One year	No	Bank transfer (automatic)	42.3
4	9237-HQITU	2019-09-01	2019-11-01 00:00:00	Month-to-month	Yes	Electronic check	70.7

In [648…

```
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   BeginDate         7043 non-null   object
 2   EndDate           7043 non-null   object
 3   Type              7043 non-null   object
 4   PaperlessBilling  7043 non-null   object
 5   PaymentMethod     7043 non-null   object
 6   MonthlyCharges    7043 non-null   float64
 7   TotalCharges      7043 non-null   object
 8   gender            7043 non-null   object
 9   SeniorCitizen     7043 non-null   int64
 10  Partner           7043 non-null   object
 11  Dependents        7043 non-null   object
 12  InternetService   5517 non-null   object
 13  OnlineSecurity    5517 non-null   object
 14  OnlineBackup      5517 non-null   object
 15  DeviceProtection  5517 non-null   object
 16  TechSupport       5517 non-null   object
 17  StreamingTV       5517 non-null   object
 18  StreamingMovies   5517 non-null   object
```

```
 19  MultipleLines     6361 non-null   object
dtypes: float64(1), int64(1), object(18)
memory usage: 1.1+ MB
```

In [649…

```
df_merged.describe()
```

Out[649…

	MonthlyCharges	SeniorCitizen
count	7043.000000	7043.000000
mean	64.761692	0.162147
std	30.090047	0.368612
min	18.250000	0.000000
25%	35.500000	0.000000
50%	70.350000	0.000000
75%	89.850000	0.000000
max	118.750000	1.000000

Notes for preprocessing:

- More than 7k observations, not all customers have phone and internet details;
- No missing values;
- Change column names to lower case;
- Check for duplicates;
- `EndDate` and `BeginDate` convert to DateTime, calculate the duration, in case there is no end date, take the 1st of February, 2020;
- Most variables are binary, convert them to numeric;
- Convert `TotalCharges` to float;
- `MonthlyCharges` : slightly positively skewed distribution mean and median are close to each other - a sign of an overall normal distribution, can be split into bins together with `TotalCharges` ;
- `SeniorCitizen` : Mostly NOT senior citizens;
- Remove N/A in the phone and internet data after merging;
- `EndDate` is the target, convert to binary, numeric.

Preprocessing

Target

Let's create a binary numeric target column.

In [650…

```
def label_data(end_date):
    if end_date == 'No':
        return 0
    else:
        return 1
```

In [651…

```
y = df_merged['EndDate'].apply(label_data)
y.value_counts()
```

Out[651…

```
0    5174
1    1869
Name: EndDate, dtype: int64
```

Lower case column names

In [652…
```
columns = []
for name in df_merged.columns.values:
    name = re.sub('([A-Z])', r' \1', name).lower().replace(' ', '_')[1:]
    columns.append(name)
```

In [653…
```
df_merged.columns = columns
```

In [654…
```
df_merged = df_merged.rename(columns = {'ustomer_i_d':'customer_i_d', 'ender'
```

In [655…
```
df_merged.head()
```

Out[655…

	customer_i_d	begin_date	end_date	type	paperless_billing	payment_method	monthly_cl
0	7590-VHVEG	2020-01-01	No	Month-to-month	Yes	Electronic check	
1	5575-GNVDE	2017-04-01	No	One year	No	Mailed check	
2	3668-QPYBK	2019-10-01	2019-12-01 00:00:00	Month-to-month	Yes	Mailed check	
3	7795-CFOCW	2016-05-01	No	One year	No	Bank transfer (automatic)	
4	9237-HQITU	2019-09-01	2019-11-01 00:00:00	Month-to-month	Yes	Electronic check	

Check for duplicates

In [656…
```
df_merged.duplicated().sum()
```

Out[656… 0

Missing values

In [657…
```
df_merged.isnull().sum()/len(df_merged)
```

Out[657…
```
customer_i_d         0.000000
begin_date           0.000000
end_date             0.000000
type                 0.000000
paperless_billing    0.000000
payment_method       0.000000
monthly_charges      0.000000
total_charges        0.000000
gender               0.000000
senior_citizen       0.000000
partner              0.000000
dependents           0.000000
internet_service     0.216669
online_security      0.216669
online_backup        0.216669
device_protection    0.216669
tech_support         0.216669
streaming_t_v        0.216669
streaming_movies     0.216669
multiple_lines       0.096834
dtype: float64
```

8 columns have missing values. All columns, except for the `internet_service` have binary values (Yes/No), so we will replace all missing values there with the 'No'.

In [658…
```
col_with_missing_values = list(df_merged.columns[df_merged.isnull().any()])
col_with_missing_values.remove('multiple_lines')
df_merged[col_with_missing_values] = df_merged[col_with_missing_values].apply
```

The missing `internet_service` values we will replace with the 'not_available' value.

In [659…
```
df_merged['multiple_lines'] = df_merged['multiple_lines'].fillna('not_availab
```

Data type change

Dates

First, let's replace the `No` value in the `end_date` column with the date of data extraction: 2020-02-01. Then we will calculate the number of days a client stayed with the company until the day of data extraction.

In [660…
```
df_merged.loc[df_merged['end_date'] == 'No', 'end_date'] = '2020-02-01 00:00:
```

In [661…
```
df_merged['begin_date'] = pd.to_datetime(df_merged['begin_date'], format='%Y-
df_merged['end_date'] = pd.to_datetime(df_merged['end_date'], format='%Y-%m-%
```

In [662…
```
extraction_date = pd.to_datetime('2020-02-01 00:00:00')
df_merged['days_since_join'] = (extraction_date - df_merged['begin_date']).dt
```

In [663…
```
def new_client(begin_date):
    extraction_date = pd.to_datetime('2020-02-01 00:00:00')
    if (extraction_date - begin_date).days/30 < 1:
        return 1
    else:
        return 0
```

In [664…
```
df_merged['new_client'] = df_merged['begin_date'].apply(new_client)
df_merged['new_client'].value_counts()
```

Out[664…
```
0    7032
1      11
Name: new_client, dtype: int64
```

In [665…
```
df_merged.head(3)
```

Out[665…

	customer_i_d	begin_date	end_date	type	paperless_billing	payment_method	monthly_cl
0	7590-VHVEG	2020-01-01	2020-02-01	Month-to-month	Yes	Electronic check	
1	5575-GNVDE	2017-04-01	2020-02-01	One year	No	Mailed check	
2	3668-QPYBK	2019-10-01	2019-12-01	Month-to-month	Yes	Mailed check	

Total_charges

Based on analysis, there are some missing values in this column. Let's have a look at them.

In [666…
```
df_merged[df_merged['total_charges'] == ' ']
```

Out[666…

	customer_i_d	begin_date	end_date	type	paperless_billing	payment_method	monthly_
488	4472-LVYGI	2020-02-01	2020-02-01	Two year	Yes	Bank transfer (automatic)	
753	3115-CZMZD	2020-02-01	2020-02-01	Two year	No	Mailed check	
936	5709-LVOEQ	2020-02-01	2020-02-01	Two year	No	Mailed check	
1082	4367-NUYAO	2020-02-01	2020-02-01	Two year	No	Mailed check	
1340	1371-DWPAZ	2020-02-01	2020-02-01	Two year	No	Credit card (automatic)	
3331	7644-OMVMY	2020-02-01	2020-02-01	Two year	No	Mailed check	
3826	3213-VVOLG	2020-02-01	2020-02-01	Two year	No	Mailed check	
4380	2520-SGTTA	2020-02-01	2020-02-01	Two year	No	Mailed check	
5218	2923-ARZLG	2020-02-01	2020-02-01	One year	Yes	Mailed check	
6670	4075-WKNIU	2020-02-01	2020-02-01	Two year	No	Mailed check	
6754	2775-SEFEE	2020-02-01	2020-02-01	Two year	Yes	Bank transfer (automatic)	

These clients just started using Telecom services the same month as the data was extracted. We will fill the missing `total_charges` with the value of the `monthly_charges`.

In [667…
```
df_merged.loc[df_merged['total_charges'] == ' ', 'total_charges'] = df_merged
```

In [668…
```
df_merged['total_charges'] = df_merged['total_charges'].astype('float')
```

Categorical features encoding

In [669…
```
df_merged['type'].value_counts()
```

Out[669…
```
Month-to-month    3875
Two year          1695
One year          1473
Name: type, dtype: int64
```

In [670…
```
df_merged['payment_method'].value_counts()
```

Out[670…
```
Electronic check             2365
Mailed check                 1612
Bank transfer (automatic)    1544
Credit card (automatic)      1522
Name: payment_method, dtype: int64
```

As none of the variables have high dimensionality, we will encode them with the OHE method. We will also drop some non-informative columns.

In [671…
```
df_final = df_merged.drop(['customer_i_d', 'begin_date', 'end_date'], axis=1)
```

```
df_final['exited'] = y
X_OHE = pd.get_dummies(df_final, drop_first=True)
X_OHE.drop(['exited','monthly_charges'], axis=1, inplace=True)
X_OHE.head()
```

Out[671…

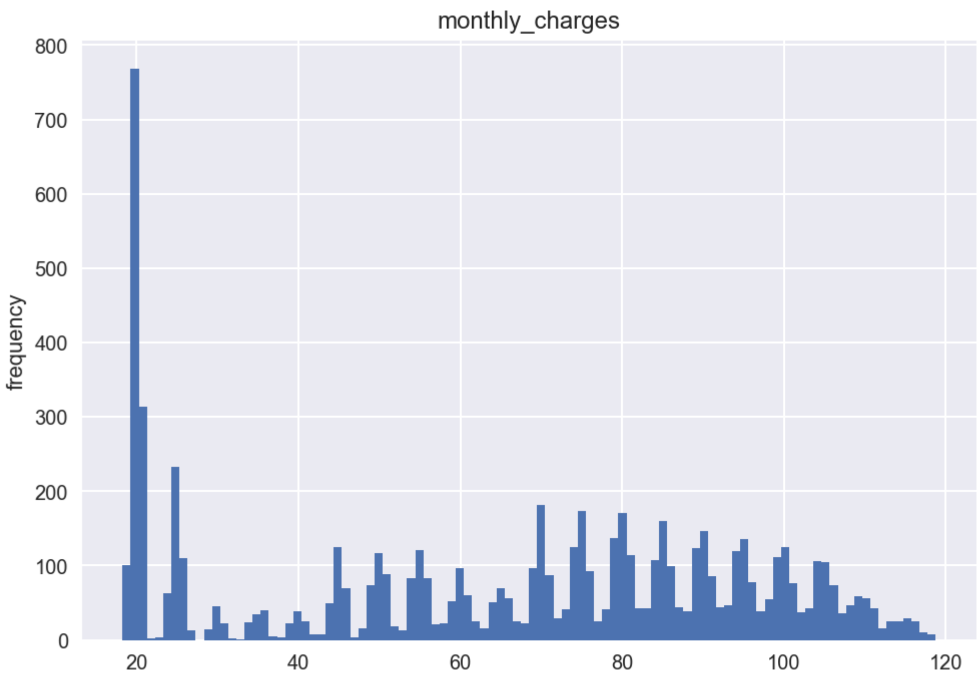
	total_charges	senior_citizen	days_since_join	new_client	type_One year	type_Two year	paperless_bil
0	29.85	0	31	0	0	0	
1	1889.50	0	1036	0	1	0	
2	108.15	0	123	0	0	0	
3	1840.75	0	1371	0	1	0	
4	151.65	0	153	0	0	0	

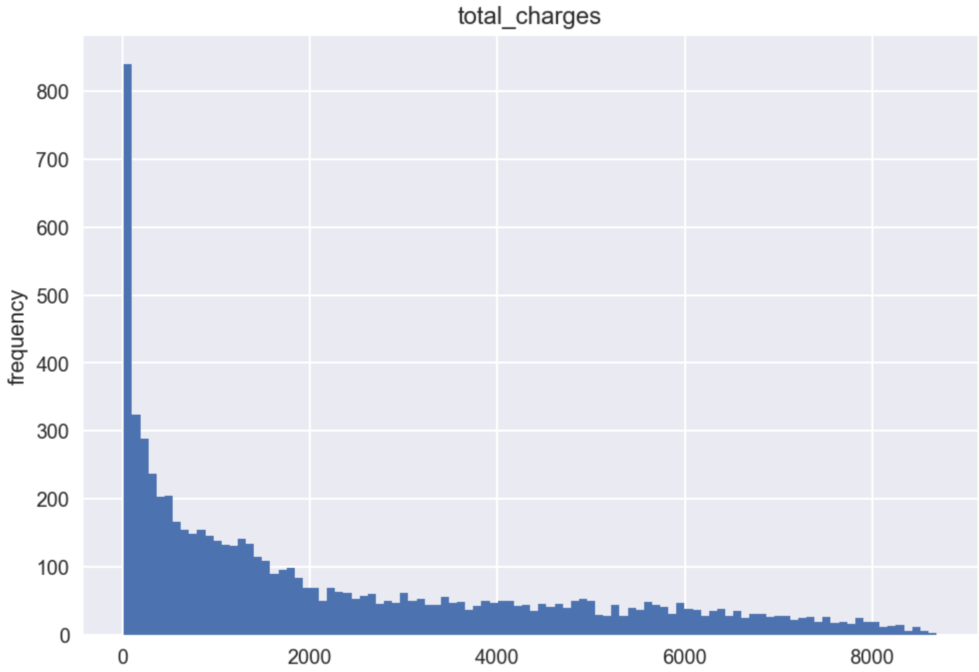
EDA

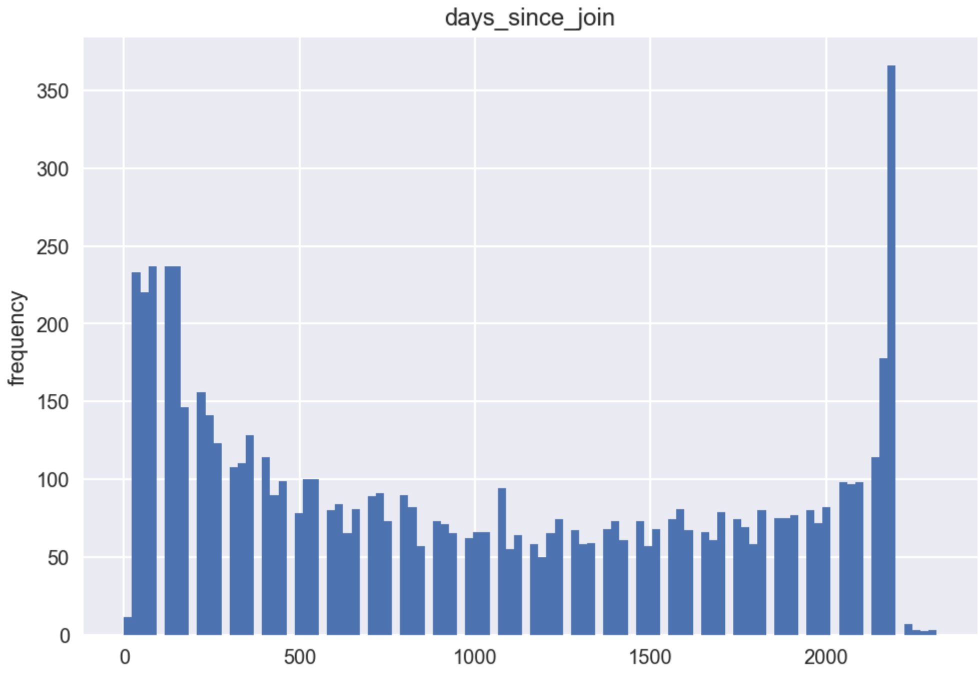
Features analysis

In [672…

```
for feature in ['monthly_charges','total_charges','days_since_join']:
    df_final.hist(feature, bins=100)
    plt.ylabel('frequency');
```







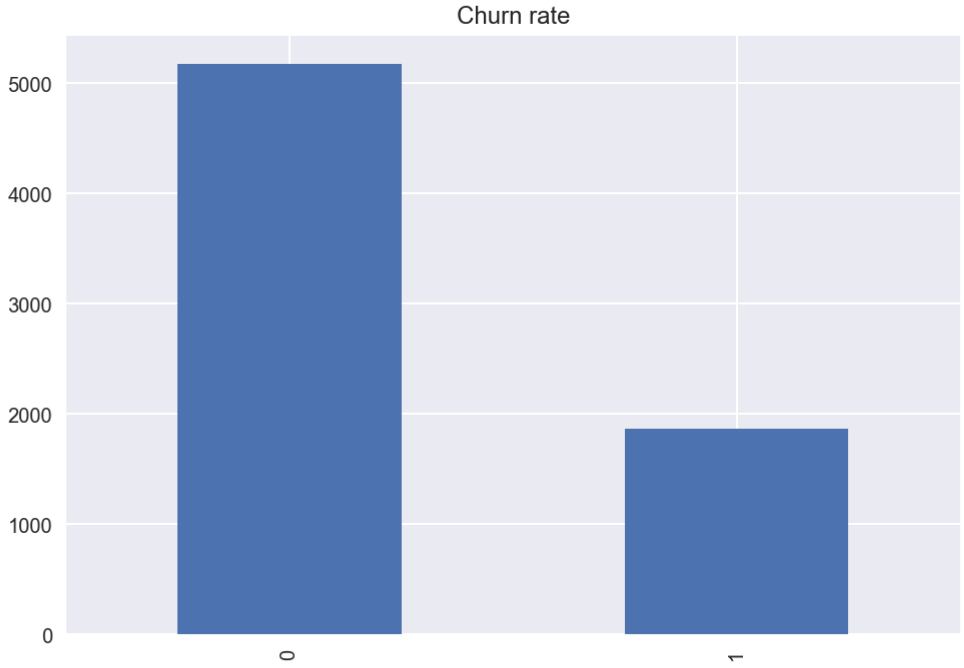
There are a lot of observations with the `total_charges` values less than 100 dollars. These are most likely either new clients or those who quickly left the company after just a month. There are quite a lot of 'loyal' customers as well who stayed with the company for more than 5 years based on the `days_since_join` variable distribution.

A big amount of clients pay around 20 dollars a month for Telecom services. Otherwise, there are no visible outliers.

Target analysis

In [673…

```
y.value_counts().plot(kind='bar')
plt.title('Churn rate');
```



We see that our classes are imbalanced: there are at least twice as many observations for the customers who stayed with Telecom than for those who left.

Imbalanced classifications pose a challenge for predictive modeling as most of the machine learning algorithms used for classification are designed around the assumption of an equal number of examples for each class. This results in models that have poor predictive performance, specifically for the minority class. In this case the minority class is more important and therefore the model is more sensitive to classification errors for the minority class than the majority class. We will be applying some techniques to correct class imbalance.

Let's also calculate an average churn rate of the whole dataset:

In [674...
```
df_final['exited'].mean()
```

Out[674...
```
0.26536987079369590
```

On average, there is a 26.5% probability that a customer will leave the company.

Additional assignment

Monthly charges

In [675...
```
df_final.groupby('exited')['monthly_charges'].describe()
```

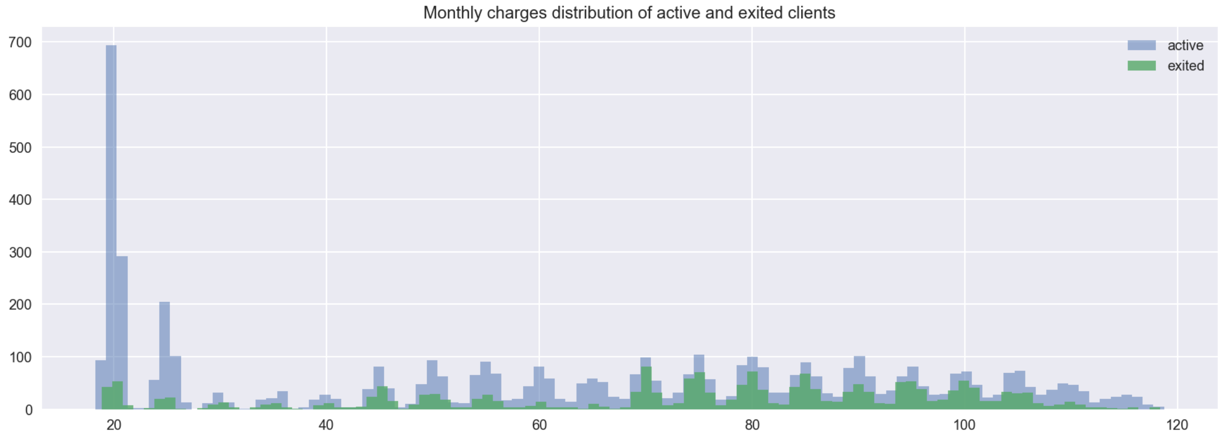
Out[675...

exited	count	mean	std	min	25%	50%	75%	max
0	5174.0	61.265124	31.092648	18.25	25.10	64.425	88.4	118.75
1	1869.0	74.441332	24.666053	18.85	56.15	79.650	94.2	118.35

In [676...
```
active = df_final[df_final['exited'] == 0]
exited = df_final[df_final['exited'] == 1]
```

In [677...
```
plt.figure(figsize=(15,5))
```

```
plt.hist(active['monthly_charges'], bins=100, alpha=0.5, label='active')
plt.hist(exited['monthly_charges'], bins=100, alpha=0.8, label='exited')
plt.legend(loc='upper right')
plt.title('Monthly charges distribution of active and exited clients');
```

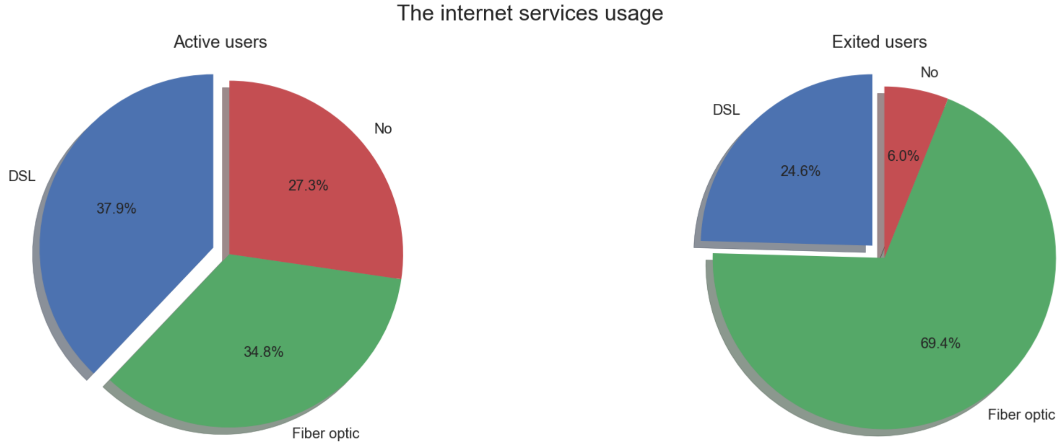


The biggest difference between the 2 groups is that there is a big number of active clients who pay only 20 dollars monthly for Telecom services. Exited clients used to pay more, on average. Maybe that was one of the reasons why they left the company.

The internet services

In [678…
```
active_internet_classes = active.groupby('internet_service')['internet_servic
exited_internet_classes = exited.groupby('internet_service')['internet_servic
```

In [679…
```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(15,5))

fig.suptitle('The internet services usage', fontsize=15)

labels = active_internet_classes.index
sizes = active_internet_classes
explode = (0.1, 0, 0)  # only "explode" the 1st slice

ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
ax1.set_title('Active users');

labels = exited_internet_classes.index
sizes = exited_internet_classes
explode = (0.1, 0, 0)  # only "explode" the 1st slice

ax2.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax2.axis('equal');  # Equal aspect ratio ensures that pie is drawn as a circl
ax2.set_title('Exited users');
```



The share of the internet users from the 'active' clients is 4 times smaller than that of the 'exited' users. There are more 'active' clients who are not using the internet than in the 'exited' clients group. We also see that the network is set up through a fiber optic cable for the 'exited' clients twice as many times as for the 'active' clients. This observation could mean that the quality of the fiber cable internet is lower than that of the DSL internet and that was one of the reasons for the customers to leave the company.

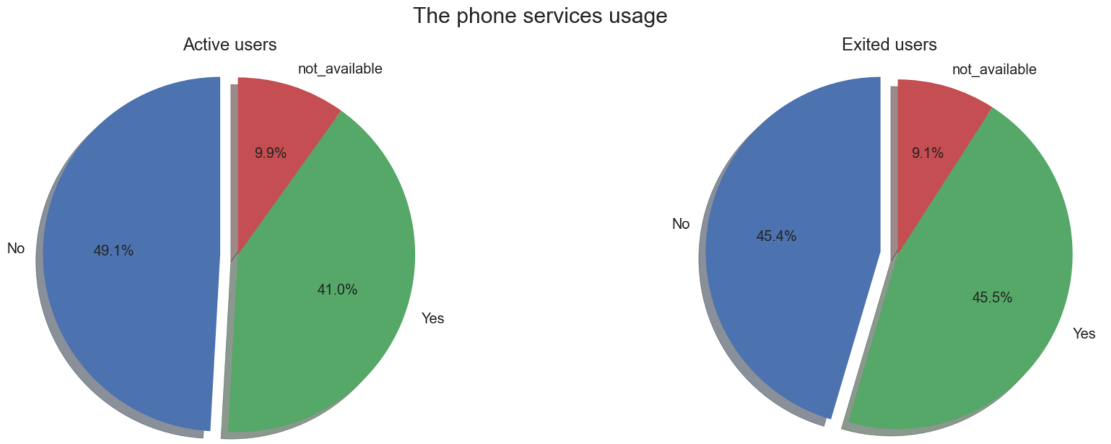
The phone services

In [680…

```
active_phone_classes = active.groupby('multiple_lines')['multiple_lines'].cou
exited_phone_classes = exited.groupby('multiple_lines')['multiple_lines'].cou
```

In [681…

```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(15,5))

fig.suptitle('The phone services usage', fontsize=15)

labels = active_phone_classes.index
sizes = active_phone_classes
explode = (0.1, 0, 0)  # only "explode" the 1st slice

ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
ax1.set_title('Active users');

labels = exited_phone_classes.index
sizes = exited_phone_classes
explode = (0.1, 0, 0)  # only "explode" the 1st slice

ax2.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax2.axis('equal');  # Equal aspect ratio ensures that pie is drawn as a circl
ax2.set_title('Exited users');
```



The phone services usage distribution is almost the same for both groups. There are slightly (3.5%) more 'active' users who are not using multiple lines than in the 'exited' group but the difference is not very significant.

Evaluation Procedure

Composing an evaluation routine which can be used for all models in this project

In [682…

```
def evaluate_model(model, train_features, train_target, test_features, test_t

    eval_stats = {}

    fig, ax = plt.subplots(figsize=(10, 6))

    for type, features, target in (('train', train_features, train_target), (

        eval_stats[type] = {}

        pred_target = model.predict(features)
        pred_proba = model.predict_proba(features)[:, 1]


        # ROC
        fpr, tpr, roc_thresholds = metrics.roc_curve(target, pred_proba)
        roc_auc = metrics.roc_auc_score(target, pred_proba)
        eval_stats[type]['ROC AUC'] = roc_auc


        if type == 'train':
            color = 'blue'
        else:
            color = 'green'

        # ROC
        ax.plot(fpr, tpr, color=color, label=f'{type}, ROC AUC={roc_auc:.2f}'
        # setting crosses for some thresholds
        for threshold in (0.2, 0.4, 0.5, 0.6, 0.8):
            closest_value_idx = np.argmin(np.abs(roc_thresholds-threshold))
            marker_color = 'orange' if threshold != 0.5 else 'red'
            ax.plot(fpr[closest_value_idx], tpr[closest_value_idx], color=mar
        ax.plot([0, 1], [0, 1], color='grey', linestyle='--')
        ax.set_xlim([-0.02, 1.02])
        ax.set_ylim([-0.02, 1.02])
        ax.set_xlabel('FPR')
        ax.set_ylabel('TPR')
        ax.legend(loc='lower center')
        ax.set_title(f'ROC Curve')
```

```
        eval_stats[type]['Accuracy'] = metrics.accuracy_score(target, pred_ta

    df_eval_stats = pd.DataFrame(eval_stats)
    df_eval_stats = df_eval_stats.round(2)
    df_eval_stats = df_eval_stats.reindex(index=('Accuracy', 'ROC AUC'))

    print(df_eval_stats)

    return df_eval_stats
```

Train/test/validation split

In [683…
```
X_train, X_test, y_train, y_test = train_test_split(X_OHE, y, test_size = 0.2
```

In [684…
```
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_
```

Standard Scaling

Finally, we will scale our features with Standard Scaler. It will convert our data frames into numpy arrays, so after they are transformed, let's convert them back to data frames.

In [685…
```
sc = ss()
X_train_scaled = sc.fit_transform(X_train)
X_valid_scaled = sc.transform(X_valid)
X_test_scaled = sc.transform(X_test)
```

In [686…
```
X_train = pd.DataFrame(data=X_train_scaled,
                        index=X_train.index,
                        columns=X_train.columns)
```

In [687…
```
X_valid = pd.DataFrame(data=X_valid_scaled,
                        index=X_valid.index,
                        columns=X_valid.columns)
```

In [688…
```
X_test = pd.DataFrame(data=X_test_scaled,
                        index=X_test.index,
                        columns=X_test.columns)
```

In [689…
```
print('Train set shape:', X_train.shape)
print('Train set shape:', X_valid.shape)
print('Train set shape:', X_test.shape)
```

```
Train set shape: (4507, 23)
Train set shape: (1127, 23)
Train set shape: (1409, 23)
```

Modelling

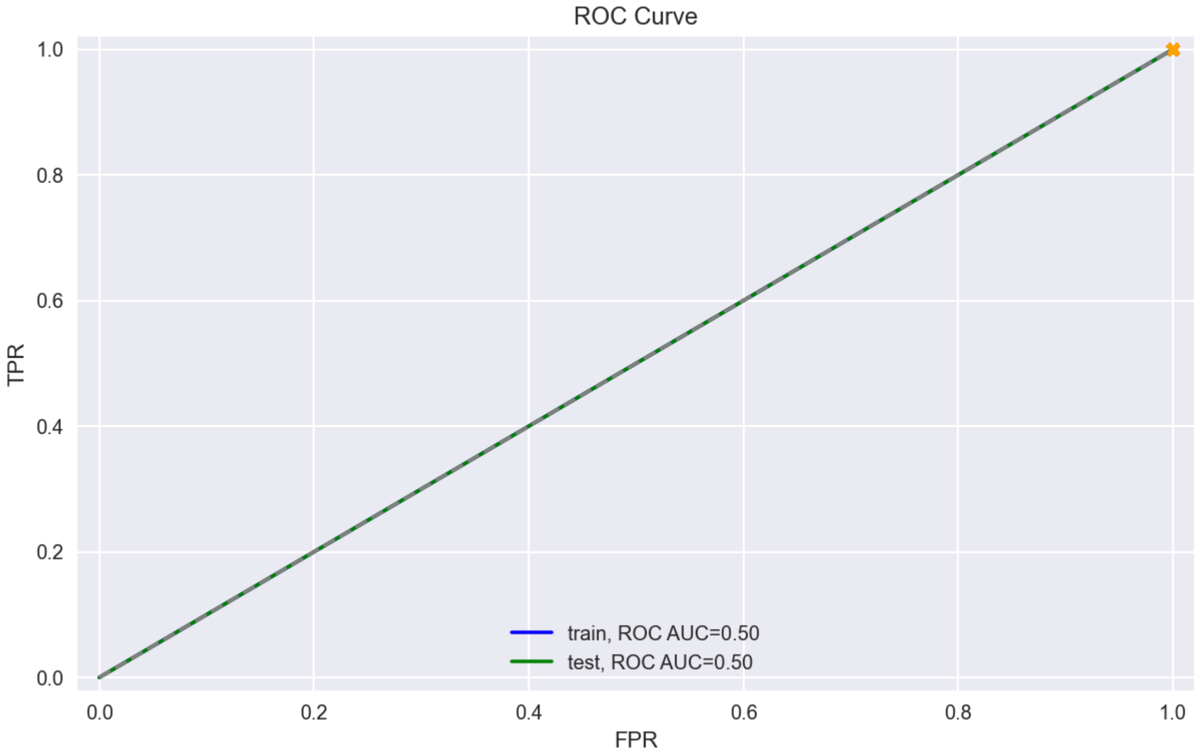
Baseline model

In our case it is more important that a model correctly predicts the minority class - whether a customer left the bank, that's why we will choose the strategy `constant` for the dummy classifier.

In [690…
```
base_model = DummyClassifier(strategy='constant', constant=1, random_state=12
base_model.fit(X_train, y_train)
```

```
result = evaluate_model(base_model, X_train, y_train, X_valid, y_valid)
```

```
          train  test
Accuracy   0.27  0.27
ROC AUC    0.50  0.50
```



In [691…

```
acc_baseline = result['test']['Accuracy']
roc_auc_baseline = result['test']['ROC AUC']
```

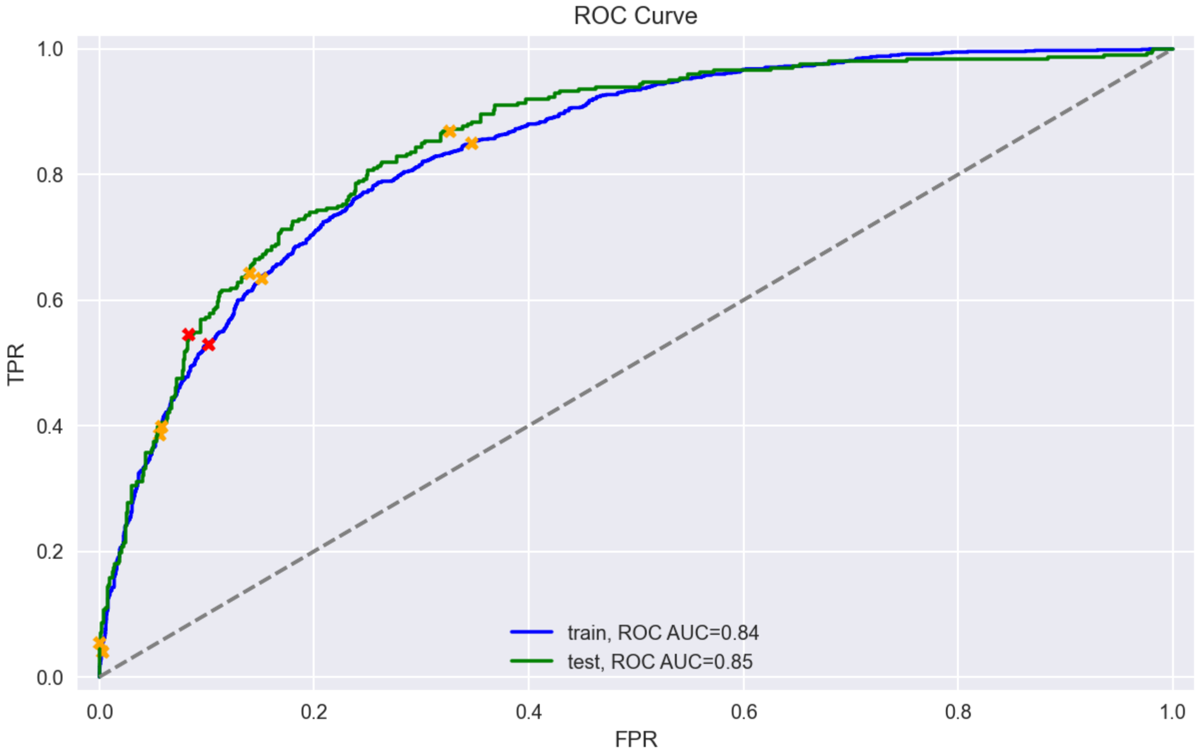
It means that to pass the sanity check our model must do better than 0.27 on the Accuracy metric.

Logistic regression

In [692…

```
LR = LogisticRegression(random_state=12345)
LR.fit(X_train, y_train)

result = evaluate_model(LR, X_train, y_train, X_valid, y_valid)
```

```
          train  test
Accuracy   0.80  0.82
ROC AUC    0.84  0.85
```



In [693…

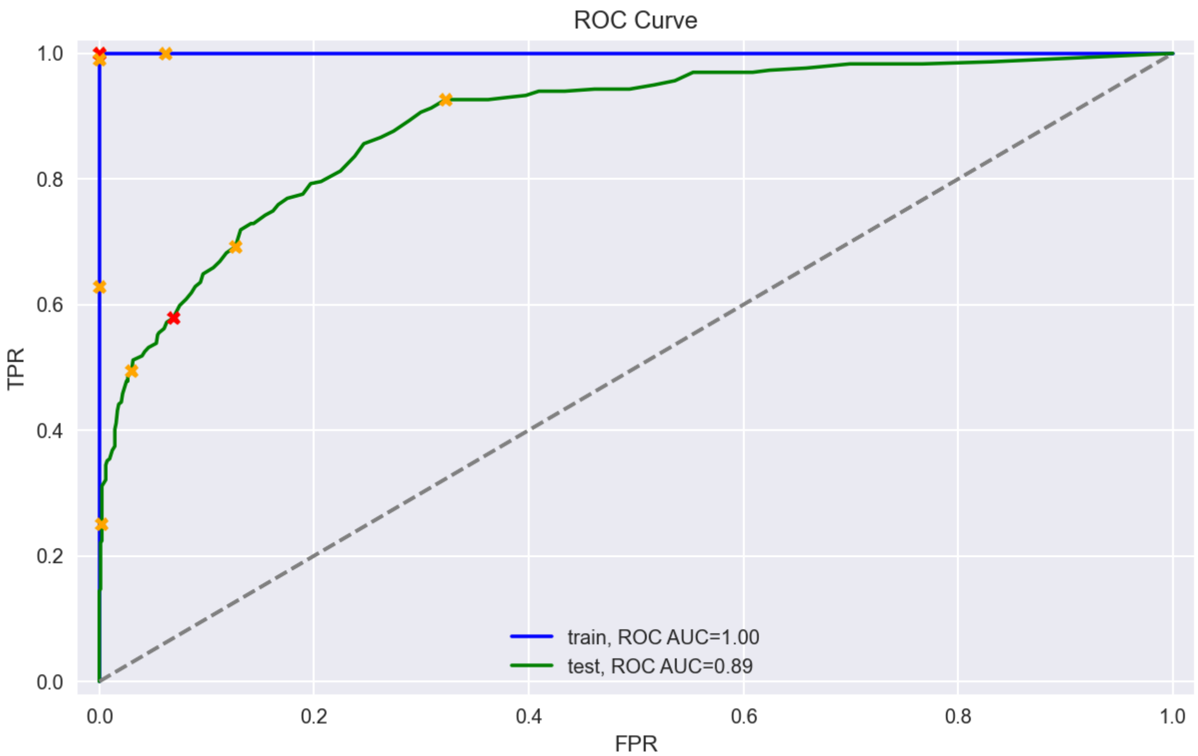
```
acc_LR = result['test']['Accuracy']
roc_auc_LR = result['test']['ROC AUC']
```

Random Forest

In [694…

```
rfc = RandomForestClassifier(random_state=12345)
rfc.fit(X_train, y_train)

result = evaluate_model(rfc, X_train, y_train, X_valid, y_valid)
```

```
          train  test
Accuracy    1.0  0.84
ROC AUC     1.0  0.89
```



In [695…

```
acc_rfc = result['test']['Accuracy']
```

```
roc_auc_rfc = result['test']['ROC AUC']
```

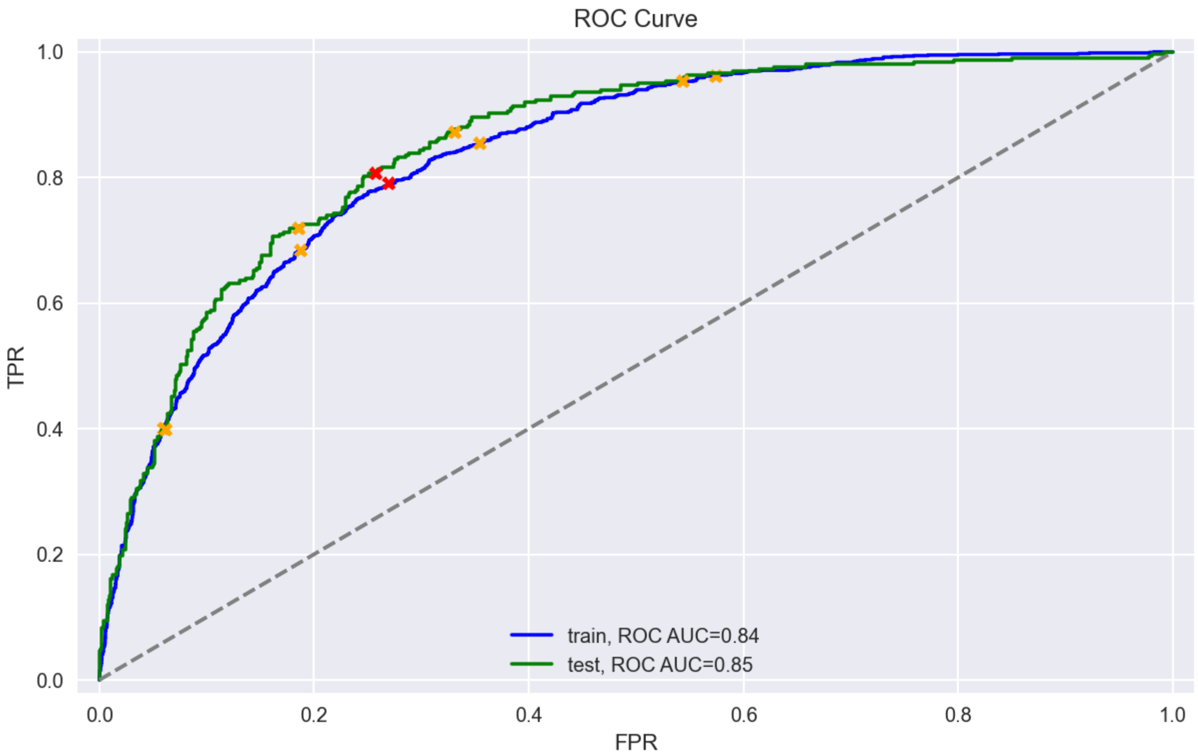
This model is overfitting to the train data, we should try to tune its hyperparameters.

LR + Class weight correction

In [696…

```
LR_class_weight = LogisticRegression(random_state=12345, class_weight='balanc
LR_class_weight.fit(X_train, y_train)

result = evaluate_model(LR_class_weight, X_train, y_train, X_valid, y_valid)
```

```
          train  test
Accuracy   0.75  0.76
ROC AUC    0.84  0.85
```



In [697…

```
acc_LR_class_weight = result['test']['Accuracy']
roc_auc_LR_class_weight = result['test']['ROC AUC']
```

LR + Upsampling

In [698…

```
def upsample(features, target, repeat):
    features_zeros = features[target == 0]
    features_ones = features[target == 1]
    target_zeros = target[target == 0]
    target_ones = target[target == 1]

    features_upsampled = pd.concat([features_zeros] + [features_ones] * repea
    target_upsampled = pd.concat([target_zeros] + [target_ones] * repeat)

    features_upsampled, target_upsampled = shuffle(
        features_upsampled, target_upsampled, random_state=12345)

    return features_upsampled, target_upsampled

X_train_upsampled, y_train_upsampled = upsample(X_train, y_train, 4)
```

In [699…

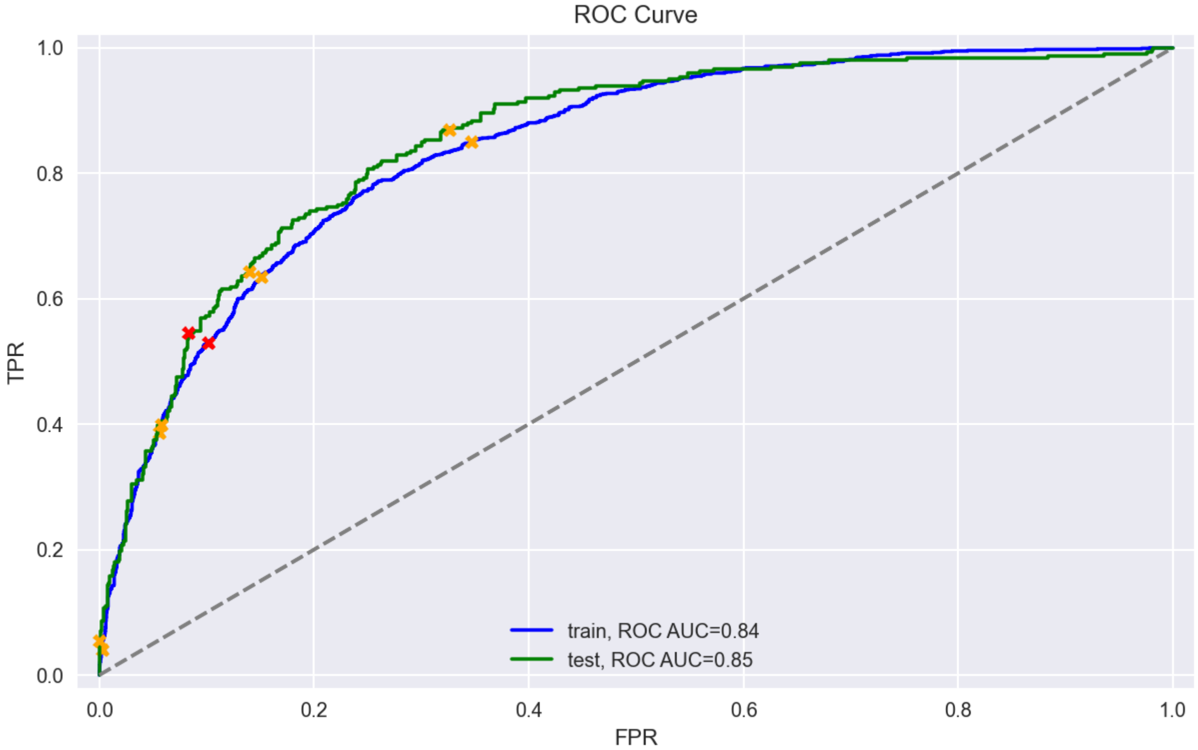
```
X_train_upsampled.shape
```

Out[699… (8095, 23)

In [700…

```
LR_upsample = LogisticRegression(random_state=12345)
LR_upsample.fit(X_train, y_train)

result = evaluate_model(LR_upsample, X_train_upsampled, y_train_upsampled, X_
```

```
          train  test
Accuracy   0.68  0.82
ROC AUC    0.84  0.85
```



In [701…

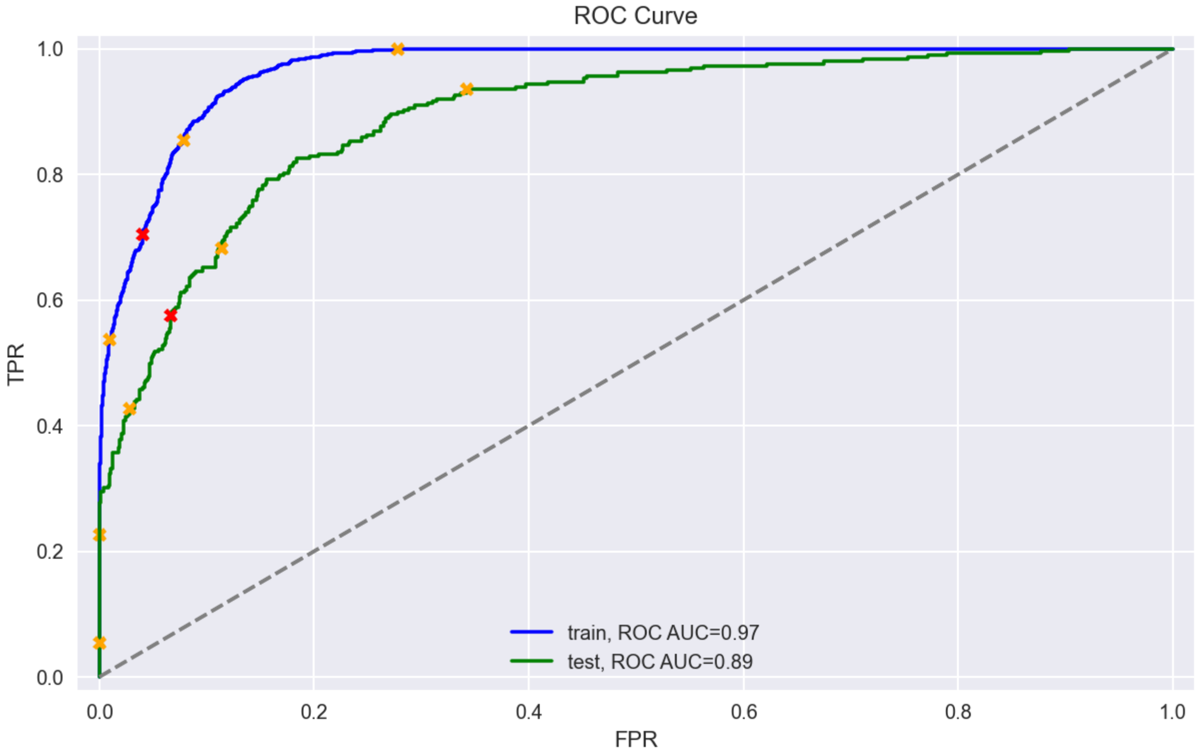
```
acc_LR_upsamp = result['test']['Accuracy']
roc_auc_LR_upsamp = result['test']['ROC AUC']
```

RFC + Hyperparameter tuning

In [702…

```
params = {"n_estimators" : [500, 700],
          "max_depth" : [6, 7, 8, 9, 10]}

gsSVR = GridSearchCV(estimator=rfc, param_grid=params, n_jobs=-1, verbose=0, 
gsSVR.fit(X_train, y_train)

result = evaluate_model(gsSVR, X_train, y_train, X_valid, y_valid)
```

```
          train  test
Accuracy   0.89  0.84
ROC AUC    0.97  0.89
```



In [703…

```
SVR_best = gsSVR.best_estimator_
SVR_best
```

Out[703…

```
RandomForestClassifier(max_depth=10, n_estimators=700, random_state=12345)
```

In [704…

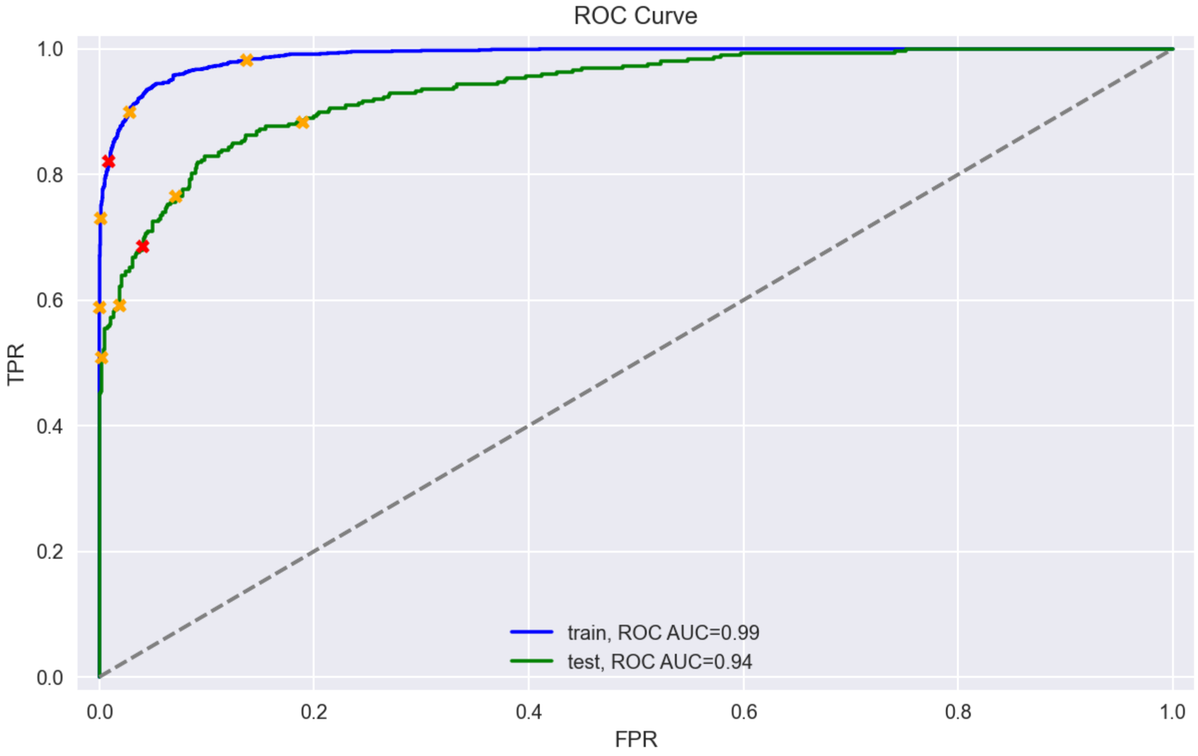
```
acc_rfc_tuned = result['test']['Accuracy']
roc_auc_rfc_tuned = result['test']['ROC AUC']
```

LGBM

In [705…

```
LGB = LGBMClassifier(random_state=12345)
LGB.fit(X_train, y_train)

result = evaluate_model(LGB, X_train, y_train, X_valid, y_valid)
```

	train	test
Accuracy	0.94	0.89
ROC AUC	0.99	0.94

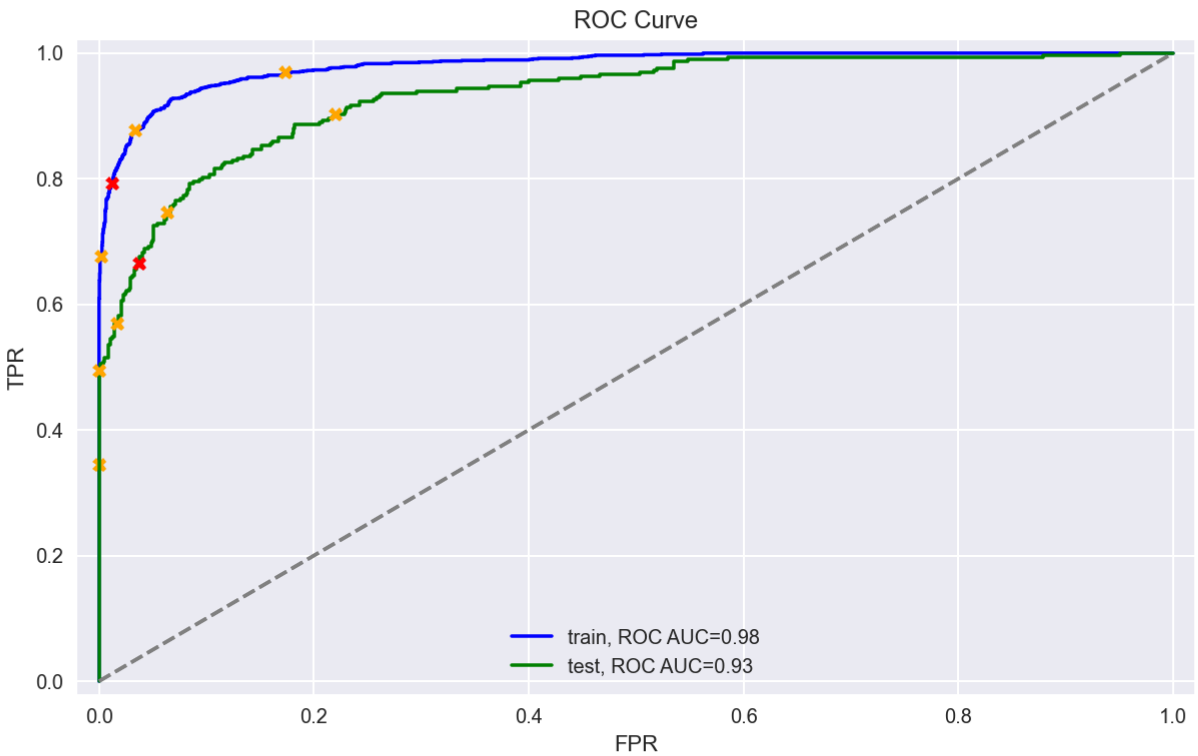


In [706…
```
acc_LGBM = result['test']['Accuracy']
roc_auc_LGBM = result['test']['ROC AUC']
```

CatBoost

In [707…
```
CB = CatBoostClassifier(random_state=12345, verbose=0)
CB.fit(X_train, y_train)

result = evaluate_model(CB, X_train, y_train, X_valid, y_valid)
```

```
          train  test
Accuracy   0.94  0.88
ROC AUC    0.98  0.93
```



In [708…
```
acc_cat_boost = result['test']['Accuracy']
```

```
roc_auc_cat_boost = result['test']['ROC AUC']
```

Model selection

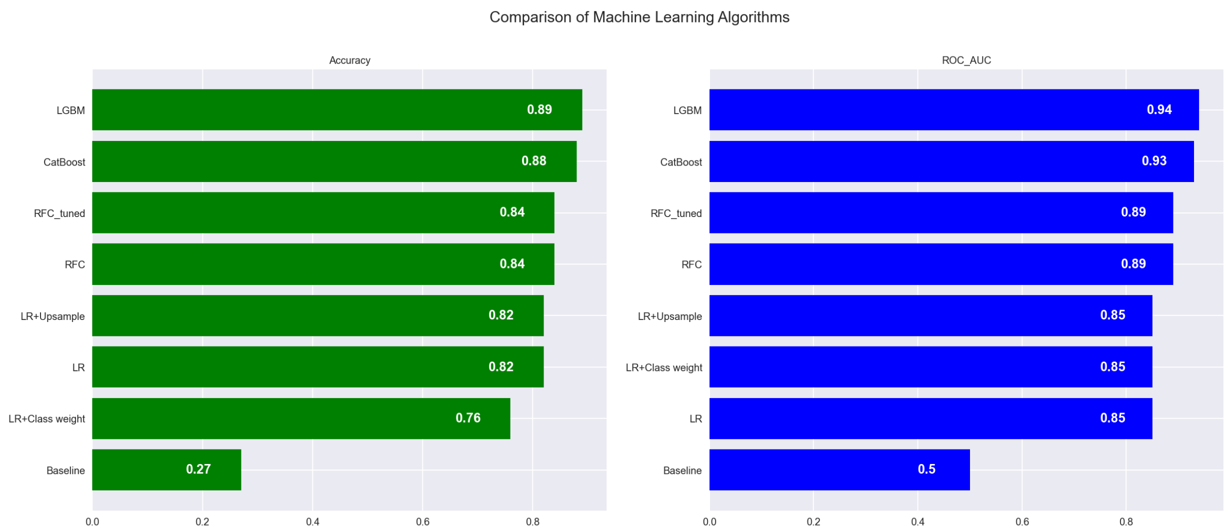
In [709…

```
models = pd.DataFrame({
    'Model': ['Baseline', 'LR', 'RFC', 'LR+Class weight', 'LR+Upsample', 'RFC_
    'Accuracy': [acc_baseline, acc_LR, acc_rfc, acc_LR_class_weight, acc_LR_u
    'ROC_AUC': [roc_auc_baseline, roc_auc_LR, roc_auc_rfc, roc_auc_LR_class_w
})
```

In [710…

```
fig, axs = plt.subplots(1,2,figsize=(20,8))
fig.suptitle('Comparison of Machine Learning Algorithms', fontsize=15)

labels = models.sort_values(by='Accuracy')['Model']
values = models.sort_values(by='Accuracy')['Accuracy']
axs[0].barh(labels, values, color = 'g')
axs[0].set_title('Accuracy', fontsize=10)
axs[0].set_yticklabels(labels, fontsize=10)
for counter, value in enumerate(values):
    axs[0].text(value - 0.1, counter, round(value,2), color='white', va='cent

labels = models.sort_values(by='ROC_AUC')['Model']
values = models.sort_values(by='ROC_AUC')['ROC_AUC']
axs[1].barh(labels, values, color = 'b')
axs[1].set_title('ROC_AUC', fontsize=10)
axs[1].set_yticklabels(labels, fontsize=10)
for counter, value in enumerate(values):
    axs[1].text(value - 0.1, counter, round(value,2), color='white', va='cent
;
```

Out[710… ''



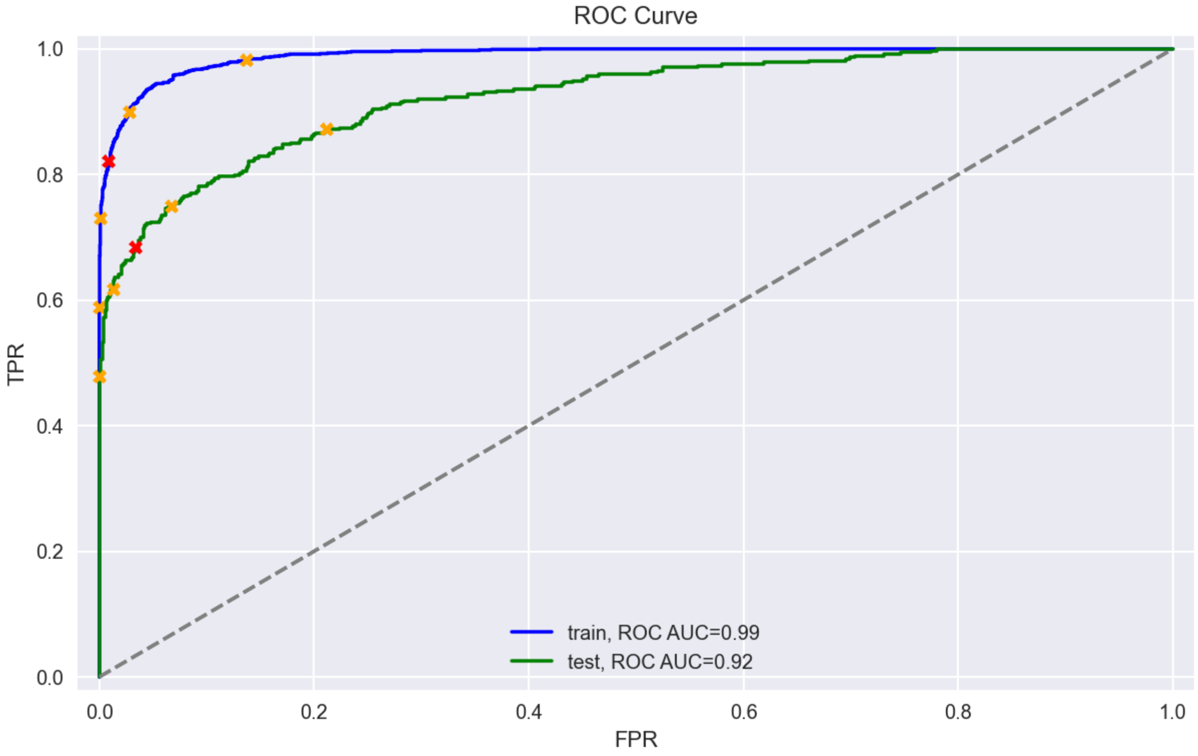
LGBM is showing the best results for our dataset, it has also exceeded the targeted metric, so we will be choosing this model as our final classifier.

Test the model

In [711…

```
result = evaluate_model(LGB, X_train, y_train, X_test, y_test)
```

```
           train  test
Accuracy    0.94  0.89
ROC AUC     0.99  0.92
```



In [712…
```
round((0.89 - 0.94)/0.89 * 100, 2)
```

Out[712…
```
-5.62
```

The test score is 5.62% lower than the train score, overfitting is rather minor. The target metric is met.

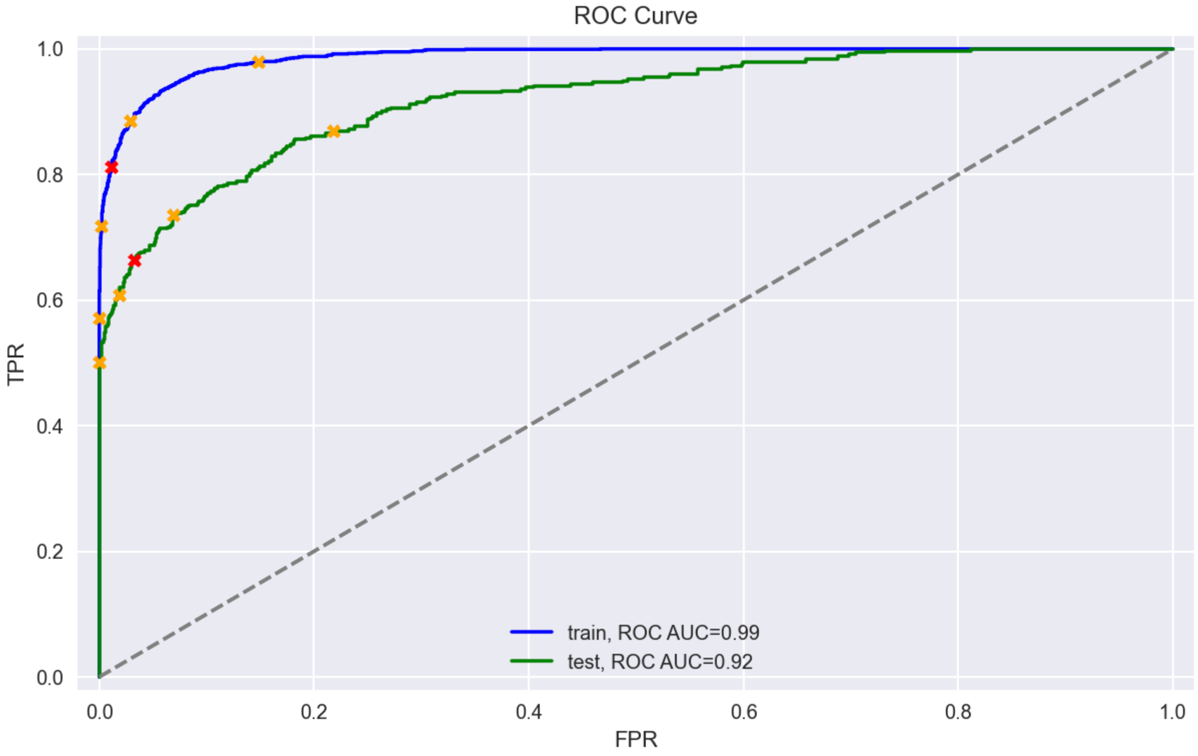
Retrain the model

Let's retrain the best model on the whole training set and test it on the test set

In [714…
```
X_train, X_test, y_train, y_test = train_test_split(X_OHE, y, test_size = 0.2
```

In [715…
```
LGB = LGBMClassifier(random_state=12345)
LGB.fit(X_train, y_train)

result = evaluate_model(LGB, X_train, y_train, X_test, y_test)
```

```
           train  test
Accuracy    0.94  0.89
ROC AUC     0.99  0.92
```



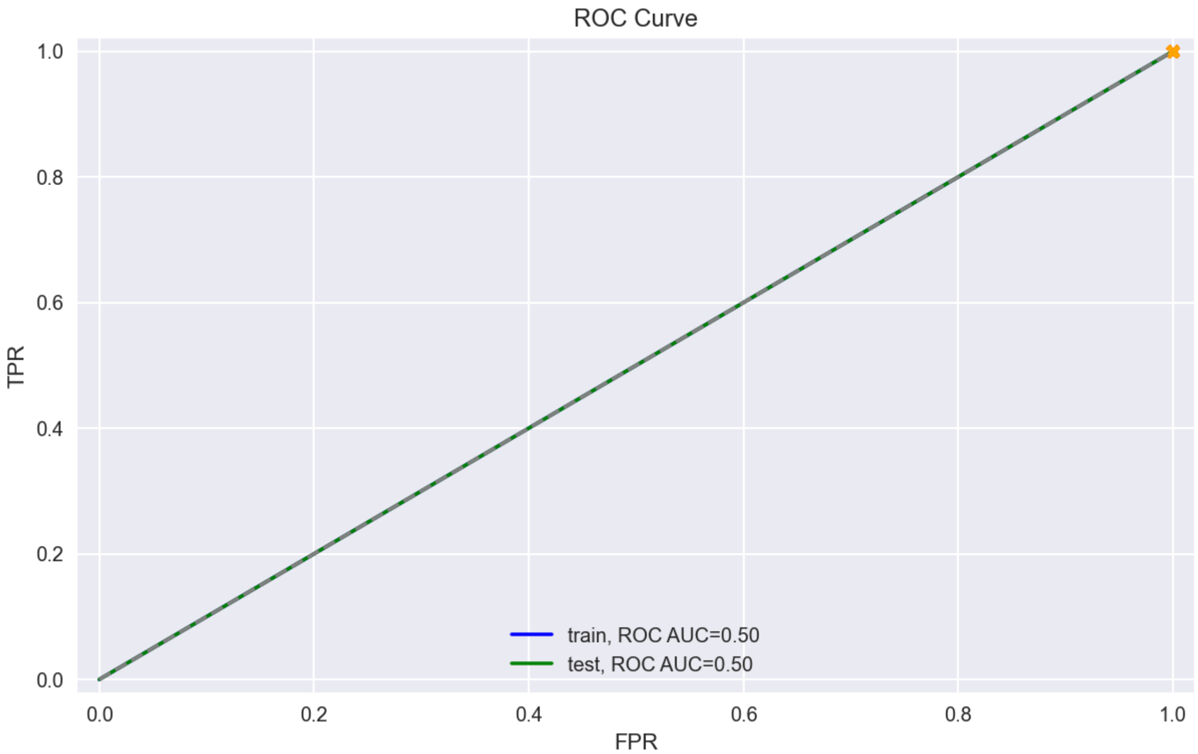
It didn't change the scores, it confirmed our previous findings.

Sanity check

In [716…

```
base_model = DummyClassifier(strategy='constant', constant=1, random_state=12
base_model.fit(X_train, y_train)

result = evaluate_model(base_model, X_train, y_train, X_test, y_test)
```

```
          train  test
Accuracy   0.27  0.27
ROC AUC    0.50  0.50
```



In [718…

```
round((0.92-0.5)/0.92 * 100, 2)
```

Out[718… 45.65

The test score of our final chosen model is 45.65% higher than that of the Dummy classifier model, that we use as a baseline to analyze the model quality. It means that the modeling was useful.

Conclusion

The **goal** of this project was to develop a binary classification model that predicts whether a customer will leave the company soon based on the clientele's personal data, including information about their plans and contracts.

The ROC_AUC metric on the test set had to be no more than **0.88**.

We have completed the following steps in this project:

1. Descriptive statistics

We have merged 3 provided dataframes and studied the input.

2. Data preprocessing

First, we converted the target variable into a binary numeric one. Then we have converted all column names to lower case letters. We checked the data for duplicates.

Next step was to fill in missing values. It was done by filling them with the 'No' value for those cases when clients were not using the service. The missing `internet_service` values we will replace with the 'not_available' value.

Then we made some necessary changes in the data types of a few variables. Finally, we have converted all categorical variables to numeric using the OHE technique.

3. EDA

We examined both the features and the target.

There are a lot of observations with the `total_charges` values less than 100 dollars. These are most likely either new clients or those who quickly left the company after just a month. There are quite a lot of 'loyal' customers as well who stayed with the company for more than 5 years based on the days_since_join variable distribution. A big amount of clients pay around 20 dollars a month for Telecom services. Otherwise, there were no visible outliers.

We noticed that our target classes were imbalanced: there are at least twice as many observations for the customers who stayed with Telecom than for those who left. On average, there is a 26.5% probability that a customer will leave the company.

4. Additional assignment

We compared the monthly payment distribution (`monthly_charges`) of all active clients with the clients who have left. The biggest difference between the 2 groups is that there is a big number of active clients who pay only 20 dollars monthly for Telecom services. Exited clients used to pay more, on average. Maybe that was one of the reasons why they left the company.

Then we compared the behavior of the clients from the two above groups for both the usage of the internet services and the phone services.

The share of the internet users from the 'active' clients is 4 times smaller than that of the 'exited' users. There are more 'active' clients who are not using the internet than in the 'exited' clients group. We also see that the network is set up through a fiber optic cable for the 'exited' clients twice as many times as for the 'active' clients. This observation could mean that the quality of the fiber cable internet is lower than that of the DSL internet and that was one of the reasons for the customers to leave the company.

The phone services usage distribution is almost the same for both groups. There are slightly (3.5%) more 'active' users who are not using multiple lines than in the 'exited' group but the difference is not very significant.

5. Splitting the data

Data was split into train, validation and test sets with the 20/80 ratio.

6. Standard scaling

It was performed to be able to compare feature importances.

7. Model selection

We have compared Linear Regression, Random Forest, LightGBM and CatBoost models. We have also tuned a few hyperparameters for the Random Forest algorithm. We have chosen the LGBM model based on the ROC_AUC and Accuracy scores.

8. Test the model

The test score is 5.62% lower than the train score, no overfitting observed. The target metric was met.

9. Sanity check

The test score of our final chosen model is 45.65% higher than that of the Dummy classifier model, that we use as a baseline to analyze the model quality. It means that the modeling was useful.

Results

The LGBM model has shown the best results (**test ROC_AUC of 0.92**) in terms of quality. The target metric of 0.88 or higher has been reached.