

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

СИСТЕМА АНАЛИЗА ЦЕННЫХ БУМАГ

Студент БХ 09.06.2024 Бахтияров Руслан Рафикович
(подпись, дата) (ФИО полностью)

Руководитель КС 10.06.2024 Карамышева Н.С.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр МБ Бычков
(подпись, дата) (фамилия, инициалы)

Рецензент Доцент Пенз ГТУ 10.06.24 Петерский Р.В.
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 № 11)

Заведующий кафедрой М М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой Отлично (протокол заседания ГЭК от
14.06.2024 № 13)

Секретарь ГЭК Е Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

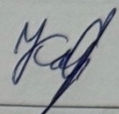
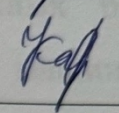
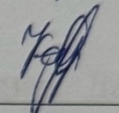
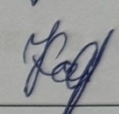
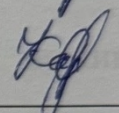
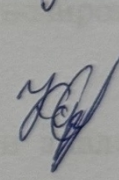
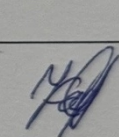
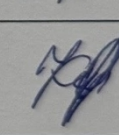
Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 13 » 09 2022г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРА

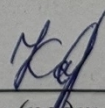
1. Студент Бахтияров Руслан Рафикович гр. 22ВВИм1 факультета вычислительной техники, направления 09.04.01
2. Тема работы «Система анализа ценных бумаг»
Тема утверждена приказом ПГУ №765/о от "13" сентября 2022г.
3. Руководитель магистерской диссертации доцент кафедры ВТ ПГУ Карамышева Н.С.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Назначение разработки: система анализа ценных бумаг.
Разработать автоматизированную систему, помогающую через телеграм-бота пользователям инвестировать в ценные бумаги.
5. Перечень подлежащих разработке вопросов:
 - 1) Анализ системы оценочных критериев, шкалы оценок по каждому критерию, а также сбор данных для обучения.
 - 2) Разработка системы анализа ценных бумаг на основе микросервисной архитектуры.
 - 3) Подготовка датасетов, выбор алгоритма машинного обучения, разработка алгоритмов стратегий инвестирования, обучение модели.
 - 4) Разработка интерфейса телеграм-бота.
 - 5) Тестирование разработанной системы анализа ценных бумаг

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучить теоритические основы	6%	27.09.2022- 15.11.2022	
2.	Анализ требований	6%	16.11.2022- 22.03.2023	
3.	Разработка архитектуры системы	11%	23.03.2023- 14.06.2023	
4.	Разработка серверной части системы	16%	15.06.2023- 10.09.2023	
5.	Разработка клиентской части системы	15%	11.09.2023- 05.01.2024	
6.	Подготовка датасетов, выбор алгоритма машинного обучения, разработка алгоритмов стратегий инвестирования, обучение модели	35%	06.01.2024- 17.04.2024	
7.	Тестирование разработанной системы	4%	18.04.2024- 9.05.2024	
8.	Оформление выпускной работы	7%	10.05.2024- 28.05.2024	

Дата выдачи задания "13" сентября 2022г.

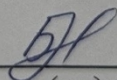
Руководитель магистерской диссертации



Карамышева Н.С

(подпись, дата)

Задание к исполнению принял студент

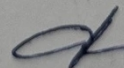


Бахтияров Р.Р.

(подпись, дата)

Работу к защите допустить

Декан факультета



10.06.2024

Фионова Л.Р.

(подпись, дата)

РЕФЕРАТ

Магистерская работа содержит 93 листа, 20 рисунков, 27 источников, 17 приложений, 3 таблицы

СИСТЕМА АНАЛИЗА, ЦЕННЫЕ БУМАГИ, АКЦИИ, МАШИННОЕ ОБУЧЕНИЕ, НЕЙРОСЕТИ

Цель работы - повышение эффективности анализа ценных бумаг аналитиками и инвесторами с помощью применения технологий машинного обучения.

Объектом разработки является система анализа ценных бумаг. Данная система позволяет анализировать текущие котировки фондового рынка на основе применения технологий машинного обучения.

В результате проведенной работы с помощью алгоритма машинного обучения была реализована модель, которая имеет функционал прогнозирования движения акций.

Результаты данной работы могут быть применены в области аналитики ценных бумаг.

<h3 style="margin: 0;">ПГУ1.090401.1.001 ПЗ</h3>											
Изм.	Лист	№ докум.	Подпись	Дата							
Разраб.		Бахтияров Р.Р.		10.06.24	Система анализа ценных бумаг Пояснительная записка						
Провер.		Карамышева Н.С.		10.06.24							
Н. Контр		Бычков А.С.		10.06.24							
Утверд.		Митрохин М.А.		10.06.24							
					<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;">Лит.</td> <td style="width: 10%;">Лист</td> <td style="width: 10%;">Листов</td> </tr> <tr> <td></td> <td style="text-align: center;">2</td> <td style="text-align: center;">93</td> </tr> </table>	Лит.	Лист	Листов		2	93
Лит.	Лист	Листов									
	2	93									
					ФВТ, каф. ВТ гр. 22ВВИм1						

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	6
ВВЕДЕНИЕ	7
1 Анализ заданной проблемы и обоснование ее актуальности	9
1.1 Аналогии	10
1.1.1 Тинькофф инвестиции	10
1.1.2 Приложение Tradingview	11
1.1.3 Торговый робот Divergence-Bot	12
1.2 Постановка задачи	12
2 Архитектура приложения	14
2.1 Микросервисная архитектура	14
2.1.1 Характеристики микросервисов	14
2.1.2 Шаблон Controller-Service-Repository	15
2.1.3 Событийно-ориентированная архитектура	16
2.1.4 Состав компонентов системы	17
2.1.5 Схема архитектуры системы	
Error! Bookmark not defined.	
3 Обучение модели для анализа ценных бумаг	19
3.1 Сбор исторических котировок	19
3.2 Формирование датасетов	20
3.2.1 Формирование датасета на основе индикаторной стратегии «PinBar»	20
3.2.2 Формирование датасета на основе индикаторной стратегии «InsideBar»	22
3.2.3 Формирование датасета на основе индикаторной стратегии «Pivot Point Reversal»	24
3.2.4 Формирование датасета на основе индикаторной стратегии «Fakey»	26
3.2.5 Объединение датасетов	28

3.3 Выбор алгоритма машинного обучения	29
3.3.1 Обучение модели Random Forest	29
3.3.2 Обучение модели LogisticRegression	31
3.3.3 Обучение модели SVC	33
3.3.4 Обучение модели Gradient Boosting	34
3.3.5 Выбор итогового метода обучения	36
4 Реализация	38
4.1 Разработка микросервиса «StockAnalyzer»	38
4.2 Разработка микросервиса «PortfolioManager»	40
4.3 Разработка микросервиса «AnalyticManager»	43
5 Запуск и тестирование приложения	46
5.1 Сборка проекта	46
5.2 Запуск и тестирование	48
ЗАКЛЮЧЕНИЕ	56
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	58
Приложение А Листинг	61
Приложение А.1 Файл «Insidebarstrategy.java»	62
Приложение А.2 Файл «Pinbarstrategy.Java»	63
Приложение А.3 Файл «PPRStrategy.java»	65
Приложение А.4 Файл «FakeyStrategy.java»	67
Приложение А.5 Файл «Candle.java»	69
Приложение А.6 Файл «UserController.java»	70
Приложение А.7 Файл «UserService.java»	71
Приложение А.8 Файл «UserRepository.java»	72
Приложение А.9 Файл «User.java»	73
Приложение А.10 Файл «RandomForestClassifier.py»	75
Приложение А.11 Файл «GradientBoostingClassifier.py»	76
Приложение А.12 Файл «PortfolioManagerService.java»	77
Приложение А.13 Файл «MessageService.java»	78

Приложение А.14 Файл «TelegramBot.java»	83
Приложение А.15 Файл «BotComandsEnum.java»	88
Приложение А.16 Файл «MessageEnum.java	89
Приложение Б Презентация	90

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей магистерской диссертации применяются следующие сокращения и обозначения

API (Application Programming Interface) - интерфейс прикладного программирования, который представляет собой набор определений и протоколов для создания и интеграции прикладного программного обеспечения

Java – язык программирования

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста

Python – язык программирования

REST - архитектурный стиль взаимодействия компонентов распределённого приложения в сети

UML (Unified Modeling Language) – унифицированный язык моделирования

PostgreSQL - свободная реляционная система управления базами данных

ВВЕДЕНИЕ

В современном мире инвестирование играет важную роль для людей, предоставляя им шанс построить свое собственное успешное финансовое будущее. Инвестирование – это покупка акций и активов с целью выйти в прибыль в долгосрочной перспективе.

Навык инвестирования имеет большое количесвто достоинств, который позволит накапливать бюджет многие годы, не затрачивая при этом много времени.

У инвестирования есть свои преимущества и недостатки. Для начало стоит выделить основные плюсы:

- нивилирование инфляции. Инвестиции помогает сохранить бюджет от инфляции и не сниижать вашу покапательную способность,
- инвестиции приносят более высокую норму прибыли по сравнению со сберегательными счетами и вкладами,
- небольшие временные затраты. Инвестор просто держит позиции в долгосрок, чтобы достичь средней прибыли,
- большой выбор приложений и инструментов, которые помогают гибко управлять своим портфелем.

Теперь выделим основные недостатки:

- возможные потери. Несмотря на то, что многие основные индексы растут в течение длительных периодов, никто не может дать гарантий, что конкретная акция вырастет в течение определенного периода,
- нестабильный рынок. Многие позиции очень волатильны и предсказать их движение бывает очень сложно,
- внешние факторы. На движение инвестиций на рынке может влиять экономическая ситуация в стране, новости, глобальные мировые проблемы, политические события.

Самый популярный метод инвестирование на сегодняшний день – это покупка акций. Чтобы выбрать акцию для покупки, инвесторы анализируют

фондовый рынок, котировки, новости и зачастую могут не уследить за какими-то событиями, не увидеть причинно-следственные связи, последовать неправильной стратегии. В решении многих перечисленных проблем могут помочь нейронные сети, они могут обработать огромные объемы данных, автоматизировав рутинные задачи, такие как анализ текстов, извлечение информации и прогнозирование.

1 Анализ заданной проблемы и обоснование ее актуальности

Инвестирование в акции или другие активы способно закрыть многие финансовые потребности человека. Чтобы проделать точную аналитику рынка инвесторы пользуются огромным множеством приложений, которые доступны на данный момент. Данные приложения предлагают различные функционал по анализу рынка, такие как:

- выбор различных типов графиков (линейных, свечных и т.д.),
- масштабирование графиков, выбор разных тайм-фреймов,
- наложение различных слоев данных,
- возможность рисовать линии, фигуры и уровни Фибоначи на графике, считать отношение прибыли и риска,
- использование алгоритмов и ботов.

Помимо multifunctional приложений, инвесторы делают свои прогнозы и покупки, основываясь на стратегиях инвестирования. Стратегия инвестирования – это план, по которому инвестор продает или покупает акции. В мире инвестиций огромное множество видов стратегий, но одними из популярных считаются индикаторные. Индикаторные стратегии основаны на использовании индикаторов – методы анализа, ориентированные на числовые показатели.

Основная сложность инвестирования по индикаторным стратегиям заключается в том, что они всегда подстраиваются под исторические данные. Следовательно, инвестору практически каждый день необходимо анализировать огромное количество исторических котировок, смотреть графики, высчитывать отношения риска и прибыли.

Для обработки большого количества данных сегодня нам активно может помочь искусственный интеллект и связанные с ним модели машинного обучения. Данные модели обучаются на наборе данных предоставляя ей алгоритм, который она может использовать для анализа и обучения на основе этих данных. После завершения обучения модели, можно будет ее применять

для прогнозирования и принятия решений. искусственный интеллект активно интегрируется в различные области и сферы деятельности повседневной жизни человека.

Актуальность данной работы обусловлена тем, что в настоящее время технологии и методы искусственного интеллекта не так активно применяются в анализе фондового рынка на основе индикаторных стратегий. И многие приложения имеют либо не внедряют искусственный интеллект, либо достаточно ограниченно его используют.

1.1 Аналоги

Из всего многообразия приложений можно отметить 3 наиболее популярных. Это Тинькофф Инвестиции, приложение Tradingview и Divergence-Bot. Рассмотрим их наиболее подробно.

1.1.1 Тинькофф инвестиции

Тинькофф Инвестиции — это онлайн-брокер и инвестиционная платформа, предоставляемая Тинькофф Банком. Они позволяют инвесторам покупать и продавать акции, облигации и предоставляют широкий выбор инструментов для анализа фондового рынка.

Рассмотрим плюсы данной платформы:

- удобство использования. Платформа Тинькофф Инвестиции имеет интуитивно понятный интерфейс и простой процесс открытия счета. Они предлагают мобильное приложение и веб-версию, что обеспечивает доступность и удобство использования,
- удобный интерфейс,
- обучение и аналитика. Платформа предлагает обучающие материалы, аналитические отчеты, новости и другую информацию, которая может помочь инвесторам принимать решения и анализировать рынок,
- широкий выбор инструментов.

Однако Тинькофф Инвестиции не решает поставленную задачу из-за некоторых минусов:

- нет анализа тех акций, которых нет на московской и СПб биржах,
- нет инструмента, который бы показывал, когда покупать или продавать акции,
- на графиках нельзя выбрать различные индикаторы,
- не предусмотрены алгоритмы машинного обучения.

1.1.2 Приложение Tradingview

Tradingview – это социальная сеть, аналитическая платформа и мобильное приложение для трейдеров и инвесторов. TradingView предоставляет широкий набор инструментов, как фундаментальных, так и технических, для анализа рынков.

Достоинства данной платформы:

- tradingView предоставляет мощные инструменты для технического анализа акций. Графики нескольких типов, включая свечи, Heiken Ashi, Renko, point & Figure и другие.
- tradingView имеет мобильное приложение и веб-сайт, что обеспечивает доступность использования в любое время и в любом месте,
- tradingView работает с различными брокерами, поэтому есть возможность торговать прямо с платформы,
- tradingView имеет огромное количество готовых и созданных сообществом индикаторов, что позволяет торговать с помощью индикаторных стратегий [2].

Однако TradingView функционал не решает поставленную задачу из-за некоторых недостатков:

- индикаторные стратегии, предоставляющие данной платформой, не имеют алгоритмов машинного обучения, которые позволяли бы на основе исторического набора данных делать прогнозы.

1.1.3 Торговый робот Divergence-Bot

Divergence-Bot - это торговый робот, разработанный для автоматической торговли на основе стратегии дивергенции. Дивергенция – это расхождение направленности индикатора и графика цены [3].

Достоинства данного решения:

- торговля на основе стратегии дивергенции. Это позволяет трейдерам автоматизировать процесс принятия решений и исполнения сделок,
- бот может мгновенно обрабатывать большие объемы данных и принимать решения на основе заранее заданных параметров. Это увеличивает скорость реакции на изменения рынка и улучшает точность торговых операций.

По итогу можно сказать, что Divergence-Bot - хорошее решение для выполнения нашей задачи, но у него есть ряд недостатков:

- divergence-Bot полагается на стратегию дивергенции, и его производительность будет зависеть от эффективности этой стратегии на конкретных рынках и условиях,
- нет мобильного приложения,
- нельзя выбирать сектора инвестирования,
- нет прогнозов, основанных на алгоритмах машинного обучения.

1.2 Постановка задачи

Необходимо разработать программное решение, которое позволит инвесторам принимать решения при инвестировании в ценные бумаги. Программа будет основываться на нескольких стратегиях инвестирования и будет составлять прогнозы о покупке или продаже ценных бумаг с помощью модели машинного обучения.

Программное решение должно удовлетворять следующим требованиям:

- необходимо обеспечить сбор исторических котировок по 300 акциям за последний год;
- возможность выбора желаемой области инвестирования, которые будут предложены чат-ботом. Сектора инвестирования:

- 1) финанансовый сектор;
- 2) здравоохранение;
- 3) энергетика;
- 4) потребительский сектор;
- 5) недвижимость;
- 6) IT и инновационная деятельность;
- 7) автомобильный сектор;

– сформировать датасеты на основе исторических котировок для обучения модели. Датасеты должны быть сформированы на основе инвестиционных стратегий, таких как:

- 1) стратегия PPR;
- 2) стратегия InsideBar;
- 3) стратегия PinBar;
- 4) стратегия Fakey;

– возможность согласиться или отказаться от покупки предложенной акции;

– интерфейс для взаимодействия с данной системой нужно реализовать с помощью чат-бота в Telegram.

2 Архитектура приложения

В данном проекте планируется использование микросервисной архитектуры, которая в настоящее время считается наиболее оптимальным и актуальным подходом.

В данном проекте планируется система, основанная на микросервисной архитектуре, базах данных и событийно-ориентированной архитектуре, которая разработана с целью помочь инвесторам принимать обоснованные инвестиционные решения. Система объединяет в себе передовые алгоритмы анализа рынка и различные стратегии инвестирования, чтобы предоставить инвесторам ценную информацию и рекомендации в реальном времени.

Микросервисная архитектура обеспечивает гибкость и масштабируемость нашей системы, позволяя легко добавлять новые компоненты и функциональность. Базы данных играют ключевую роль в хранении и обработке больших объемов финансовых данных, обеспечивая надежность и быстродействие. Событийно-ориентированная архитектура позволяет системе реагировать на изменения на рынке мгновенно и предоставлять актуальные рекомендации инвесторам.

2.1 Микросервисная архитектура

Микросервисная архитектура представляет собой методологию разработки программного обеспечения, при которой приложение разбивается на небольшие независимые сервисы, взаимодействующие через четко определенные интерфейсы (API). Каждый из этих сервисов находится под ответственностью автономной команды разработчиков.

Архитектура микросервисов обеспечивает легкость масштабирования и ускорение процесса разработки, что позволяет внедрять новые идеи и быстрее выпускать новые функциональные возможности на рынок.

2.1.1 Характеристики микросервисов

Основные характеристики микросервисов:

- автономность. Каждый сервис в микросервисной архитектуре может быть разработан, развернут, управляться и масштабироваться независимо от других сервисов. Сервисы не делятся своим кодом или реализацией с другими службами, их взаимодействие происходит только через четко определенные интерфейсы,

- специализация. Каждый сервис ориентирован на выполнение конкретной задачи и предоставляет определенный набор функциональных возможностей. Если служба со временем становится слишком сложной, разработчики могут разделить её на более мелкие сервисы,

- легкость развертывания. Микросервисы могут быть легко развернуты с помощью различных оркестраторов или серверов,

- обработка ошибок. Каждый сервис самостоятельно справляется с ошибками и восстановлением после сбоев. Если один сервис перестает работать, это не останавливает работу остальных сервисов,

- простая замена. Если нужно заменить один сервис, это можно сделать без воздействия на остальные. Это облегчает обновление системы и внедрение новых функций,

- возможность выбора технологий. Каждый микросервис может использовать разные технологии, подходящие для выполнения своих специфических задач.

2.1.2 Шаблон Controller-Service-Repository

Шаблон Controller-Service-Repository распространен во многих приложениях Spring Boot. Этот паттерн используют, потому что он отлично справляется с разделением проблем. Уровень контроллера в верхней части этой картины несет полную ответственность за раскрытие функциональности, чтобы она могла потребляться внешними объектами (включая, возможно, компонент пользовательского интерфейса). Слой работы с хранимыми данными (Repository) в нижней части этой картины отвечает за хранение и извлечение некоторого набора данных. Уровень сервиса — это то, куда должна идти вся бизнес-

логика. Если бизнес-логика требует извлечения или сохранения данных, она подключается к слою данных. Если кто-то хочет получить доступ к этой бизнес-логике, они проходят через контроллер, чтобы добраться туда [1].

Это довольно простое разделение проблем. Если код связан с хранением или поиском, он должен идти в слой данных. Если речь идет о раскрытии функциональности, она входит в контроллер. Все уникальное в бизнес-логике будет идти на уровне сервиса. Уровень сервиса не заботится о том, как к нему обращаются, он просто выполняет свою работу, используя слой данных там, где это важно. И контроллер просто передает работу на уровень сервиса, поэтому он может оставаться «худым».

2.1.3 Событийно-ориентированная архитектура

Некоторые микросервисы в данном проекте будут общаться через брокера сообщений Kafka, которая играет центральную роль в событийно-ориентированной архитектуре.

Событийно-ориентированная архитектура - это шаблон проектирования с применением событий для запуска и передачи изменений между компонентами системы. Службы здесь взаимодействуют, обмениваясь событиями, то есть сообщениями о наступлении события или изменении состояния [3].

Преимущества событийно-ориентированной архитектуры:

- слабая связность между компонентами благодаря асинхронным сообщениям о событиях,
- надежность и отказоустойчивость с помощью событийно-ориентированным взаимодействием между компонентами системы. Сообщения могут храниться в хранилище, что позволяет восстанавливать события при восстановлении компонентов после аварийного отключения,
- интеграция между различными системами, которые могут иметь различные технологии и языки программирования.

Недостатки событийно-ориентированной архитектуры:

- сложность разработки и отладки,

- сложность управления состоянием,
- сложность отладки. Отследить ошибки в распределенной системе событий намного сложнее, чем в синхронных,
- потеря событий при сбоях системы. Приходится предусматривать самые неочевидные поведения, чтобы избежать такой проблемы.

2.1.4 Схема компонентов системы

Система анализа ценных бумаг состоит из следующих компонентов:

- telegram-бот. Бот регистрируется через специально для этого предназначенный интерфейс мессенджера Telegram. Регистрации включает в себя использование BotFather, через которого можно настраивать своих ботов и получать все необходимые токены для взаимодействия с API на стороне сервера,
- микросервис управления Telegram-ботом. Данный микросервис отвечает за взаимодействие с чат-ботом, созданием в нем необходимого интерфейса, меню, уведомлений в зависимости от контекста диалога, получения событий от клиента, таких как сообщения, нажатия кнопок и т.д. и передач данных в другие микросервисы,
- сервис управления портфелем. Этот сервис предоставляет функциональность для создания, редактирования и отслеживания портфеля клиента. К данному микросервису по REST API обращается микросервис управления Telegram-ботом для получения информации о конкретном пользователе, а также для передачи данных клиента для регистрации новых пользователей или передачи информации о уже существующем пользователях,
- сервис аналитики. Сервис, который на основе полученных данных от сервиса мониторинга рынка, будет анализировать котировки, применять алгоритмы индикаторных стратегий и создавать те или иные рекомендации для клиента. Все рекомендации будут отправляться в топики Kafka.

2.1.5 Схема архитектуры системы

Схема архитектуры системы показана на рисунке 1. Данная схема показывает, как связаны компоненты системы между собой.

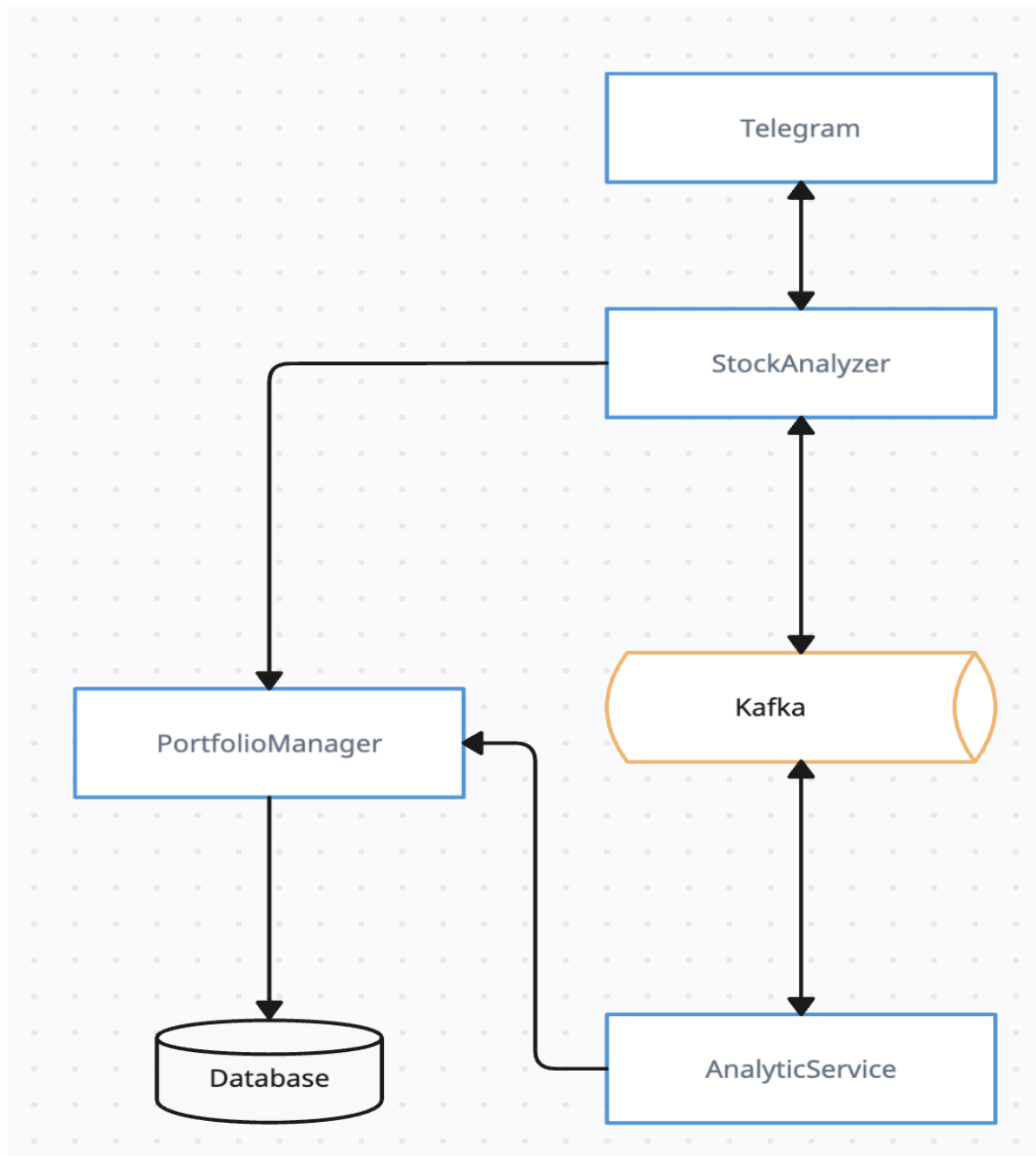


Рисунок 1 – Схема архитектуры проекта

3 Обучение модели для анализа ценных бумаг

В рамках данного исследования была разработана модель для принятия решения о покупке акции на основе инвестиционных стратегий. Процесс обучения модели включал в себя несколько этапов.

3.1 Сбор исторических котировок

Для начала нам нужно собрать исторические котировки. Эти данные включают цены открытия, максимума, минимума и закрытия для каждого дня:

- цена открытия (open),
- максимальная цена (high),
- минимальная цена (low),
- цена закрытия (close).

Исторические котировки были получены через API Tinkoff. API предоставляет различные таймфреймы от одной минуты до одного месяца, а также историю свечек за последний год. Такой широкий диапазон позволяет гибко анализировать данные и адаптировать стратегии к различным таймфреймам.

Были созданы набор данных из около 300 акций из различных секторов инвестирования. Что позволило создать более репрезентативную выборку и учесть влияние различных секторов на общий рынок.

Каждый из этих секторов может обладать уникальными характеристиками и тенденциями, что делает анализ более полным и глубоким. Например, акции технологических компаний могут демонстрировать большую волатильность, а финансовый сектор может быть более чувствителен к макроэкономическим изменениям и решениям ЦБ.

Процесс сбора данных состоял из нескольких этапов. Сначала были созданы API-запросы для получения данных по каждому показателю за нужные периоды времени. Затем данные были обработаны и сохранены в подходящей структуре для последующего анализа. Этот этап включал в себя очистку данных,

проверку их полноты и достоверности, а также преобразование в нужный формат.

3.2 Формирование датасетов

После получение исторических котировок необходимо создать датасеты, на которых будет обучаться модель. Датасеты будут формироваться на основе разных стратегий, а после объединятся в один общий датасет. Этот процесс является критически важным для последующего обучения модели, так как качество и разнообразие данных напрямую влияют на точность прогнозов и способность модели к обобщению.

3.2.1 Формирование датасета на основе индикаторной стратегии «PinBar»

Пин-Бар (Pin Bar) — это индикаторный паттерн технического анализа, сигнализирующий о возможном развороте цены. Свеча характеризуется малым телом и длинными тенями, где одна из теней значительно длиннее тела свечи. Пин-бар может быть бычьим (с длинной верхней тенью) или медвежьим (с длинной нижней тенью). Этот паттерн используется трейдерами для принятия решений о входе или выходе из рынка, так как он часто указывает на потенциальные точки разворота тренда.

Признаки (features) - это параметры, которые используются для обучения модели машинного обучения. В данной задаче выделены следующие признаки:

- тело свечи (Body). Разность между ценой открытия и закрытия ($|close - open|$),
- верхняя тень (UpperShadow). Разность между максимальной ценой и максимумом из цены открытия и закрытия,
- нижняя тень (LowerShadow). Разность между минимумом из цены открытия и закрытия и минимальной ценой,

Метки (labels) - это целевые переменные, которые модель будет предсказывать. В данной задаче метка указывает на успешность паттерна Пин-Бар, что определяется по следующим критериям:

– успешный «Pin Bar». Если после появления «Pin Bar» цена достигает уровня 61.8% коррекции по Фибоначчи в течение 10 дней, паттерн считается успешным (метка = 1),

– неуспешный «Pin Bar». Если цена не достигает уровня 61.8% коррекции по Фибоначчи в течение 10 дней, паттерн считается неуспешным (метка = 0).

В результате создается набор данных, содержащий как признаки, так и метки, которые используются для обучения модели машинного обучения. Такой подход обеспечивает модель необходимыми данными для эффективного выявления и прогнозирования разворотных моделей на финансовых рынках.

Код, который реализует формирование датасета на основе стратегии «Pin Bar» представлен ниже:

```
private static List<String[]> generateDataset(List<Candle>
candles) {
    List<String[]> dataset = new ArrayList<>();
    dataset.add(new String[]{"Body", "UpperShadow", "LowerShadow",
"Success"});

    for (int i = 1; i < candles.size() - 1; i++) {
        Candle curr = candles.get(i);

        double body = Math.abs(curr.getClose() - curr.getOpen());
        double upperShadow = curr.getHigh() -
Math.max(curr.getClose(), curr.getOpen());
        double lowerShadow = Math.min(curr.getClose(),
curr.getOpen()) - curr.getLow();

        boolean isPinBar = (body < upperShadow && upperShadow >
lowerShadow * 2) ||
            (body < lowerShadow && lowerShadow > upperShadow *
2);

        if (isPinBar) {
            boolean success = isSuccess(candles, i);
            dataset.add(new String[]{String.valueOf(body),
String.valueOf(upperShadow), String.valueOf(lowerShadow), success ?
"1" : "0"});
        }
    }
    return dataset;
}

private static boolean isSuccess(List<Candle> candles, int index) {
    Candle pinBar = candles.get(index);
```

```

double fibHigh = pinBar.getHigh();
double fibLow = pinBar.getLow();
double fibLevel61_8 = fibLow + (fibHigh - fibLow) * 0.618;

for (int i = index + 1; i < Math.min(index + 10,
candles.size()); i++) {
    if (candles.get(i).getClose() > fibLevel61_8) {
        return true;
    }
}
return false;
}

```

3.2.2 Формирование датасета на основе индикаторной стратегии

«InsideBar»

Inside Bar (Внутренний Бар) — это свечной паттерн технического анализа, который указывает на период консолидации или нерешительности на рынке. Свеча «Inside Bar» полностью находится в диапазоне предыдущей свечи, то есть высшая точка находится ниже высшей точки предыдущей свечи, а низшая точка - выше низшей точки предыдущей свечи. Этот паттерн может предшествовать значительному движению цены в любом направлении, что делает его важным инструментом для трейдеров, ищущих сигналы для входа или выхода из позиций.

В данной задаче выделены следующие признаки:

- тело свечи (Body). Разница между ценой открытия и закрытия,
- диапазон (Range). Разница между максимальной и минимальной ценой текущей свечи,
- отношение диапазонов (InsideRange). Отношение диапазона текущей свечи к диапазону предыдущей свечи.

В данной задаче метка указывает на успешность паттерна Inside Bar, что определяется по следующим критериям:

- успешный «Inside Bar». Если после появления Inside Bar цена пробивает максимум или минимум текущей свечи в течение 10 дней, паттерн считается успешным (метка = 1),

– неуспешный «Inside Bar». Если цена не пробивает максимум или минимум текущей свечи в течение 10 дней, паттерн считается неуспешным (метка = 0).

Для определения внутренних баров используются критерии, оценивающие положение текущей свечи относительно предыдущей. На основе этих критериев алгоритм анализирует исторические данные для выявления внутренних баров и проверки их достоверности. Код, который реализует формирование датасета на основе стратегии «Inside Bar» представлен ниже:

```
public static List<String[]>
generateInsideBarDataset(List<Candle> candles) {
    List<String[]> dataset = new ArrayList<>();
    dataset.add(new String[]{"Body", "Range", "InsideRange",
"Success"});

    for (int i = 1; i < candles.size() - 1; i++) {
        Candle prev = candles.get(i - 1);
        Candle curr = candles.get(i);

        if (curr.getHigh() < prev.getHigh() && curr.getLow() >
prev.getLow()) {
            double body = Math.abs(curr.getClose() -
curr.getOpen());
            double range = curr.getHigh() - curr.getLow();
            double insideRange = range / (prev.getHigh() -
prev.getLow());

            boolean success = isSuccess(candles, i);
            dataset.add(new String[]{String.valueOf(body),
String.valueOf(range), String.valueOf(insideRange), success ? "1" :
"0"});
        }
    }
    return dataset;
}

private static boolean isSuccess(List<Candle> candles, int index) {
    Candle insideBar = candles.get(index);
    double high = insideBar.getHigh();
    double low = insideBar.getLow();

    for (int i = index + 1; i < Math.min(index + 10,
candles.size()); i++) {
        Candle curr = candles.get(i);
        if (curr.getHigh() > high || curr.getLow() < low) {
            return true;
        }
    }
}
```

```
}  
return false;  
}
```

3.2.3 Формирование датасета на основе индикаторной стратегии «Pivot Point Reversal»

Pivot Point Reversal (PPR) — это свечной паттерн технического анализа, который сигнализирует о возможном развороте цены. Паттерн «PPR» состоит из трех свечей: первая свеча имеет минимум, вторая свеча устанавливает более низкий минимум, а третья свеча закрывается выше, чем вторая свеча. Этот паттерн может указывать на разворот вниз (медвежий разворот) или вверх (бычий разворот).

Признаки данной стратегии:

- тело первой свечи (Body1). Разница между ценой открытия и закрытия первой свечи,
- тело второй свечи (Body2). Разница между ценой открытия и закрытия второй свечи,
- тело третьей свечи (Body3). Разница между ценой открытия и закрытия третьей свечи,
- разница минимумов (LowDiff). Разница между минимумом первой и второй свечей,
- разница закрытий (CloseDiff). Разница между закрытием второй и третьей свечей.

Алгоритм успешности данного паттерна:

- успешный «PPR». Если после появления PPR цена пробивает максимум или минимум третьей свечи в течение 10 дней, паттерн считается успешным (метка = 1),
- неуспешный «PPR». Если цена не пробивает максимум или минимум третьей свечи в течение 10 дней, паттерн считается неуспешным (метка = 0).

Для идентификации «PPR» используются критерии, оценивающие соотношения трех свеч. На основе этих критериев алгоритм анализирует исторические данные, идентифицирует «PPR» и проверяет его успешность.

Код, который реализует формирование датасета на основе стратегии «PPR» представлен ниже:

```
public static List<String[]> generatePPRDataset(List<Candle>
candles) {
    List<String[]> dataset = new ArrayList<>();
    dataset.add(new String[]{"Body1", "Body2", "Body3",
"LowDiff", "CloseDiff", "Success"});

    for (int i = 2; i < candles.size() - 1; i++) {
        Candle first = candles.get(i - 2);
        Candle second = candles.get(i - 1);
        Candle third = candles.get(i);

        if (second.getLow() < first.getLow() &&
third.getClose() > second.getClose()) {
            double body1 = Math.abs(first.getClose() -
first.getOpen());
            double body2 = Math.abs(second.getClose() -
second.getOpen());
            double body3 = Math.abs(third.getClose() -
third.getOpen());
            double lowDiff = first.getLow() -
second.getLow();
            double closeDiff = third.getClose() -
second.getClose();

            boolean success = isSuccess(candles, i);
            dataset.add(new String[]{
                String.valueOf(body1),
                String.valueOf(body2),
                String.valueOf(body3),
                String.valueOf(lowDiff),
                String.valueOf(closeDiff),
                success ? "1" : "0"});
        }
    }
    return dataset;
}

private static boolean isSuccess(List<Candle> candles, int
index) {
    Candle third = candles.get(index);
    double high = third.getHigh();
    double low = third.getLow();
```

```

        for (int i = index + 1; i < Math.min(index + 10,
candles.size()); i++) {
            Candle curr = candles.get(i);
            if (curr.getHigh() > high || curr.getLow() < low) {
                return true;
            }
        }
        return false;
    }
}

```

3.2.4 Формирование датасета на основе стратегии «Fakey»

Fakey (или False Breakout) — это свечной паттерн технического анализа, который указывает на ложное пробитие уровня консолидации или поддержки/сопротивления на рынке. Он состоит из двух частей:

- ложного пробития уровня (фальшивого сигнала),
- возвращения цены в исходный диапазон.

Признаки данной стратегии:

- тело свечи (Body). Разность между ценой открытия и закрытия,
- диапазон (Range). Разница между максимальной и минимальной ценой текущей свечи,
- отношение диапазонов (FakeyRange). Отношение диапазона текущей свечи к диапазону предыдущей свечи.

Алгоритм успешности данного паттерна:

- успешный «Fakey». Если после появления «Fakey» цена движется в направлении, противоположном ложному пробитию, и достигает уровня, равному диапазону предыдущей свечи в течение 10 дней, паттерн считается успешным (метка = 1),
- неуспешный «Fakey». Если цена не достигает указанного уровня в течение 10 дней, паттерн считается неуспешным (метка = 0).

Для распознавания паттерна «Fakey» используются критерии, оценивающие ложное пробитие ценового диапазона. Алгоритм анализирует исторические данные, чтобы выявить паттерны «Fakey» и проверить их достоверность на основе этих критериев.

Код, который реализует формирование датасета на основе стратегии «Fakey» представлен ниже:

```
public static List<String[]> generateFakeyDataset(List<Candle>
candles) {
    List<String[]> dataset = new ArrayList<>();
    dataset.add(new String[]{"Body", "Range", "FakeyRange",
"Success"});

    for (int i = 2; i < candles.size() - 1; i++) {
        Candle prev = candles.get(i - 1);
        Candle curr = candles.get(i);

        // Проверка на ложное пробитие (Fakey)
        if ((curr.getHigh() > prev.getHigh() && curr.getClose()
< prev.getClose()) ||
            (curr.getLow() < prev.getLow() && curr.getClose() >
prev.getClose())) {

            double body = Math.abs(curr.getClose() -
curr.getOpen());
            double range = curr.getHigh() - curr.getLow();
            double fakeyRange = range / (prev.getHigh() -
prev.getLow());

            boolean success = isFakeySuccess(candles, i, prev);
            dataset.add(new String[]{String.valueOf(body),
String.valueOf(range), String.valueOf(fakeyRange), success ? "1" :
"0"});
        }
    }
    return dataset;
}

private static boolean isFakeySuccess(List<Candle> candles, int
index, Candle prevCandle) {
    Candle fakeyCandle = candles.get(index);
    double targetUp = fakeyCandle.getLow() -
(prevCandle.getHigh() - prevCandle.getLow());
    double targetDown = fakeyCandle.getHigh() +
(prevCandle.getHigh() - prevCandle.getLow());

    for (int i = index + 1; i < Math.min(index + 10,
candles.size()); i++) {
        Candle curr = candles.get(i);
        if (curr.getLow() <= targetUp || curr.getHigh() >=
targetDown) {
            return true;
        }
    }
    return false;
}
```

}

3.2.5 Объединение датасетов

Объединение четырех датасетов в один позволяет получить более гибкую модель, учитывающую различные паттерны. Это также позволяет проводить более полный анализ и повышать точность прогнозов.

Итоговый набор данных будет включать атрибуты и метки для четырех стратегий (Pin Bar, Inside Bar и Pivot Point Reversal, Fakey), а также объединенную метку успеха. Итоговая структура датасета будет выглядеть из следующих параметров:

- разность между ценой открытия и закрытия (body),
- разность между максимальной ценой и максимумом из цены открытия и закрытия (upperShadow),
- разница между минимумом из цены открытия и закрытия и минимальной ценой (lowerShadow),
- разница между максимальной и минимальной ценой текущей свечи (range),
- отношение диапазона текущей свечи к диапазону предыдущей свечи (insideRange),
- разница между ценой открытия и закрытия второй свечи для «PPR» (body2),
- разница между ценой открытия и закрытия третьей свечи для «PPR» (body3),
- разница между минимумом первой и второй свечи для «PPR» (lowDiff),
- отношение диапазона текущей свечи к диапазону предыдущей свечи (FakeyRange),
- направление тренда за последний 10 дней (trend10Days),
- итоговая метка успешности (successful).

Итоговая метка успешности, которая принимает значение 1, если хотя бы один из паттернов был успешным в течение 10 дней после его формирования.

Также был добавлен параметр «trend10Days», который будет показывать направление тренда за последний 10 дней. Есть три типа тренда:

- восходящий тренд, значение 1,
- нисходящий тренд, значение -1,
- боковой тренд, значение 0.

3.3 Выбор алгоритма машинного обучения

Для тестового обучения было попробовано несколько методов обучения с учителем, среди которых: Logistic Regression, SVC, Random Forest, и Gradient Boosting.

3.3.1 Обучение модели Random Forest

Для обучения моделей используется алгоритм Random Forest, который является ансамблевым методом и хорошо справляется с задачами классификации и регрессии. Random Forest строит множество деревьев решений и использует голосование для принятия окончательного решения. Random Forest имеет следующие достоинства:

- способность эффективно работать с данными, имеющими множество признаков и классов,
- имеет высокую точность предсказания по сравнению с линейными алгоритмами,
- не чувствителен к выбросам данных
- Обработка данных с пропусками: Модель справляется с пропущенными значениями, заполняя их на основе наиболее вероятного пути в дереве.
- хорошо работает с пропущенными данными,
- эффективно обрабатывает большое количество данных.

Недостатки данного метода:

- большой размер получаемой модели, особенно при больших деревьях,
- достаточно сложно интерпретировать результат при десятке или сотне деревьев.

Для обучения модели Random Forest мы используем библиотеку scikit-learn на языке Python.

Обученная модель была оценена на тестовом наборе данных с использованием метрики Accuracy. Accuracy — это показатель, который описывает общую точность предсказания модели по всем классам.

Для сохранения модели используется библиотека joblib, которая является частью библиотеки sklearn.

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

df = pd.read_csv('dataset.csv')

X = df[['Body', 'UpperShadow', 'LowerShadow', 'Range',
'InsideRange', 'Body2', 'Body3', 'LowDiff', 'FakeyRange',
'trend10Days']]
y = df['successful']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

model = RandomForestClassifier(n_estimators=100,
random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
# Оценка модели
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')

# Сохранение модели
dump(model, 'logistic_regression_model.joblib')
```

В Random Forest были использованы следующие параметры:

- `n_estimators = 100`. Устанавливает количество деревьев в лесу. При увеличении этого параметра повышает точность модели, но также увеличивает

время обучение, поэтому это всегда компромисс. Значение 100 в данной задаче оптимально подходит,

- `random_state = 42`. Этот параметр предуславливает генератор псевдослучайных чисел, который необходим для построения воспроизводимых экспериментов,

- `min_samples_split`. Отвечает за минимальное количество образцов, необходимых для разделения узла. При увеличении параметров модель становится более устойчива к шуму данных. В нашем случае значение стоит по умолчанию,

- `min_samples_leaf`. Минимальное количество образцов в листовом узле. Значение этого параметра также установлено по умолчанию.

После использования подхода для подбора параметров была определена наилучшая модель с точностью: $\text{Accuracy\%} = 75.45$.

3.3.2 Обучение модели LogisticRegression

Алгоритм логистической регрессии является простым и базовым методом для задач классификации. Он использует логистическую функцию для оценки вероятности того, что образец принадлежит к одному из двух классов. Логистическая регрессия обладает следующими достоинствами:

- простота и интерпретируемость модели,
- эффективность при работе с большими наборами данных,
- возможность обновления модели с новыми данными,
- хорошо справляется с задачами, где классы линейно разделимы.

Недостатки данного метода:

- линейность модели, что может быть ограничением при работе с нелинейными данными.

Для обучения модели Logistic Regression мы используем библиотеку `scikit-learn` на языке Python. Обученная модель была оценена на тестовом наборе данных с использованием метрик F1-score и Accuracy. Для сохранения модели используется библиотека `joblib`, которая является частью библиотеки `sklearn`.

```

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from joblib import dump, load

# Загрузка данных
df = pd.read_csv('dataset.csv')

# Определение признаков и целевой переменной
X = df[['Body', 'UpperShadow', 'LowerShadow', 'Range',
'InsideRange', 'Body2', 'Body3', 'LowDiff', 'FakeyRange',
'trend10Days']]
y = df['successful']

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Инициализация и обучение модели Logistic Regression
model = LogisticRegression(random_state=42, max_iter=10000)
model.fit(X_train, y_train)

# Предсказание на тестовой выборке
y_pred = model.predict(X_test)

# Оценка модели
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')

# Сохранение модели
dump(model, 'logistic_regression_model.joblib')

```

В Logistic Regression были использованы следующие параметры:

- `random_state = 42`. Параметр предуславливает генератор псевдослучайных чисел, что необходимо для воспроизводимых экспериментов.
- `max_iter = 10000`. Устанавливает максимальное количество итераций для алгоритма оптимизации. При большом количестве данных может потребоваться увеличить это значение для достижения сходимости.

После использования подхода для подбора параметров была определена наилучшая модель с точностью: `Accuracy% = 71.71247454901`.

3.3.3 Обучение модели SVC

Для обучения моделей используется алгоритм SVC (Support Vector Classifier), который является методом машинного обучения, основанным на теории опорных векторов, и используется для задач классификации.

SVC имеет следующие достоинства:

- эффективно работает в пространствах с высокой размерностью,
- хорошо справляется с задачами, где количество признаков превышает количество образцов,
- эффективен при работе как с линейно, так и с нелинейно разделимыми данными благодаря использованию различных ядерных функций,
- показывает хорошие результаты даже при нелинейной разделимости классов за счет использования ядерных методов.

Недостатки данного метода:

- модель чувствительна к выбору параметров и ядра,
- требует много времени на обучение при больших объемах данных,
- плохо масштабируется с ростом числа образцов.

Обученная модель также была оценена на тестовом наборе данных с использованием метрик F1-score и Accuracy. Код представлен ниже:

```
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from joblib import dump, load

# Загрузка данных
df = pd.read_csv('dataset.csv')

# Определение признаков и целевой переменной
X = df[['Body', 'UpperShadow', 'LowerShadow', 'Range',
'InsideRange', 'Body2', 'Body3', 'LowDiff', 'FakeyRange',
'trend10Days']]
y = df['successful']

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

```
# Инициализация и обучение модели SVC
model = SVC(kernel='rbf', C=1.0, random_state=42)
model.fit(X_train, y_train)

# Предсказание на тестовой выборке
y_pred = model.predict(X_test)

# Оценка модели
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')

# Сохранение модели
dump(model, 'svc_model.joblib')
```

В SVC были использованы следующие параметры:

- kernel='rbf'. Использование радиально-базисной функции (RBF) в качестве ядра для учета нелинейных зависимостей в данных,
- C=1.0. Регуляризационный параметр, контролирующий жесткость разделяющей гиперплоскости. Низкие значения параметра делают модель более жесткой и уменьшают переобучение, высокие значения — менее жесткой,
- random_state=42. Этот параметр предуславливает генератор псевдослучайных чисел для построения воспроизводимых экспериментов.

После использования подхода для подбора параметров была определена наилучшая модель с точностью: Accuracy% = 69.73.

3.3.4 Обучение модели Gradient Boosting

Для обучения моделей используется алгоритм Gradient Boosting, который является ансамблевым методом машинного обучения, объединяющим множество слабых моделей для создания сильной модели. Gradient Boosting строит деревья последовательно, каждое следующее дерево исправляет ошибки предыдущего. Gradient Boosting имеет следующие достоинства:

- высокая точность предсказания по сравнению с линейными моделями,
- эффективность при работе с данными, содержащими много признаков и сложные зависимости,
- способность корректировать ошибки, допущенные предыдущими моделями,

– хорошо работает с различными типами данных и подходит как для задач регрессии, так и для классификации.

Недостатки данного метода:

– высокая вычислительная сложность и длительное время обучения, особенно при большом количестве деревьев,

– модель чувствительна к переобучению, если параметры не настроены должным образом,

– сложность интерпретации результатов, так как итоговая модель представляет собой ансамбль большого количества деревьев.

Код обучения модели представлен ниже:

```
import pandas as pd
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from joblib import dump, load

# Загрузка данных
df = pd.read_csv('dataset.csv')

# Определение признаков и целевой переменной
X = df[['Body', 'UpperShadow', 'LowerShadow', 'Range',
'InsideRange', 'Body2', 'Body3', 'LowDiff', 'FakeyRange',
'trend10Days']]
y = df['successful']

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Инициализация и обучение модели Gradient Boosting
model = GradientBoostingClassifier(n_estimators=100,
learning_rate=0.1, max_depth=3, random_state=42)
model.fit(X_train, y_train)

# Предсказание на тестовой выборке
y_pred = model.predict(X_test)

# Оценка модели
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')

# Сохранение модели
```

```
dump(model, 'gradient_boosting_model.joblib')
```

В Gradient Boosting были использованы следующие параметры:

- `n_estimators=100`. Устанавливает количество деревьев в ансамбле.

Большее количество деревьев может повысить точность, но увеличивает время обучения,

- `learning_rate=0.1`. Определяет, насколько каждая новая модель корректирует ошибки предыдущих. Меньшие значения `learning_rate` делают процесс обучения более устойчивым, но требуют большего количества деревьев,

- `max_depth=3`. Максимальная глубина дерева. Небольшие значения этого параметра помогают избежать переобучения,

- `random_state=42`. Этот параметр предусматливает генератор псевдослучайных чисел для построения воспроизводимых экспериментов.

После использования подхода для подбора параметров была определена модель с точностью: Accuracy% = 73.91.

3.3.5 Выбор итогового метода обучения

Сравнительная характеристика разных методов обучения приведена на рисунке 2.

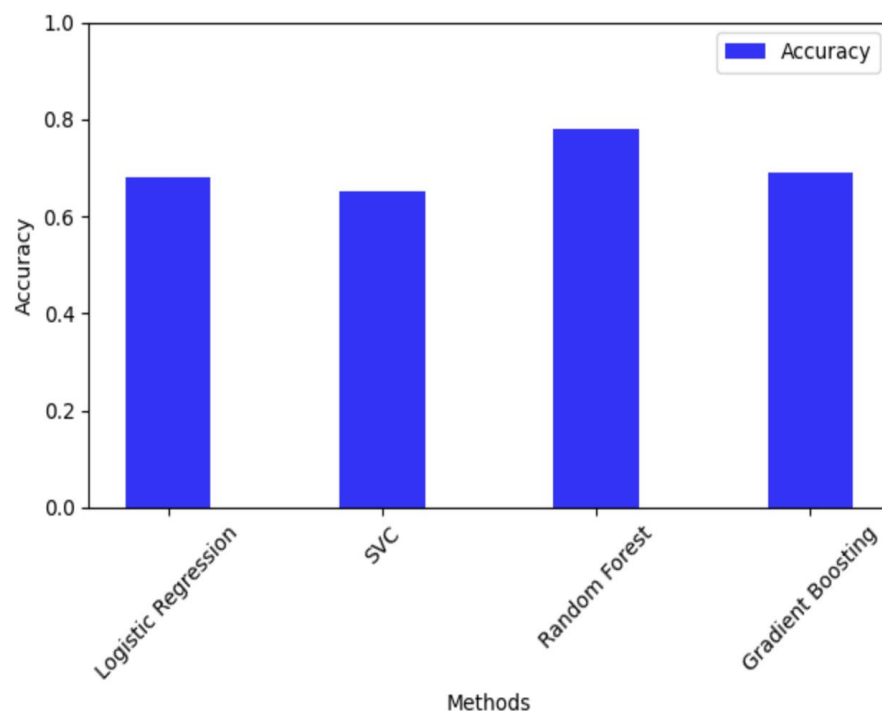


Рисунок 2 – Сравнение эффективности разных методов обучения

По итогу сравнения лучше всех оказался метод Random Forest, так как он показал себя лучше, чем остальные методы. Данный метод основан на использовании ансамбля решающих деревьев.

4 Реализация

В данной работе реализована система, которая состоит из трех микросервисов, кафки и базы данных.

4.1 Разработка микросервиса «StockAnalyzer»

Данный микросервис отвечает за взаимодействие с чат-ботом, созданием в нем необходимого интерфейса, получения событий от клиента и передачей данных в другие системы.

Для взаимодействия с Telegram API используется библиотека «org.telegram.telegrambots». Она облегчает работу с Telegram благодаря огромному количеству интерфейсов.

Для удобного пользования ботом были разработаны специальные inline-кнопки, которые позволяют выполнять различные действия прямо в чат-боте. Они создаются динамически и появляются в зависимости от контекста. Пример создания кнопок «Продать» и «Не продавать» представлен ниже:

```
List<SendMessage> sendMessages = new ArrayList<>();
var message = new SendMessage();
List<InlineKeyboardButton> row1 = new ArrayList<>();
row1.add(new InlineKeyboardButton("Продать", "Продать"));
row1.add(new InlineKeyboardButton("Не продавать", "Не
продавать"));
List<List<InlineKeyboardButton>> rowsInLine = new ArrayList<>();
rowsInLine.add(row1);

stockOffer.getChatIds().forEach(chatId -> {
    message.setChatId(String.valueOf(chatId));
    String messagee;
    if(stockOffer.getIsPositive()){
        messagee = String.format("Привет! По стратегии %s сегодня
отличное время, чтобы выйти в прибыль, продав акцию %s, которая
торгуется по цене %s", stockOffer.getStrategy(),
stockOffer.getStockName(), stockOffer.getPurchasePrice());
    } else {
        messagee = String.format("Привет! К сожалению, стратегия %s
не сработала. Рекомендуется продать акцию %s, так как она опустилась
ниже стоп-цены %s", stockOffer.getStrategy(),
stockOffer.getStockName(), stockOffer.getPurchasePrice());
    }
    message.setText(messagee);
    var markupInLine = new InlineKeyboardMarkup();
    markupInLine.setKeyboard(rowsInLine);
    message.setReplyMarkup(markupInLine);
});
```

```
        sendMessages.add(message);  
    });  
  
    return sendMessages;  
}
```

Данный микросервис также получит события от пользователя через Telegram WebHook. События обрабатываются соответствующими обработчиками, которые далее передают информацию в другие микросервисы через Kafka.

Основную функции данного микросервиса:

- создание интерфейса в телеграм-боте,
- отправка информации о действиях пользователя в другие микросервисы для его регистрации, формирования портфеля и обновления данных,
- получения сообщений из топиков кафки от других микросервисов для отправки предложений о покупке или продажи акций.

Диаграмма классов микросервиса «StockAnalyzer» представлена на рисунке 3.

- создание нового пользователь,
- получение пользователь по индентификатору,
- получение всех пользователей,
- обновление пользователя.

В микросервисе также реализована конвертация DTO (Data Transfer Objects) в Entity. DTO используется для отправки в другие микросервисы, тогда как Entity представляет собой объекты, которые хранятся в базе данных.

При разработке данного сервиса была использована СУБД PostgreSQL. Далее представлено описание структуры, разработанной БД. Таблица 1 содержит описание структуры таблицы «Пользователь (user)».

Таблица 1 – Структура таблицы «Пользователь (user)».

Наименование поля	Описание поля	Тип данных
chat_id	Идентификатор пользователя	VarChar(255)

Каждый пользователь может выбирать интересующую его область инвестирования. Таблица 2 описывает структуры таблицы «Индустрия (industry)».

Таблица 2 – Структура таблицы «Индустрия (industry)».

Наименование поля	Описание поля	Тип данных
id	Идентификатор сектора	VarChar(255)
name	Наименование сектора	VarChar(255)
user_fk_id	Внешний ключ для таблицы «user»	VarChar(255)

Таблица «Stock» была создана для хранения информации об акции, цене ее покупки, а также ценах, когда ее следует продать. Таблица 3 содержит описание структуры таблицы «Акция (Stock)».

Таблица 3 – Структура таблицы «Акция (Stock)».

Наименование поля	Описание поля	Тип данных
id	Идентификатор акции	VarChar(255)
stock_name	Наименование сектора	VarChar(255)
purchase_price	Цена покупки	Decimal

profit_price	Цена продажи в прибыль	Decimal
stop_price	Цена продажи в убыток	Decimal
sector_fk_id	Внешний ключ таблицы «industry»	VarChar(255)

Физическая модель базы данных представлена на рисунке 4.

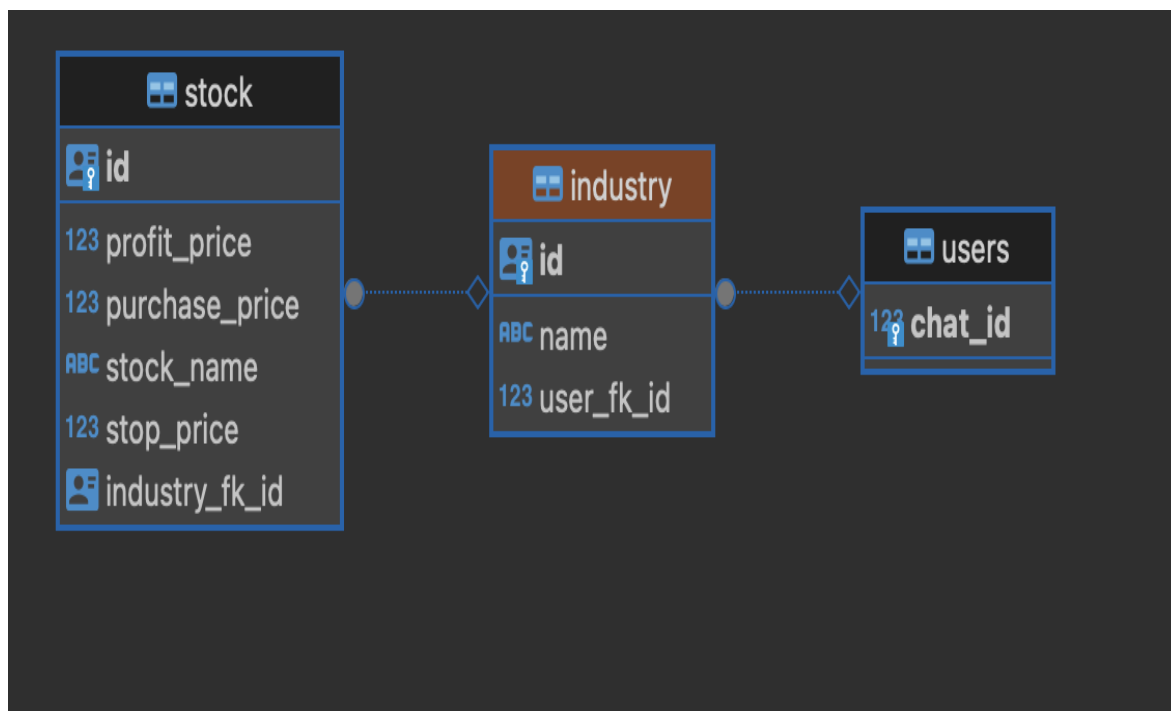


Рисунок 4 – Физическая модель базы данных

На рисунке 5 представлена диаграмма классов микросервиса «PortfolioManager».

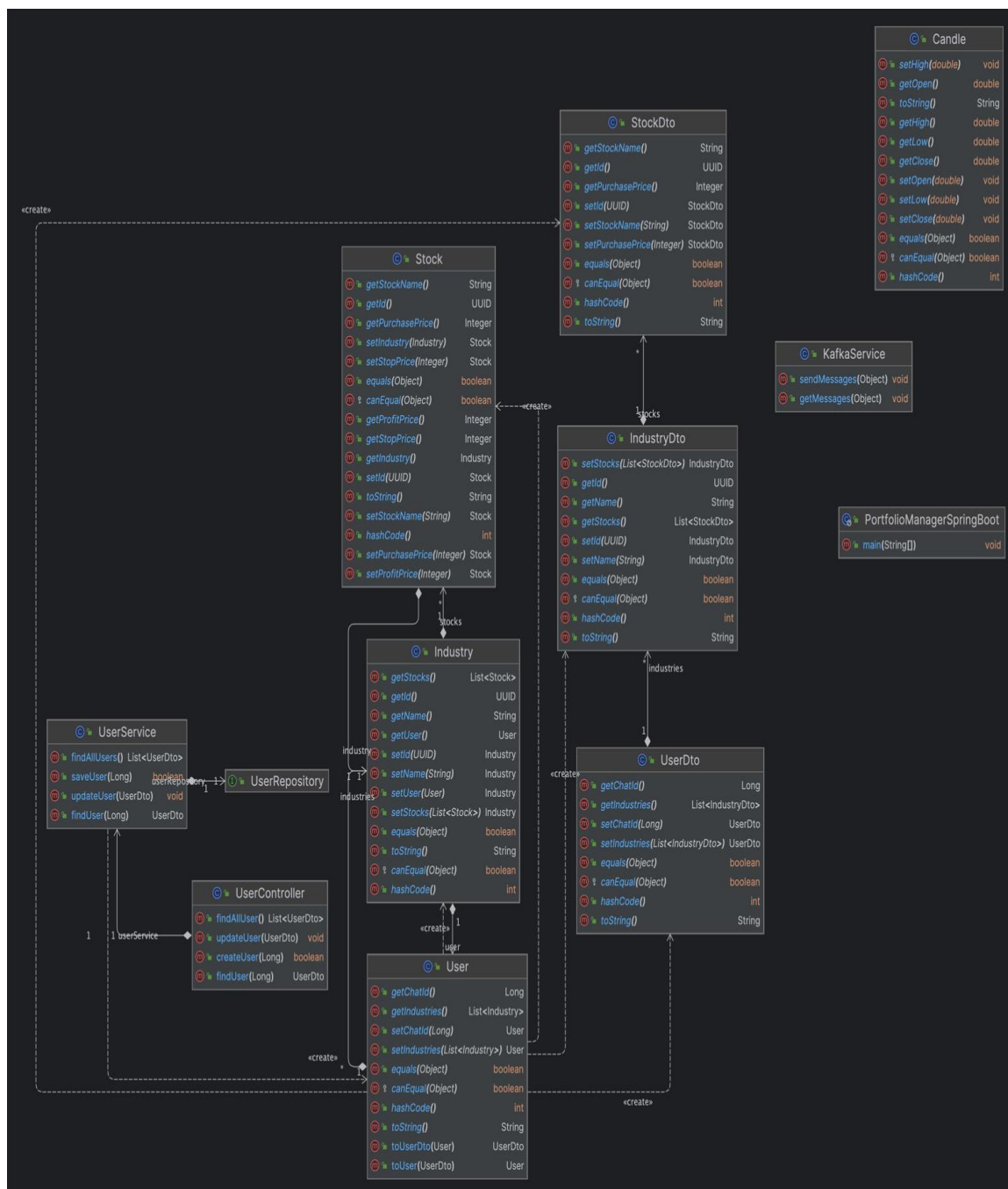


Рисунок 5 – Диаграмма классов микросервиса «PortfolioManager»

4.3 Разработка микросервиса «AnalyticManager»

Микросервис «AnalyticManager» отвечает за анализ рыночных котировок, предоставление рекомендаций по покупке или продаже акций. Главная функция микросервиса заключается в автоматическом анализе котировок и взаимодействии с обученной моделью с помощью алгоритма машинного обучения RandomForese для принятия инвестиционных решений. «StockAnalyzer» имеет несколько планировщиков задач (шедулеров):

– шедулер анализа котировок. Данный шедулер обращается к API Tinkoff для получения актуальных котировок по 500 различным акциям, которые относятся к шести видам секторов инвестирования. Далее он анализирует свечи, которые соответствуют заданным инвестиционным стратегиям, и отправляет данные этих свечей в обученную модель на Python для получения рекомендаций по покупке акций,

– шедулер мониторинга купленных акций пользователей. Этот шедулер также запускается два раза в день для проверки уже купленных акций пользователей. Он оценивает, достигли ли эти акции заданной цены для продажи, будь то прибыльная или убыточная позиция, и передает соответствующие рекомендации.

– На рисунке 6 представлена диаграмма классов микросервиса «AnalyticManager».

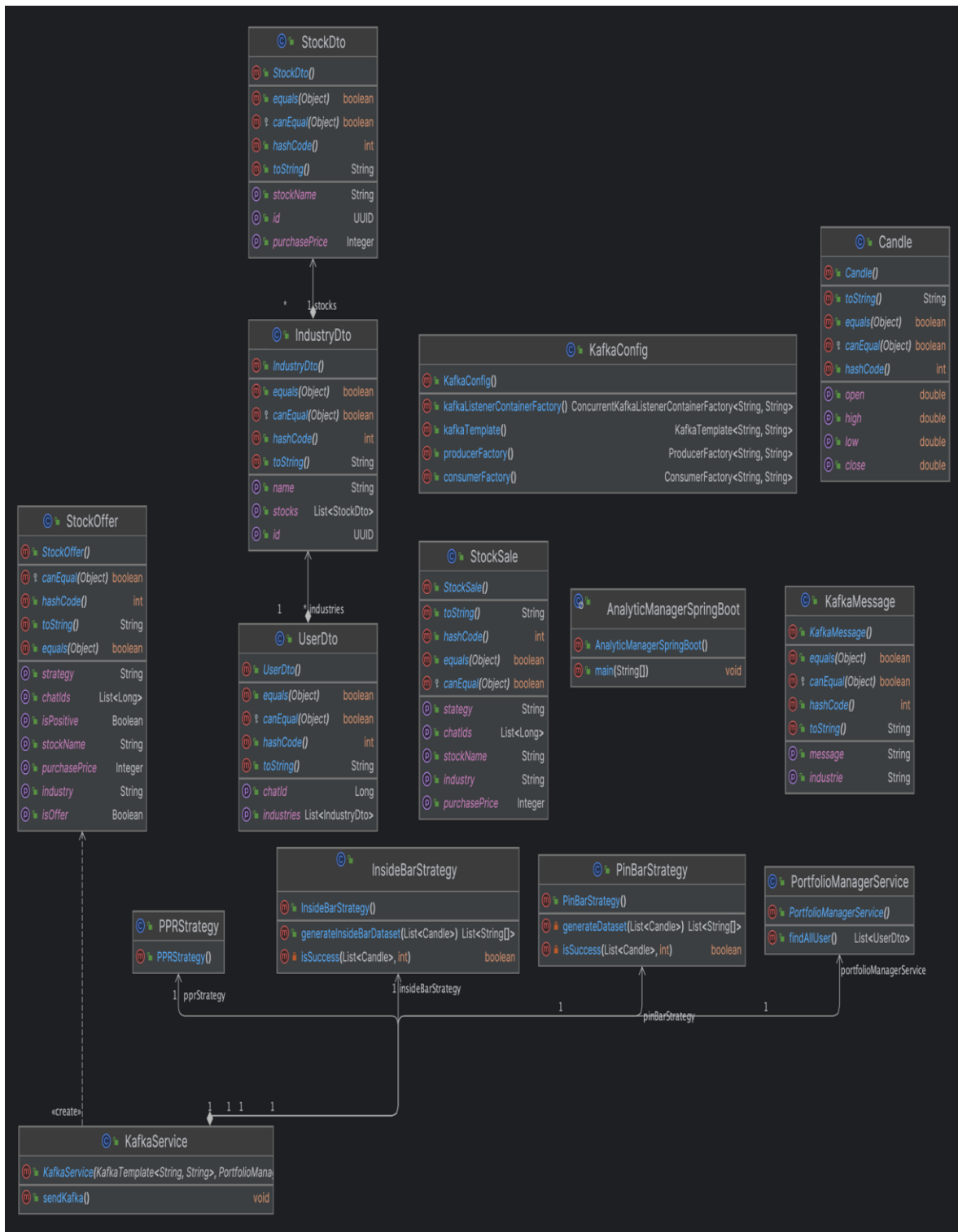


Рисунок 6 – Диаграмма классов микросервиса «AnalyticManager»

5 Запуск и тестирование приложения

5.1 Сборка проекта

Для сборки и развертывания нашего проекта используется Docker и Docker Compose. Эти инструменты облегчают управление зависимостями и обеспечивают изоляцию среды, что упрощает развертывание и масштабирование приложений.

Docker - это платформа, которая позволяет разработчикам упаковывать приложения и их зависимости в контейнеры. Контейнеры гарантируют, что приложение будет работать одинаково, независимо от среды, в которой оно запущено.

Docker Compose - это инструмент для определения и управления многоконтейнерными Docker приложениями. С его помощью можно описать конфигурацию всех сервисов вашего приложения в одном YAML файле и управлять ими как единым целым.

В работе реализован следующий конфигурационный файл:

```
version: "3.9"
services:
  postgres:
    image: postgres:13.3
    environment:
      POSTGRES_DB: "****"
      POSTGRES_USER: "****"
      POSTGRES_PASSWORD: "****"
    ports:
      - "5432:5432"
  zookeeper:
    image: confluentinc/cp-zookeeper:7.0.9
    container_name: zookeeper-otuskafka
    healthcheck:
      test: "[[ $(echo srvr | nc localhost 2181 | grep -oG 'Mode: standalone') = \"Mode: standalone\" ]]"
      interval: 10s
      timeout: 1s
      retries: 30
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - "2181:2181"
```

```

kafka:
  image: confluentinc/cp-kafka:7.0.9
  container_name: kafka-otuskafka
  depends_on:
    zookeeper:
      condition: service_healthy
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
    KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://kafka:9092,PLAINTEXT_HOST://localhost:9091
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
    KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
    KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
    KAFKA_JMX_PORT: 9101
    KAFKA_JMX_HOSTNAME: localhost
  ports:
    - "9092:9092"
    - "9091:9091"
    - "9101:9101"
kafdrop:
  image: obsidiandynamics/kafdrop
  container_name: kafdrop
  restart: "no"
  ports:
    - "9000:9000"
  environment:
    KAFKA_BROKERCONNECT: "kafka:9092"
    JVM_OPTS: "-Xms16M -Xmx48M -Xss500K -XX:-TieredCompilation -
XX:+UseStringDeduplication -noverify"
  depends_on:
    - "kafka"

```

Для запуска всех сервисов используйте следующие команды:

- `docker-compose up -d`. Запускает все контейнеры в фоновом режиме,
- `docker-compose down`. Останавливает и удаляет все контейнеры, сети и тома, созданные командой `up`.

Эти команды помогут вам быстро и эффективно развернуть все необходимые компоненты и начать работу с проектом.

Для запуска приложения необходимо установить Docker. Далее нужно открыть терминал так, чтобы по команде ls или dir отображался файл docker-compose.yml. После нужно выполнить команды docker-compose up, после чего запустятся все необходимые компоненты системы.

5.2 Запуск и тестирование

Далее в приложение “Telegram” в строке поиска вводим название бота «StockMarketAnalyzer». Чтобы включить бота нужно нажать кнопку «Запустить» (рисунок 7), после чего введется автоматически команда «/start» и придет приветственное сообщение, показанное на рисунке 8.

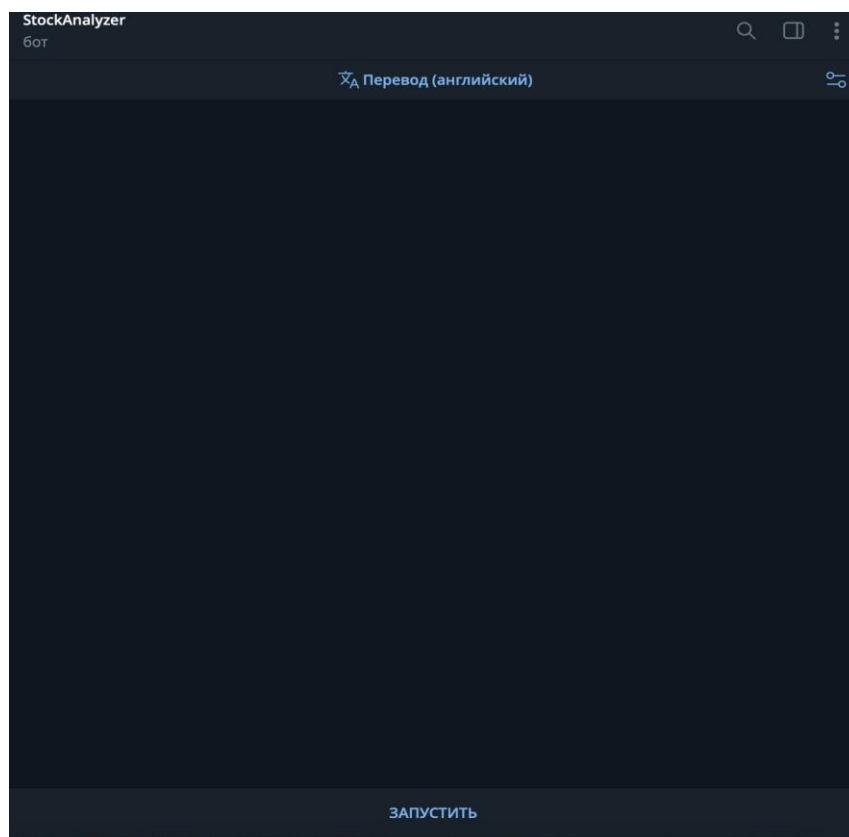


Рисунок 7 – Стартовое окно

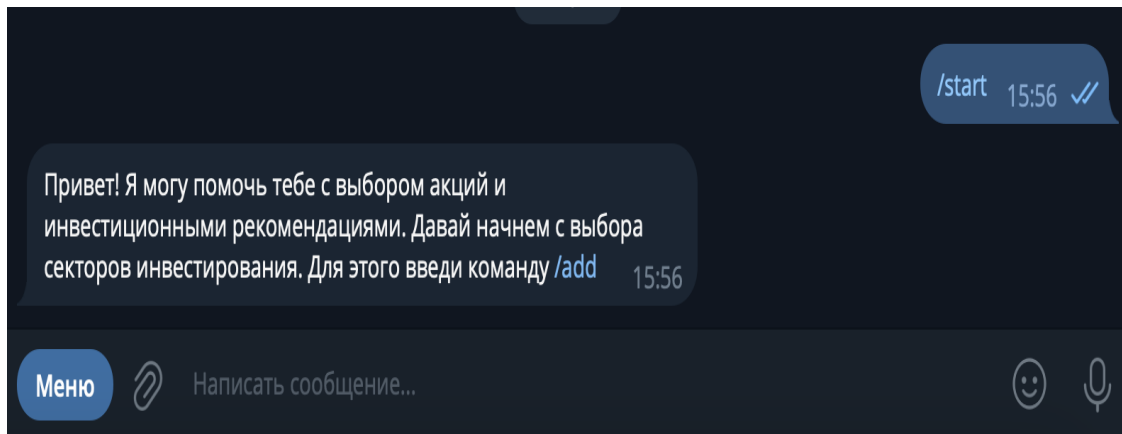


Рисунок 8 – Приветственное сообщение

Приветственное сообщение предлагает нам ввести команду «/add», чтобы выбрать сектора инвестирования. Список всех доступных команд можно получить, нажав на кнопку «Меню» в нижнем левом углу чата, как показано на рисунке 9.

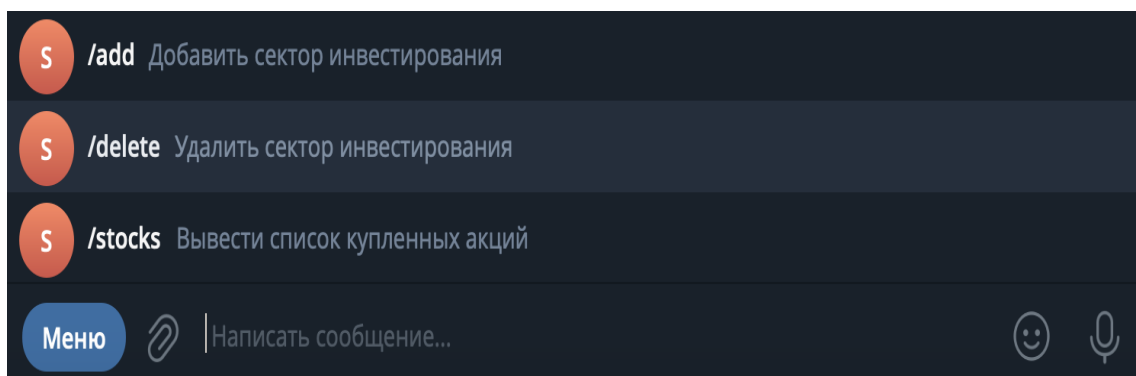


Рисунок 9 – Список команд

Чтобы бот нам предлагал рекомендации по покупке акций, необходимо выбрать сектора инвестирования. После ввода команды «/add» в ответном сообщении появится список доступных на данный момент секторов.

Список секторов включает в себя:

- IT,
- финансы,
- энергетика,
- недвижимость,
- здравоохранение,
- горнопромышленность.

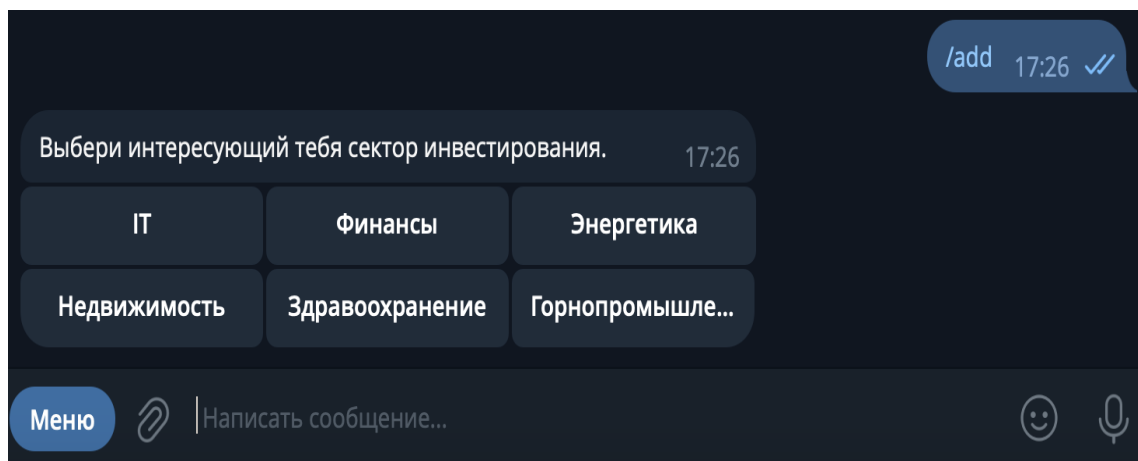


Рисунок 10 – Список секторов инвестирования

После выбора интересующего сектора и при условии отсутствия ошибок, бот выведет сообщение о том, что сектор успешно добавлен. Это показано на рисунке 11.

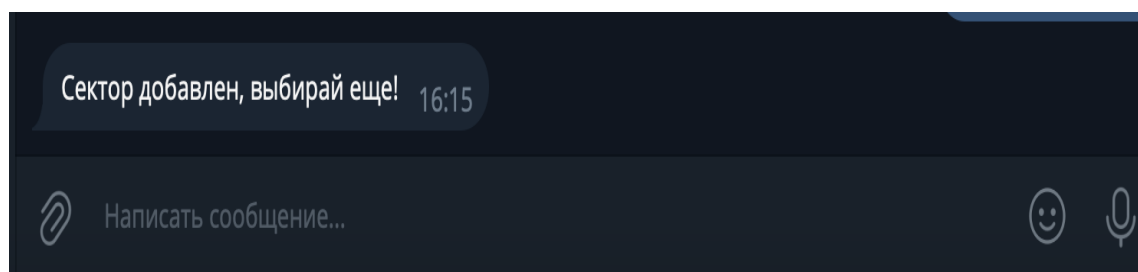


Рисунок 11 – Сообщение о добавление сектора инвестирования

Пользователь может выбрать до шести различных секторов. Это обеспечивает гибкость и разнообразие в инвестиционном портфеле. При повторном нажатии на команду «/add», ранее выбранные сектора не будут отображаться в списке, что предотвращает их повторное добавление. Пример списка секторов, которые еще не были выбраны пользователем, представлен на рисунке 12.

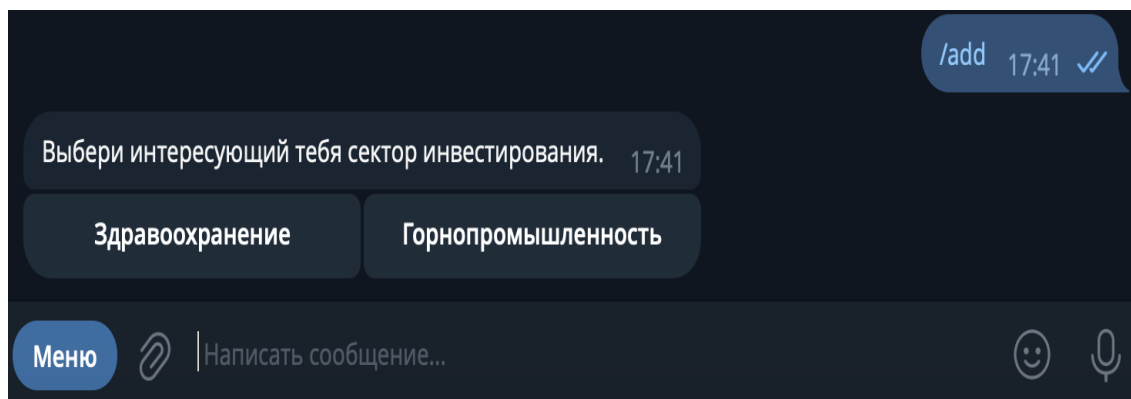


Рисунок 12 – Не выбранные сектора инвестирования

Если какой-то из выбранных секторов больше не интересует пользователя, его можно удалить. Для этого нужно ввести команду «/delete». Бот выведет список всех добавленных секторов, и пользователь может выбрать любой из них для удаления. Пример сообщения об удалении сектора инвестирования приведен на рисунке 13.

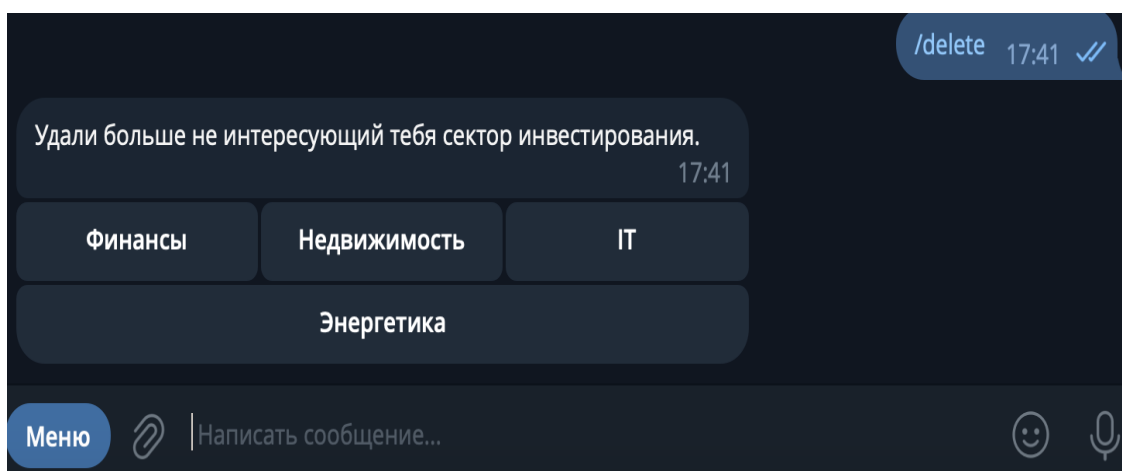


Рисунок 13 – Сообщение о удалении сектора инвестирования

После удаления сектора, бот подтвердит это действие сообщением и обновит внутренний список секторов, доступных для выбора. Это позволяет пользователю всегда держать свой инвестиционный портфель актуальным и соответствующим текущим интересам.

В течение дня могут приходить предложения о покупке акций. В сообщении будет говориться о стратегии, исходя из которой следует рекомендация, название акции и ее цена, а также кнопки «Купить» и «Отклонить». Данное сообщение представлено на рисунке 14.

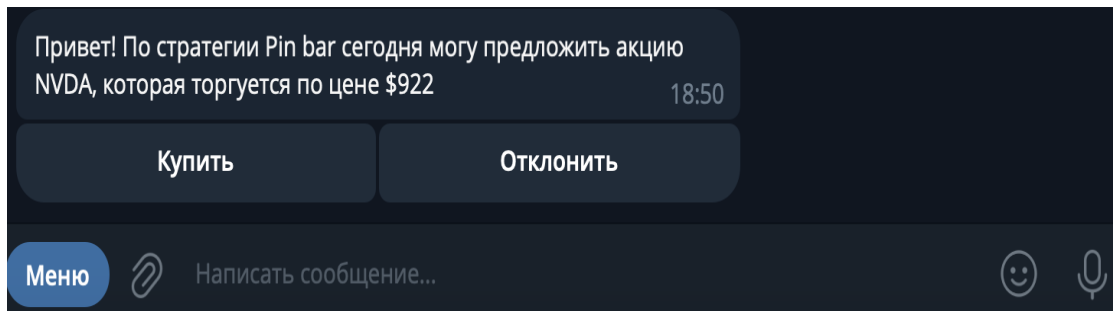


Рисунок 14 – Предложение купить акцию

Сообщение включает следующие элементы:

- стратегия. Бот указывает, по какой стратегии выдвинута данная рекомендация. В примере это стратегия Pin bar.
- название акции. В примере приводится акция компании NVDA,
- текущая цена. Указана актуальная цена акции на момент предложения, в данном случае \$922,
- кнопки выбора. Пользователь может выбрать между двумя кнопками – «Купить» и «Отклонить».

После того как пользователь подтвердит покупку акции, бот отправит уведомление о том, что акция успешно добавлена в портфель. Рисунок 15 демонстрирует пример такого уведомления.

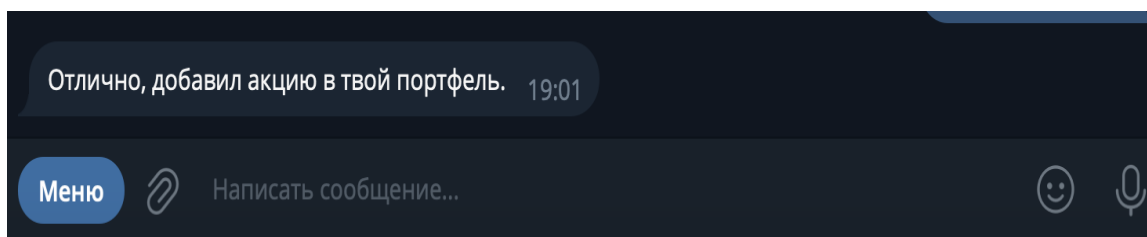


Рисунок 15 – Сообщение об успешной покупке акции

Если пользователь решает не покупать предложенную акцию, он может нажать кнопку «Отклонить». В этом случае акция не будет добавлена в портфель, и бот отправит соответствующее уведомление. Это показано на рисунке 16.

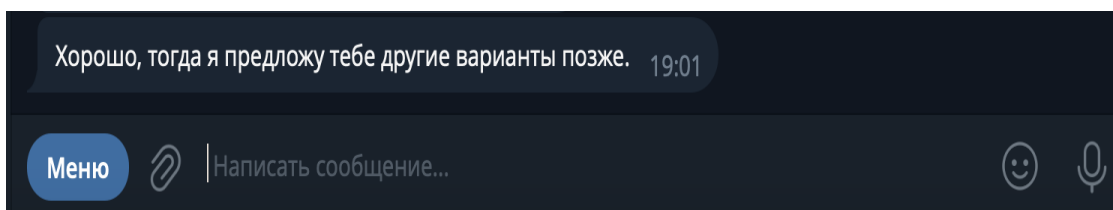


Рисунок 16 – Сообщение об отклонение предложения о покупке

Акции, добавленные в портфель, находятся под постоянным анализом бота. Бот использует различные стратегии для мониторинга изменений цен на акции. В случае достижения необходимого максимума цены бот предлагает пользователю зафиксировать прибыль. Данное предложение представлено на рисунке 17. Сообщение включает:

- уведомление о достижении максимума цены,
- рекомендацию продать акцию для фиксации прибыли.

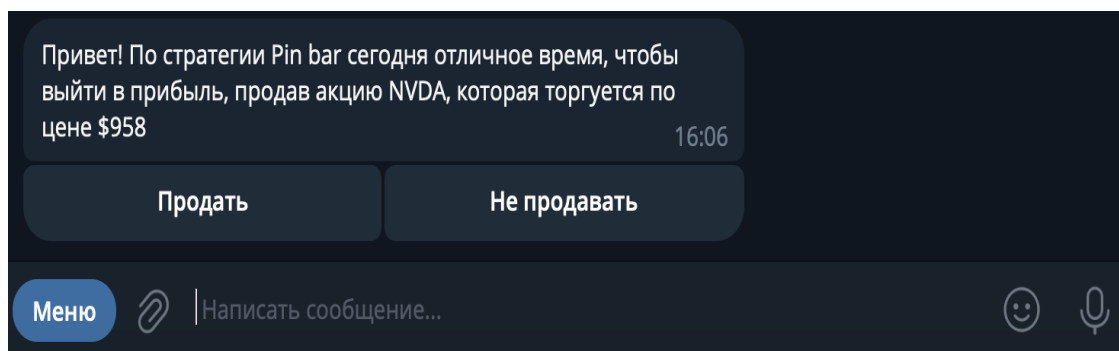


Рисунок 17 – Предложение зафиксировать прибыль по купленной акции

После продажи акции бот отправит сообщение, подтверждающее успешную продажу. Это уведомление приведено на рисунке 18.



Рисунок 18 – Сообщение об успешной продаже акции

Когда цена акции опускается ниже установленного стоп-уровня, бот немедленно уведомляет пользователя и предлагает продать акцию для минимизации убытков. Это предложение помогает пользователю быстро реагировать на неблагоприятные изменения на рынке и ограничивать потери. Пример такого сообщения от бота представлен на рисунке 19.

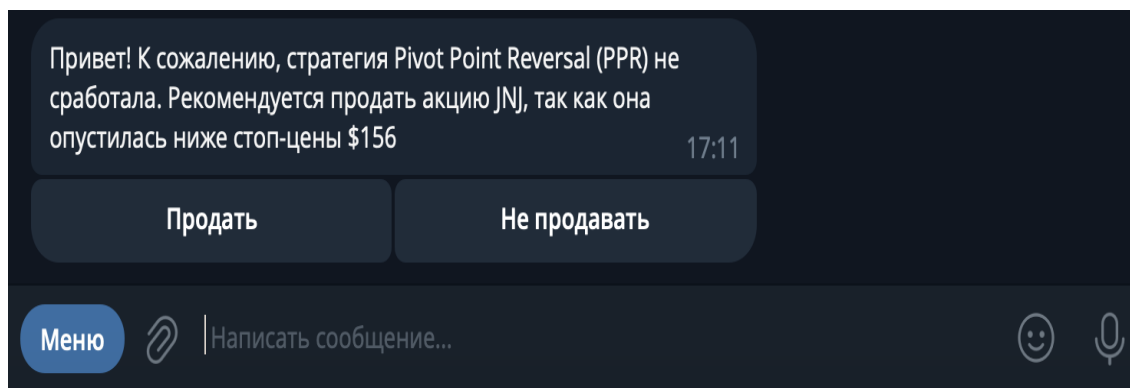


Рисунок 19 – Предложение зафиксировать убыток по купленной акции
Сообщение содержит следующие элементы:

- причина уведомления,
- рекомендация действия,
- текущая цена. В примере указано, что акция компании (JNJ) опустилась ниже \$156.
- кнопки выбора. Пользователь может выбрать между двумя кнопками – «Продать» и «Не продавать».

Для удобства пользователя бот предоставляет возможность в любой момент посмотреть список всех акций, находящихся в его портфеле. Это позволяет пользователю отслеживать свои акции и принимать необходимые решения.

Чтобы получить список акций, необходимо ввести команду «/stocks». Бот немедленно отобразит текущий портфель пользователя. Список акций включает в себя:

- название акций. Все акции, которые пользователь добавил в портфель.
- цена покупки Актуальная рыночная цена каждой акции.

Список акций показан на рисунке 20.



Рисунок 20 – Список акций

ЗАКЛЮЧЕНИЕ

В ходе выполнения настоящей работы выполнены следующие задачи:

- проведен анализ методов машинного обучения для прогнозирования движений фондового рынка,
- определена архитектура автоматизированной системы, включающая в себя интерфейс через бот в Telegram, микросервисы и разные способы взаимодействия между ними,
- собрано большое количество исторических биржевых котировок,
- на основе исторических котировок и различных стратегий инвестирования были сформированы датасеты для обучения модели,
- формирование обучающих данных для модели, обученной алгоритмом RandomForest.

Реализация системы анализа ценных бумаг была проведена с помощью языков программирования Java и Python с использованием объектно-ориентированной парадигмы программирования.

Вся тестовые данные подверглась предварительной обработки, что позволило оптимизировать объема необходимой памяти и вычислительных затрат в процессе обучения модели.

Была продемонстрирована работа бота в Telegram, а именно то, как на основе обученной модели он рекомендует инвестиционные предложения по покупке или продаже акций.

Полученные результаты позволят помочь инвесторам достичь своих финансовых целей. Данный подход может быть полезен в других системах и областях.

Предложенная система может быть расширена для работы с другими типами активов, такими как облигации или криптовалюты.

Планируется дальнейшее усовершенствование модели и включение новых методов машинного обучения для повышения точности прогнозов. Разработанная система также может быть применена для образовательных целей,

помогая студентам и исследователям в изучении алгоритмов машинного обучения и анализа данных. Таким образом, проект демонстрирует значительный потенциал для дальнейшего развития и применения в различных областях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сайт habr.com [электронный ресурс] URL: <https://habr.com/ru/articles/396505/> (дата обращения 14.01.2023).
2. Сайт [tadviser.com](https://www.tadviser.ru) [электронный ресурс] URL: <https://www.tadviser.ru/index.php> (дата обращения 21.02.2023).
3. Сайт <https://ft.com/> [электронный ресурс] URL: <https://www.ft.com/content/c8de53ac-6978-4b88-b7ed-7cdbcbfebbbf> (дата обращения 21.04.2024).
4. Рутковская, Д. Нейронные сети, генетические алгоритмы и нечеткие системы: Пер.с польск.И.Д.Рудинского : учебное пособие / Д. Рутковская, М. Пилиньский, Л. Рутковский. — 2-е изд. — Москва : Горячая линия-Телеком, 2013. — 384 с. Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/11843> (дата обращения: 25.03.2024).
5. Ruder S. An overview of gradient descent optimization algorithms //arXiv preprint arXiv:1609.04747. – 2016.
6. Hecht-Nielsen R. Theory of the backpropagation neural network //Neural networks for perception. – Academic Press, 1992. – С. 65-93.
7. Werbos P. J. Backpropagation through time: what it does and how to do it //Proceedings of the IEEE. – 1990. – Т. 78. – №. 10. – С. 1550-1560.
8. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение / пер. с англ. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.: цв. ил.
9. Wright L. G. et al. Deep physical neural networks trained with backpropagation //Nature. – 2022. – Т. 601. – №. 7894. – С. 549-555.
10. Jozefowicz R., Vinyals O., Schuster M., Shazeer N., Wu Y. Exploring the Limits of Language Modeling. 2016.
11. Singh A. K., Shashi M. Vectorization of text documents for identifying unifiable news articles //International Journal of Advanced Computer Science and Applications. – 2019. – Т. 10. – №. 7. <http://doi.org/10.14569/IJACSA.2019.0100742>

- 12.Thavareesan S., Mahesan S. Sentiment lexicon expansion using Word2vec and fastText for sentiment prediction in Tamil texts //2020 Moratuwa engineering research conference (MERCon). – IEEE, 2020
- 13.Ruder S. An overview of gradient descent optimization algorithms //arXiv preprint arXiv:1609.04747. – 2016.
- 14.Duchi J., Hazan E., Singer Y. Adaptive subgradient methods for online learning and stochastic optimization //Journal of machine learning research. – 2011. – Т. 12. – №. 7.
- 15.Hecht-Nielsen R. Theory of the backpropagation neural network //Neural networks for perception. – Academic Press, 1992. – С. 65-93.
- 16.Werbos P. J. Backpropagation through time: what it does and how to do it //Proceedings of the IEEE. – 1990. – Т. 78. – №. 10. – С. 1550-1560.
- 17.Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение / пер. с англ. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.: цв. ил.
- 18.Wright L. G. et al. Deep physical neural networks trained with backpropagation //Nature. – 2022. – Т. 601. – №. 7894. – С. 549-555.
- 19.Jozefowicz R., Vinyals O., Schuster M., Shazeer N., Wu Y. Exploring the Limits of Language Modeling. 2016.
- 20.ASPNET [Электронный ресурс]. URL: <https://dotnet.microsoft.com/en-us/apps/aspnet> (дата обращения: 22.03.2024)
- 21.Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Паттерны объектно-ориентированного проектирования. — СПб.: Питер, 2020. — 448 с. ISBN 978-5-4461-1595-2.
- 22.Nltk [Электронный ресурс]. URL: <https://www.nltk.org/> (дата обращения: 22.03.2024)
- 24.Fastapi [Электронный ресурс]. URL: <https://fastapi.tiangolo.com/> (дата обращения: 26.03.2024)
- 25.КиберЛенинка [Электронный ресурс]. URL: <https://cyberleninka.ru/> (дата обращения: 20.04.2024)

26.Российская государственная библиотека [Электронный ресурс]. URL: <https://www.rsl.ru/> (дата обращения: 20.04.2024)

27.UMAP [Электронный ресурс]. URL: <https://umap-learn.readthedocs.io/en/latest/index.html> (дата обращения: 19.05.2024)

ПРИЛОЖЕНИЕ А

Листинг

ПРИЛОЖЕНИЕ А.1

Файл «Insidebarstrategy.java»

```
@Component
public class InsideBarStrategy {

    public static List<String[]>
generateInsideBarDataset(List<Candle> candles) {
        List<String[]> dataset = new ArrayList<>();
        dataset.add(new String[]{"Body", "Range", "InsideRange",
"Success"});

        for (int i = 1; i < candles.size() - 1; i++) {
            Candle prev = candles.get(i - 1);
            Candle curr = candles.get(i);

            if (curr.getHigh() < prev.getHigh() && curr.getLow()
> prev.getLow()) {
                double body = Math.abs(curr.getClose() -
curr.getOpen());
                double range = curr.getHigh() - curr.getLow();
                double insideRange = range / (prev.getHigh() -
prev.getLow());

                boolean success = isSuccess(candles, i);
                dataset.add(new String[]{String.valueOf(body),
String.valueOf(range), String.valueOf(insideRange), success ?
"1" : "0"});
            }
        }
        return dataset;
    }

    private static boolean isSuccess(List<Candle> candles, int
index) {
        Candle insideBar = candles.get(index);
        double high = insideBar.getHigh();
        double low = insideBar.getLow();

        for (int i = index + 1; i < Math.min(index + 10,
candles.size()); i++) {
            Candle curr = candles.get(i);
            if (curr.getHigh() > high || curr.getLow() < low) {
                return true;
            }
        }
        return false;
    }
}
```

Приложение A.2

Файл «Pinbarstrategy.Java»

```
@Component
public class PinBarStrategy {

    private static List<String[]> generateDataset(List<Candle>
candles) {
        List<String[]> dataset = new ArrayList<>();
        dataset.add(new String[]{"Body", "UpperShadow",
"LowerShadow", "Success"});

        for (int i = 1; i < candles.size() - 1; i++) {
            Candle curr = candles.get(i);

            double body = Math.abs(curr.getClose() -
curr.getOpen());
            double upperShadow = curr.getHigh() -
Math.max(curr.getClose(), curr.getOpen());
            double lowerShadow = Math.min(curr.getClose(),
curr.getOpen()) - curr.getLow();

            boolean isPinBar = (body < upperShadow &&
upperShadow > lowerShadow * 2) ||
                (body < lowerShadow && lowerShadow >
upperShadow * 2);

            if (isPinBar) {
                boolean success = isSuccess(candles, i);
                dataset.add(new String[]{String.valueOf(body),
String.valueOf(upperShadow), String.valueOf(lowerShadow),
success ? "1" : "0"});
            }
        }
        return dataset;
    }

    private static boolean isSuccess(List<Candle> candles, int
index) {
        Candle pinBar = candles.get(index);
        double fibHigh = pinBar.getHigh();
        double fibLow = pinBar.getLow();
        double fibLevel61_8 = fibLow + (fibHigh - fibLow) *
0.618;

        for (int i = index + 1; i < Math.min(index + 10,
candles.size()); i++) {
            if (candles.get(i).getClose() > fibLevel61_8) {
                return true;
            }
        }
    }
}
```

```
        return false;
    }
}
```

Приложение А.3

Файл «PPRStrategy.java»

```
@Component
public class PPRStrategy {

    public static List<String[]> generatePPRDataset(List<Candle>
candles) {
        List<String[]> dataset = new ArrayList<>();
        dataset.add(new String[]{"Body1", "Body2", "Body3",
"LowDiff", "CloseDiff", "Success"});

        for (int i = 2; i < candles.size() - 1; i++) {
            Candle first = candles.get(i - 2);
            Candle second = candles.get(i - 1);
            Candle third = candles.get(i);

            if (second.getLow() < first.getLow() &&
third.getClose() > second.getClose()) {
                double body1 = Math.abs(first.getClose() -
first.getOpen());
                double body2 = Math.abs(second.getClose() -
second.getOpen());
                double body3 = Math.abs(third.getClose() -
third.getOpen());
                double lowDiff = first.getLow() -
second.getLow();
                double closeDiff = third.getClose() -
second.getClose();

                boolean success = isSuccess(candles, i);
                dataset.add(new String[]{
                    String.valueOf(body1),
                    String.valueOf(body2),
                    String.valueOf(body3),
                    String.valueOf(lowDiff),
                    String.valueOf(closeDiff),
                    success ? "1" : "0"});
            }
        }
        return dataset;
    }

    private static boolean isSuccess(List<Candle> candles, int
index) {
        Candle third = candles.get(index);
        double high = third.getHigh();
        double low = third.getLow();

        for (int i = index + 1; i < Math.min(index + 10,
```

```
candles.size()); i++) {  
    Candle curr = candles.get(i);  
    if (curr.getHigh() > high || curr.getLow() < low) {  
        return true;  
    }  
}  
return false;  
}  
}
```

Приложение A.4

Файл «FakeyStrategy.java»

```
public class FakeyStrategy {

    public static List<String[]>
generateFakeyDataset(List<Candle> candles) {
        List<String[]> dataset = new ArrayList<>();
        dataset.add(new String[]{"Body", "Range", "FakeyRange",
"Success"});

        for (int i = 2; i < candles.size() - 1; i++) {
            Candle prev = candles.get(i - 1);
            Candle curr = candles.get(i);

            // Проверка на ложное пробитие (Fakey)
            if ((curr.getHigh() > prev.getHigh() &&
curr.getClose() < prev.getClose()) ||
                (curr.getLow() < prev.getLow() &&
curr.getClose() > prev.getClose())) {

                double body = Math.abs(curr.getClose() -
curr.getOpen());
                double range = curr.getHigh() - curr.getLow();
                double fakeyRange = range / (prev.getHigh() -
prev.getLow());

                boolean success = isFakeySuccess(candles, i,
prev);

                dataset.add(new String[]{String.valueOf(body),
String.valueOf(range), String.valueOf(fakeyRange), success ? "1"
: "0"});
            }
        }
        return dataset;
    }

    private static boolean isFakeySuccess(List<Candle> candles,
int index, Candle prevCandle) {
        Candle fakeyCandle = candles.get(index);
        double targetUp = fakeyCandle.getLow() -
(prevCandle.getHigh() - prevCandle.getLow());
        double targetDown = fakeyCandle.getHigh() +
(prevCandle.getHigh() - prevCandle.getLow());

        for (int i = index + 1; i < Math.min(index + 10,
candles.size()); i++) {
            Candle curr = candles.get(i);
            if (curr.getLow() <= targetUp || curr.getHigh() >=
targetDown) {
                return true;
            }
        }
    }
}
```

```
        }  
    }  
    return false;  
}  
  
}
```

Приложение A.5

Файл «Candle.java»

```
package ru.diplom.stock.model;

import lombok.Data;

@Data
public class Candle {
    private double open;
    private double high;
    private double low;
    private double close;
}
```

Приложение А.6

Файл «UserController.java»

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/user/")
public class UserController {

    private final UserService userService;

    @PostMapping("/create/{chatId}")
    public boolean createUser(@PathVariable Long chatId) {
        return userService.saveUser(chatId);
    }

    @GetMapping("/find/{chatId}")
    public UserDto findUser(@PathVariable Long chatId) {
        return userService.findUser(chatId);
    }

    @GetMapping("/findAllUser")
    public List<UserDto> findAllUser() {
        return userService.findAllUsers();
    }

    @PutMapping("/update/")
    public void updateUser(@RequestBody UserDto user) {
        userService.updateUser(user);
    }
}
```

Приложение А.7

Файл «UserService.java»

```
@Component
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;

    public UserDto findUser(Long chatId) {
        var user = userRepository.findById(chatId).orElse(null);
        if (user != null) {
            return User.toUserDto(user);
        }
        return null;
    }

    public List<UserDto> findAllUsers() {
        var users = userRepository.findAll();
        List<UserDto> userDtos = new ArrayList<>();
        users.forEach(user -> {
            userDtos.add(User.toUserDto(user));
        });
        return userDtos;
    }

    public boolean saveUser(Long chatId) {
        if (userRepository.findById(chatId).isEmpty()) {
            var user = new User().setChatId(chatId);
            userRepository.save(user);
            return true;
        }
        return false;
    }

    public void updateUser(UserDto userDto) {
        userRepository.save(User.toUser(userDto));
    }
}
```

Приложение А.8

Файл «UserRepository.java»

```
package ru.diplom.stock.repository;

import org.springframework.data.repository.CrudRepository;
import ru.diplom.stock.model.entity.User;

public interface UserRepository extends CrudRepository<User,
Long> {
}
```

Приложение А.9

Файл «User.java»

```
@Entity
@Data
@NoArgsConstructor
@Table(name = "users")
@Accessors(chain = true)
public class User {
    @Id
    private Long chatId;
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL,
orphanRemoval = true, fetch = FetchType.EAGER)
    private List<Industry> industries;

    public static User toUser(UserDto userDto) {
        var user = new User().setChatId(userDto.getChatId());
        var industriesDto = userDto.getIndustries();
        var industries = new ArrayList<Industry>();
        if (industriesDto != null) {
            industriesDto.forEach(industryDto -> {
                var industry = new Industry()
                    .setName(industryDto.getName())
                    .setUser(user);
                var stocks = new ArrayList<Stock>();
                var stocksDto = industryDto.getStocks();
                if (stocksDto != null) {
                    stocksDto.forEach(stockDto -> {
                        var stock = new Stock()

.setStockName(stockDto.getStockName())

.setPurchasePrice(stockDto.getPurchasePrice())

.setProfitPrice(stockDto.getProfitPrice())

.setStopPrice(stockDto.getStopPrice())
                            .setIndustry(industry);
                        stocks.add(stock);
                    });
                }
                industry.setStocks(stocks);
                industries.add(industry);
            });
        }
        return user.setIndustries(industries);
    }

    public static UserDto toUserDto(User user) {
        var userDto = new UserDto().setChatId(user.getChatId());
        var industries = user.getIndustries();
```

```

        var industryDtos = new ArrayList<IndustryDto>();
        if (industries != null) {
            industries.forEach(industryDto -> {
                var stocksDto = new ArrayList<StockDto>();
                var stocks = industryDto.getStocks();
                if (stocks != null) {
                    stocks.forEach(stock -> {
                        var stockDto = new StockDto();

stockDto.setStockName(stock.getStockName())
                        .setId(stock.getId())

                        .setPurchasePrice(stock.getPurchasePrice())

                        .setStopPrice(stock.getStopPrice())

                        .setProfitPrice(stock.getProfitPrice());
                        stocksDto.add(stockDto);
                    });
                }
                var industry = new IndustryDto()
                    .setName(industryDto.getName())
                    .setStocks(stocksDto);
                industryDtos.add(industry);
            });
        }
        return userDto.setIndustries(industryDtos);
    }
}

```

Приложение А.10

Файл «RandomForestClassifier.py»

```
import pandas as pd
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from joblib import dump, load

df = pd.read_csv('dataset.csv')

X = df[['Body', 'UpperShadow', 'LowerShadow', 'Range',
'InsideRange', 'Body2', 'Body3', 'LowDiff', 'FakeyRange',
'trend10Days']]
y = df['successful']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

model = GradientBoostingClassifier(n_estimators=100,
learning_rate=0.1, max_depth=3, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')

dump(model, 'gradient_boosting_model.joblib')
```

Приложение А.11

Файл «GradientBoostingClassifier.py»

```
import pandas as pd
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from joblib import dump, load

df = pd.read_csv('dataset.csv')

X = df[['Body', 'UpperShadow', 'LowerShadow', 'Range',
'InsideRange', 'Body2', 'Body3', 'LowDiff', 'FakeyRange',
'trend10Days']]
y = df['successful']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

model = GradientBoostingClassifier(n_estimators=100,
learning_rate=0.1, max_depth=3, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')

dump(model, 'gradient_boosting_model.joblib')
```

Приложение A.12

Файл «PortfolioManagerService.java»

```
@Component
@RequiredArgsConstructor
public class PortfolioManagerService {

    private final RestTemplate restTemplate = new
RestTemplate();
    private final static String BASE_URL =
"http://localhost:9090/api/user/";

    public boolean createUser(@PathVariable Long chatId) {
        ResponseEntity<Boolean> response =
restTemplate.exchange(BASE_URL + "/create/" + chatId,
HttpMethod.POST, null, Boolean.class);
        return Boolean.TRUE.equals(response.getBody());
    }

    public User findUser(@PathVariable Long chatId) {
        return restTemplate.getForObject(BASE_URL + "/find/" +
chatId, User.class);
    }

    public void updateUser(@RequestBody User user) {
        restTemplate.exchange(BASE_URL + "/update/",
HttpMethod.PUT, new HttpEntity<>(user), Void.class);
    }
}
```

Приложение A.13

Файл «MessageService.java»

```
@Slf4j
@Component
@RequiredArgsConstructor
public class MessageService {

    private final PortfolioManagerService
portfolioManagerService;

    public SendMessage getAddIndustryMessage(long chatId,
MessageEnum messageEnum) {
        var industriesEnums = getIndustriesEnums(chatId);

        var message = new SendMessage();
        message.setChatId(String.valueOf(chatId));
        message.setText(messageEnum.getMessage());

        var markupInLine = new InlineKeyboardMarkup();
        List<List<InlineKeyboardButton>> rowsInLine = new
ArrayList<>();

        if (industriesEnums.size() <= 3)

rowsInLine.add(getInlineKeyboardButton(industriesEnums.subList(0
, industriesEnums.size())));
            if (industriesEnums.size() > 3) {

rowsInLine.add(getInlineKeyboardButton(industriesEnums.subList(0
, 3)));

rowsInLine.add(getInlineKeyboardButton(industriesEnums.subList(3
, industriesEnums.size())));
            }

        markupInLine.setKeyboard(rowsInLine);
        message.setReplyMarkup(markupInLine);

        return message;
    }

    @SneakyThrows
    public List<SendMessage> getStockSale(StockOffer stockOffer)
{
        List<SendMessage> sendMessages = new ArrayList<>();

        var message = new SendMessage();
        List<InlineKeyboardButton> row1 = new ArrayList<>();
        row1.add(initInitInlineKeyboardButton("Продать",
```

```

        "Продать"));
        row1.add(initInitInlineKeyboardButton("Не продавать",
        "Не продавать"));
        List<List<InlineKeyboardButton>> rowsInLine = new
        ArrayList<>();
        rowsInLine.add(row1);

        stockOffer.getChatIds().forEach(chatId -> {
            message.setChatId(String.valueOf(chatId));
            String messagee;
            if(stockOffer.getIsPositive()){
                messagee = String.format("Привет! По стратегии
        %s сегодня отличное время, чтобы выйти в прибыль, продав акцию
        %s, которая торгуется по цене %s", stockOffer.getStrategy(),
        stockOffer.getStockName(), stockOffer.getPurchasePrice());
            } else {
                messagee = String.format("Привет! К сожалению,
        стратегия %s не сработала. Рекомендуется продать акцию %s, так
        как она опустилась ниже стоп-цены %s",
        stockOffer.getStrategy(), stockOffer.getStockName(),
        stockOffer.getPurchasePrice());
            }
            message.setText(messagee);
            var markupInLine = new InlineKeyboardMarkup();
            markupInLine.setKeyboard(rowsInLine);
            message.setReplyMarkup(markupInLine);
            sendMessages.add(message);
        });

        return sendMessages;
    }

    @SneakyThrows
    public List<SendMessage> getStockOfferMessage(StockOffer
    stockOffer) {
        List<SendMessage> sendMessages = new ArrayList<>();

        var message = new SendMessage();
        List<InlineKeyboardButton> row1 = new ArrayList<>();
        row1.add(initInitInlineKeyboardButton("Купить",
        "Купить"));
        row1.add(initInitInlineKeyboardButton("Отклонить",
        "Отклонить"));
        List<List<InlineKeyboardButton>> rowsInLine = new
        ArrayList<>();
        rowsInLine.add(row1);

        stockOffer.getChatIds().forEach(chatId -> {
            message.setChatId(String.valueOf(chatId));
            message.setText(String.format("Привет! По стратегии
        %s сегодня могу предложить акцию %s, которая торгуется по цене
        %s", stockOffer.getStrategy(), stockOffer.getStockName(),

```

```

stockOffer.getPurchasePrice()));
        var markupInLine = new InlineKeyboardMarkup();
        markupInLine.setKeyboard(rowsInLine);
        message.setReplyMarkup(markupInLine);
        sendMessages.add(message);
    });

    return sendMessages;
}

public SendMessage getDeleteIndustryMessage(long chatId,
MessageEnum messageEnum) {
    var industriesEnums = getDeletedIndustriesEnums(chatId);

    var message = new SendMessage();
    message.setChatId(String.valueOf(chatId));
    message.setText(messageEnum.getMessage());

    var markupInLine = new InlineKeyboardMarkup();
    List<List<InlineKeyboardButton>> rowsInLine = new
ArrayList<>();

    if (industriesEnums.size() <= 3)

rowsInLine.add(getDeletedInlineKeyboardButton(industriesEnums.su
bList(0, industriesEnums.size())));
        if (industriesEnums.size() > 3) {

rowsInLine.add(getDeletedInlineKeyboardButton(industriesEnums.su
bList(0, 3)));

rowsInLine.add(getDeletedInlineKeyboardButton(industriesEnums.su
bList(3, industriesEnums.size())));
        }

    markupInLine.setKeyboard(rowsInLine);
    message.setReplyMarkup(markupInLine);

    return message;
}

public SendMessage getStocks(Long chatId) {
    var user = portfolioManagerService.findUser(chatId);
    StringBuilder stringBuilder = new StringBuilder();
    Optional.ofNullable(user.getIndustries())
        .orElse(Collections.emptyList())
        .stream()
        .map(Industry::getStocks)
        .filter(stocks1 -> stocks1 != null &&
!stocks1.isEmpty())
        .flatMap(Collection::stream)
        .forEach(stock -> {

```

```

stringBuilder.append(stock.getStockName()).append(", цена
покупки: ").append(stock.getPurchasePrice()).append("$\n");
    });

    SendMessage sendMessage = new SendMessage();
    sendMessage.setChatId(String.valueOf(chatId));

    if (stringBuilder.isEmpty()) {
        sendMessage.setText("Акции еще не приобретено.");
    } else {
        sendMessage.setText(stringBuilder.toString());
    }
    return sendMessage;
}

private List<InlineKeyboardButton>
getInlineKeyboardButton(List<IndustriesEnum> industriesEnums) {
    List<InlineKeyboardButton> row1 = new ArrayList<>();
    industriesEnums.forEach(industriesEnum ->
row1.add(initInitInlineKeyboardButton(industriesEnum.getText(),
industriesEnum.getCallback())));
    return row1;
}

private List<InlineKeyboardButton>
getDeletedInlineKeyboardButton(List<IndustriesEnum>
industriesEnums) {
    List<InlineKeyboardButton> row1 = new ArrayList<>();
    industriesEnums.forEach(industriesEnum ->
row1.add(initInitInlineKeyboardButton(industriesEnum.getText(),
"delete " + industriesEnum.getCallback())));
    return row1;
}

private List<IndustriesEnum> getIndustriesEnums(long chatId)
{
    var user = portfolioManagerService.findUser(chatId);

    List<IndustriesEnum> industriesEnums =
Arrays.stream(values()).toList();
    if (user != null) {
        var industriesUser = user.getIndustries();
        industriesEnums = industriesEnums.stream()
            .filter(industriesEnum ->
!isExistIndustry(industriesUser, industriesEnum))
            .toList();
    }
    return industriesEnums;
}

private List<IndustriesEnum> getDeletedIndustriesEnums(long

```

```

chatId) {
    var user = portfolioManagerService.findUser(chatId);

    List<Industry> industries = user.getIndustries();
    List<IndustriesEnum> industriesEnums = new
ArrayList<>();
    industries.forEach(industry -> {
        industriesEnums.add(valueOf(industry.getName()));
    });

    return industriesEnums;
}

private boolean isExistIndustry(List<Industry> industries,
IndustriesEnum industriesEnum) {
    return industries.stream()
        .anyMatch(industry ->
industry.getName().equals(industriesEnum.getCallback()));
}

private InlineKeyboardButton
initInitInlineKeyboardButton(String text, String callback) {
    var inlineKeyboardButton = new InlineKeyboardButton();
    inlineKeyboardButton.setText(text);
    inlineKeyboardButton.setCallbackData(callback);
    return inlineKeyboardButton;
}
}

```

Приложение А.14

Файл «TelegramBot.java»

```
@Slf4j
@Component
public class TelegramBot extends TelegramLongPollingBot {

    final BotConfig config;

    @Autowired
    private MessageService messageService;
    @Autowired
    private PortfolioManagerService portfolioManagerService;

    public TelegramBot(BotConfig config) {
        this.config = config;
        initListOfCommands(of(START));
    }

    @Override
    public String getBotUsername() {
        return config.getBotName();
    }

    @Override
    public String getBotToken() {
        return config.getToken();
    }

    @Override
    public void onUpdateReceived(Update update) {
        if (update.hasMessage() &&
update.getMessage().hasText()) {
            var messageText = update.getMessage().getText();
            var chatId = update.getMessage().getChatId();

            SendMessage message = null;
            switch (messageText) {
                case "/start":
                    portfolioManagerService.createUser(chatId);
                    message = new SendMessage();
                    message.setChatId(String.valueOf(chatId));
                    message.setText(MESSAGE_START.getMessage());
                    executeMessage(message);
                    initListOfCommands(of(ADD, DELETE, STOCKS));
                    break;
                case "/add":
                    message =
messageService.getAddIndustryMessage(chatId, MESSAGE_ADD);
                    executeMessage(message);
            }
        }
    }
}
```

```

        break;
        case "/delete":
            message =
messageService.getDeleteIndustryMessage(chatId, MESSAGE_DELETE);
            executeMessage(message);
            break;
        case "/stocks":
            message = messageService.getStocks(chatId);
            executeMessage(message);
            break;
    }
} else if (update.hasCallbackQuery()) {
    processCallback(update);
}

}

public void sendOffers(StockOffer stockOffer) {
    if(stockOffer.getIsOffer()) {
        var messages =
messageService.getStockOfferMessage(stockOffer);
        messages.forEach(this::executeMessage);
        log.info("Акция сохранена");
        stockOffer.getChatIds().forEach(chatId ->
processBuyStock(chatId, stockOffer));
    } else {
        var messages =
messageService.getStockSale(stockOffer);
        messages.forEach(this::executeMessage);
        stockOffer.getChatIds().forEach(chatId ->
processDeleteStock(chatId, stockOffer));
    }
}

public void processCallback(Update update) {
    var callbackData = update.getCallbackQuery().getData();
    var messageId =
update.getCallbackQuery().getMessage().getMessageId();
    var chatId =
update.getCallbackQuery().getMessage().getChatId();

    if (callbackData.contains("Купить")) {
        executeEditMessageText("Отлично, добавил акцию в
твой портфель.", chatId, messageId);
    } else if (callbackData.contains("Отклонить")) {
        executeEditMessageText("Хорошо, тогда я предложу
тебе другие варианты позже.", chatId, messageId);
    } else if (callbackData.contains("Продать")) {
        executeEditMessageText("Акция продана!", chatId,
messageId);
    } else if (callbackData.contains("Не продавать")) {
        executeEditMessageText("Хорошо, позже дам другие
сигналы для продажи", chatId, messageId);
    }
}

```

```

        }
        else if (callbackData.contains("delete ")) {
            var industriesEnum =
IndustriesEnum.valueOf(callbackData.replaceAll("delete ", ""));
            processDeleteIndustries(messageId, chatId,
industriesEnum);
        } else {
            var industriesEnum =
IndustriesEnum.valueOf(callbackData);
            processAddIndustries(messageId, chatId,
industriesEnum);
        }
    }

    private void processDeleteStock(Long chatId, StockOffer
stockOffer) {
        var user = portfolioManagerService.findUser(chatId);
        var suitableIndustry = user.getIndustries()
            .stream()
            .filter(industry ->
industry.getName().equals(stockOffer.getIndustry()))
            .findAny()
            .orElse(null);

        suitableIndustry.setStocks(suitableIndustry.getStocks()
            .stream()
            .filter(stock ->
!stock.getStockName().equals(stockOffer.getStockName()))
            .toList());

        portfolioManagerService.updateUser(user);
    }

    private void processBuyStock(Long chatId, StockOffer
stockOffer) {
        var user = portfolioManagerService.findUser(chatId);
        var industries = user.getIndustries();
        var suitableIndustry = Optional.ofNullable(industries)
            .orElse(List.of(new
Industry().setName(stockOffer.getIndustry()).setId(UUID.randomUUID()
ID()))))
            .stream()
            .filter(industry ->
industry.getName().equals(stockOffer.getIndustry()))
            .findAny()
            .orElse(new
Industry().setName(stockOffer.getIndustry()).setId(UUID.randomUUID()
ID())));

        var stock = new Stock()
            .setStockName(stockOffer.getStockName())

```

```

.setPurchasePrice(stockOffer.getPurchasePrice());

        if (suitableIndustry.getStocks() != null) {
            suitableIndustry.getStocks().add(stock);
        } else {
            suitableIndustry.setStocks(List.of(stock));
        }

        portfolioManagerService.updateUser(user);
    }

    private void processDeleteIndustries(Integer messageId, Long
chatId, IndustriesEnum industriesEnum) {
        var text = "Данный сектор удален!";
        if (of(IT, FINANCE, ENERGY, REAL_ESTATE,
MINING_INDUSTRY, HEALTHCARE).contains(industriesEnum)) {
            var user = portfolioManagerService.findUser(chatId);

            user.setIndustries(user.getIndustries().stream().filter(industry
->
!industriesEnum.getCallback().equals(industry.getName()))).toList
());

            portfolioManagerService.updateUser(user);
            executeEditMessageText(text, chatId, messageId);
        }
    }

    private void processAddIndustries(Integer messageId, Long
chatId, IndustriesEnum industriesEnum) {
        var text = "Сектор добавлен, выбирай еще!";
        if (of(IT, FINANCE, ENERGY, REAL_ESTATE,
MINING_INDUSTRY, HEALTHCARE).contains(industriesEnum)) {
            var user = portfolioManagerService.findUser(chatId);
            user.getIndustries().add(new
Industry().setName(industriesEnum.getCallback()));
            portfolioManagerService.updateUser(user);
            executeEditMessageText(text, chatId, messageId);
        }
    }

    private void initListOfCommands(List<BotCommandsEnum>
botCommands) {
        List<BotCommand> listOfCommands = new ArrayList<>();
        botCommands.forEach(botCommandsEnum ->
listOfCommands.add(new BotCommand(botCommandsEnum.getCommand(),
botCommandsEnum.getDescription())));
        try {
            this.execute(new SetMyCommands(listOfCommands, new
BotCommandScopeDefault(), null));
        } catch (TelegramApiException e) {
            log.error("Error setting bot's command list: " +

```

```

e.getMessage());
    }
}

private void executeEditMessageText(String text, long
chatId, long messageId) {
    EditMessageText message = new EditMessageText();
    message.setChatId(String.valueOf(chatId));
    message.setText(text);
    message.setMessageId((int) messageId);

    try {
        execute(message);
    } catch (TelegramApiException e) {
        log.error(e.getMessage());
    }
}

private void executeMessage(SendMessage message) {
    try {
        execute(message);
    } catch (TelegramApiException e) {
        log.error(e.getMessage());
        throw new RuntimeException(e);
    }
}
}

```

Приложение А.15

Файл «BotComandsEnum.java»

```
@Getter
@RequiredArgsConstructor
public enum BotComandsEnum {
    START("/start", "Привет, бот! Я хочу начать инвестировать в
акции, но не знаю, с чего начать."),
    ADD("/add", "Добавить сектор инвестирования"),
    DELETE("/delete", "Удалить сектор инвестирования"),
    STOCKS("/stocks", "Вывести список купленных акций");

    private final String command;
    private final String description;
}
```

Приложение А.16

Файл «MessageEnum.java»

```
@RequiredArgsConstructor
@Getter
public enum MessageEnum {
    MESSAGE_START("Привет! Я могу помочь тебе с выбором акций и  
инвестиционными рекомендациями. Давай начнем с выбора секторов  
инвестирования. Для этого введи команду /add "),
    MESSAGE_ADD("Выбери интересующий тебя сектор  
инвестирования."),
    MESSAGE_DELETE("Удали больше не интересующий тебя сектор  
инвестирования.");

    private final String message;
}
```

ПРИЛОЖЕНИЕ Б

Презентация

Система анализа ценных бумаг

Автор работы:

Студент гр. 22ВВИм1 Бахтияров Руслан Рафикович

Научный руководитель:

к.т.н., доцент каф. ВТ ПГУ Карамышева Н.С

Цель и объект работы

Цель работы – разработать систему анализа ценных бумаг.

Объект работы – система, которая позволяет:

- прогнозировать движение акции на фондовом рынке на основе индикаторных стратегий с помощью моделей машинного обучения;
- взаимодействовать с пользователем через Telegram-бот
- выбирать желаемую область инвестирования, которая будет предложена чат-ботом

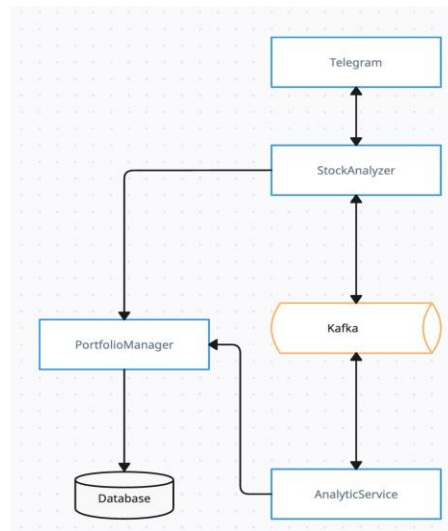
Популярные решения



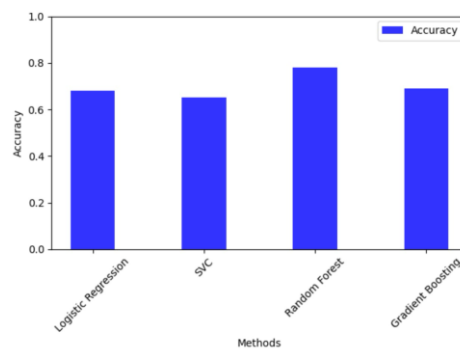
Стек технологий



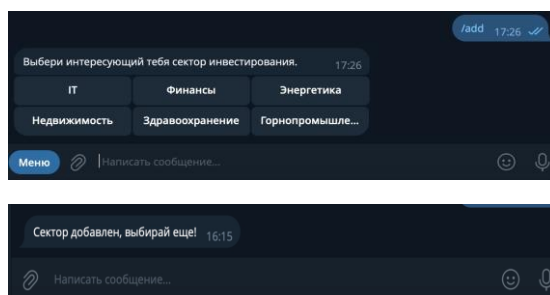
Архитектура системы



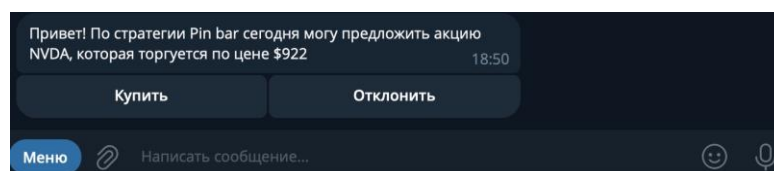
Статистика точности прогнозирования при разных методах обучения



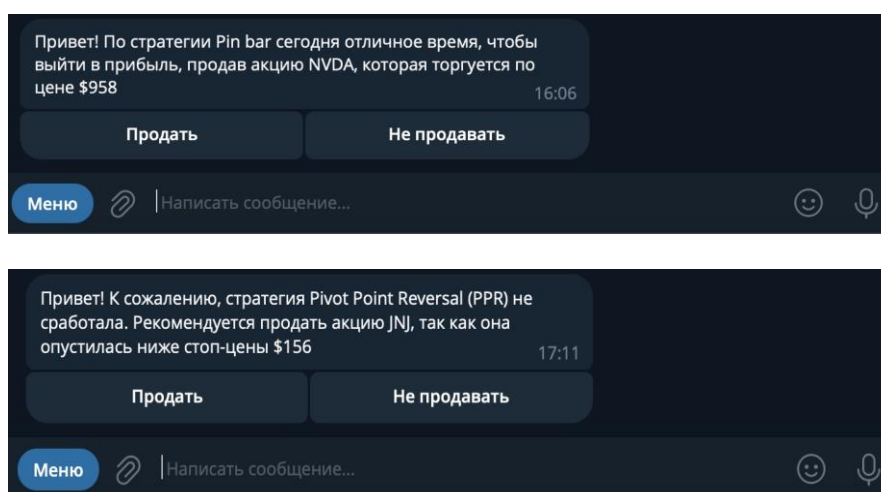
Выбор сектора инвестирования



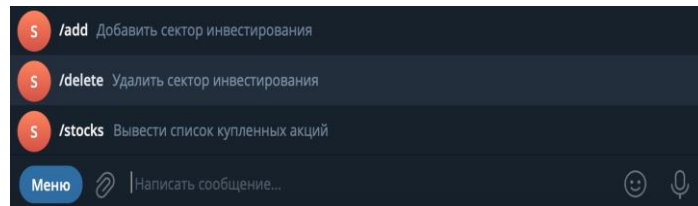
Предложения о покупке



Предложения о продаже



Меню бота



Благодарю за
внимание!