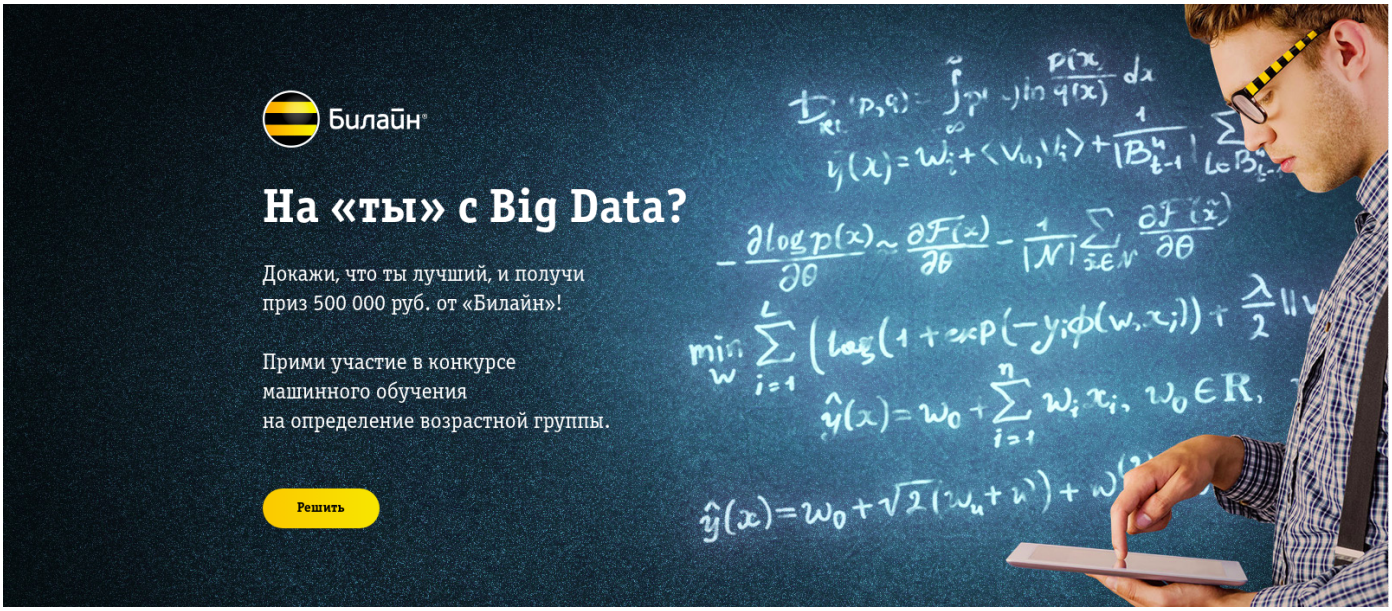


Задание

Мы постоянно работаем над улучшением тарифных планов и предлагаем своим клиентам новые услуги. Каждое предложение должно быть адресовано определенной возрастной группе. В этом задании вам предстоит попытаться определить возраст абонента по имеющимся данным.

Победитель конкурса

Место	Ник	Попадания	Попытки	Дата
1	Александр Куменко	76.39%	52	27.10.2015

https://special.habrahabr.ru/beeline/ (https://special.habrahabr.ru/beeline/)

In [1]:

```
import numpy as np
import pandas as pd
from scipy import sparse
from collections import Counter
import warnings

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import RandomizedSearchCV

from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

sns.set(font_scale=1.5, palette='Set2')
warnings.filterwarnings("ignore")
%matplotlib inline
```

started 14:08:14 2020-03-19, finished in 945ms

1. Чтение и изучение данных

Читаем данные с помощью `pandas` и делим их на признаки и целевую метку.

In [2]:

```
data = pd.read_csv('beeline_bigdata/train.csv')
data, labels = data.iloc[:, :-1], np.array(data['y'])
```

started 14:08:15 2020-03-19, finished in 657ms

Читаем также тестовые данные и истинные ответы к ним. Их будем использовать только для финального сравнения

In [3]:

```
x_test = pd.read_csv('beeline_bigdata/test.csv', index_col=0)
y_test = pd.read_csv('beeline_bigdata/ans.csv', index_col=0)
```

started 14:08:16 2020-03-19, finished in 588ms

Изучим, что у нас за данные

Что имеем

Анонимизированные данные об абонентах: регион, тарифный план, тип тарифного плана, информация об объёмах потребления различных услуг оператора и другие.

В файле train.csv содержится информация для построения модели. Формат строк: признаки (x0, ..., x61) и целевая переменная — y.

Файл test.csv содержит тестовое множество. Формат строк: ID, признаки (x0, ..., x61)

В общем-то больше ничего про данные не написано. Посмотрев на данные, видим, много категориальных признаков. Примером такого признака может быть город. Мы можем сравнивать, например, число жителей в этих городах, но это уже другой признак. Сами города мы сравнивать не можем.

In [4]:

```
1 data
```

started 14:08:16 2020-03-19, finished in 72ms

584c2d52	1746600cb0	1	1	0.524298	5624b8f759	...	0.000	-0.934382	-0.443953	0.000000	0.125000	0.37
584c2d52	1746600cb0	1	1	0.351012	5624b8f759	...	0.671	1.277250	0.707646	0.002564	0.633333	0.24
584c2d52	1746600cb0	1	1	0.437655	fc150fd13a	...	0.000	-0.955307	-0.521525	0.000000	0.068966	0.89
584c2d52	1746600cb0	1	1	-0.428777	f67f142e40	...	0.355	-0.423061	-0.783963	0.154930	0.316901	0.21
584c2d52	1746600cb0	1	1	-0.082204	4cf172e00e	...	0.000	-1.600326	-1.838680	NaN	NaN	
584c2d52	1746600cb0	1	0	NaN	4cf172e00e	...	0.535	-1.144339	-1.319283	0.000000	1.000000	0.00
584c2d52	1746600cb0	1	1	0.784228	4cf172e00e	...	0.748	-0.240922	0.297479	0.000000	0.918919	0.03
584c2d52	1746600cb0	1	1	0.697585	fc150fd13a	...	0.000	-1.600326	-1.705970	NaN	NaN	
a82606c6	1746600cb0	1	1	0.480977	5624b8f759	...	0.474	0.448061	0.019468	0.013825	0.382488	0.11
584c2d52	1746600cb0	1	1	0.524298	fc150fd13a	...	0.000	-1.352160	-1.638386	0.015625	0.445312	0.42

Таргет

In [5]:

```
labels
```

started 14:08:17 2020-03-19, finished in 7ms

Out[5]:

```
array([2, 4, 3, ..., 5, 2, 2])
```

Найдем номера категориальных признаков. Данная процедура, вообще говоря, не гарантирует правильный ответ, но никакой информации о признаках в соревновании не дается

In [6]:

```
text_features = []
real_features = []

for name in data.columns:
    value = data[name].iloc[0]

    if np.isreal(value):
        real_features.append(name)
    else:
        text_features.append(name)

print(text_features)
print(real_features)
```

started 14:08:17 2020-03-19, finished in 8ms

```
['x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x9', 'x10', 'x11', 'x12', 'x14',
'x15', 'x16', 'x17', 'x18', 'x19', 'x20', 'x21', 'x22']
['x6', 'x7', 'x8', 'x13', 'x23', 'x24', 'x25', 'x26', 'x27', 'x28', 'x
29', 'x30', 'x31', 'x32', 'x33', 'x34', 'x35', 'x36', 'x37', 'x38', 'x
39', 'x40', 'x41', 'x42', 'x43', 'x44', 'x45', 'x46', 'x47', 'x48', 'x
49', 'x50', 'x51', 'x52', 'x53', 'x54', 'x55', 'x56', 'x57', 'x58', 'x
59', 'x60', 'x61']
```

Выданные данные разобьем на обучающие и валидационные

In [7]:

```
x_train, x_valid, y_train, y_valid = train_test_split(data, labels,
                                                      test_size=0.2)
```

started 14:08:17 2020-03-19, finished in 28ms

2. Преобразования данных

Все категориальные признаки преобразуем к строковому типу. При такой процедуре все значения `np.nan` заменяются на строку `nan`, что эквивалентно введению еще одной категории.

In [8]:

```
x_train[text_features] = x_train[text_features].astype(str)
x_valid[text_features] = x_valid[text_features].astype(str)
x_test[text_features] = x_test[text_features].astype(str)
```

started 14:08:18 2020-03-19, finished in 196ms

В зависимости от требований модели к данным разные модели будем обучать на данных с различным препроцессингом.

Данные A --- оригинальные данные, в которых только заменены пропуски в категориальных признаках на строку `nan`. По сути в данных ничего не изменилось.

In [9]:

```
1  x_train_origin = x_train.copy()
2  x_valid_origin = x_valid.copy()
3  x_test_origin = x_test.copy()
```

started 14:08:19 2020-03-19, finished in 29ms

Модели из `sklearn` не умеют обрабатывать пропуски, а они в данных есть:

In [10]:

```
1  x_train.info()
```

started 14:08:19 2020-03-19, finished in 40ms

```
x44    38289 non-null float64
x45    38289 non-null float64
x46    38289 non-null float64
x47    38289 non-null float64
x48    38289 non-null float64
x49    38289 non-null float64
x50    38289 non-null float64
x51    38289 non-null float64
x52    38289 non-null float64
x53    38289 non-null float64
x54    38289 non-null float64
x55    36905 non-null float64
x56    36905 non-null float64
x57    36905 non-null float64
x58    36905 non-null float64
x59    36905 non-null float64
x60    36905 non-null float64
x61    36905 non-null float64
dtypes: float64(41), int64(2), object(19)
```

Для кодировки категориальных признаков будем использовать `MeanEncoder`, который заменяет значение категории на среднее значение таргета в этой категории. Если категория не встречалась в трейне, то на глобальное среднее таргета.

In [11]:

```python
# https://github.com/AndreyKoceruba/mean-encoding/blob/master/mean_encoder.py

from sklearn.base import BaseEstimator
from sklearn.base import TransformerMixin

class MeanEncoder(BaseEstimator, TransformerMixin):

    def __init__(self, target_type='binary',
                 encoding='likelihood', func=None):
        if target_type == 'continuous' and encoding in ['woe', 'diff']:
            raise ValueError(
                '{} target_type can\'t be used with {} encoding'.format(targe
            )
        self.target_type = target_type
        self.encoding = encoding
        self.func = func

    def goods(self, x):
        return np.sum(x == 1)

    def bads(self, x):
        return np.sum(x == 0)

    def encode(self, X, y, agg_func):
        self.means = dict()
        self.global_mean = np.nan
        X['target'] = y
        for col in X.columns:
            if col != 'target':
                col_means = X.groupby(col)['target'].agg(agg_func)
                self.means[col] = col_means
        X.drop(['target'], axis=1, inplace = True)

    def fit(self, X, y):
        if self.encoding == 'woe':
            self.encode(X, y, lambda x: np.log(self.goods(x) / self.bads(x)) 
            self.global_mean = np.log(self.goods(y) / self.bads(y)) * 100
        elif self.encoding == 'diff':
            self.encode(X, y, lambda x: self.goods(x) - self.bads(x))
            self.global_mean = self.goods(y) - self.bads(y)
        elif self.encoding == 'likelihood':
            self.encode(X, y, np.mean)
            self.global_mean = np.mean(y)
        elif self.encoding == 'count':
            self.encode(X, y, np.sum)
            self.global_mean = np.sum(y)
        elif self.encoding == 'function':
            self.encode(X, y, lambda x: self.func(x))
            self.global_mean = self.func(y)
        return self

    def transform(self, X):
        X_new = pd.DataFrame()
        for col in X.columns:
            X_new[col] = X[col].map(self.means[col]).fillna(self.global_mean)
        return X_new

    def fit_transform(self, X, y):
        self.fit(X, y)
```

```
60            return self.transform(X)
```

started 14:08:19 2020-03-19, finished in 12ms

Применим его к нашим данным

In [12]:

```
encoder = MeanEncoder()
x_train[text_features] = encoder.fit_transform(x_train[text_features],
                                               y_train)
x_valid[text_features] = encoder.transform(x_valid[text_features])
x_test[text_features] = encoder.transform(x_test[text_features])
```

started 14:08:20 2020-03-19, finished in 369ms

Данные B --- данные, полученные из **Данных A** с помощью `MeanEncoder`, то есть кодированием категориальных с помощью среднего отклика по категории.

In [13]:

```
x_train_encoder = x_train.copy()
x_valid_encoder = x_valid.copy()
x_test_encoder = x_test.copy()
```

started 14:08:21 2020-03-19, finished in 79ms

В вещественных признаках заполним пропуски средним по признаку.

Данные C --- данные, полученные из **Данных B** с помощью заполненяия пропусков в вещественных признаках средним значением

In [14]:

```
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

x_train[real_features] = imputer.fit_transform(
    x_train[real_features].astype(float)
)
x_valid[real_features] = imputer.transform(
    x_valid[real_features].astype(float)
)
x_test[real_features] = imputer.transform(
    x_test[real_features].astype(float)
)
```

started 14:08:21 2020-03-19, finished in 553ms

3. RandomForestClassifier

Обучаем случайный лес на 200 деревьев на **Данных C**

In [15]:

```
%%time

n_estimators = 200
rf = RandomForestClassifier(
    n_estimators=n_estimators, n_jobs=4
).fit(x_train, y_train)
```

started 14:08:22 2020-03-19, finished in 10.8s

```
CPU times: user 34.2 s, sys: 248 ms, total: 34.5 s
Wall time: 10.8 s
```

Качество на валидации

In [16]:

```
accuracy_score(rf.predict(x_valid), y_valid)
```

started 14:08:33 2020-03-19, finished in 516ms

Out[16]:

```
0.7206
```

Определим функцию, которая посчитает значение метрики в зависимости от количества деревьев

In [17]:

```
def accuracy_by_tree_count(model, test, target_test):
    '''
    Вычисляет точность модели в зависимости от кол-ва деревьев.
    model --- модель,
    test --- данные, на которых надо построить предсказания,
    target_test --- соответствующая целевая метка
    '''

    n_estimators = model.get_params()['n_estimators']
    accuracy_values = np.zeros(n_estimators)
    trees_labels = np.zeros((test.shape[0], n_estimators))

    for n in range(n_estimators):
        trees_labels[:, n] = model.estimators_[n].predict(test)

        accuracy_values[n] = accuracy_score(
            [Counter(labels).most_common(1)[0][0]
             for labels in trees_labels[:, :n + 1]],
            target_test)

    return accuracy_values
```

started 14:08:34 2020-03-19, finished in 4ms

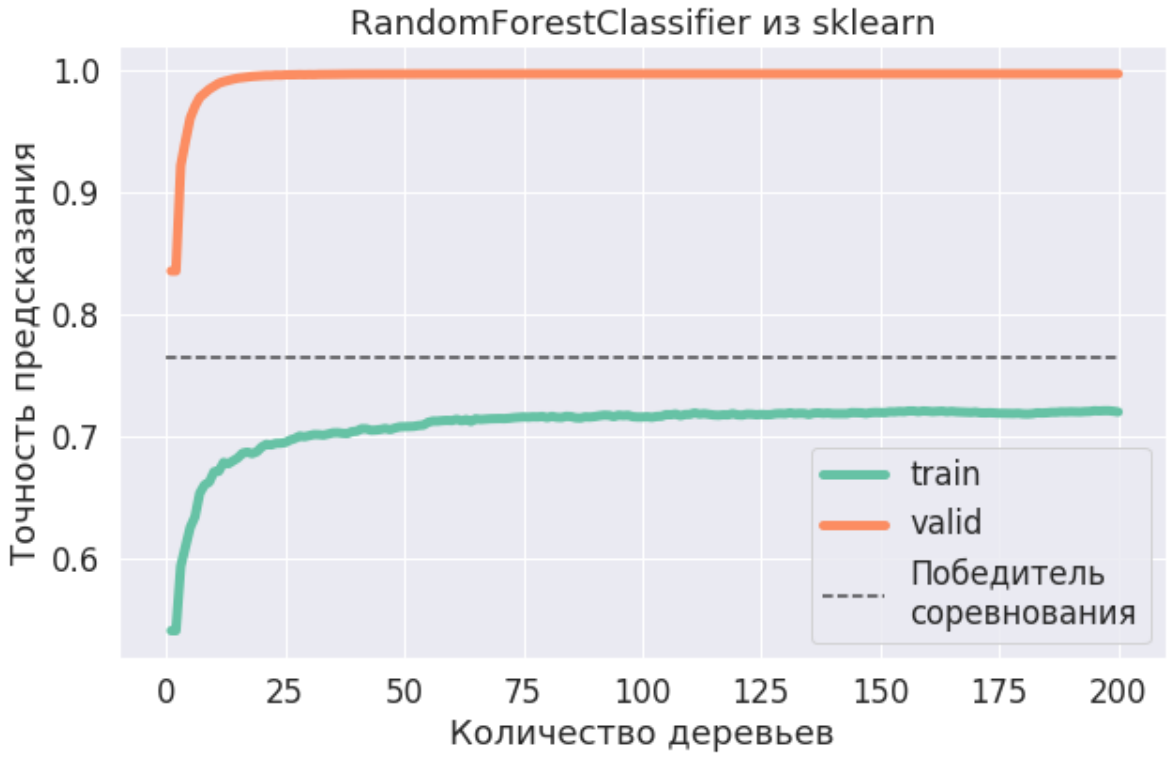
График зависимости точности предсказания от количества деревьев

In [18]:

```
accuracy_train = accuracy_by_tree_count(rf, x_valid, y_valid)
accuracy_valid = accuracy_by_tree_count(rf, x_train, y_train)

plt.figure(figsize=(10, 6))
plt.plot(np.arange(n_estimators) + 1, accuracy_train,
         lw=5, label='train')
plt.plot(np.arange(n_estimators) + 1, accuracy_valid,
         lw=5, label='valid')
plt.hlines(0.7639, 0, n_estimators, linestyles='--',
           alpha=0.7, label='Победитель\nсоревнования')
plt.xlabel('Количество деревьев')
plt.ylabel('Точность предсказания')
plt.title('RandomForestClassifier из sklearn')
plt.legend()
plt.show()
```

started 14:08:34 2020-03-19, finished in 3m 58s



Точность на тесте

In [19]:

```
accuracy_score(rf.predict(x_test), y_test)
```

started 14:12:32 2020-03-19, finished in 1.57s

Out[19]:

0.72266

Leaderboard моделей

Место	Имя модели	Тип данных	Качество на валидации	Качество на тесте
1	Победитель			0.7639
2	RandomForestClassifier	C	0.7206	0.72266

Далее рассмотрим еще несколько моделей, в которых проделаем аналогичные операции.

4. GradientBoostingClassifier

Подберем гиперпараметры бустинга на **Данных С**

In [20]:

```
model = RandomizedSearchCV(
    estimator=GradientBoostingClassifier(),
    param_distributions={
        'max_depth': np.arange(3, 6),
        'n_estimators': np.arange(10, 200),
        'learning_rate': np.linspace(0.05, 0.3, 300)
    },
    cv=5,  # разбиение выборки на 5 фолдов
    verbose=10,  # насколько часто печатать сообщения
    n_jobs=4,  # кол-во параллельных процессов
    n_iter=30  # кол-во итераций случайного выбора гиперпараметров
)

model.fit(x_train, y_train)
```

started 14:12:33 2020-03-19, finished in 1h 34m 46s

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent work
ers.
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:  6.7min
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:  7.4min
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed: 10.3min
[Parallel(n_jobs=4)]: Done  24 tasks      | elapsed: 13.7min
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed: 16.6min
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed: 26.3min
[Parallel(n_jobs=4)]: Done  53 tasks      | elapsed: 32.5min
[Parallel(n_jobs=4)]: Done  64 tasks      | elapsed: 40.0min
[Parallel(n_jobs=4)]: Done  77 tasks      | elapsed: 47.3min
[Parallel(n_jobs=4)]: Done  90 tasks      | elapsed: 54.5min
[Parallel(n_jobs=4)]: Done 105 tasks      | elapsed: 62.4min
[Parallel(n_jobs=4)]: Done 120 tasks      | elapsed: 73.3min
[Parallel(n_jobs=4)]: Done 137 tasks      | elapsed: 84.2min
[Parallel(n_jobs=4)]: Done 150 out of 150 | elapsed: 91.6min finished
```

Out[20]:

```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=GradientBoostingClassifier(criterion='fri
edman_mse',
                                                        init=None,
                                                        learning_rate=
0.1,
                                                        loss='devianc
e',
                                                        max_depth=3,
                                                        max_features=N
one,
                                                        max_leaf_nodes
=None,
                                                        min_impurity_d
ecrease=0.0,
                                                        min_impurity_s
plit=None,
                                                        min_samples_le
af=1,
                                                        min_samples_sp
lit=2,
                                                        min_weight_fra
ction_leaf=0.0,
```

```
                                                    n_estimators=1
00,
                                                    n_i...
       127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 13
9,
       140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 15
2,
       153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 16
5,
       166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 17
8,
       179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 19
1,
       192, 193, 194, 195, 196, 197, 198, 199])},
                  pre_dispatch='2*n_jobs', random_state=None, refit=T
rue,
                  return_train_score=False, scoring=None, verbose=10)
```

In [21]:

```
1  accuracy_score(model.predict(x_valid), y_valid)
```

started 15:47:20 2020-03-19, finished in 319ms

Out[21]:

```
0.74
```

Делаем предсказания модели в зависимости от количества деревьев и считаем метрику

In [22]:

```
1  predictions_train = model.best_estimator_.staged_predict(x_train)
2  predictions_valid = model.best_estimator_.staged_predict(x_valid)
3  accuracy_train = [accuracy_score(p, y_train) for p in predictions_train]
4  accuracy_valid = [accuracy_score(p, y_valid) for p in predictions_valid]
```

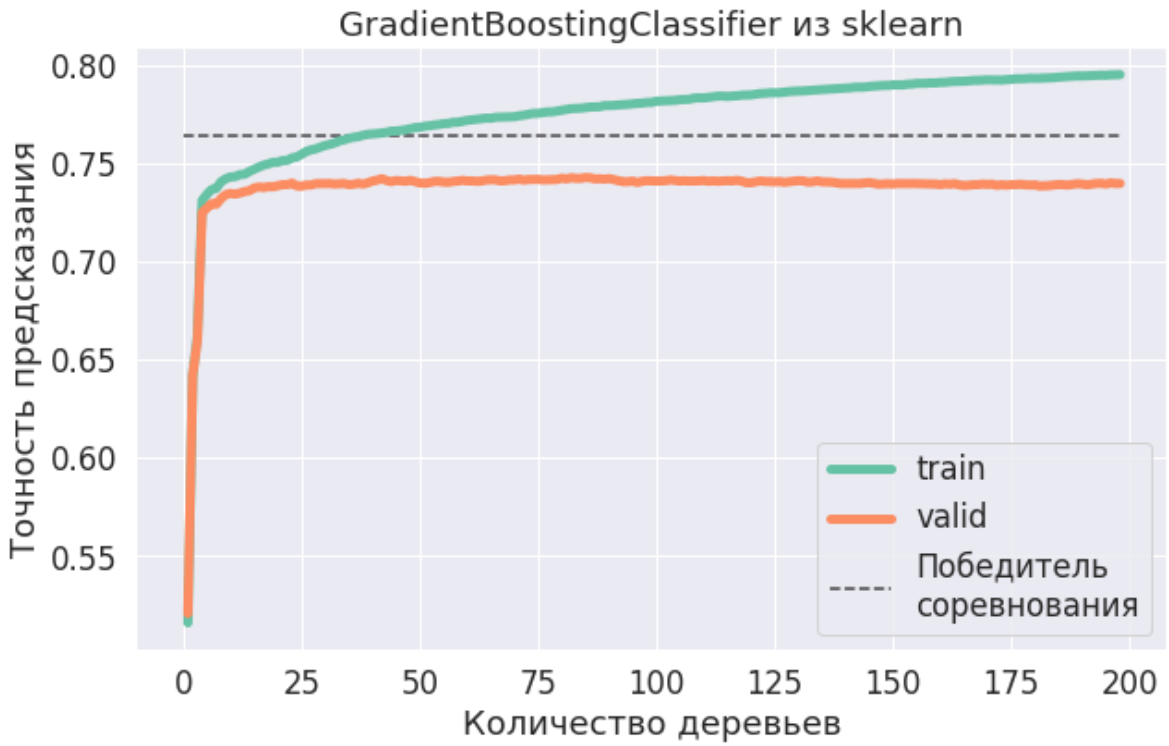
started 15:47:20 2020-03-19, finished in 5.50s

In [23]:

```
n_estimators = model.best_params_['n_estimators']

plt.figure(figsize=(10, 6))
plt.plot(np.arange(n_estimators) + 1, accuracy_train,
         lw=5, label='train')
plt.plot(np.arange(n_estimators) + 1, accuracy_valid,
         lw=5, label='valid')
plt.hlines(0.7639, 0, n_estimators, linestyles='--',
           alpha=0.7, label='Победитель\nсоревнования')
plt.xlabel('Количество деревьев')
plt.ylabel('Точность предсказания')
plt.title('GradientBoostingClassifier из sklearn')
plt.legend()
plt.show()
```

started 15:47:25 2020-03-19, finished in 361ms



In [24]:

```
accuracy_score(model.predict(x_test), y_test)
```

started 15:47:26 2020-03-19, finished in 1.88s

Out[24]:

```
0.74136
```

Leaderboard моделей

Место	Имя модели	Тип данных	Качество на валидации	Качество на тесте
1	Победитель			0.7639
2	GradientBoostingClassifier	C	0.74	0.74136
3	RandomForestClassifier	C	0.7206	0.72266

5. RandomForestClassifier + OneHotEncoder

Попробуем иначе кодировать категориальные признаки.

Данные D --- данные, полученные из **Данных A** с помощью заполненяия пропусков в вещественных признаках средним значением и с помощью OneHotEncoder на категориальных. Имеют вид разреженной матрицы.

In [25]:

```
oh_encoder = OneHotEncoder(handle_unknown='ignore')
cat_train = oh_encoder.fit_transform(x_train_origin[text_features])
cat_valid = oh_encoder.transform(x_valid_origin[text_features])
cat_test = oh_encoder.transform(x_test_origin[text_features])
```

started 15:47:28 2020-03-19, finished in 512ms

In [26]:

```
cat_train
```

started 15:47:28 2020-03-19, finished in 7ms

Out[26]:

```
<40000x9234 sparse matrix of type '<class 'numpy.float64'>'
        with 760000 stored elements in Compressed Sparse Row format>
```

Соединяем кодировку категориальных с вещественными признаким

In [27]:

```
x_train_sparse = sparse.hstack([
    cat_train, sparse.coo_matrix(x_train[real_features])
])
x_valid_sparse = sparse.hstack([
    cat_valid, sparse.coo_matrix(x_valid[real_features])
])
x_test_sparse = sparse.hstack([
    cat_test, sparse.coo_matrix(x_test[real_features])
])
```

started 15:47:28 2020-03-19, finished in 283ms

In [28]:

```
%%time

n_estimators = 200
rf = RandomForestClassifier(
    n_estimators=n_estimators, n_jobs=4
).fit(x_train_sparse, y_train)
```

started 15:47:28 2020-03-19, finished in 1m 6.38s

```
CPU times: user 3min 41s, sys: 772 ms, total: 3min 41s
Wall time: 1min 6s
```

In [29]:

```
accuracy_score(rf.predict(x_valid_sparse), y_valid)
```

started 15:48:35 2020-03-19, finished in 517ms

Out[29]:

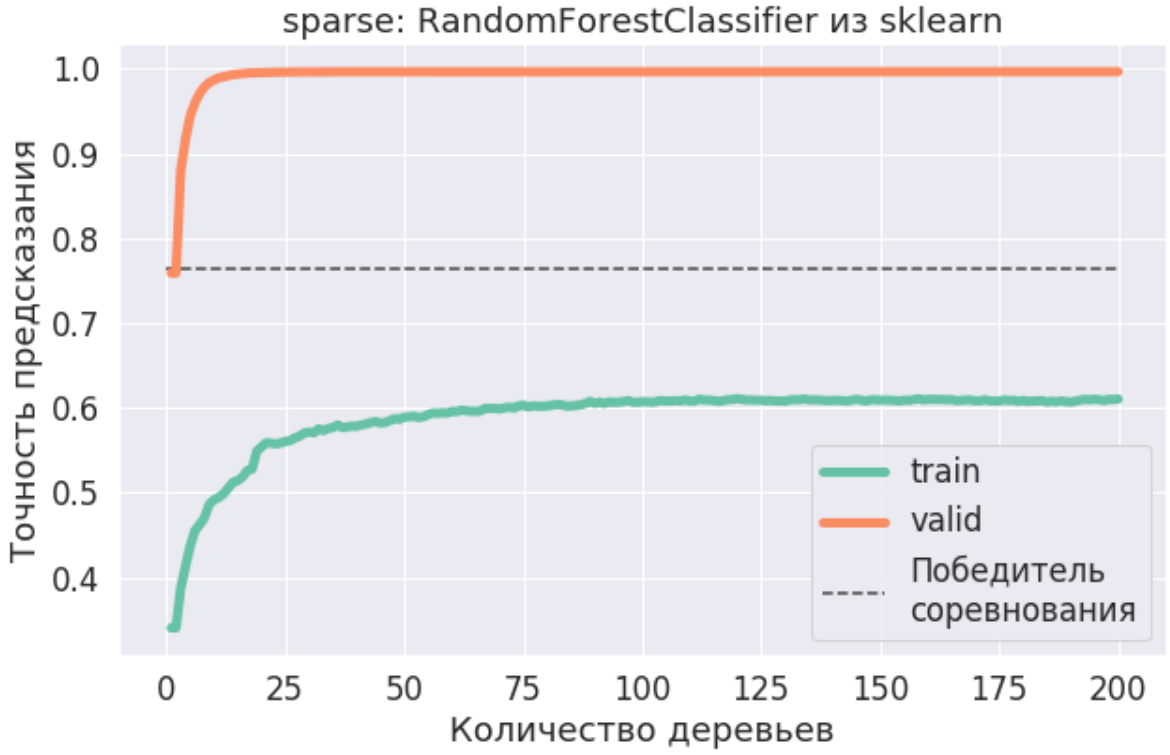
```
0.6095
```

In [30]:

```
accuracy_train = accuracy_by_tree_count(rf, x_valid_sparse, y_valid)
accuracy_valid = accuracy_by_tree_count(rf, x_train_sparse, y_train)

plt.figure(figsize=(10, 6))
plt.plot(np.arange(n_estimators) + 1, accuracy_train,
         lw=5, label='train')
plt.plot(np.arange(n_estimators) + 1, accuracy_valid,
         lw=5, label='valid')
plt.hlines(0.7639, 0, n_estimators, linestyles='--',
           alpha=0.7, label='Победитель\nсоревнования')
plt.xlabel('Количество деревьев')
plt.ylabel('Точность предсказания')
plt.title('sparse: RandomForestClassifier из sklearn')
plt.legend()
plt.show()
```

started 15:48:35 2020-03-19, finished in 4m 22s



In [31]:

```
accuracy_score(rf.predict(x_test_sparse), y_test)
```

started 15:52:57 2020-03-19, finished in 2.17s

Out[31]:

```
0.61292
```

Leaderboard моделей

Место	Имя модели	Тип данных	Качество на валидации	Качество на тесте
1	Победитель			0.7639
2	GradientBoostingClassifier	C	0.74	0.74136
3	RandomForestClassifier	C	0.7206	0.72266
4	RandomForestClassifier	D	0.6095	0.61292

Ну тут совсем налажали...

6. XGBoost

Выкатываем тяжелую артиллерию. Она умеет работать с пропусками, поэтому берем **Данные B**

In [32]:

```
model = RandomizedSearchCV(
    estimator=XGBClassifier(),
    param_distributions={
        'max_depth': np.arange(3, 8),
        'n_estimators': np.arange(10, 200),
        'learning_rate': np.linspace(0.05, 0.3, 300)
    },
    cv=5,  # разбиение выборки на 5 фолдов
    verbose=10,  # насколько часто печатать сообщения
    n_jobs=4,  # кол-во параллельных процессов
    n_iter=30  # кол-во итераций случайного выбора гиперпараметров
)

model.fit(x_train_encoder, y_train)
```

started 16:18:00 2020-03-19, finished in 2h 1m 6s

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent work
ers.
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:  9.9min
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed: 15.1min
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed: 20.1min
[Parallel(n_jobs=4)]: Done  24 tasks      | elapsed: 29.4min
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed: 34.3min
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed: 43.5min
[Parallel(n_jobs=4)]: Done  53 tasks      | elapsed: 54.3min
[Parallel(n_jobs=4)]: Done  64 tasks      | elapsed: 63.2min
[Parallel(n_jobs=4)]: Done  77 tasks      | elapsed: 68.4min
[Parallel(n_jobs=4)]: Done  90 tasks      | elapsed: 73.7min
[Parallel(n_jobs=4)]: Done 105 tasks      | elapsed: 79.5min
[Parallel(n_jobs=4)]: Done 120 tasks      | elapsed: 92.5min
[Parallel(n_jobs=4)]: Done 137 tasks      | elapsed: 108.0min
[Parallel(n_jobs=4)]: Done 150 out of 150 | elapsed: 118.3min finished
```

Out[32]:

```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gb
tree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=
0,
                                           learning_rate=0.1, max_delt
a_step=0,
                                           max_depth=3, min_child_weig
ht=1,
                                           missing=None, n_estimators=
100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logisti
c',
                                           random_state=0, reg_alpha=
0...
       127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 13
9,
       140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 15
2,
       153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 16
```

```
5,
       166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 17
8,
       179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 19
1,
       192, 193, 194, 195, 196, 197, 198, 199])},
                  pre_dispatch='2*n_jobs', random_state=None, refit=T
rue,
                  return_train_score=False, scoring=None, verbose=10)
```

In [33]:

```
1  accuracy_score(model.predict(x_valid_encoder), y_valid)
```

started 18:19:06 2020-03-19, finished in 865ms

Out[33]:

```
0.7397
```

Делаем предсказания модели в зависимости от количества деревьев и считаем метрику

In [34]:

```
1  n_estimators = model.best_params_['n_estimators']
2
3  accuracy_train = [
4      accuracy_score(
5          y_train, model.best_estimator_.predict(x_train_encoder,
6                                                 ntree_limit=i+1)
7      )
8      for i in range(n_estimators)
9  ]
10
11 accuracy_valid = [
12     accuracy_score(
13         y_valid, model.best_estimator_.predict(x_valid_encoder,
14                                                ntree_limit=i+1)
15     )
16     for i in range(n_estimators)
17 ]
```

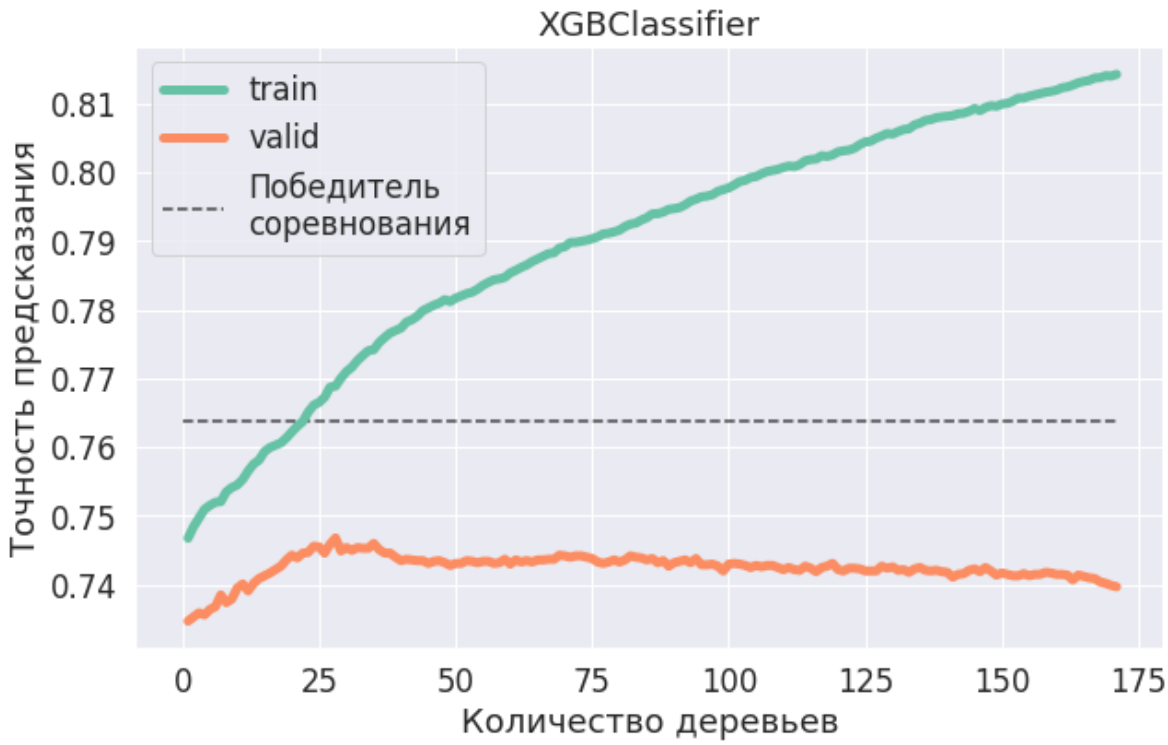
started 18:19:07 2020-03-19, finished in 5m 6s

In [35]:

```
n_estimators = model.best_params_['n_estimators']

plt.figure(figsize=(10, 6))
plt.plot(np.arange(n_estimators) + 1, accuracy_train,
         lw=5, label='train')
plt.plot(np.arange(n_estimators) + 1, accuracy_valid,
         lw=5, label='valid')
plt.hlines(0.7639, 0, n_estimators, linestyles='--',
           alpha=0.7, label='Победитель\nсоревнования')
plt.xlabel('Количество деревьев')
plt.ylabel('Точность предсказания')
plt.title('XGBClassifier')
plt.legend()
plt.show()
```

started 18:24:12 2020-03-19, finished in 356ms



In [36]:

```
accuracy_score(model.predict(x_test_encoder), y_test)
```

started 18:24:13 2020-03-19, finished in 4.03s

Out[36]:

0.74262

Leaderboard моделей

Место	Имя модели	Тип данных	Качество на валидации	Качество на тесте
1	Победитель			0.7639
2	XGBClassifier	B	0.7397	0.74262
3	GradientBoostingClassifier	C	0.74	0.74136
4	RandomForestClassifier	C	0.7206	0.72266

Место	Имя модели	Тип данных	Качество на валидации	Качество на тесте
5	RandomForestClassifier	D	0.6095	0.61292

Не сильно то лучше стало. На валидации наоборот чуть проиграли.

7. LGBMClassifier

Другая ~~тяжелая~~ легкая артиллерия с **Данными B**

In [42]:

```
model = RandomizedSearchCV(
    estimator=LGBMClassifier(),
    param_distributions={
        'max_depth': np.arange(3, 6),
        'n_estimators': np.arange(10, 100),
        'learning_rate': np.linspace(0.05, 0.3, 300)
    },
    cv=5,  # разбиение выборки на 5 фолдов
    verbose=10,  # насколько часто печатать сообщения
    n_jobs=4,  # кол-во параллельных процессов
    n_iter=30  # кол-во итераций случайного выбора гиперпараметров
)

model.fit(x_train_encoder, y_train)
```

started 18:26:34 2020-03-19, finished in 4m 3s

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent work
ers.
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:   11.5s
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:   15.6s
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed:   28.2s
[Parallel(n_jobs=4)]: Done  24 tasks      | elapsed:   36.3s
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:   54.3s
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:  1.4min
[Parallel(n_jobs=4)]: Done  53 tasks      | elapsed:  1.7min
[Parallel(n_jobs=4)]: Done  64 tasks      | elapsed:  2.2min
[Parallel(n_jobs=4)]: Done  77 tasks      | elapsed:  2.4min
[Parallel(n_jobs=4)]: Done  90 tasks      | elapsed:  2.8min
[Parallel(n_jobs=4)]: Done 105 tasks      | elapsed:  3.1min
[Parallel(n_jobs=4)]: Done 120 tasks      | elapsed:  3.2min
[Parallel(n_jobs=4)]: Done 137 tasks      | elapsed:  3.7min
[Parallel(n_jobs=4)]: Done 150 out of 150 | elapsed:  3.9min finished
```

Out[42]:

```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=LGBMClassifier(boosting_type='gbdt',
                                            class_weight=None,
                                            colsample_bytree=1.0,
                                            importance_type='split',
                                            learning_rate=0.1, max_dep
th=-1,
                                            min_child_samples=20,
                                            min_child_weight=0.001,
                                            min_split_gain=0.0,
                                            n_estimators=100, n_jobs=-
1,
                                            num_leaves=31, objective=N
one,
                                            random_state=None, reg_alp
ha=0.0,
                                            reg...
                                         'n_estimators': array([10, 11,
12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
       27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
43,
       44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
60,
```

```
       61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
77,
       78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93,
94,
       95, 96, 97, 98, 99])},
                    pre_dispatch='2*n_jobs', random_state=None, refit=T
rue,
                    return_train_score=False, scoring=None, verbose=10)
```

In [43]:

```
accuracy_score(model.predict(x_valid_encoder), y_valid)
```

started 18:30:37 2020-03-19, finished in 354ms

Out[43]:

```
0.7402
```

In [44]:

```
n_estimators = model.best_params_['n_estimators']

accuracy_train = [
    accuracy_score(
        y_train, model.best_estimator_.predict(x_train_encoder,
                                               num_iteration=i+1)
    )
    for i in range(n_estimators)
]

accuracy_valid = [
    accuracy_score(
        y_valid, model.best_estimator_.predict(x_valid_encoder,
                                               num_iteration=i+1)
    )
    for i in range(n_estimators)
]
```

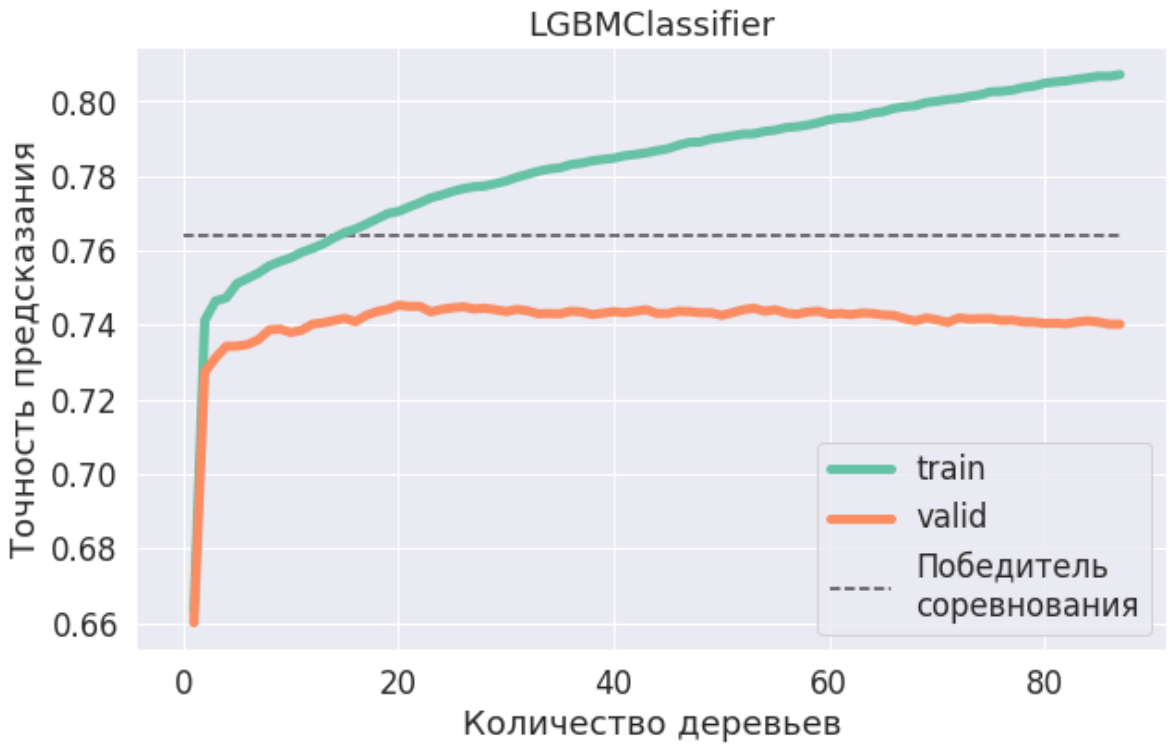
started 18:30:37 2020-03-19, finished in 1m 14.6s

In [45]:

```
n_estimators = model.best_params_['n_estimators']

plt.figure(figsize=(10, 6))
plt.plot(np.arange(n_estimators) + 1, accuracy_train,
         lw=5, label='train')
plt.plot(np.arange(n_estimators) + 1, accuracy_valid,
         lw=5, label='valid')
plt.hlines(0.7639, 0, n_estimators, linestyles='--',
           alpha=0.7, label='Победитель\nсоревнования')
plt.xlabel('Количество деревьев')
plt.ylabel('Точность предсказания')
plt.title('LGBMClassifier')
plt.legend()
plt.show()
```

started 18:31:52 2020-03-19, finished in 322ms



In [46]:

```
accuracy_score(model.predict(x_test_encoder), y_test)
```

started 18:31:52 2020-03-19, finished in 1.53s

Out[46]:

```
0.74344
```

Leaderboard моделей

Место	Имя модели	Тип данных	Качество на валидации	Качество на тесте
1	Победитель			0.7639
2	LGBMClassifier	B	0.7402	0.74344
3	XGBClassifier	B	0.7397	0.74262
4	GradientBoostingClassifier	C	0.74	0.74136

Место	Имя модели	Тип данных	Качество на валидации	Качество на тесте
5	RandomForestClassifier	C	0.7206	0.72266
6	RandomForestClassifier	D	0.6095	0.61292

Немного стало лучше. Плюс в том, что обучение прошло очень быстро. Так и не подумаешь, что эта модель -- разработка Microsoft...

8. CatBoostClassifier

А что наши? Приятно то, что катбуст сам обрабатывает категориальные признаки, нужно только подать ему их индексы. Поэтому используем **Данные А**, которые почти не отличаются от исходных.

In [47]:

```
# задаем пространство поиска
param_distributions = {
    'n_estimators' : np.arange(1, 200),
    'max_depth' : list(range(3, 8)),
    'learning_rate' : np.linspace(0.01, 0.3, 10000),
    'min_data_in_leaf' : np.arange(1, 7),
    'l2_leaf_reg' : np.linspace(0, 10, 101),
    'rsm' : np.linspace(0.5, 1, 100)
}


# определяем поиск по сетке
model = RandomizedSearchCV(
    estimator=CatBoostClassifier(cat_features=text_features,
                                 verbose=10000),
    param_distributions=param_distributions,
    scoring='accuracy',
    n_iter=30,
    n_jobs=4,
    cv=5,
    verbose=10,
    error_score='raise'
)
```

started 18:34:03 2020-03-19, finished in 9ms

In [48]:

```
%%time

# выполняем поиск по сетке
model.fit(x_train_origin, y_train)
```

started 18:34:05 2020-03-19, finished in 48m 12s

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent work
ers.
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:   15.9s
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:  2.9min
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed:  3.9min
[Parallel(n_jobs=4)]: Done  24 tasks      | elapsed:  5.8min
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:  9.8min
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed: 12.7min
[Parallel(n_jobs=4)]: Done  53 tasks      | elapsed: 17.8min
[Parallel(n_jobs=4)]: Done  64 tasks      | elapsed: 21.2min
[Parallel(n_jobs=4)]: Done  77 tasks      | elapsed: 25.1min
[Parallel(n_jobs=4)]: Done  90 tasks      | elapsed: 27.4min
[Parallel(n_jobs=4)]: Done 105 tasks      | elapsed: 31.6min
[Parallel(n_jobs=4)]: Done 120 tasks      | elapsed: 36.8min
[Parallel(n_jobs=4)]: Done 137 tasks      | elapsed: 41.9min
[Parallel(n_jobs=4)]: Done 150 out of 150 | elapsed: 45.9min finished

0:      learn: 1.5623914        total: 1.1s     remaining: 2m 54s
159:    learn: 0.6592887        total: 2m 14s   remaining: 0us
CPU times: user 6min 22s, sys: 7.18 s, total: 6min 29s
Wall time: 48min 12s
```

Out[48]:

```
RandomizedSearchCV(cv=5, error_score='raise',
                   estimator=<catboost.core.CatBoostClassifier object
at 0x7f45ae450898>,
                   iid='warn', n_iter=30, n_jobs=4,
                   param_distributions={'l2_leaf_reg': array([ 0. ,
0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,
        1.1,  1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9,  2. ,  2.
1,
        2.2,  2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9,  3. ,  3.1,  3.
2,
        3.3,  3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,...
       0.87878788, 0.88383838, 0.88888889, 0.89393939, 0.8989899 ,
       0.9040404 , 0.90909091, 0.91414141, 0.91919192, 0.92424242,
       0.92929293, 0.93434343, 0.93939394, 0.94444444, 0.94949495,
       0.95454545, 0.95959596, 0.96464646, 0.96969697, 0.97474747,
       0.97979798, 0.98484848, 0.98989899, 0.99494949, 1.        ])},
                   pre_dispatch='2*n_jobs', random_state=None, refit=T
rue,
                   return_train_score=False, scoring='accuracy', verbo
se=10)
```

In [49]:

```
accuracy_score(model.predict(x_valid_origin), y_valid)
```

started 19:22:17 2020-03-19, finished in 395ms

Out[49]:

0.7583

In [50]:

```
n_estimators = model.best_params_['n_estimators']

accuracy_train = [
    accuracy_score(
        y_train, model.best_estimator_.predict(x_train_origin,
                                               ntree_end=i+1)
    )
    for i in range(n_estimators)
]

accuracy_valid = [
    accuracy_score(
        y_valid, model.best_estimator_.predict(x_valid_origin,
                                               ntree_end=i+1)
    )
    for i in range(n_estimators)
]
```

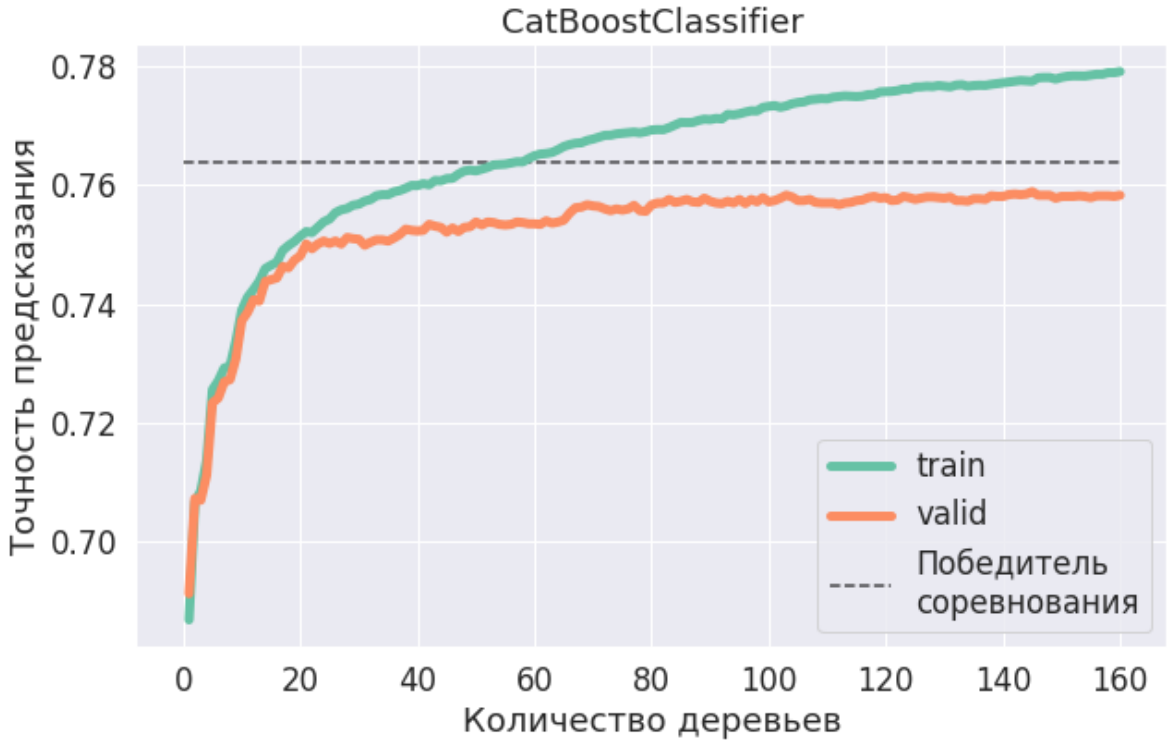
started 19:22:18 2020-03-19, finished in 44.7s

In [51]:

```
n_estimators = model.best_params_['n_estimators']

plt.figure(figsize=(10, 6))
plt.plot(np.arange(n_estimators) + 1, accuracy_train, lw=5, label='train')
plt.plot(np.arange(n_estimators) + 1, accuracy_valid, lw=5, label='valid')
plt.hlines(0.7639, 0, n_estimators, linestyles='--', alpha=0.7, label='Победи
plt.xlabel('Количество деревьев')
plt.ylabel('Точность предсказания')
plt.title('CatBoostClassifier')
plt.legend()
plt.show()
```

started 19:23:02 2020-03-19, finished in 447ms



In [52]:

```
accuracy_score(model.predict(x_test_origin), y_test)
```

started 19:23:03 2020-03-19, finished in 806ms

Out[52]:

```
0.75924
```

Leaderboard моделей

Место	Имя модели	Тип данных	Качество на валидации	Качество на тесте
1	Победитель			0.7639
2	CatBoostClassifier	A	0.7583	0.75924
3	LGBMClassifier	B	0.7402	0.74344
4	XGBClassifier	B	0.7397	0.74262
5	GradientBoostingClassifier	C	0.74	0.74136
6	RandomForestClassifier	C	0.7206	0.72266

Место	Имя модели	Тип данных	Качество на валидации	Качество на тесте
7	RandomForestClassifier	D	0.6095	0.61292

Воу! А Яндекс могет!

Теперь от победителя нас отделяют менее 0.5% качества.