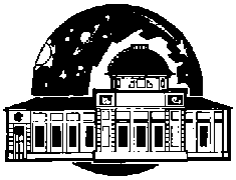


Государственное общеобразовательное учреждение Ярославской области «Средняя школа «Провинциальный колледж»

Исследовательское направление – информатика и ИКТ

Прогноз продаж торговой точки на основе методов машинного обучения

Курсовая работа

Выполнена ученицей
11 информационно-технологического класса
ГОУ ЯО Средняя школа «Провинциальный колледж»

Ситкиной Аленой

Научный руководитель –
учитель математики и информатики
ГОУ ЯО Средняя школа «Провинциальный колледж»

Легкова Мария Николаевна

Ярославль, 2021

Оглавление

Введение

Каждая организация стремится к оптимизации процесса закупок. В идеале, товар всегда должен быть в наличии, при этом его не должно быть в избытке. Прогнозирование того, сколько единиц каждого вида товара будет продано за определенный промежуток времени, очень важно для организации бизнеса. Анализ и прогноз продаж необходимы для корректных заказов товара, для уменьшения количества просроченного и списанного товара, для расширения и сбалансирования ассортимента торговой точки.

Актуальность исследования: отчасти алгоритм прогнозирования продаж торговой точки реализован в таких программах, как СБИС++ и 1С. Но, например, предприятия малого бизнеса зачастую не имеют доступа к полному пакету возможностей данных программ ввиду, во-первых, их стоимости, во-вторых, необходимости обращаться к услугам технических специалистов для их обслуживания. Поэтому прогноз прибыли предприятия зачастую осуществляется вручную. А это отнимает значительное количество времени, что влечет за собой дополнительные расходы в виде трудочасов (зарплаты). К тому же данный прогноз, во-первых, не является оперативным, во-вторых, из-за большого количества данных и медленного выполнения прогноза к моменту получения окончательных результатов он зачастую становится уже невостребованным.

Проект, разработанный в рамках данной курсовой работы, направлен на автоматизацию процесса прогнозирования продаж. Он позволит сэкономить рабочее время сотрудников, а также наиболее эффективно составить ассортимент, что приведет к привлечению новых покупателей и увеличению объема потребительской корзины существующих покупателей. Как итог, это позволит наиболее эффективно использовать оборотные средства предприятия, что повлечет за собой увеличение доходности торговой точки.

В работе мы будем опираться на данные торговой точки, расположенной в поселке Прибрежный, экспортированные в файл с расширением .xls из складской программы «1С: Торговое предприятие».

Цель исследования: разработать программу, прогнозирующую продажи торговой точки на основе технологии машинного обучения.

Задачи исследования:

1. Исследовать основные понятия, используемые в машинном обучении.
2. Исследовать принцип работы различных моделей.
3. Исследовать возможности языка Python и его дополнительных библиотек.
4. Сравнить качество работы различных моделей на исходных данных.

5. Создать графический интерфейс программы, осуществляющей прогноз продаж выбранной торговой точки.

Объект исследования: машинное обучение.

Предмет исследования: прогнозирование продаж сети торговых точек на основе машинного обучения.

Методы исследования:

- Анализ (необходимо для выявления принципов работы методов машинного обучения, принципов устройства различных функций потерь и способов оценки обобщающей способности)
- Обобщение (обобщение проделанной нами работы, подведение итогов)
- Синтез (использование и сочетание методов машинного обучения, функций потерь и различных инструментов в создаваемой нами программе)
- Эмпирический анализ (исследование информации о продажах выбранной торговой точки)
- Аналогия (используется при построении модели – закономерности, выявленные в обучающей выборке, будут применены к тестовым данным)
- Эксперимент (используется при тестировании написанного кода)
- Описание (описание принципа работы модели линейной регрессии)
- Моделирование (моделирование реальной ситуации – продаж торговой точки)
- Индукция (на основе определенной ограниченной выборки делаем выводы об общих закономерностях)
- Сравнение (сравнение моделей машинного обучения)

Обзор литературы и основных идей исследования:

Теоретический материал нашей работы построен на двух основных источниках – это книга Себастьяна Рашки и Вахида Мирджалили «Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow 2» и обучающий курс от НИУ ВШЭ на портале openedu.ru. В работе нами произведен обзор основных понятий (типы задач в машинном обучении, вопросы, связанные с предобработкой данных, переобучение и недообучение, функционал ошибки и кросс-валидация), а также принципов работы основных алгоритмов машинного обучения (kNN, линейная регрессия, решающие деревья, композиции алгоритмов). Для написания практической части нашей работы, а также глав, посвященных этому, мы обращались к документациям языка Python и его библиотек (NumPy, Seaborn, CatBoost, LightGBM, Pandas, Scikit-learn). На основе изученных идей, принципов и методов была реализована программа, прогнозирующая продажи выбранной торговой точки.

Характеристика степени изученности темы:

Основные идеи и принципы работы моделей машинного обучения на данный момент изучены очень хорошо. И, как известно, машинное обучение находит широкое применение в сфере торговли. Но при этом предприятия малого бизнеса, ввиду описанных выше причин, зачастую не имеют доступа к программам, в которых реализованы необходимые алгоритмы машинного обучения.

Характеристика личного вклада автора в решение избранной проблемы и практическая значимость исследования:

В ходе данной работы нами была реализована программа, решающая задачу прогнозирования продаж торговой точки. Это решает первоначально стоящую проблему доступа предприятий малого бизнеса к данным инструментам.

Глава 1. Основные понятия

1.1. Развитие машинного обучения

Как пишут Себастьян Рашка и Вахид Мирджалили в книге «Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow 2», при нынешней зрелости современных технологий есть один ресурс, которого у нас вдоволь: значительный объем структурированных и неструктурированных данных. Во второй половине двадцатого века машинное обучение развивалось как подобласть искусственного интеллекта, включающая самообучающиеся алгоритмы, которые выводили знания из данных с целью вырабатывания прогнозов. Вместо того чтобы требовать от людей выводить правила вручную и строить модели путем анализа крупных объемов данных, машинное обучение предлагает более эффективную альтернативу для сбора знаний из данных, которая постепенно улучшает эффективность прогнозирующих моделей и принимает решения, управляемые данными. Важность машинного обучения возрастает не только в исследованиях, имеющих отношение к компьютерным наукам; его роль в нашей повседневной жизни становится еще больше. Машинное обучение позволяет нам пользоваться надежными фильтрами почтового спама, удобным программным обеспечением распознавания текста и речи и испытанными поисковыми механизмами. Кроме того, заметный прогресс был достигнут в медицинских приложениях.

1.2. Основные типы машинного обучения

Данный раздел написан по материалам книги С.Рашки и В.Мирджалили «Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow 2», страницы 31-38.

Теперь введем некоторую терминологию. Свойства объектов будем называть «признаками» (features), а переменная, значение которой мы учимся предсказывать – «целевая переменная» (target). Основные виды признаков (и целевых переменных): бинарные, категориальные, числовые.

Модели (методы) машинного обучения используются для решения множества самых разных задач. Опишем основные из них.

1. Обучение с учителем

При обучении с учителем у нас есть размеченные обучающие данные. Цель - обучить модель с их использованием, чтобы она могла предсказывать значение целевой переменной для не встречавшихся ей ранее объектов.

Классификация – это подкатегория обучения с учителем, где целевой переменной является категориальная метка класса нового объекта. Метки представляют собой дискретные неупорядоченные значения.

Регрессия (регрессионный анализ) – это подкатегория обучения с учителем, в которой модель прогнозирует непрерывный результат.

2. Обучение без учителя

В данном случае мы работаем с неразмеченными данными; целью является поиск неизвестной структуры в исходных данных.

Кластеризация – это методика анализа данных, целью которой является разбиение объектов на подгруппы (кластеры). Объекты из одного кластера при этом должны обладать определенной степенью подобия, но при этом они должны быть менее похожи на объекты других кластеров.

Понижение размерности – это подобласть обучения без учителя, целью которой является устранение из данных шума и сжатие данных в подпространство меньшей размерности с сохранением большей части существенной информации.

3. Обучение с подкреплением

Тип машинного обучения, целью которого является разработка системы (агента), улучшающей свои характеристики на основе взаимодействия со средой.

Так как мы работаем с задачей обучения с учителем (говоря более конкретно, с задачей регрессионного анализа), далее мы будем рассматривать именно ее.

1.3. Предобработка данных

Перед тем, как обучать нашу модель на данных, необходимо произвести их предобработку. Перечислим основные причины, почему это стоит делать.

- Поступившие к нам данные могут иметь форму, непригодную для дальнейшей работы;
- В данных могут быть пропуски. Не все модели поддерживают работу с данными с пропусками (например, дерево решений не поддерживает);
- В данных могут быть выбросы (объекты, имеющие значение целевой переменной, резко не соответствующее общей закономерности в данных); соответственно, нужно найти их (либо удостовериться, что их нет) и либо устранить, либо выбрать модель, устойчивую к ним;

- В данных может быть избыточная информация; соответственно, следует производить отбор признаков, прежде чем обучать на этих данных модель;
- Иногда бывает необходима стандартизация[1] данных для достижения наибольшей эффективности модели;
- Некоторые модели (например, линейная регрессия) не поддерживают работу с категориальными признаками, которые обычно встречаются практически в любом датасете. В таких случаях обычно используется one-hot кодирование: вместо одного признака мы формируем несколько (количество соответствует количеству уникальных значений в исходной колонке), один из них равен 1 (соответствует значению, которое было записано в исходном признаке), а остальные равны 0.

1.4. Переобучение и недообучение

Переобучение – это очень распространенная проблема в машинном обучении. Она возникает тогда, когда модель показывает хороший результат на тренировочной выборке, но плохо обобщается на тестовую.[2] При недообучении, напротив, модель недостаточно сложна для выявления структуры в данных.[3] Эти проблемы решаются разными способами в зависимости от того, какую модель машинного обучения мы используем. Переобучение и недообучение, а также способы решения этих проблем будут подробнее рассмотрены в последующих главах при описании используемых нами моделей.

1.5. Функционал ошибки и кросс-валидация

При создании модели машинного обучения важно понимать, насколько хорошо она предсказывает целевую переменную. Функционал ошибки – это своего рода мера эффективности модели. Он выбирается в зависимости от решаемой задачи.

Заметим, что ошибка модели складывается из трех компонент: шум (noise), смещение (bias) и разброс (variance). Шум – это характеристика сложности и противоречивости данных. Он является свойством решаемой задачи. Смещение модели – это её способность приблизить лучшую среди всех возможных моделей. Разброс модели характеризует её

[1] *Стандартизация – процесс придания данным характеристик стандартного нормального распределения: нулевого среднего и единичной дисперсии. (источник – тот же, что указан ниже,* С. 75-76)

[2] *Рашка С., Мирджалили В. Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow 2/ Пер. с англ. СПб.: ООО "Диалектика", 2020. С.109*

[3] *Там же.*

устойчивость к изменениям в обучающей выборке. Данная концепция пригодится нам в дальнейшем при изучении некоторых моделей.[4]

Для оценки качества модели нам необходимо разбить выборку на две части: тренировочную и тестовую. Но у такого подхода есть некоторые недостатки. При таком подходе мы подбираем гиперпараметры[5] модели так, чтобы она показывала максимально хороший результат на тестовой выборке. То есть мы «подгоняем» гиперпараметры под тестовую выборку. Это значит, модель переобучается. Кросс-валидация – это способ, позволяющий бороться с данной проблемой. Мы по-прежнему будем разбивать выборку на тренировочную и тестовую, но при обучении тренировочная выборка разбивается на k блоков (обычно k=5) и каждый по очереди делается тестовым (правильнее сказать, валидирующим), остальные при этом формируют обучающую выборку. Так мы подбираем наилучшие гиперпараметры. Далее тестируем модель на тестовой (отложенной вначале) выборке.

1.6. Визуализация

Визуализация занимает очень важное место в машинном обучении. Зачастую она помогает увидеть новые закономерности в данных, обосновать введение нового признака. Также визуализировать можно модель и результаты ее работы. Это может подсказать нам, например, что модель переобучена. В языке Python существуют специальные библиотеки, предназначенные для визуализации: matplotlib и Seaborn, они будут подробнее рассмотрены в Главе 3.

В последующих главах описывается принцип работы, а также некоторые особенности разных моделей машинного обучения. Данные главы написаны на основе курса «Основы машинного обучения» от НИУ ВШЭ.[6]

[4] *Соколов Е.А., Зимовнов А.В., Ковалев Е.И., Кохтев В.М., Рысьмятова А.А., Филатов А.А. Основы машинного обучения: [Электронный ресурс]// Образовательная платформа «Открытое образование». URL: https://courses.openedu.ru/courses/course-v1:hse+INTRML+fall_2020/course/ (Дата обращения: 04.11.2021).*

[5] *Гиперпараметры – это параметры модели, которые подбираются человеком вручную.*

[6] *Там же, что и 4.*

Глава 2. Основные модели машинного обучения

2.1. Линейная регрессия и градиентный спуск

2.1.1. Общее описание, функционал ошибки

Так как мы решаем задачу регрессии (предсказываем количество единиц товара, которое будет продано), мы будем использовать модель, которая чаще всего применяется для данного типа задач: линейную регрессию.

Итак, пусть мы рассматриваем объект x, который обладает d признаками. Тогда наша модель $a(x)$ будет иметь следующий вид: $a(x) = w_0 + w_1x_1 + \cdots + w_dx_d$, где w_i - вес i-ого признака. Заметим, что данную формулу можно переписать в виде скалярного произведения векторов: $a(x) = w_0 + \langle w, x\rangle$. В общем случае можно считать, что у нас есть признак, который всегда равен единице, тогда формула переписывается в виде $a(x) = \langle w, x\rangle$.

Теперь опишем решаемую нами задачу. Для этого введем функционал ошибки. Мы будем использовать один из самых распространенных – MSE (mean squared error – среднеквадратичная ошибка). Она также является универсальной и может применяться при использовании практически любого алгоритма машинного обучения. Так как одной из наших задач является сравнение различных моделей, нами была выбрана именно эта функция потерь. Вычисляется она следующим образом: $\frac{1}{l}\sum_{i=1}^{l}(\langle w, x^i\rangle - y_i)^2$, где l – количество объектов в обучающей выборке. Тогда наша задача записывается следующим образом: $\frac{1}{l}\sum_{i=1}^{l}(\langle w, x^i\rangle - y_i)^2 \to min_w$.

2.1.2. Запись задачи в матричной форме

Для удобства запишем нашу задачу в матричной форме. Введем матрицу объекты-признаки. Она содержит в себе всю обучающую выборку и записывается следующим образом: $X = \begin{pmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{l1} & \cdots & x_{ld} \end{pmatrix} \in \mathbb{R}^{l\times d}$, где строки соответствуют объектам, а столбцы – признакам (считаем, что все они числовые или булевые). Также введем вектор весов $w = \begin{pmatrix} w_1 \\ \vdots \\ w_d \end{pmatrix} \in \mathbb{R}^d$ и вектор ответов (значений целевой переменной) для обучающей выборки $y = \begin{pmatrix} y_1 \\ \vdots \\ y_l \end{pmatrix} \in$

$\mathbb{R}^l$. Теперь заметим, что мы можем умножить матрицу X на вектор w (данная операция допустима). Сделаем это: $Xw = \begin{pmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{l1} & \cdots & x_{ld} \end{pmatrix} * \begin{pmatrix} w_1 \\ \vdots \\ w_d \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{d} w_i x_{1i} \\ \vdots \\ \sum_{i=1}^{d} w_i x_{li} \end{pmatrix} = \begin{pmatrix} \langle w, x_1 \rangle \\ \vdots \\ \langle w, x_l \rangle \end{pmatrix} \in \mathbb{R}^l$.

Заметим, что мы получили вектор предсказаний нашей модели для объектов обучающей выборки. Теперь получим вектор отклонений предсказаний модели от ответов: $Xw - y = \begin{pmatrix} \langle w, x_1 \rangle - y_1 \\ \vdots \\ \langle w, x_l \rangle - y_l \end{pmatrix}$. Вспомним такое понятие из линейной алгебры, как евклидова норма. Для произвольного вектора z она записывается следующим образом: $\|z\|^2 = \sum_{j=1}^{n} z_j^2$. Посчитаем евклидову норму для нашего вектора отклонений, умножим ее на $\frac{1}{l}$ и заметим, что мы как раз получим формулу для среднеквадратичной ошибки: $\frac{1}{l}\|Xw - y\|^2 = \frac{1}{l}\sum_{i=1}^{l}(\langle w, x_i \rangle - y_i)^2$. Таким образом, мы записали нашу задачу в матричной форме.

2.1.3. Решение задачи

Чтобы решить поставленную задачу, введем понятие градиента. Градиент – это вектор частных производных функции (обобщение понятия производной на функции нескольких переменных).

Нашей задачей является нахождение минимума для функции MSE. Она является строго выпуклой, следовательно, экстремум у неё ровно один – это и есть искомый глобальный минимум. При этом, как известно из высшей математики, в точке экстремума градиент функции равен 0. Это значит, что для получения ответа нам необходимо посчитать градиент MSE, приравнять его к нулю и решить полученную систему линейных уравнений. Проделаем эти операции:

$$Q(a, X) = \frac{1}{l}\sum_{i=1}^{l}\left(y_i - a(x_i)\right)^2 \to min_w$$

$$Q(a, X) = \frac{1}{l}\sum_{i=1}^{l}(y_i - \langle w, x_i \rangle)^2 \to min_w$$

$$\nabla_w Q = \begin{pmatrix} \frac{dQ}{dw_1} & \cdots & \frac{dQ}{dw_d} \end{pmatrix}$$

$$\frac{dQ}{dw_j} = \frac{1}{l}\sum_{i=1}^{l} \frac{d(y_i^2 - 2y_i\langle w, x_i\rangle + (\langle w, x_i\rangle)^2)}{dw_j}$$

$$= \frac{1}{l}\sum_{i=1}^{l} \frac{dy_i^2}{dw_j} - \frac{2 * dy_i(w_1x_{i1} + \cdots + w_dx_{id})}{dw_j} + \frac{d(w_1x_{i1} + \cdots)(w_1x_{i1} + \cdots)}{dw_j}$$

$$= \frac{1}{l}\sum_{i=1}^{l} 0 - 2y_ix_{ij} + 2x_{ij}\langle w, x_i\rangle = \frac{2}{l}\sum_{i=1}^{l} x_{ij}(\langle w, x_i\rangle - y_i)$$

Перепишем полученный результат в матричном виде:

$$\nabla\frac{1}{l}\|Xw - y\|^2 = \frac{2}{l}X^T(Xw - y)$$

Приравняем к нулю и решим полученную систему уравнений:

$$\frac{2}{l}X^T(Xw - y) = 0$$

$$w = (X^TX)^{-1}X^Ty$$

Итак, мы в явном виде получили решение нашей задачи.

2.1.4. Градиентный спуск

Полученное нами выше решение имеет ряд существенных недостатков, как-то:

- Если матрица X^TX – вырожденная, то обратной к ней не существует, и решение по данной формуле мы получить не сможем;
- Если матрица X^TX не является вырожденной, но является почти вырожденной (определитель близок к нулю) будут проблемы с вычислением обратной матрицы;
- Нахождение обратной матрицы к матрице размера $n \times n$ выполняется за $O(n^3)$ операций; для больших n эти вычисления будут производиться очень долго;
- Решение в аналитическом виде существует не для всех возможных функций потерь; мы решили задачу только для одной выбранной нами функции.

Данные проблемы решаются применением градиентного спуска. Опишем алгоритм его работы.

Для начал введем понятие антиградиента. Антиградиент – это градиент, взятый с противоположным знаком. Он указывает на направление наискорейшего убывания функции.

Алгоритм градиентного спуска выглядит следующим образом:

1. Стартуем из случайной точки w^0 – это инициализация весов (может быть взята, например, из стандартного нормального распределения);
2. Делаем шаг спуска (сдвигаемся по антиградиенту): $w^t = w^{t-1} - \eta\nabla Q(w^{t-1})$, где η – размер шага;

3. Повторяем пункт 2, пока не окажемся достаточно близко к искомой точке. Можно рассматривать различные варианты определения момента останова, вот основные из них:
   - $||w^t - w^{t-1}|| < \varepsilon$
   - $||\nabla Q(w^t)|| < \varepsilon$

Также важно сделать несколько замечаний:

- Рассматриваемая функция потерь может, помимо глобального минимума, иметь несколько локальных минимумов. Чтобы получить корректный результат, используется мультистарт (мы запускаем алгоритм несколько раз, каждый раз выбирая разные стартовые точки);
- Длина шага является гиперпараметром: если шаг слишком большой, это может помешать сходимости алгоритма, а если слишком маленький, алгоритм может выполняться долго;
- Также помогает сходимости и ускоряет процесс масштабирование признаков;
- Из-за того, что данный алгоритм может работать долго (особенно на достаточно больших выборках), часто используют его модификацию – стохастический градиентный спуск. Идея заключается в том, что на каждом шаге мы берем градиент по нескольким случайно выбранным объектам, а не по всей выборке. Но стохастический градиентный спуск бывает нестабильным, при его использовании нужно аккуратно выбирать длину шага, иначе алгоритм может не сойтись.

Теперь мы можем записать, используя результаты, полученные ранее, как будет выглядеть формула для шага градиентного спуска, если мы используем MSE в качестве функции потерь. Она пригодится нам при реализации данного алгоритма:

$$w^t = w^{t-1} - \eta_t \frac{2}{l} X^T (X w^{t-1} - y)$$

2.1.5. Переобучение, регуляризация, виды линейной регрессии

1. Гребневая регрессия

Известно, что одним из самых явных признаков переобучения модели линейной регрессии являются большие веса. Тогда можно рассмотреть следующую идею: будем штрафовать модель за большие веса (регуляризировать модель). Такая модель называется гребневой регрессией (используемый здесь регуляризатор ещё иногда называют L2-нормой).

Теперь приведем строгую запись данной идеи:

$$\frac{1}{l}\sum_{i=1}^{l}(\langle w, x_i\rangle - y_i)^2 + \lambda\|w\|^2 \to min_w$$

Здесь λ – коэффициент регуляризации, он является гиперпараметром модели. Используя идеи, приведенные выше (можно показать, что данный функционал также строго выпуклый), решим поставленную задачу для регуляризованного функционала:

$$w = (X^T X + \lambda I)^{-1} X^T y$$

Здесь I – единичная матрица.[7] Можно заметить, что добавление слагаемого λI зачастую будет даже из вырожденной матрицы делать невырожденную, и мы сможем найти решение задачи, используя аналитически выведенную формулу.

Осталось сделать лишь последнее замечание: в регуляризатор нельзя включать свободный коэффициент, так как он нужен для того, чтобы порядок предсказываемых значений был сопоставим с порядком целевой переменной; если мы потребуем все веса быть небольшими, а целевая переменная будет достаточно большой, модель не сможет качественно решить задачу.

2. LASSO

LASSO (Least Absolute Shrinkage and Selection Operator – в дословном переводе «оператор наименьшей абсолютной усадки и выбора»; используемый здесь регуляризатор ещё иногда называют L1-нормой) – это метод, идея которого заключается в том, что при регуляризации идет суммирование не квадратов весов, а модулей весов.

Приведем строгую запись данной идеи:

$$\frac{1}{l}\sum_{i=1}^{l}(\langle w, x_i\rangle - y_i)^2 + \lambda\sum_{j=1}^{d}|w_i| \to min_w$$

Можно заметить, что при данном подходе некоторые веса обнуляются, что приводит к отбору признаков.

Итак, мы рассмотрели первую модель машинного обучения. Но можно заметить, что линейная регрессия будет показывать не самые лучшие результаты, если зависимость в данных не является линейной. Можно обучать модель с полиномиальными признаками, но тогда задача обучения модели становится на порядки сложнее и данных нужно на порядки больше. Поэтому необходимо рассмотреть и другие модели.

2.2. kNN

Принцип работы метода k ближайших соседей (k nearest neighbors) (напомним, мы разбираем только случай задачи регрессии) основан на гипотезе непрерывности. Согласно

[7] *Единичная матрица – это матрица с единицами на главной диагонали и нулями на остальных позициях.*

этой гипотезе, мы предполагаем, что близким объектам соответствуют близкие ответы. Обучение данной модели состоит в запоминании обучающей выборки.

Опишем алгоритм работы данной модели:

1. Пусть нам дан объект x. Для начала мы сортируем объекты обучающей выборки по расстоянию до объекта x: $\rho(x, x_1) \leq \rho(x, x_2) \leq \cdots \leq \rho(x, x_l)$ (здесь l – это размер обучающей выборки, ρ – некоторая функция для вычисления расстояния между объектами (метрика));
2. Выбираем k ближайших соседей (здесь k – гиперпараметр);
3. Усредняем значения целевой переменной по выбранным ближайшим соседям: $a(x) = \frac{1}{k}\sum_{i=1}^{k} y_i$.

Теперь остановимся подробнее на том, что такое метрика и какими бывают метрики. Метрика – это обобщение понятия “расстояние” на многомерные пространства. Метрикой называют функцию, принимающую два аргумента и удовлетворяющую следующим требованиям:

- $\rho(x, z) = 0 \Leftrightarrow x = z$;
- $\rho(x, z) = \rho(z, x)$;
- $\rho(x, z) \leq \rho(x, v) + \rho(v, z)$ $(\forall v)$ – неравенство треугольника.

Приведем несколько примеров самых распространенных метрик (для числовых признаков):

- Евклидова метрика: $\rho(x, z) = \sqrt{\sum_{j=1}^{d}(x_j - z_j)^2}$;
- Манхэттенская метрика: $\rho(x, z) = \sum_{j=1}^{d}|x_j - z_j|$ (полезна в случаях, когда в данных много выбросов):
- Метрика Минковкого: $\rho(x, z) = \sqrt[t]{\sum_{j=1}^{d}(x_j - z_j)^t}$ (здесь t – гиперпараметр).

Как можно заметить, для категориальных признаков эти формулы неприменимы. Существует много способов измерить расстояние между объектами, основываясь на категориальных признаках, но их рассмотрение выходит за рамки нашей курсовой работы. Как уже было сказано, мы можем применить one-hot кодирование и считать категориальные признаки числовыми.

Также на практике часто используют kNN с весами: более близкие объекты больше влияют на результат (имеют больший вес). Для расчета весов используют парзеновское окно: $w_i = K(\frac{\rho(x,x_i)}{h})$, где K(z) – ядро (функция, которая не возрастает и положительна на отрезке [0;1]), h – ширина окна (гиперпараметр; он позволяет регулировать, с какого расстояния объекты практически перестают учитываться). Приведем некоторые примеры:

- Гауссовское ядро: $\mathrm{K}(z) = (2\pi)^{-0.5}\exp\left(-\frac{1}{2}z^2\right)$;
- $w_i = 1/\rho(x, x_i)$ (используется в библиотеке sklearn).

Таким образом, формула для расчета значения нового объекта x будет выглядеть следующим образом (её называют формулой Надарая-Ватсона):

$$a(x) = \frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i}$$

Можно заметить, что модель kNN показывает хорошие результаты, если обучающая выборка довольно большая. Также модель легко обучается. Но у неё есть несколько определенных недостатков, а именно: модель может работать достаточно долго (осуществляя поиск k ближайших соседей) и подбор гиперпараметров для неё довольно трудоемок. Поэтому необходимо также рассмотреть и другие модели.

2.3. Решающие деревья

Как понятно из названия модели, она основывается на построении графа. В его внутренних вершинах записываются предикаты, в листьях – прогнозы модели. Предикаты могут быть сколь угодно сложными (они могут содержать линейные модели, деревья и т. д.), но, как мы увидим позже, даже с использованием простейших предикатов (пороговых) можно построить очень хорошие (и, соответственно, сложные) модели. Также, модели, использующие сложные предикаты, сложны в обучении. Поэтому далее мы будем рассматривать только простейшие предикаты.

Опишем алгоритм построения дерева. Дерево строится жадно: на каждом шаге мы определяем наилучший предикат и разбиваем по нему обучающую выборку. Так мы продолжаем действовать до тех пор, пока не достигаем критерия останова. Его можно выбрать различными способами: можно ограничивать глубину дерева, количество листьев в нем, число объектов в листовой вершине и т. д. Обычно, выбор критерия останова не оказывает значительного влияния на качество модели (в отличие от величины ограничения, являющейся гиперпараметром). Далее прогнозы в листовых вершинах определяются как среднее арифметическое значений целевой переменной объектов обучающей выборки, попавших в данную вершину: $c_v = \frac{1}{|R_v|}\sum_{(x_i,y_i)\in R_v} y_i$ (здесь мы вводим некоторые обозначения, которыми будем пользоваться в дальнейшем: R_v – это множество объектов в вершине v, c_v – прогноз модели для объектов листа v).

Теперь посмотрим, как определять наилучший предикат для каждой из вершин. Пусть на данный момент в рассматриваемой вершине содержится множество объектов R. Хаотичностью вершины будем называть дисперсию ответов в ней:

$H(R) = \frac{1}{|R|}\sum_{(x_i,y_i)\in R_v}(y_i - y_R)^2$, где $y_R = \frac{1}{|R|}\sum_{(x_i,y_i)\in R} y_i$ – среднее арифметическое значений целевой переменной объектов множества R. Пусть каждый предикат разбивает множество R на два других множества – R_l и R_r (здесь l и r – индексы дочерних вершин). Тогда наилучшим будем считать предикат, который минимизирует взвешенную сумму хаотичностей в двух итоговых вершинах R_l и R_r: $Q(R,j,t) = \frac{|R_l|}{|R|}H(R_l) + \frac{|R_r|}{|R|}H(R_r) \to \min\limits_{j,t}$, где j – признак, t – порог разбиения.

Итак, мы рассмотрели алгоритм построения решающего дерева. Но у данной модели есть существенный недостаток – она очень неустойчива. То есть при незначительном изменении обучающей выборки итоговая модель может довольно сильно меняться. Для устранения данного недостатка используют композиции моделей, которые будут подробно рассмотрены в следующий главах.

2.4. Случайный лес

Для начала рассмотрим общие идеи, которые используют композиции моделей. Пусть у нас есть N базовых моделей $b_1(x), b_2(x)$ ... $b_N(x)$ (при этом к ним предъявляется формальное требование: вероятность их ошибки должна быть ниже 50%). Тогда прогноз композиции строится как среднее арифметическое прогнозов базовых моделей. При этом можно рассматривать разные принципы построения базовых моделей. Мы рассмотрим два основных варианта. Первый заключается в том, что мы будем обучать модели независимо, на разных подвыборках. Второй заключается в последовательном обучении моделей, причем строятся они так, чтобы каждая корректировала ошибки предыдущей. Первая идея лежит в основе принципа работы бэггинга и случайного леса и будет рассмотрена в этой главе, вторая же является основой бустинга и рассматривается в соответствующей главе.

Принцип работы бэггинга (bagging – bootstrap aggregating) основан на бутстрэпе. Метод бутстрэпа заключается в следующем. Пусть имеется выборка размера k. Возьмем из выборки k объектов с возвращением. Это означает, что мы будем k раз выбирать произвольный объект выборки (считаем, что каждый объект «достается» с одинаковой вероятностью), причем каждый раз мы выбираем из всех исходных объектов. Отметим, что из-за возвращения среди выбранных объектов окажутся повторы. Далее мы обучаем базовые модели на подвыборках, построенных бутстрэпом.[8]

[8] *Открытый курс машинного обучения. Тема 5. Композиции: бэггинг, случайный лес: [Электронный ресурс]// IT-сообщество «Хабр». URL: https://habr.com/ru/company/ods/blog/324402/ (Дата обращения: 04.11.2021).*

Заметим, что композиция, построенная таким образом, будет иметь такое же смещение, как и базовые модели. При этом её разброс зависит от ковариации базовых моделей (ковариация – это мера линейной зависимости двух случайных величин)[9] и вычисляется следующим образом:

$$a(x) = \frac{\text{разброс}(b_n(x))}{n} + \text{ковариация}(b_n(x), b_m(x))$$

Итак, опишем алгоритм работы случайного леса. Для построения каждого из N деревьев с помощью бутстрэпа генерируется подвыборка и на ней обучается решающее дерево (по алгоритму, описанному ранее). При этом для уменьшения ковариации деревьев используется следующий прием: выбор оптимального предиката в очередной вершине очередного дерева осуществляется не по всем признакам, а по некоторому подмножеству множества признаков, которое генерируется отдельно для каждой вершины (рекомендуемый размер для этого подмножества – треть от исходного количества признаков). Для получения итогового прогноза модели необходимо усреднить прогнозы всех N деревьев.

Можно заметить, что с ростом числа деревьев N ошибка случайного леса выходит на асимптоту и не растет. Поэтому случайный лес практически не содержит гиперпараметров, в следствие чего его часто называют универсальным алгоритмом машинного обучения. Но у бэггинга есть ряд проблем. Если базовая модель окажется смещенной, то и композиция в итоге окажется смещенной. Обучение и применение базовых моделей происходит довольно долго, также необходим большой объем памяти для хранения этих моделей.

2.5. Градиентный бустинг

Идея, как уже было сказано ранее, заключается в том, чтобы строить модели, составляющие композицию, так, чтобы каждая следующая максимально корректировала ошибки уже построенных моделей. Обычно в бустинге используются простые базовые модели (как мы увидим в дальнейшем, даже с помощью простых базовых моделей можно добиться хороших результатов). Итак, опишем жадный алгоритм построения данной композиции моделей:

1. Пусть N – число моделей в композиции, $b_1(x), b_2(x) \dots b_N(x)$ – сами модели. Итоговым прогнозом композиции моделей будем считать сумму прогнозов базовых моделей: $a_N(x) = \sum_{n=1}^{N} b_n(x)$. Первую модель мы уже умеем строить: $\frac{1}{l}\sum_{i=1}^{l} L(y_i, b_1(x_i)) \to \min\limits_{b_1(x)}$ (здесь L – функция потерь);

[9] *Ковариация – Википедия: [Электронный ресурс]// Википедия. URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D1%86%D0%B8%D1%8F (Дата обращения: 04.11.2021).*

2. Пусть мы уже построили *n-1* модель, тогда модель с номером *n* должна строиться следующим образом: $\frac{1}{l}\sum_{i=1}^{l} L(y_i, a_{n-1}(x_i) + b_n(x_i)) \to \min\limits_{b_n(x)}$. Но если используются недифференцируемые базовые модели, то пока мы не умеем обучать их на такой функционал;
3. Возьмем антипроизводную по второму аргументу функции потерь в точке, равной текущему значению прогноза композиции: $s_i^n = -\frac{dL(y_i,z)}{dz}|_{z=a_{n-1}(x_i)}$ (эти величины называют сдвигами). Тогда будем обучать новую базовую модель следующим образом: : $\frac{1}{l}\sum_{i=1}^{l}(b_n(x_i) - s_i^n)^2 \to \min\limits_{b_n(x)}$.

Теперь применим данный алгоритм для MSE (при подсчете сдвигов для простоты мы домножим производную на 0,5):

Сдвиги: $s_i^n = -\frac{1}{2} * \frac{dL(y_i,z)}{dz}|_{z=a_{n-1}(x_i)} = -\frac{d}{dz}\frac{1}{2}(z - y_i)^2|_{z=a_{n-1}(x_i)} = y_i - a_{n-1}(x_i)$;

Итог: $\frac{1}{l}\sum_{i=1}^{l}(b_n(x_i) - (y_i - a_{n-1}(x_i)))^2 \to \min\limits_{b_n(x)}$.

Сделаем несколько замечаний, касающихся бустинга. Можно заметить, что при таком способе построения композиции моделей их смещение уменьшается, а разброс может увеличиваться. Поэтому в качестве базовых моделей следует брать модели, имеющие небольшой разброс, например, неглубокие решающие деревья. Гиперпараметрами данной композиции являются глубина базовых деревьев (можно считать, что глубиной мы регулируем их сложность, но в общем случае тут могут быть и другие варианты) и число деревьев (причем, в отличие от случайного леса, бустинг переобучается, если деревьев слишком много). Но у бустинга может возникнуть следующая проблема: вследствие своей простоты, базовые модели могут не справиться с приближением направления, указанного сдвигами. В качестве решения можно использовать регуляризацию: $a_n(x) = a_{n-1}(x) + \eta b_n(x)$, где $\eta \in (0; 1]$ – длина шага (чем меньше длина шага, тем больше деревьев потребуется для построения качественной композиции). Также для улучшения качества композиции можно для очередной базовой модели использовать рандомизацию по признакам (то есть обучать модель с использованием некоторого случайного подмножества признаков) и объектам.

Глава 3. Используемые инструменты

3.1. Язык программирования Python

Python – это высокоуровневый язык программирования, который на данный момент является одним из самых популярных в мире. Он предоставляет эффективные высокоуровневые структуры данных, а также простой, но эффективный подход к объектно-ориентированному программированию. Его простой синтаксис и динамическая типизация наряду с тем, что он является легко читаемым, делают его идеальным языком для написания сценариев и быстрой разработки приложений в различных областях и на большинстве платформ.[10]

Также Python предоставляет множество инструментов для работы с данными и машинного обучения, из-за чего последнее время он стал одним из основных языков, используемых в этой области. По этой же причине он был выбран и нами.

Мы будем пользоваться последней версией Python 3, так как поддержка версии Python 2 недавно прекратилась.

3.2. Jupiter Notebook

Jupiter Notebook – это браузерный инструмент для интерактивного создания документов, сочетающих в себе пояснительный текст, математику, вычисления и их мультимедийный вывод.[11] Он является одним из самых популярных среди тех, кто занимается машинным обучением. За счет своей структуры он позволяет разбивать код на блоки и сразу запускать его, что обеспечивает быстрый и последовательный процесс разработки. Именно поэтому он был выбран как один из основных инструментов для выполнения работы.

3.3. Необходимые библиотеки

- Pandas

Pandas (обычно импортируется под названием pd) – это библиотека, используемая для анализа данных. Основными структурами данных являются Series (серия (колонка с

[10] *Документация языка Python. [Электронный ресурс]// Документация языка Python 3.10.0. URL: https://docs.python.org/3/ (Дата обращения: 04.11.2021).*

[11] *Описание Jupiter Notebook. [Электронный ресурс]// Документация Jupiter Notebook. URL: https://jupyter-notebook.readthedocs.io/en/latest/notebook.html (Дата обращения: 04.11.2021).*

данными)) и DataFrame (датафрейм (таблица с данными, состоит из серий)).[12] Это очень распространенная, продвинутая и быстроразвивающаяся библиотека, она предоставляет широкий набор инструментов для работы с данными (в основном с табличными), почему и была выбрана.

- NumPy

NumPy (обычно импортируется под названием np) – это библиотека, используемая для машинного обучения и научных вычислений. Она предоставляет широкий функционал для работы с многомерными массивами, в ней реализованы различные математические функции.[13] Поэтому она и была выбрана.

- Datetime

Datetime – это модуль, содержащий широкий и удобный функционал для работы с информацией о датах и времени. В работе нами были использованы атрибут x.day, в котором записан день (x – дата, принадлежит классу date) и метод x.weekday(), возвращающий день недели для даты x (число от 0 до 6 соответственно).[14]

- Matplotlib и seaborn

Matplotlib – это библиотека визуализации. На ее основе работают функции визуализации в библиотеке pandas и вся библиотека seaborn, являющаяся зачастую более удобной, чем сама matplotlib.

Seaborn (обычно импортируется под названием sns), как было сказано ранее, также библиотека визуализации, основанная на matplotlib, но более удобная, чем matplotlib, почему и была выбрана.[15]

- Scikit-learn (sklearn)

Scikit-learn (sklearn) – это библиотека, в которой реализованы основные алгоритмы машинного обучения. Она является очень распространенным, продвинутым и удобным инструментом для реализации программ, решающих задачи машинного обучения, почему и была выбрана. Из этой библиотеки нами были взяты классы моделей машинного обучения, функции метрик а также некоторые другие инструменты.[16]

---

[12] *Документация библиотеки Pandas. [Электронный ресурс]// Официальный сайт библиотеки Pandas. URL: https://pandas.pydata.org/docs/user_guide/index.html (Дата обращения: 04.11.2021).*

[13] *Документация библиотеки NumPy. [Электронный ресурс]// Официальный сайт библиотеки NumPy. URL: https://numpy.org/doc/stable/ (Дата обращения: 04.11.2021).*

[14] *Документация языка Python. [Электронный ресурс]// Документация языка Python 3.10.0. URL: https://docs.python.org/3/ (Дата обращения: 04.11.2021).*

[15] *Документация библиотеки Seaborn. [Электронный ресурс]// Официальный сайт библиотеки Seaborn. URL: https://seaborn.pydata.org (Дата обращения: 04.11.2021).*

[16] *Документация библиотеки Scikit-learn. [Электронный ресурс]// Официальный сайт библиотеки Scikit-learn. URL: https://www.sklearn.org/user_guide.html (Дата обращения: 04.11.2021).*

- LightGBM

LightGBM – это одна из самых популярных современных библиотек, в которых реализован градиентный бустинг. У неё есть ряд преимуществ: высокая скорость обучения, снижение использования памяти, улучшенная точность и т. д. Поэтому она и была выбрана нами.[17]

- CatBoost

CatBoost – это алгоритм для градиентного бустинга на деревьях решений. Он разработан исследователями и инженерами Яндекса. Его отличительными чертами являются: хорошее качество работы с гиперпараметрами, используемыми по умолчанию, повышенная точность и быстрое прогнозирование.[18] В этой библиотеке используется особый вид решающих деревьев – oblivious decision trees (симметричные решающие деревья). Главная их особенность заключается в том, что на каждом их уровне используется один и тот же предикат, что значительно ускоряет время обучения моделей.[19]

- tkinter

tkinter – это графическая библиотека, являющаяся стандартным интерфейсом Python для инструментария Tcl/Tk GUI.[20] Главное её преимущество заключается в том, что она является кроссплатформенной, почему и была выбрана нами.

---

[17] Документация библиотеки LightGBM. [Электронный ресурс] // Официальный сайт библиотеки LightGBM. URL: https://lightgbm.readthedocs.io/en/latest/index.html (Дата обращения: 04.11.2021).

[18] Документация библиотеки CatBoost. [Электронный ресурс] // Официальный сайт библиотеки CatBoost. URL: https://catboost.ai/ (Дата обращения: 04.11.2021).

[19] Соколов Е.А., Зимовнов А.В., Ковалев Е.И., Кохтев В.М., Рысьмятова А.А., Филатов А.А. Основы машинного обучения: [Электронный ресурс]// Образовательная платформа «Открытое образование». URL: https://courses.openedu.ru/courses/course-v1:hse+INTRML+fall_2020/course/ (Дата обращения: 04.11.2021).

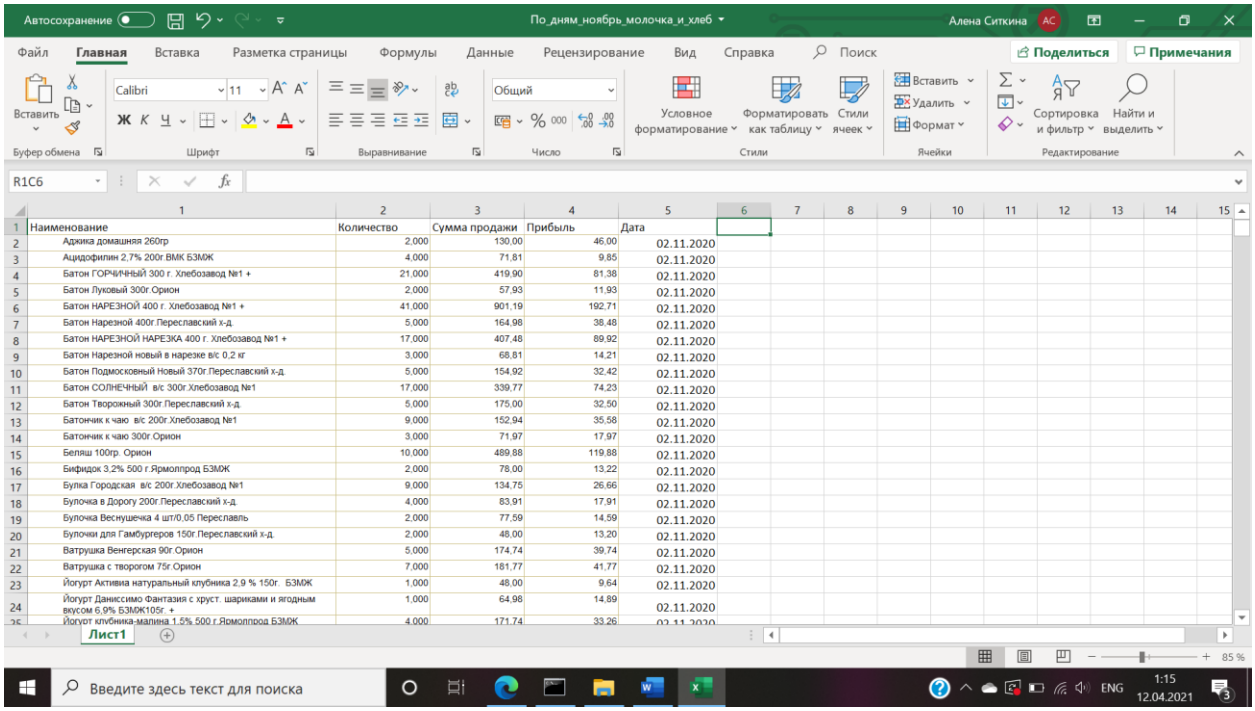
[20] Документация языка Python. [Электронный ресурс]// Документация языка Python 3.10.0. URL: https://docs.python.org/3/ (Дата обращения: 04.11.2021).

Глава 4. Реализация программы

В данном разделе в общем виде разбираются принципы работы нашей программы. Более подробные комментарии вместе с кодом программы находятся в Приложениях 1 и 2.

1. Исходные данные

После подключения необходимых библиотек происходит загрузка данных. Они записаны в файл с расширением .xls. В нем содержится информация о продажах товаров из категорий «Молочная продукция» и «Хлеб и хлебобулочные изделия» (взят несезонный товар, чтобы данная выборка была репрезентативна[21]) продуктового магазина в поселке Прибрежный за ноябрь 2020 года (см. Рис. 1).



Наименование	Количество	Сумма продажи	Прибыль	Дата
Аджика домашняя 260гр	2,000	130,00	46,00	02.11.2020
Ацидофилин 2,7% 200г ВМК БЗМЖ	4,000	71,81	9,85	02.11.2020
Батон ГОРЧИЧНЫЙ 300 г. Хлебозавод №1 +	21,000	419,90	81,38	02.11.2020
Батон Луковый 300г.Орион	2,000	57,93	11,93	02.11.2020
Батон НАРЕЗНОЙ 400 г. Хлебозавод №1 +	41,000	901,19	192,71	02.11.2020
Батон Нарезной 400г.Переславский х-д.	5,000	164,98	38,48	02.11.2020
Батон НАРЕЗНОЙ НАРЕЗКА 400 г. Хлебозавод №1 +	17,000	407,48	89,92	02.11.2020
Батон Нарезной новый в нарезке в/с 0,2 кг	3,000	68,81	14,21	02.11.2020
Батон Подмосковный Новый 370г.Переславский х-д.	5,000	154,92	32,42	02.11.2020
Батон СОЛНЕЧНЫЙ в/с 300г.Хлебозавод №1	17,000	339,77	74,23	02.11.2020
Батон Творожный 300г.Переславский х-д.	5,000	175,00	32,50	02.11.2020
Батончик к чаю в/с 200г.Хлебозавод №1	9,000	152,94	35,58	02.11.2020
Батончик к чаю 300г.Орион	3,000	71,97	17,97	02.11.2020
Беляш 100гр. Орион	10,000	489,88	119,88	02.11.2020
Бифидок 3,2% 500 г.Ярмолпрод БЗМЖ	2,000	78,00	13,22	02.11.2020
Булка Городская в/с 200г.Хлебозавод №1	9,000	134,75	26,66	02.11.2020
Булочка в Дорогу 200г.Переславский х-д.	4,000	83,91	17,91	02.11.2020
Булочка Веснушечка 4 шт/0,05 Переславль	2,000	77,59	14,59	02.11.2020
Булочки для Гамбургеров 150г.Переславский х-д.	2,000	48,00	13,20	02.11.2020
Ватрушка Венгерская 90г.Орион	5,000	174,74	39,74	02.11.2020
Ватрушка с творогом 75г.Орион	7,000	181,77	41,77	02.11.2020
Йогурт Активиа натуральный клубника 2,9 % 150г. БЗМЖ	1,000	48,00	9,64	02.11.2020
Йогурт Даниссимо Фантазия с хруст. шариками и ягодным вкусом 6,9% БЗМЖ105г. +	1,000	64,98	14,89	02.11.2020
Йогурт клубника-малина 1,5% 500 г.Ярмолпрод БЗМЖ	4,000	171,74	33,26	02.11.2020

Рис. 1 Исходные данные

2. Предобработка

Далее мы выводим некоторые характеристики наших данных (розничная цена, день недели), создаем новые колонки (они содержат данные, которые в дальнейшем мы будем использовать как признаки), вручную обработаем признак cathegory, так как иначе (из-за большого количества возможных значений) есть риск переобучения модели.

3. Визуализация

Визуализируем распределение целевой переменной и ее зависимости от признаков (чтобы убедиться, что данные зависимости действительно наблюдаются). В нашей работе были визуализированы распределение целевой переменной quantity, зависимость

[21] *Репрезентативная выборка – это выборка, с хорошей точностью моделирующая генеральную совокупность (множество всех рассматриваемых объектов).*

количества продаж товара (quantity) от его розничной цены (price), график зависимости числа продаж (quantity) по дням (day), гистограмма зависимости числа продаж (quantity) от дня недели (week_day) и гистограмма зависимости числа продаж (quantity) от категории товара (bread, hotcake, milk_or_sour_milk, cheese_cottage_cheese, others). Так мы смогли убедиться в наличии зависимостей между признаками и целевой переменной.

4. Подготовка данных, разбиение на обучающую и тестовую выборки

Подготовим данные: уберем ненужные колонки (содержащие излишнюю информацию: целевую переменную и данные, напрямую с ней связанные, излишне подробные категории), применим one-hot кодирование для категориального признака week_day. Отделим колонку с целевой переменной, разобьем данные на обучающую и тестовую выборки (мы делаем это вручную, так как в данных как признак используется день недели; иначе при стандартном разбиении 80/20 или 70/30 в тестовую выборку может попасть либо очень маленькое количество объектов, у которых week_day=0, либо они могут не попасть вовсе; мы делаем следующее разбиение: первые три недели месяца – обучающая выборка, последняя неделя - тренировочная).

5. Обучение и сравнение моделей

Далее мы переходим к обучению моделей. Нами были обучены следующие модели: линейная регрессия (в том числе с регуляризацией, с использованием L-1 нормы и с сочетанием L-1 и L-2 норм, с использованием реализованного нами градиентного спуска), kNN, случайный лес и градиентный бустинг (здесь нами было проведено сравнение трех его реализаций в разных библиотеках: sklearn, CatBoost, LightGBM), а также были рассмотрены блендинг (среднее взвешенное) бустинга и случайного леса, блендинг бустинга и kNN. Ниже приводится таблица результатов, которые показывает каждая из моделей (с наилучшими гиперпараметрами).

Табл. 1 Сравнение моделей

Модель	MSE	MAE
Линейная регрессия	21.918	2.678
Гребневая регрессия	21.868	2.671
Lasso	22.562	2.694
ElasticNet	22.421	2.710
Вручную реализованный градиентный спуск	33.108	3.483
kNN	20.283	2.292
Случайный лес	17.681	2.194
Бустинг (CatBoostRegressor)	15.908	2.202
Бустинг и случайный лес	15.669	2.117

Бустинг и kNN	15.615	2.082

Таким образом, в качестве наилучшей модели нами был выбран блендинг градиентного бустинга и kNN.

6. Создание графического интерфейса программы

Кратко опишем функционал созданной нами программы и её интерфейса. При открытии окна программы (после закрытия приветственного окна, см. Рис. 2) можно увидеть, что можно ввести данные о товаре и увидеть прогноз нашей модели (см. Рис. 3). Также программа умеет обрабатывать файлы (список допустимых форматов можно увидеть, кликнув по кнопке «Помощь», там же можно увидеть и некоторые другие инструкции по работе с программой). Для создания графического интерфейса программы была использована библиотека tkinter. С полным кодом программы можно ознакомиться в Приложении 2.

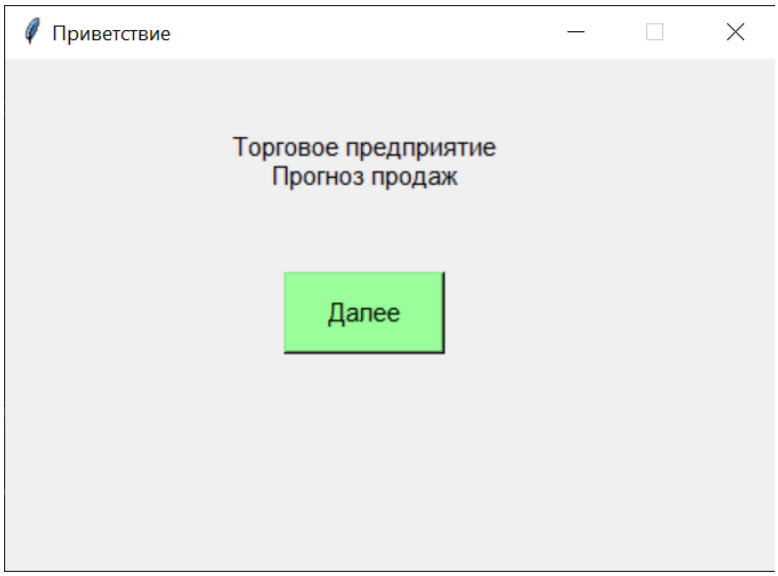


Рис. 2 Окно приветствия

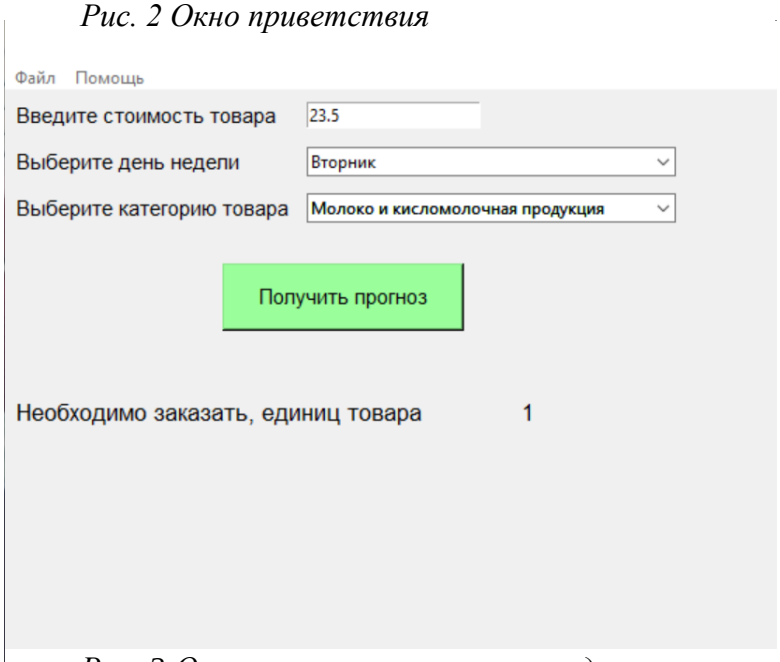


Рис. 2 Основное окно программы и демонстрация её работы

Заключение

В ходе данной исследовательской работы мы изучили основные понятия, лежащие в основе методов машинного обучения: виды задач, решаемых данными алгоритмами, проблемы переобучения и недообучения, понятия метрики и кросс-валидации. Далее, нами были исследованы различные модели машинного обучения: метод kNN (метод k ближайших соседей), линейная регрессия (составление линейного уравнения с весами), решающие деревья, случайный лес и градиентный бустинг (композиции решающих деревьев). Также нами были исследованы возможности языка Python и его дополнительных библиотек: matplotlib и Seaborn (визуализация данных), Pandas (работа с данными), NumPy (вычисления и линейная алгебра), sklearn (библиотека машинного обучения), datetime (работа с информацией о датах и времени) и tkinter (библиотека для создания графического интерфейса). Также были изучены библиотеки, в которых реализован градиентный бустинг, а именно CatBoost и LightGBM. В результате сравнения различных моделей было выявлено, что наилучший результат на наших данных показывает блендинг (среднее взвешенное) градиентного бустинга и kNN. После этого нами был реализован графический интерфейс для взаимодействия пользователя с программой (за основу взята выявленная ранее наилучшая модель).

Можно считать, что цель, как и все поставленные задачи, была выполнена. Созданный нами продукт решает проблему доступа предприятий малого бизнеса к программам, осуществляющим прогноз продаж. Также, наша программа может стать помощником для работников, занимающихся закупкой товара. В этом заключается практическая значимость нашей работы.

Список использованных источников и литературы

1. Документация библиотеки CatBoost. [Электронный ресурс] // Официальный сайт библиотеки CatBoost. URL: https://catboost.ai/ (Дата обращения: 04.11.2021).
2. Документация библиотеки LightGBM. [Электронный ресурс] // Официальный сайт библиотеки LightGBM. URL: https://lightgbm.readthedocs.io/en/latest/index.html (Дата обращения: 04.11.2021).
3. Документация библиотеки NumPy. [Электронный ресурс] // Официальный сайт библиотеки NumPy. URL: https://numpy.org/doc/stable/ (Дата обращения: 04.11.2021).
4. Документация библиотеки Pandas. [Электронный ресурс] // Официальный сайт библиотеки Pandas. URL: https://pandas.pydata.org/docs/user_guide/index.html (Дата обращения: 04.11.2021).
5. Документация библиотеки Scikit-learn. [Электронный ресурс] // Официальный сайт библиотеки Scikit-learn. URL: https://scikit-learn.org/stable/index.html (Дата обращения: 04.11.2021).
6. Документация библиотеки Seaborn. [Электронный ресурс] // Официальный сайт библиотеки Seaborn. URL: https://seaborn.pydata.org (Дата обращения: 04.11.2021).
7. Документация языка Python. [Электронный ресурс] // Документация языка Python 3.10.0. URL: https://docs.python.org/3/ (Дата обращения: 04.11.2021).
8. Ковариация – Википедия: [Электронный ресурс]// Википедия. URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D1%86%D0%B8%D1%8F (Дата обращения: 04.11.2021).
9. Описание Jupiter Notebook. [Электронный ресурс] // Документация Jupiter Notebook. URL: https://jupyter-notebook.readthedocs.io/en/latest/notebook.html (Дата обращения: 05.01.2021).
10. Открытый курс машинного обучения. Тема 5. Композиции: бэггинг, случайный лес: [Электронный ресурс]// IT-сообщество «Хабр». URL: https://habr.com/ru/company/ods/blog/324402/ (Дата обращения: 04.11.2021).
11. Рашка С., Мирджалили В. Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow 2/ Пер. с англ. СПб.: ООО "Диалектика", 2020. 848 с.
12. Соколов Е.А., Зимовнов А.В., Ковалев Е.И., Кохтев В.М., Рысьмятова А.А., Филатов А.А. Основы машинного обучения: [Электронный ресурс]// Образовательная

платформа «Открытое образование». URL: https://courses.openedu.ru/courses/course-v1:hse+INTRML+fall_2020/course/ (Дата обращения: 04.11.2021).

Приложение 1. Код ноутбуков, в которых проводилась исследовательская часть работы

```
# Импортируем необходимые библиотеки
import pandas as pd
import seaborn as sns
import numpy as np
import datetime
```

In [3]:

```
# Считаем данные (pd.read_excel - функция для считывания данных из файла с расширением .xls)
df = pd.read_excel(r'C:\Users\sitki\Desktop\Курсовая\Курсовая_2.0\По_дням_ноябрь_молочка_и_хлеб.xls')
```

In [4]:

```
# Посмотрим на наши данные
df.head()
```

Out[4]:

	Наименование	Количество	Сумма продажи	Прибыль	Дата
0	Аджика домашняя 260гр	2.0	130.00	46.00	2020-11-02
1	Ацидофилин 2,7% 200г.ВМК БЗМЖ	4.0	71.81	9.85	2020-11-02
2	Батон ГОРЧИЧНЫЙ 300 г. Хлебозавод №1 +	21.0	419.90	81.38	2020-11-02
3	Батон Луковый 300г.Орион	2.0	57.93	11.93	2020-11-02
4	Батон НАРЕЗНОЙ 400 г. Хлебозавод №1 +	41.0	901.19	192.71	2020-11-02

In [5]:

```
# Для удобства переименуем колонки
df = df.rename(columns={'Наименование': 'name',
                        'Количество': 'quantity',
                        'Сумма продажи': 'total_sum',
                        'Прибыль': 'income',
                        'Дата': 'date'})
```

In [6]:

```
# Размеры датафрейма
df.shape
```

Out[6]:

```
(2510, 5)
```

In [7]:

```
# Пропусков в датафрейме нет
df.isna().sum()
```

Out[7]:

```
name         0
quantity     0
total_sum    0
income       0
```

```
date         0
dtype: int64
```

In [8]:

```
# Типы значений в колонках датафрейма
df.dtypes
```

Out[8]:

```
name                 object
quantity            float64
total_sum           float64
income              float64
date         datetime64[ns]
dtype: object
```

In [9]:

```
# Создадим колонку, содержащую розничную цену товара
df['price'] = (df.income / df.quantity).round(2)
```

In [10]:

```
# Создадим колонку, содержащую день
df['day'] = df.date.apply(lambda x: x.day)
```

In [11]:

```
# Создадим колонку категории товара, чтобы,
# во-первых, посмотреть, нет ли избыточной информации в данных (товары, не интересующие нас),
# во-вторых, использовать категорию товара как один из признаков
df['cathegory'] = df.name.apply(lambda x: x.split()[0])
```

In [12]:

```
df.head()
```

Out[12]:

	name	quantity	total_sum	income	date	price	day	cathegory
0	Аджика домашняя 260гр	2.0	130.00	46.00	2020-11-02	23.00	2	Аджика
1	Ацидофилин 2,7% 200г.ВМК БЗМЖ	4.0	71.81	9.85	2020-11-02	2.46	2	Ацидофилин
2	Батон ГОРЧИЧНЫЙ 300 г. Хлебозавод №1 +	21.0	419.90	81.38	2020-11-02	3.88	2	Батон
3	Батон Луковый 300г.Орион	2.0	57.93	11.93	2020-11-02	5.96	2	Батон
4	Батон НАРЕЗНОЙ 400 г. Хлебозавод №1 +	41.0	901.19	192.71	2020-11-02	4.70	2	Батон

In [13]:

```
# Видов категорий очень много; модель может переобучиться, если мы оставим данные в таком виде
df.cathegory.unique()
```

Out[13]:

```
array(['Аджика', 'Ацидофилин', 'Батон', 'Батончик', 'Беляш', 'Бифидок',
       'Булка', 'Булочка', 'Булочки', 'Ватрушка', 'Йогурт', 'Йогуртный',
       'Кефир', 'Коктейль', 'Котлета', 'Лаваш', 'Майонез', 'Маргарин',
       'Масло', 'Молоко', 'Напиток', 'Палочка', 'Пампушки', 'Пирог',
       'Пицца', 'Плавленной', 'Плавленый', 'Плетенка', 'Плюшка',
       'Пончики', 'Продукт', 'Простокваша', 'Ромашка', 'Ромовая',
```

```
       'Ряженка', 'Сгущенка', 'Сливки', 'Слойка', 'Сметана', 'Сметанный',
       'Снежок', 'Сосиска', 'Соус', 'Сыр', 'Сырный', 'Сырок', 'Творог',
       'Творожная', 'Хачапури', 'Хлеб', 'Хлеб-е', 'Чиабатта', 'Шанежки',
       'Желе', 'Рожок', 'ХБИ', 'Майонезный', 'Масса', 'Сгущеночка',
       'Обереженка', 'Пироженое', 'Творожок', 'Палочки', 'Пуддинг',
       'Сырки'], dtype=object)
```

In [14]:

```
# Удаление избыточной информации
df = df.query('cathegory != "Ромашка" and cathegory != "Аджика" and cathegory
!= "Желе"')
```

In [15]:

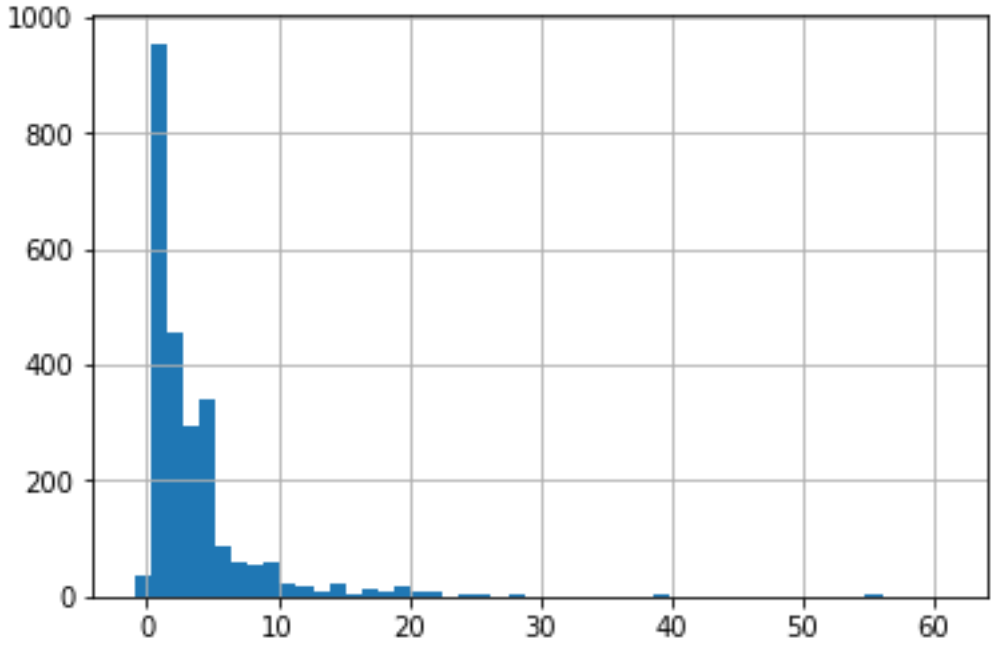
```
# Создание более общих категорий
df['bread'] = np.where(df.cathegory.apply(lambda x: x in [
    'Батон', 'Батончик', 'Булка', 'Булочка', 'Булочки', 'Пампушки', 'Хлеб',
'Хлеб-е', 'Чиабатта', 'ХБИ', 'Лаваш', 'Плетенка'
    ]), 1, 0)
df['hotcake'] = np.where(df.cathegory.apply(lambda x: x in [
    'Беляш', 'Ватрушка', 'Котлета', 'Пирог', 'Пицца', 'Плюшка', 'Пончики',
'Ромовая', 'Слойка', 'Сосиска',
    'Хачапури', 'Шанежки', 'Рожок', 'Пироженое']), 1, 0)
df['milk_or_sour_milk'] = np.where(df.cathegory.apply(lambda x: x in [
    'Ацидофилин', 'Бифидок', 'Йогурт', 'Йогуртный', 'Кефир', 'Коктейль',
'Молоко', 'Напиток', 'Ряженка',
    'Сливки', 'Снежок', 'Обереженка', 'Простокваша']), 1, 0)
df['cheese_cottage_cheese'] = np.where(df.cathegory.apply(lambda x: x in [
    'Палочка', 'Продукт', 'Сырок', 'Творог', 'Плавленной', 'Плавленый',
'Сыр', 'Сырный',
    'Творожная', 'Масса', 'Творожок', 'Палочки', 'Сырки']), 1, 0)
df['others'] = np.where(df.cathegory.apply(lambda x: x in [
    'Майонез', 'Маргарин', 'Масло', 'Сгущенка', 'Соус', 'Сметана',
'Сметанный', 'Майонезный', 'Сгущеночка',
    'Пуддинг']), 1, 0)
```

In [15]:

```
# Построим график распределения целевой переменной
df.quantity.hist(bins=50)
```

Out[15]:

```
<AxesSubplot:>
```

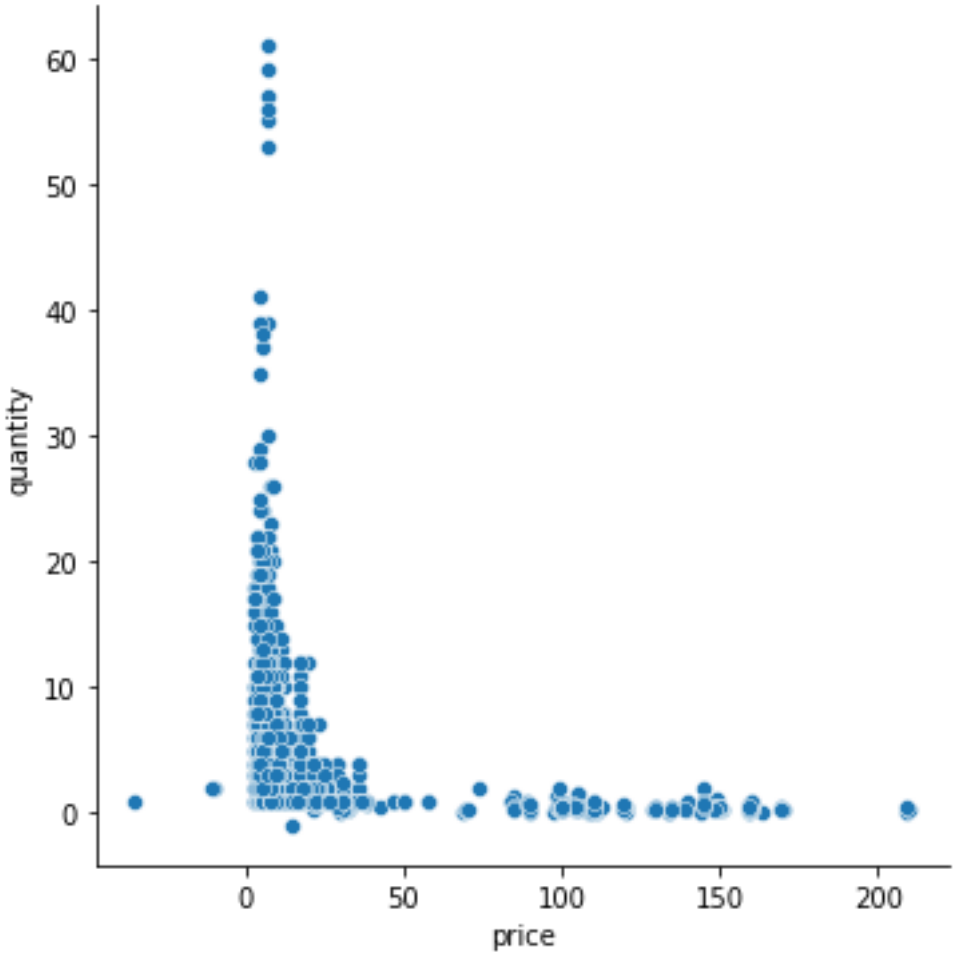


In [16]:

```
# График зависимости количества продаж от их розничной цены
sns.relplot(x='price', y='quantity', data=df)
```

Out[16]:

```
<seaborn.axisgrid.FacetGrid at 0x1dbf7ce0940>
```

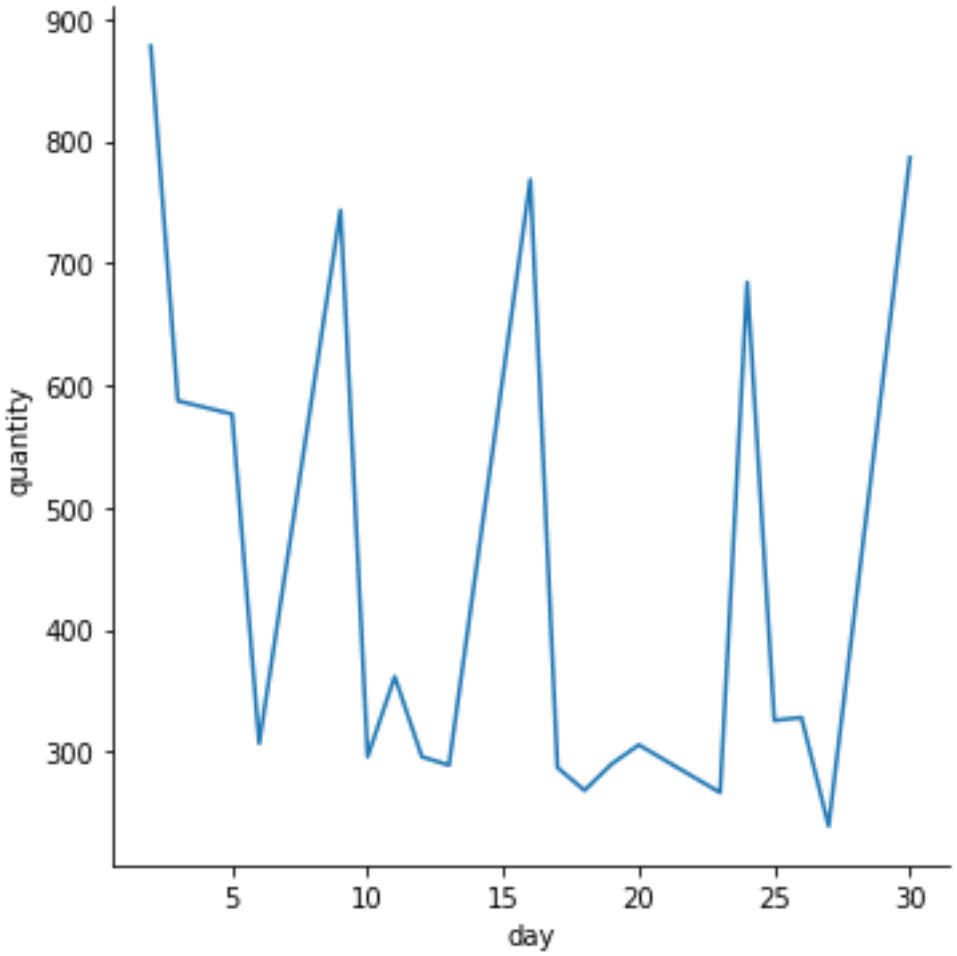


In [17]:

```
# График зависимости количества продаж от даты говорит нам о том, что явной взаимосвязи между этими параметрами нет
# При этом мы видим скачки продаж; можно предположить, что существует зависимость количества продаж от дня недели
sns.relplot(x='day', y='quantity', kind='line', data=df.groupby('day', as_index=False).agg({'quantity': 'sum'}))
```

Out[17]:

```
<seaborn.axisgrid.FacetGrid at 0x1dbf7d92b80>
```



In [16]:

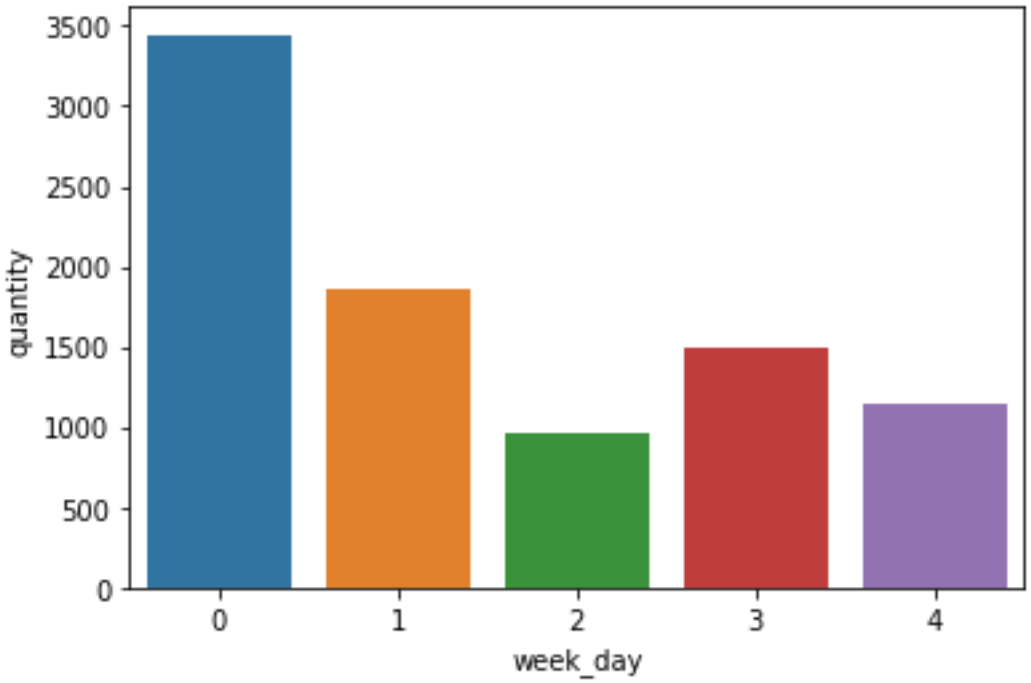
```
# Создадим колонку, содержащую информацию о дне недели, в который произведена продажа
df['week_day'] = df.date.apply(lambda x: x.weekday())
```

In [19]:

```
# Гистограмма зависимости продаж от дня недели (к сожалению, в данных представлены не все дни недели)
# Из гистограммы видно, что наше предположение верно, и, следовательно, день недели можно использовать как ещё один признак
df_by_week_day = df.groupby('week_day', as_index=False).agg({'quantity': 'sum'})
sns.barplot(x='week_day', y='quantity', data=df_by_week_day)
```

Out[19]:

```
<AxesSubplot:xlabel='week_day', ylabel='quantity'>
```

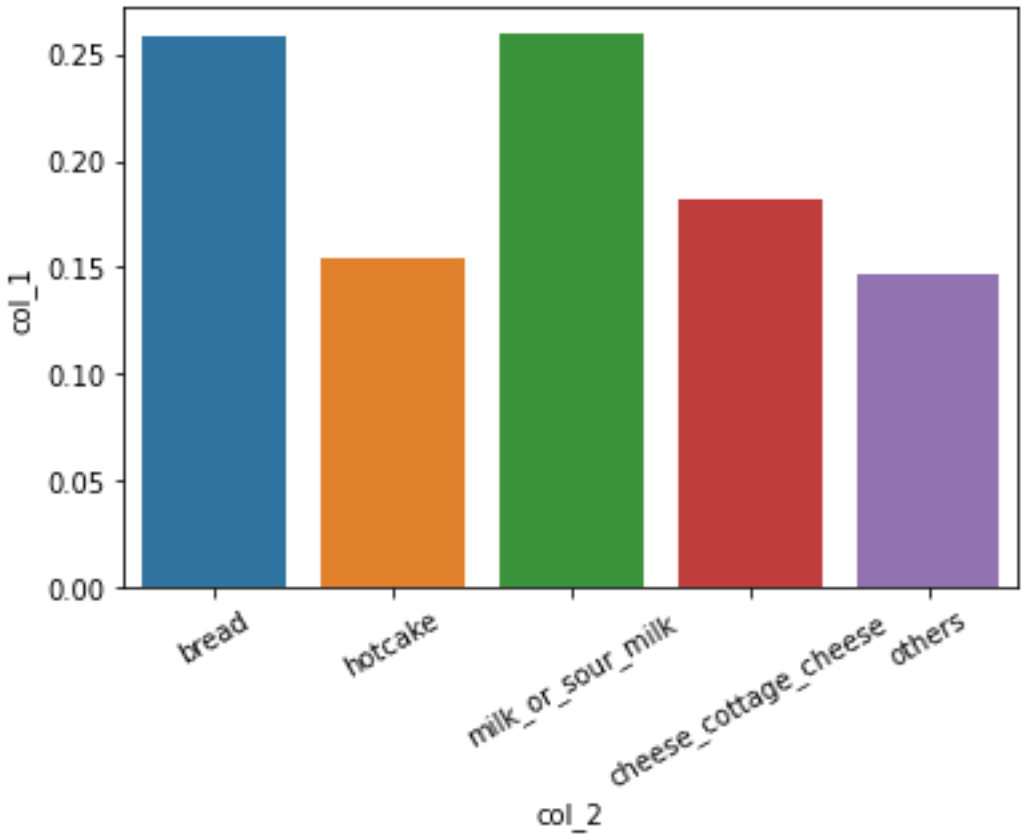


In [20]:

```
# Зависимость средней цены от категории товара
products_cathegories = pd.DataFrame({'col_1': pd.Series(
    [df.bread.mean(), df.hotcake.mean(), df.milk_or_sour_milk.mean(),
df.cheese_cottage_cheese.mean(), df.others.mean()]),
    'col_2': pd.Series(['bread', 'hotcake', 'milk_or_sour_milk', 'cheese_cot-
tage_cheese', 'others'])})
pl = sns.barplot(x='col_2', y='col_1', data=products_cathegories)
pl.set_xticklabels(pl.get_xticklabels(), rotation=30)
```

Out[20]:

```
[Text(0, 0, 'bread'),
 Text(1, 0, 'hotcake'),
 Text(2, 0, 'milk_or_sour_milk'),
 Text(3, 0, 'cheese_cottage_cheese'),
 Text(4, 0, 'others')]
```



In [21]:

```
df.head()
```

Out[21]:

	name	quantity	total_sum	income	date	price	day	cathegory	bread	hotcake	milk_or_sour_milk	cheese_cottage_cheese	others	week_day
1	Ацидофилин 2,7% 200г. ВМК БЗМЖ	4.0	71.81	9.85	2020-11-02	2.46	2	Ацидофилин	0	0	1	0	0	0
2	Батон ГОРЧИЧНЫЙ 300 г. Хлебозавод №1 +	21.0	419.90	81.38	2020-11-02	3.88	2	Батон	1	0	0	0	0	0
3	Батон Луковый 300г. Орион	2.0	57.93	11.93	2020-11-02	5.96	2	Батон	1	0	0	0	0	0

	name	quantity	total_sum	income	date	price	day	cathegory	bread	hotcake	milk_or_sour_milk	cheese_cottage_cheese	others	week_day
4	Батон НАРЕЗНОЙ 400 г. Хлебозавод №1 +	41.0	901.19	192.71	2020-11-02	4.70	2	Батон	1	0	0	0	0	0
5	Батон Нарезной 400г. Переславский х-д.	5.0	164.98	38.48	2020-11-02	7.70	2	Батон	1	0	0	0	0	0

In [17]:

```
# Удалим ненужные колонки (содержащие данные, которые не могут быть нам доступны на момент предсказания и
# вспомогательные колонки)
df = df.drop(['total_sum', 'income', 'date', 'cathegory'], axis=1)
```

In [18]:

```
# День недели - категориальный признак. Чтобы использовать его для модели линейной регрессии, применим one-hot кодирование
df = pd.get_dummies(df, columns=['week_day'])
```

In [19]:

```
df.head()
```

Out[19]:

	name	quantity	price	day	bread	hotcake	milk_or_sour_milk	cheese_cottage_cheese	others	week_day_0	week_day_1	week_day_2	week_day_3	week_day_4
1	Ацидофилин 2,7% 200г. ВМК БЗМЖ	4.0	2.46	2	0	0	1	0	0	1	0	0	0	0
2	Батон ГОРЧИЧНЫЙ 300 г. Хлебозавод №1 +	21.0	3.88	2	1	0	0	0	0	1	0	0	0	0

	name	quantity	price	day	bread	hotcake	milk_or_sour_milk	cheese_cottage_cheese	others	week_day_0	week_day_1	week_day_2	week_day_3	week_day_4
3	Батон Луковый 300г. Орион	2.0	5.96	2	1	0	0	0	0	1	0	0	0	0
4	Батон НАРЕЗНОЙ 400 г. Хлебозавод №1 +	41.0	4.70	2	1	0	0	0	0	1	0	0	0	0
5	Батон Нарезной 400г. Переславский х-д.	5.0	7.70	2	1	0	0	0	0	1	0	0	0	0

In [20]:

```
df.to_csv('data.csv', index=False)
```

In [ ]:

```
# Импортируем необходимые библиотеки
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings


warnings.filterwarnings("ignore", category=UserWarning)
```

In [2]:

```
# Считаем данные
df = pd.read_csv(r'data.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	name	quantity	price	day	bread	hotcake	milk_or_sour_milk	cheese_cottage_cheese	others	week_day_0	week_day_1	week_day_2	week_day_3	week_day_4
0	Ацидофилин 2,7% 200г. ВМК БЗМ Ж	4.0	2.46	2	0	0	1	0	0	1	0	0	0	0
1	Батон ГОРЧИЧНЫЙ 300 г. Хлебозавод №1 +	21.0	3.88	2	1	0	0	0	0	1	0	0	0	0
2	Батон Луковый 300г. Орион	2.0	5.96	2	1	0	0	0	0	1	0	0	0	0
3	Батон НАРЕЗНОЙ 400 г. Хлебозавод №1 +	41.0	4.70	2	1	0	0	0	0	1	0	0	0	0
4	Батон Нарезной 400г. Переславский х-д.	5.0	7.70	2	1	0	0	0	0	1	0	0	0	0

In [4]:

```
# Разобьем выборку на обучающую и тестовую
# Мы делаем это вручную, так как в датафрейме используется день недели как признак
# (одна неделя идет в тестовую выборку, остальное - в обучающую)
```

```
# Здесь же удаляем ненужные колонки
train_df = df.query('day < 23').drop(['day', 'name'], axis=1)
test_df = df.query('day >= 23').drop(['day', 'name'], axis=1)
```

In [5]:

```
# Отделим целевую переменную
X_train, y_train = train_df.drop('quantity', axis=1), train_df.quantity
X_test, y_test = test_df.drop('quantity', axis=1), test_df.quantity
```

In [6]:

```
# Теперь мы можем перейти к обучению моделей
```

In [7]:

```
# Импортируем необходимые инструменты
# Также часть инстументов будет импортироваться далее
from sklearn.metrics import mean_squared_error as mse, mean_absolute_error as
mae
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
```

In [8]:

```
# Первой мы обучим линейную регрессию с разными её вариациями
```

In [9]:

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
```

In [10]:

```
# Обучение обычной линейной регрессии
reg = LinearRegression().fit(X_train, y_train)
```

In [11]:

```
# Метрики качества модели
print('mse:', mse(y_test, reg.predict(X_test)))
print('mae:', mae(y_test, reg.predict(X_test)))
mse: 21.91779155646977
mae: 2.6781852582607693
```

In [12]:

```
# Посмотрим, улучшится ли результат, если масштабировать признаки
sc = StandardScaler()
X_train_scaled = pd.DataFrame(sc.fit_transform(X_train), columns=X_train.col-
umns, index=X_train.index)
X_test_scaled = pd.DataFrame(sc.transform(X_test), columns=X_test.columns,
index=X_test.index)
```

In [13]:

```
reg2 = LinearRegression().fit(X_train_scaled, y_train)
```

In [14]:

```
print('mse:', mse(y_test, reg2.predict(X_test_scaled)))
print('mae:', mae(y_test, reg2.predict(X_test_scaled)))
mse: 22.132034816033652
mae: 2.7425151877595053
```

In [15]:

```
# Посмотрим, улучшится ли результат, если мы добавим регуляризацию
ridge = GridSearchCV(
    Ridge(),
    param_grid={'alpha': [i for i in range (1, 200)]}
)
ridge.fit(X_train, y_train)
ridge.best_params_
```

Out[15]:

```
{'alpha': 34}
```

In [16]:

```
print('mse:', mse(y_test, ridge.predict(X_test)))
print('mae:', mae(y_test, ridge.predict(X_test)))
mse: 21.868158995924603
mae: 2.671261623395296
```

In [17]:
```
# Посмотрим, улучшится ли результат, если мы будем использовать L1 норму
(Lasso)
l=[i for i in range (1, 200)]
lasso = GridSearchCV(
    Lasso(),
    param_grid={'alpha': l}
)
lasso.fit(X_train, y_train)
lasso.best_params_
```

Out[17]:
```
{'alpha': 3}
```

In [18]:
```
print('mse:', mse(y_test, lasso.predict(X_test)))
print('mae:', mae(y_test, lasso.predict(X_test)))
mse: 22.56229605365303
mae: 2.694127261776902
```

In [19]:
```
# Попробуем совместить L1-норму и L2-норму (используем ElasticNet)
l_1=[i for i in range (1, 50)]
l_2=[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
el_net = GridSearchCV(
    ElasticNet(),
    param_grid={'alpha': l_1, 'l1_ratio': l_2}
)
el_net.fit(X_train, y_train)
el_net.best_params_
```

Out[19]:
```
{'alpha': 1, 'l1_ratio': 0.1}
```

In [20]:
```
print('mse:', mse(y_test, el_net.predict(X_test)))
print('mae:', mae(y_test, el_net.predict(X_test)))
mse: 22.421067538909234
mae: 2.7100039005587533
```

In [21]:
```
# Также, как было сказано в теоретической части, можно обучать линейную ре-
грессию с помощью градиентного спуска
# Давайте попробуем использовать вручную реализованный градиетный спуск и по-
смотрим, изменится ли значение ошибки модели
```

In [22]:
```
from sklearn.base import BaseEstimator

class Gradient_descent(BaseEstimator):
    def __init__(self, eps=1e-4, max_steps=1000, w0=None, alpha=1e-2):
        """
        eps: разница для нормы изменения весов
        max_steps: максимальное кол-во шагов
        w0: np.array(d) - начальные веса
        alpha: шаг обучения
        """
        self.eps = eps
        self.max_steps = max_steps
        self.w0 = w0
        self.alpha = alpha
        self.w = None

    def fit(self, X, y):
        """
```

```
        X: np.array(l, d)
        y: np.array(l)
        ---
        output: self
        """

        if self.w0 is None:
            self.w0 = np.zeros(X.shape[1])

        self.w = self.w0

        for step in range(self.max_steps):
            w_new = self.w - self.alpha * self.calc_gradient(X, y)

            if np.linalg.norm(w_new - self.w) < self.eps:
                break

            self.w = w_new

        return self

    def calc_gradient(self, X, y):
        """
        X: np.array(l, d)
        y: np.array(l)
        ---
        output: np.array(d)
        """

        return (2/X.shape[0]) * np.dot(X.T, (np.dot(X, self.w) - y))

    def predict(self, X):
        """
        X: np.array(l, d)
        ---
        output: np.array(l)
        """

        if self.w is None:
            raise Exception('Not trained yet')

        return np.dot(X, self.w)
```

In [23]:

```
gd = Gradient_descent()
gd.fit(X_train_scaled, y_train)
print('mse:', mse(y_test, gd.predict(X_test_scaled)))
print('mae:', mae(y_test, gd.predict(X_test_scaled)))
```

```
mse: 33.107698865297436
mae: 3.4829881370522395
```

In [24]:

```
# Теперь обучим kNN
```

In [25]:

```
from sklearn.neighbors import KNeighborsRegressor
```

In [26]:

```
knn = GridSearchCV(
    KNeighborsRegressor(),
    param_grid={
        'n_neighbors': list(range(2, 30)),
        'weights': ['uniform', 'distance'],
```

```
        'p': [1, 2, 3]
    }
)
```

In [27]:

```
knn.fit(X_train, y_train)
```

Out[27]:

```
GridSearchCV(estimator=KNeighborsRegressor(),
             param_grid={'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13,
                                         14, 15, 16, 17, 18, 19, 20, 21, 22,
23,
                                         24, 25, 26, 27, 28, 29],
                         'p': [1, 2, 3], 'weights': ['uniform', 'distance']})
```

In [28]:

```
knn.best_params_
```

Out[28]:

```
{'n_neighbors': 25, 'p': 1, 'weights': 'distance'}
```

In [29]:

```
print('mse:', mse(y_test, knn.predict(X_test)))
print('mae:', mae(y_test, knn.predict(X_test)))
```

```
mse: 20.283274136864765
mae: 2.292419053467014
```

In [30]:

```
knn.predict(X_train)
```

Out[30]:

```
array([ 4. , 21. ,  4.5, ...,  3. ,  4.5,  7. ])
```

In [31]:

```
# Попробуем обучить KNN на масштабированных данных
knn.fit(X_train_scaled, y_train)
```

Out[31]:

```
GridSearchCV(estimator=KNeighborsRegressor(),
             param_grid={'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13,
                                         14, 15, 16, 17, 18, 19, 20, 21, 22,
23,
                                         24, 25, 26, 27, 28, 29],
                         'p': [1, 2, 3], 'weights': ['uniform', 'distance']})
```

In [32]:

```
knn.best_params_
```

Out[32]:

```
{'n_neighbors': 28, 'p': 3, 'weights': 'distance'}
```

In [33]:

```
print('mse:', mse(y_test, knn.predict(X_test_scaled)))
print('mae:', mae(y_test, knn.predict(X_test_scaled)))
```

```
mse: 20.299726280616284
mae: 2.30012771888199
```

In [34]:

```
# Теперь обучим случайный лес
```

In [35]:

```
from sklearn.ensemble import RandomForestRegressor
```

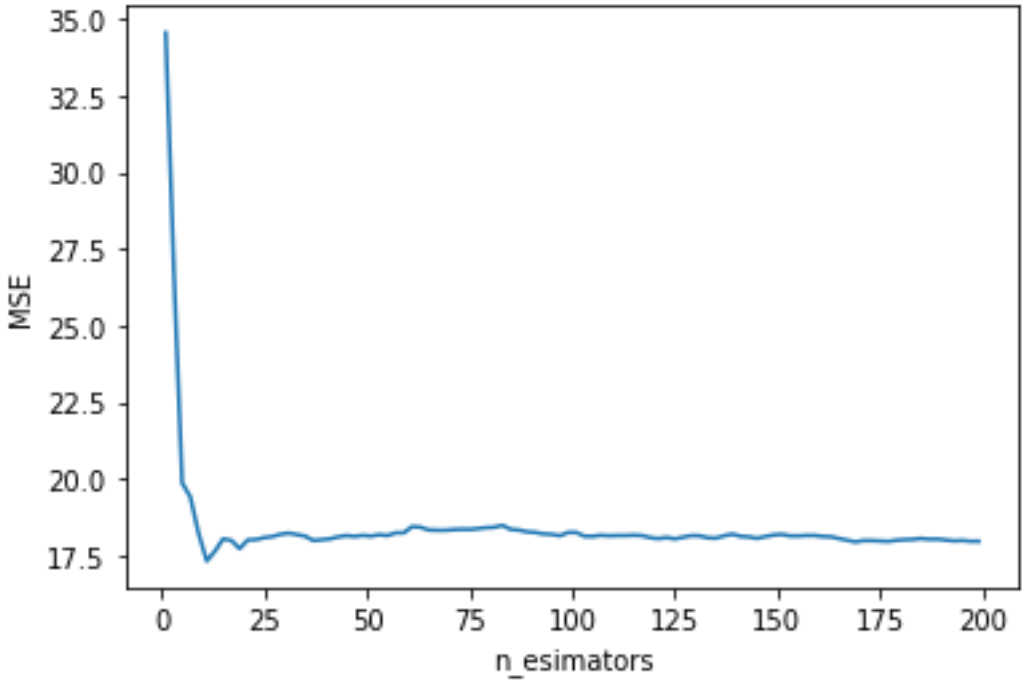
In [36]:

```
forest = RandomForestRegressor(min_samples_split=5, max_features=0.3)
forest.fit(X_train, y_train)
print('mse:', mse(y_test, forest.predict(X_test)))
print('mae:', mae(y_test, forest.predict(X_test)))
```

```
mse: 17.68097674692905
mae: 2.193616321124681
```

In [37]:

```
# При построении графика зависимости качества модели от количества дереьев
видно, что
# качество в определенный момент выходит на асимптоту
Q = []
for n_est in range(1, 200, 2):
    model = RandomForestRegressor(n_estimators=n_est, min_samples_split=7,
max_features=0.3, random_state=13)
    model.fit(X_train, y_train)
    Q.append(mse(y_test, model.predict(X_test)))

plt.plot(range(1, 200, 2), Q)
plt.xlabel('n_esimators')
plt.ylabel('MSE')
plt.show()
```



In [38]:

```
# Попробуем обучить случайный лес на масштабированных данных
forest.fit(X_train_scaled, y_train)
print('mse:', mse(y_test, forest.predict(X_test_scaled)))
print('mae:', mae(y_test, forest.predict(X_test_scaled)))
mse: 17.801231013083605
mae: 2.2078948081322474
```

In [39]:

```
# Теперь обучим градиентный бустинг
```

In [40]:

```
from sklearn.ensemble import GradientBoostingRegressor
```

In [41]:

```
gbm = GradientBoostingRegressor()
gbm.fit(X_train, y_train)
```

Out[41]:

```
GradientBoostingRegressor()
```

In [42]:

```
print('mse:', mse(y_test, gbm.predict(X_test)))
print('mae:', mae(y_test, gbm.predict(X_test)))
mse: 16.815422122748263
mae: 2.2071922070011767
```

In [43]:

```
# Попробуем подобрать гиперпараметры модели
```

```
# Также сравним модели, представленные в разных библиотеках
```

In [44]:

```
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
```

In [45]:

```
def trees_num_plot(X_train, y_train, X_test, y_test):
    n_trees = [1, 5, 10, 50, 100, 300, 500, 700, 1000]
    quals_train_lgb = []
    quals_test_lgb = []
    quals_train_cbm = []
    quals_test_cbm = []
    quals_train_gb = []
    quals_test_gb = []

    for n in n_trees:
        lgb = LGBMRegressor(n_estimators=n)
        lgb.fit(X_train, y_train)
        q_train_lgb = mse(y_train, lgb.predict(X_train))
        q_test_lgb = mse(y_test, lgb.predict(X_test))

        cbm = CatBoostRegressor(n_estimators=n, logging_level="Silent")
        cbm.fit(X_train, y_train)
        q_train_cbm = mse(y_train, cbm.predict(X_train))
        q_test_cbm = mse(y_test, cbm.predict(X_test))

        gb = GradientBoostingRegressor(n_estimators=n)
        gb.fit(X_train, y_train)
        q_train_gb = mse(y_train, gb.predict(X_train))
        q_test_gb = mse(y_test, gb.predict(X_test))

        quals_train_lgb.append(q_train_lgb)
        quals_test_lgb.append(q_test_lgb)
        quals_train_cbm.append(q_train_cbm)
        quals_test_cbm.append(q_test_cbm)
        quals_train_gb.append(q_train_gb)
        quals_test_gb.append(q_test_gb)


    plt.figure(figsize=(11.5, 7))
    plt.plot(n_trees, quals_train_lgb, marker='.', label='LightGBM train')
    plt.plot(n_trees, quals_test_lgb, marker='.', label='LightGBM test')
    plt.plot(n_trees, quals_train_cbm, marker='.', label='CatBoost train')
    plt.plot(n_trees, quals_test_cbm, marker='.', label='CatBoost test')
    plt.plot(n_trees, quals_train_gb, marker='.', label='Sklearn train')
    plt.plot(n_trees, quals_test_gb, marker='.', label='Sklearn test')
    plt.xlabel('Number of trees')
    plt.ylabel('MSE')
    plt.legend()

    plt.show()
```

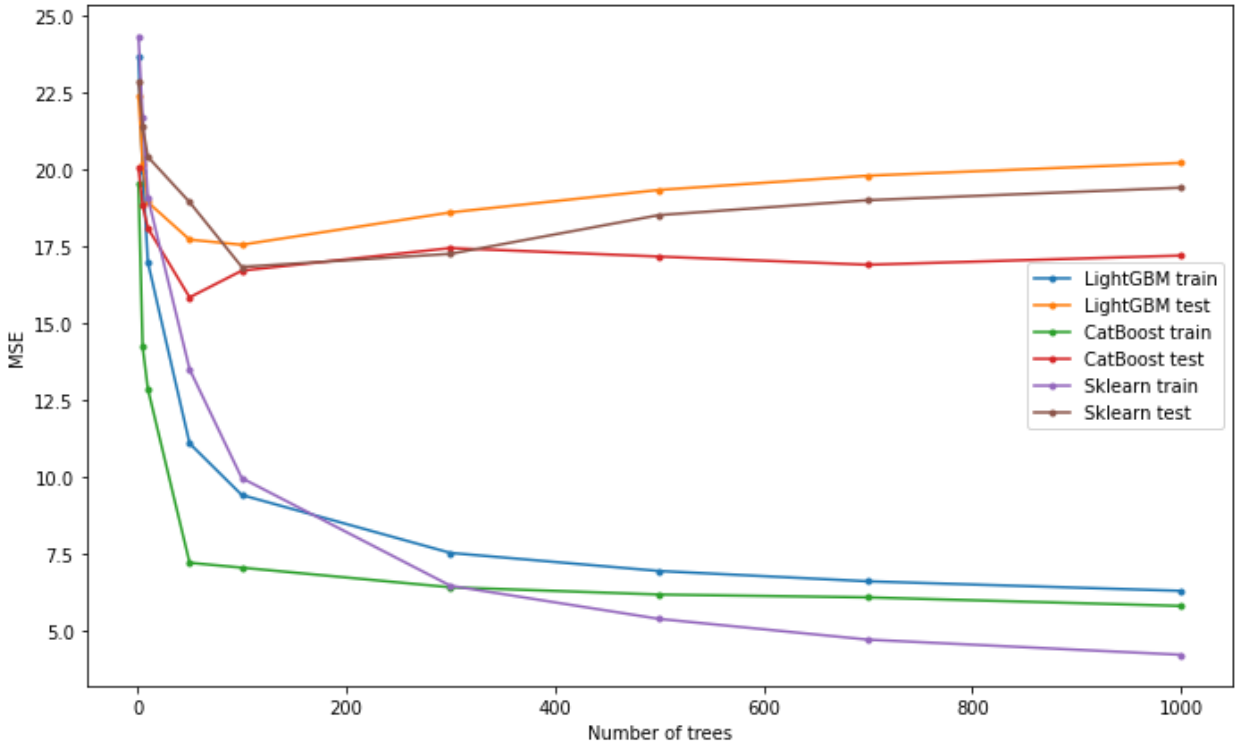
In [46]:

```
def trees_depth_plot(X_train, y_train, X_test, y_test):
    depth = list(range(1, 12, 2))
    n_trees = 100
    quals_train_lgb = []
    quals_test_lgb = []
    quals_train_cbm = []
    quals_test_cbm = []
    quals_train_gb = []
```

```
    quals_test_gb = []

    for d in depth:
        lgb = LGBMRegressor(n_estimators=n_trees, max_depth=d)
        lgb.fit(X_train, y_train)
        q_train_lgb = mse(y_train, lgb.predict(X_train))
        q_test_lgb = mse(y_test, lgb.predict(X_test))

        cbm = CatBoostRegressor(n_estimators=n_trees, max_depth=d, log-
ging_level="Silent")
        cbm.fit(X_train, y_train)
        q_train_cbm = mse(y_train, cbm.predict(X_train))
        q_test_cbm = mse(y_test, cbm.predict(X_test))

        gb = GradientBoostingRegressor(n_estimators=n_trees, max_depth=d)
        gb.fit(X_train, y_train)
        q_train_gb = mse(y_train, gb.predict(X_train))
        q_test_gb = mse(y_test, gb.predict(X_test))

        quals_train_lgb.append(q_train_lgb)
        quals_test_lgb.append(q_test_lgb)
        quals_train_cbm.append(q_train_cbm)
        quals_test_cbm.append(q_test_cbm)
        quals_train_gb.append(q_train_gb)
        quals_test_gb.append(q_test_gb)


    plt.figure(figsize=(11.5, 7))
    plt.plot(depth, quals_train_lgb, marker='.', label='LightGBM train')
    plt.plot(depth, quals_test_lgb, marker='.', label='LightGBM test')
    plt.plot(depth, quals_train_cbm, marker='.', label='CatBoost train')
    plt.plot(depth, quals_test_cbm, marker='.', label='CatBoost test')
    plt.plot(depth, quals_train_gb, marker='.', label='Sklearn train')
    plt.plot(depth, quals_test_gb, marker='.', label='Sklearn test')
    plt.xlabel('Depth of trees')
    plt.ylabel('MSE')
    plt.legend()

    plt.show()
```

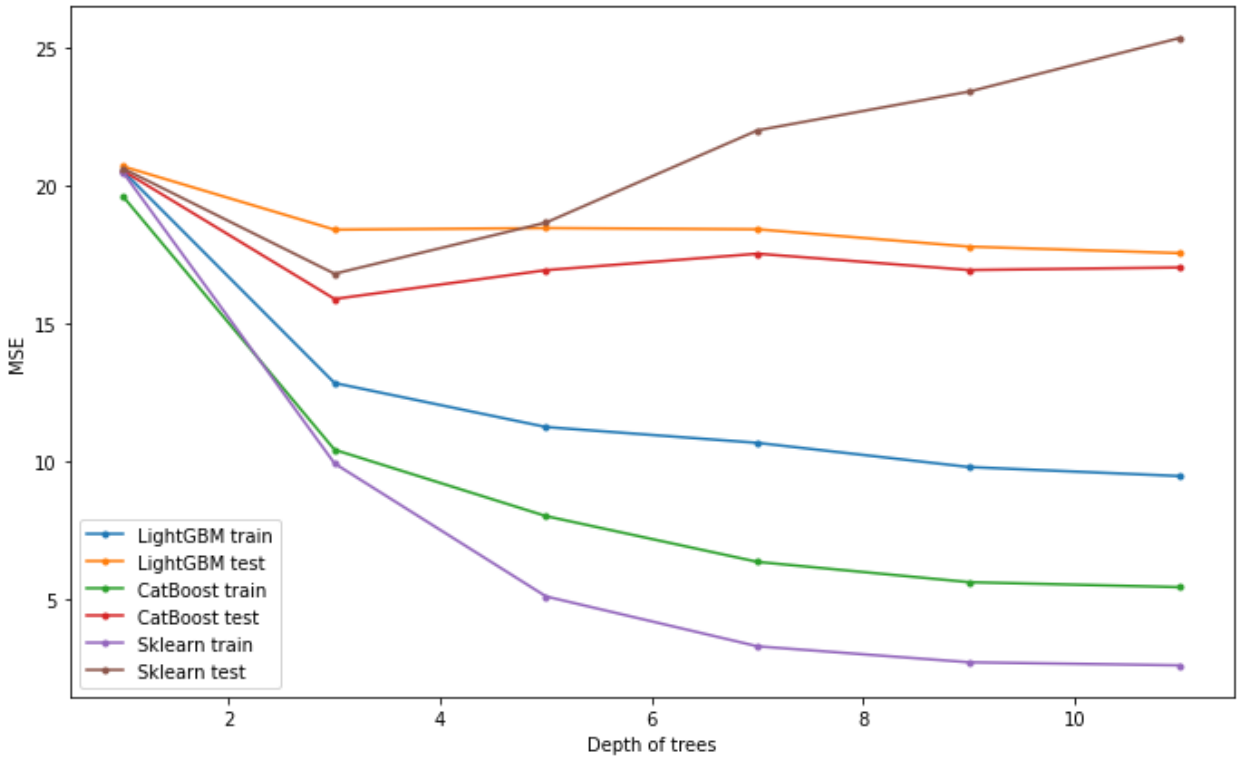
In [47]:

```
trees_num_plot(X_train, y_train, X_test, y_test)
```



In [48]:

```
trees_depth_plot(X_train, y_train, X_test, y_test)
```



In [49]:

```
# Из графиков видно, что модель, представленная ниже, наилучшая
cb_reg = CatBoostRegressor(logging_level='Silent', n_estimators=100,
max_depth=3)
cb_reg.fit(X_train, y_train)
print('mse:', mse(y_test, cb_reg.predict(X_test)))
print('mae:', mae(y_test, cb_reg.predict(X_test)))
mse: 15.907881599713244
mae: 2.20195774294398
```

In [50]:

```
# Также можно заметить, что масштабирование практически не изменяет значения ошибок
cb_reg_sc = CatBoostRegressor(logging_level='Silent', n_estimators=100, max_depth=3)
cb_reg_sc.fit(X_train_scaled, y_train)
print('mse:', mse(y_test, cb_reg_sc.predict(X_test_scaled)))
print('mae:', mae(y_test, cb_reg_sc.predict(X_test_scaled)))
mse: 15.854802008501085
mae: 2.2014166657616823
```

In [51]:

```
# Последнее, что мы попробуем - блендинг бустинга и случайного леса, блендинг бустинга и kNN
```

In [52]:

```
def select_weights(y_true, y_pred_1, y_pred_2):
    metric = []
    grid = np.linspace(0, 1, 1000)
    for w_0 in grid:
        w_1 = 1 - w_0
        y = y_pred_1 * w_0 + y_pred_2 * w_1
        metric.append([mse(y_true, y), mae(y_true, y), w_0, w_1])

    return min(metric, key=lambda x: x[0])
```

In [53]:

```
select_weights(y_test, forest.predict(X_test), cb_reg.predict(X_test))
```

Out[53]:

```
[15.669335150035092,
 2.1167794857921525,
 0.12412412412412413,
 0.8758758758758759]
```

In [54]:

```
select_weights(y_test, knn.predict(X_test), cb_reg.predict(X_test))
```

Out[54]:

```
[15.614716100212577,
 2.0816342057209103,
 0.11711711711711711,
 0.8828828828828829]
```

In [55]:

```
# Заключение: как можно заметить, лучшей моделью оказался блендинг градиентного бустинга из библиотеки CatBoost и KNN
# Далее мы реализуем графический интерфейс, в котором будем использовать именно эту модель
```

In [56]:

```
# Загрузим обученную нами модель в файл
cb_reg.save_model("cb_reg.txt")
```

In [57]:

```
with open(r"weights.txt", "w") as file:
    param = select_weights(y_test, knn.predict(X_test), cb_reg.predict(X_test))
    file.write(str(param[2]))
    file.write('\n')
    file.write(str(param[3]))
```

In [58]:

```
X_train.to_csv('train_data.csv', index=False)
y_train.to_csv('train_data_targets.csv', index=False)
```

In [59]:

```
df.drop(['day', 'name', 'quantity'], axis=1).to_csv('data_to_test.csv', index=False)
```

In [ ]:

Приложение 2. Код программы с графическим интерфейсом

```
from tkinter import filedialog as fd
from tkinter import messagebox as mb
from tkinter import *
from tkinter.ttk import Combobox as Cb
import prediction_model as prm
import warnings

warnings.filterwarnings("ignore")


def is_digit(string):
    if string.isdigit():
        return True
    else:
        try:
            float(string)
            return True
        except ValueError:
            return False


def write_help():
    text = '''Инструкции по использованию программы:
    1) В качестве цены товара может быть введено только число, целая и дробная части разделяются точкой
    2) Программа обрабатывает файлы с расширениями .csv и .xlsx
    3) При использовании файлового ввода данных программа сохраняет файл с результатами в той же директории, где лежит исходный файл;
    4) Файл сохраняется с расширением .xlsx, с названием <имя_входного файла>_predicted.xlsx'''
    mb.showinfo('Помощь', text)


def return_predicted_file():
    file_name = fd.askopenfilename()
    exp = file_name[-4:]
    if exp != '.csv' and exp != '.xlsx' and file_name != '':
        mb.showerror('Неверный формат файла', '''Работа с данным типом файлов не поддерживается''')
        return
    try:
        res_name = prm.predict_file(file_name, exp)
        mb.showinfo('Готово', f'Результаты прогноза были сохранены в файл {res_name}')
    except FileNotFoundError:
        pass
    except:
        mb.showerror('Некорректные данные', 'Данные в файле некорректны')


class Window:
    def __init__(self, title='Прогноз продаж', geometry='550x450+175+125', resizable=(False, False)):
        self.root = Tk()
        self.root.title(title)
        self.root.geometry(geometry)
```

```
        self.root.resizable(resizable[0], resizable[1])
        self.price_entry = Entry(self.root)
        self.week_day_cb = Cb(self.root,
                              values=('Понедельник', 'Вторник', 'Среда',
'Четверг', 'Пятница'),
                              state='readonly')
        self.category_cb = Cb(self.root,
                              values=('Хлеб', 'Пирожки', 'Молоко и кисломо-
лочная продукция', 'Сыр и творог', 'Другое'),
                              state='readonly')
        self.lbl_ans = Label(self.root, font='Calibri, 13')

    def greeting(self):
        greet_window = Toplevel(self.root)
        greet_window.title('Приветствие')
        greet_window.geometry('450x300+200+150')
        greet_window.resizable(False, False)
        Label(greet_window,
              text='''Торговое предприятие\nПрогноз продаж''',
              font='Calibri, 11') \
            .grid(column=0, row=0, padx=130, pady=40, sticky=W + E)
        Button(greet_window, text='Далее', command=greet_window.destroy,
padx=20, pady=10, font='Calibri, 11',
               bg='PaleGreen1') \
            .grid(column=0, row=1, padx=5, pady=5)
        greet_window.grab_set()
        greet_window.focus_set()
        greet_window.wait_window()

    def draw_menu(self):
        menu_bar = Menu(self.root, tearoff=0)

        file_menu = Menu(menu_bar, tearoff=0)
        file_menu.add_command(label='Открыть', command=return_predicted_file)
        menu_bar.add_cascade(label='Файл', menu=file_menu)

        menu_bar.add_command(label='Помощь', command=write_help)

        self.root.configure(menu=menu_bar)

    def draw_widgets(self):
        self.draw_menu()
        Label(self.root, text='Введите стоимость товара', font='Calibri, 11')
\
            .grid(column=0, row=0, sticky=W, padx=5, pady=5)
        Label(self.root, text='Выберите день недели', font='Calibri, 11') \
            .grid(column=0, row=1, sticky=W, padx=5, pady=5)
        Label(self.root, text='Выберите категорию товара', font='Calibri,
11') \
            .grid(column=0, row=2, sticky=W, padx=5, pady=5)

        self.price_entry.grid(column=1, row=0, sticky=W, padx=5, pady=5)
        self.week_day_cb.grid(column=1, row=1, sticky=W, padx=5, pady=5)
        self.week_day_cb.configure(width=40)
        self.category_cb.grid(column=1, row=2, sticky=W, padx=5, pady=5)
        self.category_cb.configure(width=40)

        Button(self.root, text='Получить прогноз', command=self.output_pre-
diction,
               padx=20, pady=10, font='Calibri, 11', bg='PaleGreen1') \
            .grid(column=0, row=3, columnspan=3, pady=23)

    def output_prediction(self):
        price = self.price_entry.get()
```

```
        week_day = self.week_day_cb.current()
        category = self.category_cb.current()

        if week_day == -1 or category == -1 or price is None:
            mb.showwarning('Данные не были введены', 'Пожалуйста, введите
данные')
            return

        if not is_digit(price):
            mb.showerror('Некорректные данные', 'Введите корректные данные')
            return

        res = prm.predict_object(price, week_day, category)

        self.lbl_ans.configure(text=f'Необходимо заказать, единиц то-
вара\t{res}')
        self.lbl_ans.grid(column=0, row=4, columnspan=10, sticky=W, padx=5,
pady=23)

    def run(self):
        self.root.withdraw()
        self.greeting()
        self.root.deiconify()
        self.draw_widgets()
        self.root.mainloop()


if __name__ == '__main__':
    window = Window()
    window.run()



from catboost import CatBoostRegressor
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
import numpy as np
import warnings


warnings.filterwarnings("ignore", category=UserWarning)

X_train = pd.read_csv(r'train_data.csv')
y_train = pd.read_csv(r'train_data_targets.csv')

knn = KNeighborsRegressor(n_neighbors=25, p=1, weights='distance')
knn.fit(X_train, y_train)

cb_reg = CatBoostRegressor()
cb_reg.load_model("cb_reg.txt")

with open(r'weights.txt', 'r') as f:
    w_knn = float(f.readline()[:-1])
    w_cb_reg = float(f.readline())


def predict_object(price, week_day, category):
    cat_list = ['bread', 'hotcake', 'milk_or_sour_milk', 'cheese_cot-
tage_cheese', 'others']

    df = pd.DataFrame({'price': pd.Series([price]),
                       'bread': pd.Series([0]),
                       'hotcake': pd.Series([0]),
                       'milk_or_sour_milk': pd.Series([0]),
```

```
                        'cheese_cottage_cheese': pd.Series([0]),
                        'others': pd.Series([0]),
                        'week_day_0': pd.Series([0]),
                        'week_day_1': pd.Series([0]),
                        'week_day_2': pd.Series([0]),
                        'week_day_3': pd.Series([0]),
                        'week_day_4': pd.Series([0])})

    df[cat_list[category]] = 1
    df['week_day_' + str(week_day)] = 1

    return int(predict(df).round()[0])


def predict(df):
    return np.reshape(knn.predict(df) * w_knn + np.reshape(cb_reg.pre-
dict(df), (df.shape[0], 1)) * w_cb_reg, (df.shape[0]))


def predict_file(file, exp):
    if exp == '.csv':
        df = pd.read_csv(file)
        res_name = file[:-4] + '_predicted.xlsx'
    else:
        df = pd.read_excel(file)
        res_name = file[:-5] + '_predicted.xlsx'

    df['res'] = pd.Series(predict(df).round())
    df.to_excel(res_name)

    return res_name


if __name__ == '__main__':
    print(predict_file(r'C:/Users/sitki/PycharmPro-
jects/data/train_data_1.csv', '.csv'))
    print(predict_file(r'C:/Users/sitki/PycharmPro-
jects/data/train_data_2.xlsx', '.xlsx'))
```