Recommendation of the mobile phone plans for the mobile operator's company

After the completed analysis in Project 5 it's required to create a system that could predict the clients behavior and suggest to the client to switch on a new plans (such as "Smart" and "Ultra"). Using the provided data from project 5 it's required to train the classification models for selection of the optimal plan for clients.

Additional tasks:

- to get the accuracy score on models testing higher than 0.75;
- check the efficacy of the models

Table of Content

Data import and overview

Libraries import

In [1]:
```
import pandas as pd
import math
from sklearn.tree import DecisionTreeClassifier
```

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

Data loading

In [2]:
```python
df_data = pd.read_csv('users_behavior.csv')
df_data.head()
```

Out[2]:

	calls	minutes	messages	mb_used	is_ultra
0	40.0	311.90	83.0	19915.42	0
1	85.0	516.75	56.0	22696.96	0
2	77.0	467.66	86.0	21060.45	0
3	106.0	745.53	81.0	8437.39	1
4	66.0	418.74	1.0	14502.75	0

In [3]:
```python
df_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3214 entries, 0 to 3213
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   calls     3214 non-null   float64
 1   minutes   3214 non-null   float64
 2   messages  3214 non-null   float64
 3   mb_used   3214 non-null   float64
 4   is_ultra  3214 non-null   int64
dtypes: float64(4), int64(1)
memory usage: 125.7 KB
```

Conclusion

1) **Data was successfully loaded, the target columns is named 'is_ultra', other columns to be used as parameters for model training.**

2) **Dataset has 3214 rows and 5 columns: quantity of calls, used minutes, used messages, used internet traffic and type of plan.**

Splitting of dataset to samples

In [4]:
```
# set the columns _is ultra as target, other as features
features = df_data.drop(columns = 'is_ultra')
target = df_data.is_ultra
```

In [5]:
```
# split the data to train and valid samples
features_train,features_valid_temp,target_train,target_valid_temp = train_test_split(features,target,
                                                                     test_size=0.4, random_state=12345)
```

In [6]:
```
# percentage check
features_train['calls'].count()/features['calls'].count()
```

Out[6]:
```
0.5998755444928439
```

In [7]:
```
# percentage check
features_valid_temp['calls'].count()/features['calls'].count()
```

Out[7]:
```
0.4001244555071562
```

In [8]:
```
# features check
features_valid_temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1286 entries, 1415 to 711
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   calls     1286 non-null   float64
 1   minutes   1286 non-null   float64
 2   messages  1286 non-null   float64
 3   mb_used   1286 non-null   float64
dtypes: float64(4)
memory usage: 50.2 KB
```

In [9]:
```
# splitting data to test and valid
features_valid,features_test,target_valid,target_test = train_test_split(features_valid_temp,target_valid_temp,test_size=0.5, ra
```

In [10]:
```
# percentage check
features_valid['calls'].count()/features['calls'].count()
```

Out[10]:
```
0.2000622277535781
```

In [11]:
```
# percentage check
features_test['calls'].count()/features['calls'].count()
```

Out[11]:
```
0.2000622277535781
```

Conclusion

Dataset was splat on target and features, train sample has 60% of data, valid and test 20%

3 Model training

Decision tree model training (model_a)

Searching for optimal depth (from 1 to 10)

In [12]:
```
best_model_a = 'none'
best_accuracy_a = 0
```

```python
best_depth_a = 0
for depth in range (1,10):
    model_a = DecisionTreeClassifier(random_state=12345, max_depth = depth)
    model_a.fit(features_train, target_train)
    predictions_valid_a = model_a.predict(features_valid)
    accuracy_a = accuracy_score(target_valid,predictions_valid_a)
    if accuracy_a > best_accuracy_a:
        best_model_a = model_a
        best_accuracy_a = accuracy_a
        best_depth_a = depth

print('\n','Best_model =',best_model_a, '\n','Best accuracy:',best_accuracy_a,'\n','depth:',best_depth_a)
```

```
 Best_model = DecisionTreeClassifier(max_depth=3, random_state=12345)
 Best accuracy: 0.7853810264385692
 depth: 3
```

Random forest model training (model_b)

Searching for optimal quantity of leaves (from 10 to 70 using step equal to 10) and optimal depth (from 1 to 10)

In [13]:

```python
best_model_b = 'none'
best_accuracy_b = 0
best_depth_b = 0
best_est = 0
for est in range(10,71,10):
    for depth in range (1,10):
        model_b = RandomForestClassifier(random_state=12345,n_estimators = est, max_depth = depth)
        model_b.fit(features_train, target_train)
        predictions_valid_b = model_b.predict(features_valid)
        accuracy_b = accuracy_score(target_valid,predictions_valid_b)
        if accuracy_b > best_accuracy_b:
            best_model_b = model_b
            best_accuracy_b = accuracy_b
            best_depth_b = depth
            best_est = est

print( '\n', 'Best model =',best_model_b, '\n','Best accuracy:',best_accuracy_b,'\n','Depth:',best_depth_b,
      '\n','Quantity of leaves =',best_est)
```

```
Best model = RandomForestClassifier(max_depth=8, n_estimators=40, random_state=12345)
Best accuracy: 0.8087091757387247
Depth: 8
Quantity of leaves = 40
```

Logistic regression model trainig (model_c)

In [14]:
```
model_c = LogisticRegression(random_state=12345)
model_c.fit(features_train, target_train)
predictions_valid_c = model_c.predict(features_valid)
accuracy_c = accuracy_score(target_valid,predictions_valid_c)

print('\n','best model =',model_c, '\n','model accuracy:',accuracy_c)
```

```
best model = LogisticRegression(random_state=12345)
model accuracy: 0.7107309486780715
```

Conclusion

During model training the different types of the models were trained using different hyperparameters.

Best models were selected for further use.

Hyperparameters tuning

Hyperparameters tuning for random forest model using grid search

In [15]:
```
parameters = { 'n_estimators': range (10, 71, 10),
              'max_depth': range (1,10)}

grid_a = GridSearchCV(RandomForestClassifier(), parameters)
```

In [16]:
```
grid_a.fit(features_train, target_train)
```

Out[16]:

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [17]:

```
grid_a.best_params_
```

Out[17]:

```
{'max_depth': 8, 'n_estimators': 70}
```

In [18]:

```
# display the accuracy of rf model with tuned hyperparameters
model_d = RandomForestClassifier(random_state=12345,n_estimators =grid_a.best_params_['n_estimators'],max_depth = grid_a.best_pa
model_d.fit(features_train, target_train)
predictions_valid_d = model_d.predict(features_valid)
accuracy_d = accuracy_score(target_valid,predictions_valid_d)
accuracy_d
```

Out[18]:

```
0.7978227060653188
```

Hyperparameters tuning for randomforest model usig random search

In [19]:

```
grid_b = RandomizedSearchCV(RandomForestClassifier(),parameters)
```

In [20]:

```
grid_b.fit(features_train, target_train)
```

Out[20]:

```
RandomizedSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [21]:

```
grid_b.best_params_
```

Out[21]:

```
{'n_estimators': 30, 'max_depth': 8}
```

In [22]:

```
# display the accuracy of rf model with tuned hyperparameters
model_e = RandomForestClassifier(random_state=12345,n_estimators =grid_b.best_params_['n_estimators'],max_depth = grid_b.best_pa
model_e.fit(features_train, target_train)
```

```
predictions_valid_e = model_e.predict(features_valid)
accuracy_e = accuracy_score(target_valid,predictions_valid_e)
accuracy_e
```

Out[22]:

```
0.7993779160186625
```

Models testing

model_a testing

In [23]:

```
test_predictions_a = best_model_a.predict(features_test)
test_accuracy_a =  accuracy_score(target_test,test_predictions_a)

print('model_a accuracy =', test_accuracy_a)
```

```
model_a accuracy = 0.7791601866251944
```

model_b testing

In [24]:

```
test_predictions_b = best_model_b.predict(features_test)
test_accuracy_b =  accuracy_score(target_test,test_predictions_b)

print('model_b accuracy = ', test_accuracy_b)
```

```
model_b accuracy =  0.7962674961119751
```

model_c testing

In [25]:

```
test_predictions_c = model_c.predict(features_test)
test_accuracy_c =  accuracy_score(target_test,test_predictions_c)

print('model_c accuracy ', test_accuracy_c)
```

```
model_c accuracy  0.6842923794712286
```

model_d testing

In [26]:

```
test_predictions_d = model_d.predict(features_test)
test_accuracy_d =  accuracy_score(target_test,test_predictions_d)
```

```
print('model_d accuracy ', test_accuracy_d)
```

```
model_d accuracy  0.8055987558320373
```

model_e testing

In [27]:
```
test_predictions_e = model_e.predict(features_test)
test_accuracy_e =  accuracy_score(target_test,test_predictions_e)

print('model_e accuracy', test_accuracy_e)
```

```
model_e accuracy 0.7931570762052877
```

comparison of results

In [28]:
```
models_df = pd.DataFrame({'model_name': ['model_a','model_b','model_c','model_d','model_e'],
                          'model_accuracy': [test_accuracy_a,test_accuracy_b,test_accuracy_c,
                                             test_accuracy_e,test_accuracy_d,],
                         'prediction':[test_predictions_a,test_predictions_b,test_predictions_c,test_predictions_d,test_predicti
```

In [29]:
```
models_df = models_df.sort_values(by = 'model_accuracy', ascending = False).reset_index(drop = True)
```

In [30]:
```
models_df
```

Out[30]:

	model_name	model_accuracy	prediction
0	model_e	0.805599	[0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, ...
1	model_b	0.796267	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, ...
2	model_d	0.793157	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, ...
3	model_a	0.779160	[0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, ...
4	model_c	0.684292	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...

Conclusion

Models testing were done, models E, B and D have the best accuracy score - higher than 79%

Model efficacy testing

Creating of test dataset

In [31]:
```
test_df = features_test
test_df = test_df.join(target_test,rsuffix='r')
test_df = test_df.rename(columns={'is_ultrar': 'is_ultra'})
test_df
```

Out[31]:

	calls	minutes	messages	mb_used	is_ultra
160	61.0	495.11	8.0	10891.23	0
2498	80.0	555.04	28.0	28083.58	0
1748	87.0	697.23	0.0	8335.70	0
1816	41.0	275.80	9.0	10032.39	0
1077	60.0	428.49	20.0	29389.52	1
...	...	...	...	...	...
2401	55.0	446.06	79.0	26526.28	0
2928	102.0	742.65	58.0	16089.24	1
1985	52.0	349.94	42.0	12150.72	0
357	39.0	221.18	59.0	17865.23	0
2313	40.0	301.03	102.0	6057.63	0

643 rows × 5 columns

Monthly payment calculation

In [32]:
```
smart = pd.DataFrame({'messages_included':[50], 'mb_per_month_included': [15360],'minutes_included': [500],
                      'rub_monthly_fee':[550], 'rub_per_gb':[200],'rub_per_message': [3],'rub_per_minute':[3]})
```

```
ultra = pd.DataFrame({'messages_included':[1000], 'mb_per_month_included': [30720],'minutes_included': [3000],
                      'rub_monthly_fee':[1950], 'rub_per_gb':[150],'rub_per_message': [1],'rub_per_minute':[1]})
```

In [33]:
```
# function for calculation of monthly fee
def total_fee (df):
    calls = df['minutes']
    msgs = df['messages']
    internet = df['mb_used']
    tarif = df['is_ultra']
    if tarif == 0:
        total_fee=smart['rub_monthly_fee'][0]
        if calls > smart['minutes_included'][0]:
            total_fee += (calls-smart['minutes_included'][0])*smart['rub_per_minute'][0]
        if msgs>smart['messages_included'][0]:
            total_fee+= (msgs-smart['rub_per_message'][0])*3
        if internet > smart['mb_per_month_included'][0]:
            total_fee+= math.ceil((internet-smart['mb_per_month_included'][0])/1024)*smart['rub_per_gb'][0]
        return(total_fee)
    else:
        total_fee=ultra['rub_monthly_fee'][0]
        if calls > ultra['minutes_included'][0]:
            total_fee += (calls-ultra['minutes_included'][0])*ultra['rub_per_minute'][0]
        if msgs>ultra['messages_included'][0]:
            total_fee+= (msgs-ultra['rub_per_message'][0])*3
        if internet > ultra['mb_per_month_included'][0]:
            total_fee+= math.ceil((internet-ultra['mb_per_month_included'][0])/1024)*ultra['rub_per_gb'][0]
        return(total_fee)
```

In [34]:
```
test_df['total_fee'] = test_df.apply(total_fee,axis=1)
test_df
```

Out[34]:

	calls	minutes	messages	mb_used	is_ultra	total_fee
160	61.0	495.11	8.0	10891.23	0	550.00
2498	80.0	555.04	28.0	28083.58	0	3315.12
1748	87.0	697.23	0.0	8335.70	0	1141.69
1816	41.0	275.80	9.0	10032.39	0	550.00
1077	60.0	428.49	20.0	29389.52	1	1950.00
...	...	...	...	...	...	...
2401	55.0	446.06	79.0	26526.28	0	2978.00
2928	102.0	742.65	58.0	16089.24	1	1950.00
1985	52.0	349.94	42.0	12150.72	0	550.00
357	39.0	221.18	59.0	17865.23	0	1318.00
2313	40.0	301.03	102.0	6057.63	0	847.00

643 rows × 6 columns

Insert of data obtained from three models with best accuracy score to dataset and check it efficacy

In [35]:

```
test_df[models_df.iloc[0,0]]= models_df.iloc[0,2]
test_df[models_df.iloc[1,0]]= models_df.iloc[1,2]
test_df[models_df.iloc[2,0]]= models_df.iloc[2,2]

def new_tarif (df):
    if df['total_fee']>=1950:
        return(1)
    else:
        return(0)

test_df['correct_answer'] = test_df.apply(new_tarif,axis=1)
test_df
```

Out[35]:

	calls	minutes	messages	mb_used	is_ultra	total_fee	model_e	model_b	model_d	correct_answer
160	61.0	495.11	8.0	10891.23	0	550.00	0	0	0	0
2498	80.0	555.04	28.0	28083.58	0	3315.12	1	1	1	1
1748	87.0	697.23	0.0	8335.70	0	1141.69	1	0	0	0
1816	41.0	275.80	9.0	10032.39	0	550.00	0	0	0	0
1077	60.0	428.49	20.0	29389.52	1	1950.00	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...
2401	55.0	446.06	79.0	26526.28	0	2978.00	1	1	0	1
2928	102.0	742.65	58.0	16089.24	1	1950.00	0	0	0	1
1985	52.0	349.94	42.0	12150.72	0	550.00	0	0	0	0
357	39.0	221.18	59.0	17865.23	0	1318.00	0	0	0	0
2313	40.0	301.03	102.0	6057.63	0	847.00	1	1	1	0

643 rows × 10 columns

In [36]:

```
test_df[models_df.iloc[0,0]+'_check'] = test_df[models_df.iloc[0,0]] == test_df['correct_answer']
test_df[models_df.iloc[1,0]+'_check'] = test_df[models_df.iloc[1,0]] == test_df['correct_answer']
test_df[models_df.iloc[2,0]+'_check'] = test_df[models_df.iloc[2,0]] == test_df['correct_answer']
test_df
```

Out[36]:

	calls	minutes	messages	mb_used	is_ultra	total_fee	model_e	model_b	model_d	correct_answer	model_e_check	model_b_check	model_d_che
160	61.0	495.11	8.0	10891.23	0	550.00	0	0	0	0	True	True	Tr
2498	80.0	555.04	28.0	28083.58	0	3315.12	1	1	1	1	True	True	Tr
1748	87.0	697.23	0.0	8335.70	0	1141.69	1	0	0	0	False	True	Tr
1816	41.0	275.80	9.0	10032.39	0	550.00	0	0	0	0	True	True	Tr
1077	60.0	428.49	20.0	29389.52	1	1950.00	0	0	0	1	False	False	Fa
...	...	...	...	...	...	...	...	...	...	...	...	...	
2401	55.0	446.06	79.0	26526.28	0	2978.00	1	1	0	1	True	True	Fa
2928	102.0	742.65	58.0	16089.24	1	1950.00	0	0	0	1	False	False	Fa
1985	52.0	349.94	42.0	12150.72	0	550.00	0	0	0	0	True	True	Tr
357	39.0	221.18	59.0	17865.23	0	1318.00	0	0	0	0	True	True	Tr
2313	40.0	301.03	102.0	6057.63	0	847.00	1	1	1	0	False	False	Fa

643 rows × 13 columns

Efficacy Calculation

In [37]:

```
model_e_percentage = test_df.query('model_e_check == True')['model_e_check'].count()/test_df['model_e_check'].count()
model_e_percentage
```

Out[37]:

```
0.7247278382581649
```

In [38]:

```
model_d_percentage = test_df.query('model_d_check == True')['model_d_check'].count()/test_df['model_d_check'].count()
model_d_percentage
```

Out[38]:

```
0.7309486780715396
```

In [39]:

```
model_b_percentage = test_df.query('model_b_check == True')['model_b_check'].count()/test_df['model_b_check'].count()
model_b_percentage
```

Out[39]:

```
0.7278382581648523
```

Conclusion

Based on performed testing of model the models has the following efficacy:

- Model e efficacy is 72,4%
- Model d efficacy is 73.0%
- Model b efficacy is 72,78%

It's recommended to use the "model D" for dertmination of proposal to client to swith on different mobile plan

General Conclusion

1) Data was successfully loaded, the target columns is named 'is_ultra', other columns to be used as parameters for model training.

2) Dataset was splat on target and features and three samples: train sample has 60% of data, valid and test 20%

3) Random forest, Decision tree and Regression models were trained. The validation accuracy scores are following:

- Random Forest model 0.78
- Decision Tree model 0.80
- Logistic Regression model 0.71

4) Hyperparameters were tuned for random forest models. The validation accuracy scores are following:

- GridSearchCV 0.79
- RandomSearchCV 0.79

5) Models testing was successfully executed. The accuracy scores on the test sample are following:

- Random Forest model 0.77
- Decision Tree model 0.79
- Logistic Regression model 0.68

- GridSearchCV 0.79
- RandomSearchCV 0.8

6) Models efficacy were tested, the model with higher efficacy is "model D". Efficacy is 73%.