

Дисциплина: Методы и технологии машинного обучения

Уровень подготовки: бакалавриат

Направление подготовки: 01.03.02 Прикладная математика и информатика

Семестр: осень 2021/2022

In [1]:

```
# настройка ширины страницы блокнота .....
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:80% !important; }</style>"))

# расширение watermark для вывода информации о версиях пакетов
# https://github.com/rasbt/watermark
%load_ext watermark
```

Лабораторная работа №5: Методы, основанные на деревьях решений. Регрессионные деревья. Деревья классификации. Случайный лес. Бустинг.

В практических примерах ниже показано:

- как делать перекодировку признаков в номинальной и порядковой шкалах
- как вырастить дерево и сделать обрезку его ветвей
- как настроить модель бэггинга
- как вырастить случайный лес
- как настроить модель бустинга на деревьях решений
- как подбирать настроечные параметры моделей методом сеточного поиска

Точность всех моделей оценивается методом перекрёстной проверки по 5 блокам.

Модели: дерево классификации, бэггинг, случайный лес, бустинг, дерево регрессии

Данные: in-vehicle-coupon-recommendation.csv . Источник: [сайт Калифорнийского университета в Ирвине](#)

In [2]:

```
# выводим информацию о версиях python и пакетов
%watermark -a "aksyuk@github.com" -d -v -p numpy,pandas,matplotlib,sklearn
```

Author: aksyuk@github.com

Python implementation: CPython
Python version : 3.8.8
IPython version : 7.22.0

numpy : 1.20.1
pandas : 1.2.4
matplotlib: 3.3.4
sklearn : 0.24.1

Указания к выполнению

Загружаем пакеты

In [3]:

```
# загрузка пакетов: инструменты -----
# работа с массивами
import numpy as np
# фреймы данных
import pandas as pd
# графики
import matplotlib as mpl
# стили и шаблоны графиков на основе matplotlib
import seaborn as sns
# загрузка файлов по URL
import urllib
# проверка существования файла на диске
from pathlib import Path
# для форматирования результатов с помощью Markdown
from IPython.display import Markdown, display
# перекодировка категориальных переменных
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
# хи-квадрат тест на независимость по таблице сопряжённости
from scipy.stats import chi2_contingency
# для таймера
import time

# загрузка пакетов: данные -----
from sklearn import datasets

# загрузка пакетов: модели -----
# дерево классификации
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
# перекрёстная проверка и метод проверочной выборки
from sklearn.model_selection import cross_val_score, train_test_split
# для перекрёстной проверки и сеточного поиска
from sklearn.model_selection import KFold, GridSearchCV
# бэггинг
from sklearn.ensemble import BaggingClassifier
# случайный лес
from sklearn.ensemble import RandomForestClassifier
# бустинг
from sklearn.ensemble import GradientBoostingClassifier
# сводка по точности классификации
from sklearn.metrics import classification_report
```

In [4]:

```
# константы
# ядро для генератора случайных чисел
my_seed = 9212
# создаём псевдоним для короткого обращения к графикам
plt = mpl.pyplot
# настройка стиля и отображения графиков
# примеры стилей и шаблонов графиков:
# http://tonysyu.github.io/raw\_content/matplotlib-style-gallery/gallery.html
mpl.style.use('seaborn-whitegrid')
sns.set_palette("Set2")
# раскомментируйте следующую строку, чтобы посмотреть палитру
# sns.color_palette("Set2")
```

In [5]:

```
# функция форматирования результатов с использованием Markdown
def printmd(string):
    display(Markdown(string))

# функции для попарной конкатенации элементов двух списков
concat_func_md = lambda x, y: '`' + str(x) + "`:&ensp;&ensp;&ensp;&ensp;" + str(y)
concat_func = lambda x, y: str(x) + ' ' * 4 + str(y)

# функция, которая строит график важности признаков в модели случайного леса
# источник: https://www.analyseup.com/learn-python-for-data-science/python-random-forest
def plot_feature_importance(importance, names, model_type):
    # Create arrays from feature importance and feature names
    feature_importance = np.array(importance)
    feature_names = np.array(names)

    # Create a DataFrame using a Dictionary
    data = {'feature_names': feature_names, 'feature_importance': feature_importance}
    fi_df = pd.DataFrame(data)

    # Sort the DataFrame in order decreasing feature importance
    fi_df.sort_values(by=['feature_importance'], ascending=False,
                      inplace=True)

    # Define size of bar plot
    plt.figure(figsize=(10, 8))
    # Plot Searborn bar chart
    sns.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])
    # Add chart labels
    plt.title('Важность признаков в модели: ' + model_type)
    plt.xlabel('Важность признака')
    plt.ylabel('')
```

Загружаем данные

Набор данных можно загрузить напрямую по ссылке:

<https://raw.githubusercontent.com/aksyuk/MTML/main/Labs/data/in-vehicle-coupon-recommendation.csv>. Справочник к данным доступен по адресу:

https://github.com/aksyuk/MTML/blob/main/Labs/data/CodeBook_in-vehicle-coupon-recommendation.md.

Загружаем данные во фрейм и выясняем их размерность. В таблице много строк, поэтому для экономии времени загрузку сделаем в два шага: сначала скачаем таблицу и сохраним в папку `'./data'`, затем прочитаем её во фрейм. Перед скачиванием проверим, нет ли уже такого файла в папке с данными.

In [6]:

```
# путь к локальному файлу для сохранения
localFilePath = './data/in-vehicle-coupon-recommendation.csv'

# проверяем, нет ли уже такого файла на диске
if not Path(localFilePath):
    # загружаем таблицу и превращаем её во фрейм
    fileURL = 'https://raw.githubusercontent.com/aksyuk/MTML/main/Labs/data/in-v'
    # скачиваем
```

```

urllib.request.urlretrieve(fileURL, localFilePath)
print('Файл', localFilePath, 'успешно загружен с адреса ', fileURL, '\n')
else:
    print('Файл', localFilePath, 'уже есть на диске\n')

# читаем
DF_raw = pd.read_csv(localFilePath)

# выясняем размерность фрейма
print('Число строк и столбцов в наборе данных:\n', DF_raw.shape)

```

Файл ./data/in-vehicle-coupon-recommendation.csv уже есть на диске

Число строк и столбцов в наборе данных:
(12684, 26)

```

In [7]: # типы столбцов
        DF_raw.dtypes

```

```

Out[7]: destination      object
passanger               object
weather                 object
temperature             int64
time                   object
coupon                 object
expiration              object
gender                 object
age                   object
maritalStatus           object
has_children            int64
education               object
occupation              object
income                 object
car                   object
Bar                   object
CoffeeHouse             object
CarryAway               object
RestaurantLessThan20    object
Restaurant20To50        object
toCoupon_GEQ5min        int64
toCoupon_GEQ15min       int64
toCoupon_GEQ25min       int64
direction_same          int64
direction_opp           int64
Y                       int64
dtype: object

```

Проблема в том, что, судя по справочнику к данным, все столбцы таблицы являются категориальными. Однако некоторые (бинарные) воспринимаются как `int`, а остальные как `object`. Посмотрим на столбцы типа `int`.

```

In [8]: # первые 7 строк столбцов типа int64
        DF_raw.loc[:, DF_raw.columns[DF_raw.dtypes == 'int64']].head(7)

```

```

Out[8]:   temperature  has_children  toCoupon_GEQ5min  toCoupon_GEQ15min  toCoupon_GEQ25min  direction_
0          55           1             1             0             0
1          80           1             1             0             0

```

	temperature	has_children	toCoupon_GEQ5min	toCoupon_GEQ15min	toCoupon_GEQ25min	direction_
2	80	1	1	1	1	0
3	80	1	1	1	1	0
4	80	1	1	1	1	0
5	80	1	1	1	1	0
6	55	1	1	1	1	0

Функция построения дерева классификации `DecisionTreeClassifier()` требует числовых порядковых значений переменных. Видно, что столбцы типа `int64` либо порядковые (`temperature`), либо бинарные (все остальные), их преобразовывать нет необходимости. А вот столбцы типа `object` придётся кодировать вручную.

При этом на этапе предварительного анализа данных нам удобнее будет работать с исходными категориальными столбцами. Поэтому сейчас просто изменим тип столбцов `object` на `category`.

```
In [9]: # меняем тип столбцов на категориальные
for col in DF_raw.columns[DF_raw.dtypes == 'object'] :
    DF_raw[col] = DF_raw[col].astype('category')
```

Отложим 30% наблюдений для прогноза.

```
In [10]: # наблюдения для моделирования
DF = DF_raw.sample(frac=0.7, random_state=my_seed)
# отложенные наблюдения
DF_predict = DF_raw.drop(DF.index)
```

Предварительный анализ данных

Описательные статистики

Стандартный подсчёт статистик с помощью функции `describe()` бесполезен для категориальных столбцов, поэтому рассчитаем частоты категорий по каждому столбцу. Для вывода отчёта воспользуемся форматированием на Markdown.

```
In [61]: # считаем частоты по столбцам, учитывая пропуски
for col in DF.columns:
    freq_col = DF[col].value_counts(dropna=False)
    str_freqs = np.around(freq_col / sum(freq_col), 3).astype(str)
    str_names = freq_col.index.values.astype(str)
    # для вывода в html
    # printmd('***' + col + '***<br>' +
    #         '<br>'.join(list(map(concat_func_md, str_names, str_freqs))))
    # для сохранения в pdf
    print('\n', col, '\n',
          '\n'.join(list(map(concat_func, str_names, str_freqs))))
```

```
destination
No Urgent Place    0.494
Home    0.257
Work    0.249

passanger
Alone    0.578
Friend(s)    0.261
Partner    0.083
Kid(s)    0.077

weather
Sunny    0.794
Snowy    0.11
Rainy    0.095

temperature
80    0.514
55    0.303
30    0.183

time
6PM    0.255
7AM    0.249
10AM    0.179
2PM    0.161
10PM    0.155

coupon
Coffee House    0.317
Restaurant(<20)    0.221
Carry out & Take away    0.189
Bar    0.157
Restaurant(20-50)    0.116

expiration
1d    0.56
2h    0.44

gender
Female    0.513
Male    0.487

age
21    0.211
26    0.202
31    0.156
50plus    0.143
36    0.103
41    0.088
46    0.054
below21    0.042

maritalStatus
Married partner    0.401
Single    0.383
Unmarried partner    0.165
Divorced    0.042
Widowed    0.01

has_children
0    0.593
1    0.407

education
```

Some college - no degree	0.347	
Bachelors degree	0.328	
Graduate degree (Masters or Doctorate)		0.151
Associates degree	0.091	
High School Graduate	0.075	
Some High School	0.008	

occupation		
Unemployed	0.154	
Student	0.123	
Computer & Mathematical	0.11	
Sales & Related	0.088	
Education&Training&Library	0.07	
Management	0.063	
Office & Administrative Support	0.051	
Arts Design Entertainment Sports & Media		0.049
Business & Financial	0.042	
Retired	0.04	
Food Preparation & Serving Related	0.024	
Healthcare Support	0.021	
Healthcare Practitioners & Technical		0.019
Community & Social Services	0.018	
Legal	0.017	
Transportation & Material Moving	0.017	
Protective Service	0.016	
Architecture & Engineering	0.015	
Life Physical Social Science	0.014	
Personal Care & Service	0.012	
Construction & Extraction	0.012	
Installation Maintenance & Repair	0.012	
Production Occupations	0.007	
Building & Grounds Cleaning & Maintenance		0.004
Farming Fishing & Forestry	0.004	

income		
\$25000 - \$37499	0.161	
\$12500 - \$24999	0.143	
\$37500 - \$49999	0.141	
\$100000 or More	0.136	
\$50000 - \$62499	0.127	
Less than \$12500	0.087	
\$62500 - \$74999	0.07	
\$87500 - \$99999	0.069	
\$75000 - \$87499	0.065	

Bar		
never	0.411	
less1	0.276	
1~3	0.194	
4~8	0.09	
gt8	0.029	

CoffeeHouse		
less1	0.274	
1~3	0.264	
never	0.236	
4~8	0.131	
gt8	0.095	

CarryAway		
1~3	0.374	
4~8	0.337	
less1	0.148	
gt8	0.131	
never	0.01	

```

RestaurantLessThan20
1~3      0.428
4~8      0.287
less1    0.165
gt8      0.104
never    0.016

```

```

Restaurant20To50
less1    0.483
1~3      0.266
never    0.173
4~8      0.056
gt8      0.022

```

```

toCoupon_GEQ15min
1        0.565
0        0.435

```

```

toCoupon_GEQ25min
0        0.882
1        0.118

```

```

direction_same
0        0.784
1        0.216

```

```

Y
1        0.568
0        0.432

```

```
In [12]: str_freqs.astype(str)
```

```

Out[12]: 1        0.568
0        0.432
Name: Y, dtype: object

```

Обратим внимание на столбцы `car` и `toCoupon_GEQ5min`, которые есть в таблице, но отсутствовали в справочнике к данным. В первом (тип автомобиля) пропущено 99,1% наблюдений, во втором (до ресторана/кофейни, в которую выдан купон, более 5 минут езды) значения во всех наблюдениях одинаковы. Уберём эти столбцы из обучающих и отложенных данных.

```
In [13]: # выбрасываем столбцы с большинством пропусков или с нулевой дисперсией
# из обучающей выборки
DF = DF.drop(['car', 'toCoupon_GEQ5min'], axis=1)
# и из отложенных наблюдений
DF_predict = DF_predict.drop(['car', 'toCoupon_GEQ5min'], axis=1)
```

Ещё раз оценим количество пропусков.

```
In [14]: # считаем пропуски в столбцах, выводим ненулевые значения
nas = DF.isna().sum()
nas = np.around(nas / DF.shape[0], 3)
nas[nas > 0]
```

```

Out[14]: Bar                0.008
CoffeeHouse                0.017
CarryAway                  0.012

```



```
RestaurantLessThan20    0.011
Restaurant20To50        0.014
dtype: float64
```

Подсчитаем, сколько наблюдений мы потеряем, если выбросим все строки хотя бы с одним пропуском.

```
In [15]: na_rows = sum([True for idx, row in DF.iterrows() if any(row.isnull())])
print('Из-за пропусков пропадает ', na_rows, ' строк (',
      np.around(na_rows / DF.shape[0] * 100, 1), '%)', sep='')

```

Из-за пропусков пропадает 419 строк (4.7%)

Выводы по описательным статистикам: доли классов (Y) сопоставимы, наибольшее количество категорий у объясняющей переменной occupation . Строки с пропусками составляют не более 5%, поэтому мы уберём их из обучающей выборки.

```
In [16]: # выкидываем пропуски из обучающей
DF = DF.dropna()
DF.shape

```

Out[16]: (8460, 24)

```
In [17]: # выкидываем пропуски из отложенных наблюдений
DF_predict = DF_predict.dropna()
DF_predict.shape

```

Out[17]: (3619, 24)

Распределение предикторов внутри классов по зависимой переменной

Все объясняющие переменные являются категориальными, поэтому оценивать их связь с зависимой переменной с помощью корреляционной матрицы некорректно. Вместо этого можно воспользоваться [критерием согласия Хи-квадрат](#), который рассчитывается по таблице сопряжённости. Нулевая гипотеза теста: распределение долей в таблице сопряжённости случайно, т.е. два показателя независимы друг от друга.

Проведём тест для всех пар "объясняющая переменная" – "зависимая переменная" и выведем те пары, для которых соответствующее критерию р-значение больше 0.05 (т.е. нулевая гипотеза принимается, переменные независимы).

```
In [18]: for col in DF.columns[:24] :
          con_tab = pd.crosstab(DF[col], DF['Y'])
          c, p, dof, expected = chi2_contingency(con_tab)
          if p > 0.05 :
              print(col, 'и Y',
                    '\nH_0: переменные распределены независимо друг от друга',
                    '\nP-значение:', np.around(p, 4))

```

direction_same и Y

H_0: переменные распределены независимо друг от друга

P-значение: 0.1525

direction_opp и Y
H_0: переменные распределены независимо друг от друга
Р-значение: 0.1525

Интересный результат: полное совпадение р-значений – объясняется тем, что на самом деле `direction_same` и `direction_opp` противоположны друг другу. Связь между ними функциональная: если направление на ресторан/кофейню, в который предлагается купон, не совпадает с направлением на исходное место назначения (`direction_same == 0`), то оно противоположно (`direction_opp == 1`), и наоборот. Поэтому в модель имеет смысл включать только одну из этих переменных.

```
In [19]: # исключаем direction_opp
# из обучающей выборки
DF = DF.drop(['direction_opp'], axis=1)
# и из отложенных наблюдений
DF_predict = DF_predict.drop(['direction_opp'], axis=1)
```

Перекодировка номинальной и порядковой шкалы

Теперь перекодировем признаки так, чтобы воспользоваться функцией классификации на дереве решений. Начнём с тех, которые содержат признаки в номинальной шкале (между позициями нет отношения порядка). Перекодировем их в фиктивные с помощью функции `OneHotEncoder()`.

```
In [20]: # имена столбцов с номинальными показателями
nom_col_names = ['destination', 'passanger', 'weather', 'coupon', 'gender',
                 'maritalStatus', 'occupation']

# создаём объект кодировщика
one_hot = OneHotEncoder()

# кодируем, результат - массив
recoded = one_hot.fit_transform(DF[nom_col_names]).toarray()

# создаём из результата новый фрейм с фиктивными переменными
clmns = one_hot.get_feature_names(nom_col_names)
df_dummy_nom = pd.DataFrame(recoded, columns=clmns)

# выводим размерность итога
print(df_dummy_nom.shape)

# смотрим результат
df_dummy_nom.head()
```

(8460, 47)

Out [20]:

	destination_Home	destination_No Urgent Place	destination_Work	passanger_Alone	passanger_Friend(s)	passanger
0	0.0	1.0	0.0	0.0	0.0	
1	0.0	1.0	0.0	1.0	0.0	
2	1.0	0.0	0.0	0.0	0.0	

	destination_Home	destination_No Urgent Place	destination_Work	passanger_Alone	passanger_Friend(s)	passanger
3	0.0	0.0	1.0	1.0	0.0	
4	1.0	0.0	0.0	0.0	0.0	

5 rows × 7 columns

In [21]:

```
# исходник для сравнения
print(DF[nom_col_names].shape)
DF[nom_col_names].head()
```

(8460, 7)

Out[21]:

	destination	passanger	weather	coupon	gender	maritalStatus	occupation
5266	No Urgent Place	Kid(s)	Sunny	Bar	Male	Married partner	Retiree
1655	No Urgent Place	Alone	Sunny	Restaurant(<20)	Male	Unmarried partner	Personal Care & Service
8097	Home	Partner	Sunny	Carry out & Take away	Female	Married partner	Education&Training&Librarian
9400	Work	Alone	Rainy	Bar	Male	Single	Unemployed
10457	Home	Kid(s)	Sunny	Carry out & Take away	Female	Divorced	Office & Administrative Support

Теперь разбираемся с показателями в порядковой шкале. Для этого воспользуемся `OrdinalEncoder()`. Для начала убедимся, что на этапе исключения пропущенных всё прошло штатно, и значений 'nan', которые `OrdinalEncoder()` не умеет обрабатывать, не осталось.

In [22]:

```
# имена столбцов с порядковыми показателями
ord_col_names = ['time', 'expiration', 'age', 'education', 'income',
                 'Bar', 'CoffeeHouse', 'CarryAway', 'RestaurantLessThan20',
                 'Restaurant20To50']

# считаем пропуски в столбцах
for col in ord_col_names:
    print('Пропусков в столбце', col, ':',
          sum(DF[col].isnull().astype(int)))
```

```
Пропусков в столбце time : 0
Пропусков в столбце expiration : 0
Пропусков в столбце age : 0
Пропусков в столбце education : 0
Пропусков в столбце income : 0
Пропусков в столбце Bar : 0
Пропусков в столбце CoffeeHouse : 0
Пропусков в столбце CarryAway : 0
Пропусков в столбце RestaurantLessThan20 : 0
Пропусков в столбце Restaurant20To50 : 0
```

Всё отлично, пропусков нет, поэтому можно перекодировать все порядковые столбцы в одно

действие.

In [23]:

```
# создаём списки с порядком кодировки для каждого столбца
enc_time = ['7AM', '10AM', '2PM', '6PM', '10PM']
enc_expiration = ['2h', '1d']
enc_age = ['below21', '21', '26', '31', '36', '41', '46', '50plus']
enc_education = ['Some High School', 'High School Graduate',
                 'Some college - no degree', 'Associates degree',
                 'Bachelors degree',
                 'Graduate degree (Masters or Doctorate)']
enc_income = ['Less than $12500', '$12500 - $24999', '$25000 - $37499',
              '$37500 - $49999', '$50000 - $62499', '$62500 - $74999',
              '$75000 - $87499', '$87500 - $99999', '$100000 or More']
enc_how_often = ['never', 'less1', '1~3', '4~8', 'gt8']

# перекодировщик
ordinal = OrdinalEncoder(categories=[enc_time, enc_expiration, enc_age,
                                    enc_education, enc_income,
                                    enc_how_often, enc_how_often,
                                    enc_how_often, enc_how_often,
                                    enc_how_often])

# кодируем
df_ord = pd.DataFrame(ordinal.fit_transform(Df[ord_col_names]),
                      columns = ord_col_names)

# выводим размерность итога
print(df_ord.shape)

# результат
df_ord.head()
```

(8460, 10)

Out[23]:

	time	expiration	age	education	income	Bar	CoffeeHouse	CarryAway	RestaurantLessThan20	Res
0	1.0	0.0	7.0	5.0	5.0	0.0	3.0	2.0	3.0	
1	2.0	1.0	7.0	4.0	8.0	4.0	0.0	2.0	3.0	
2	4.0	0.0	7.0	2.0	4.0	1.0	0.0	3.0	2.0	
3	0.0	1.0	1.0	4.0	8.0	3.0	1.0	2.0	3.0	
4	4.0	0.0	4.0	2.0	2.0	0.0	4.0	3.0	3.0	

In [24]:

```
# исходник для сравнения
print(Df[ord_col_names].shape)
Df[ord_col_names].head()
```

(8460, 10)

Out[24]:

	time	expiration	age	education	income	Bar	CoffeeHouse	CarryAway	RestaurantLessTI
5266	10AM	2h	50plus	Graduate degree (Masters or Doctorate)	62500 — 74999	never	4~8	1~3	

	time	expiration	age	education	income	Bar	CoffeeHouse	CarryAway	RestaurantLessTI
1655	2PM	1d	50plus	Bachelors degree	\$100000 or More	gt8	never		1~3
8097	10PM	2h	50plus	Some college - no degree	50000 — 62499	less1	never		4~8
9400	7AM	1d	21	Bachelors degree	\$100000 or More	4~8	less1		1~3
10457	10PM	2h	36	Some college - no degree	25000 — 37499	never	gt8		4~8

Объединим результаты: исходно числовые столбцы, дамми для признаков в номинальной шкале и перекодированные признаки в порядковой шкале – во фрейм под названием `DF_num`.

In [25]:

```
# объединяем результаты перекодировки в один фрейм
DF_num = pd.concat([DF.loc[:, DF.dtypes == 'int64'].reset_index(),
                    df_dummy_nom, df_ord], axis=1)

print('Размерность обучающего фрейма после исключения NaN',
      '\nи перекодировки: ', DF_num.shape)

# результат
DF_num.head()
```

Размерность обучающего фрейма после исключения NaN
и перекодировки: (8460, 64)

Out[25]:

	index	temperature	has_children	toCoupon_GEQ15min	toCoupon_GEQ25min	direction_same	Y	des
0	5266	80	1	1	0	0	0	0
1	1655	55	1	0	0	0	0	1
2	8097	30	1	1	0	0	0	1
3	9400	55	0	1	1	0	0	1
4	10457	80	1	1	0	0	0	1

5 rows × 64 columns

Повторяем перекодировку для фрейма с отложенными наблюдениями `DF_predict`.

In [26]:

```
# перекодировка отложенных наблюдений
# номинальная шкала -----
# кодируем, результат - массив
recoded = one_hot.fit_transform(DF_predict[nom_col_names]).toarray()

# создаём из результата новый фрейм с фиктивными переменными
clmns = one_hot.get_feature_names(nom_col_names)
df_dummy_nom = pd.DataFrame(recoded, columns=clmns)
```

```

# порядковая шкала -----
# кодируем
df_ord = pd.DataFrame(ordinal.fit_transform(DF_predict[ord_col_names]),
                      columns = ord_col_names)

# объединяем результаты
DF_predict_num = pd.concat([DF_predict.loc[:,
    DF_predict.dtypes == 'int64'].reset_index(),
    df_dummy_nom, df_ord], axis=1)

print('Размерность фрейма с отложенными наблюдениями после исключения NaN',
      '\nи перекодировки: ', DF_predict_num.shape)

# результат
DF_predict_num.head()

```

Размерность фрейма с отложенными наблюдениями после исключения NaN
и перекодировки: (3619, 64)

Out[26]:

	index	temperature	has_children	toCoupon_GEQ15min	toCoupon_GEQ25min	direction_same	Y	des
0	30	80	0	0	0	0	0	
1	33	55	0	1	0	0	1	
2	42	55	0	0	0	0	1	
3	46	80	0	0	0	0	0	
4	49	80	0	1	0	0	1	

5 rows × 64 columns

Модель дерева

В этом разделе построим:

- дерево классификации
- дерево классификации с обрезкой ветвей

Дерево на всех признаках

Построим модель и выведем изображение дерева в виде текста.

In [27]:

```

# выращиваем дерево на всех объясняющих
X = DF_num.drop(['index', 'Y'], axis=1)
y = DF_num['Y']

# классификатор
cls_one_tree = DecisionTreeClassifier(criterion='entropy',
                                     random_state=my_seed)

tree_full = cls_one_tree.fit(X, y)

# выводим количество листьев (количество узлов)
tree_full.get_n_leaves()

```

Out[27]: 2220

```
In [28]: # глубина дерева: количество узлов от корня до листа
# в самой длинной ветви
tree_full.get_depth()
```

Out[28]: 32

Очевидно, дерево получилось слишком большое для отображения в текстовом формате. Графическая визуализация тоже не поможет в данном случае. Посчитаем показатели точности с перекрёстной проверкой.

```
In [29]: # будем сохранять точность моделей в один массив:
score = list()
score_models = list()

# считаем точность с перекрёстной проверкой, показатель Асс
cv = cross_val_score(estimator=cls_one_tree, X=X, y=y, cv=5,
                      scoring='accuracy')

# записываем точность
score.append(np.around(np.mean(cv), 3))
score_models.append('one_tree')

print('Асс с перекрёстной проверкой',
      '\ndля модели', score_models[0], ':', score[0])
```

Асс с перекрёстной проверкой
для модели one_tree : 0.671

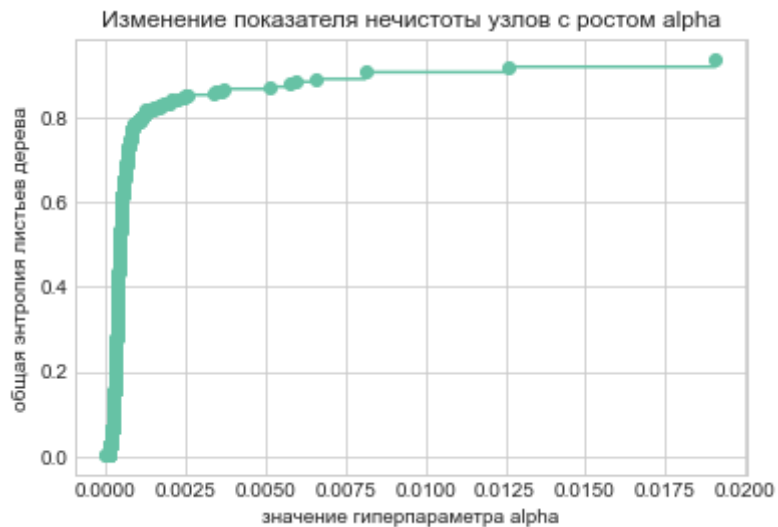
Дерево с обрезкой ветвей

Подберём оптимальное количество ветвей, которое максимизирует *Асс*, для экономии времени рассчитанный методом проверочной выборки.

```
In [30]: # рассчитываем параметры alpha для эффективных вариантов обрезки ветвей
path = cls_one_tree.cost_complexity_pruning_path(X, y)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
print('Всего значений alpha:', len(ccp_alphas))
print('Энтропия листьев для первых 5 значений alpha:', impurities[:5])
```

Всего значений alpha: 986
Энтропия листьев для первых 5 значений alpha: [0.00141844 0.00141844 0.00150767 0.0015969 0.00169793]

```
In [31]: # изображаем на графике
plt.plot(ccp_alphas[:-1], impurities[:-1], marker='o', drawstyle="steps-post")
plt.xlabel("значение гиперпараметра alpha")
plt.ylabel("общая энтропия листьев дерева")
plt.title("Изменение показателя нечистоты узлов с ростом alpha")
plt.show()
```



In [32]:

```
# обучающая и тестовая выборки, чтобы сэкономить время
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=my_seed)

# модели
clfs = list()

# таймер
tic = time.perf_counter()
# цикл по значениям alpha
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=my_seed, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)

# таймер
toc = time.perf_counter()
print(f"Расчёты по обрезке дерева заняли {toc - tic:0.2f} секунд")
```

Расчёты по обрезке дерева заняли 44.06 секунд

In [33]:

```
# извлекаем характеристики глубины и точности
# таймер
tic = time.perf_counter()
node_counts = [clf.tree_.node_count for clf in clfs]
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]
# таймер
toc = time.perf_counter()
print(f"Расчёты показателей точности заняли {toc - tic:0.2f} секунд")
```

Расчёты показателей точности заняли 5.90 секунд

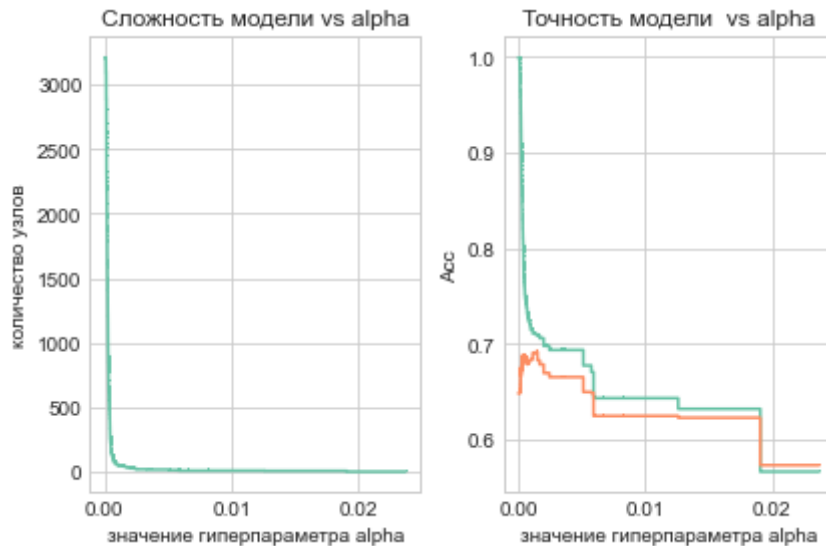
In [34]:

```
# изображаем на графике
fig, ax = plt.subplots(1, 2)

# график глубины дерева
ax[0].plot(ccp_alphas, node_counts, marker=',', drawstyle="steps-post")
ax[0].set_xlabel("значение гиперпараметра alpha")
ax[0].set_ylabel("количество узлов")
ax[0].set_title("Сложность модели vs alpha")
```



```
# график точности
ax[1].plot(ccp_alphas, train_scores, marker=',', label='train',
           drawstyle="steps-post")
ax[1].plot(ccp_alphas, test_scores, marker=',', label='test',
           drawstyle="steps-post")
ax[1].set_xlabel("значение гиперпараметра alpha")
ax[1].set_ylabel("Acc")
ax[1].set_title("Точность модели vs alpha")
fig.tight_layout()
```



Находим оптимальный размер дерева по максимуму *Acc* на тестовой выборке.

```
In [35]: # ОПТИМАЛЬНОЕ КОЛИЧЕСТВО УЗЛОВ
opt_nodes_num = node_counts[test_scores.index(max(test_scores))]

# считаем точность с перекрёстной проверкой, показатель Acc
cv = cross_val_score(estimator=clfs[opt_nodes_num], X=X, y=y, cv=5,
                     scoring='accuracy')

# записываем точность
score.append(np.around(np.mean(cv), 3))
score_models.append('pruned_tree')

print('Оптимальное количество узлов:', opt_nodes_num,
      '\nсоответствующая Acc на тестовой:', np.around(max(test_scores), 3),
      '\n\nAcc с перекрёстной проверкой',
      '\nдля модели', score_models[1], ':', score[1])
```

Оптимальное количество узлов: 45
соответствующая Acc на тестовой: 0.691

Acc с перекрёстной проверкой
для модели pruned_tree : 0.681

Посмотрим на характеристики глубины и сложности построенного дерева с обрезкой ветвей.

```
In [36]: # ВЫВОДИМ КОЛИЧЕСТВО ЛИСТЬЕВ (КОЛИЧЕСТВО УЗЛОВ)
clfs[opt_nodes_num].get_n_leaves()
```

Out[36]: 1096

```
In [37]: # глубина дерева: количество узлов от корня до листа
# в самой длинной ветви
clfs[opt_nodes_num].get_depth()
```

Out[37]: 25

Пример визуализации небольшого дерева

Лучшее дерево с обрезкой по-прежнему слишком велико для визуализации. Для примера нарисует одно из небольших деревьев с обрезкой и выведем его же в виде текста.

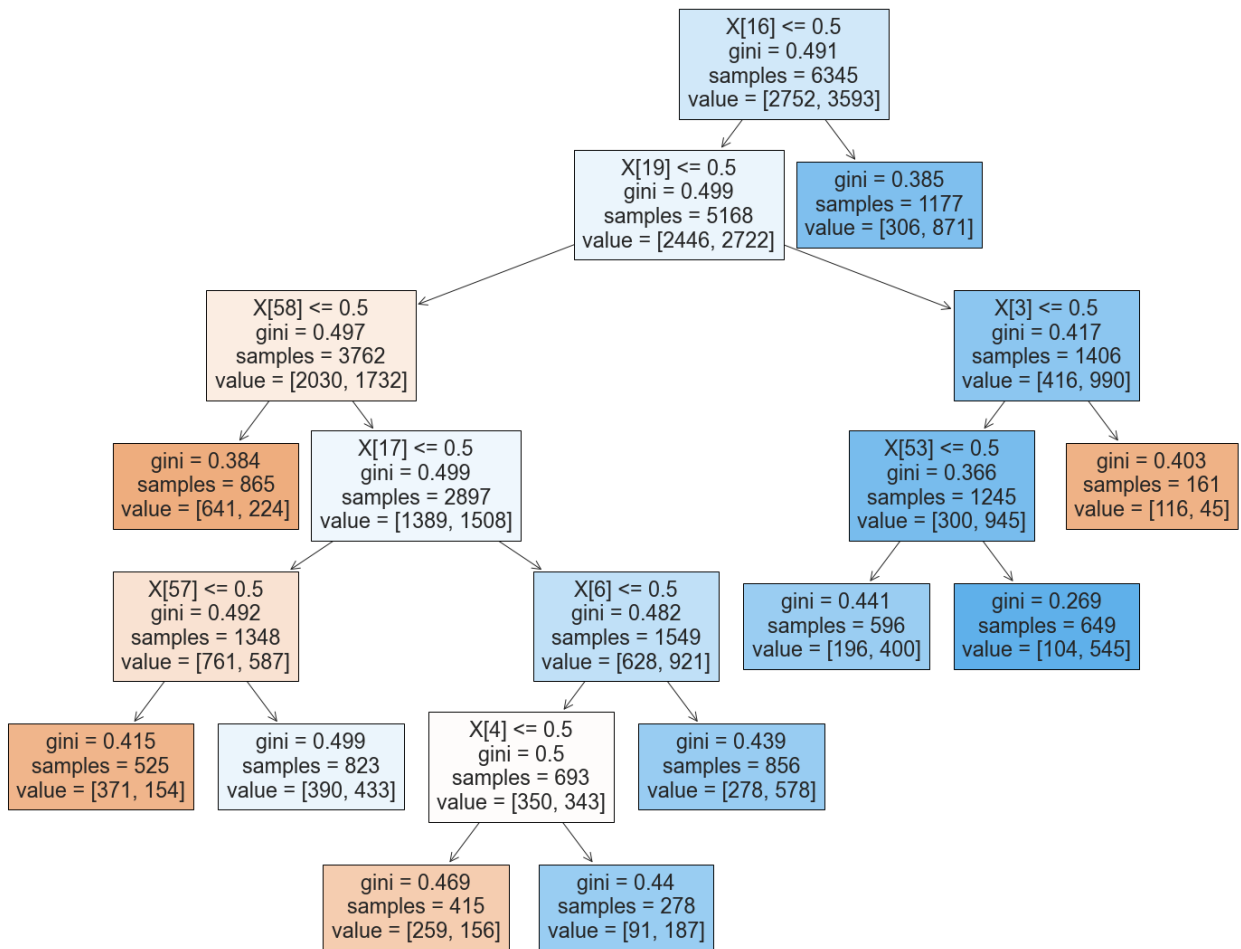
```
In [38]: # находим деревья с количеством листьев меньше 20
[i for i in node_counts if i < 20]
```

Out[38]: [19, 19, 19, 19, 17, 17, 17, 17, 13, 11, 9, 9, 9, 7, 1, 1]

```
In [39]: # визуализация на схеме НА ПРИМЕРЕ МАЛЕНЬКОГО ДЕРЕВА
nodes_num = 19
print('Количество узлов:', nodes_num,
      '\nТочность дерева на тестовой:',
      np.around(test_scores[node_counts.index(nodes_num)], 3))

fig = plt.figure(figsize=(25,20))
_ = plot_tree(clfs[node_counts.index(nodes_num)], filled=True)
```

Количество узлов: 19
Точность дерева на тестовой: 0.665



In [40]:

```

# визуализируем дерево в виде текстовой схемы
viz = export_text(clfs[node_counts.index(nodes_num)],
                  feature_names=list(X.columns))
print(viz)

|--- coupon_Carry out & Take away <= 0.50
|   |--- coupon_Restaurant(<20) <= 0.50
|   |   |--- CoffeeHouse <= 0.50
|   |   |   |--- class: 0
|   |   |   |--- CoffeeHouse > 0.50
|   |   |       |--- coupon_Coffee House <= 0.50
|   |   |       |   |--- Bar <= 0.50
|   |   |       |   |   |--- class: 0
|   |   |       |   |   |--- Bar > 0.50
|   |   |       |   |       |--- class: 1
|   |   |       |--- coupon_Coffee House > 0.50
|   |   |       |   |--- destination_No Urgent Place <= 0.50
|   |   |       |       |--- direction_same <= 0.50
|   |   |       |       |   |--- class: 0
|   |   |       |       |   |--- direction_same > 0.50
|   |   |       |       |       |--- class: 1
|   |   |       |       |--- destination_No Urgent Place > 0.50
|   |   |       |       |   |--- class: 1
|   |   |--- coupon_Restaurant(<20) > 0.50
|   |       |--- toCoupon_GEQ25min <= 0.50
|   |       |   |--- expiration <= 0.50
|   |       |   |   |--- class: 1
|   |       |   |   |--- expiration > 0.50
|   |       |   |       |--- class: 1

```

```
|      |      |--- toCoupon_GEQ25min > 0.50
|      |      |      |--- class: 0
|--- coupon_Carry out & Take away > 0.50
|      |--- class: 1
```

Бэггинг

Модель бэггинга использует бутстреп, чтобы вырастить B деревьев на выборках с повторами из обучающих данных. Построим модель для $B = 50$ деревьев.

```
In [41]: # параметр B: количество деревьев
num_trees = 50

# разбиения для перекрёстной проверки
kfold = KFold(n_splits=5, random_state=my_seed, shuffle=True)

# таймер
tic = time.perf_counter()
# модель с бэггингом
tree_bag = BaggingClassifier(base_estimator=cls_one_tree,
                             n_estimators=num_trees,
                             random_state=my_seed)
cv = cross_val_score(tree_bag, X, y, cv=kfold)

# таймер
toc = time.perf_counter()
print(f"Обучение модели с бэггингом на {num_trees:0.0f} деревьях",
      " и перекрёстной проверкой ",
      f"заняло {toc - tic:0.2f} секунд", sep='')
```

Обучение модели с бэггингом на 50 деревьях и перекрёстной проверкой заняло 8.14 секунд

```
In [42]: # точность
np.around(np.mean(cv), 3)
```

Out[42]: 0.746

Итак, мы построили модель, выбрав параметр B случайным образом. Воспользуемся функцией `GridSearchCV()`, чтобы перебрать 5 вариантов значений для параметра B .

```
In [43]: # настроим параметры бэггинга с помощью сеточного поиска
param_grid = {'n_estimators': [10, 20, 30, 40, 50]}

# таймер
tic = time.perf_counter()
clf = GridSearchCV(BaggingClassifier(DecisionTreeClassifier()),
                  param_grid, scoring='accuracy', cv=kfold)
tree_bag = clf.fit(X, y)
# таймер
toc = time.perf_counter()
print(f"Сеточный поиск занял {toc - tic:0.2f} секунд", sep='')
```

Сеточный поиск занял 24.61 секунд

```
In [44]: # точность лучшей модели
np.around(tree_bag.best_score_, 3)
```

Out[44]: 0.746

```
In [45]: # количество деревьев у лучшей модели
tree_bag.best_estimator_.get_params()['n_estimators']
```

Out[45]: 40

Таким образом, перебрав несколько вариантов для B , мы немного улучшили первоначальную точность модели бэггинга.

```
In [46]: # записываем точность
score.append(np.around(tree_bag.best_score_, 3))
score_models.append('bagging_GS')

print('Асс с перекрёстной проверкой',
      '\nдля модели', score_models[2], ': ', score[2])
```

Асс с перекрёстной проверкой
для модели bagging_GS : 0.746

Случайный лес

У модели случайного леса два настроечных параметра: количество деревьев B и количество признаков для построения отдельного дерева m . Настроим сеточный поиск для их подбора.

```
In [47]: # сколько столбцов в обучающих данных (p)
X_m = X.shape[1]
# возьмём значения для m: p, p/2, sqrt(p) и log2(p)
ms = np.around([X_m, X_m / 2, np.sqrt(X_m), np.log2(X_m)]).astype(int)
ms
```

Out[47]: array([62, 31, 8, 6])

```
In [48]: # настроим параметры случайного леса с помощью сеточного поиска
param_grid = {'n_estimators' : [10, 20, 30, 40, 50],
              'max_features' : ms}

# таймер
tic = time.perf_counter()
clf = GridSearchCV(RandomForestClassifier(DecisionTreeClassifier()),
                  param_grid, scoring='accuracy', cv=kfold)
random_forest = clf.fit(X, y)
# таймер
toc = time.perf_counter()
print(f"Сеточный поиск занял {toc - tic:0.2f} секунд", sep='')

Сеточный поиск занял 44.71 секунд
```

```
In [49]: # точность лучшей модели
np.around(random_forest.best_score_, 3)
```

Out[49]: 0.75

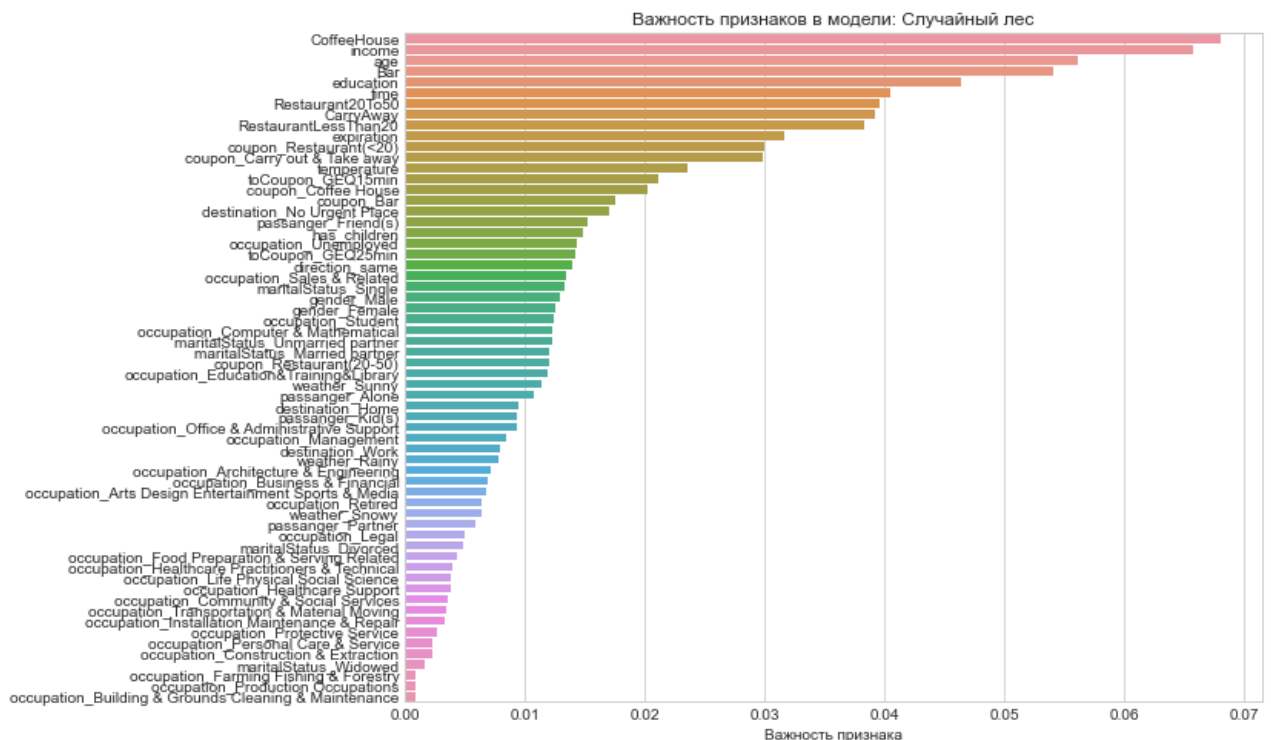
```
In [50]: # количество деревьев у лучшей модели
random_forest.best_estimator_.get_params()['n_estimators']
```

Out[50]: 40

```
In [51]: # количество объясняющих у лучшей модели
random_forest.best_estimator_.get_params()['max_features']
```

Out[51]: 31

```
In [52]: # рисуем график относительной важности каждого признака
plot_feature_importance(random_forest.best_estimator_.feature_importances_,
X.columns, 'Случайный лес')
```



```
In [53]: # записываем точность
score.append(np.around(random_forest.best_score_, 3))
score_models.append('random_forest_GS')

print('Асс с перекрёстной проверкой',
      '\nдля модели', score_models[3], ': ', score[3])
```

Асс с перекрёстной проверкой
для модели random_forest_GS : 0.75

Бустинг

Подберём сеточным поиском настроечные параметры модели:

- B – число деревьев,
- λ – скорость обучения,
- d – глубина взаимодействия предикторов.

```
In [54]: # обучаем модель с параметрами по умолчанию
clf_tst = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
                                     max_depth=1, random_state=my_seed)
cv = cross_val_score(clf_tst, X, y, cv=kfold, scoring='accuracy')
np.around(np.mean(cv), 3)
```

Out[54]: 0.688

```
In [55]: # настроим параметры бустинга с помощью сеточного поиска
param_grid = {'n_estimators' : [10, 20, 30, 40, 50],
              'learning_rate' : np.linspace(start=0.01, stop=0.25, num=15),
              'max_depth' : [1, 2]}

# таймер
tic = time.perf_counter()
clf = GridSearchCV(GradientBoostingClassifier(),
                  param_grid, scoring='accuracy', cv=kfold)
boost_tree = clf.fit(X, y)
# таймер
toc = time.perf_counter()
print(f"Сеточный поиск занял {toc - tic:0.2f} секунд", sep='')

Сеточный поиск занял 118.05 секунд
```

```
In [56]: # точность лучшей модели
np.around(boost_tree.best_score_, 3)
```

Out[56]: 0.712

```
In [57]: # параметры лучшей модели
print('n_estimators:',
      boost_tree.best_estimator_.get_params()['n_estimators'],
      '\nlearning_rate:',
      boost_tree.best_estimator_.get_params()['learning_rate'],
      '\nmax_depth:',
      boost_tree.best_estimator_.get_params()['max_depth'])
```

```
n_estimators: 50
learning_rate: 0.23285714285714287
max_depth: 2
```

```
In [58]: # записываем точность
score.append(np.around(boost_tree.best_score_, 3))
score_models.append('boost_tree_GS')

print('Асс с перекрёстной проверкой',
      '\nдля модели', score_models[4], ':', score[4])
```

Асс с перекрёстной проверкой
для модели boost_tree_GS : 0.712

Прогноз на отложенные наблюдения по лучшей модели

Ещё раз посмотрим на точность построенных моделей.

```
In [59]: # сводка по точности моделей
pd.DataFrame({'Модель' : score_models, 'Асс' : score})
```

```
Out[59]:
```

	Модель	Асс
0	one_tree	0.671
1	pruned_tree	0.681
2	bagging_GS	0.746
3	random_forest_GS	0.750
4	boost_tree_GS	0.712

Все модели показывают среднюю точность по показателю *Асс*, при этом самой точной оказывается модель случайного леса. Сделаем прогноз на отложенные наблюдения.

```
In [60]: # данные для прогноза
X_pred = DF_predict_num.drop(['index', 'Y'], axis=1)
# строим прогноз
y_hat = random_forest.best_estimator_.predict(X_pred)
# характеристики точности
print(classification_report(DF_predict_num['Y'], y_hat))
```

	precision	recall	f1-score	support
0	0.71	0.68	0.70	1547
1	0.77	0.80	0.78	2072
accuracy			0.75	3619
macro avg	0.74	0.74	0.74	3619
weighted avg	0.75	0.75	0.75	3619

Источники

1. Джеймс Г., Уиттон Д., Хасты Т., Тибширани Р. Введение в статистическое обучение с примерами на языке R. Пер. с англ. С.Э. Мاستицкого – М.: ДМК Пресс, 2016 – 450 с.
2. Рашка С. Python и машинное обучение: крайне необходимое пособие по новейшей предсказательной аналитике, обязательное для более глубокого понимания методологии машинного обучения / пер. с англ. А.В. Логунова. – М.: ДМК Пресс, 2017. – 418 с.: ил.
3. Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, Perry MacNeille A Bayesian Framework for Learning Rule Sets for Interpretable Classification / Journal of Machine

- Learning Research 18 (2017) 1-37. URL: <https://jmlr.org/papers/volume18/16-003/16-003.pdf>
4. *George Pipis* How to Run the Chi-Square Test in Python / medium.com. URL: <https://medium.com/swlh/how-to-run-chi-square-test-in-python-4e9f5d10249d>
 5. *Bernd Klein* What are Decision Trees? / python-course.eu. URL: https://www.python-course.eu/Decision_Trees.php
 6. Pruning decision trees - tutorial / kaggle.com. URL: <https://www.kaggle.com/arunmohan003/pruning-decision-trees-tutorial>
 7. Post pruning decision trees with cost complexity pruning / scikit-learn.org. URL: https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html
 8. *Piotr Płoński* Visualize a Decision Tree in 4 Ways with Scikit-Learn and Python / mljar.com. URL: <https://mljar.com/blog/visualize-decision-tree/>
 9. Random Forest Feature Importance Plot / www.analyseup.com. URL: <https://www.analyseup.com/learn-python-for-data-science/python-random-forest-feature-importance-plot.html>