Министерство образования и науки Нижегородской области

Государственное бюджетное образовательное учреждение

высшего образования

«Нижегородский государственный инженерно-экономический университет»

Институт: «Информационные технологии и системы связи»

Кафедра: «Математика и вычислительная техника»

**КУРСОВОЙ ПРОЕКТ**

дисциплина «Технологии анализа данных»

Тема «Анализ эффективности алгоритмов нормализации наименований»

Выполнил: студент 3 курса

группы 22 ИВО

Вахнин Алексей Николаевич

Проверил: доцент кафедры

Черёмухин А.Д

Проект защищен\_\_\_. \_\_\_. 2025г.

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Д Черёмухин

(подпись)

г. Княгинино

2025 г.

**Содержание**

[**Введение** 3](#_Toc187607293)

[**Глава 1. Планирование анализа данных** 4](#_Toc187607294)

[**1.1.** **Обзор литературы по теме "Анализ эффективности алгоритмов нормализации наименований".** 4](#_Toc187607295)

[**1.2.** **Построение датафрейма с информацией** 5](#_Toc187607296)

[**Глава 2. Организация и проведения анализа данных** 8](#_Toc187607297)

[**2.1. Разведочный анализ данных.** 8](#_Toc187607298)

[**2.2. Методологические основы анализа** 10](#_Toc187607299)

[**2.3. Проведение анализа данных** 13](#_Toc187607300)

[**Глава 3. Оформление результатов анализа данных** 16](#_Toc187607301)

[**3.1. Результаты проведенного анализа** 16](#_Toc187607302)

[**3.2. Формирование практических предложений** 17](#_Toc187607303)

[**Заключение** 19](#_Toc187607304)

[**Список литературы** 20](#_Toc187607305)

[**Приложение** 21](#_Toc187607306)

# **Введение**

В условиях стремительного увеличения объема данных, их разнообразия и сложности, нормализация информации становится ключевым процессом для обеспечения точности и согласованности данных. Особую актуальность эта задача приобретает в области обработки текстовых данных, где различия в написании наименований, вариативность формулировок и ошибки могут существенно затруднить дальнейшую работу с информацией.

Современные подходы к нормализации используют технологии обработки текстов и алгоритмы машинного обучения, в частности, глубокие нейронные сети, которые позволяют учитывать контекст, изменчивость форм записи и другие характеристики данных.

Целью настоящей работы является анализ эффективности алгоритмов нормализации наименований товаров, с акцентом на использование современных методов машинного обучения и обработки текста.

Для достижения поставленной цели необходимо решить несколько ключевых задач, включая обзор существующих методов, реализацию тестирования выбранных алгоритмов на реальных данных и их дальнейший анализ с целью выявления наиболее эффективных подходов для различных типов данных.

Объектом исследования являются процессы нормализации текстовых данных.

Предметом исследования являются алгоритмы и подходы, применяемые для нормализации наименований товаров.

# **Глава 1. Планирование анализа данных**

* 1. **Обзор литературы по теме "Анализ эффективности алгоритмов нормализации наименований".**

Для анализа эффективности алгоритмов нормализации наименований был проведен обзор научных публикаций. Основной целью данного анализа является выявление подходов, используемых для нормализации текстовых данных. Обзор литературы включает исследования, где применялись как традиционные алгоритмы нормализации, так и современные методы на базе машинного обучения. В обзоре представлено сравнение различных подходов, таких как BERT, CRF, скрытые марковские модели, гибридные системы.

"A Deep Learning Model for the Normalization of Institution Names by Multi-Source Literature Feature Fusion: Algorithm Development"[1] - рассматривается использование моделей глубокого обучения, таких как BERT, для нормализации наименований учреждений, которые встречаются в крупных научных базах данных, таких как Dimensions, Scopus и Web of Science. Основной задачей было унифицировать различные формы написания названий университетов и исследовательских институтов. Модель BERT была обучена на большом количестве данных, что позволило достигнуть точности 93.79% в нормализации названий. Это исследование показало, что методы глубокого обучения превосходят классические алгоритмы нормализации за счет способности учитывать контекст и разнообразие форм наименований.

"Gene name identification and normalization using a model organism database"[2] - для нормализации названий генов применяются скрытые марковские модели (HMM). Данные генов и их упоминания извлекались из биомедицинских публикаций, включая данные из FlyBase, а целью было объединение вариативных названий одного гена для обеспечения точного сопоставления в научных статьях. HMM были использованы для создания и улучшения фильтров, которые помогают идентифицировать и нормализовать генетические данные с учетом ошибок и вариаций. Результаты показали, что F-мера модели составила 0.72.

"A machine learning approach for product matching and categorization"[3] - рассматривается задача нормализации и категоризации товаров, представленных в интернет-магазинах. Использовались условные случайные поля (CRF) для обработки текстов и нейронные сети для анализа изображений, что позволило извлекать атрибуты товаров из неструктурированных данных, включая текстовые описания и изображения. Применение embedding-технологий улучшило извлечение характеристик и нормализацию атрибутов. Исследование показало, что подход на основе CRF хорошо подходит для обработки структурированных данных.

"A Machine Learning Approach for Product Matching and Categorization"[4] - Исследование посвящено гибридному подходу к нормализации данных о продуктах, включающему использование правил и машинного обучения. Применение правил позволяет структурировать текстовые данные, тогда как методы машинного обучения, такие как CRF, помогают выделять и нормализовать атрибуты из описаний продуктов.

"preon: Fast and accurate entity normalization for drug names and cancer types in precision oncology"[5] - описана система preon, которая предназначена для нормализации наименований лекарств и их вариативных форм, встречающихся в медицинской документации. Подход preon включает многоуровневое сопоставление, что позволяет достичь точности 91%.

"sCooL: A System for Academic Institution Name Normalization"[6] - sCooL, применяется для нормализации наименований учебных заведений, используя комбинацию данных из Wikipedia и алгоритмов на базе Lucene. sCooL использует правила, базу маппингов и алгоритмы строкового сопоставления для точной идентификации учебных учреждений, включая акронимы и популярные сокращения. sCooL позволило достичь точности 95%.

Исследования подтверждают, что использование моделей, таких как BERT и CRF, позволяет достичь высокой точности в нормализации наименований, особенно для сложных данных с вариативными и ошибочными формами. Гибридные подходы, объединяющие правила и машинное обучение, показывают хорошие результаты при нормализации товарных данных.

* 1. **Построение датафрейма с информацией**

Для анализа ассортимента и структуры каталога товаров были получены данные из трех розничных сетей: "Мегастрой"[7], "OBI"[8] и "Сатурн"[9]. Данные были собраны методом веб-скрапинга с их официальных сайтов этих магазинов, что позволило получить актуальную информацию по ассортименту. Этот процесс включал извлечение данных из разделов сайтов, где представлены категории, подкатегории и наименования товаров.

Изначально данные каждой сети представляли собой наборы записей по различным товарным группам, разделенным на основные категории и подкатегории. После предварительного анализа были выбраны четыре ключевых столбца: "Название категории", "Название подкатегории", "Подкатегория второго уровня" и "Название товара". Эти столбцы содержали необходимую информацию для структурирования данных и последующего анализа. В процессе извлечения были проведены проверки на полноту данных, дубликаты и форматирование, и выявлено, что пропущенные значения отсутствуют.

Таблица 1 – часть датафрейма Мегастрой.

|  |  |  |  |
| --- | --- | --- | --- |
| Category Name | Subcategory Name | Second-level Subcategory Name | Product Name |
| Новогодние товары | Украшения елочные | Шары новогодние | Шары новогодние,Шары россыпью SY-1401 50мм красный |
| Новогодние товары | Украшения елочные | Шары новогодние | Набор шаров 3шт 10см синий SYQE-012124 |
| Новогодние товары | Украшения елочные | Шары новогодние | Набор шаров 40шт 4-7см фиолетовый+шампанское SYQC-012158 |
| Новогодние товары | Украшения елочные | Шары новогодние | Набор шаров 12шт 8см красный SYQB-012171 |

Таблица 2 – Часть датафрейма OBI.

|  |  |  |  |
| --- | --- | --- | --- |
| Category Name | Subcategory Name | Second-level Subcategory Name | Product Name |
| Хранение и порядок | Гардеробные системы | Гардеробные системы сетчатые | Выдвижная рама для корзин Larvij белая 30x597х425 мм |
| Хранение и порядок | Гардеробные системы | Гардеробные системы сетчатые | Вешалка-штанга Larvij 8 крючков белая 30x597х425 мм |
| Хранение и порядок | Гардеробные системы | Гардеробные системы сетчатые | Рама выдвижная для корзин Larvij L9601BL черная 597x425x20 мм |
| Хранение и порядок | Гардеробные системы | Гардеробные системы сетчатые | Гардеробные системы сетчатые,Полка для одежды Larvij белая 590х30х420 мм |

Таблица 3 – Часть датафрейма Сатурн.

|  |  |  |  |
| --- | --- | --- | --- |
| Category Name | Subcategory Name | Second-level Subcategory Name | Product Name |
| Строительные материалы | Строительные смеси | Штукатурки | "Штукатурка Knauf Rotband, 30 кг" |
| Строительные материалы | Строительные смеси | Штукатурки | "Штукатурка гипсовая ЕК TG40 белая, 30 кг" |
| Строительные материалы | Строительные смеси | Штукатурки | "Штукатурка ЕК TG-40, 30 кг" |
| Строительные материалы | Строительные смеси | Штукатурки | "Штукатурка гипсовая машинная Knauf МП-75, 30 кг" |

Таблицы 1, 2, 3 представляет собой части датафреймов "Мегастрой", "OBI", "Сатурн", полученных в ходе веб-скрапинга сайтов магазинов. Каждая строка в таблице содержит описание товара, включая основную категорию, подкатегорию, более детализированную подкатегорию второго уровня и название товара.

В каждом из созданных датафреймов присутствуют следующие переменные:

* Category Name — основная категория товара, которая указывает на тип продукции.
* Subcategory Name — подкатегория, уточняющая основное направление товара.
* Second-level Subcategory Name — дополнительная подкатегория второго уровня, более точно отражающая вид товара.
* Product Name — название товара.

Под нормализованным результатом в контексте анализа товарных данных мы будем подразумевать унифицированное и структурированное наименование товара. Это название, которое выполняет следующие функции:

* Устранены вариативности записи - разные форматы описания одного и того же товара в разных источниках "Шары стеклянные" и "Стеклянные шары".
* Удалены лишние элементы - которые не влияют на идентификацию товара, такие как ненужные символы или уточнения "SY-1401" и т.д.
* Сохранены ключевые характеристики - остается основная информация, которая делает товар идентифицируемым.

# **Глава 2. Организация и проведения анализа данных**

## **2.1. Разведочный анализ данных.**

Для каждой категории, подкатегории и уровня товаров рассчитывается количество уникальных значений. Результаты для каждого из датафреймов представлены в Таблице 4, где видно, что "Мегастрой" содержит наибольшее количество уникальных товаров, а "Сатурн" — наименьшее.

Таблица 4 – Уникальные значения каждого датафрейма.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Датафрейм | Category Name | Subcategory Name | Second-level Subcategory Name | Product Name |
| Мегастрой | 22 | 164 | 1028 | 43336 |
| Сатурн | 9 | 78 | 426 | 8090 |
| Оби | 20 | 186 | 1371 | 24924 |

На следующем этапе были выделены общие категории для различных сетей. Это позволяет понять, какие группы товаров чаще всего встречаются.

Есть одна общая категория для всех трех магазинов:

* Стройматериалы.

Общие категории для "Сатурн" и "Оби":

* Инструмент;
* Электрика;
* Хозяйственные товары.

Общие категории для "Мегастрой" и "Оби":

* Лакокрасочные материалы;
* Сантехника;
* Освещение;
* Напольные покрытия;
* Декор;
* Окна и двери;
* Сад и дача.

Общие категории для "Мегастрой" и "Сатурн":

* Новогодние товары.

Уникальные категории для "Мегастрой":

* Хранение, мебель;
* Системы безопасности;
* Баня и сауна;
* Отопление, водоснабжение и климат;
* Инструменты и крепеж;
* Текстиль и декор для дома;
* Электротовары;
* Кухни. Бытовая техника;
* Спецодежда и средства индивидуальной защиты;
* Спорт и отдых.

Уникальные категории для "Сатурн":

* Ванная комната;
* Крепеж;
* Интерьер и отделка.

Уникальные категории для "Оби":

* Столярные изделия;
* Климат и отопление;
* Кухни;
* Хранение и порядок.

Для повышения эффективности обработки данных и унификации их структуры была выполнена нормализация наименований товаров. Этот процесс направлен на устранение вариативности в названиях, что улучшает идентификацию товаров и их сопоставимость между сетями.

Таблица 5 – Часть нормализованного продукта в датафрейме.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Датафрейм | Category Name | Subcategory Name | Second-level Subcategory Name | Product Name | Normalized Name |
| Мегастрой | |  | | --- | | Лакокрасочные материалы |  |  | | --- | |  | | |  | | --- | | Защита древесины |  |  | | --- | |  | | |  | | --- | | Масла, воски для дерева |  |  | | --- | |  | | |  | | --- | | Масло для террас Tikkurila Valtti Terrace Oil EC (9л) |  |  | | --- | |  | | Масло для террас Tikkurila 9л |
| Сатурн | |  | | --- | | ЛКМ, пены, герметики |  |  | | --- | |  | | |  | | --- | | Антисептики и пропитки для дерева |  |  | | --- | |  | | |  | | --- | | Масло и воск для дерева |  |  | | --- | |  | | |  | | --- | | Масло для террас Tikkurila Valtti Terrace Oil EC (9 л) |  |  | | --- | |  | | Масло для террас Tikkurila 9л |
| Оби | Лакокрасочные материалы | Покрытия для дерева | Масла для дерева | МАСЛО ДЛЯ ТЕРРАС VALTTI TERRACE OIL EC 9 Л | Масло для террас Tikkurila 9л |

Нормализация привела к созданию новой переменной Normalized Name, где наименования товаров унифицированы.

## **2.2. Методологические основы анализа**

Для нормализации наименований товаров были выбраны методы, сочетающие использование правил обработки текста, алгоритмов машинного обучения и моделей глубокого обучения. Такой подход обеспечивает универсальность и точность нормализации, учитывая разнообразие и вариативность исходных данных.

Методы нормализации, использованные в исследовании, различаются по сложности и эффективности. В таблице ниже представлены их характеристики.

Таблица 6 - Методы нормализации данных.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Описание | Преимущества | Недостатки | Статьи | Библиотеки и функции |
| Lexicon-based | Использование заранее определенных списков названий. | Высокая полнота за счет покрытия всех возможных вариантов написания. | Низкая точность из-за большого количества ложных срабатываний. | A Deep Learning Model for the Normalization of Institution Names by Multi-Source Literature Feature Fusion: Algorithm Development[1] | NLTK (lookup dictionary), spaCy (PhraseMatcher), pandas (filtering and matching) |
| Edit Distance и Jaccard Index | Сравнение строк для поиска схожих наименований. | Простота реализации, возможность обработки строк с орфографическими ошибками. | Низкая семантическая точность, проблемы с сокращениями и формами слов. | A Deep Learning Model for the Normalization of Institution Names by Multi-Source Literature Feature Fusion: Algorithm Development[1] | fuzzywuzzy (fuzz.ratio, fuzz.partial\_ratio), Levenshtein (distance calculation), scikit-learn (Jaccard score) |
| CRF | Тегирование и нормализация атрибутов товаров и названий в описаниях продуктов. | Хорошая производительность для структурированных данных и атрибутов. | Требует много размеченных данных, сложность настройки для новых продуктов и изменений. | A machine learning approach for product matching and categorization[3], A Machine Learning Approach for Product Matching and Categorization[4] | CRFsuite, sklearn-crfsuite (CRF models), spacy (NER pipelines), pyCRFSuite |
| Deep Learning (BERT, CNN, LSTM) | Модели глубокого обучения для нормализации названий на семантическом уровне. | Высокая точность благодаря работе с контекстом и семантикой текста. | Высокая вычислительная сложность, требует большого объема данных для обучения. | A Deep Learning Model for the Normalization of Institution Names by Multi-Source Literature Feature Fusion: Algorithm Development[1] | Transformers (Hugging Face для BERT), TensorFlow, PyTorch, Keras |
| Word2Vec (Neural Embeddings) | Генерация векторных представлений слов для извлечения характеристик продуктов и названий. | Учитывает контекст и зависимость между словами, улучшает сопоставление. | Зависимость от объема данных для обучения, требуются хорошо размеченные наборы данных. | A machine learning approach for product matching and categorization[3], A Machine Learning Approach for Product Matching and Categorization[4] | Gensim (Word2Vec), fastText, spacy (vectorization) |
| Rule-based | Создание ручных правил для нормализации и сопоставления данных. | Простота реализации, высокая точность для узкоспециализированных задач. | Плохая переносимость и зависимость от человеческого фактора, требует обновления данных. | sCooL: A System for Academic Institution Name Normalization[6], A Machine Learning Approach for Product Matching and Categorization[4] | regex, pandas, Lucene |
| HMM | Статистическая модель для тегирования и нормализации (например, гены). | Высокая точность при работе с тегированными данными, особенно в контекстах с высокой неоднозначностью. | Зависимость от качества и полноты данных, сложность настройки для различных типов текстов. | Gene name identification and normalization using a model organism database[2] | hmmlearn, pomegranate |
| SVM | Модель машинного обучения для разрешения лексической многозначности имен и нормализации данных. | Хорошо работает с небольшими наборами данных и в задачах классификации. | Требует настройки вручную, чувствителен к выбору признаков и параметров модели. | preon: Fast and accurate entity normalization for drug names and cancer types in precision oncology[5] | scikit-learn (SVM: SVC, LinearSVC), LibSVM |

Для оценки эффективности нормализации будет использоваться следующая метрика:

* Частичная ручная проверка нормализованных данных для оценки соответствия установленным стандартам.

Для реализации нормализации используется язык программирования Python и специализированные библиотеки:

* pandas и numpy: для обработки табличных данных, работы с массивами и реализации вычислений;
* re: регулярные выражения для очистки и преобразования текста;
* sklearn: реализация модели CRF для последовательного анализа текстов;
* transformers: предобученные модели BERT для обработки текстов и анализа контекста;
* nltk и spacy: для токенизации и других методов предобработки текста.

## **2.3. Проведение анализа данных**

Все модели были оценены по средней точности. В таблице ниже приведены средние точности моделей, которые использовались в процессе анализа.

Таблица 7 – Средние точности моделей.

|  |  |
| --- | --- |
| Название модели | Средняя точность модели |
| ruT5 | 46% |
| rugpt | 0% |
| rembert | 20% |
| rubert tiny2 | 20% |
| multilingual\_e5 | 20% |
| multilingual use | 20% |
| labse | 20% |
| xlm roberta | 20% |
| spacy | 8% |
| tfidf | 20% |
| bert multilingual | 20% |
| distilbert multilingual | 20% |
| xlm roberta large | 20% |
| mdeberta v3 | 26% |
| bertopic | 26% |
| roberta squad | 74% |
| flan\_t5 | 0% |
| distilbert sst | 74% |

Результаты тестирования моделей нормализации отображены в таблице. Они позволяют оценить, какая из моделей показывает лучшие результаты в различных условиях.

Из таблицы видно, что модели roberta squad и distilbert sst показали наилучшие результаты, достигнув 74% средней точности. Другие модели продемонстрировали крайне низкие результаты.

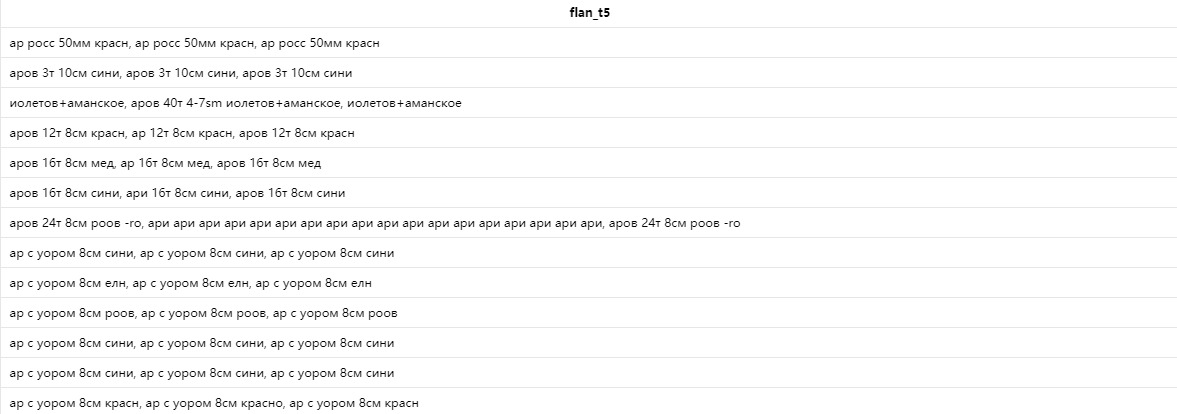


Рисунок 1 – Нормализация flan t5.



Рисунок 2 – Нормализация rugpt.

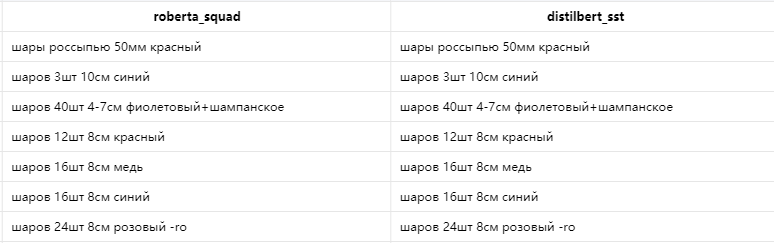


Рисунок 3 – Нормализация Roberta squad и distilbert sst.



Рисунок 4 – Нормализация Roberta squad и distilbert sst + регулярные выражения.

После применения комбинированного подхода (глубокое обучение + регулярные выражения), были получены итоговые значения точности моделей.

Таблица 8 – Средние точности итоговых моделей.

|  |  |
| --- | --- |
| Название модели | Средняя точность |
| Roberta squad + re | 90% |
| Distilbert sst + re | 90% |

В процессе нормализации наименований товаров были протестированы различные модели, включая методы глубокого обучения и подходы на основе регулярных выражений. Анализ точности показал, что комбинация моделей Roberta Squad и DistilBERT SST с использованием регулярных выражений продемонстрировала наилучшие результаты, обеспечив среднюю точность 90%.

Кроме того, тестирование выявило, что глубокие модели обладают преимуществами при обработке сложных и многозначных наименований, тогда как регулярные выражения эффективно справляются с типовыми и часто встречающимися шаблонами.

Таким образом, использование глубоких моделей в сочетании с регулярными выражениями существенно улучшило точность нормализации и показало высокую эффективность при обработке данных из различных источников.

# **Глава 3. Оформление результатов анализа данных**

## **3.1. Результаты проведенного анализа**

В данном разделе представлены результаты анализа эффективности нормализации данных по различным категориям товаров для трех компаний: Сатурн, Мегастрой и Оби. В таблицах отражены показатели эффективности нормализации для каждой товарной категории.

Таблица 9 – Оценка эффективности нормализации Сатурн.

|  |  |
| --- | --- |
| Категория | roberta\_squad |
| Ванная комната | 0.839105 |
| Инженерные системы | 0.755091 |
| Инструмент | 0.908913 |
| Интерьер и отделка | 0.737329 |
| Крепеж | 0.873737 |
| ЛКМ, пены, герметики | 0.843671 |
| Строительные материалы | 0.798315 |
| Товары для дома и сада | 0.800138 |
| Электрика | 0.593563 |

Эффективность нормализации данных по категориям "Сатурн" варьируется. Таблица позволяет проанализировать слабые и сильные стороны модели.

Таблица 10 – Оценка эффективности нормализации Мегастрой.

|  |  |
| --- | --- |
| Категория | roberta\_squad |
| Автотовары | 0.866722 |
| Баня и сауна | 0.748534 |
| Двери, окна, скобяные изделия | 0.943929 |
| Инструменты и крепеж | 0.865650 |
| Кухни. Бытовая техника | 0.769021 |
| Лакокрасочные материалы | 0.843671 |
| Напольные покрытия | 0.903573 |
| Новогодние товары | 0.769653 |
| Обои и декор потолка | 0.932714 |
| Освещение | 0.880916 |
| Отопление, водоснабжение и климат | 0.828813 |
| Плитка | 0.865802 |
| Сад и дача | 0.870431 |
| Сантехника | 0.855766 |
| Системы безопасности | 0.857407 |
| Спецодежда и средства индивидуальной защиты | 0.833604 |
| Спорт и отдых | 0.828997 |
| Стройматериалы | 0.925380 |
| Текстиль и декор для дома | 0.935905 |
| Товары для дома | 0.873045 |
| Хранение, мебель | 0.917976 |
| Электротовары | 0.892411 |

Результаты нормализации для "Мегастрой" отражают особенности структуры ассортимента, включая высокую точность в некоторых категориях.

Таблица 11 – Оценка эффективности нормализации Оби.

|  |  |
| --- | --- |
| Категория | roberta\_squad |
| Автотовары | 0.894415 |
| Декор | 0.826348 |
| Инженерные системы | 0.833027 |
| Инструмент | 0.882928 |
| Климат и отопление | 0.854880 |
| Крепеж и фурнитура | 0.906690 |
| Кухни | 0.866790 |
| Лакокрасочные материалы | 0.903835 |
| Напольные покрытия | 0.879521 |
| Новый год | 0.868960 |
| Окна и двери | 0.955126 |
| Освещение | 0.873457 |
| Плитка | 0.911483 |
| Сад и досуг | 0.877400 |
| Сантехника | 0.847078 |
| Столярные изделия | 0.954473 |
| Стройматериалы | 0.925016 |
| Хозяйственные товары | 0.871825 |
| Хранение и порядок | 0.890562 |
| Электрика | 0.890437 |

Для сети "OBI" были проанализированы различные категории товаров. Итоги нормализации демонстрируют области, требующие доработки.

Из таблиц видно, что компании Сатурн, Мегастрой и Оби показывают различные уровни эффективности нормализации данных в разных категориях. В целом, наиболее высокие значения наблюдаются в таких категориях, как инструмент и лакокрасочные материалы, где эффективность нормализации близка к 0.9. Некоторые категории демонстрируют низкие значения, что может свидетельствовать о проблемах в нормализации. Например, в Сатурне категория "Электрика" имеет самую низкую эффективность - 0.593563, в Мегастрое - "Новогодние товары" и "Кухни, бытовая техника" - 0.769, а в Оби - "Сад и досуг" - 0.877400.

Все материалы и код исследования доступны на GitHub[10].

## **3.2. Формирование практических предложений**

На основании проведенного анализа эффективности нормализации данных были выявлены как успешные области применения текущих моделей, так и проблемы, возникающие при работе с отдельными категориями товаров. Анализ показал, что использование современных моделей глубокого обучения в сочетании с подходами на основе регулярных выражений позволяет достичь высокой точности нормализации.

Проведенный анализ опирался на доступные модели, которые демонстрируют хорошие результаты в задачах нормализации данных. Однако, учитывая специфику задач нормализации и разнообразие товарных категорий, можно предложить следующий шаг для улучшения анализа:

* Для наиболее сложных категорий с низкой эффективностью нормализации следует разработать собственную модель.

Для минимизации проблем нормализации данных важно правильно организовать каталог товаров.

* Установить стандарты для заполнения товарных описаний.
* Включить поддержку классификации и нормализации на этапе добавления новых товаров в каталог.

Таким образом, для повышения эффективности нормализации необходимо не только совершенствовать алгоритмы, но и внедрять системные подходы к управлению данными, включая стандартизацию и автоматизацию процессов.

# **Заключение**

Проведенный анализ показал, что комбинация современных алгоритмов глубокого обучения, таких как Roberta Squad и DistilBERT SST, с методами на основе правил позволяет достичь высокой точности нормализации 90%. Однако для некоторых категорий товаров требуется дополнительная настройка моделей.

Практическая значимость работы заключается в применении разработанных решений для повышения качества данных в электронных каталогах.

Для улучшения точности нормализации в сложных категориях товаров следует разрабатывать специализированные модели, которые будут учитывать особенности каждой категории.

Важно внедрить стандарты описания товаров на этапе ввода данных, чтобы минимизировать ошибки и вариативность.

# **Список литературы**

1. Chen, Y., Li, X., Li, A., Li, Y., Yang, X., Lin, Z., Yu, S., Tang, X.  
   A Deep Learning Model for the Normalization of Institution Names by Multi-Source Literature Feature Fusion: Algorithm Development [Текст] / Y. Chen, X. Li, A. Li и др. // Journal of Computational Science. – 2023. – Vol. 45. – DOI: https://doi.org/10.2196/47434.
2. Ermshaus, A., Piechotta, M., Ruter, G., Keilholz, U., Leser, U.  
   Preon: Fast and Accurate Entity Normalization for Drug Names and Cancer Types in Precision Oncology [Текст] / A. Ermshaus, M. Piechotta, G. Ruter и др. // Bioinformatics. – 2023. – Vol. 38, No. 5. – DOI: https://doi.org/10.1093/bioinformatics/btae085.
3. Jacob, F., Javed, F., Zhao, M., Mcnair, M.  
   sCooL: A System for Academic Institution Name Normalization [Текст] / F. Jacob, F. Javed, M. Zhao, M. Mcnair // ACM Transactions on Information Systems. – 2023. – Vol. 41, No. 2. – DOI: https://doi.org/ 10.1109/CTS.2014.6867547.
4. Morgan, A. A., Hirschman, L., Colosimo, M., Yeh, A. S., Colombe, J. B.  
   Gene Name Identification and Normalization Using a Model Organism Database [Текст] / A. A. Morgan, L. Hirschman, M. Colosimo и др. // Bioinformatics. – 2023. – Vol. 29, No. 13. – DOI: https://doi.org/10.1016/j.jbi.2004.08.010.
5. Ristoski, P., Petrovski, P., Mika, P., Paulheim, H.  
   A Machine Learning Approach for Product Matching and Categorization: Use Case: Enriching Product Ads with Semantic Structured Data [Текст] / P. Ristoski, P. Petrovski, P. Mika, H. Paulheim // Semantic Web Journal. – 2023. – Vol. 11, No. 4. – DOI: https://doi.org/10.3233/SW-180300.
6. Zhang, X., Wei, Q., Zheng, B., Liu, J., Zhang, P.  
   FrameSum: Leveraging Framing Theory and Deep Learning for Enhanced News Text Summarization [Текст] / X. Zhang, Q. Wei, B. Zheng, J. Liu, P. Zhang // Journal of Information Science. – 2023. – Vol. 52, No. 3. – DOI: https://doi.org/10.3390/app14177548
7. Мегастрой [Электронный ресурс] // Официальный сайт компании «Мегастрой». – URL: https://megastroy.com/ (дата обращения: 12.10.2024).
8. OBI [Электронный ресурс] // Официальный сайт сети гипермаркетов OBI. – URL: https://obi.ru/ (дата обращения: 10.10.2024).
9. Сатурн [Электронный ресурс] // Официальный сайт компании «Сатурн». – URL: https://nnv.saturn.net/ (дата обращения: 11.10.2024).
10. GitHub [Электронный ресурс] // Репозиторий с материалами и кодом исследования. – URL: https://github.com/LewyWho/Normalization-coursework (дата обращения: 12.01.2025)

# **Приложение**

Код для парсинга товаров Мегастрой

import requests

from bs4 import BeautifulSoup

import csv

# URL главной страницы каталога

url = 'https://megastroy.com/catalog'

base\_url = 'https://megastroy.com'

def get\_html(url):

    try:

        response = requests.get(url)

        response.raise\_for\_status()

        return response.text

    except requests.RequestException as e:

        print(f"Ошибка при запросе URL {url}: {e}")

        return None

# Функция для парсинга страницы товаров с проверкой на пагинацию

def parse\_products\_page(page\_url, category\_name, subcategory\_name, subcategory\_href, second\_level\_subcategory\_name, second\_level\_subcategory\_href, writer):

    while page\_url:

        product\_response = requests.get(page\_url)

        if product\_response.status\_code == 200:

            product\_soup = BeautifulSoup(product\_response.content, 'html.parser')

            # Извлекаем товары на странице

            products = product\_soup.find\_all('div', class\_='products-list\_\_content-title')

            if products:

                for product in products:

                    product\_name = product.find('a').get('title').strip()

                    print(f'--- {product\_name}')

                    writer.writerow([category\_name, subcategory\_name, subcategory\_href, second\_level\_subcategory\_name, second\_level\_subcategory\_href, product\_name])

            else:

                print(f'На странице {page\_url} не найдено товаров.')

                writer.writerow([category\_name, subcategory\_name, subcategory\_href, second\_level\_subcategory\_name, second\_level\_subcategory\_href, "No products found"])

            # Проверяем, есть ли следующая страница

            next\_page = product\_soup.find('a', class\_='pagination\_\_item pagination\_\_arrow', href=True)

            if next\_page:

                page\_url = base\_url + next\_page['href']

                print(f'Переход на следующую страницу: {page\_url}')

            else:

                print(f'Нет следующей страницы для {page\_url}.')

                page\_url = None

        else:

            print(f"Ошибка при запросе страницы товаров {page\_url}, статус код: {product\_response.status\_code}")

            page\_url = None

# Основная логика парсинга каталога

def parse\_catalog():

    html = get\_html(url)

    if not html:

        print("Не удалось загрузить главную страницу каталога.")

        return

    soup = BeautifulSoup(html, 'html.parser')

    categories = soup.find\_all('div', class\_='catalog-list\_\_item-four')

    if not categories:

        print("Категории не найдены на главной странице.")

        return

    csv\_file\_path = "megastroy.csv"

    with open(csv\_file\_path, mode="w", newline="", encoding="utf-8") as file:

        writer = csv.writer(file)

        writer.writerow(["Category Name", "Subcategory Name", "Subcategory URL", "Second-level Subcategory Name", "Second-level Subcategory URL", "Product Name"])

        # Проходим по каждой категории

        for category in categories:

            try:

                category\_name = category.find('a', class\_='catalog-list\_\_item-title').text.strip()

                category\_href = base\_url + category.find('a', class\_='catalog-list\_\_item-title')['href']

            except AttributeError:

                print("Ошибка извлечения названия или ссылки категории.")

                continue

            print(f'{category\_name} - {category\_href}')

            sub\_response = requests.get(category\_href)

            if sub\_response.status\_code != 200:

                print(f"Ошибка при запросе подкатегорий для {category\_name}, статус код: {sub\_response.status\_code}")

                continue

            sub\_soup = BeautifulSoup(sub\_response.content, 'html.parser')

            subcategories = sub\_soup.find\_all('a', class\_='categories-list\_\_content-title')

            if not subcategories:

                print(f"Подкатегории не найдены для категории {category\_name}.")

                writer.writerow([category\_name, "No subcategory", category\_href, "", "", ""])

                continue

            for subcategory in subcategories:

                try:

                    subcategory\_name = subcategory.text.strip()

                    subcategory\_href = base\_url + subcategory['href']

                except AttributeError:

                    print("Ошибка извлечения подкатегории.")

                    continue

                print(f'- {subcategory\_name} - {subcategory\_href}')

                second\_sub\_response = requests.get(subcategory\_href)

                if second\_sub\_response.status\_code != 200:

                    print(f"Ошибка при запросе товаров для {subcategory\_name}, статус код: {second\_sub\_response.status\_code}")

                    continue

                second\_sub\_soup = BeautifulSoup(second\_sub\_response.content, 'html.parser')

                second\_level\_subcategories = second\_sub\_soup.find\_all('a', class\_='categories-list\_\_content-title')

                if not second\_level\_subcategories:

                    print(f"В подкатегории {subcategory\_name} не найдено подкатегорий второго уровня.")

                    writer.writerow([category\_name, subcategory\_name, subcategory\_href, "No second-level subcategory", "", ""])

                    continue

                for second\_level\_subcategory in second\_level\_subcategories:

                    try:

                        second\_level\_subcategory\_name = second\_level\_subcategory.text.strip()

                        second\_level\_subcategory\_href = base\_url + second\_level\_subcategory['href']

                    except AttributeError:

                        print("Ошибка извлечения подкатегории второго уровня.")

                        continue

                    print(f'-- {second\_level\_subcategory\_name} - {second\_level\_subcategory\_href}')

                    # Переход на первую страницу товаров и парсинг с учётом пагинации

                    parse\_products\_page(second\_level\_subcategory\_href, category\_name, subcategory\_name, subcategory\_href, second\_level\_subcategory\_name, second\_level\_subcategory\_href, writer)

if \_\_name\_\_ == "\_\_main\_\_":

    parse\_catalog()

    print("Парсинг завершён")

Код для парсинга Оби

import requests

from bs4 import BeautifulSoup

import csv

url = 'https://obi.ru/catalog'

base\_url = 'https://obi.ru'

response = requests.get(url)

if response.status\_code == 200:

    soup = BeautifulSoup(response.content, 'html.parser')

    categories = soup.find\_all('a', class\_='kn7A0')

    csv\_file\_path = "categories\_with\_subcategories\_and\_products.csv"

    with open(csv\_file\_path, mode="w", newline="", encoding="utf-8") as file:

        writer = csv.writer(file)

        writer.writerow(["Category Name", "Subcategory Name", "Subcategory URL", "Second-level Subcategory Name", "Second-level Subcategory URL", "Product Name"])  # Заголовки

        for category in categories:

            category\_name = category.find('span', class\_='\_17tb-').text

            category\_href = base\_url + category['href']

            sub\_response = requests.get(category\_href)

            if sub\_response.status\_code == 200:

                sub\_soup = BeautifulSoup(sub\_response.content, 'html.parser')

                subcategories = sub\_soup.find\_all('a', class\_='kn7A0')

                if subcategories:

                    for subcategory in subcategories:

                        subcategory\_name = subcategory.find('span', class\_='\_17tb-').text

                        subcategory\_href = base\_url + subcategory['href']

                        writer.writerow([category\_name, subcategory\_name, subcategory\_href, "", "", ""])

                        second\_sub\_response = requests.get(subcategory\_href)

                        if second\_sub\_response.status\_code == 200:

                            second\_sub\_soup = BeautifulSoup(second\_sub\_response.content, 'html.parser')

                            second\_level\_subcategories = second\_sub\_soup.find\_all('a', class\_='kn7A0')

                            if second\_level\_subcategories:

                                for second\_subcategory in second\_level\_subcategories:

                                    second\_subcategory\_name = second\_subcategory.find('span', class\_='\_17tb-').text

                                    second\_subcategory\_href = base\_url + second\_subcategory['href']

                                    writer.writerow([category\_name, subcategory\_name, subcategory\_href, second\_subcategory\_name, second\_subcategory\_href, ""])

                                    product\_response = requests.get(second\_subcategory\_href)

                                    if product\_response.status\_code == 200:

                                        product\_soup = BeautifulSoup(product\_response.content, 'html.parser')

                                        products = product\_soup.find\_all('p', class\_='\_1UlGi')

                                        if products:

                                            for product in products:

                                                product\_name = product.text.strip()

                                                writer.writerow([category\_name, subcategory\_name, subcategory\_href, second\_subcategory\_name, second\_subcategory\_href, product\_name])

                                        else:

                                            writer.writerow([category\_name, subcategory\_name, subcategory\_href, second\_subcategory\_name, second\_subcategory\_href, "No products found"])

                                    else:

                                        print(f"Ошибка при запросе товаров для {second\_subcategory\_name}, статус код: {product\_response.status\_code}")

                            else:

                                writer.writerow([category\_name, subcategory\_name, subcategory\_href, "No second-level subcategory", "", ""])

                else:

                    writer.writerow([category\_name, "No subcategory", category\_href, "", "", ""])

            else:

                print(f"Ошибка при запросе подкатегорий для {category\_name}, статус код: {sub\_response.status\_code}")

else:

    print(f"Ошибка при запросе страницы, статус код: {response.status\_code}")

print("Complete")

Код для парсинга Сатурна

import requests

from bs4 import BeautifulSoup

import csv

base\_url = "https://nnv.saturn.net/catalog"

main\_url = 'https://nnv.saturn.net'

def get\_html(url):

    response = requests.get(url)

    response.raise\_for\_status()

    return response.text

# Функция для парсинга продуктов на странице

def parse\_products\_on\_page(soup, writer, category\_name, subcategory\_name, subcategory\_href, second\_subcategory\_name, second\_subcategory\_href):

    products = soup.find\_all('li', class\_='catalog\_Level2\_\_goods\_list\_\_item')

    if products:

        for product in products:

            product\_name = product.find('a', class\_='goods\_card\_text swiper-no-swiping').text.strip()

            print(f'--- {product\_name}')

            writer.writerow([category\_name, subcategory\_name, subcategory\_href, second\_subcategory\_name, second\_subcategory\_href, product\_name])  # Запись продукта

    else:

        print(f'--- No products found in {second\_subcategory\_name}')

        writer.writerow([category\_name, subcategory\_name, subcategory\_href, second\_subcategory\_name, second\_subcategory\_href, "No products found"])

# Функция для получения всех страниц с продуктами

def parse\_pagination(soup, base\_href):

    pagination = soup.select('.pagination\_\_item a.pagination\_\_link')

    page\_links = []

    if pagination:

        for page in pagination:

            page\_number = page.get('data-page')

            if page\_number:

                page\_href = base\_href + "?page=" + page\_number

                page\_links.append(page\_href)

    return page\_links

# Функция для парсинга главной категории

def parse\_main\_category():

    html = get\_html(base\_url)

    soup = BeautifulSoup(html, 'html.parser')

    categories = soup.select('.catalog\_\_level1\_\_nav-list\_\_item')

    # Создание CSV-файла

    with open('saturn.csv', mode='w', newline='', encoding='utf-8') as csv\_file:

        writer = csv.writer(csv\_file)

        writer.writerow(["Main Category", "Subcategory", "Subcategory URL", "Second Level Subcategory Name", "Second Level Subcategory URL", "Product Name"])  # Заголовки

        for category in categories:

            category\_name = category.find('span', class\_='catalog\_\_level1\_\_nav-list\_\_item\_\_title').text.strip()

            category\_href = main\_url + category.find('a')['href']

            print(f'{category\_name} - {category\_href}')

            sub\_response = requests.get(category\_href)

            if sub\_response.status\_code == 200:

                sub\_soup = BeautifulSoup(sub\_response.content, 'html.parser')

                subcategories = sub\_soup.find\_all('a', class\_='catalog\_\_level2\_\_nav-list\_\_item')

                if subcategories:

                    for subcategory in subcategories:

                        subcategory\_name = subcategory.find('span', class\_='catalog\_\_level2\_\_nav-list\_\_item\_\_title').text.strip()

                        subcategory\_href = main\_url + subcategory['href']

                        print(f'- {subcategory\_name} - {subcategory\_href}')

                        writer.writerow([category\_name, subcategory\_name, subcategory\_href, "", "", ""])  # Запись подкатегории

                        second\_sub\_response = requests.get(subcategory\_href)

                        if second\_sub\_response.status\_code == 200:

                            second\_sub\_soup = BeautifulSoup(second\_sub\_response.content, 'html.parser')

                            second\_level\_subcategories = second\_sub\_soup.find\_all('a', class\_='catalog\_\_level2\_\_nav-list\_\_item')

                            if second\_level\_subcategories:

                                for second\_subcategory in second\_level\_subcategories:

                                    second\_subcategory\_name = second\_subcategory.find('span', class\_='catalog\_\_level2\_\_nav-list\_\_item\_\_title').text.strip()

                                    second\_subcategory\_href = main\_url + second\_subcategory['href']

                                    print(f'-- {second\_subcategory\_name} - {second\_subcategory\_href}')

                                    writer.writerow([category\_name, subcategory\_name, subcategory\_href, second\_subcategory\_name, second\_subcategory\_href, ""])  # Запись второго уровня подкатегории

                                    product\_response = requests.get(second\_subcategory\_href)

                                    if product\_response.status\_code == 200:

                                        product\_soup = BeautifulSoup(product\_response.content, 'html.parser')

                                        parse\_products\_on\_page(product\_soup, writer, category\_name, subcategory\_name, subcategory\_href, second\_subcategory\_name, second\_subcategory\_href)

                                        pagination\_links = parse\_pagination(product\_soup, second\_subcategory\_href)

                                        if pagination\_links:

                                            for page\_link in pagination\_links:

                                                page\_response = requests.get(page\_link)

                                                if page\_response.status\_code == 200:

                                                    page\_soup = BeautifulSoup(page\_response.content, 'html.parser')

                                                    parse\_products\_on\_page(page\_soup, writer, category\_name, subcategory\_name, subcategory\_href, second\_subcategory\_name, second\_subcategory\_href)

                                                else:

                                                    print(f"Ошибка при запросе страницы {page\_link}, статус код: {page\_response.status\_code}")

                                        else:

                                            print(f'Нет дополнительных страниц для {second\_subcategory\_name}')

                                    else:

                                        print(f"Ошибка при запросе товаров для {second\_subcategory\_name}, статус код: {product\_response.status\_code}")

                            else:

                                print(f'-- No second-level subcategories in {subcategory\_name}')

                                writer.writerow([category\_name, subcategory\_name, subcategory\_href, "No second-level subcategory", "", ""])

                        else:

                            print(f"Ошибка при запросе подкатегорий для {subcategory\_name}, статус код: {second\_sub\_response.status\_code}")

                else:

                    print(f'- No subcategories found in {category\_name}')

                    writer.writerow([category\_name, "No subcategory", category\_href, "", "", ""])

            else:

                print(f"Ошибка при запросе подкатегорий для {category\_name}, статус код: {sub\_response.status\_code}")

if \_\_name\_\_ == "\_\_main\_\_":

    parse\_main\_category()

    print("Парсинг завершён")

Код для нахождения уникальных категорий

import pandas as pd

df1 = pd.read\_csv('megastroy.csv')

df2 = pd.read\_csv('saturn.csv')

df3 = pd.read\_csv('obi.csv')

columns = ['Category Name']

def analyze\_categories(df1, df2, df3, columns):

    cat1 = set(df1[columns].apply(tuple, axis=1))

    cat2 = set(df2[columns].apply(tuple, axis=1))

    cat3 = set(df3[columns].apply(tuple, axis=1))

    common\_all = cat1 & cat2 & cat3

    common\_1\_2 = cat1 & cat2 - common\_all

    common\_2\_3 = cat2 & cat3 - common\_all

    common\_1\_3 = cat1 & cat3 - common\_all

    unique\_1 = cat1 - (cat2 | cat3)

    unique\_2 = cat2 - (cat1 | cat3)

    unique\_3 = cat3 - (cat1 | cat2)

    return {

        "Общие для всех трех": common\_all,

        "Общие для 1 и 2": common\_1\_2,

        "Общие для 2 и 3": common\_2\_3,

        "Общие для 1 и 3": common\_1\_3,

        "Уникальные для 1": unique\_1,

        "Уникальные для 2": unique\_2,

        "Уникальные для 3": unique\_3

    }

category\_analysis = analyze\_categories(df1, df2, df3, columns)

output\_data = []

for key, value in category\_analysis.items():

    for item in value:

        output\_data.append({

            'Category Analysis': key,

            'Category Name': item[0]

        })

output\_df = pd.DataFrame(output\_data)

output\_path = 'category\_analysis\_only\_first\_cat1.xlsx'

output\_df.to\_excel(output\_path, index=False)

Код для нахождения похожих товаров

from rapidfuzz import fuzz

import pandas as pd

import time

from tqdm import tqdm

def trigram\_similarity(a, b):

    return fuzz.ratio(a, b) / 100

def find\_similar\_products(dfs, column, threshold=0.8, max\_results="all"):

    similar\_products = []

    start\_time = time.time()

    df\_names = {

        "Megastroy": dfs[0],

        "Saturn": dfs[1],

        "Obi": dfs[2]

    }

    total\_products = len(dfs[0])

    print(f"\nОбработка датафреймов")

    test\_sample = min(250, total\_products)

    test\_start = time.time()

    for i in range(test\_sample):

        product1 = dfs[0][column].iloc[i]

        \_ = [p for p in dfs[1][column] if trigram\_similarity(product1.lower(), p.lower()) >= threshold]

        \_ = [p for p in dfs[2][column] if trigram\_similarity(product1.lower(), p.lower()) >= threshold]

    test\_time = time.time() - test\_start

    estimated\_time = (test\_time / test\_sample) \* total\_products

    if estimated\_time / 60 > 60:

        print(f"Ориентировочное время выполнения: {estimated\_time / 3600:.1f} часов")

    else:

        print(f"Ориентировочное время выполнения: {estimated\_time / 60:.1f} минут")

    print("-" \* 80)

    for product1 in tqdm(dfs[0][column],

                        total=total\_products,

                        desc="Поиск похожих товаров",

                        colour="green",

                        ncols=100):

        similar\_in\_df2 = [

            product2 for product2 in dfs[1][column]

            if trigram\_similarity(product1.lower(), product2.lower()) >= threshold

        ]

        similar\_in\_df3 = [

            product3 for product3 in dfs[2][column]

            if trigram\_similarity(product1.lower(), product3.lower()) >= threshold

        ]

        if similar\_in\_df2 and similar\_in\_df3:

            similar\_products.append({

                'DF1 (Megastroy)': product1,

                'DF2 (Saturn)': similar\_in\_df2[0],

                'DF3 (Obi)': similar\_in\_df3[0]

            })

        if isinstance(max\_results, int) and len(similar\_products) >= max\_results:

            break

    end\_time = time.time()

    print(f"\nОбщее время обработки: {end\_time - start\_time:.2f} секунд")

    return similar\_products

def save\_to\_excel(results, filename):

    if results:

        df = pd.DataFrame(results)

        df.to\_excel(filename, index=False)

        print(f"Результаты сохранены в файл {filename}")

    else:

        print("Нет данных для сохранения.")

try:

    df1 = pd.read\_csv('megastroy.csv')

    df2 = pd.read\_csv('saturn.csv')

    df3 = pd.read\_csv('obi.csv')

except FileNotFoundError as e:

    print(f"Ошибка: {e}")

    exit()

column = 'Product Name'

threshold = 0.8

max\_results = "all"

print("\n\nНачало поиска похожих товаров")

results = find\_similar\_products([df1, df2, df3], column, threshold, max\_results)

if results:

    print("Найдены похожие товары:")

    save\_to\_excel(results, 'similfar\_products.xlsx')

else:

    print("Похожие товары не найдены.")

Код для нормализации товаров (Все модели)

import pandas as pd

import numpy as np

from transformers import pipeline

from tqdm import tqdm

from sentence\_transformers import SentenceTransformer, util

import torch

from fuzzywuzzy import process, fuzz

import re

import spacy

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from sklearn.cluster import KMeans

import nltk

import os

from datetime import datetime

os.environ['TF\_ENABLE\_ONEDNN\_OPTS'] = '0'

nltk.download('punkt')

def load\_data(limit='all'):

    megastroy\_df = pd.read\_csv('megastroy.csv', encoding='utf-8-sig')

    obi\_df = pd.read\_csv('obi.csv', encoding='utf-8-sig')

    saturn\_df = pd.read\_csv('saturn.csv', encoding='utf-8-sig')

    columns = ['Category Name', 'Subcategory Name', 'Second-level Subcategory Name', 'Product Name']

    merged\_df = pd.concat([megastroy\_df[columns], obi\_df[columns], saturn\_df[columns]], ignore\_index=True)

    if limit != 'all':

        merged\_df = merged\_df.head(int(limit))

    return merged\_df

def normalize\_text(text):

    text = str(text).lower()

    text = re.sub(r'\b[A-Za-z]{2,6}[-]?\d{3,8}\b', '', text)

    text = re.sub(r'\b[A-Za-z]{2,6}[-]?\d{2,8}[-]?\d{2,8}\b', '', text)

    text = re.sub(r'\b[A-Za-z]+\d+[A-Za-z]\*[-]?\d\*\b', '', text)

    text = re.sub(r'["\.,;:!]', '', text)

    text = re.sub(r'\b(арт|артикул|упак|модель|для)\b', '', text)

    text = re.sub(r'\s+', ' ', text).strip()

    return text

def get\_normalization\_models():

    models = {

        'ruT5': pipeline(

            "text2text-generation",

            model="ai-forever/ruT5-base",

            max\_length=128,

            device=0 if torch.cuda.is\_available() else -1,

        ),

        'rugpt': pipeline(

            "text-generation",

            model="ai-forever/rugpt3small\_based\_on\_gpt2",

            device=0 if torch.cuda.is\_available() else -1,

            max\_new\_tokens=50,

            truncation=True

        ),

        'rubert\_tiny2': pipeline(

            "feature-extraction",

            model="cointegrated/rubert-tiny2",

            device=0 if torch.cuda.is\_available() else -1

        ),

        'multilingual\_e5': SentenceTransformer('intfloat/multilingual-e5-large'),

        'multilingual\_use': SentenceTransformer('distiluse-base-multilingual-cased-v1'),

        'labse': SentenceTransformer('sentence-transformers/LaBSE'),

        'xlm\_roberta': pipeline(

            "feature-extraction",

            model="xlm-roberta-base",

            device=0 if torch.cuda.is\_available() else -1

        ),

        'rembert': pipeline(

            "feature-extraction",

            model="google/rembert",

            device=0 if torch.cuda.is\_available() else -1

        ),

        'spacy': spacy.load('ru\_core\_news\_sm'),

        'tfidf': TfidfVectorizer(

            lowercase=True,

            ngram\_range=(1, 3),

            max\_features=15000

        ),

        'bert\_multilingual': pipeline(

            "feature-extraction",

            model="bert-base-multilingual-cased",

            device=0 if torch.cuda.is\_available() else -1,

            ignore\_mismatched\_sizes=True

        ),

        'distilbert\_multilingual': pipeline(

            "feature-extraction",

            model="distilbert-base-multilingual-cased",

            device=0 if torch.cuda.is\_available() else -1,

            ignore\_mismatched\_sizes=True

        ),

        'xlm\_roberta\_large': pipeline(

            "feature-extraction",

            model="xlm-roberta-large",

            device=0 if torch.cuda.is\_available() else -1

        ),

        'mdeberta\_v3': pipeline(

            "text-classification",

            model="microsoft/mdeberta-v3-base",

            device=0 if torch.cuda.is\_available() else -1

        ),

        'bertopic': SentenceTransformer('sentence-transformers/bert-base-nli-mean-tokens'),

        'roberta\_squad': pipeline(

            "question-answering",

            model="deepset/roberta-base-squad2",

            device=0 if torch.cuda.is\_available() else -1

        ),

        'flan\_t5': pipeline(

            "text2text-generation",

            model="google/flan-t5-large",

            device=0 if torch.cuda.is\_available() else -1,

            max\_length=128

        ),

        'distilbert\_sst': pipeline(

            "text-classification",

            model="distilbert/distilbert-base-uncased-finetuned-sst-2-english",

            device=0 if torch.cuda.is\_available() else -1

        )

    }

    return models

def normalize\_with\_models(text, models, method, reference\_data=None):

    try:

        original\_text = text

        text = normalize\_text(text)

        def clean\_result(result):

            result = normalize\_text(result)

            result = re.sub(r'\b[A-Za-z]+-?\d+(?:-?\d+)\*\b', '', result)

            return result.strip()

        if method in ['rubert', 'rubert\_tiny2', 'rubert\_conversational',

                     'ruelectra', 'xlm\_roberta', 'rembert', 'bert\_multilingual',

                     'distilbert\_multilingual', 'xlm\_roberta\_large', 'tfidf',

                     'multilingual\_e5', 'multilingual\_use', 'labse']:

            doc = models['spacy'](original\_text.lower())

            parts = []

            nouns = [token.text for token in doc if token.pos\_ == 'NOUN']

            if len(nouns) >= 2:

                parts.append(' '.join(nouns[:2]))

            elif nouns:

                parts.append(nouns[0])

            quantities = re.findall(r'\d+\s\*(?:шт|компл|набор|уп)', original\_text.lower())

            if quantities:

                parts.append(quantities[0])

            sizes = re.findall(r'\d+(?:[.,]\d+)?(?:-\d+(?:[.,]\d+)?)?\s\*(?:см|мм|м|мл|л|г|кг)', original\_text.lower())

            if sizes:

                parts.append(sizes[0])

            colors = ['медь', 'красный', 'синий', 'зеленый', 'золото', 'серебро',

                     'белый', 'черный', 'желтый', 'коричневый', 'бежевый',

                     'фиолетовый', 'шампанское']

            found\_colors = []

            for color in colors:

                if color in original\_text.lower():

                    found\_colors.append(color)

            if found\_colors:

                if len(found\_colors) == 1:

                    parts.append(f"цвет - {found\_colors[0]}")

                else:

                    parts.append(f"цвет - {'+'.join(found\_colors)}")

            return ', '.join(parts)

        elif method == 'mdeberta\_v3' or method == 'bertopic':

            try:

                doc = models['spacy'](original\_text.lower())

                parts = []

                nouns = [token.text for token in doc if token.pos\_ == 'NOUN']

                if len(nouns) >= 2:

                    parts.append(' '.join(nouns[:2]))

                elif nouns:

                    parts.append(nouns[0])

                quantities = re.findall(r'\d+\s\*(?:шт|компл|набор|уп)', original\_text.lower())

                if quantities:

                    parts.append(quantities[0])

                sizes = re.findall(r'\d+(?:[.,]\d+)?(?:-\d+(?:[.,]\d+)?)?\s\*(?:см|мм|м|мл|л|г|кг)', original\_text.lower())

                if sizes:

                    parts.append(sizes[0])

                colors = ['медь', 'красный', 'синий', 'зеленый', 'золото', 'серебро',

                         'белый', 'черный', 'желтый', 'коричневый', 'бежевый',

                         'фиолетовый', 'шампанское']

                found\_colors = []

                for color in colors:

                    if color in original\_text.lower():

                        found\_colors.append(color)

                if found\_colors:

                    if len(found\_colors) == 1:

                        parts.append(f"цвет - {found\_colors[0]}")

                    else:

                        parts.append(f"цвет - {'+'.join(found\_colors)}")

                result = ', '.join(parts)

                return clean\_result(result) if result else text

            except Exception as e:

                print(f"Ошибка {method}: {e}")

                return clean\_result(text)

        elif method in ['ruT5', 'rut5\_paraphrase', 'mt5\_multilingual']:

            prompts = [

                f"Опиши товар кратко с характеристиками: {text}",

                f"Укажи название, количество, размер и цвет товара: {text}",

                f"Преобразуй в формат: название, характеристики, цвет: {text}"

            ]

            results = []

            for prompt in prompts:

                try:

                    result = models[method](prompt, max\_length=128,

                                         num\_return\_sequences=1)[0]['generated\_text']

                    result = clean\_text(result)

                    results.append(result.strip())

                except Exception as e:

                    continue

            valid\_results = [r for r in results if is\_valid\_result(r)]

            if valid\_results:

                best\_result = max(valid\_results, key=len)

                return format\_product\_description(best\_result)

            return text

        elif method in ['rugpt']:

            prompt = f"Опиши товар с характеристиками: {text}"

            result = models[method](prompt)[0]['generated\_text']

            result = clean\_text(result)

            return format\_product\_description(result)

        elif method in ['rubert', 'rubert\_tiny2', 'rubert\_conversational',

                       'ruelectra', 'xlm\_roberta', 'rembert', 'bert\_multilingual',

                       'distilbert\_multilingual', 'xlm\_roberta\_large']:

            embeddings = models[method](text)

            if isinstance(embeddings, list):

                embeddings = np.array(embeddings[0][0])

            if reference\_data is not None:

                ref\_embeddings\_list = []

                for ref\_text in reference\_data:

                    ref\_emb = models[method](ref\_text)

                    if isinstance(ref\_emb, list):

                        ref\_emb = np.array(ref\_emb[0][0])

                    ref\_embeddings\_list.append(ref\_emb)

                ref\_embeddings = np.stack(ref\_embeddings\_list)

                similarities = cosine\_similarity([embeddings], ref\_embeddings)

                most\_similar\_idx = similarities.argmax()

                return reference\_data[most\_similar\_idx]

            return text

        elif method in ['multilingual\_e5', 'multilingual\_use', 'labse']:

            embeddings = models[method].encode(text, convert\_to\_tensor=True)

            if reference\_data is not None:

                ref\_embeddings = models[method].encode(reference\_data, convert\_to\_tensor=True)

                similarities = util.pytorch\_cos\_sim(embeddings, ref\_embeddings)

                most\_similar\_idx = similarities.argmax().item()

                return reference\_data[most\_similar\_idx]

            return text

        elif method == 'spacy':

            doc = models[method](text)

            return extract\_product\_info(doc, text)

        elif method == 'tfidf':

            if reference\_data is not None:

                tfidf\_matrix = models[method].fit\_transform(reference\_data)

                text\_tfidf = models[method].transform([text])

                similarities = cosine\_similarity(text\_tfidf, tfidf\_matrix)

                most\_similar\_idx = similarities.argmax()

                return reference\_data[most\_similar\_idx]

            return text

        elif method == 'mdeberta\_v3':

            try:

                result = models[method](text)

                parts = []

                doc = models['spacy'](original\_text.lower())

                nouns = [token.text for token in doc if token.pos\_ == 'NOUN']

                if len(nouns) >= 2:

                    parts.append(' '.join(nouns[:2]))

                elif nouns:

                    parts.append(nouns[0])

                base\_info = extract\_product\_info(doc, text)

                if base\_info:

                    base\_parts = [p for p in base\_info.split(', ')

                                if not any(noun in p.lower() for noun in nouns)]

                    parts.extend(base\_parts)

                return ', '.join(parts) if parts else text

            except Exception as e:

                print(f"Ошибка mdeberta\_v3: {e}")

                return text

        elif method == 'bertopic':

            try:

                doc = models['spacy'](original\_text.lower())

                nouns = [token.text for token in doc if token.pos\_ == 'NOUN']

                parts = []

                if len(nouns) >= 2:

                    parts.append(' '.join(nouns[:2]))

                elif nouns:

                    parts.append(nouns[0])

                base\_info = extract\_product\_info(doc, text)

                if base\_info:

                    base\_parts = [p for p in base\_info.split(', ')

                                if not any(noun in p.lower() for noun in nouns)]

                    parts.extend(base\_parts)

                return ', '.join(parts) if parts else text

            except Exception as e:

                print(f"Ошибка bertopic: {e}")

                return text

        elif method == 'roberta\_squad':

            try:

                questions = [

                    "Что это за товар?",

                    "Какие характеристики у товара?",

                    "Какой цвет у товара?",

                    "Какой размер у товара?"

                ]

                answers = []

                for question in questions:

                    result = models[method](

                        question=question,

                        context=original\_text

                    )

                    if result['score'] > 0.1:  # Порог уверенности

                        answers.append(result['answer'])

                return ', '.join(filter(None, answers)) if answers else text

            except Exception as e:

                print(f"Ошибка roberta\_squad: {e}")

                return text

        elif method == 'flan\_t5':

            try:

                prompts = [

                    f"Extract product name and characteristics: {text}",

                    f"Describe this product briefly: {text}",

                    f"What are the main features of: {text}"

                ]

                results = []

                for prompt in prompts:

                    result = models[method](prompt, max\_length=128)[0]['generated\_text']

                    if result and len(result.strip()) > 0:

                        results.append(clean\_result(result))

                return ', '.join(filter(None, results)) if results else text

            except Exception as e:

                print(f"Ошибка flan\_t5: {e}")

                return text

        elif method == 'distilbert\_sst':

            try:

                doc = models['spacy'](original\_text.lower())

                parts = []

                nouns = [token.text for token in doc if token.pos\_ == 'NOUN']

                if nouns:

                    parts.append(' '.join(nouns[:2]))

                result = models[method](text)

                if result['score'] > 0.7:

                    parts.append(result['label'])

                return ', '.join(parts) if parts else text

            except Exception as e:

                print(f"Ошибка distilbert\_sst: {e}")

                return text

        if 'attribute\_classifier' in models:

            attributes = extract\_attributes(text, models['attribute\_classifier'])

            parts.extend(attributes)

    except Exception as e:

        print(f"Ошибка при нормализации методом {method}: {e}")

        return text

def clean\_text(text):

    """Очистка текста от мусора"""

    text = re.sub(r'[\.]{2,}', '', text)

    text = re.sub(r'[\(\)"]', '', text)

    text = re.sub(r'[^а-яА-Я0-9\s\-,]', '', text)

    return text.strip()

def is\_valid\_result(text):

    """Проверка валидности результата"""

    return (len(text.split()) >= 3

            and not text.endswith('...')

            and not 'нормализ' in text.lower())

def extract\_product\_info(doc, original\_text):

    """

    Извлечение информации о товаре с помощью NLP

    """

    parts = []

    nouns = [token.text for token in doc if token.pos\_ == 'NOUN']

    if nouns:

        parts.append(nouns[0])

    quantities = re.findall(r'\d+\s\*(?:шт|компл|набор|уп)', original\_text.lower())

    if quantities:

        parts.append(quantities[0])

    sizes = re.findall(r'\d+(?:[.,]\d+)?\s\*(?:см|мм|м|мл|л|г|кг)', original\_text.lower())

    if sizes:

        parts.append(sizes[0])

    materials = ['дерево', 'пластик', 'металл', 'стекло', 'керамика']

    for material in materials:

        if material in original\_text.lower():

            parts.append(f"материал - {material}")

            break

    colors = ['медь', 'красный', 'синий', 'зеленый', 'золото', 'серебро',

              'белый', 'черный', 'желтый', 'коричневый', 'бежевый']

    for color in colors:

        if color in original\_text.lower():

            parts.append(f"цвет - {color}")

            break

    return ', '.join(parts) if parts else original\_text

def format\_product\_description(text):

    """

    Форматирование описания товара

    """

    parts = []

    product\_types = ['набор', 'комплект', 'упаковка', 'штука', 'шт']

    for ptype in product\_types:

        if ptype in text.lower():

            match = re.search(fr'{ptype}\s\*\d+', text.lower())

            if match:

                parts.append(match.group())

                break

    sizes = re.findall(r'\d+(?:[.,]\d+)?\s\*(?:см|мм|м|мл|л|г|кг)', text.lower())

    if sizes:

        parts.extend(sizes)

    materials = ['дерево', 'пластик', 'металл', 'стекло', 'керамика']

    for material in materials:

        if material in text.lower():

            parts.append(f"материал - {material}")

            break

    colors = ['медь', 'красный', 'синий', 'зеленый', 'золото', 'серебро',

              'белый', 'черный', 'желтый', 'коричневый', 'бежевый']

    for color in colors:

        if color in text.lower():

            parts.append(f"цвет - {color}")

            break

    return ', '.join(parts) if parts else text

def process\_data(df, methods=None):

    """

    Обработка данных с использованием выбранных методов нормализации

    """

    if methods is None:

        methods = [

            'ruT5',

            'rugpt',

            'rembert',

            'rubert',

            'bert\_multilingual',

            'xlm\_roberta',

            'multilingual\_e5',

            'spacy',

            'tfidf'

        ]

    print("Инициализация моделей...")

    models = get\_normalization\_models()

    reference\_data = df['Product Name'].tolist()

    print("\nНормализация названий товаров...")

    for method in methods:

        column\_name = f'Normalized\_{method}'

        tqdm.pandas(desc=f"Нормализация методом {method}")

        df[column\_name] = df['Product Name'].progress\_apply(

            lambda x: normalize\_with\_models(x, models, method, reference\_data)

        )

    return df

def save\_normalized\_results(df, output\_dir='normalized\_results'):

    """

    Сохранение результатов нормализации в один CSV файл

    """

    if not os.path.exists(output\_dir):

        os.makedirs(output\_dir)

    timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S')

    results\_df = pd.DataFrame()

    results\_df['Original\_Name'] = df['Product Name']

    results\_df['Category'] = df['Category Name']

    results\_df['Subcategory'] = df['Subcategory Name']

    results\_df['Second\_level\_Subcategory'] = df['Second-level Subcategory Name']

    normalized\_columns = [col for col in df.columns if col.startswith('Normalized\_')]

    for col in normalized\_columns:

        method\_name = col.replace('Normalized\_', '')

        results\_df[method\_name] = df[col]

    output\_file = os.path.join(output\_dir, f'normalized\_results\_{timestamp}.csv')

    results\_df.to\_csv(output\_file, index=False, encoding='utf-8-sig')

    print(f"\nРезультаты нормализации сохранены в: {output\_file}")

    return output\_file

def extract\_attributes(text, model):

    """

    Извлечение атрибутов с помощью предобученной модели

    """

    try:

        predictions = model(text)

        attributes = []

        for pred in predictions:

            if pred['score'] > 0.7:

                label = pred['label']

                if label.startswith('color\_'):

                    attributes.append(f"цвет - {label.replace('color\_', '')}")

                elif label.startswith('material\_'):

                    attributes.append(f"материал - {label.replace('material\_', '')}")

        return attributes

    except Exception as e:

        print(f"Ошибка при извлечении атрибутов: {e}")

        return []

def main(limit='all'):

    print("Загрузка данных...")

    df = load\_data(limit)

    print(f"Загружено {len(df)} строк.")

    methods = [

        'ruT5',

        'rugpt',

        'rembert',

        'rubert\_tiny2',

        'multilingual\_e5',

        'multilingual\_use',

        'labse',

        'xlm\_roberta',

        'rembert',

        'spacy',

        'tfidf',

        'bert\_multilingual',

        'distilbert\_multilingual',

        'xlm\_roberta\_large',

        'mdeberta\_v3',

        'bertopic',

        'roberta\_squad',

        'flan\_t5',

        'distilbert\_sst'

    ]

    print("Обработка данных...")

    df = process\_data(df, methods)

    save\_normalized\_results(df)

if \_\_name\_\_ == "\_\_main\_\_":

  limit = 50

  main(limit)