Содержание

Аннотация

Задача ценообразования является актуальной во многих сферах, так как от процесса расчета цен во многом зависит итоговая прибыль компании. Применение классических методов требует большого количества данных, человеческих ресурсов и потому не может быстро реагировать на изменения на рынке. В то время как динамическое ценообразование предполагает автоматизацию данного процесса и включает в себя подходы, способные учитывать текущие характеристики рынка.

В данной работе исследуется задача динамического ценообразования для объектов недвижимости. Динамическое ценообразование заключается в моделировании цен с учетом результатов прошлого временного периода, а также изменений внешних факторов. Одним из методов решения поставленной задачи является применение алгоритма многоруких бандитов. Алгоритм позволяет находить оптимальную цену объекта для максимизации прибыли в долгосрочной перспективе.

В работе были исследованы различные варианты алгоритма многоруких бандитов, проведены эксперименты на реальных данных, оценены результаты. Гипотеза о том, что данный алгоритм применим к задаче ценообразования недвижимости оказалась верна.

Ключевые слова: Алгоритм многоруких бандитов, Томпсоновское сэмплирование, динамическое ценообразование, ценообразование недвижимости, парсер, датасет, континуальная интеграция и доставка

Распределение задач

- Сбор данных и DevOps - Глава 1 **(Анохов Владислав Дмитриевич, БПМИ 218)**
  - Исследование методов сбора данных из внешних источников
  - Реализация алгоритма сбора данных с сайта Центрального Банка для получения актуального курса валют и внешнеэкономических показателей
  - Реализация алгоритмов по сбору открытых данных про недвижимость на различных интернет-ресурсах
  - Упаковывание проекта в контейнер Docker с целью обеспечения его максимальной портабельности и удобства развертывания на различных платформах.
  - Автоматизация процесса сборки и запуска контейнера при помощи инструмента Jenkins, который позволил настроить постоянный запуск скрипта в контейнере.
- Построение модели - Глава 2 **(Горбач Марина Павловна, БПМИ 212)**
  - Исследование методов динамического ценообразования, изучение литературы по теме работы
  - Изучение работы алгоритма многоруких бандитов, вариаций его архитектуры
  - Построение модели для нахождения оптимальной цены для объектов недвижимости
  - Обучение модели, анализ данных, выбор признаков с наибольшим вкладом в качество модели, выбор параметров
  - Тестирование модели, определение качества

Основные определения и утверждения

Парсер – скрипт или программа, собирающая и анализирующая данные с веб-ресурсов, для выдачи их в требуемом формате для дальнейшей работы.

Континуальная интеграция и доставка – методология разработки программного обеспечения, которая предполагает автоматизацию процессов сборки, тестирования и развертывания приложений.

Динамическое ценообразование – стратегия ценообразования, при которой предприятия устанавливают гибкие цены на продукты или услуги на основе текущих потребностей рынка.

Алгоритм многоруких бандитов – алгоритм, решающий задачу, в которой фиксированный ограниченный набор ресурсов должен быть распределен между конкурирующими вариантами таким образом, чтобы максимизировать ожидаемую выгоду, когда свойства каждого выбора известны лишь частично во время распределения и могут стать более определёнными по прошествии времени или путем выделения ресурсов для выбора.

При реализации алгоритма многоруких бандитов с использованием Томпсоновского сэмплирования потребуется использование некоторых рассуждений из теории вероятностей. Приведем в этом разделе необходимые теоретические сведения.

Теорема 1 (Теорема Байеса)**.**

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

P(A) — априорная вероятность гипотезы A;

$P(A \mid B)$ *— вероятность гипотезы A при наступлении события B (апостериорная вероятность);*

$P(B \mid A)$ *— вероятность наступления события B при истинности гипотезы A;*

P(B) — полная вероятность наступления события B.

Определение 1. *Рассмотрим задачу поиска распределения случайной величины θ. Тогда по теореме Байеса (1):*

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{p(x)}$$

Если апостериорная семейство распределений $p(\theta \mid x)$ принадлежит тому же семейству, что и априорное $p(\theta)$, то распределение $p(\theta)$ называется ***сопряженным априорным распределением*** *для для семейства правдоподобий $p(x \mid \theta)$.*

Утверждение 1. *Сопряженным априорным распределением для распределения Бернулли* $Be(\theta)$ *является Бета-распределение* $Beta(\alpha, \beta)$.

Доказательство. Представим распределение Бернулли, как обычно это делается, через подбрасывание монеты.

Предполагаем, что априорное распределение $p(\theta)$ – Бета-распределение с параметрами α и β:

$$p(\theta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha, \beta)}$$

Пусть сделано уже $k+t$ подбрасываний, среди которых выпало k орлов. Тогда запишем теорему Байеса:

$$p(\theta \mid (k,t)) = \frac{p((k,t) \mid \theta)p(\theta)}{p((k,t))} = \frac{\binom{k+t}{k}\theta^k(1-\theta)^t\left(\theta^{\alpha-1}(1-\theta)^{\beta-1}\right)}{\left(\int_0^1 \frac{\binom{k+t}{k}x^{k+\alpha-1}(1-x)^{t+\beta-1}}{B(\alpha,\beta)}\,dx\right)B(\alpha,\beta)} = \frac{\theta^{k+\alpha-1}(1-\theta)^{t+\beta-1}}{B(k+\alpha, t+\beta)}$$

Это говорит о том, что апостериорное распределение – тоже Бета распределение. ■

Замечание 1. *Из доказательства утверждения 1 прямо следуют правила обновления параметров:*

Если было делано $k+t$ *подбрасываний, среди которых выпало* k *орлов (успехов), а параметры априорного Бета-распределения равны* α *и* β*, то параметры апостериорного Бета-распределения будут равны:* $\alpha + k$ *и* $\beta + t$.

Утверждение 2. *Сопряженным априорным распределением для нормального распределения* $N(\mu, \sigma^2)$ *с известным математическим ожиданием является Гамма-распределение* $Gamma(\alpha, \beta)$.

Доказательство. Обозначим $\theta = \frac{1}{\sigma^2}$ - параметр, для которого оцениваем распределение.

Предполагаем, что априорное распределение $p(\theta)$ – Гамма распределение с параметрами α и β:

$$p(\theta) = \frac{\theta^{\alpha-1}e^{-\theta\beta}\beta^\alpha}{\Gamma(\alpha)}$$

Тогда пусть сделан ещё один шаг сэмплирования θ и получено значения θ_1. Запишем апостериорное распределение:

$$p(\theta \mid \theta_1) = \frac{p(\theta_1 \mid \theta)p(\theta)}{p(\theta_1)} = \frac{\frac{\sqrt{\theta}}{\sqrt{2\pi}} e^{-\theta\frac{(\theta_1-\mu)^2}{2}} \frac{\theta^{\alpha-1}e^{-\theta\beta}\beta^\alpha}{\Gamma(\alpha)}}{\int_0^\infty \frac{\beta^\alpha x^{\alpha-\frac{1}{2}} e^{-x\left(\frac{(\theta_1-\mu)^2}{2}+\beta\right)}}{\sqrt{2\pi}\Gamma(\alpha)}\, dx} =$$

$$= \frac{\theta^{\alpha-\frac{1}{2}} e^{-\theta\left(\frac{(\theta_1-\mu)^2}{2}+\beta\right)}}{\int_0^\infty x^{\alpha-\frac{1}{2}} e^{-x\left(\frac{(\theta_1-\mu)^2}{2}+\beta\right)}\, dx} = \left[t = x\left(\frac{(\theta_1-\mu)^2}{2}+\beta\right)\right] =$$

$$= \frac{\theta^{\alpha-\frac{1}{2}} e^{-\theta\left(\frac{(\theta_1-\mu)^2}{2}+\beta\right)}}{\int_0^\infty \left(\frac{t}{\left(\frac{(\theta_1-\mu)^2}{2}+\beta\right)}\right)^{\alpha-\frac{1}{2}} e^{-t}\, d\left(\frac{t}{\left(\frac{(\theta_1-\mu)^2}{2}+\beta\right)}\right)} = \frac{\left(\frac{(\theta_1-\mu)^2}{2}+\beta\right)^{\alpha+\frac{1}{2}} \theta^{\alpha-\frac{1}{2}} e^{-\theta\left(\frac{(\theta_1-\mu)^2}{2}+\beta\right)}}{\int_0^\infty t^{\alpha-\frac{1}{2}} e^{-t}\, dt} =$$

$$= \frac{\left(\frac{(\theta_1-\mu)^2}{2}+\beta\right)^{\alpha+\frac{1}{2}} \theta^{\alpha-\frac{1}{2}} e^{-\theta\left(\frac{(\theta_1-\mu)^2}{2}+\beta\right)}}{\Gamma(\alpha+\frac{1}{2})} = \frac{\theta^{\alpha'-1}e^{-\theta\beta'}\beta'^{\alpha'}}{\Gamma(\alpha')}$$

■

Замечание 2. *Из доказательства утверждения 2 прямо следуют правила обновления параметров. После сделанного шага параметры Гамма-распределения изменяются следующим образом:*

$$\begin{cases} \alpha_{i+1} &= \alpha_i + \frac{1}{2}, \\ \beta_{i+1} &= \beta_i + \frac{(x_i-\mu)^2}{2}, \end{cases} \text{ где } x_{i+1} \text{ – полученное при сэмплировании значение} \tag{1}$$

Введение

В настоящее время методы машинного обучения широко применяются в различных сферах бизнеса с целью оптимизации, усовершенствования и создания новых бизнес-процессов. Одной из важных областей является установление цены на товар. Благодаря алгоритмам машинного обучения стало возможным провести оптимизацию цены в зависимости от цели ценообразования, такой как увеличение выручки или прибыли.

Динамическое ценообразование является распространенной практикой в нескольких отраслях, таких как гостиничный бизнес, туризм, развлечения, розничная торговля, электроэнергетика и общественный транспорт. Каждая отрасль использует свой подход к динамическому ценообразованию, основанный на индивидуальных потребностях и спросе на

продукт.

В динамическом ценообразовании можно выделить несколько классических этапов:

- Выявление эластичности через сбор и анализ данных;
- Определение модели (часто параметрической) зависимости спроса от цены;
- Генерация оптимальной цены, через оптимизацию целевой метрики.

Однако, такие методы не учитывают изменения экономической ситуации на рынке, в то время как она значительно влияет на покупателей.

Выбор алгоритма для решения задачи динамического ценообразования может основываться на идее нахождения оптимальной цены в реальном времени с учетом текущих изменений. Один из таких алгоритмов - алгоритм многоруких бандитов.

В последнее время на рынке все чаще встречаются решения для моделирования и оптимизации стоимости объектов недвижимости. Они позволяют компаниям-застройщикам сократить влияние человеческого фактора на прибыль, увеличить конкурентоспособность и снизить трудозатраты на процесс формирования стоимости.

Обзор и сравнительный анализ литературы

Была изучена теория по теме динамического ценообразования, реализована базовая модель предсказания цены на основе признаков объекта недвижимости ([5]). Далее были рассмотрены решения, использующие алгоритм многоруких бандитов для различных задач ценообразования [12], [7], [1], [3], [4]. Для понимания специфики рынка недвижимости было рассмотрено исследование [2]. Ниже опишем наиболее распространенные методы решения схожих задач.

Неформально идею алгоритма можно сформулировать так. Есть k ручек (рычагов), в каждый момент времени агент дёргает за одну из них и получает некоторый выигрыш, который сэмплируется из заранее заданного, но неизвестного агенту распределения. Цель агента: получить наибольший выигрыш за T шагов.

В литературе были выделены следующие варианты алгоритма:

- Жадный алгоритм: дёрнуть каждую из ручек m раз, после чего продолжать дёргать ту, у которой средний выигрыш максимален.
- ε-жадный алгоритм: с вероятностью $1 - \varepsilon$ выбираем ручку с максимальным средним выигрышем на данный момент, а с вероятностью ε - случайную.

- Алгоритм UCB (upper confidence bound): этот алгоритм высчитывает верхние доверительные границы для каждой из ручек, затем выбирает ту, у которой граница наибольшая. Проблема здесь заключается в том, что вычисление доверительных границ обычно очень затруднено.

- Алгоритм Томпсоновского сэмплирования: задаётся априорное распределение, которое будет обновляться на каждой итерации алгоритма. Идея заключается в том, что алгоритм выбирает каждую из ручек с той вероятностью, с которой она является оптимальной.

Алгоритмы будут подробнее рассмотрены в разделе 2.5, для ε-жадного (2.5.1) алгоритма и алгоритма Томпсоновского сэмплирования (2.5.2, 2.5.3) приведены реализации. Была изучена и протестирована реализация алгоритма многоруких бандитов из [12] - Томпсоновское сэмплирование. Результаты статьи были воспроизведены. Алгоритм действительно работает эффективно на сгенерированных авторами данных.

Для оценки моделей были выбраны стандартные метрики, которые описаны в таблице 1. Для оценки качества цены более показательной является метрика MAPE, в то время, как для оценки спроса наибольшее внимание уделялось MSE.

Таблица 1: Метрики оценки качества.

Метрика	Название	Формула
MAE (mean absolute error)	Средняя абсолютная ошибка	$\frac{\sum_{i=1}^{n} \lvert y_i - f(x_i) \rvert}{n}$
MSE (mean squared error)	Среднеквадратическая ошибка	$\frac{\sum_{i=1}^{n} (y_i - f(x_i))^2}{n}$
MAPE (mean absolute percentage error)	Средняя абсолютная ошибка в процентах	$\frac{100\%}{n} \sum_{i=1}^{n} \lvert \frac{y_i - f(x_i)}{y_i} \rvert$

1 Сбор данных

1.1 Анализ и выбор технологий

Была изучена теория [10] [9] и рассмотрены различные решения и подходы к сбору публичных данных, не защищенных законом об авторских правах с веб-ресурсов. Из основных методов можно выделить:

- Исследование XHR запросов сайта и вызова их в своем коде. XHR (XMLHttpRequest) запросы являются важной частью разработки веб-приложений, позволяя взаимодействовать с сервером без необходимости перезагрузки страницы. Это API предоставляет возможность отправлять HTTP-запросы на сервер и обрабатывать полученные ответы в асинхронном режиме. XHR запросы используются для множества задач, включая загрузку данных с сервера, отправку данных на сервер, обновление информации на странице без перезагрузки и другие сценарии, где требуется асинхронное взаимодействие с сервером.

- Исследование HTML документа, отправляемого сайтом, на наличие JSON документа с нужными нам данными. Зачастую, при отсутствии необходимости в постоянном обновлении информации на сайте, разработчики загружают данные сразу вместе с HTML файлом, без использования XHR запросов. Таким образом можно получить нужные данные из JSON документа.

- В некоторых случаях методы, упомянутые выше, могут оказаться неэффективными из-за наличия различных механизмов защиты, установленных на веб-сайтах. Для преодоления этих ограничений и обеспечения автоматизации браузера часто применяются специализированные технологии, такие как Selenium, Puppeteer, PlayWright и другие.

  Автоматизация браузера – это процесс, при котором программное обеспечение имитирует действия пользователя в веб-браузере. Она позволяет разработчикам выполнять автоматические тесты, собирать данные или взаимодействовать с веб-сайтами, обходя ограничения, связанные с обычными HTTP запросами.

- Парсинг HTML файла является методом, который часто используется в ситуациях, когда статический анализ HTML кода недостаточен или невозможен из-за динамической природы страницы. При многократных запросах к веб-сайту возрастает вероятность столкнуться с A/B тестированием, что может сказаться на правильной работе алгоритма или обработке данных.

A/B тестирование – это методика, применяемая в маркетинге и веб-разработке, где пользователи разделяются на группы, и в зависимости от группы им предлагаются различные варианты пользовательского интерфейса или функциональности. Это позволяет оценить влияние изменений на поведение пользователей и принять обоснованные решения для улучшения производительности, конверсии и пользовательского опыта.

В случае многократных запросов к сайту, когда он применяет A/B тестирование, может возникнуть проблема с непостоянством HTML кода. Вариации A и B могут иметь разные версии HTML, что может повлиять на правильность алгоритма, который полагается на определенную структуру и содержание страницы. Это может привести к непредсказуемым результатам и ошибкам в обработке данных.

В ходе исследований была изучена теория контейнеризации и значительная информация о Docker, одной из наиболее популярных платформ контейнеризации. Исследование включало анализ основных принципов контейнеризации, в том числе изоляцию, портативность и масштабируемость, а также роль Docker в достижении этих преимуществ. Были изучены концепции образов и контейнеров, а также ключевые компоненты Docker, такие как Docker Daemon и Docker CLI.

Docker — это открытая платформа, предоставляющая среду для создания, развертывания и управления приложениями с использованием контейнеризации. Она предоставляет изолированные и легковесные контейнеры, в которых можно запускать приложения и их зависимости на различных операционных системах.

Основные компоненты Docker

Docker состоит из нескольких ключевых компонентов, каждый из которых играет важную роль в процессе создания и управления контейнерами.

- **Образы (Images):** Образы Docker являются основными строительными блоками. Они представляют собой снимки состояния контейнера и содержат все необходимые компоненты приложения и его зависимости. Образы создаются с использованием Dockerfile – текстового файла, в котором определяются инструкции для сборки образа.

- **Контейнеры (Containers):** Контейнеры Docker являются запущенными экземплярами образов. Они обеспечивают изоляцию и независимость приложений и их зависимостей, позволяя запускать приложения в однородной среде, независимо от хост-системы.

- **Docker Daemon:** Docker Daemon является фоновым процессом, отвечающим за управление Docker-сервером и контейнерами. Он обрабатывает команды Docker API, управ-

ляет образами, контейнерами и другими ресурсами Docker.

- **Docker CLI:** Docker CLI (Command Line Interface) представляет собой интерфейс командной строки, позволяющий разработчикам взаимодействовать с Docker. Он предоставляет набор команд для создания, запуска, управления и мониторинга контейнеров.

Преимущества Docker

Использование Docker предоставляет ряд значительных преимуществ для разработки и развертывания приложений:

- **Портативность:** Docker контейнеры являются портативными и независимыми от операционной системы или хост-системы. Приложения, упакованные в контейнеры, могут быть запущены на любой совместимой с Docker платформе без необходимости изменения кода или конфигурации.

- **Изоляция и безопасность:** Контейнеры Docker обеспечивают изоляцию приложений друг от друга и от хост-системы. Это позволяет устранить возможные конфликты зависимостей и обеспечить безопасное развертывание и выполнение приложений.

- **Масштабируемость:** Docker позволяет гибко масштабировать приложения, добавлять и удалять контейнеры в зависимости от нагрузки. Это обеспечивает высокую доступность и возможность горизонтального масштабирования приложений.

- **Управление зависимостями:** Docker упрощает управление зависимостями и конфигурацией приложений. Образы Docker включают все необходимые компоненты, что облегчает развертывание и повторяемость в различных средах.

Области применения Docker

Docker нашел широкое применение в различных областях:

- **Разработка и тестирование:** Docker облегчает создание и управление тестовыми окружениями, что ускоряет процесс разработки и обеспечивает надежность тестирования.

- **Контейнеризация микросервисов:** Docker позволяет упаковывать и запускать отдельные компоненты микросервисной архитектуры в контейнерах, обеспечивая гибкость и масштабируемость системы.

- **Облачные вычисления:** Docker облегчает развертывание и управление приложениями в облачных средах, таких как Amazon Web Services (AWS) или Microsoft Azure.

- **Континуальная интеграция и доставка:** Docker позволяет создавать и развертывать образы с помощью инструментов для непрерывной интеграции и доставки, таких как Jenkins или GitLab CI/CD.

На основе проведенного анализа и углубленного изучения контейнеризации и технологии Docker, было принято решение упаковать наш проект в контейнеры Docker. Это решение было принято ввиду ряда преимуществ, которые Docker предлагает. Первоначально, использование Docker обеспечивает нам высокую портативность и совместимость, позволяя запускать наш проект на различных платформах и операционных системах с минимальными изменениями. Кроме того, Docker обеспечивает изоляцию приложения и его зависимостей, что гарантирует надежность и предотвращает конфликты между компонентами проекта. Наконец, использование Docker упрощает управление зависимостями и конфигурацией проекта, облегчая развертывание и повторяемость в различных средах. Исходя из всех этих факторов, мы пришли к выводу, что упаковка нашего проекта в Docker-контейнеры будет полезной и эффективной стратегией для разработки и развертывания утилит для парсинга.

Также была изучена теория непрерывной интеграции и доставки (CI/CD) и различные методы ее применения. Исследование включало анализ основных принципов и концепций CI/CD, включая автоматизацию сборки, тестирования и развертывания приложений. Были изучены различные подходы к настройке CI/CD, такие как интеграционные пайплайны, автоматическое тестирование, непрерывная доставка и развертывание контейнеров. В результате изучения вышеперечисленных технологий было приобретено понимание принципов и методов применения CI/CD, что позволило принять информированное решение о настройке этого процесса для проекта.

CI/CD играет ключевую роль в разработке программного обеспечения, обеспечивая эффективность и надежность в процессе развертывания приложений. Он позволяет автоматизировать сборку, тестирование и доставку кода, ускоряя процесс разработки и обеспечивая быструю обратную связь о возможных проблемах. CI/CD также помогает минимизировать риски и ошибки, обнаруживая и исправляя их на ранних этапах разработки.

CI/CD находит широкое применение в различных областях разработки программного обеспечения, включая:

- **Разработка и тестирование:** CI/CD упрощает и автоматизирует процессы сборки, тестирования и интеграции кода, позволяя командам разработчиков быстро выявлять и исправлять ошибки.

- **Контейнеризация и микросервисы:** CI/CD облегчает развертывание и управление контейнеризированными приложениями и микросервисами, обеспечивая непрерывную доставку и интеграцию новых изменений.
- **Облачные вычисления:** CI/CD позволяет автоматизировать развертывание и управление приложениями в облачных средах, обеспечивая масштабируемость и гибкость.

На сегодняшний день существует несколько платформ, которые предлагают возможности для настройки CI/CD. Некоторые из наиболее популярных вариантов включают Jenkins, GitLab CI/CD, Travis CI, CircleCI и TeamCity. Каждая из этих платформ имеет свои особенности и преимущества, и выбор платформы должен основываться на конкретных требованиях проекта.

В нашем проекте было принято решение использовать Jenkins в качестве платформы для настройки CI/CD. Это решение было обусловлено несколькими факторами. Во-первых, Jenkins является одним из самых популярных и широко используемых инструментов CI/CD. Он обладает богатым набором функций, гибкой конфигурацией и поддерживает большое количество плагинов и интеграций. Во-вторых, Jenkins обладает масштабируемостью и надежностью, что важно для нашего проекта, который предполагает постоянное развертывание и интеграцию новых изменений. Наконец, Jenkins предоставляет широкие возможности для настройки автоматических тестов, уведомлений и мониторинга, что позволяет нам обеспечить качество и стабильность проекта.

В итоге, основываясь на оценке требований проекта и учете преимуществ и возможностей различных платформ, был осуществлен выбор в пользу Jenkins в качестве платформы для настройки CI/CD.

1.2 Архитектура и реализация парсинга веб-сайтов

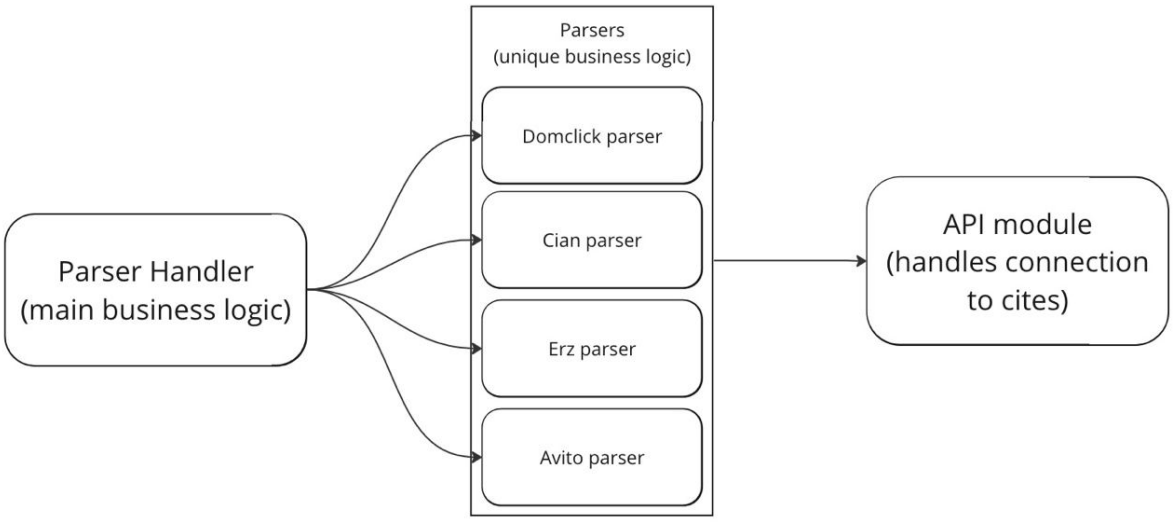


Рис. 1: Графическое представление архитектуры утилиты сбора данных

В архитектуре утилиты парсинга можно выделить 3 основные части:

- **Parser Handler** – описание основной бизнес-логики утилиты, одинаковой для всех парсеров. В нее входит запуск обхода сайта, построение материала для парсинга, запуск парсинга и запись полученных данных в файловую систему.
- **Parser fabric** – Уникальная логика для отдельных ресурсов, на которых осуществлялся сбор данных. В нее входит реализация обхода сайта, а также извлечение и обработка данных.
- **API module** – Модуль, ответственный за поддержание стабильного подключения к сайту, избегания блокировок и предоставление удобного интерфейса для получения нужных данных из бизнес-логики утилиты.

Плюсы данной архитектуры в простоте изменения количества парсеров. Добавление или удаление некоторых из них никак не повлияет на работоспособность системы. Отдельный модуль для подключения к сайтам позволяет быстро реагировать на блокировки, связанные с изменением в защитной системе сайта. Выделение общей логики для всех сайтов позволяет избавиться от дублирования кода и облегчает процесс реализации логики для отдельных ресурсов, при этом она устойчива к изменениям в работе сайтов. Исходя из данных преимуществ данной архитектуры было принято решение реализовать именно ее.

Конфигурация проекта

Создание конфигурации для проекта в формате JSON файла является привлекательным выбором по ряду причин. Во-первых, JSON является легким и удобочитаемым форматом данных, который позволяет представлять информацию в виде пар ключ-значение. Это обеспечивает простоту в создании и обработке конфигурационных файлов. Во-вторых, JSON является широко поддерживаемым и распространенным форматом данных в различных программных платформах и языках программирования. Благодаря этому, JSON файлы могут быть легко использованы и взаимодействовать с другими компонентами проекта. Кроме того, JSON обладает гибкостью и расширяемостью. В нем можно представлять сложные структуры данных, использовать вложенные объекты и массивы, что позволяет организовывать конфигурацию в удобном и иерархическом формате. Это особенно полезно, когда проект имеет разнообразные настройки и параметры, требующие структурированного представления. Для каждого из парсеров имеется возможность конфигурации следующих параметров:

- Название выходного файла с данными
- Получаемые из ресурсов данные и правила их получения. Данный параметр позволяет добавить или убрать нужную характеристику из выходного файла, не меняя код. Это упрощает и ускоряет работу с данной утилитой.
- Настройка фильтров и остальных параметров для обхода сайта. Зачастую пользователь сайта не имеет возможности просмотреть все объявления, расположенные на сайте, отвечающие его запросам (ошибка 404 или перенаправление на 1 страницу при переходе на страницы после определенной и т.д.). Для устранения данной проблемы было принято решение использовать набор непересекающихся по предложениям фильтров. В проекте настроена их удобная конфигурация. Фильтры и множества фильтров можно задать в конфигурационном файле, используя удобочитаемый синтаксис генераторов языка Python.
- А также остальные параметры, присущие отдельным парсерам.

1.3 Описание наборов данных

Данные, использующиеся для обучения и проверки качества итоговой модели были собраны с площадки Avito. Собирались только предложения от застройщиков в пределах Москвы. Последовательно использовалась фильтрация по непрекращающимся интервалам цен. Описание параметров, получаемых с данного ресурса:

Таблица 2: Описание параметров, получаемых с Avito.

Название	Описание	Примеры значений
Балкон или лоджия	Наличие балкона и/или лоджии	
Окна	Направление окон в квартире	во двор; на улицу; во двор на солнечную сторону;во двор, на улицу; ...
Отделка	Отделка в квартире	чистовая; предчистовая; ...
Название новостройки		
Официальный застройщик		
Тип дома		блочный; кирпичный; монолитно-кирпичный; панельный; ...
Этажей в доме		
Цена		
Метро	Список ближайших станций метро и времени дороги до них	
Общая площадь		
Площадь кухни		
Жилая площадь		
Этаж	Этаж, на котором находится квартира	
Высота потолков		
Срок сдачи	Предполагаемый срок сдачи новостройки	
Санузел	Тип санузлов в квартире	раздельный; совмещенный; ...
Пассажирский лифт	Количество пассажирских лифтов в доме	
Грузовой лифт	Количество грузовых лифтов в доме	

Двор	Инфраструктура во дворе дома	детская площадка; закрытая территория; спортивная площадка
Парковка	Тип парковки у дома	открытая во дворе; за шлагбаумом во дворе; наземная многоуровневая; ...
Тип комнат		изолированные; смежные; ...
Ремонт	Тип ремонта в квартире	дизайнерский, косметический, евро, без ремонта, ...
Техника	Техника, установленная в квартире	посудомоечная машина; холодильник; кондиционер; стиральная машина
В доме	Общие удобства дома	консьерж; мусоропровод
Теплый пол	Наличие теплого пола в квартире	
Мебель	Установленная в квартире мебель	
Просмотры	Количетсво просмотров на сайте на момент сбора данных	

2 Построение модели

Изначально данная работа была посвящена алгоритму многоруких бандитов, и должна была представлять собой его реализацию. Однако, в процессе разработки и исследования было решено добавить некоторые составные части, которые улучшают работу итоговой модели.

Решение состоит из следующих частей: модель для предсказания первичной цены по описанию квартиры, модель для предсказания спроса по описанию квартиры по её цене, блок обработки данных и непосредственно алгоритм многоруких бандитов, который выдает ответ - цену для каждого объекта (для последнего шага реализовано несколько вариантов, которые будет описаны далее). Перед запуском основной части алгоритма обучаются модели предсказания цены и спроса, их веса сохраняются для дальнейшего использования. При запуске алгоритма модели инициализируются сохраненными весами. Данные передаются в формате описанном в таблице 2. Также в момент запуска происходит реализация необходимых параметров: T – количество шагов в алгоритме многоруких бандитов, n – количество объектов недвижимости (квартир), m – количество вариантов цены (ручек), для каждого объекта и другие, которые зависят от версии алгоритма и будут описаны далее.

Схема полученного решения представлена в виде псевдокода 1. Рассмотрим в отдельности каждую часть.

Algorithm 1 Схема полученного решения

input: Множество объектов недвижимости (данные), значения параметров
Обработка данных: 2.1
Предсказание первичных цен: 2.2 ▷ инициализация вектора цен
Генерация ручек 2.5
for step in range(T) **do** ▷ шаг алгоритма многоруких бандитов: 2.5
    Расчет нового вектора цен
    Предсказание спроса для полученных цен: 2.3
    Оценка прибыли
    Обновление параметров
end for

2.1 Обработка данных

На данном этапе происходит обработка данных и приведение их к виду, которые способна обработать выбранная модель, очистка данных от выбросов, создание новых признаков для улучшения качества.

```
def prepare_all(data):
    data = shuffle(data)
    data = parse_yard_json(data)
    set_to_left = set(...) - delete field if not in set_to_left
    data = parse_date(data)
    keyrate = pd.read_csv(...)
    here some preperations
    currency = pd.read_csv(...)
    here some preparations - USD and EUR dataframes created
    data = parse_currency(data, 'USD', USD)
    data = parse_currency(data, 'EUR', EUR)
    list_for_drop = [...] - drop if in list
    list_for_factorize = [...] - factorize if in list
    data['days'] = data['days'].replace(0, 0.5)
    data['view_av'] = data['views'] / data['days]
    data.drop(data[data['view_av'] > 500].index, inplace=True)
    data.drop(data[data['area'] > 5000].index, inplace=True)
    del data[...]
    data = parse_subways_json(data)
    return data
```

Рассмотрим функцию, которая осуществляет обработку. Первый этап (сразу после

очистки от типа NaN) – удаление ненужных признаков, тех, что не используются моделью. (Сами списки довольно велики, поэтому здесь опускаем их содержание.) Далее происходит обработка текстовых категориальных признаков, так как выбранная модель предсказания спроса не принимает текстовые объекты. Также это положительно влияет на качество модели.

Следующий этап - добавление данных о курсе валюты и ключевой ставке на момент публикации объявления о квартире (при использовании клиентом это будет желаемая дата продажи). Они позволяют учитывать общую экономическую ситуацию на рынке.

Далее происходит подсчет среднесуточного количества просмотров, так как изначально у нас есть данные лишь об общем количестве. Последний этап – преобразование некоторых признаков, а именно инфраструктура и метро. Для метро был протестирован принцип target encoding, но он не улучшил качество моделей, поэтому и для метро, и для инфраструктуры был выбран one-hot encoding. За эти действия отвечают отдельные функции. После обработки получается 333 признака (часть из них отвечает за географическое положение объекта, оно описывается ближайшими станциями метро).

2.2 Предсказание первичной цены

При использовании алгоритма многоруких бандитов требуется проинициализировать вектор стартовых цен. Обычно их инициализируют средним значением (например, так сделано в [12]). Однако, идея разработанного решения заключается в следующем: чем ближе стартовые значения к реальным, тем быстрее и точнее сойдется алгоритм. Для проверки этой гипотезы был проведен эксперимент.

На одних и тех же данных был запущен ε-жадный алгоритм 2.5.1, но в первом случае вектор цен был инициализировал значениями, которые предсказала модель, а во втором – средней ценой в обучающей выборке. Результат можно наблюдать в таблице 3. Можно наблюдать, что при сильном отличии продаж (на 39% больше), прибыль отличается не так сильно (на 30% больше), в то время, как метрика MAPE говорит о том, что итоговые цены сильно отличаются от реальных. Были рассмотрены предсказания алгоритма и сделан вывод, что при инициализации средним значением, получается сильное занижение цен, что может быть не выгодно заказчику.

При выборе модели были рассмотрены описания наилучших решений в соревновании [5]. Одно из решений с высоким рейтингом в своей основе содержало модель градиентного бустинга из библиотеки LightGBM. Было решено использовать её для предсказания стартовой

Таблица 3: Сравнение результатов экспериментов с инициализацией вектора цен

Эксперимент	Прибыль	Количество проданных квартир	MAPE
Предсказания модели	40713738681	2680	8.05%
Среднее значение	52991632536	3721	56.57%

цены. После подбора параметров такая архитектура показала достаточно высокое качество (таблица 4). Размер выборки, для которой проводилось обучение и тестирование - 20000 объектов.

Таблица 4: Метрики качества модели предсказания первичной цены.

MAE	540251.3	MAPE	3.32%

Также стоит обратить внимание на наиболее значимые для модели поля (таблица 5). Ожидаемо, наиболее важным факторов ценообразования является площадь квартиры (в том числе общая и жилая). Ближайшая станция метро здесь отвечает за расположение квартиры относительно города, поэтому также играет не последнюю роль. Остальные поля не перечислены в таблице, так как их вклад довольно мал.

Таблица 5: Наиболее значимые для предсказания цены моделью поля

	Поле	Вклад (%)
1	Общая площадь	40.926798
2	Жилая площадь	10.475430
3	Высота потолков	9.831439
4	Время пути до ближайшей станции метро	9.674080
5	Тип комнат	3.401647
6	Ближайшая станция метро	2.428353
7	Площадь кухни	2.323301
	Остальное	20.939032

2.3 Предсказание спроса на недвижимость

На каждом шаге алгоритма многоруких бандитов необходимо оценивать прибыль при заданных ценах. У нас не было возможности проводить эксперименты на реальных данных (например, в виде AB-тестирования, как это делалось в [7]), а результат работы предполагает готовый инструмент, поэтому возникла необходимость симулировать поведение покупателей на рынке недвижимости.

В литературе, например в [12], поведение покупателей генерировалось из заранее заданного распределения. Однако, чтобы приблизиться к реальному поведению покупателей на рынке было решено обучить модель, аналогичную той, что использовалась для предсказания первичной цены: градиентный бустинг из библиотеки LightGBM.

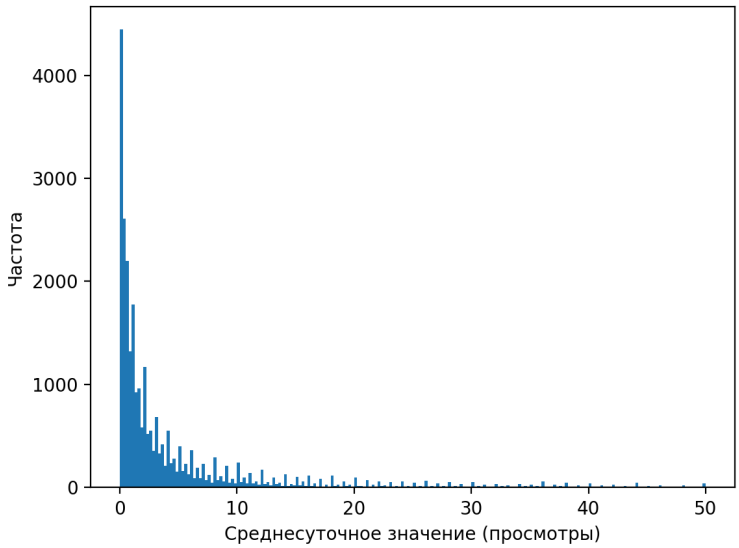


Рис. 2: Распределение среднесуточного количества просмотров на сайте

Сложной задачей был поиск статистики спроса на недвижимость для обучения. В качестве показателя спроса было выбрано среднесуточное количество просмотров на сайте, так как данных о покупках в открытом доступе нет. То есть, по описанию квартиры и её цене модель училась предсказывать среднесуточное количество просмотров. Распределение данной величины в полученных данных можно увидеть на гистограмме 2.

По результатам обучения были достигнуты значения метрик, указанные в таблице 6.

Таблица 6: Метрики качества модели предсказания спроса.

MAE	7.06	MSE	353.41

В этой главе также обратимся к рейтингу полей, которые наиболее сильно влияют на решение модели (таблица 7). Можно заметить, что предсказания достаточно сильно зависят от курсов валют, что соотносится с наблюдаемой на рынке ситуацией. Можно заметить, что цена имеет достаточно большой вес, но является далеко не определяющим фактором – её вклад незначительно превышает вклад полей с площадью. Это соответствует экономическим исследованиям [2], говорящим о том, что спрос на недвижимость не эластичен по цене, за исключением редких экстремальных периодов.

Таблица 7: Наиболее значимые для предсказания спроса моделью поля

	Поле	Вклад (%)
1	Курс Доллара США	10.017689
2	Цена	9.066184
3	Курс Евро	8.675113
4	Общая площадь	7.412777
5	Жилая площадь	7.069347
6	Этаж	6.791552
7	Площадь кухни	6.732922
8	Количество этажей в доме	3.589005
	Остальное	40.645411

2.4 Постановка задачи многорукого бандита

Дано:

- B – множество из n объектов недвижимости;
- T – период времени, на котором максимизируем прибыль;
- K – множество векторов с признаками для каждого объекта.

Необходимо построить алгоритм ценообразования, максимизирующий ожидаемый доход за период времени T.

Введем обозначения:

- $p_t = (p_{1,t}, \ldots, p_{n,t})$ - вектор цен;
- $D(p_t) = (d_{1,t}, \ldots, d_{n,t})$ - спрос на объекты при заданных ценах p_t.
- $R_t(p_t) = \sum\limits_{i \in B} D_{i,t}(p_t) \cdot p_{i,t}$ - функция прибыли.
- $p* =_p R_t(p)$ - оптимальные постоянные цены при известном распределении $D(p_t)$.

Задача: Построить алгоритм подбора цен p_t, максимизирующий выражение:

$$\sum_{t=1}^{T} R_t(p_t)$$

при условии, что распределение $D(p_t)$ изначально неизвестно.

2.5 Многорукие бандиты

Рассмотрим известные методы решения задачи многоруких бандитов. Напомним, дано множество из n игровых автоматов, у каждого из которых есть m вариантов действий (ручек, которые можно потянуть). На каждом шаге алгоритма для каждого из автоматов совершается одно из доступных действий (тянется одна из ручек), затем автомат выдает некоторое значение – выигрыш. Агенту неизвестно, каким образом определяется выигрыш. Задача состоит в том, чтобы оптимизировать суммарный выигрыш по всем автоматам.

Напомним необходимые для этого раздела понятия, которые уже упоминались в обзоре литературы.

- Сыграть ручку – выбрать определенный вариант действия.
- Алгоритм многоруких бандитов – алгоритм, который принимает решение о том, какие ручки сыграть у заданных автоматов.
- Выигрыш – значение, которое выдает автомат, когда в нем играется определенная ручка.

Существует множество вариантов решения задачи многоруких бандитов, среди прочих, как наиболее популярные, выделяются следующие: жадный алгоритм, ε-жадный алгоритм, алгоритм верхних доверительных границ, алгоритм Томпсоновского сэмплирования. Рассмотрим каждый из них и ещё некоторые в отдельности. Для простоты рассуждения, рассмотрим алгоритмы на примере алгоритма с одним автоматом.

Наиболее простым вариантом решения является жадный алгоритм. Агенту предлагается сыграть каждую ручку автомата некоторое фиксированное число раз (пусть k), после чего посчитать средний выигрыш для каждой из ручек. После этого на каждой итерации выбирать ту ручку, у которой средний выигрыш был наибольшим (можно несколько улучшить алгоритм, если обновлять среднее значение на каждом шаге). Преимуществом данного алгоритма является простота реализации. Однако, в этом случае также наблюдается ряд проблем. По той причине, что агенту заранее неизвестно распределение, из которого сэмплируется выигрыш у каждого из автоматов, а среднее значение далеко не всегда является качественной оценкой математического ожидания случайной величины, вывод, сделанный на основании среднего значения за первые km шагов, может привести к неоптимальному результату. Если также обратить внимание на более общий случай алгоритма, с большим числом автоматов, то можно заметить, что такой способ быстро сводит все к одну случаю, перебрав при этом небольшое количество вариантов, тем самым лишая себя возможности

найти наиболее выигрышную комбинацию. Можно сказать, что первые km шагов производится исследование автоматов, а далее – эксплуатация результатов [12].

Учитывая перечисленные выше проблемы, приходим к некоторой модификации жадного алгоритма, а именно к ε-жадному алгоритму. Он схож со своей изначальной версией, но имеет преимущество. Алгоритм заключается в следующем, все также делаем первые km шагов, сыграв по k раз каждую из m ручек, вычисляем средний выигрыш для каждой из них. Далее требуется зафиксировать некоторое $\varepsilon \in [0, 1]$ – это число послужит вероятностью, с которой при дальнейших шагах будет выбираться случайная ручка. То есть, на каждом шаге, с вероятностью ε будет выбираться случайная ручка (обычно ε выбирают небольшим), а с вероятностью $1-\varepsilon$ – ручка с наибольшим средним выигрышем. Стоит уточнить, что есть два варианта дальнейшего развития событий: можно не обновлять средний выигрыш у ручек, но более оптимальным решением является обновлять его каждый раз, когда играем ручку. Таким образом, сохраняется простота реализации, при этом увеличивается вариативность комбинаций: даже если на первых km шагах ручка проявила себя не наилучшим образом, далее есть вероятность, что она будет проверена снова. Тем самым, случайно упущенный вначале оптимальный вариант не будет окончательно утерян. Данный алгоритм всё ещё не является оптимальным, так оценивает распределение выигрышей только с точки зрения среднего значения. Хотя и может получать хороший результат. Его реализация приведена в разделе 2.5.1.

Далее речь пойдет о более сложных алгоритмах, решающих задачу многоруких бандитов.

Начнем с алгоритма softmax-исследования [8]. Он идейно близок ε-жадному алгоритму, однако, если в случае жадного алгоритма при выборе случайной ручки все варианты были равноправными, то теперь вероятность выбрать каждую из них определяется распределением Больцмана, а именно как описывает формула 2, где m_i - текущий средний выигрыш у ручки номер i, p_i - вероятность, с которой будет сыграна эта ручка, ε - коэффициент, регулирующий исследование ручек: чем больше его значение, тем ближе распределение к равномерному.

$$p_k = \frac{e^{\frac{m_k}{\varepsilon}}}{\sum\limits_{i=0}^{n} e^{\frac{m_i}{\varepsilon}}}, \tag{2}$$

Таким образом данный алгоритм является ещё одной модификацией жадного алгоритма, которая позволяет находит оптимальное соотношение между исследованием и экс-

плуатацией.

Рассмотрим также алгоритм верхнего доверительного интервала (UCB algorithm - Upper confidence bound). Этот алгоритм предпринимает попытку смотреть на распределение не только с точки зрения среднего значения, но также учитывать, насколько полученная информация соответствует действительности. Алгоритм предполагает вычисление доверительного интервала для среднего выигрыша. У каждого доверительного интервала существует верхняя и нижняя границы. Поэтому далее выбирается ручка с наибольшей верхней границей. После получения результата, доверительный интервал пересчитывается и действия повторяются. Данный алгоритм используется реже других из-за того, что эффективно оценить доверительный интервал зачастую бывает невозможно. Было решено не использовать данный алгоритм в работе.

Теперь перейдём к методу, который считается одним из наиболее оптимально решающих задачу о многоруком бандите - Байесовские бандиты [6]. В начале такого алгоритма задаётся априорное распределение (при отсутствии дополнительной информации его принято брать равномерным). Далее на каждом шаге распределение обновляется следующим образом. Для каждой ручки сэмплируется значение из текущего распределения, выбирается ручка, у которой результат наибольший. Агент играет выбранную ручку и получает от автомата некоторый выигрыш. С учётом полученного выигрыша происходит обновление распределения. Таким образом, с каждым шагом распределение у каждой ручки будет приближаться к реальному распределению выигрыша. Стоит заметить, что на первых шагах (это станет особенно заметно, когда мы рассмотрим реализацию с нормальным распределением) предпочтение отдается исследованию, то есть из-за того, что в начале каждой ручке доступен довольно широкий спектр значений (в случае нормального распределения за это отвечает дисперсия), практически любая из них может быть выбрана. Но ближе к концу алгоритма, неоптимальные ручки уже не смогут превосходить по значениям лучшие варианты. Таким образом, алгоритм можно будет завершать. В работе реализованы две версии данного алгоритма: для распределения Бернулли (2.5.2) и для нормального распределения (2.5.3).

Для начала в работе был реализован простой алгоритм, а именно модификация жадного – ε-жадный алгоритм. Затем фокус был смещен на байесовские варианты, так как по изученной литературе можно сделать вывод, что именно эти алгоритмы работают наиболее точно. Результаты и подробности реализации каждого из алгоритмов будут приведены далее в работе.

Перед тем, как приступать к рассмотрению деталей реализации шага алгоритма многоруких бандитов, стоит обратить внимание на ещё один немаловажный этап решения – гене-

рацию ручек (вариантов цены) для каждого объекта на основании предсказанной первичной цены. Идея этого этапа следующая: исходя из качества модели предсказания первичной цены, был сделан вывод, что полученные на этом этапе цены сопоставимы с настоящими. Поэтому основная задача алгоритма многоруких бандитов – рассмотреть близкие к предсказанным цены и выбрать наилучшую. Поэтому ручки сэмплируются из нормального распределения с математическим ожиданием, равным предсказанной цене и заранее заданной дисперсией (в экспериментах используется: 500000). Таким образом рассматриваются наиболее близкие к предсказанной цены, причем из характера нормального распределения следует, что большая часть ручек будет особенно близка к заданному математическому ожиданию. Это позволяет рассмотреть наиболее вероятные варианты цен.

Перейдем непосредственно к реализациям шага алгоритма многоруких бандитов.

2.5.1 ε-жадный алгоритм

Идея работы данного алгоритма была описана выше в 2.5. Ниже представлен шаг алгоритма в виде псевдокода 2. Рассмотрим также реализацию функции, которая рассчитывает суммарную прибыль в данном алгоритме: 3. В остальных реализациях используются схожие функции, поэтому далее на них останавливаться не будем. В этом разделе обратимся к выбору параметров, результатам экспериментов и значениям метрик.

Algorithm 2 Шаг ε-жадного алгоритма

input: Множество объектов недвижимости (данные), значения параметров, первичные цены, ручки
for flat in dataset **do** ▷ Делаем шаг для всех квартир
    Сэмплируем c из $Be(\varepsilon), \varepsilon \in (0, 1)$
    if c == 1 **then**
        Играем случайную ручку
    else
        Играем ручку с наибольшим текущим средним выигрышем
    end if
end for
Предсказание спроса для полученных цен: 2.3
Оценка прибыли
Обновление параметров (в том числе среднего выигрыша)

Algorithm 3 Подсчёт суммарной прибыли: с учетом количества продаж

input: данные по квартирам, текущие цены
Предсказываем среднесуточное количество просмотров (спрос) 2.3
Рассчитываем, сколько будет покупок
Считаем суммарную прибыть

Таблица 8: Параметры ε-жадного алгоритма многоруких бандитов.

Параметр	Имя переменной	Значение
Количество квартир	n	10000
Количество ручек	m	1000
Дисперсия для генерации вариантов цен	dispertion	500000
Количество просмотров в сутки, необходимое для продажи	selling_treshold	30
Количество шагов алгоритма	T	1000
Вероятность исследования	epsilon	0.3, 0.5, 0.8

Опишем выбранные параметры (таблица 8). В эксперименте считается, что на каждые 30 просмотров в сутки приходится покупка. Так как в открытом доступе нет данных о покупках, то такое значение было выбрано, так как среднее значение по квартирам существенно меньше – это видно на гистограмме 2, но и какой-то процент квартир должен быть продан. В данном случае по результатам эксперимента с симуляцией поведения пользователей с помощью модели 2.3 было продано около 25% квартир. В зависимости от данных клиента параметр будет меняться. Было рассмотрено различное количество шагов алгоритма и сделан вывод о том, что спустя 1000 шагов прибыль и количество проданных квартир останавливает рост. О дисперсии для генерации ручек уже говорилось ранее в работе, далее в экспериментах она будет сохранять данное значение.

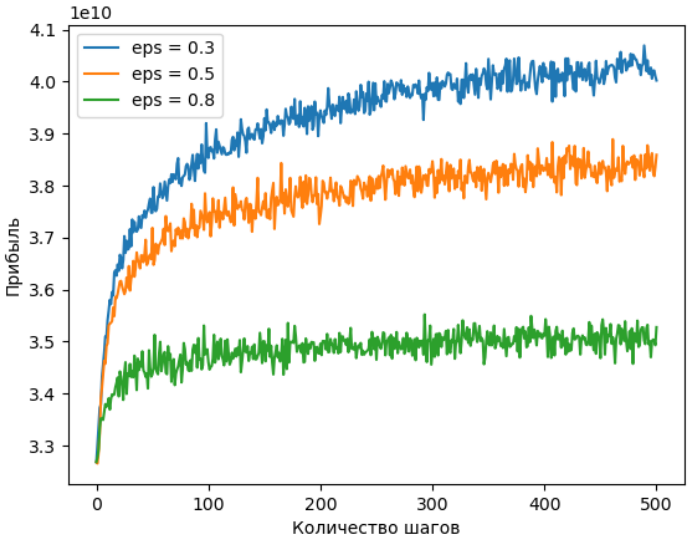
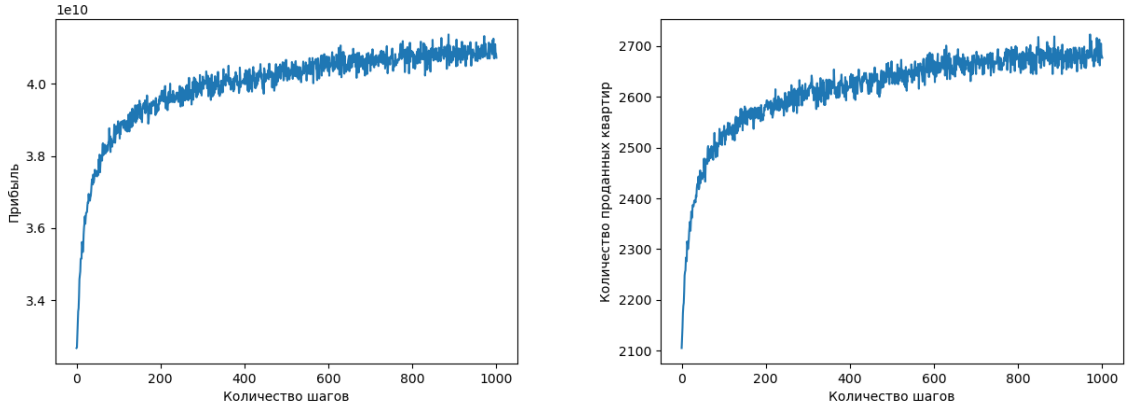


Рис. 3: Выбор параметра ε: сравнение вариантов.

Отдельный интерес представляет выбор ε. Этот параметр отвечает за соотношение исследования и эксплуатации в алгоритме. Было протестировано три варианта, которые можно у видеть в таблице 8. Результаты можно увидеть на графике 3. Для основного эксперимента

было выбрано значение 0.3, так как показало наилучший результат.



(a) Прибыль от продажи квартир (b) Количество проданных квартир

Рис. 4: Результаты работы ε-жадного алгоритма при $\varepsilon = 0.3$

Таблица 9: Метрики качества ε-жадного алгоритма алгоритма.

$\varepsilon = 0.5$	MAE	881599.31	MAPE	6.40%
$\varepsilon = 0.3$	MAE	1082075.28	MAPE	8.05%

Из таблицы 9 делаем вывод о том, что полученные цены достаточно близки к настоящим, при этом рост продаж и прибыли заметен на графике 4. Получается, что цены на сайте недалеки от оптимальных, однако, небольшим их изменением, всё же можно достичь увеличения суммарной прибыли по квартирам. То есть, алгоритм хорошо справляется со своей задачей. Недостатком такой реализации является неустойчивость – даже после 1000 шагов алгоритма значения продолжают колебаться. Проверим, улучшится ли ситуация в Байесовком варианте алгоритма.

2.5.2 Сэмплирование Бернулли-Томпсона

Напомним, что основной идеей алгоритма является предположение, что выигрыш каждой ручки можно приблизить некоторым распределением. В данном разделе рассмотрим наиболее простой случай [11]: выигрыш каждой ручки может принимать лишь значения 1 (продано) и 0 (не продано). То есть результат игры ручки можно представлять как сэмплирование из распределения Бернулли. Тогда попытаемся каждую ручку параметризовать Бета-распределением, так как по утверждению 1 оно является сопряженным апостериорным распределением для распределения Бернулли.

После определения типа сопряженного апостериорного распределения требуется задать исходные значения параметров (в данном случае это α и β – параметры Бета-распределения,

где α – количество успехов, то есть продаж, а β – число неудач). Положим их равными 1, это даст нам равномерное распределение, что соответствует действительности - изначально все ручки для нас равноправны. Далее на каждом шаге они будут обновляться по формуле 4, что следует из замечания 1. Весь шаг алгоритма описан в виде псевдокода 4.

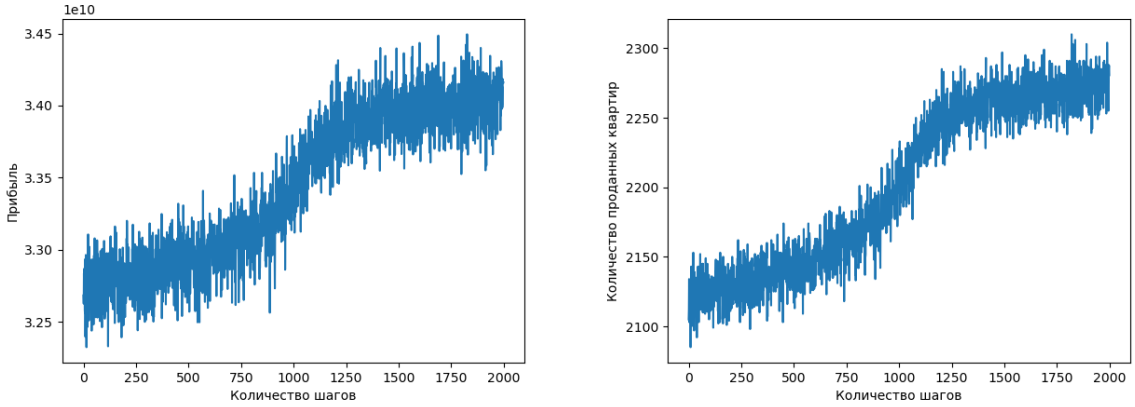
$$\begin{aligned} \alpha_{i+1} &= \alpha_i + x_{i+1}, \\ \beta_{i+1} &= \beta_i + (1 - x_{i+1}), \\ \text{где } x_{i+1} &= \begin{cases} 1, & \text{в случае продажи,} \\ 0, & \text{иначе.} \end{cases} \end{aligned} \tag{3}$$

Algorithm 4 Шаг алгоритма с сэмплированием Бернулли-Томпсона

1: **input:** Множество объектов недвижимости (данные), значения параметров, первичные цены, ручки
2: **for** flat in dataset **do** ▷ Делаем шаг для всех квартир
3: Сэмплируем значения для всех ручек из $Beta(\alpha, \beta)$
4: Играем ручку с наибольшим из полученных значений
5: **end for**
6: Предсказание спроса для полученных цен: 2.3
7: Оценка прибыли
8: Обновление параметров (α, β)

Параметры в большинстве своём были сохранены как в ε-жадном алгоритме. Единственным отличием явилось то, что после 1000 шагов в это реализации прибыль продолжала возрастать, как и количество продаж. Поэтому было решено сделать 2000 шагов. И как можно видеть на графике 5, этого количества шагов достаточно.

Результаты эксперимента можно наблюдать в таблице 10. Сразу заметим, то количество продаж и прибыль в этой реализации получилась меньше, чем для ε-жадной версии. Здесь могут быть две причины. Во-первых, судя по значению MAPE, а в последнем эксперименте оно меньше, полученные цены ближе к реальным, то есть шанс на неправдоподобное понижение или повышение в данном случае ниже. Этого говорит о том, что результатам нового алгоритма можно доверять с большей вероятностью. В то же время, так как алгоритм пытался приблизить выигрыши с помощью распределения Бернулли, которое учитывает не количество проданных квартир каждого типа, а лишь факт продажи, качество могло ухудшиться от недостаточно точной оценки. Стоит также заметить, что результаты данного алгоритма также неустойчивы, однако, в целом к двухтысячному шагу уже четко видна тенденция.



(a) Прибыль от продажи квартир (b) Количество проданных квартир

Рис. 5: Результаты работы алгоритма с сэмплированием Бернулли-Томпсона.

Таблица 10: Метрики качества алгоритма с сэмплированием Бернулли-Томпсона.

MAE	1017612.816	MAPE	7.42%

2.5.3 Сэмплирование Гаусса-Томпсона

Ещё одна версия алгоритма, которая была реализована – сэмплирование Гаусса-Томпсона. Структура алгоритма близка к описанной в разделе 2.5.2. Но если в предыдущем случае получаемый на каждом шаге выигрыш был параметризован распределением Бернулли, то теперь будет использовано нормальное распределение с известным средним.

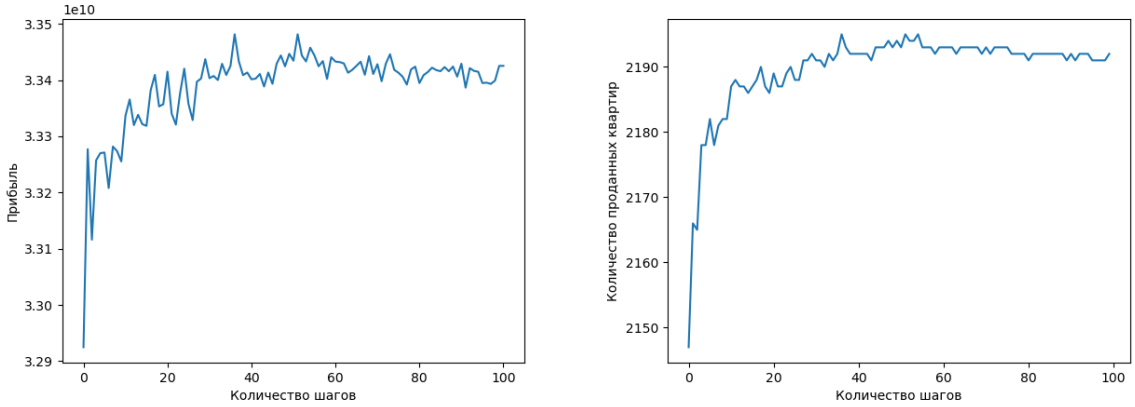
Было протестировано два варианта реализации. В первом из них в качестве выигрыша рассматривалась полученная прибыль по квартире данного типа, а во втором – предсказанное моделью 2.3 среднесуточное количество просмотров. Второй вариант выдал лучшее качество, поэтому далее будем рассматривать только его.

Исходя из рассуждений выше в качестве известного параметра математического ожидания ($N(\mu, \sigma^2)$, где μ – математическое ожидание, σ^2 – дисперсия) выбирается следующая величина. Как уже говорилось ранее, перед запуском алгоритма многоруких бандитов происходит предсказание первичной цены с помощью модели 2.2. В текущей реализации будет добавлен ещё один шаг: предсказание среднесуточного количества просмотров для этих цен. Именно эти значения будут приняты в качестве математического ожидания для ручек каждой из квартир. Так как спрос на квартиры практически не эластичен по цене, то это будет достаточно близкой к действительности оценкой.

Аналогично случаю с сэмплированием Бернулли будем использовать сопряженное априорное распределение. В качестве неизвестного параметра возьмём точность нормального распределения: $\tau = \frac{1}{\sigma^2}$. Тогда по утверждению 2 сопряженное распределение – это

Гамма-распределение. И согласно замечанию 2 обновление параметров происходит так:

$$\begin{cases} \alpha_{i+1} & = \alpha_i + \frac{1}{2}, \\ \beta_{i+1} & = \beta_i + \frac{(x_i - \mu)^2}{2}, \end{cases} \text{ где } x_{i+1} \text{ – полученный выигрыш (количество просмотров)} \quad (4)$$



(a) Прибыль от продажи квартир (b) Количество проданных квартир

Рис. 6: Результаты работы алгоритма с сэмплированием Гаусса-Томпсона.

Таблица 11: Метрики качества алгоритма с сэмплированием Гаусса-Томпсона.

MAE	1085491.746	MAPE	8.13%

Результаты эксперимента можно наблюдать на графике 6. Видно, что после 100 шагов алгоритма прибыль стабилизируется, хоть и продолжает немного колебаться. Это значительно быстрее, чем у предыдущих алгоритмов. В то же время из-за особенностей реализации время работы остается сравнимым с другими реализациями. А именно, для данного алгоритма был написаны классы BanditArm и Bandit для удобства чтения кода. Однако, при необходимости, можно отказаться от такой реализации в пользу работы с методами библиотеки NumPy, как в предыдущих реализациях. Рассмотрим реализацию классов. Класс BanditArm отвечает за реализацию одной ручки и содержит следующие поля: a – текущее значение параметра α, b – текущее значение параметра β, n – сколько раз на данный момент уже была сыграна данная ручка, price – цена, соответствующая данной ручке, m – μ, известный нам параметр нормального распределения (математическое ожидание). Для данного класса реализованы методы Sample – сэмплировать значение из текущего распределения и Play – сыграть ручку (то есть обновить параметры с выигрыша, переданного в метод). Теперь перейдём к описанию класса Bandit. У него есть такие поля: len_arms – количество ручек у

этого бандита, arms - набор ручек (объектов класса BanditArm), curr_play – номер ручки, которая играется у этого бандита на текущем шаге. Теперь рассмотрим методы: метод Step отвечает за выбор оптимальной на данный момент ручки, а метод Update – за обновление параметров всех ручек данного бандита.

Результаты

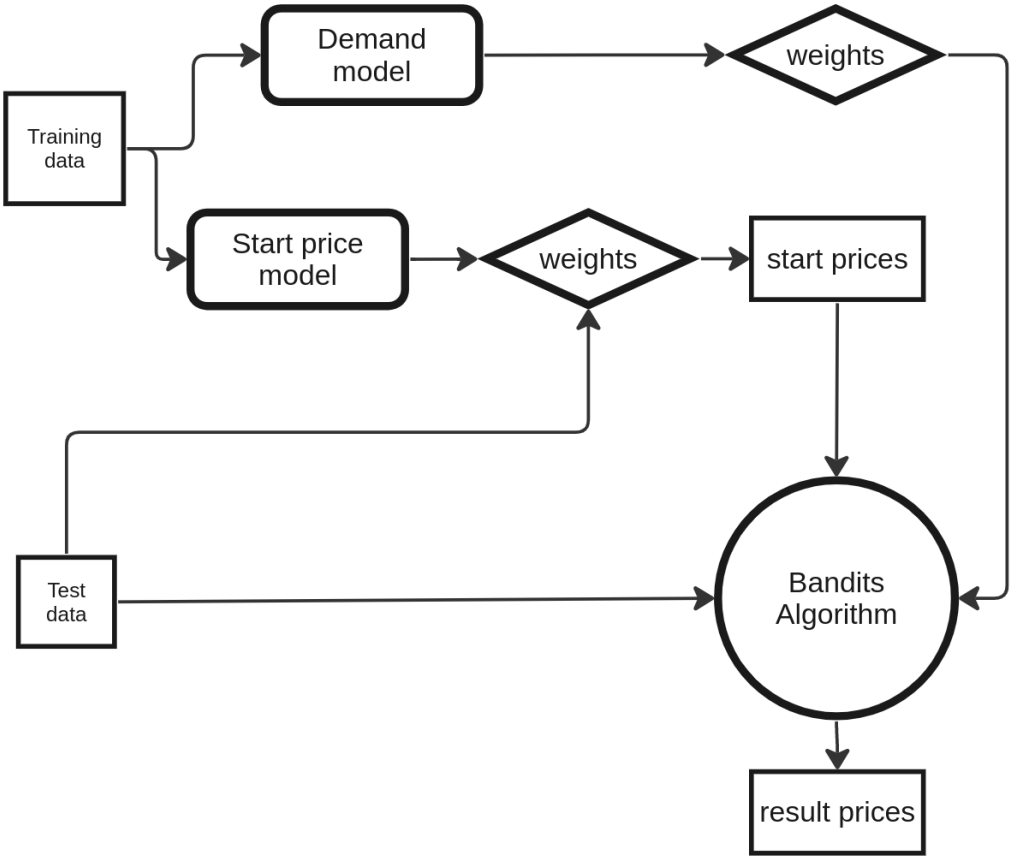


Рис. 7: Общая схема решения.

В данном исследовании была разработана утилита для парсинга данных, которая была упакована в докер-контейнер. Для обеспечения надежности и качества данной утилиты был использован подход CI/CD при помощи Jenkins. Полученная утилита была использована для сбора данных об объектах недвижимости, которые далее были использованы для обучения модели ценообразования.

В результате работы по ценообразованию было получено решение, изображенное на схеме 7. Обе вспомогательные модели были обучены и готовы к использованию. Отметим, что результаты были получены для следующих объёмов данных: модель предсказания спроса и модель предсказания первичной цены обучались на наборе из 20000 квартир, все реализации

алгоритма многоруких бандитов тестировались на 10000 объектах. Для алгоритма многоруких бандитов подготовлено три варианта реализации. По итогам тестирования каждого из них было проведено сравнение по различным параметрам, которые можно увидеть в таблице 12. Также все алгоритмы были успешно проверены на неправдоподобное завышение и занижение цен (график 8). Наиболее эффективным оказался ε-жадный алгоритм, так как он приносит наибольшие количество продаж и значение прибыли. Однако, как видно на графике 8, ему всё же свойственно занижение цен в некоторых случаях, а также получение большей части прибыли от наращивания объёма продаж более дешевых вариантов, в то время как для Байесовских бандитов такая проблема менее выражена. Поэтому использование той или иной архитектуры зависит от цели клиента: для быстрой продажи недвижимости с наибольшей выгодой стоит использовать ε-жадный алгоритм, в то время как для большей выгоды в долгосрочном периоде имеет смысл не занижать цену и воспользоваться одной из вероятностных реализаций. Для использования достаточно создать выборку данных о квартирах, для которых нужно определить цену, в формате, описанном в таблице 2. Далее задаются параметры, описанные в разделе 2.5.1 (с уточнением на количество шагов в таблице 12). После чего происходит запуск алгоритма многоруких бандитов.

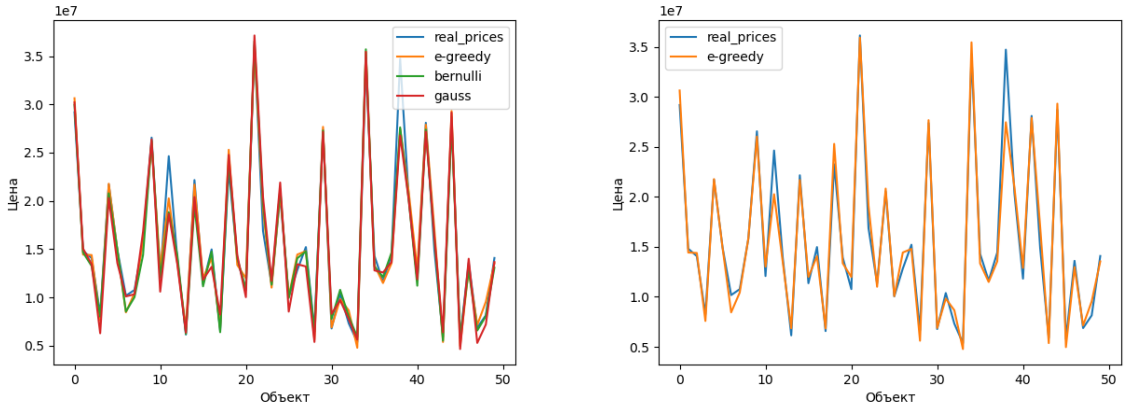


Рис. 8: Сравнение полученных цен с реальными.

Заключение

По итогам проделанной работы можно сделать вывод о том, что алгоритм многоруких бандитов может применяться в динамическом ценообразовании недвижимости несмотря на особенности данного рынка. Основной трудностью в таком подходе является симуляция поведения покупателей, так как данных о продажах нет в открытом доступе. В данной работе для этого использовалась бустинговая модель, обученная на статистике просмотров объявлений

Таблица 12: Сравнение реализаций алгоритма многоруких бандитов

Алгоритм	Время выполнения одного шага	Прибыль	Количество проданных квартир	Нужное число шагов	Время выполнения	MAPE
ε-жадный алгоритм 2.5.1	0.842 с	40713738681.22475	2680	1000	843.36 с	8.05%
Сэмплирование Бернулли-Томпсона 2.5.2	1.27 с	34160953495.073708	2287	2000	2547.05 с	7.42%
Сэмплирование Гаусса-Томпсона 2.5.3	33.66 с	33425342107.034	2192	100	3375.70	8.13%

на сайте. Также из-за того, что спрос на недвижимость не эластичен по цене, нет единственного варианта оптимальной цены, поэтому качество полученных результатов может быть оценено с погрешностью. Дальнейшим направлением развития работы может стать создание программного обеспечения для расчета цены недвижимости на основе разработанных алгоритмов.

Список литературы

[1] Elad Hazan и Kfir Levy. “Bandit convex optimization: Towards tight bounds”. В: *Advances in Neural Information Processing Systems* 27 (2014).

[2] Katarzyna Kobylińska Justyna Brzezicka. *An Analysis of the Income and Price Elasticity of Demand for Housing in View of Price Dynamics on the Residential Property Market*. https://www.researchgate.net/publication/357210551_An_Analysis_of_the_Income_and_Price_Elasticity_of_Demand_for_Housing_in_View_of_Price_Dynamics_on_the_Residential_Property_Market.

[3] *Mercari Price Suggestion Challenge*. https://github.com/pjankiewicz/mercari-solution.

[4] Jonas W Mueller, Vasilis Syrgkanis и Matt Taddy. “Low-rank bandit methods for high-dimensional dynamic pricing”. В: *Advances in Neural Information Processing Systems* 32 (2019).

[5] *Sberbank Russian Housing Market*. https://www.kaggle.com/c/sberbank-russian-housing-market.

[6] *Алгоритмы многоруких бандитов*. https://dyzzet.ru/a/multiarmed-bandits/.

[7] Руслан Гунаев. “Онлайн ценообразование с помощью структурированных многоруких бандитов”. В: *Москва* (2021).

[8] *Кто же такой этот многорукий бандит?* https://habr.com/ru/articles/689364/.

[9] Максим Кульгин. “10 инструментов, позволяющих парсить информацию с веб-сайтов”. В: *https://habr.com/ru/articles/340038/* (2021).

[10] Даниил Охлопков. “Парсинг сайтов”. В: *https://habr.com/ru/articles/579336/* (2021).

[11] *Сэмплирование Томпсона*. https://habr.com/ru/companies/domclick/articles/547258/.

[12] АА Харь, ЮВ Дорн и ВВ Стрижов. “Динамическое ценообразование с помощью томсоновского сэмплирования”. В: *http://www.machinelearning.ru/wiki/images/3/3e/Khar_diplom.pdf* (2021).