МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**КУРСОВАЯ РАБОТА**

**МОДУЛЬ** **ИССЛЕДОВАНИЯ КАЧЕСТВЕННЫХ И КОЛИЧЕСТВЕННЫХ ХАРАКТЕРИСТИК РЫНКА НЕДВИЖИМОСТИ**

Работу выполнил Н.Д. Тиунов

(подпись)

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность Программирование и информационные технологии

Научный руководитель

канд. пед. наук, доц. А. В. Харченко

(подпись)

Нормоконтролер

канд. пед. наук, доц. А. В. Харченко

(подпись)

Краснодар

2022

**РЕФЕРАТ**

Курсовая работа 35 с., 23 рис., 7 источников.

РЫНОК НЕДВИЖИМОСТИ, АНАЛИЗ ДАННЫХ, МАШИННОЕ ОБУЧЕНИЕ, BIG DATA, PYTHON

Цель работы: исследовать качественные и количественные признаки рынка недвижимости.

В процессе работы были изучены методы внедрения машинного обучения в рынок недвижимости, а также исследование и предобработка данных.

В практической части были реализованы классические методы машинного обучения и применены к данным, характеризующим стоимость квартир.

Средства разработки: Python, Scikit-Learn, CatBoost, XGBoost, Jupyter Notebook.

**СОДЕРЖАНИЕ**

[Введение 4](#_Toc122165056)

[1 Линейные модели машинного обучения 5](#_Toc122165057)

[2 Нелинейные модели машинного обучения 8](#_Toc122165058)

[2.1 K-Nearest Neigbor 8](#_Toc122165059)

[2.2 Решающие деревья 10](#_Toc122165060)

[2.3 Ансамбли моделей 17](#_Toc122165061)

[3 Практические применения алгоритмов 25](#_Toc122165062)

[3.1 EDA-анализ 25](#_Toc122165063)

[3.2 Применение линейных моделей 29](#_Toc122165064)

[3.3 Сегментирование данных 30](#_Toc122165065)

[3.4 Применение алгоритмов градиентного бустинга 31](#_Toc122165066)

[Заключение 32](#_Toc122165067)

[Список использованных источников 33](#_Toc122165068)

ВВЕДЕНИЕ

В настоящее время характеристику рынка тяжело анализировать без машинного обучения, а любую модель машинного обучения невозможно представить без данных. Большие данные, или Big Data, — гигантские массивы разнородной информации, которые нельзя обработать вручную или обычными программами типа MS Excel. Например, годовая статистика продаж какого-то одного магазина — это обычные данные. А вот сведения о том, какие именно товары в этом году приобрели покупатели всех супермаркетов страны, по каким ценам, с какими скидками и какие отзывы об этом они оставили в соцсетях, — это уже Big Data. Чтобы их собрать, обработать и использовать, нужны специальные инструменты и технологии.[1] Анализ данных – это наука, позволяющая выявлять зависимости в данных, понимать их структуру и построение методов их обработки. Сегодня предобработка данных – это важнейшая часть внедрения ML-проектов в компании. В курсовой работе поставлена цель исследовать качественные и количественные характеристики рынка недвижимости на примере набора данных некоторой компании, занимающейся продажей квартир. Описаны методы предобработки данных и рассмотрены классические алгоритмы машинного обучения. Продемонстрирована практическая реализация модели, предсказывающей стоимость квартиры, с помощью языка программирования Python и его модулей.

1. Линейные модели машинного обучения

Линейная модель – это модель, отображающая состояние или функционирование системы таким образом, что все взаимозависимости в ней являются линейными и могут быть выражены уравнением: a(x) = β0 + β1 ⋅ d1 + β2 ⋅ d2 + ... + βn ⋅ dn. Для поиска лучших коэффициентов β используется функционал среднеквадратичной ошибки: Q(a(x), X) = , которую нужно минимизировать. Измерение качества производится на кросс-валидации. Кросс-валидация (перекрестная проверка, скользящий контроль) – это метод оценки. Допустим, что есть некий набор данных.

* делим исходные данные на k частей примерно одинаково размера,
* затем на (k - 1) блоке обучаем модели, а оставшуюся часть используем для тестирования,
* повторяем процедуру k раз, при этом на каждой итерации для проверки выбирается новый блок, а обучение производится на оставшихся,
* замеряем k средних ошибок на валидационной выборке и если их среднее/их распределение нас устраивает, то строим финальную модель.

Пример на рисунке 1:



Рисунок 1 – Разделение данных для кросс-валидации

Однако, такие модели склонны к переобучению. Переобучение - явление, когда построенная модель хорошо объясняет примеры из обучающей выборки, но относительно плохо работает на примерах, не участвовавших в обучении (на примерах из тестовой выборки).

Пример недообученных и переобученных моделей на рисунке 2:

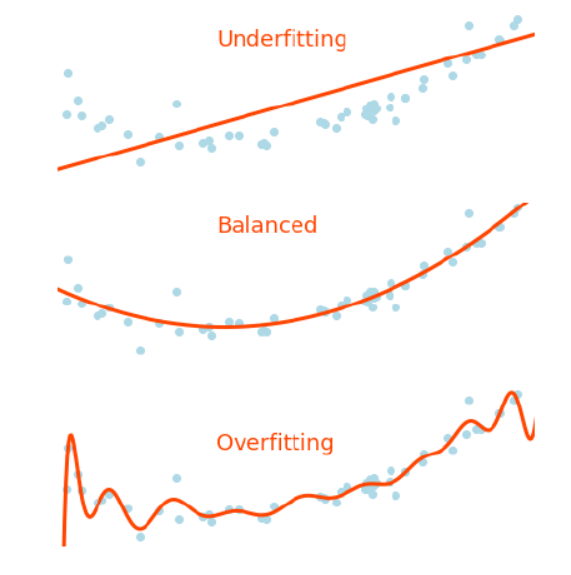


Рисунок 2 – Степени переобученности модели

Для решения этой проблемы добавляется еще одна составляющая в уже известный функционал качества: Q(a(x,β), X) + λ ⋅ R(β) —> minβ, где λ - некая константа, а R - функция регуляризации для β, например это может быть сумма квадратов β1 и β2. Цель подобрать такое значение λ, при котором модель покажет оптимальные значения на train и test. Теперь посмотрим, как можно настраивать регуляризатор R (β). Выделяют два типа:

* L-2(Ridge) - когда в качестве регуляризатора берется сумма квадратов всех параметров модели: R(β) = + +...+ .
* L-1(Lasso) - когда в качестве регуляризатора берется сумма модулей всех параметров модели: R(β) = ∣β1∣ + ∣β2∣ + ... + ∣βn∣

В описании выше могло сложиться впечатление, что трехзначные и более веса у модели - однозначный признак переобучения, от которого нужно избавляться. Однако это не всегда так. Например, предсказывается цена квартиры, выраженную в рублях (шести-семизначное число) на основе метража (двузначное число) и расстояния до метро (сотни метров): цена =β1 ⋅ м2 – β2⋅ расстояние до метро. Естественно, чтобы решить подобное уравнение коэффициент β1 должен быть порядка ста тысяч, а β2 - порядка тысячи. Бывает и такая ситуация, что только один из коэффициентов оказывается неадекватно большим и тогда при добавлении регуляризации пострадают остальные коэффициенты, так как они также будут штрафоваться.

Избежать такой ситуации может помочь приведение признаков к примерно одному порядку. То есть, если один признак находится в пределах десятка метров, а второй сотен метров, то лучше выразить последний через дециметры. Для этого в машинном обучении существуют специальные методы:

* StandardScaler

dj = , (1)

где dj - некий признак объекта, μ - среднее по выборке: μ = ∑dj, a σ - среднеквадратичное отклонение от этого среднего: σ2 = ∑(dj − μ)2

* MinMaxScaler

dj = (2)

1. Нелинейные модели машинного обучения

2.1 K-Nearest Neigbor

Алгоритм K-Nearest Neigbor или метод ближайшего соседа – метрический алгоритм, используемый при обучении с учителем. Его суть заключается в том, что таргеты похожих объектов тоже будут похожи, поэтому для предсказания нового таргета можно просто посмотреть на его соседей. Тогда для каждого нового вещественного предсказания необходимо будет посмотреть на k его соседей и усредниться по их таргету.

Общий алгоритм kNN:

* поступление алгоритму kNN нового объекта xi;
* измерение попарных расстояний между xi и всеми остальными объектами из тренировочной выборки;
* выбор k ближайших соседей;
* формирование таргетной переменной через усреднение или голосование.

Линейные модели являются параметрическими, то есть можно сначала вычислить коэффициенты β, позволяющие построить модель, а затем использовать их для всех остальных предсказаний. Более того, можно интерпретировать коэффициенты β, например, оценив важность влияния их на таргет или принцип их влияния (положительно или отрицательно влияют). Немаловажным является тот факт, что модели умеют экстраполировать зависимости и улавливать общий тренд, поэтому они могут предсказывать таргеты объектов, непохожих на предыдущие. Метод kNN в свою очередь является неинтерпретируемым, так как он смотрит на ближайшие объекты, не строя общую зависимость. С другой стороны, если зависимость переменных и таргета нелинейна, более предпочтительно применение kNN (но не всегда). Наконец из-за того, что для каждого нового объекта метод kNN считает попарные расстояния со всеми имеющимися объектами, с большими данными этот метод работает довольно медленно. Каким способом можно улучшить метод kNN?

Во-первых, можно использовать разные методы вычисления расстояния между объектами. Общая формула для вычисления расстояния между объектами — это формула расстояния Минковского:

ρ(x, z) = ( (3)

Из формулы видно, что Евклидово расстояние является частным случаем расстояния Минковского при p = 2. В свою очередь, расстояние Манхэттена также является частным случаем расстояния Минковского при p = 1

ρ(x, z) = (4)

Использование параметра p позволяет указать модели, какие объекты называть схожими и близкими друг другу, а какие нет. Другими словами, на сколько важно расстояние (разница) между объектами для классификации объекта как схожего.

p < 1 → с ростом разницы между объектами по конкретному признаку вклад этой разницы уменьшается;

p > 1 → с ростом разницы между объектами по конкретному признаку вклад разницы увеличивается.

То есть с помощью гиперпараметра p наша модель учится понимать, какие объекты называть похожими, а какие нет.

Получается, что при расчете таргета с одинаковой важностью оценивали каждого из соседей, не обращая внимания на их близость к предсказываемому объекту. Однако логичнее всё же принимать во внимание тот факт, что чем дальше сосед, тем меньше его вклад. Зная веса каждого объекта, можно вычислять таргет следующим образом: a(x) = , где w - вес каждого объекта.

Как же находить эти веса w? Рассмотрим некоторые подходы.

* базовая концепция подбора весов — возведение произвольного параметра q в степень, равную номеру объекта: wi = qi;
* другая базовая концепция — wi = , где k - произвольный параметр;
* гауссовское ядро, учитывающее расстояния между объектами wi = , где z соответствует расстоянию между двумя точками, разделенному на гиперпараметр h: z = . Чем меньше h, тем более модель становится переобученной. Поэтому под различные данные необходимо подбирать наиболее подходящий h.

2.2 Решающие деревья

Решающие деревья представляют собой невероятно мощное семейство моделей в машинном обучении. Они хоть и тесно связаны с линейными моделями, таковыми интуитивно не являются. Дадим мотивацию перед тем, как перейдем к основной концепции.

Как в реальном мире принимаются решения? Зачастую через проверку некоторых логических правил, например:

* на приеме у доктора приходится отклонять или принимать симптоматику,
* во время экзамена успешно или безуспешно отвечать на ряд вопросов,
* итоговое решение принимается сквозь призму реализовавшихся комбинаций установленных правил.

Рассмотрим пример задачи классификации для некой банковской организации, которой нужно принять решение о выдаче кредита. (рисунок 3)



Рисунок 3 – Пример данных о клиентах банка

Следующим шагом признаки переводятся в бинарные. Теперь каждый клиент описан некой комбинацией, некой цепочкой, реализацией логических признаков, т.е. таких, которые позволяют отвечать на вопрос «да» или «нет». Пример на рисунке 4:

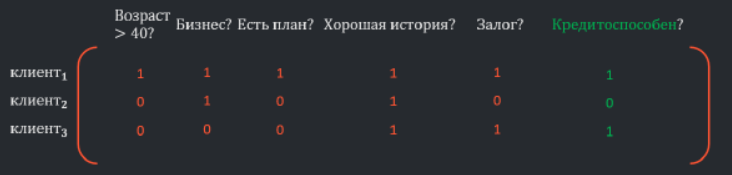


Рисунок 4 – Кодирование признаков

На основании цепочек полученных признаков можно попробовать установить между ними и таргетом зависимость. Например:

[Возраст > 40] & [Цель = Потреб] & [Хорошая история] --> Выдать кредит

[Возраст < 40] & [Цель = Бизнес] & [Нет бизнес-плана] --> Отказать

Такой алгоритм можно изобразить в виде дерева, как изображено на рисунке 5:

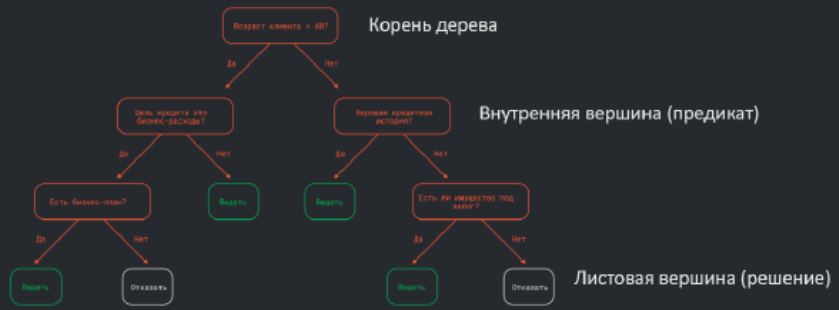


Рисунок 5 – Дерево разбиения признаков

Корень дерева — это то правило, с которого обычно начинается исследование объекта. Внутренняя вершина (предикат) — проверяет правило. Листовая вершина — в ней находится решение.

Это дерево можно назвать бинарным, так как можно посмотреть на количество потомков каждой вершины и увидеть, что их число равняется двум, т.е. можем ответить да или нет, третьего варианта не дано. Пример на рисунке 6:

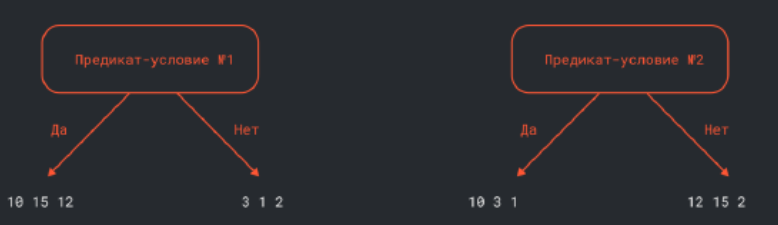


Рисунок 6 – Пример бинарного дерева

Когда необходимо решать задачу регрессии и находимся в некоторой листовой вершине, чтобы дать прогноз, смотрится на объекты из обучающей выборки и усредняемся по ним. По факту, когда выбирается лучший предикат, что для задачи регрессии, что для задачи классификации, предстоит задача сделать такое разбиение объектов на две части, чтобы в этих частях оказалось как можно больше похожих друг на друга объектов. Слева, правило №1 справляется с этой задачей значительно лучше, чем правило №2. Теперь формально научимся оценивать хаотичность в разбиении для задачи регрессией. С этим достаточно хорошо справляется дисперсия. Ведь она оценивает то, как в среднем числа отклоняются от своего среднего значения, иначе говоря, матожидания в квадрате, т.е. чем данные меньше отклоняются от среднего, тем менее хаотичны, если больше, то более хаотичны.

При помощи дисперсии замерим критерий качества, для этого нам нужно взять начальную дисперсию и подсчитать взвешенную после разбиения. Когда находились в какой-то вершине и выбрали какой-то лучший предикат, необходимо понять следующее: в правую и левую вершину от него идем, надо ли дальше строить дерево или уже в какой-то из этих вершин можно давать прогноз? Нужны какие-то формальные критерии останова, которые будут сигнализировать нашему дереву о том, что можно прекратить дальнейшее построение. Как раз в качестве такого критерия останова может выступать уровень хаотичности, если он нас внутри вершины устраивает, тогда построение заканчиваем, а если оказывается достаточно большим, то продолжаем. Пример на рисунке 7:

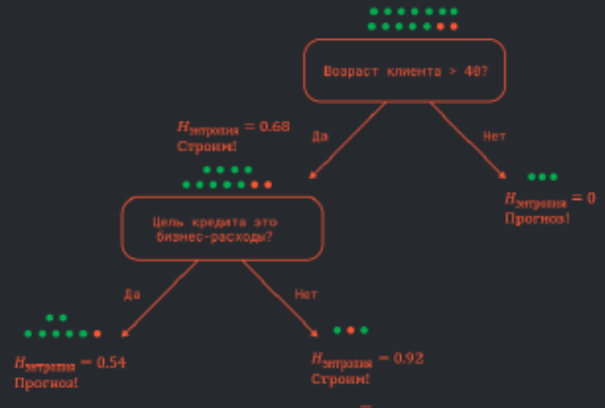


Рисунок 7 – Критерий останова на примере уровня хаотичности

Второй вариант остановки построения дерева — это проверять глубину, на которой находится вершина, например, если глубина больше пяти, то останавливаемся. Пример на рисунке 8:



Рисунок 8 – Критерий останова на примере глубины

Еще один очень популярный критерий останова — листовой, который сравнивает количество объектов, которое оказалось в вершине. Если объектов, меньше, чем пять, тогда вершину объявляем листовой, если больше, тогда объявляем внутренней и продолжаем строительство. Пример на рисунке 9:

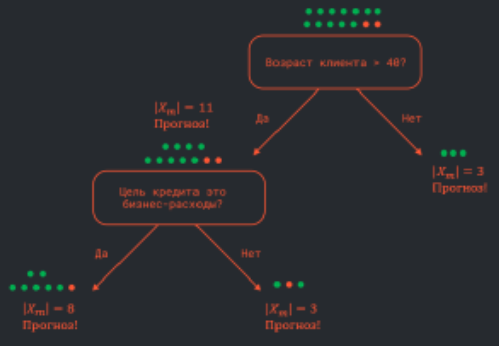


Рисунок 9 – Листовой критерий останова

Необходимо объединить воедино весь пройденный материал и придумать какой-то алгоритм, который поможет от начала и до конца построить дерево. Самый популярный алгоритм — это так называемый жадный алгоритм. Это рекурсивный алгоритм, который помогает нам построить дерево от начала и до конца, при этом для каждой вершины находя лучший предикат или сразу объявляя эту вершину листовой, если в ней выполняется критерий останова. Почему данный алгоритм называется жадным? Идея следующая: пусть на каждом этапе выбирали лучший предикат, но можно согласиться, что это ни каким образом не свидетельствует о том, что наше дерево идеально, что оно самое лучшее. Каждая вершина влияет на последующую, и, может быть, стоило начать с другого предиката. Буквально на каждом шаге алгоритма жадничаем, выбираем самый лучший предикат, но прогноз нашего дерева — это не конкретно какое-то правило, выбранное для нашего дерева, а последовательность, но при этом на практике это оказывается достаточным для того, чтобы улавливать какие-либо сложные зависимости и хорошо делить наше признаковое пространство.

Решающие деревья - мощные модели, способные добиваться идеального качества на тренировочной выборке. При этом основная проблема таких моделей - это их склонность к переобучению. Поэтому необходимо как-то их сдерживать и регулировать. Пример переобученной модели на рисунке 10:

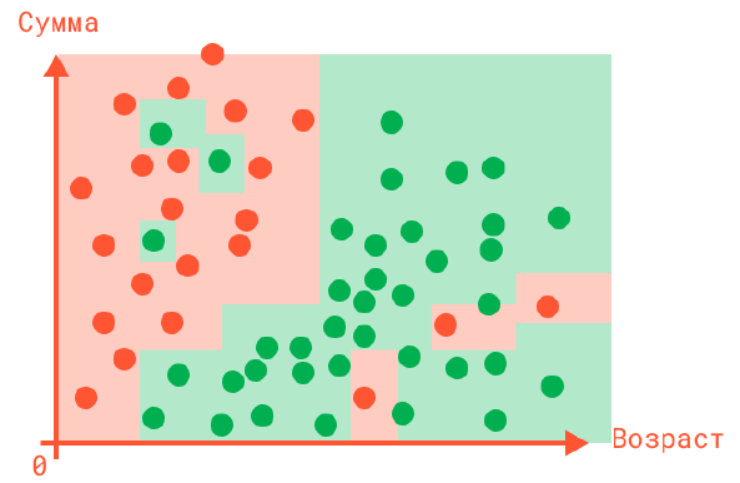


Рисунок 10 – Сегментация переобученной модели

Пример оптимальной модели на рисунке 11:

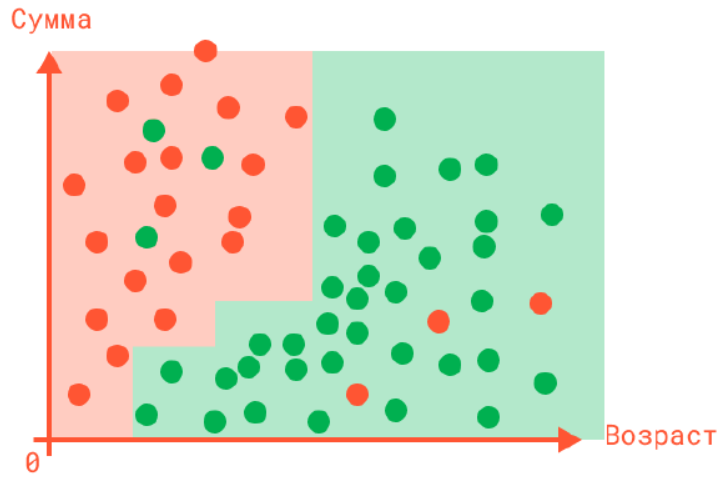


Рисунок 11 – Сегментация оптимальной модели

Основные признаки переобученного дерева решений есть ряд признаков, которые можно использовать в качестве гиперпараметров.

* Гиперпараметр 1: Максимальная глубина дерева- у переобученного дерева очень большая глубина;
* Гиперпараметр 2: Число объектов-признаков во внутренних вершинах- в переобученном дереве мало объектов во внутренних вершинах;
* Гиперпараметр 3: Число объектов в листовой вершине - в переобученном дереве мало объектов в листовых вершинах;
* Гиперпараметр 4: Максимальное количество листов - в переобученнном дереве как правило огромное количество листов;
* Гиперпараметр 5: Минимальный прирост качества. Требование, что функционал качества при дроблении улучшался как минимум на n процентов. Если с последующим разбиением на новые вершины энтропия уменьшается незначительно, то можно останавливать обучение. Задача - найти лучшие комбинации этих гиперпараметров с помощью кросс-валидации.

Деревья осуществляют прогноз следующим образом:

* при построении дерева пространство признаков разбивается на множество непересекающихся областей D1 ,D2,…, Dk;
* каждой области соответствует определенный прогноз wj;
* когда модели поступает новый объект, он определяется в ту или иную область размеченного пространства.

В общем виде формула данной модели выгляди следующим образом:

a(x) = w1 ⋅ [x ∈ D1 ] + w2 ⋅ [x ∈ D2 ] +…+ wk ⋅ [x ∈ Dk], где w1, w2,…, wk - веса-прогнозы каждой области , x ∈ Dk - индикатор того, принадлежит ли объект данной области пространства (1 - принадлежит, 0 – не принадлежит). Теперь вспомним как строятся линейные модели. Имеются некоторые признаки, каждый признак домножается на какой-либо вес, результат суммируется. По сути, тут происходит то же самое, только здесь не строится модель над базовыми признаками, а используются новые бинарные признаки принадлежности к определенным областям пространства, в зависимости от соответствия ряду правил.

2.3 Ансамбли моделей

До этого было рассмотрено несколько моделей машинного обучения: метод ближайших соседей, линейные алгоритмы и решающее дерево. Все эти алгоритмы имеют свои преимущества и недостатки: у метода ближайших соседей недостаточная обобщающая способность, но он прост в понимании. Линейный алгоритм не имеет достаточное количество степеней свободы для сложных зависимостей, но стабилен к выбросам. Решающее дерево обладает значительной вариативностью, но склонно к переобучению и требовательно к размеру обучающей выборки.

Что если, использовать несколько алгоритмов машинного обучения одновременно? Чтобы вместе они образовывали более сильную модель, как-бы, компенсируя недостатки друг друга. Построение композиции алгоритмов (ансамбль моделей, Ensemble of models) - является важной концепцией в машинном обучении. Идея композиции проста: при решении задачи, каждый, даже очень качественный, алгоритм может ошибаться. Но если, разные виды моделей могут "изучить" разные детали одной и той же задачи, то скомбинировав результаты прогнозов различных моделей, можно получить более точный результат. Применение нескольких алгоритмов для повышения точности результата можно пронаблюдать на примере теоремы Кондорсе о жюри присяжных. Если каждый алгоритм угадывает ответ на конкретном объекте более чем в 50% случаев и количество алгоритмов стремится к бесконечности, то с вероятностью 1, большинство из них окажутся правы. Иллюстрацией принципа Кондорсе является эксперимент Фрэнсиса Кальтона и такой метод принятия решений называется методом простого голосования.

Бэггинг (bootstrap aggregation) - принцип построения композиции, построенный на простом голосовании. Обычно, в случае бэггинга базовые алгоритмы используются из одного семейства. Пусть имеем N базовых алгоритмов: b1(x), b2(x), ..., bn(x). Тогда, в случае регрессии финальным ответом будет среднее по всем алгоритмам, а в случае классификации решение через простое голосование: a(x) = . Описанный алгоритм, если в качестве базовых алгоритмов выступают решающие деревья, ещё называют случайным лесом. Базовые алгоритмы могут быть похожи, ведь если на одной и той же выборке каждый раз строить дерево с одинаковыми гиперпараметрами, тогда в результате получится одинаковая модель. И строить композицию из одинаковых моделей, а потом усреднять по ним прогноз бессмысленно. Значит необходимо как-то изменить модели, например, добавить в процесс построения выборки случайность. Называется этот подход bootstrap.

Пусть у нас имеется выборка X = . Возьмем случайным образом элемент этой выборки l раз с возвратом (возвращаем выбранные элементы в выборку), получим псевдовыборку. Повторим процедуру N раз и получим N подвыборок X1 ,…,XN размера l. Если можно рандомным образом сгенерировать N подвыборок, значит можно обучить N различных моделей. Пусть у нас есть большой датасет, с помощью процедуры бутстрапа можно разбить этот датасет на 3 подвыборки, т.е сгенерировать 3 экземпляра, которые будут приблежать нашу изначальную выборку. Затем, обучим решающее дерево на полученных подвыборках и получим 3 разных модели. Финальная модель, бэггинг ансамбль - среднее значение от построенных моделей. Представим, обучили N базовых алгоритмов: b1(x), b2(x), ..., bn(x). Ошибку каждого алгоритма в случае MSE на объекте x можно записать как = (bi(x) − y(x))2. А среднее её значение, то есть ожидаемое: Ex(bi(x) − y(x))2 = Ex. Если для каждого i-того базового алгоритма можно усредниться по его ошибкам, то также можно усредниться от среднего, т.е посчитать какие в среднем средние ошибки имеют базовые алгоритмы. Обозначим за величину E1 среднюю ожидаемую ошибку по всем обученным алгоритмам: E1 = . Построим бэггинг-алгоритм a(x) = и замерим его ожидаемую ошибку EN: E = Ex().

Допустим, ошибки базовых алгоритмов нескоррелированы, тогда = 0 Если это слагаемое равно нулю, то его можно опустить: E = Ex = = E1*.* Получается, ошибка бэггинг - алгоритма уменьшится в N раз!Можно сделать вывод: если построить бэггинг на нескоррелированных базовыхалгоритмах, то средняя ошибка итогового алгоритма уменьшиться в N раз (всравнении со средней ошибкой базовых алгоритмов).

Случайный лес - один из популярнейших способов применить бэггинг. Суть заключается в том, чтобы в качестве базовых моделей b1(x), b2(x),…, bN (x) взять решающие деревья. И желательно, чтобы базовые модели были максимально некоррелированные, чтобы прогнозы и ошибки были разные. Но как этого достичь? Во-первых, рандомизация достигается за счет бутстрапа. Во-вторых, включается метод случайных подпространств: при построении решающих деревьев выберется лучший предикат не среди всех признаков, а из случайно выбранных p < m. Данная технология позволяет получать различные друг от друга решающие деревья и это добавляет рандомизации в построение базовых алгоритмов. При решении задачи регрессии/классификации число выбираемых признаков должно быть равно примерно или (округляя вниз). Пример на рисунке 12:



Рисунок 12 – Добавление случайности

Пусть у нас есть большой датасет, с помощью процедуры бутстрапа можно разбить этот датасет на 3 подвыборки, т.е сгенерировать 3 экземпляра, которые будут приблежать нашу изначальную выборку. Затем, обучим глубокое решающее дерево на полученных подвыборках и получим 3 разных модели b1(x), b2(x),…, bN (x). Финальная модель, бэггинг ансамбль — среднее значение от построенных моделей.

Каждый раз, при построении ансамбля, который генерирует подвыборки (с помощью бутстрап) и на них обучается, можно посчитать ошибку особого вида. Строя случайный лес не обязательно делить выборку на train и test, чтобы понять обобщающую способность алгоритма. При использовании случайного леса нет необходимости в кросс-валидации, чтобы получить несмещенную оценку ошибки набора тестов. Бутстрапируя выборку, для отдельных алгоритмов часть объектов выкидываются. Тогда для каждого объекта можно замерить ошибку по тем деревьям, которые на нем не обучались, и получить out-of-bag ошибку: OOB = . Недостаток случайного леса - это слишком долгое и дорогое вычисление. Чтобы его вычислить нужно спросить все базовые алгоритмы, а затем усреднить предсказания. Что если, не хочется обучать тысячи обучающих деревьев, а хотим обойтись небольшим количеством алгоритмов. Пусть у нас есть N базовых алгоритмов b1(x), b2(x),…, bN(x). Эти модели уже обучены и могут давать какие-то прогнозы. И для каждого объекта x можно получить различных прогнозов, отправив его в каждую обученную базовую модель.

Теперь можно использовать эти прогнозы как новое признаковое пространство и отдать их мета-алгоритму, который обучается на выходах базовых моделей. То есть использовать прогнозы N базовых алгоритмов как новые фичи и скормить их в финальную модель. Пусть у нас есть большой датасет, с помощью процедуры бутстрапа можно разбить этот датасет на 3 подвыборки, т.е сгенерировать 3 экземпляра, которые будут приближать нашу изначальную выборку. Затем, обучим 3 различных моделей на полученных подвыборках, а прогнозы моделей будем использовать как новое признаковое описание объектов. И уже на полученных фичах обучим финальную модель. Проблема классического стекинга заключается в склонности к переобучению. Например, среди базовых алгоритмов затесалось глубокое переобученное дерево, тогда мета-алгоритм подстроится под такую базовую модель и тоже переобучится. Например, a(x) = 1 ⋅ b1(x) + 0 ⋅ b2(x) + 0 ⋅ b3(x), где b1(x) переобученный базовый алгоритм.

Бороться с переобучением в стекинге можно с помощью отложенной выборки. Обучать базовые модели будем не на всем трейне Xtrain, а на отложенном кусочке Xtraink Метамодель будем обучать на оставшейся части, то есть на Xtrain/Xtraink. Пример на рисунке 13:

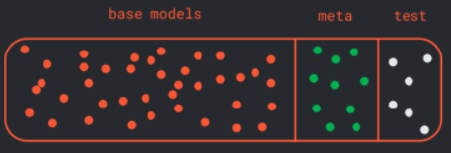


Рисунок 13 – Разбиение данных

Или более сложный вариант-модификация: Разобьем обучающую выборку на K частей, каждый из базовых алгоритмов b1(x), b2(x),…, bN(x) обучим K раз на разной отложенной выборке. Финальную метамодель будем строить, минимизируя сумму ошибок по каждому фолду. Например, рисунок 14:



Рисунок 14 – Разбиение данных на фолды

До этого в композициях базовые алгоритмы обучались независимо. В этом есть свой плюс: так как весь процесс можно сделать параллельным. Идея бустинга: что, если базовые алгоритмы строить не независимо, а последовательно? Чтобы каждая последующая модель исправляла ошибки предыдущей. Пусть решаем задачу обучения с учителем и построили какой-то изначальный алгоритм: b1(x) = argmin. Можно замерить ошибки такого алгоритма на каждом объекте i:

= yi – b1(xi). Следующий алгоритм должен исправлять ошибки первого. Сделаем ошибки базового алгоритма новым таргетом для следующего алгоритма. Т.е обучим еще одну модель на сдвигах . b2(x)=argmin. Как получить финальную композицию? Чтобы получить финальный прогноз, нужно просуммировать результаты всех моделей: a2(x) = . На практике двух моделей недостаточно, поэтому новые модели добавляются по такому же принципу. Формула в общем виде выглядит следующим образом: bN = argmin = argmin.

Пусть оптимизируется какая-то метрика с функцией потерь L. Ансамблевая модель - это сумма базовых моделей с некоторыми весами. Теперь, когда строим градиентный бустинг, не просто суммируем модели, а еще умножаем каждую модель на какую-то константу. a(N)(x)=

Чтобы понять, как работает градиентный бустинг, пойдем от обратного: пусть построили композицию a(N-1)(x) из N - 1 базовых алгоритмов. Как происходит выбор следующей модели? Обычно производилась минимизация суммы Loss по каждому объекту. После того, как добавили следующую модель, какой прогноз будет давать композиция? → . Посмотрим на формулу и подумаем, какие значения должна возвращать новая базовая модель. Пусть это какие-то числа si, но какими они должны быть? → . Вообще, с добавлением нового алгоритма хотим как можно сильнее минимизировать функционал выше, то есть сделать шаг в сторону антиградиента. Тогда желаемые si: si = -. Тогда будем подбирать bN(x) по следующей схеме: bN(x) = argmin2. Заметим, что здесь используем квадратичную функцию потерь. Можно использовать и другие, но, как правило, данной функции потерь вполне достаточно. После того, как новый алгоритм bN(x) найден, осталось найти вес этой базовой модели. Здесь уже в явном виде оптимизируем изначальный функционал ошибки по единственному параметру yN, что будет легко сделать. После всех действий добавляем новую модель к ансаблю: aN(x) = γNbN(x) + .

1. Практические применения алгоритмов
   1. EDA-анализ

Рассмотрим задачу предсказания стоимости квартир на основе данных, взятых из открытого источника [2]. Произведем EDA-анализ, реализуем классические алгоритмы машинного обучения и построим модель с наилучшим качеством. Будем рассматривать MSLE в качестве метрики, используемой для оценки качества моделей.

Необработанная таблица с данными содержит 292 признака. Для начала удалим признаки, описывающие идентификационные номера некоторых метро, автобусных станций и так далее. То есть избавимся от колонок: “ID\_metro”, “ID\_big\_road1”, “ID\_railroad\_station\_walk”, “ID\_railroad\_station\_avto”, “ID\_big\_road2”, “ID\_railroad\_terminal”, “ID\_bus\_terminal”, “id” . Для оценки качества прологарифмируем колонку с уже известными ценами домов “price\_doc”.

Удалим все колонки, в которых более 50% пропусков. Заполним все пропуски в вещественных данных средним значением, а в категориальных – самым популярным значением. Посмотрим на матрицу попарных корреляций для вещественных признаков на рисунке 15:

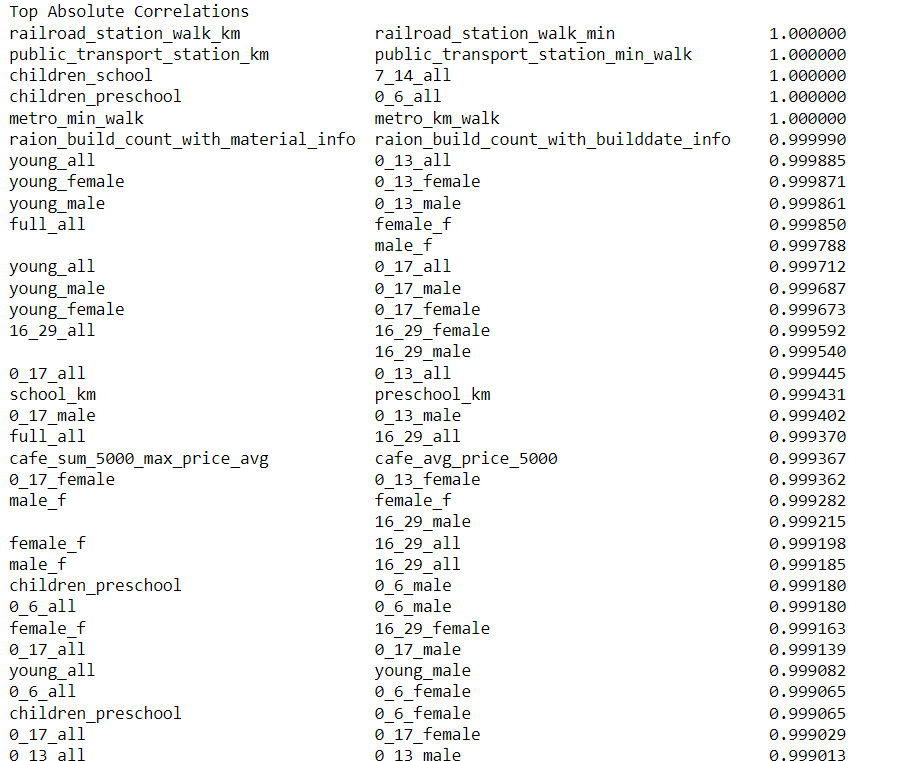


Рисунок 15 – Попарные корреляции

Видно, что достаточно большое количество сильно коррелирующих признаков. Удалим все колонки, корреляция которых выше 0.9 с хотя бы одним из остальных, чтобы решить проблему мультиколлиниарности. Таким образом в данных остается 150 признаков.

Рассмотрим все квазикостантные вещественные признаки и уберем те, у которых дисперсия менее 0.1. Итого избавимся от колонок: “green\_zone\_part”, “indust\_part”, “green\_zone\_km”, “cafe\_count\_500\_price\_high”, “mosque\_count\_500”, “mosque\_count\_1000”, “mosque\_count\_1500”, “mosque\_count\_2000”.

Преобразуем все категориальные колонки по следующему правилу: для всех колонок, где менее 5 уникальных значений применим One-Hot-Encoding, а для остальных Mean-Target-Encoding. Так же отдельно обработаем колонку “timestamp”. Разделим ее на две колонки – месяц и год. Разумеется, что стоимость квартиры должно зависеть от года и месяца. Проверим данное предположение. Посмотрим на распределение цены в зависимости от месяца при помощи boxplot на рисунке 16:

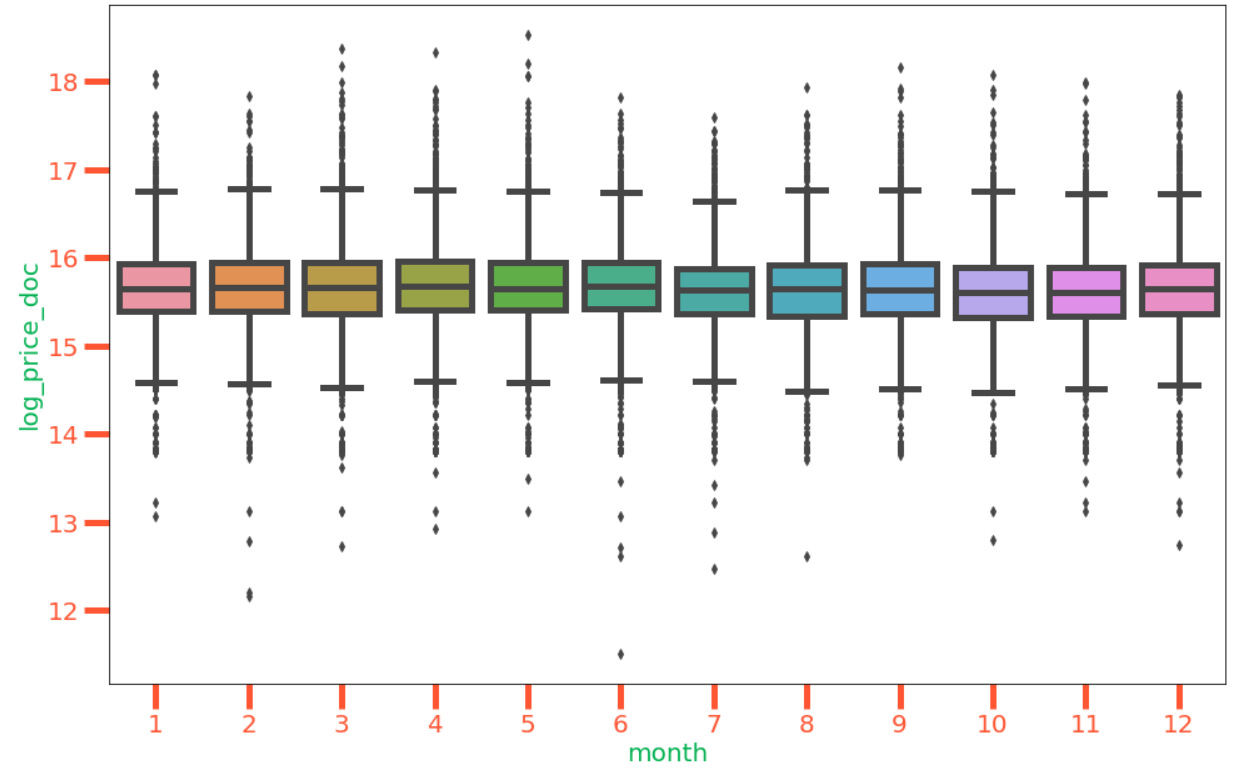


Рисунок 16 – График зависимости цены от месяца

Видим, что цена действительно, хоть и не сильно, но варьируется в зависимости от месяца. Тогда оставим данный признак, закодировав его при помощи One-Hot-Encoding. Теперь посмотрим на зависимость цены квартиры от года на рисунке 17:

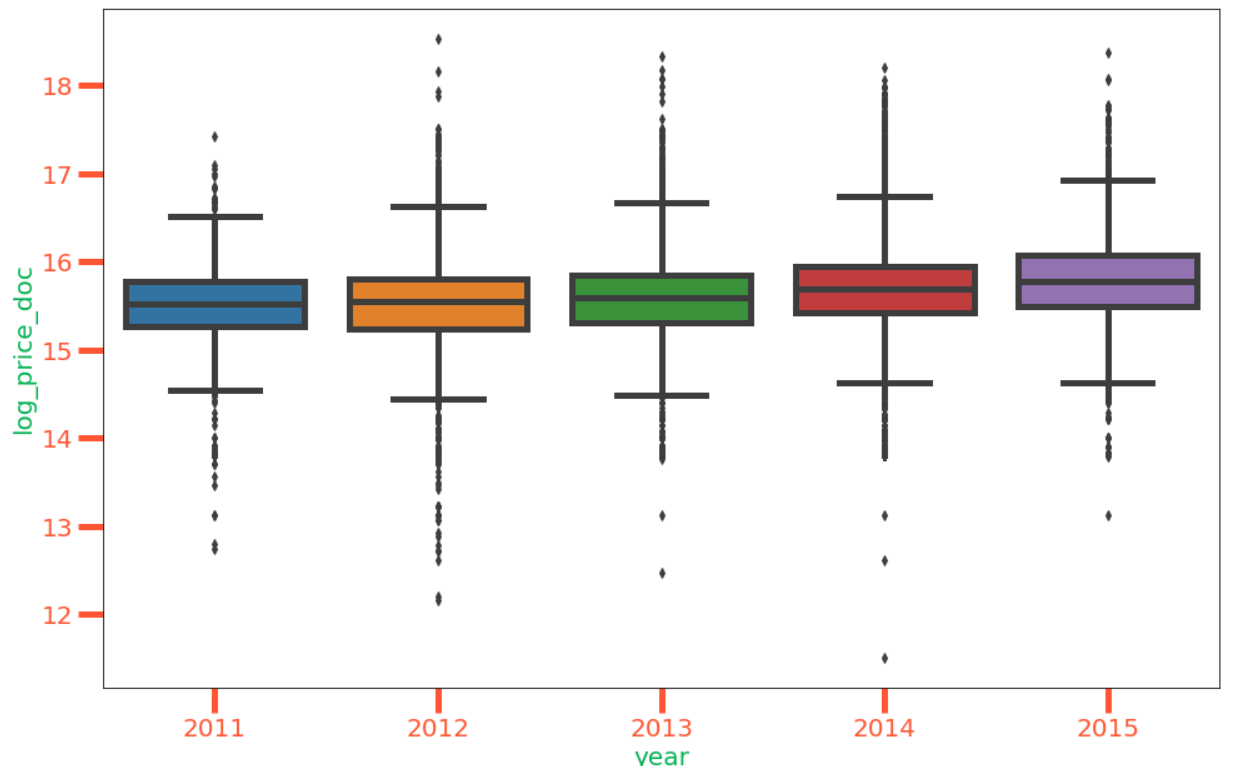


Рисунок 17 – График зависимости цены от года

Из графика очевидно, что гипотеза о том, что год влияет на цену квартиры, верна, поэтому аналогично закодируем столбец.

Посмотрим на признаки, которые у нас остались. Среди наиболее сильных признаков можно выделить этаж квартиры, количество магазинов в радиусе 3км, эта квартира в новостройке или нет. Для убеждения в этом посмотрим на зависимости между этими признаками и цены квартиры.

Boxplot между этажом и ценой на рисунке 18:

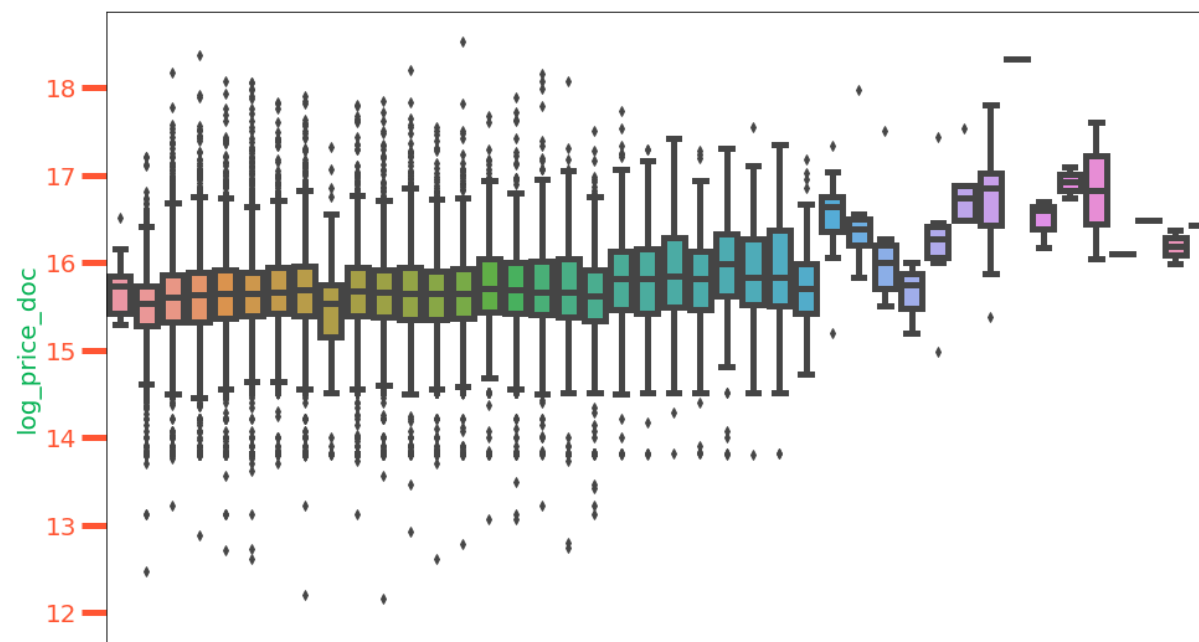


Рисунок 18 – График зависимости цены от этажа

Между количеством магазинов в радиусе 3км и ценой на рисунке 19:

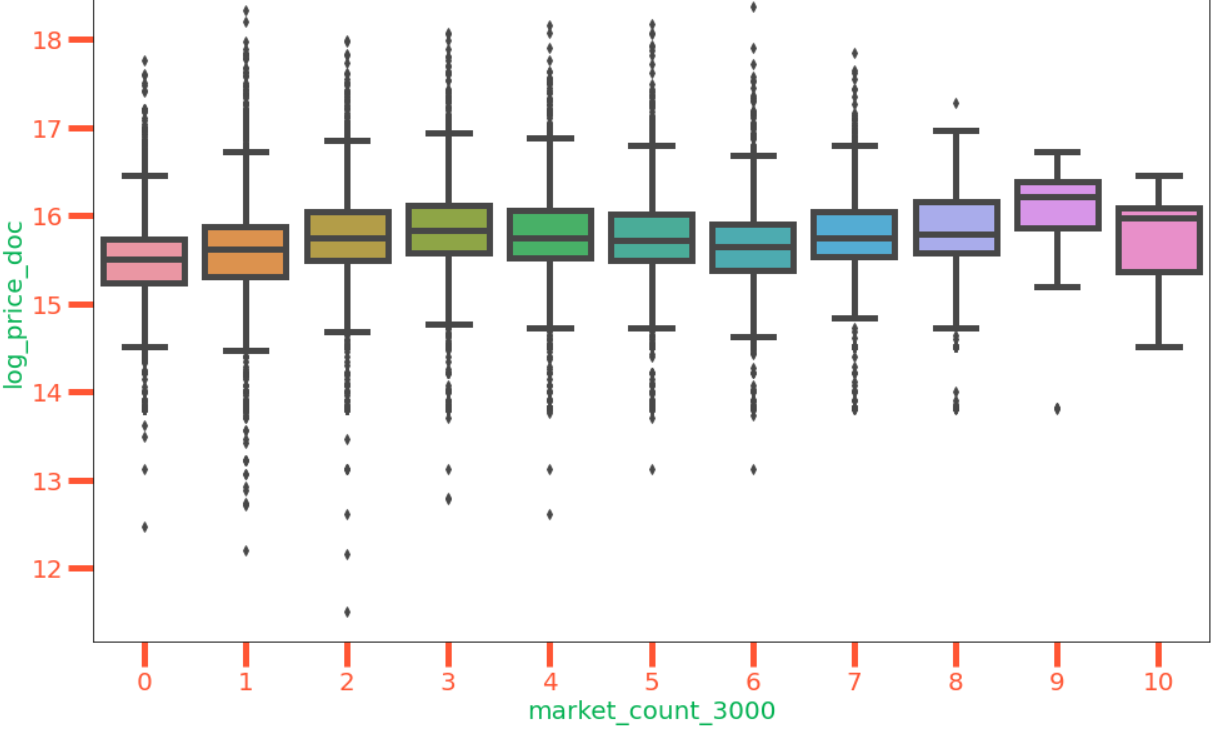


Рисунок 19 – График зависимости цены от количества ближайших магазинов

Между типом квартиры и ценой на рисунке 20:

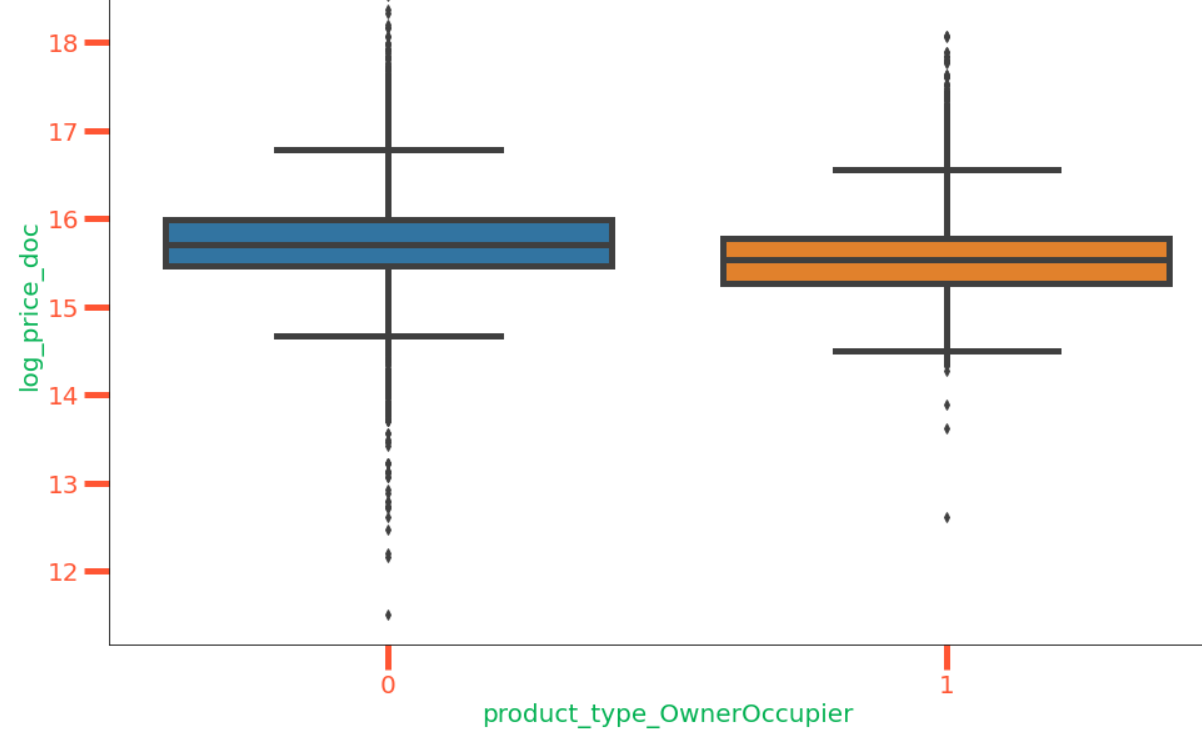


Рисунок 20 – График зависимости цены от типа квартиры

Анализ выбросов: уберем из данных те объекты, у которых значения вне отрезка от 2,5% до 97,5% всех цен.

3.2 Применение линейных моделей

Выбранные данные обладают временной структурой, поэтому, чтобы получить хорошую обобщающую способность, необходимо построить не просто модель, хорошо работающую на новых данных, а модель, которая угадывает распределение данных в будущем хотя бы на коротком горизонте. Поэтому, когда происходит валидация дизайна модели, нам важно делить на каждом шаге train и test таким образом, чтобы по временной шкале они не пересекались, и точки из второго множества появлялись позже точек из первого. Воспользуемся TimeSeriesSplit из библиотеки Scikit-learn и сделаем разбиение на 4 части.

Scikit-learn – это библиотека Python, которая является одной из самых полезных библиотек Python для машинного обучения. Она включает все алгоритмы и инструменты, которые нужны для задач классификации, регрессии и кластеризации. Она также включает все методы оценки производительности модели машинного обучения. [3]

Применим стандартную линейную регрессию и получим, что MSLE на выборке train будет 0.24, а на выборке test будет 1.315. Можно понять, как и предсказывалось до этого, что модель переобучилась. Рассмотрим на практике L\_1 регуляризацию, однако при его применении получаем ужасное качество на выборке test: 66.603. Добавим масштабирование данных и подготовим итоговый PipeLine, а также подберем лучшее λ при помощи GridSearchCV. В итоге получим MSLE на train 0.29, а на test 0.328. Видим, что качество заметно улучшилось.

3.3 Сегментирование данных

Одним из самых важных признаков является тип квартиры: первичка или нет. В целом интуитивно понятно, что для разных типов нужны разные модели. Поэтому разделим данные по типу квартиры, а затем в каждом из двух типов возьмем выборки train и test. Теперь необходимо определиться с тем, как оценивать качество сразу двух моделей. Если просто найдти MSLE каждой из них и усреднить, то это будет не совсем точным качеством, так как каждая категория имеет разную долю в общей таблице. Поэтому произведем рассчет по следующей формуле:

n = share\_first \* error\_first\_train + share\_second \* error\_second\_train, где share\_first и share\_second – доли первой и второй категориях соответственно, а error\_first\_train и error\_second\_train – ошибки на выборке train каждой модели, обученной на первой и второй категориях. В качестве моделей для обеих выборок будем обучать финальную версию собранного нами Pipeline. Итоговое качество на рисунке 21:

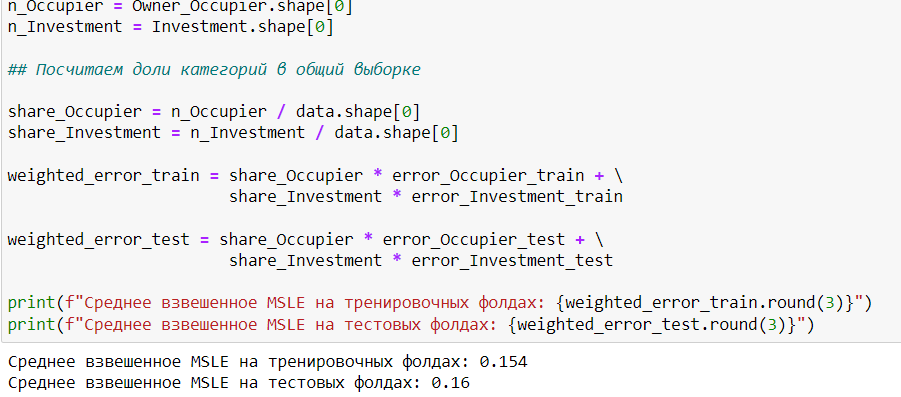


Рисунок 21 – Итоговое качество

3.4 Применение алгоритмов градиентного бустинга

Рассмотрим на практике 2 самых популярных метода градиентного бустинга: catboost, xgboost. CatBoost может назначать категориальные индикаторы переменных, а затем получать результат формы однократного горячего кодирования через однократный максимум (один-горячий максимум: для всех функций используйте одноразовое кодирование для разных чисел, меньших или равных заданному значению параметра). Если в операторе CatBoost не задано значение «skip», CatBoost будет обрабатывать все столбцы как числовые переменные. Если столбец данных содержит строковые значения, алгоритм CatBoost выдаст ошибку. Кроме того, переменные типа int со значениями по умолчанию также будут обрабатываться как числовые данные по умолчанию. В CatBoost переменные должны быть объявлены до того, как алгоритм сможет рассматривать их как категориальные переменные.

Для категориальных переменных, количество возможных значений которых превышает максимальное количество одного цикла, CatBoost использует очень эффективный метод кодирования, который аналогичен среднему кодированию, но может уменьшить переобучение.

XGBoost не может обрабатывать категориальные переменные, но, как и случайный лес, он принимает только числовые данные. Следовательно, прежде чем классифицированные данные будут переданы в XGBoost, данные должны быть обработаны с помощью различных методов кодирования: таких как кодирование маркеров, среднее кодирование или одноразовое кодирование. [5]

После обучения модель XGBoost выделяет наиболее важные признаки на рисунке 21:

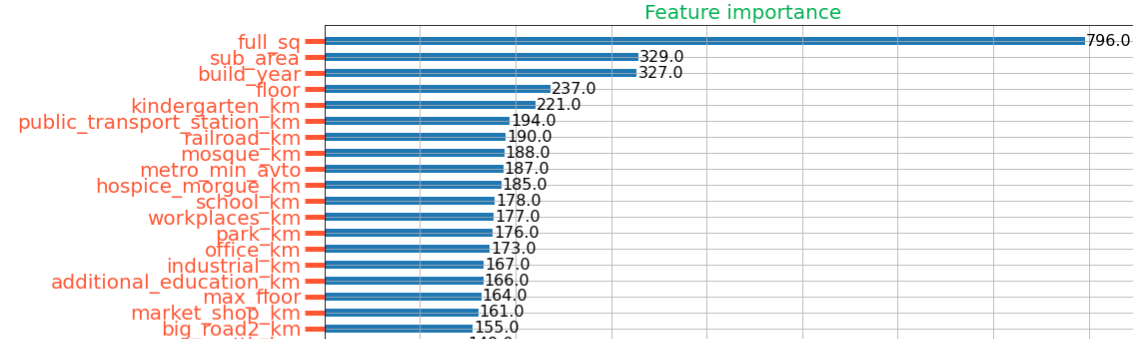


Рисунок 22 - Важность признаков

Подбираем лучшие гиперпараметры для модели XGBoost на train выборке при помощи GridSearchCV, обучаем ее на разделенных данных и получаем итоговое качество: 0.08 на тестовой выборке.

По сравнению с XGBoost, catboost обучается гораздо быстрее. В процессе обучения получали качества на рисунке 22:

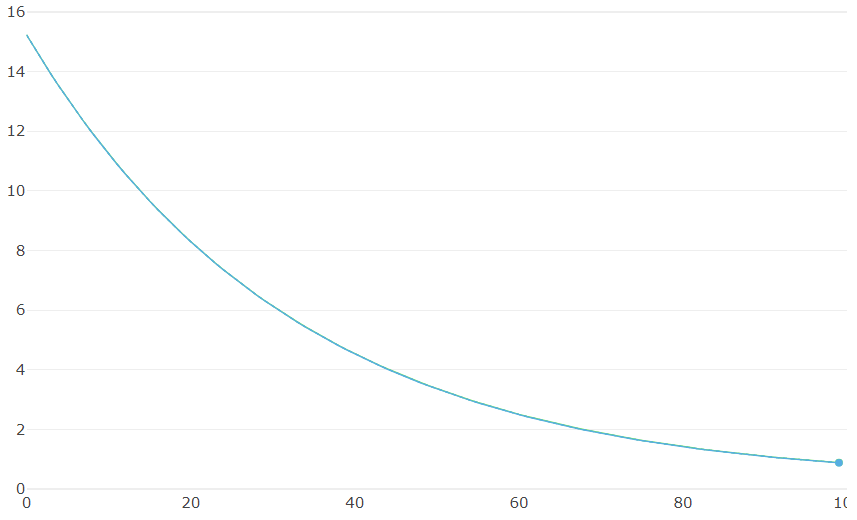


Рисунок 23 – MSE на обучающей выборке

Аналогично подбираем лучшие гиперпараметры для модели catboost на train выборке, обучаем ее на разделенных данных и получаем итоговое качество: 0.1 на тестовой выборке.

Можем сделать вывод о том, что лучшее качество на тестовой выборке выдает самая тяжелая модель – XGBoost. Следовательно, как основную модель в prod необходимо выбирать именно ее.

ЗАКЛЮЧЕНИЕ

В работе были представлены исследования качественных и количественных характеристик рынка недвижимости при помощи классических моделей машинного обучения. Были проведены анализ и сегментация данных, а также разобраны линейные и нелинейные модели машинного обучения с последующим их практическим применением при помощи модуля scikit-learn.

Изучен рынок недвижимости на основе предоставленных компанией данных. Рассмотрены методы внедрения машинного обучения в компании, работающие в сфере недвижимости. Главное преимущество внедрения ML-моделей в рынок недвижимости – это точность предсказанных стоимостей, что помогает компаниям эффективно оценивать рыночную ситуацию.

В дальнейшем планируется разобрать рекомендательные системы для пользователей, а также способы их внедрения в бизнес. Будут разобраны нейросетевые технологии рекомендаций и проведение A/B тестирования для проверки качества работы рекомендательных сетей в онлайне.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Образовательно-медийный портал SkillBox, официальный сайт – URL: <https://skillbox.ru/media/code/dannye_eto_novaya_neft_ili_obyknovennyy_khayp/> (дата обращения: 04.12.2022)
2. Образовательно-соревновательный портал Kaggle, официальный сайт – URL: https://www.kaggle.com/ (дата обращения: 01.11.2022)
3. Документация библиотеки scikit-learn, официальный сайт – URL: <https://scikit-learn.org/stable/> (дата обращения: 01.12.2022)
4. Новостной интернет-ресурс Habr, официальны сайт – URL: <https://habr.com/ru/company/ods/blog/327250/> (дата обращения: 10.11.2022)
5. Интернет-блог RussianBlogs, официальный сайт – URL: <https://russianblogs.com/article/66121235194/> (дата обращения: 01.12.2022)
6. Интернет-энциклопедия Википедия, официальный сайт – URL: <https://en.wikipedia.org/wiki/Linear_regression> (дата обращения: 01.12.2022)
7. Интернет-энциклопедия Википедия, официальный сайт – URL: https://en.wikipedia.org/wiki/K-nearest\_neighbors\_algorithm (дата обращения: 01.12.2022)