Московский Государственный Университет им. Ломоносова
Факультет Вычислительной математики и кибернетики
Кафедра Математической статистики

Градиентный бустинг

Gradient Boosting

Работу выполнил
студент 316 группы
Харинаев Артём

Руководитель
д.ф.-м.н., доцент
Горшенин Андрей Константинович

г. Москва

Содержание

Введение

Градиентный бустинг - один из наиболее популярных и применяемых на практике алгоритмов машинного обучения. Поисковый запрос в Яндексе, выбор отеля на Booking.com или сериала в Netflix задействует несколько десятков моделей градиентного бустинга над решающими деревьями.

Алгоритм строит модель предсказания в форме ансамбля слабых предсказывающих моделей, обычно деревьев решений. Хорошо отрабатывает на неоднородных данных (категориальные, количественные, бинарные), помещающихся в оперативную память компьютера.

Рассмотрим, как устроен метод, в чем его преимущества и недостатки, попробуем на практике и сравним с другими алгоритмами машинного обучения.

Алгоритм

Интуитивные соображения

Рассмотрим задачу регрессии $Y = \mathbb{R}$. Базовый метод - решающее дерево.

Обучим 1 алгоритм, его качество, скорее всего, будет низким. Однако, теперь в качестве целевого признака можно взять разность исходных, верных ответов обучающей выборки и ответов 1-го алгоритма.

Обучим 2-ой алгоритм приближать полученный целевой признак, тогда сумма 1-го и 2-го алгоритмов будет приближать верные ответы точнее, чем один только 1-ый.

На следующем этапе в качестве целевого признака возьмем разность целевого признака на 2-ом этапе (это разность исходного целевого признака и ответов 1-го алгоритма) и ответов 2-го алгоритма. Обучим 3-ий и т.д., пока

не достигнем каких-либо ограничений по вычислению (определенной точности, количества итераций и т.п.)

В итоге, получается композиция алгоритмов, оптимально приближающих исходный целевой признак.

Заметим, что квадратичной функции потерь:

$$\mathcal{L}(y, a(x)) = \frac{1}{2}\|y - a(x)\|^2 \to \min$$

Соответствует градиент:

$$\nabla\mathcal{L}(y, a(x)) = y - a(x)$$

Что как раз является той величиной, которую стараются приблизить алгоритмы на каждой итерации. Дадим более строгое объяснение.

Математическое обоснование

Пусть даны X - пространство объектов (их признаковых описаний), Y - пространство ответов, $X^l = \{x_i, y_i\}_{i=1}^l$ - обучающая выборка

Пусть $\mathcal{L}$ - дифференцируемая функция потерь - либо сама функция потерь для задачи регрессии, либо аппроксимация пороговой функции потерь для классификации (задание функции потерь таким образом позволит описанный ниже алгоритм применять универсально). $\mathfrak{B}$ - пространство базовых алгоритмов. Алгоритм $a(x)$ представляет собой композицию T базовых алгоритмов:

$$a_T(x) = \sum_{t=1}^{T} b_t(x), \;\; x \in X, \;\; b_t : X \mapsto Y, \;\; b_t \in \mathfrak{B}$$

Композиция строится “жадно”: $a_T(x) = a_{T-1}(x) + b_T(x)$, где алгоритм $b_T(x)$ обучается так, чтобы минимизировать ошибку текущей композиции:

$$b_T(x) = \underset{b \in \mathfrak{B}}{\operatorname{argmin}} \sum_{i=1}^{l} \mathcal{L}(y_i, a_{T-1}(x_i) + b(x_i))$$

Начальный алгоритм обучается так, чтобы минимизировать потери на обучающей выборке:

$$b_1(x) = \underset{b \in \mathfrak{B}}{\operatorname{argmin}} \sum_{i=1}^{l} \mathcal{L}(y_i, b(x_i))$$

Рассмотрим разложение Тейлора функции потерь $\mathcal{L}$ до первого члена в окрестности $(y_i, a_{T-1}(x_i))$:

$$\mathcal{L}(y_i, a_{T-1}(x_i) + b(x_i)) \approx \mathcal{L}(y_i, a_{T-1}(x_i)) + b(x_i) \frac{\partial \mathcal{L}(y_i, z)}{\partial z}\bigg|_{z=a_{T-1}(x_i)} =$$
$$= \mathcal{L}(y_i, a_{T-1}(x_i)) + b(x_i) g_i^{T-1}$$

Если не учитывать постоянные члены, то получается следующая оптимизационная задача:

$$b_T \approx \underset{b \in \mathcal{B}}{\operatorname{argmin}} \sum_{i=1}^{l} b(x_i) g_i^{T-1}$$

Сумма компонент эквивалентна скалярному произведению векторов $b(x)$ и g^{T-1}. Тогда минимум достигается при $b(x)$ максимально близком к антиградиенту $-g^{T-1}$. То есть, на каждой итерации базовые алгоритмы $b_T(x)$ обучаются предсказывать значения антиградиента функции потерь по текущим предсказаниям композиции.

Задача

Постановка

В данной работе я ставлю перед собой задачу сравнить качество алгоритма градиентного бустинга с другими популярными алгоритмами машинного обучения - линейными моделями на данных разного типа (для задачи регрессии и классификации) и исследовать зависимости в данных “Ценообразование недвижимости”

Описание данных

Для демонстрации решения задачи классификации взят датасет “Болезни сердца” [1]. Он содержит 303 записей о пациентах, наблюдающихся у

кардиолога, и 14 колонок: 13 описательных (пол, возраст, различные анализы крови и т.п.) и 1 целевую - наличие болезни сердца.

Для демонстрации решения задачи регрессии взят датасет “Ценообразование недвижимости” [2]. Он содержит записи о 1460 объектах недвижимости города Эймс в штате Айова (Ames, Iowa), 80 колонок: 79 описательных (от площади ванной до типа дорожки, ведущей к крыльцу) и 1 целевая - стоимость продажи объекта. Описательные данные разнообразны: есть бинарные, порядковые, количественные и категориальные признаки.

Предобработка данных

Для обучения линейный моделей необходимо проделать достаточно большую работу по предобработке данных:

- Обработать недостающие значения (заполнить или удалить строки, содержащие их)
- Преобразовать текстовые данные в численные
    - Закодировать (one-hot-encoding, target mean, label encoder)
- Для лучшей сходимости нужно масштабировать данные
    - Преобразование к стандартному нормальному распределению
    - Min-Max преобразование
- Выявить и удалить выбросы

Для многих реализаций градиентного бустинга, таких как XGBoost, LightGBM, CatBoost, не обязательно выполнять все эти шаги. Например, LightGBM и CatBoost могут взаимодействовать с категориальными переменными без предварительного их преобразования в числа, необходимо лишь указать индексы категориальных признаков. Почти все алгоритмы градиентного бустинга (основным алгоритмом которых являются решающие деревья) могут работать с пропущенными значениями в данных.

Обучение алгоритмов

В качестве основной модели градиентного бустинга я взял реализацию CatBoost от компании Яндекс [3]. В задаче классификации я сравниваю его с логистической регрессией, а в задаче регрессии - с Ridge-регрессией.

Классификация

Перед обучением логистической регрессии проведем предобработку данных: закодируем небинарные категориальные методом One-hot-Encoding (добавление признаков, количество которых равно количеству уникальных значений исходного, каждый такой признак - бинарный и служит индикатором определенного уникального значения исходного признака) и масштабируем числовые данные (вычитание из каждого значения среднего по признаку и деление на стандартное отклонение по признаку отдельно для каждого числового признака).

Разобьем данные на обучающую и тестовую выборки. Найдем оптимальный коэффициент регуляризации с помощью кросс-валидации, используя F-метрику (чтобы модель “старалась” предсказать верно не только больных, но и здоровых). Затем обучим модель с найденным оптимальным коэффициентом, вычислим ответы для тестовой выборки и рассчитаем метрику качества классификации Recall на тестовой выборке. Результат - 0.813953488372093. Матрица ошибок:

23	10
8	35

Теперь применим градиентный бустинг к этим данным. Предобработка почти не нужна, необходимо лишь указать, какие признаки являются категориальными.

Разобьем данные на обучающую и тестовую выборки. Обучим модель, вычислим ответы для тестовой выборки и рассчитаем метрику качества Recall. Результат - 0.8636363636363636. Матрица ошибок:

29	3
6	38

Качество классификации градиентного бустинга выше на 5% без какого-либо подбора оптимальных параметров. Важно отметить, что при градиентном бустинге больных, ошибочно опознанных, как здоровые, меньше, что важно для данной сферы.

Регрессия

Для Ridge-регрессии осуществим предобработку данных: закодируем категориальные признаки, заполним пропуски медианой, масштабируем числовые данные

Разобьем данные на обучающую и тестовую выборки, метрика качества - RMSLE, среднеквадратичная логарифмическая ошибка - чтобы ошибки на недвижимости с малой ценой были сравнимы с ошибками на недвижимости с высокой ценой. Для удобства вычисления метрики логарифмируем целевой признак, будем предсказывать логарифм цены и считать RMSE.

Оптимальный коэффициент l_2 регуляризации подберем по кросс-валидации. Обучим модель с оптимальным параметром, рассчитаем ответы на тестовой выборке, вычислим ошибку: 0.13748253362426197

Попробуем проанализировать важность признаков для целевой переменной, можем рассмотреть абсолютные значения весов признаков, с которыми они входят в линейную модель.

Заметно, что из-за бинаризации категориальных признаков, их значения учитываются отдельно. С одной стороны, это помогает понять, какие значения признака оказываются “дороже”, чем другие (районы Crawford (Neighborhood_Crawfor) и Stone Brook (Neighborhood_StoneBr) престижнее других, т.к. их веса больше по модулю). С другой стороны, загромождает общую картину важности признаков. Здесь же видны некоторые аномалии: оценка 3 условий проживания (OverallCond) влияет на цену больше, чем оценка 9, а оценка 3 качества ремонта и материалов (OverallQual) больше, чем оценка 8. В этом недостаток бинаризации категориальных признаков.

Теперь обучим алгоритм градиентного бустинга на этих данных.

Разобьем данные на обучающую и валидационную выборку (для контроля переобучения). Тестовая выборка дана отдельно, метрика качества - RMSLE (среднеквадратичная логарифмическая ошибка), результат рассчитывается на площадке Kaggle после загрузки предсказанных ответов на тестовых данных.

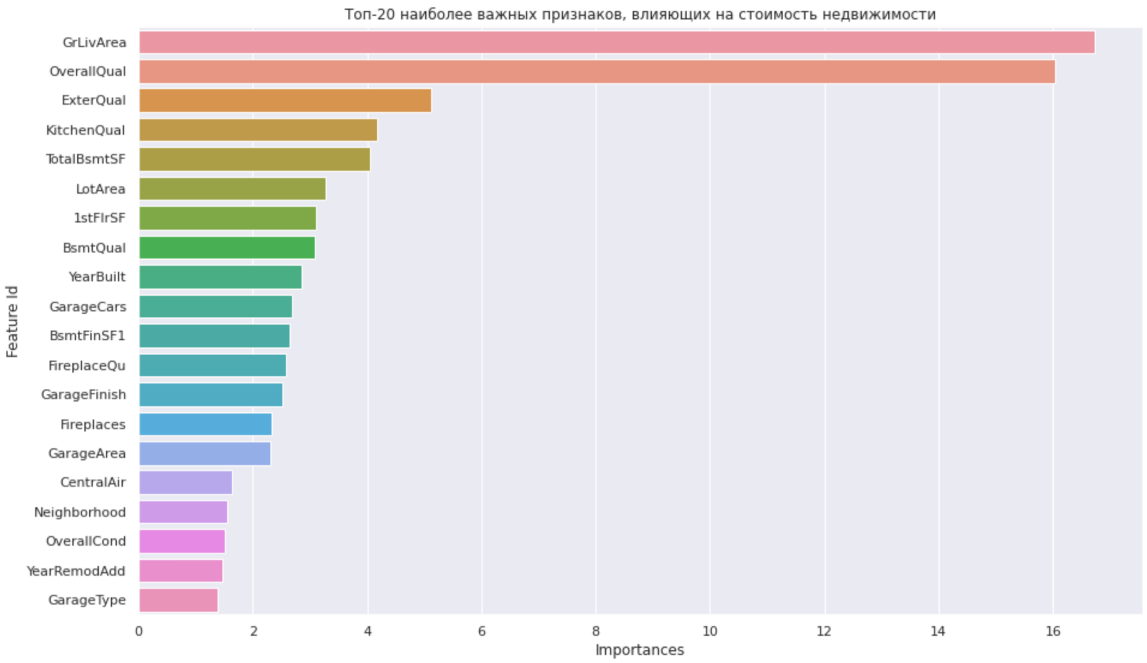
Для начала, обучим модель без явного указания параметров, “из коробки”. Оценим качество регрессии: RMSLE = 0.12948. Это будет опорный результат (baseline), который мы будет стараться улучшить с помощью подбора оптимальных параметров. Даже опорный результат уже лучше, чем результат регрессии на 6%.

Подберем оптимальные параметры:

- число итераций (iterations) - число деревьев (число T из математического обоснования алгоритма)
- максимальную глубину деревьев (depth)
- коэффициент l_2 -регуляризации (l2_leaf_reg)
- темп обучения или градиентный шаг (learning_rate)

Используем случайный поиск по сетке параметров (перебираются не все возможные комбинации параметров, а только случайная подвыборка), чтобы сократить время расчетов. Обучим модель с оптимизированными параметрами, используя валидационную выборку для контроля переобучения. Оценим качество регрессии: RMSLE = 0.12420. Подбор гиперпараметров позволил улучшить качество на 5%.
Важным преимуществом решающих деревьев, а значит ансамблей над решающими деревьями, включая градиентный бустинг, является их интерпретируемость. Признаки, используемые в верхней части дерева, влияют на окончательное предсказание для большей доли обучающих объектов, чем признаки, попавшие на более глубокие уровни. Таким образом, ожидаемая доля обучающих объектов, для которых происходило ветвление по данному признаку, может быть использована в качестве оценки его относительной важности для итогового предсказания. Усредняя полученные оценки важности признаков по всем решающим деревьям из ансамбля, можно уменьшить дисперсию такой оценки и использовать ее для отбора признаков. Этот метод известен как MDI (mean decrease in impurity).

Рассмотрим важность признаков для значения целевого признака.

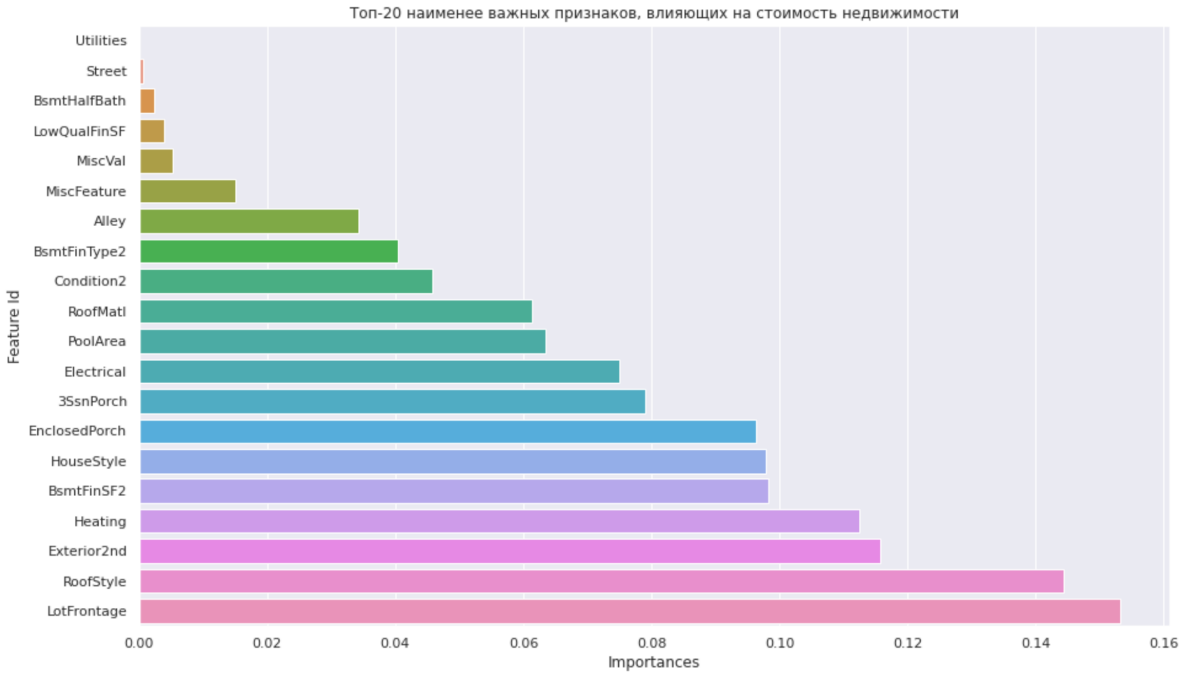


Наибольшую с отрывом важность имеют жилая площадь недвижимости (GrLivArea) и оценка качества ремонта и материалов дома (OverallQual), что довольно логично. Также, высокую важность для цены имеют качество внешней отделки дома, кухня, камины (количество и качество), гараж (вместимость, количество, качество), участок, район.

Из интересного: высокую важность имеет площадь подвала, возможно, это объясняется тем, что она зачастую коррелирует с площадью всего дома

Качество стройматериалов (дорогие, дешевые) (OverallQual) имеет больший вес, чем их состояние, износ (OverallCond), что тоже логично, т.к зачастую не только в строительстве и недвижимости качественные, хоть и не в идеальном состоянии, материалы ценятся больше, чем дешевые в идеальном состоянии.

Точно так же можно проанализировать наименее важные признаки.



Алгоритм почти обнулил признаки Utilities (доступные коммунальные услуги), Street (тип дороги, ведущей к участку) и Alley (тип аллеи, ведущей от дороги к дому) т.к. они оказались почти полностью константными.

Низким вкладом в цену также обладают:

- Стиль дома и крыши
- Площадь бассейна (большим вкладом обладает его наличие, а площадь уже вторична)
- Площадь крыльца
- Тип отопления и тип электрической проводки

Вывод

Градиентный бустинг обладает рядом преимуществ перед линейными моделями:

- Более высокая точность
- Гибкая настройка гиперпараметров
- Более адекватная оценка важности признаков
- Меньшая зависимость от предобработки данных

Однако, есть и минусы, главный из которых - большее время обучения модели. Поэтому простые алгоритмы, типа линейный моделей отлично служат “опорными” алгоритмами (baseline), дающими нижний порог качества для дальнейшего его улучшения более сложными моделями.

Заключение

Благодаря высокой точности, гибкой настройке и широкому спектру задач, на которых алгоритм хорошо работает, градиентный бустинг широко применяется во многих конкурсах по машинному обучению и в задачах из индустрии (поисковом ранжировании, рекомендательных системах, таргетировании рекламы, предсказании погоды, пункта назначения такси и многих других).

Однако, не так хорошо бустинг проявляет себя на однородных данных: текстах, изображениях, звуке, видео. В таких задачах предпочтительны нейросетевые подходы.

Универсального подхода для анализа данных нет, есть множество методов, каждый из которых обладает своей сферой применения, своими преимуществами и недостатками. Для грамотного выбора и использования необходимо разбираться теоретических и практических нюансах методов.

С устройством, применением, преимуществами и недостатками одного из таких методов - градиентного бустинга, мы познакомились в данной работе.

Источники

1. https://www.kaggle.com/ronitf/heart-disease-uci
2. https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data
3. https://catboost.ai/
4. https://ml-handbook.ru/chapters/grad_boost/intro
5. Лекции по машинному обучению К.В. Воронцова
    - http://www.machinelearning.ru/wiki/images/2/21/Voron-ML-Compositions-slides2.pdf
6. https://neurohive.io/ru/osnovy-data-science/gradientyj-busting/
7. https://habr.com/ru/company/ods/blog/327250/

Приложение 1

In [9]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
%matplotlib inline
from sklearn.metrics import mean_squared_error, recall_score, confusion_matrix
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression, Ridge
from sklearn.preprocessing import StandardScaler
from catboost import CatBoostRegressor, CatBoostClassifier, Pool
```

1. Задача классификации. Болезни сердца

In [2]:

```
heart = pd.read_csv('heart.csv')
heart
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

1.1 Логистическая регрессия

Бинаризуем категориальные небинарные переменные

In [3]:

```
categorical = ['cp', 'restecg', 'slope', 'ca', 'thal']
heart_bin = pd.get_dummies(heart, columns=categorical)
```

Масштабируем данные

Масштабируем данные

In [4]:

```
numerical = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
scaler = StandardScaler()
heart_bin[numerical] = scaler.fit_transform(heart_bin[numerical])
```

Выполняем поиск лучшего коэффициента регуляризации

In [68]:

```
X_train, X_test, y_train, y_test = train_test_split(heart_bin.drop('target',
                                                                   axis=1),
                                                    heart_bin.target,
                                                    test_size=0.25,
                                                    random_state=356)
```

In [69]:

```
C = np.logspace(-4, 4, 50)
searcher = GridSearchCV(LogisticRegression(max_iter=1000),
                        [{"C": C}],
                        scoring="f1", cv=3)
searcher.fit(X_train, y_train)
best_C = searcher.best_params_["C"]
```

Обучаем модель с лучшим коэффициентом регуляризации, проверяем качество классификации на отложенной выборке

In [70]:

```
logreg = LogisticRegression(max_iter=1000, C=best_C)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
recall_score(y_test, y_pred)
```

Out[70]:

```
0.813953488372093
```

In [71]:

```
confusion_matrix(y_test, y_pred)
```

Out[71]:

```
array([[23, 10],
       [ 8, 35]])
```

1.2 Градиентный бустинг

CatBoost почти не нуждается в предобработке данных

In [66]:

```
X_train, X_test, y_train, y_test = train_test_split(heart.drop('target',
                                                               axis=1),
                                                    heart.target,
                                                    test_size=0.25,
                                                    random_state=619)
```

```
categorical = ['cp', 'restecg', 'slope', 'ca', 'thal', 'sex', 'fbs','exang']
ctb = CatBoostClassifier(cat_features=categorical,
                         silent=True,
                         eval_metric='F1', iterations=5000)
ctb.fit(X_train, y_train, plot=True)
y_pred = ctb.predict(X_test)
recall_score(y_test, y_pred)
```

Out[66]:

```
0.8636363636363636
```

In [67]:

```
confusion_matrix(y_test, y_pred)
```

Out[67]:

```
array([[29,  3],
       [ 6, 38]])
```

Заметно, что качество классификации градиентного бустинга без какого-либо подбора оптимальных параметров выше, чем качество логистической регрессии с подобранным параметром l_2 - регуляризации

2. Задача регрессии. Ценообразование недвижимости

In [9]:

```
train = pd.read_csv('~/.kaggle/train.csv', index_col=0)
test = pd.read_csv('~/.kaggle/test.csv', index_col=0)
train.SalePrice = np.log1p(train.SalePrice) #т.к. используется метрика RMSLE
train
```

Out[9]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	...
1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	...
2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	...
3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	...
4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	...
5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	...
...	...	...	...	...	...	...	...	...	...	...	...
1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	Inside	...
1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	Inside	...
1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	Inside	...
1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	Inside	...
1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	Inside	...

1460 rows × 80 columns

2.1 Ridge-регрессия

Осуществим предобработку данных (заполним отсутствующие значения, закодируем категориальные переменные)

In [10]:

```
numerical = ['LotFrontage', 'LotArea', 'YearBuilt', 'YearRemodAdd',
             'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
             'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
             'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
             'HalfBath', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt',
             'GarageCars',  'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
             'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea',
             'MiscVal', 'MoSold', 'YrSold']
categorical = train.drop(numerical, axis=1).drop('SalePrice', axis=1).columns
categorical
```

Out[10]:

```
Index(['MSSubClass', 'MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour',
       'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1',
       'Condition2', 'BldgType', 'HouseStyle', 'OverallQual', 'OverallCond',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond',
       'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating', 'HeatingQC',
       'CentralAir', 'Electrical', 'BedroomAbvGr', 'KitchenAbvGr',
       'KitchenQual', 'Functional', 'FireplaceQu', 'GarageType',
       'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive', 'PoolQC',
       'Fence', 'MiscFeature', 'SaleType', 'SaleCondition'],
      dtype='object')
```

In [11]:

```
train[categorical] = train[categorical].fillna('NotGiven')
test[categorical] = test[categorical].fillna('NotGiven')
```

In [12]:

```
train_objs_num = len(train)
dataset = pd.concat(objs=[train, test], axis=0)
dataset_dum = pd.get_dummies(dataset, columns=categorical)
train_dum = dataset_dum[:train_objs_num]
test_dum = dataset_dum[train_objs_num:]
```

In [13]:

```
X_train, X_val, y_train, y_val = train_test_split(train_dum.drop('SalePrice',
                                                                 axis=1),
                                                  train_dum.SalePrice,
                                                  test_size=0.25,
                                                  random_state=42)
```

In [14]:

```
numeric_data = X_train[numerical]
numeric_data_median = numeric_data.median()
numeric_features = numeric_data.columns
X_train = X_train.fillna(numeric_data_median)
X_val = X_val.fillna(numeric_data_median)
```

Масштабируем данные

In [15]:

```
scaler = StandardScaler()
scaler.fit(X_train[numerical])
X_train[numerical] = scaler.transform(X_train[numerical])
X_val[numerical] = scaler.transform(X_val[numerical])
```

Выберем оптимальный коэффициент регуляризации

In [16]:

```
alpha = np.logspace(-3, 3, 100)
searcher = GridSearchCV(Ridge(max_iter=1000), [{"alpha": alpha}],
                        scoring="neg_root_mean_squared_error", cv=3)
searcher.fit(X_train, y_train)
best_alpha = searcher.best_params_["alpha"]
```

Обучим модель с оптимальным коэффициентом регуляризации

In [17]:

```
model = Ridge(max_iter=1000, alpha=best_alpha)
model.fit(X_train, y_train)
y_pred = model.predict(X_val)
mean_squared_error(y_val, y_pred, squared=False)
```

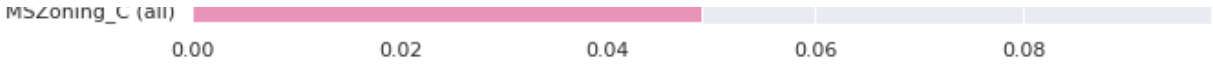
Out[17]:

```
0.13748253362426197
```

Рассмотрим **20** признаков с наибольшими по модулю коэффициентами. Заметно, что здесь за счет бинаризации категориальных переменных их значения учитываются по отдельности. Это, с одной стороны, помогает сравнить важность различных значений одного признака, но, с другой стороны, загромождает общую картину важности признаков для целевой переменной.

In [18]:

```
features = model.feature_names_in_[np.argsort(abs(model.coef_))[::-1]]
weights = np.sort(abs(model.coef_))[::-1]
sns.set(rc={'figure.figsize':(10,6)})
sns.barplot(x=weights[:20], y=features[:20]).set_title(
    'Топ-20 признаков с наибольшими по модулю регрессионными весами');
```



Два района **Crawford** и **Stone Brook (**признак **Neighborhood),** судя по всему**,** самые престижные в Эймсе**.**
Сразу же видно несколько аномалий**:** оценка **3** условий проживания **(OverallCond)** влияет на цену больше**,** чем оценка **9,** а оценка **3** качества ремонта и материалов **(OverallQual)** больше**,** чем оценка **8.**
В этом недостаток бинаризации категориальных признаков

2.2 Градиентный бустинг

Часть обучающей выборки возьмем в качестве валидационной**,** чтобы контролировать переобучение
Проверка качества модели будет осуществляться с помощью тестовой выборки**,** ответы для которой не публично хранятся на **kaggle**

In [19]:

```
X_train, X_val, y_train, y_val= train_test_split(train.drop('SalePrice',
                                                            axis=1),
                                                 train.SalePrice,
                                                 test_size=0.25,
                                                 random_state=42)
```

In [20]:

```
train_pool = Pool(X_train, label=y_train, cat_features=np.array(categorical))
val_pool = Pool(X_val, label=y_val, cat_features=np.array(categorical))
test_pool = Pool(test, cat_features=np.array(categorical))
```

Обучим модель без явного задания параметров**,** это будет опорный результат**,** который мы будем стремиться улучшить

In [21]:

```
ctb_model = CatBoostRegressor()

ctb_model.fit(train_pool, eval_set=val_pool, plot=True, verbose=False);
```

Сформируем **.csv** файл и загрузим его на **kaggle** для оценки точности на тестовой выборке

In [22]:

```
y_submit = ctb_model.predict(test_pool)
submit_dflt_params = pd.DataFrame({'Id':test.index,
                                   'SalePrice': np.expm1(y_submit)})
submit_dflt_params.to_csv('~/ds/house_prices/submit_dflt_params.csv',
                          index=False)
```

0.12948 - RMSLE на тестовой выборке

Качество предсказания "из коробки" уже сейчас выше**,** чем у **Ridge-**регрессии**,** изучим параметры

In [23]:

```
base_params = ctb_model.get_all_params()
base_params
```

Out[23]:

```
{'nan_mode': 'Min',
 'eval_metric': 'RMSE',
 'combinations_ctr': ['Borders:CtrBorderCount=15:CtrBorderType=Uniform:TargetBorde
rCount=1:TargetBorderType=MinEntropy:Prior=0/1:Prior=0.5/1:Prior=1/1',
  'Counter:CtrBorderCount=15:CtrBorderType=Uniform:Prior=0/1'],
 'iterations': 1000,
 'sampling_frequency': 'PerTree',
 'fold_permutation_block': 0,
 'leaf_estimation_method': 'Newton',
 'counter_calc_method': 'SkipTest',
 'grow_policy': 'SymmetricTree',
 'penalties_coefficient': 1,
 'boosting_type': 'Plain',
 'model_shrink_mode': 'Constant',
 'feature_border_type': 'GreedyLogSum',
 'ctr_leaf_count_limit': 18446744073709551615,
 'bayesian_matrix_reg': 0.10000000149011612,
 'one_hot_max_size': 2,
 'l2_leaf_reg': 3,
 'random_strength': 1,
 'rsm': 1,
 'boost_from_average': True,
 'max_ctr_complexity': 4,
 'model_size_reg': 0.5,
 'simple_ctr': ['Borders:CtrBorderCount=15:CtrBorderType=Uniform:TargetBorderCount
=1:TargetBorderType=MinEntropy:Prior=0/1:Prior=0.5/1:Prior=1/1',
  'Counter:CtrBorderCount=15:CtrBorderType=Uniform:Prior=0/1'],
 'pool_metainfo_options': {'tags': {}},
 'subsample': 0.800000011920929,
 'use_best_model': True,
 'random_seed': 0,
 'depth': 6,
 'ctr_target_border_count': 1,
 'posterior_sampling': False,
 'has_time': False,
 'store_all_simple_ctr': False,
 'border_count': 254,
 'classes_count': 0,
 'auto_class_weights': 'None',
 'sparse_features_conflict_fraction': 0,
 'leaf_estimation_backtracking': 'AnyImprovement',
 'best_model_min_trees': 1,
 'model_shrink_rate': 0,
 'min_data_in_leaf': 1,
 'loss_function': 'RMSE',
 'learning_rate': 0.04687400162220001,
 'score_function': 'Cosine',
 'task_type': 'CPU',
 'leaf_estimation_iterations': 1,
 'bootstrap_type': 'MVS',
 'max_leaves': 64,
 'permutation_count': 4}
```

Оптимизируем параметры: число итераций **(iterations),** максимальную глубину деревьев **(depth),** коэффициент l_2**-**регуляризации **(l2_leaf_reg)** и темп обучения или градиентный шаг **(learning_rate)**

In [24]:

```
base_learning_rate = base_params['learning_rate']
base_depth = base_params['depth']
base_l2_leaf_reg =  base_params['l2_leaf_reg']
base_iterations = base_params['iterations']
```

Выполняем поиск оптимальных параметров по сетке

In [25]:

```
grid = {'iterations' : np.array([1, 4, 8]) * base_iterations,
        'learning_rate': np.array([1, 0.1, 0.05]) * base_learning_rate,
        'depth': np.array([2/3, 1, 4/3]) * base_depth,
        'l2_leaf_reg': np.array([1/3, 1]) * base_l2_leaf_reg}

params_grid = {'eval_metric' : 'RMSE',
               'early_stopping_rounds' : 300,
               'random_seed' : 42,
               'logging_level' : 'Silent'}

ctb = CatBoostRegressor(**params_grid)
randomized_search_results = ctb.randomized_search(grid, X = train_pool,
                                                  verbose=False, n_iter=15)
best_params = randomized_search_results['params']
```

In [26]:

```
best_model = CatBoostRegressor(**best_params)
best_model.fit(train_pool, eval_set=val_pool, plot=True, verbose=False);
```

Сделаем прогноз на тестовых данных и сформируем **.csv** файл с ответами для загрузки на **Kaggle**

In [27]:

```
y_submit = best_model.predict(test_pool)
submit_best = pd.DataFrame({'Id':test.index, 'SalePrice': np.expm1(y_submit)})
submit_best.to_csv('~/ds/house_prices/submit_best.csv', index=False)
```

0.12420 - RMSLE на тестовой выборке
Подбор гиперпараметров позволил улучшить качество на **~5%**

Сохраним модель**,** чтобы в будущем для ее использования не приходилось каждый раз осуществлять поиск параметров

In [28]:

```
best_model.save_model('house_prices')
```

Загрузим модель**,** если ее имя сейчас не определено в окружении **(**если на текущем запуске ядра **Python** не искали параметры по сетке**)**

In [29]:

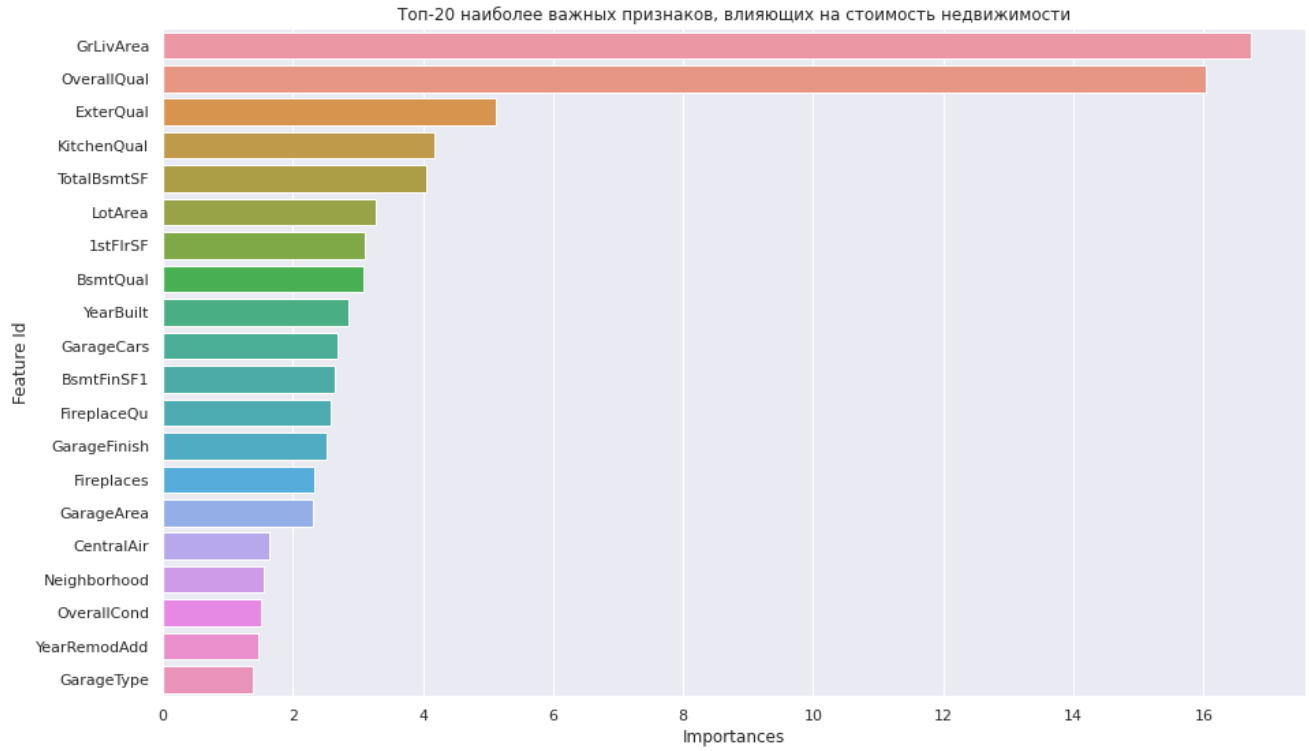
```
if 'best_model' not in globals():
    best_model = CatBoostRegressor()
    best_model.load_model('house_prices')
```

Важным преимуществом деревьев является их интерпретируемость для человека
Изучим важность признаков

In [30]:

```
fe_imp = best_model.get_feature_importance(prettified=True)
sns.set(rc={'figure.figsize':(15,9)})
sns.barplot(y = 'Feature Id', x = 'Importances', data=fe_imp[:20]).set_title(
```

```
    'Топ-20 наиболее важных признаков, влияющих на стоимость недвижимости');
```



Наибольшую с отрывом важность имеют жилая площадь недвижимости **(GrLivArea)** и оценка качества ремонта и материлов дома **(OverallQual)**

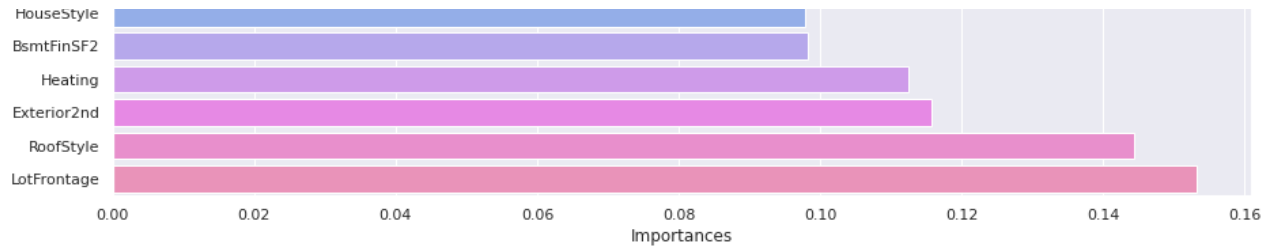
Так же, высокую важность для цены имеют качество внешней отделки дома, кухня, камины (количество и качество), гараж (вместимость, количество, качество), участок, район

Из интересного: высокую важность имеет площадь подвала, возможно, это объясняется тем, что она зачастую коррелирует с площадью всего дома

Качество стройматериалов (дорогие, дешевые) **(OverallQual)** имеет больший вес, чем их состояние, износ **(OverallCond)**, что тоже логично, т.к зачастую не только в строительстве и недвижимости качественные, хоть и не в идеальном состоянии, материалы ценятся больше, чем дешевые в идеальном состоянии

In [31]:

```
sns.barplot(y = 'Feature Id', x = 'Importances',
            data = fe_imp.sort_values(by = 'Importances')[:20]).set_title(
    'Топ-20 наименее важных признаков, влияющих на стоимость недвижимости');
```



Сразу бросается в глаза нулевой вес признака доступные коммунальные услуги **(Utilities),** посмотрим, какие значения он принимает**:**

In [62]:

```
train.Utilities.value_counts(), test.Utilities.value_counts()
```

Out[62]:

```
(AllPub    1459
 NoSeWa       1
 Name: Utilities, dtype: int64,
 AllPub      1457
 NotGiven       2
 Name: Utilities, dtype: int64)
```

И в обучающей**,** и в тестовой выборке это признак по сути константный и не влияет на результат**,** поэтому **CatBoost** его обнулил

Такая же ситуация наблюдается с типом дороги**,** ведущей к участку **(Street)** и аллеи**,** ведущей от дороги к участку **(Alley)**

In [61]:

```
train.Street.value_counts(), test.Street.value_counts()
```

Out[61]:

```
(Pave    1454
 Grvl       6
 Name: Street, dtype: int64,
 Pave    1453
 Grvl       6
 Name: Street, dtype: int64)
```

In [69]:

```
train.Alley.value_counts(), test.Alley.value_counts()
```

Out[69]:

```
(NotGiven    1369
 Grvl          50
 Pave          41
 Name: Alley, dtype: int64,
 NotGiven    1352
 Grvl          70
 Pave          37
 Name: Alley, dtype: int64)
```

Низким вкладом в цену также обладают**:**

- Стиль дома и крыши
- Площадь бассейна **(**большим вкладом обладает его наличие**,** а площадь уже вторична**)**
- Площадь крыльца

- Тип отопления и тип электрической проводки