МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет компьютерных наук

Кафедра цифровых технологий

*Использование моделей машинного обучения для поиска идентичных продуктов в электронной коммерции*

Бакалаврская работа

02.03.01 Математика и компьютерные науки

Профиль «Квантовая теория информации»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Зав. кафедрой | \_\_\_\_\_\_\_\_\_\_\_ | Кургалин С. Д. | д.ф.-м.н., профессор | \_\_.\_\_.20\_\_\_ |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_\_ | Чикирякина К. А. |  |  |
| Руководитель | \_\_\_\_\_\_\_\_\_\_\_ | Семенов М. Е. | д.ф.-м.н., профессор |  |

Воронеж 2024

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ВГБОУ ВО «ВГУ»)

Факультет компьютерных наук

Кафедра цифровых технологий

УТВЕРЖДАЮ

заведующий кафедрой

цифровых технологий

\_\_\_\_\_\_\_\_\_\_\_ С. Д. Кургалин

\_\_. \_\_. 20\_\_

**ЗАДАНИЕ**

**НА ВЫПОЛНЕНИЕ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ ОБУЧАЮЩЕГОСЯ** Чикирякиной Карины Александровны

1. Тема работы «*Использование моделей машинного обучения для поиска идентичных продуктов в электронной коммерции»*, утверждена решением ученого совета факультета компьютерных наук от 29.11.2023
2. Направление подготовки: 02.03.01 Математика и компьютерные науки
3. Срок сдачи законченной работы: 13.06.2024
4. Календарный план: (строится в соответствии со структурой ВКР)

|  |  |  |  |
| --- | --- | --- | --- |
| № | Структура ВКР | Сроки выполнения | Примечание |
| 1 | Введение | 10.10.2023-15.10.2023 |  |
| 2 | 1 Анализ литературы | 17.10.2023-20.12.2023 |  |
| 3 | 2 Основные теоретические понятия | 15.01.2024-10.04.2024 |  |
| 4 | 2.1 Матчинг продуктов | 15.01.2024-10.04.2024 |  |
| 5 | 2.2 Векторное представление | 15.01.2024-10.04.2024 |  |
| 6 | 2.3 Трансформер | 15.01.2024-10.04.2024 |  |
| 7 | 2.4 Rubert-Tiny | 15.01.2024-10.04.2024 |  |
| 8 | 2.5 ResNet34 | 15.01.2024-10.04.2024 |  |
| 9 | 2.6 ArcFace | 15.01.2024-10.04.2024 |  |
| 10 | 2.7 Кросс-энтропия | 15.01.2024-10.04.2024 |  |
| 11 | 3 Программная реализация | 18.04.2024-05.05.2024 |  |
| 12 | 3.1 Данные | 18.04.2024-05.05.2024 |  |
| 13 | 3.2 Модель | 18.04.2024-05.05.2024 |  |
| 14 | 3.3 Программная реализация | 18.04.2024-05.05.2024 |  |
| 15 | 3.4 Результаты | 18.04.2024-05.05.2024 |  |
| 16 | Заключение | 07.05.2024-02.06.2024 |  |
| 17 | Список использованных источников | 07.05.2024-02.06.2024 |  |
| 18 | Приложение А | 07.05.2024-02.06.2024 |  |

Обучающийся \_\_\_\_\_\_\_\_\_\_ Чикирякина К. А.

Руководитель \_\_\_\_\_\_\_\_\_\_ Семенов М. Е.

**РЕФЕРАТ**

Бакалаврская работа 125 с., 51 рис., 21 использованный источник, 1 приложение.

МОДЕЛЬ МАШИННОГО ОБУЧЕНИЯ, МАТЧИНГ, ТРАНСФОРМЕР, ЭЛЕКТРОННАЯ КОММЕРЦИЯ

Объект исследования - модель машинного обучения для сравнения цен

Цель исследования - создание модели машинного обучения поиска идентичных продуктов для сайта сравнения товаров

Результаты работы: изучены существующие решения в сфере «матчинга» продуктов с помощью машинного обучения, разработана математическая модель, успешно применяющаяся компанией, а также проведено сравнение результатов разработанной программы и старого алгоритма сайта на основе сравнения символов названий.

Область применения результатов: сфера электронной коммерции, а также любые другие области, где необходим поиск идентичных объектов на основе текстовых данных и изображений.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc169256555)

[1 АНАЛИЗ ЛИТЕРАТУРЫ 9](#_Toc169256556)

[2 ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОНЯТИЯ 23](#_Toc169256557)

[2.1 Матчинг продуктов 23](#_Toc169256558)

[2.2 Векторное представление 23](#_Toc169256559)

[2.3 Трансформер 28](#_Toc169256560)

[2.4 Rubert-Tiny 36](#_Toc169256561)

[2.5 ResNet34 38](#_Toc169256562)

[2.6 ArcFace 47](#_Toc169256563)

[2.7 Кросс-энтропия 50](#_Toc169256564)

[3 ПРОГРАММНАЯ ЧАСТЬ 51](#_Toc169256565)

[3.1. Данные 51](#_Toc169256566)

[3.2. Модель 52](#_Toc169256567)

[3.3 Программная реализация 54](#_Toc169256568)

[3.4. Результаты 59](#_Toc169256569)

[ЗАКЛЮЧЕНИЕ 64](#_Toc169256570)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 65](#_Toc169256571)

[ПРИЛОЖЕНИЕ А 68](#_Toc169256572)

# [ВВЕДЕНИЕ](#_ВВЕДЕНИЕ)

С развитием интернета и услуг электронной коммерции количество товаров, продаваемых через онлайн-площадки, стремительно растет с каждым годом. По оценкам недавних исследований, общий оборот рынка розничной интернет-торговли в России по итогам 2022 года составил 5,7 трлн рублей, а всего на электронную торговлю в 2023 году пришлось более 22% от всех розничных продаж в мире. Однако обилие предложений означает, что покупатели часто ошеломлены их разнообразием и не могут выбрать то, которое лучше всего соответствует их предпочтениям. В процессе поиска и покупки товара существует большая проблема, с которой приходится сталкиваться потребителям. Один и тот же продукт можно найти в разных маркетплейсах, но информация о нем в источниках сильно отличается. Кроме того, у товаров нет глобальных идентификаторов, а предложения чаще всего не связаны между собой. Таким образом, у потребителей нет простого способа найти всю необходимую информацию и лучшие цены на продукты. Но даже в рамках магазина существует огромное количество копий товара от разных поставщиков с отличающимся описанием. Это может приводить к потере клиентов, ведь никому не хочется листать бесконечную ленту одинаковых карточек. Соответственно, проблема быстроразвивающейся интернет отрасли и отсутствие общих правил описания услуг осложняет не только восприятие информации пользователем, но и вредит владельцам самих площадок. Каждый клиент, не нашедший нужное предложение, и каждая компания, чьи продукты перестали активно продаваться на площадке, вносят значительный ущерб в доход владельцев и увеличивают конкуренцию. В этой связи необходимо рассматривать возможные решения проблемы, одним из которых является использование методов машинного обучения.

Машинное обучение - это совокупность математических методов и цифровых технологий. Интеллектуальные алгоритмы обучения уже умеют распознавать голос и изображения, занимаются аналитикой бизнес-процессов и используются в решении массы других задач. Их внедрение позволило не просто облегчить процесс программирования, но и совершило настоящую революцию во многих отраслях человеческой деятельности. Отличительная черта методов искусственного интеллекта – это не прямое решение, а обучение в процессе применения решений множества сходных задач. Для построения моделей применяются средства математической статистики, теории вероятности, методов оптимизации, численных методов, теории графов и другие разделы математики.

Решающую роль в применение машинного интеллекта на практике сыграло появление больших данных и достаточных для их обработки вычислительных мощностей. Чем качественнее данные, тем эффективнее будет работать программа. Более того, каким совершенным ни был бы алгоритм, если качество данных оставляет желать лучшего, или их слишком мало, то результат будет соответствующим. Поэтому практически всегда первым шагом будет либо сбор, либо анализ и обработка уже готового датасета. В сфере электронной коммерции владелец онлайн-платформы всегда способен предоставить данные о продуктах, имеющихся на его торговой площадке. Но, как мы уже выяснили, качество информации о продуктах оставляет желать лучшего. Например, названия товаров довольно часто содержит также и некоторые показатели: вес, объем, размер или процент жирности продукта, а вот в атрибуты их заносят далеко не всегда. В зависимости от поставленной задачи или выбранной модели машинного обучения будет необходимо предобработать названия, путем удаления лишней информации или переноса ее в нужный раздел. Также при работе с текстом существует практика удаления знаков препинания, приведение слов к начальной форме, изменение регистра текста. Изображения тоже необходимо подвергать изменениям: приведение к единому размеру, сглаживание или повышение резкости, удаление шума, применение фильтров.

После сбора и обработки данных разработчик может переходить к выбору уже самой модели. «Матчингом» называют процесс соотнесения похожих объектов в машинном обучении. Это как раз тот тип алгоритмов, который подходит для нашей указанной выше задачи. Казалось бы, можно просто сравнить каждый продукт с каждым, но на практике все не так уж и просто. Для декартова произведения хотя бы миллиона товаров расчет их похожести потребует огромных вычислительных ресурсов, именно поэтому приходится придумывать изощренные способы достижения цели. Существуют разные подходы матчинга, но в основном все они сводятся к образованию скрытого представления объекта, содержащего основную информацию о продукте. Энкодер создает векторы в латентном пространстве таким образом, что идентичные или схожие товары будут располагаться рядом друг с другом, образуя кластеры. Соответственно, наименее похожие продукты будут размещены максимально далеко. Для выявления скрытого представления изображения используют, например, сверточные нейронные сети, а для текстовых данных существует большое количество моделей BERT. Мы можем использовать эти представления, избегая сравнивания всех товаров попарно друг с другом. Достаточно посмотреть только на те, которые находятся максимально близко в пространстве. Для этого можно, например, подобрать подходящую метрику и отсеять все пары, расстояние до которых превышает порог.

Заключительным этапом создания любой модели машинного обучения является анализ полученных результатов. Хорошо, если у разработчика есть возможность получить размеченные пары продуктов с метками их схожести: 0 – разные товары, 1 – идентичные. Тогда можно легко увидеть, какой процент правильных ответов программа выдает на тестовых данных. Если же результатом алгоритма является коэффициент схожести продуктов, и нет идентификатора, с которым можно было бы сравнить результат, то задача значительно усложняется. В таком случае можно уменьшить размерность пространства до трехмерного и посмотреть насколько хорошо сгруппированы товары. Если продукты из одной категории или похожие по смыслу, то они на графике скорее всего сгруппируются в кластеры. Совершенно разные товары будут находиться максимально далеко. Но не стоит полностью доверять данному способу, так как понижение размерности не всегда оправдано и применимо. Другой способ, поиск готового датасета в интернете с похожими продуктами или создание искусственного набора данных. Такой набор будет содержать часть исходных продуктов, которая не участвовала в обучении, и искусственные товары, полученные путем изменения названий или показателей первых. Они могут отличаться цветом, размером, вкусом, объемом или просто порядком слов в названии.

В задаче, которая была поставлена передо мной, изначально размеченных данных не было. Так же описания и названия продуктов требовали хорошей обработки, а изображения были не у всех товаров. Мне нужно было создать модель машинного обучения для сайта Sarawan, которая сравнивает продукты питания и бытовые товары на разных интернет-площадках, а затем предлагает своим пользователям карточки для каждого из них с разными ценами из этих магазинов. Таким образом, пользователь может в одном месте выбрать продукты по самым низким ценам, что довольно удобно и просто. На основе описанной проблемы была определена следующая цель и задачи проекта.

Цель работы – создание модели машинного обучения поиска идентичных продуктов для сайта сравнения товаров.

Задачи:

1. Исследовать существующие разработки в данной сфере, выбрать подходящую модель на основе предоставленных компанией возможностей.
2. Подготовить датасет для обучения.
3. Реализовать модель машинного обучения, используя текст и изображения товаров.
4. Протестировать программу на новых данных и проанализировать получившийся результат.

# 1 АНАЛИЗ ЛИТЕРАТУРЫ

В последние годы сопоставление товаров и их категоризация в интернете изучались с помощью различных подходов и методов. Поскольку глобальные идентификаторы отсутствуют, а ссылки между различными веб-страницами электронной коммерции скудны, выяснение того, относятся ли два предложения на разных сайтах к одному и тому же продукту, является нетривиальной задачей. Таким образом, сопоставление продуктов связано с идентификацией пар или наборов идентичных продуктов.

Szymon Lukasik и Andrzej Michalowski [1] в своей статье предлагают подход, основанный на текстовом анализе, учитывающий неполную информацию о продукте. Сравнения производятся на основе пар товаров из двух интернет магазинов бытовой электроники (A и B) (рисунок 1), результатом является набор соответствий с коэффициентом сходства Sij ∈ [0,1] (порог p = 0,5). Задачу бинарной классификации в статье предлагается решать с помощью ансамбля деревьев XGBoost.

Товары представляются вещественными векторами из шести признаков: сходство Жаккара для названий, косинусное сходство между векторами категорий построенных с помощью fastText, расстояние Дамерау-Левенштейна для бренда, относительная разница цен товаров и двумерный вектор, полученный с помощью метода главных компонент для категорий. Вручную было размечено 500 пар товаров из двух интернет-магазинов, результаты также анализировались вручную. Из 38 совпадений (p = 0.5) 31 пара была идентифицирована верно, что составило 81% от всех совпадений.

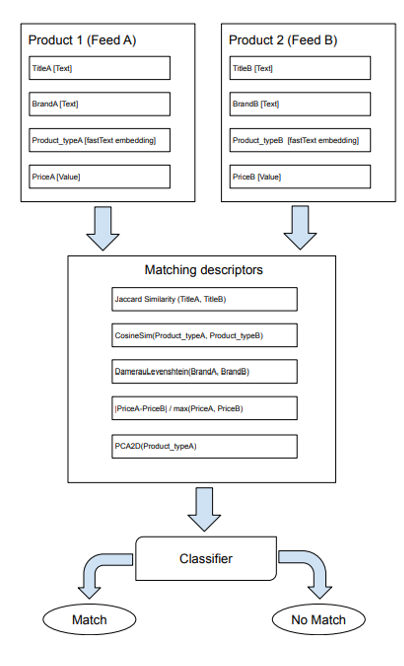


Рисунок 1 – Модель Szymon Lukasik и Andrzej Michalowski

Почему данная модель не подходит для нашей цели? На интернет-платформе Sarawan одновременно сравнивается сразу несколько магазинов, технически составить все возможные пары из тысяч и миллионов продуктов зачастую невозможно, поэтому необходимо использовать архитектуру модели, которая позволит сближать идентичные товары среди множества других. Так же в представленном решении не учитываются остальные атрибуты предложений: цвет, размер, форма, запах и другое. Интересная идея использовать косинусное расстояние между «встраиваниями» категорий, построенное с помощью fastText. Далее мы применяем эту схему для векторных представлений товара, учитывающих полную информацию о них.

Jeremy Foxcroft и Tianle Chen [2] анализировали результаты случайного леса, логистической регрессии, метода опорных векторов, гибридной модели Deep Matcher (двунаправленной рекуррентной сети с механизмом внимания) и бертов (RoBERTa и DistilBERT) на фармацевтических данных и товарах для красоты и здоровья двух интернет-магазинов. В начале данные очищаются, стандартизируются, затем удаляются совершенно разные, оставляя только продукты у границы разделения (трудно отличимые) (рисунок 2).

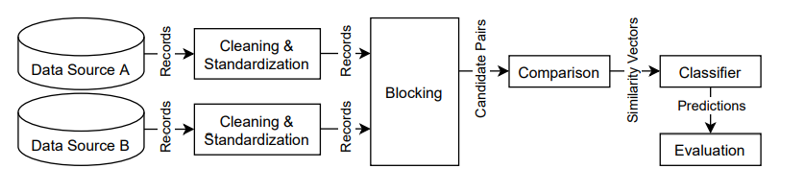


Рисунок 2 – Модель Jeremy Foxcroft и Tianle Chen

Используется информация о названии, бренде и категории, также строятся вектора и находится косинусное сходство между ними. Для текстовых данных считается: расстояние Левенштейна, Жаккара, косинусное сходство и другие (рисунок 3). Наилучшие результаты показали случайный лес и берты.

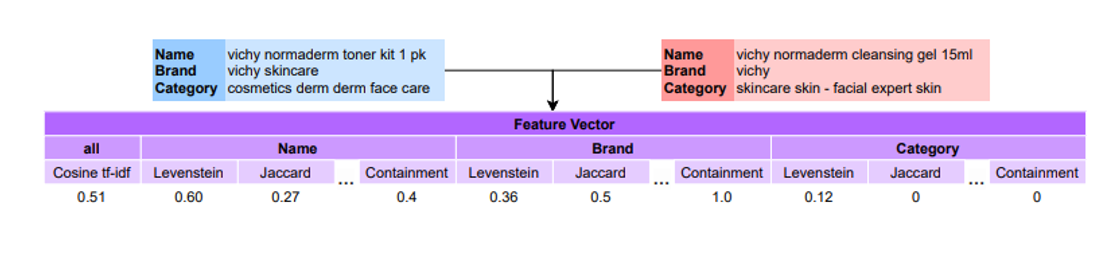


Рисунок 3 – Информация, которая учитывается в моделе Jeremy Foxcroft и Tianle Chen

Наиболее интересная структура модели с использованием глубокого обучения у Ajikya More [3]. Предлагаются разные модели для матчинга продуктов по названиям, изображениям, атрибутам и ценам, что выглядит более удачно, по сравнению с предыдущими исследованиями. Векторные представления для названий пар продуктов были построены с помощью skipgram, далее они конкатенируются, проходят через слой свертки, maxpooling, полносвязный слой и рассчитывается сходство между полученными векторами (рисунок 4).

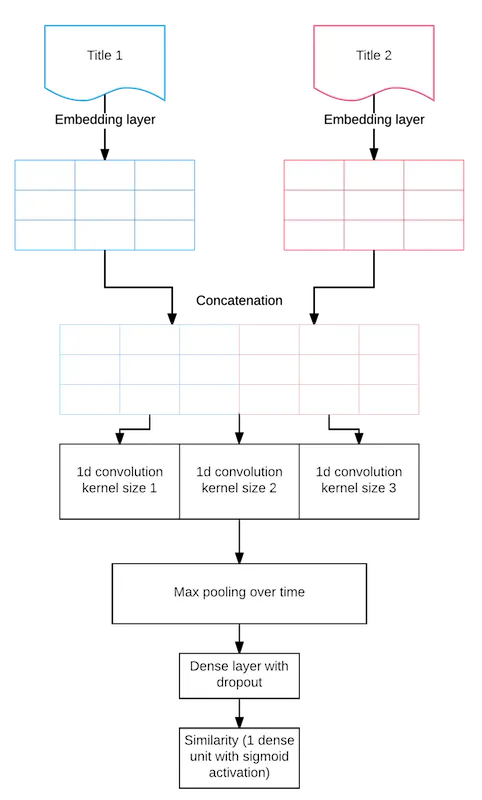


Рисунок 4 – Текстовая модель Ajikya More

Для изображений тестируются несколько архитектур: VGG16, VGG19, Inception, Resnet50 [4] (рисунок 5). Причина использования нескольких моделей заключается в том, что разные модели чувствительны к разным особенностям данных изображения продукта. Затем аналогично другим исследованиям считается косинусное сходство.

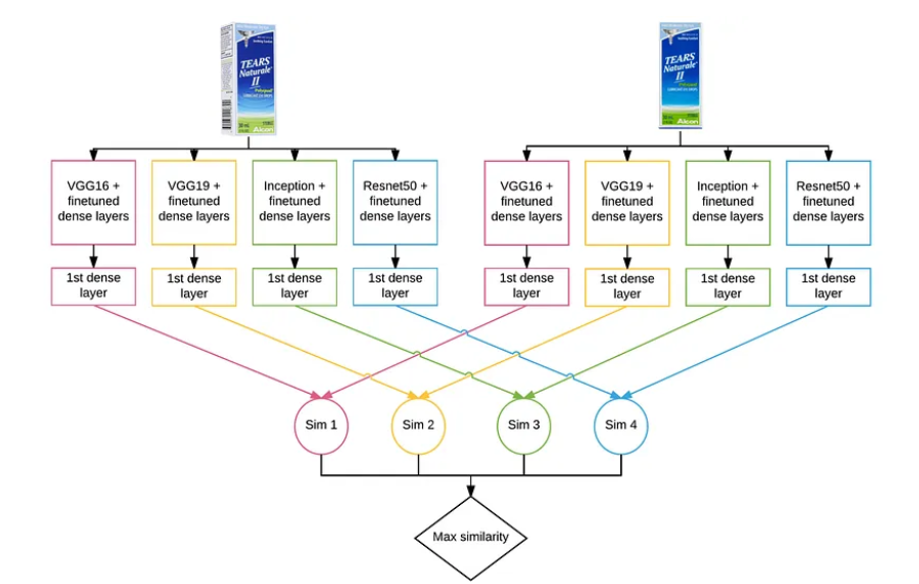


Рисунок 5 – Модель для изображений у Ajikya More

Атрибуты делятся на закрытые (фиксированное количество значений, например, цвет, формат книги) и открытые. Первые проходят ту же процедуру, что и названия (рисунок 6), а вторые токенизируются тремя метками (B-бренд начало, I-бренд конец, O-не бренд). Тестируются следующие модели: LSTM, CRF, перцептрон, цепи Маркова, KNN. Вектора для цен товаров строятся из Q-теста Диксона, теста Граббса, теста хи-квадрат, теста Бартлетта.

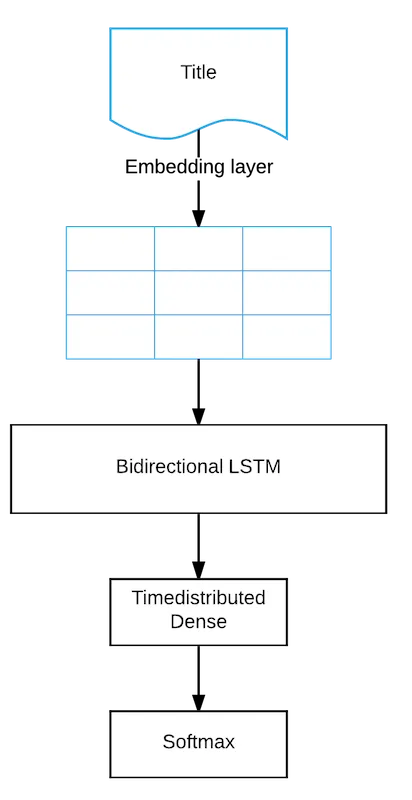


Рисунок 6 – Модель для атрибутов товаров у Ajikya More

В итоге в исследовании достигается точность около 85-90% для продуктов, которые реально разные. Идея разделить матчинг названий, изображений и атрибутов звучит интересно, но в статье достаточно скромно представлены полученные результаты автора, поэтому чуть позже рассмотрим еще одну архитектуру, которая и будет в последствие видоизменена и реализована.

После анализа существующих разработок, выяснилось, что не так уж и много существует в открытом доступе матчинга продуктов, подходящих под наши цели. Авторы либо не учитывают полную информацию о товаре, например, используют только названия, либо предоставляют не полные результаты, на основе которых можно было бы судить о работоспособности архитектуры. Используются и классические модели машинного обучения (хорошо себя показал случайный лес), и глубокое обучение на примере моделей BERT. Интересной показалась идея построения векторного представления продуктов без разметки данных и поиск косинусного сходства между ними. Для этого не придется использовать много ресурсов, что важно для нас, так как проект разрабатывается для стартап.

Нашлась и еще одна статья от компании OZON [5], в которой используются все перечисленные выше преимущества: товары представляются в виде векторов в N-мерном пространстве; считается косинусное сходство; учитывается информация от названий, изображений и атрибутов.

Обучение строится на предсказании самого мелкого класса продукта. Каждый продукт представлен деревом категорий (рисунок 7): cat1 - самая крупная категория (одежда, техника, спортивный инвентарь), cat3 – самая мелкая (футболка, телефон, скакалка). Низкоуровневых категорий насчитывается у них несколько тысяч.

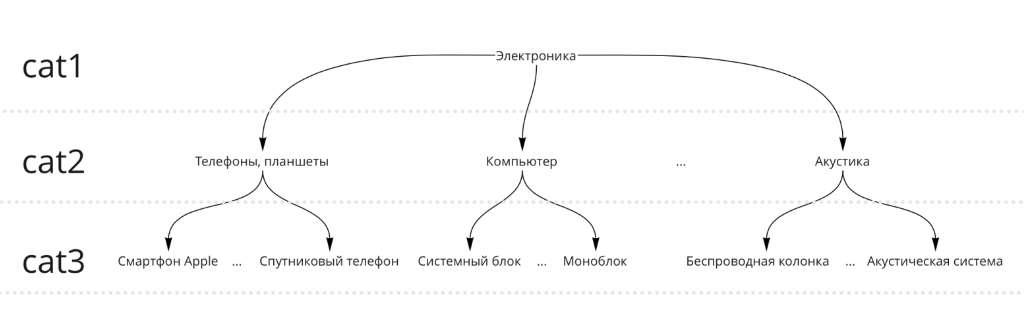


Рисунок 7 – Дерево категорий

Перед обучением данные прошли большую обработку, так как некоторые продукты очень редкие (ультразвуковой аппарат), другие вложены друг в друга или слишком похожи, но в разных категориях (витаминно-минеральные комплексы и биологически активные добавки (БАД)). Некоторые категории были просто склеены или удалены. На выходе получился датасет примерно 5 млн. строк, 1300 cat3 и минимум 500 единиц товара на одну категорию. Текст привели к нижнему регустру, убрали лишние знаки. Картинки аугментировали стандартными способами (горизонтальные, вертикальные отображения, изменения яркости и контраста). Из атрибутов убрали те, которые не несут особого смысла и встречаются почти у всех товаров (серийный номер). После этого попробовали разные варианты: подавать на вход каждый атрибут «ключ: значение» отдельно или же все атрибуты объединить в одну строку. В итоге разницы не было обнаружено, поэтому предпочли из-за удобства второй случай.

Для классификации огромного количества категорий вместо обычного softmax (показавшего не очень хорошие результаты) использовался другой подход, который изначально был предложен для задачи распознавания лиц людей - [ArcFace](https://arxiv.org/abs/1801.07698).

Преимущество ArcFace – определение близости выученных эмбеддингов внутри одного класса и отдалённость в разных: выбирая параметр margin penalty m, мы можем регулировать, насколько сильно мы хотим сближать или отдалять эмбеддинги одного или разных классов.

Различать сразу мелкую категорию товара для модели оказалось тяжело: на каждой итерации она пытается обучать картиночную, текстовую и атрибутную модели по одному финальному лоссу CrossEntropy [6] для cat3. В итоге веса плохо и медленно сходились. Поэтому было принято решение модифицировать архитектуру: после каждого слоя Softmax [7] на выходе энкодеров получаются промежуточные выходы — предсказания cat1 (более высокоуровневой категории). Итоговый лосс — взвешенная сумма всех лоссов, причём сначала придаётся больший вес лоссам cat1, а потом постепенно смещается в сторону cat3. Исходную архитектуру предложенную компанией OZON можно увидеть на рисунке 8

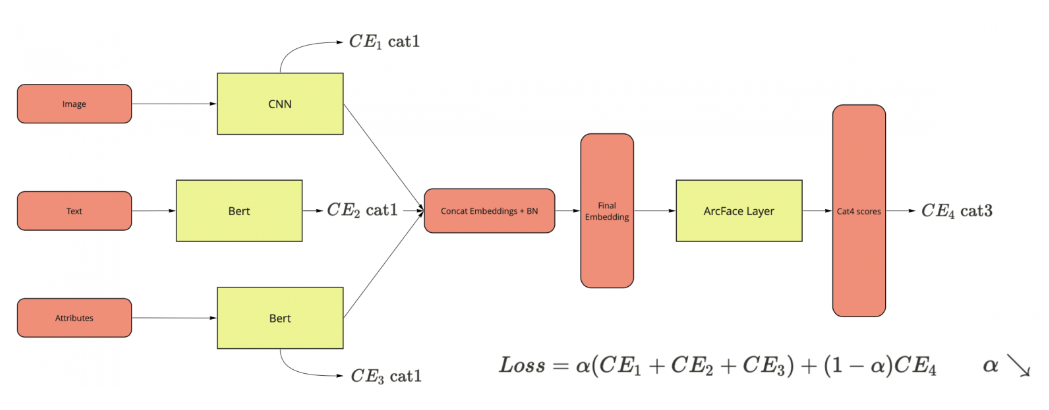


Рисунок 8 – Исходная архитектура модели «матчинга» компании OZON

Коэффициент взвешивания обычная экспоненциальная функция (рисунок 9).

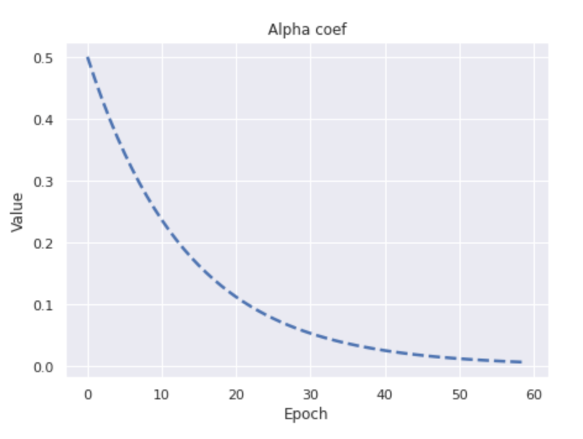


Рисунок 9 – Коэффициент взвешивания

После обучения предсказанные категории опускаются и не берутся, используются только векторы на выходе ArcFace. В качестве самих моделей для изображений использовался ResNet34 [8], оптимизатор SGD, а для текстовых данных Rubert-Tiny [9] и AdamW. Обучение протекало 60 эпох: 15 эпох с шагом (learning rate) побольше, потом уменьшалось. В результате на валидации получили 85% Acc@1 (рисунок 10). и 94% Acc@5 (рисунок 11). Для сравнения: обученный на заголовках fastText давал точность 60% Acc@1. Пример результата работы модели компании OZON можно увидеть на рисунке 12.

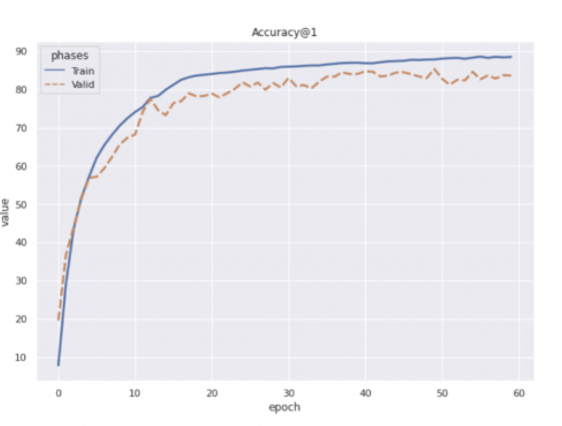


Рисунок 10 – Показатель точности Acc@5 модели компании OZON

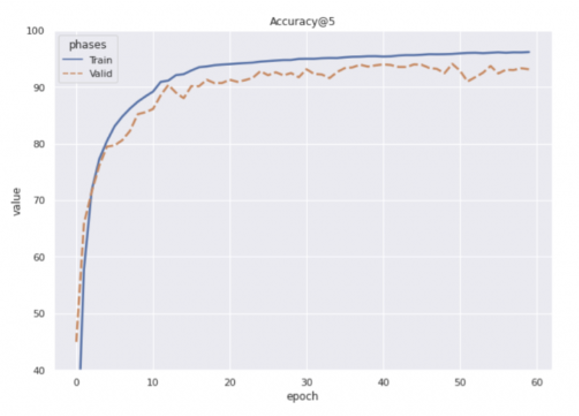


Рисунок 11 – Показатель точности Acc@1 модели компании OZON

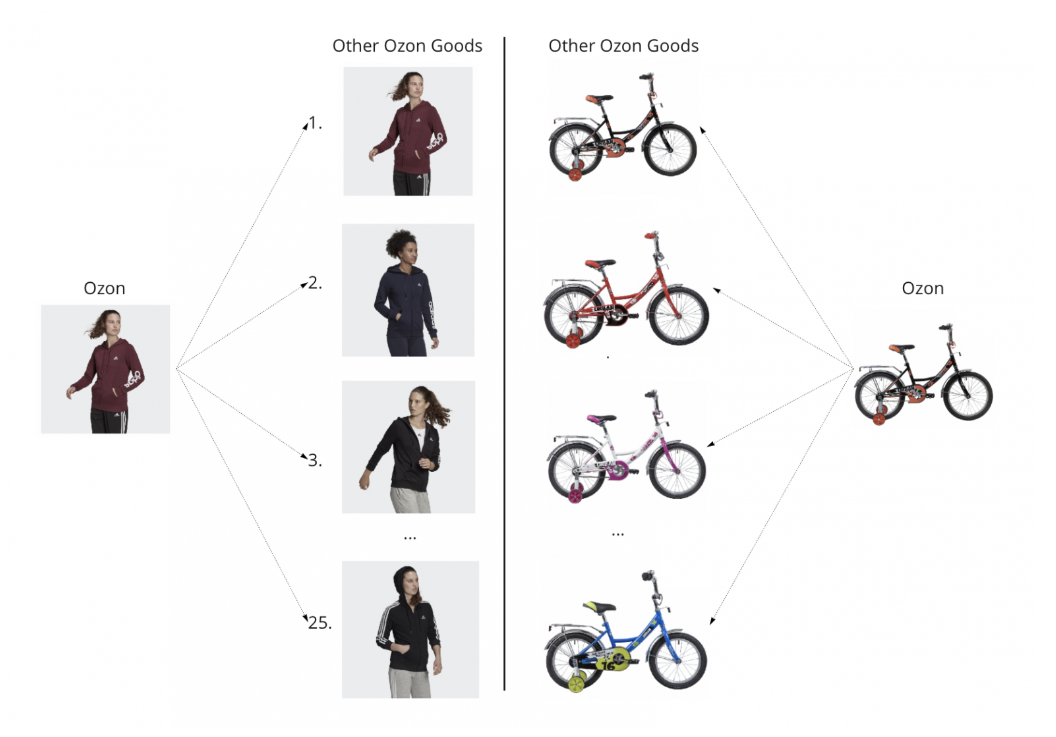


Рисунок 12 – Пример работы модели компании OZON

Архитектура матчинга товаров, представленная компанией OZON, значительно выделяется среди всех рассмотренных ранее статей: она учитывает больший объем данных о товаре (изображения, названия, атрибуты); обучается сначала на высокоуровневых категориях, затем на низкоуровневых; используются современные модели текстовой обработки данных со слоями внимания и, показавшие отличные результаты в обработке изображений, сверточные нейронные сети с остаточными соединениями.

Также в данной архитектуре благодаря замене функции softmax продукты в одной категории сильнее сближаются друг к другу и отдаляются от товаров других классов. Их идея также не предполагает разметки данных, что обходится дорого в нынешнее время. Это несомненно огромный плюс для любой компании. Полученные эмбеддинги товаров также можно использовать не только для матчинга продуктов, но и для ранжирования товаров, что может применяться в рекомендательных системах. Также компания OZON предоставила полную информацию о полученных результатах в виде графиков обучения, наборов найденных идентичных товаров и их процентном соотношении. Помимо этого, есть даже 3d-визуализация векторных представлений товаров в пространстве, построенная с помощью UMAP, на котором явно видно, что схожие товары расположены близко друг к другу (рисунок 13 и рисунок 14).

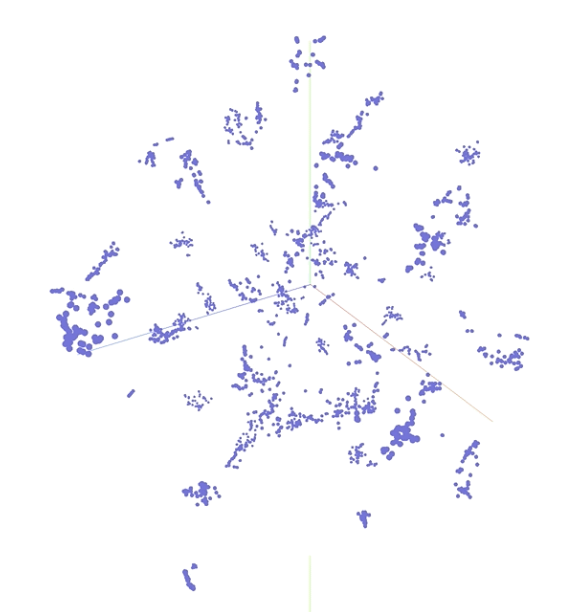


Рисунок 13 – Векторное представление товаров компании OZON

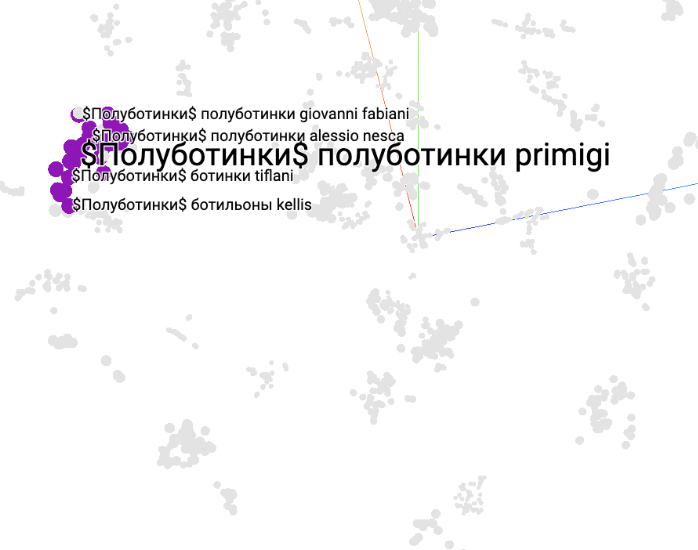


Рисунок 14 – Продукты, лежащие близко в векторном пространстве, построенном с помощью модели компании OZON

После анализа нескольких предложенных работ, было принято однозначное решение попробовать воспроизвести архитектуру OZON только уже на своих данных, которых намного меньше и не настолько идеально структурированных. Как позже выяснилось, для получения хороших результатов на своих товарах, было необходимо все-таки видоизменить предложенный вариант, а именно атрибуты обучать отдельно. Об этом будет подробнее рассказано в разделе о программной реализации ниже, но перед этим стоит ознакомиться с основными теоретическими понятиями, необходимыми для понимания темы.

# 2 ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОНЯТИЯ

## 2.1 Матчинг продуктов

В бизнесе, медицине, электронной коммерции не редкими являются задачи поиска похожих продуктов, пользователей, сравнение изображений с шаблонами. Причем область применения обширная, начиная с сайтов знакомств, рекомендаций видео и музыки, поиска товаров, заканчивая подбором терапии, наиболее подходящей для конкретного типа опухоли пациента, поиск донора и рецепиента, предложение торговых стратегий на основе анализа рыночных данных и поведения трейдера. В машинном обучении существует целое направление, которое решает класс таких задач, название которого «матчинг» (от англ. соответствие, сопоставление).

На самом деле используя текстовые данные можно решить проблему и без применения машинного обучения путем применения обработки предложений, регулярных выражений и сравнения строк, но часто возникают ошибки из-за синонимов или опечаток. При работе с изображениями в принципе не имеет смысла сравнивание пикселей, поэтому для достижения поставленной цели выбор однозначно был сделан в пользу современных решений.

В электронной коммерции распространенной проблемой являются персонализированные рекомендации или объединение идентичных товаров. Существуют разные подходы, но большая их часть сводится к созданию векторного представления товаров и пользователей.

## 2.2 Векторное представление

Векторное представление слов (embedding) [10] — это метод, используемый в обработке естественного языка, для отображения словаря в многомерные векторы фиксированной размерности. Эти векторы принадлежат пространству с произвольной размерностью и сохраняют семантические отношения между словами. Формально, пусть — множество всех слов в словаре, тогда каждому слову сопоставляется вектор , где — размерность векторного пространства. Векторы сконструированы таким образом, что косинусное сходство или евклидово расстояние между ними отражает семантическую близость.

Существуют разные подходы для векторизации слов, самым простым из которых является, наверное, «горячее кодирование» (one-hot encoding). При горячем кодировании преобразуются категориальные данные в многомерные двоичные векторы, количество измерений в которых соответствует количеству категорий. Затем кодируется каждая категория, сопоставляя ее с вектором, в котором элемент, соответствующий размерности категории, равен 1, а остальные — 0. Например, предположим, что категориальная переменная обозначает прогноз погоды и имеет три категории: солнечно, дождь и ветер. Тогда кодировка может быть солнечно = [1,0,0], дождь = [0,1,0], ветер = [0,0,1] (рисунок 16)

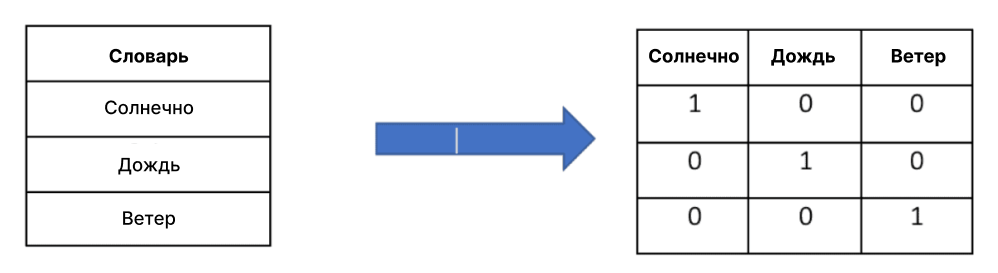


Рисунок 15 – «Горячее кодирование» (one-hot encoding)

Недостатками горячего кодирования являются то, что по векторным представлениям нельзя судить о семантическом сходстве слов, и размерности имеют очень большие значения, которые еще зависят от размера самого словаря.

Word2Vec — это другой метод для построения компактного векторного пространства слов, использующий нейронные сети. Он принимает на вход большой текстовый корпус и сопоставляет каждому слову вектор. Сначала создается словарь, а затем вычисляются векторные представления слов. Эти представления основаны на контекстной близости: слова, которые часто встречаются рядом в тексте (и, следовательно, имеют схожий смысл), имеют высокое косинусное сходство в векторном пространстве.

В Word2Vec существуют две основные модели обучения: Skip-gram и CBOW (Continuous Bag of Words). В модели Skip-gram по текущему слову предсказываются слова из его контекста, а в модели CBOW по контексту подбирается наиболее вероятное слово. На выходном слое используется функция softmax или её вариации, чтобы получить распределение вероятностей для каждого слова. Word2vec принимает на входе большой корпус текста и создает векторное пространство (пространство признаков), обычно состоящее из нескольких сотен измерений, причем каждому уникальному слову в корпусе назначается соответствующий вектор в пространстве. Это делается путем создания пар контекста и целевого слова, что дополнительно зависит от так называемого размера окна. Например, есть предложение «Русский медленно запрягает да быстро едет», тогда тренировочные данные с окном длины два будут выглядеть следующим образом: [русский, медленно], [русский запрягает], [медленно, русский], [медленно запрягает], [медленно, да], [запрягает, медленно], [запрягает, да], [запрягает, быстро] и т. д. Таким образом получаются пары «контекст-цель», которые можно использовать для обучения. Мы берем вектор быстрого кодирования контекстного слова и передаем его в нейронную сеть. Эти входные данные умножаются на веса скрытых слоев, и в конце мы получаем выходной вектор вероятностей размера словаря. Затем кросс-энтропия применяется для вычисления потерь и обновления веса.

Одной из проблем частотного анализа является то, что слова с самой высокой частотностью получают наивысшую оценку. Однако такие слова могут не содержать значимой информации для модели по сравнению с менее частыми словами. Один из методов решения этой проблемы — снижение оценки слова, которое часто встречается во всех схожих документах. Этот подход называется TF-IDF — это статистическая мера, которая позволяет оценить значимость слова в документе, входящем в состав корпуса. Оценка по TF-IDF увеличивается пропорционально частоте появления слова в документе, но это компенсируется количеством документов, содержащих это слово.

Формула для расчета оценки слова X в документе Y выглядит следующим образом:

где tf – отношение числа вхождений слова к общему числу слов документа, idf — инверсия частоты, с которой некоторое слово встречается в документах коллекции.

Важно понимать, что алгоритм не умеет определять смысловую ёмкость слов, а распределяет их по частоте использования в корпусе текста, который подается на вход при обучении. Таким образом, если корпус будет содержать разрозненные наборы текстов, в которых не прослеживается частотность слов, то хорошего обучения не произойдет. Поэтому для достижения поставленной цели были рассмотрены более сложные архитектуры, учитывающие не только частоту встречаемости слов в предложении, но и связь слов друг с другом с помощью слоев внимания.

Перед тем, как перейти к рассмотрению таких архитектур, стоит обсудить преобразования изображений в вектора. Изображения в отличие от текста уже представлены числами, которые может обработать компьютер. Самый простой способ получить вектор из изображения это вытянуть значения всех пикселей из матрицы () , где – количество строк пикселей в изображении, – количество столбцов) в один длинный вектор (рисунок 17).

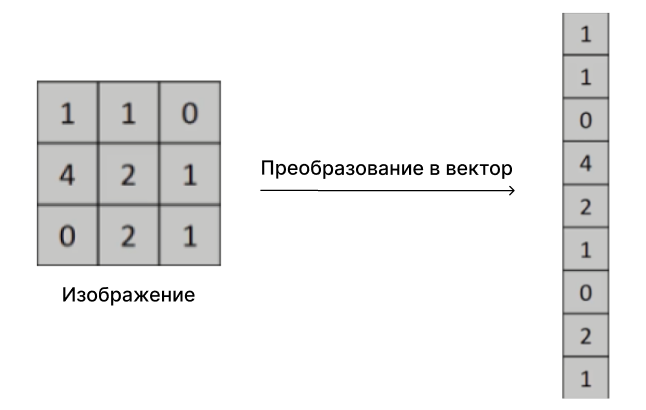


Рисунок 17 – Преобразование изображения в вектор

Недостатком данного подхода является тот факт, что при небольшом смещении объекта на фотографии, вектор уже будет иметь сильные отличия и модель машинного обучения не сможет идентифицировать объекты, как одинаковые. Данную проблему решают сверточные нейронные сети, речь о которых пойдет в следующих подразделах.

В целом после преобразования текстовых и визуальных данных о продуктах в векторные представления можно судить о схожести товаров по близости их векторов в пространстве (рисунок 18).

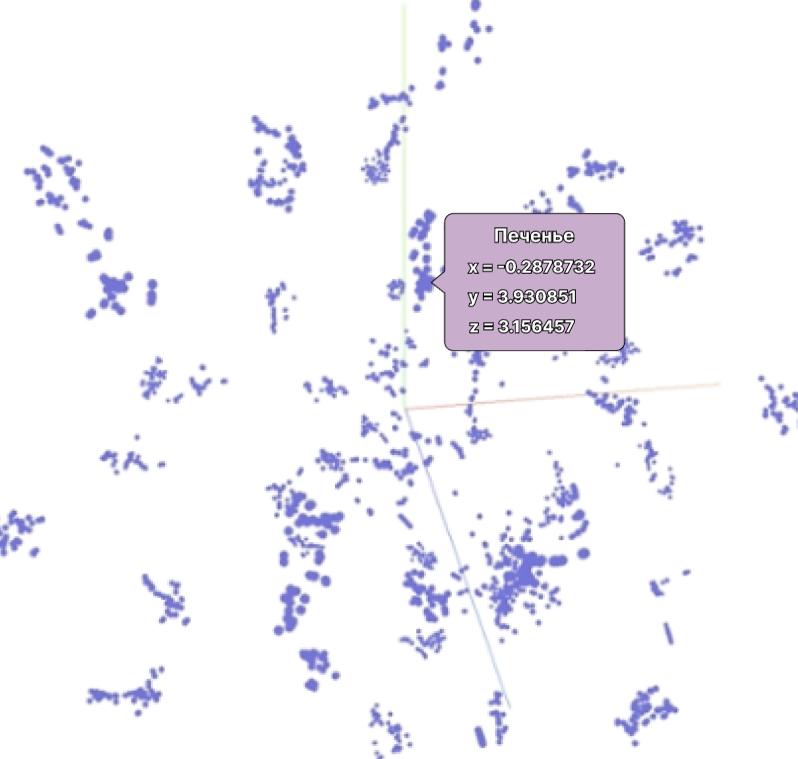


Рисунок 18 – Представление товара в векторном пространстве

Рассмотрев существующие простые подходы, перейдем теперь к более сложным архитектурам, которые лишены всех перечисленных выше недостатков.

## 2.3 Трансформер

В 2017 году исследователи из Google представили архитектуру Transformer (рисунок 19) в своей статье "Attention is all you need" [11]. Главная инновация Transformer заключается в использовании механизма внимания (self-attention), который позволяет модели фокусироваться на наиболее значимых элементах входной последовательности, а не на всех элементах сразу. До появления Transformer задачи обработки естественного языка (NLP) обычно решались с помощью рекуррентных нейронных сетей (RNN). Эти сети умеют сохранять внутреннее состояние, что позволяет им учитывать результаты предыдущих входных данных. Однако у RNN есть несколько существенных недостатков: задачи в них сложно параллелить и часто возникают проблемы затухающих и взрывающихся градиентов, что затрудняет обучение моделей на очень длинных последовательностях. Transformer решает эти проблемы благодаря механизму внимания, который заменяет рекуррентность. Внимание определяет веса важности элементов входной последовательности, что устраняет необходимость сохранять состояния и позволяет решать обе указанные проблемы RNN. Для эффективной работы языковой модели важно понимание таких аспектов, как семантика слов и их порядок в предложении. Модель должна уметь выделять ключевые слова и устанавливать между ними связи, используя исключительно числовые представления и трансформации. Эта задача была решена созданием трех матриц, работающих с векторами слов. В этих векторах представлена семантика каждого слова, вычисленная на основе частотности и совместной встречаемости слов. Кроме того, в них включена позиционная информация, которая рассчитывается с помощью синусоидальных и косинусоидальных функций.

Рассмотрим теперь более подробно архитектуру и принцип работы модели. Состоит трансформер из нескольких слоев энкодера и декодера. Здесь кодер отображает входную последовательность представлений символов (x1,...,xn) в последовательность непрерывных представлений z = (z1,...,zn). Учитывая z, декодер затем генерирует выходную последовательность (y1, ..., ym) символов по одному элементу за раз. На каждом этапе модель является авторегрессионной, используя ранее сгенерированные символы в качестве дополнительных входных данных при генерации следующего.

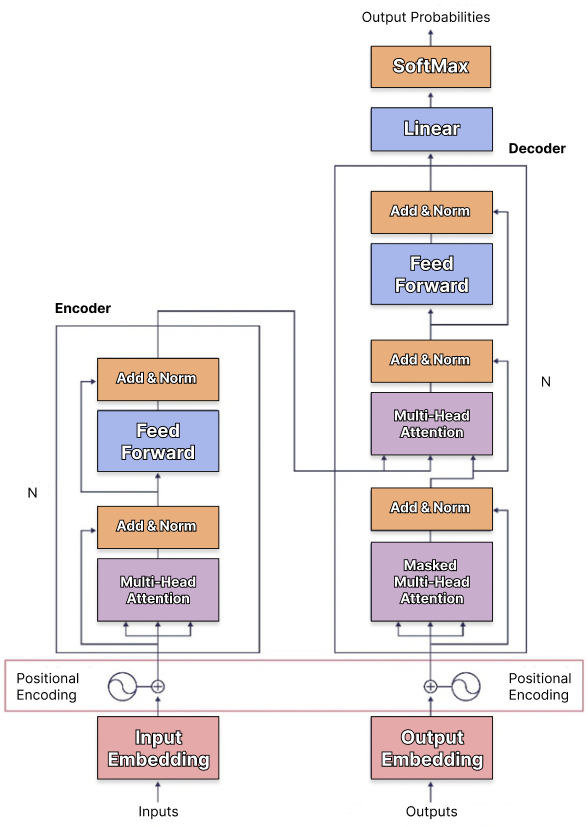


Рисунок 19 – Модель трансформер

Энкодер: Кодировщик состоит из одинаковых слоев, которые образуют стек. Каждый слой разделен на два подслоя. Первый подслой выполняет самообучение с использованием нескольких "головок" внимания, а второй подслой представляет собой простую полносвязную нейронную сеть. Для каждого подслоя применяется остаточное соединение, за которым следует нормализация слоя. Это означает, что выход каждого подслоя определяется как где — это функция, реализуемая подслоем. Формула нормализации:

Чтобы остаточные соединения работали корректно, все подслои и слои внедрения имеют одинаковую размерность выходных данных, равную обычно 512.

Декодер: Декодер также состоит из одинаковых слоев, как и кодировщик. Однако в каждом слое декодера, помимо двух подслоев, присутствует третий подслой, который выполняет многоголовое внимание к выходным данным кодировщика. Также, как и в кодировщике, в декодере используются остаточные соединения и нормализация слоя для каждого подслоя. В дополнение, в подслое самообучения декодера используется маскирование (рисунок 20), чтобы предотвратить использование информации из будущих позиций. Это маскирование, вместе с тем, что выходы смещены на одну позицию, гарантирует, что предсказания для позиции i зависят только от известных выходных данных для позиций, меньших чем i.

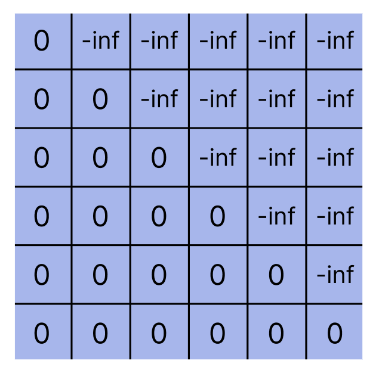


Рисунок 20 – Маска в декодере трансформера

Функцию внимания можно описать как сопоставление запроса и набора пар ключ-значение с выходными данными, где запрос, ключ, значение и выходные данные являются векторами. Выходные данные вычисляются как взвешенная сумма значений, где вес, присвоенный каждому значению, вычисляется с помощью функции совместимости запроса с соответствующим ключом.

Это особое внимание называется «Scaled Dot-Product Attention» (рисунок 21). Входные данные состоят из запроса (query) и ключа (key) измерения и значения (value) измерения . Мы вычисляем скалярное произведение запроса с ключом, делим на и применяем функцию softmax для получения вероятностей значений. В итоге получается некая карта связей каждого слова в предложении с каждым, чем больше число в ячейке, тем сильнее смысловая связь между словом в данной строке и столбце. Умножая полученную карту на исходное значение, получаем то самое внимание трансформера (рисунок 22).

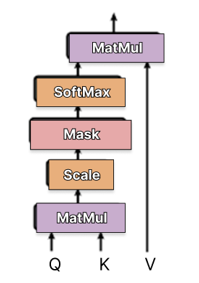


Рисунок 21 – Слой внимания в трансформере

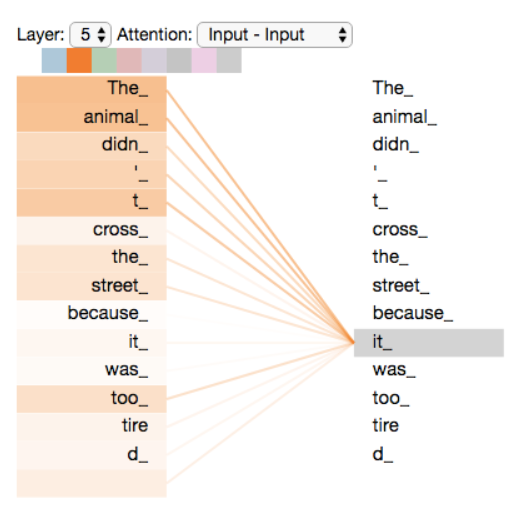


Рисунок 22 – Связи между словами в предложении выявленные слоем внимания

Вместо выполнения одной функции внимания с размерными ключами, значениями и запросами было обнаружено, что полезно выполнять операцию внимания раз, так каждый слой концентрируется на своих деталях и скрытых связях, что улучшает показатели модели, на выходе все они объединяются и проходят линейный слой. Это объединение нескольких слоев внимания называется многоголовым вниманием [12] (рисунок 23), которое позволяет модели совместно обрабатывать информацию из разных подпространств представления в разных позициях, при единой голове усреднение препятствует этому.

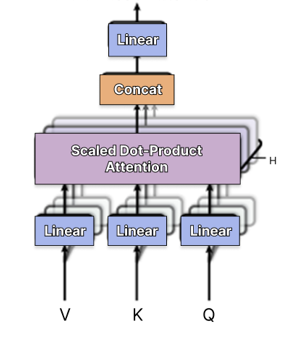


Рисунок 23 – Многоголовое внимание в трансформере

Помимо подуровней внимания, каждый из слоев в кодере и декодере содержит полносвязную сеть прямой связи, которая применяется к каждой позиции отдельно и одинаково. Он состоит из двух линейных преобразований с активацией ReLU между ними. Хотя линейные преобразования одинаковы в разных положениях, они используют разные параметры от слоя к слою.

Размерность ввода и вывода равна 512, а размерность внутреннего слоя 2048.

Поскольку модель не содержит повторений и сверток, чтобы модель могла использовать порядок последовательности, необходимо ввести некоторую информацию об относительном или абсолютном положении токенов в последовательности. С этой целью добавляются «позиционные кодировки» к входным векторам в нижней части стеков кодера и декодера. Позиционные кодировки имеют ту же размерность, что и вложения, поэтому их можно суммировать. Существует множество вариантов позиционных кодировок, изученных и фиксированных. В этой работе используются функции синус и косинус разных частот:

где pos — положение, а i — размер. То есть каждое измерение позиционного кодирования соответствует синусоиде. Длины волн образуют геометрическую прогрессию от 2π до 10000 · 2π. Мы выбрали эту функцию, потому что предположили, что она позволит модели легко научиться отслеживать относительные позиции, поскольку для любого фиксированного смещения может быть представлена ​​как линейная функция . Была выбрана синусоидальная версия, потому что она может позволить модели экстраполировать последовательности большей длины, чем те, которые встречаются во время обучения.

Архитектура Transformer обладает рядом значительных преимуществ, которые делают её особенно эффективной для задач обработки естественного языка (NLP) и других задач машинного обучения. Во-первых, Transformer позволяет выполнять вычисления параллельно, в отличие от рекуррентных нейронных сетей (RNN), что значительно ускоряет обучение и позволяет использовать более крупные модели. Во-вторых, трансформеры не страдают от проблемы затухающих и взрывающихся градиентов благодаря отсутствию рекуррентных связей, что делает обучение более стабильным, особенно на длинных последовательностях. Механизм внимания позволяет модели сосредотачиваться на наиболее важных частях входной последовательности, улучшая качество обработки информации и позволяя учитывать контекст на большом расстоянии. Transformer также эффективно справляется с последовательностями разной длины благодаря механизму внимания и позиционным векторам, которые вводят информацию о позиции слова в предложении. Модульная структура трансформера упрощает понимание, модификацию и масштабирование модели, добавляя или удаляя блоки по мере необходимости. Универсальность трансформеров позволяет успешно применять их не только в NLP, но и в других областях, таких как обработка изображений, аудио и видео, делая их мощным инструментом для различных задач машинного обучения. В итоге, трансформеры продемонстрировали превосходные результаты в задачах машинного перевода, суммаризации текста, генерации текста и ответов на вопросы, став основой многих современных моделей, таких как BERT, GPT и T5.

## 2.4 Rubert-Tiny

Главное новшество BERTа в способе предобучения. Ранние архитектуры, чтобы обучиться, предсказывали, какое слово вероятнее всего будет стоять следующим, учитывая все предшествующие до него. На решение нейросети влияли только слова слева, такие нейросети называются однонаправленными. Человек так не делает, он смотрит обычно на все предложение сразу, поэтому придумали двунаправленные нейросети. Две одинаковые нейросети работают параллельно, одна предсказывает слова слева направо, другая — справа налево, затем результат обеих сетей просто объединяется. Двунаправленная нейросеть на ряде задач справляется лучше, чем однонаправленная, но это тоже не совсем то, что хочется видеть. Поэтому BERT предобучается на «маскированной языковой модели». Ее суть в том, что нужно предсказать слово не в конце предложения, а где-то посередине. Маскированной модель называется потому, что искомый токен заменяется токеном [MASK]. Ранее нельзя было подать на вход (и учесть) все части фразы, а не только слова слева или справа. Открыть все предложение в однонаправленном Трансформере — значит дать готовый ответ системе, которая пытается угадать, что было дальше, следовательно, она ничему не научится. С новым методом обучения достигается «глубокая двунаправленность», когда модель на самом деле смотрит в обе стороны, а не склеивает отдельные представления о контексте слева и справа.

Выше говорилось в основном про предобучение, рассмотрим теперь дальнейшие шаги. Существует большое количество предобученных моделей BERT, которые понимают структуру текста и способны генерировать его. Когда же стоит задача написания модели для специфической задачи, а не простого предсказывания текста, то берется одна из таких программ и дообучается для уже конкретной задачи. Это помогает сэкономить время разработки и требует меньшей мощности компьютера. Если корпус, использованный для предварительного обучения, не слишком отличается от корпуса для дообучения, то обычно достигаются хорошие результаты. Однако есть несколько случаев, когда перед обучением специфичной для конкретной задачи головы необходимо дообучить языковые модели на своих данных. Например, если датасет содержит юридические контракты или научные статьи, модель трансформер, обычно рассматривает специфические для области слова в этом корпусе как редкие токены, и результирующее качество может быть менее чем удовлетворительным. Дообучив языковую модель на данных из домена, можно повысить качество работы во многих последующих задачах. Этот процесс дообучить предварительно обученную языковую модель на данных из домена обычно называют доменной адаптацией (domain adaptation).

По сути, BERT – это обученный стек энкодеров Трансформера. Лучше всего использовать эмбеддинги, предобученные на больших объемах текста, чем обучать их вместе с моделью на зачастую достаточно маленьком наборе данных. При дообучении замораживаются веса предварительно обученной части, сверху накладывается новый слой, который обучают уже на своих данных. Таким образом для нашей задачи был выбран Rybert-Tine [13] – это очень маленькая версия модели bert-base-multilingual-case для русского и английского языков (12М параметров). Он был обучен на корпусе Yandex Translate , OPUS-100 и Tatoeba , с использованием MLM-loss (от bert-base-multilingual-case ). Эта модель полезна, если нужно настроить ее для относительно простых задач на русском языке (например, NER или классификации настроений), и разработчика больше волнуют скорость и размер, чем точность. Он примерно в 10 раз меньше и быстрее, чем BERT базового размера. Его [CLS] вложения можно использовать в качестве представления предложения, согласованного между русским и английским языками.

## 2.5 ResNet34

Картинки в большинстве случаев представляют собой упорядоченный набор пикселей, где каждый пиксель – это вектор из трех каналов (интенсивность красного, зелёного и синего). Каждая интенсивность характеризуется числом от 0 до 1, но для привычных нам изображений этот интервал равномерно дискретизирован, чтобы уместиться в 8 бит (от 0 до 255), для экономии памяти. При этом (0, 0, 0), нулевая интенсивность, соответствует чёрному цвету, а (255, 255, 255), максимальная интенсивность, – белому. Картинка представлена несколькими строками пикселей, где длину каждой такой строки называют шириной W картинки, а количество строк – высотой H (т. е. тензор HxWx3).

Самый простой способ построить нейронную сеть для решения задачи классификации – это «развернуть» картинку в вектор, а затем использовать обычную многослойную сеть с кросс-энтропией в качестве лосса. Однако, такой подход имеет несколько недостатков: в первом слое получается  параметров, где – это количество нейронов в первом слое. Если поставить  слишком маленьким, есть риск потерять много важной информации, особенно, если рассматривать картинки размером, например, 1920x1080. Если же выставить большим, рискуем получить слишком много параметров (а это только первый слой), а с этим и все вытекающие проблемы (переобучение, сложность оптимизации), также структура данных никак не учитывается, а именно нейронная сеть не понимает после преобразований сдвига или поворота, что на картинке расположен тот же объект, то есть результат должен быть инвариантен к описанным преобразованиям.

Сначала рассмотрим, как с помощью сверточных нейронных сетей [14] решается задача инвариантности к преобразованиям. Объект может быть в любом месте картинки и нельзя точно сказать, где лучше всего модель научилась распознавать его. Для надёжного предсказания будет логично посдвигать картинку на все возможные смещения (пустоты заполнены нулями) (рисунок 24). Затем для каждого смещения предсказывается вероятность наличия объекта на картинке. Получившиеся предсказания можно агрегировать любым способом: среднее, максимум и тп. Таким образом модель обрабатывает не один пиксель, а сразу некоторую область изображения, что помогает ей шире «смотреть» и лучше распознавать различные шаблоны (линии, объекты, фигуры).

Чтобы справится с проблемой количества параметров, уменьшается размер окна с  до  (обычно нечётное и ).В этом случае получается радикальное снижение количества параметров и сложности вычислений.

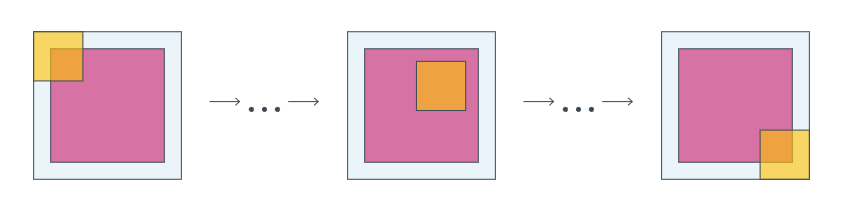


Рисунок 24 – Движение окна по изображению

К сожалению, с таким подходом возникает новая проблема: предсказание для какого-то окна никак не учитывает контекст вокруг него. Получается, мы можем получить разумные предсказания только в случае, если распознаваемый объект обладает признаками, которые «помещаются» в окно свёртки (например, лого автомобиля при классификации марок машин), либо объекты заметно отличаются по своей текстуре (шерсть кошки, а не кирпич). Тогда вместо одного слоя, берется несколько. Пусть на первой стадии делается  разных свёрток с фильтром размером . Результаты каждой свёртки можно упорядочить в виде новой картинки. После чего применяется K новых аналогичных свёрток для получения предсказаний для каждого пикселя. В итоге финальные нейроны уже могут «видеть» почти всю нужную информацию для хорошего предсказания (рисунок 25). Более того, возникает меньшее количество параметров и падает сложность вычислений в сравнении с использованием одной большой полносвязной сети.

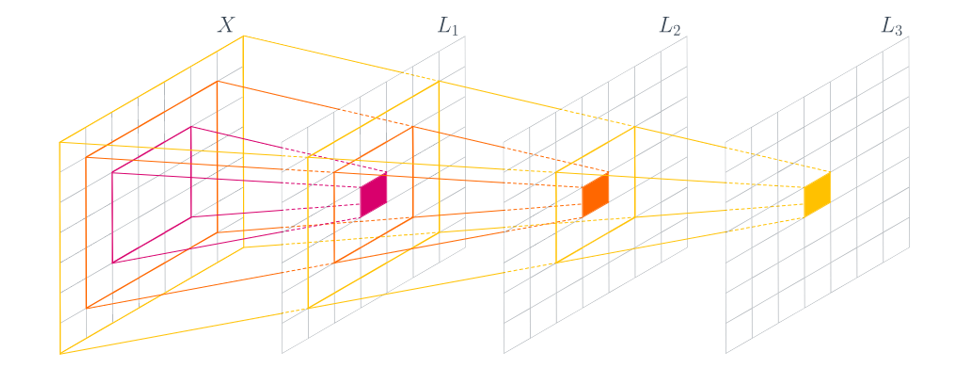


Рисунок 25 – Последовательное применение свертки 3х3

Рассмотрим теперь математическую структуру свертки. Сверткой [15] двух функций называется следующая функция:

Теперь предположим, что вместо функций используются векторы конечной длины, которые будут интерпретироваться как функции, определенные на конечном множестве точек (рисунок 26). Тогда формула принимает вид:



Рисунок 26 – Свертка в одномерном пространстве

В двумерном случае (рисунок 27) формула принимает вид:

где двумерный фильтр имеет размер Например, рассмотрим свертку входа размера с ядром размера для вычисления выхода размера :

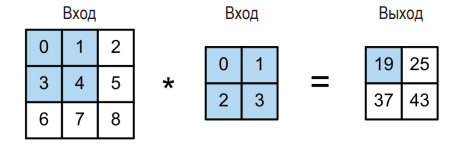


Рисунок 27 – Свертка в двумерном пространстве

Двумерную свертку можно рассматривать как сопоставление с шаблоном, поскольку выход в точке будет велик, если «патч» (область на изображении умножаемая на веса фильтра) с центром в этой точке похож на . Если соответствует ориентированному ребру, то после свертки с  ним на выходной тепловой карте «загорятся» области с такой же ориентацией. В общем случае свертку можно интерпретировать как форму выявления признаков (рисунок 28). Поэтому выход называется картой признаков.

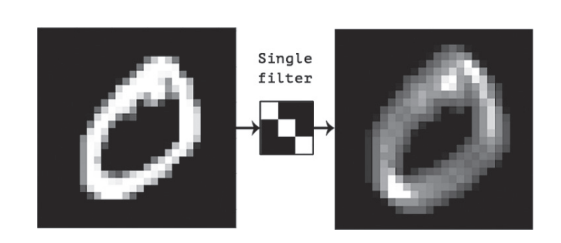


Рисунок 28 – Выявление признаков диагонального характера с помощью свертки

В формуле мы видели, что в  результате свертки изображения с  фильтром получается выход размера . В  общем случае свертка фильтра с  изображением размера порождает выход размера это называется корректной сверткой, потому что мы применяем фильтр только к «корректным» частям входа, не позволяя ему «вылезать за край». Если нужно, чтобы размер выхода был таким же, как размер входа, то можно использовать дополнение нулями, т. е. добавление к изображению рамки, состоящей из нулей. Это называется конгруэнтной сверткой. В общем случае, если вход имеет размер , используется ядро размера и с каждой стороны производится дополнение нулями шириной соответственно и  , то выход будет иметь следующий размер:

Поскольку каждый выходной пиксель является взвешенной комбинацией входов в его рецептивном поле (зависящем от размера фильтра), соседние выходные пиксели будут иметь близкие значения, потому что множества входов, на основе которых они вычисляются, пересекаются. Эту избыточность можно уменьшить (а значит, ускорить вычисления), если пропускать каждый s-й вход. Это называется сверткой с шагом.

До этого входом являлось черно-белое изображение. В общем случае вход может иметь несколько каналов (RGB). Мы можем обобщить понятие свертки на этот случай, определив ядро для каждого входного канала; таким образом, теперь  – трехмерная матрица весов, или тензор. Для вычисления выхода мы свертываем канал c входа с ядром , а затем производим суммирование по каналам.

где s – шаг (для простоты предполагается, что шаг одинаков в обоих направлениях). Каждая матрица весов может выявить один вид признаков. Обычно нужно выявлять признаки разных видов. Это можно сделать, превратив в 4-мерную матрицу весов. Фильтр для выявления признаков типа d во входном канале c хранится в подматрице . Обобщим определение свертки на этот случай следующим образом (рисунок 29):

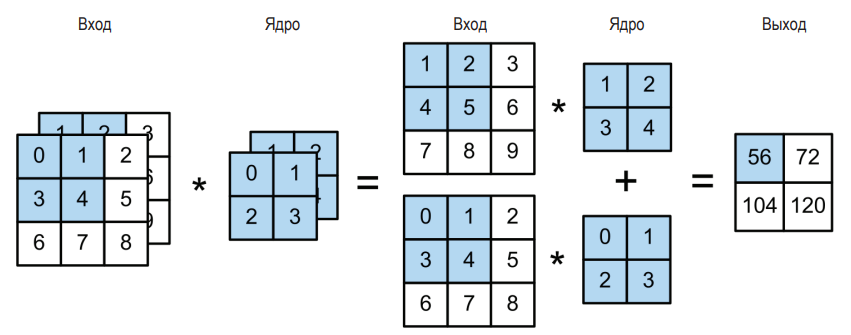


Рисунок 29 – Применение свертки с несколькими слоями и фильтрами

Свертка сохраняет информацию о положении входных признаков (с учетом уменьшенной разрешающей способности). Это свойство называется эквивариантностью. В некоторых случаях нам нужна инвариантность относительно местоположения. Например, в процессе классификации изображений нам иногда нужно только знать, присутствует ли некоторый объект (скажем, лицо) где-нибудь в изображении. Простой способ решить эту задачу называется max-пулингом (рисунок 30) и заключается в вычислении максимума поданных на вход значений. Альтернатива – использовать пулинг усреднением, когда максимум заменяется средним. В любом случае отклик выходного нейрона один и тот же независимо от того, в каком месте его рецептивного поля находится входной паттерн. (Отметим, что пулинг применяется к каждому каналу независимо)

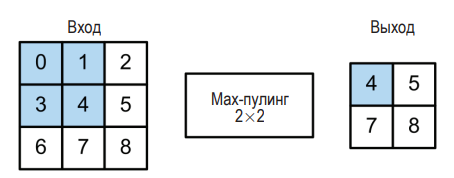


Рисунок 30 – Применение к изображению функции Max-пулинг

Таким образом были рассмотрены практически все необходимые составляющие сети ResNet34 за исключением одной. Модель ResNet34 состоит из 34-х слоев, несмотря на использование relu и batch normalization, градиенты все равно затухают и на первых слоях почти нулевые, то есть есть недостатки эффективности современных алгоритмов обучения. Пока нет других более подходящих методов обучения, авторы архитектуры resnet предлагают интересное решение данной проблемы – «остаточные соединения» (residual connection) [16]. Сеть состоит из блоков, со стандартными слоями, идущими друг за другом, обычно включающие также слой нормализации, но у этого блока есть «обходной путь», который напрямую связывает вход с выходом. В результате, выходное значение блока формируется, как:

Предположим этот блок должен аппроксимировать некоторую функцию  (то есть, описать некоторые характеристики). Другими словами, выходное значение описывается функцией:

Автоматически получается, что:

То есть слои, составляющие функцию  должны аппроксимировать не всю функцию  целиком, а лишь ее остаток. Отсюда и пошло название таких моделей – остаточные сети (residual networks) [17] (рисунок 31). А блок, соответственно, называют остаточным блоком. В таком блоке достаточно обучить сеть воспроизводить не весь сигнал целиком, а лишь отличия входных значений для получения требуемой функции.

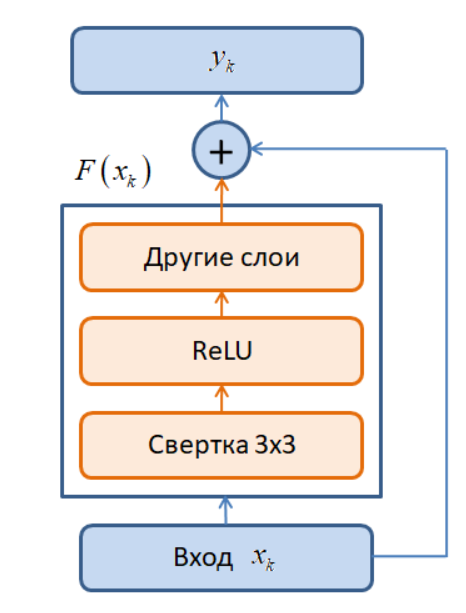


Рисунок 31 – Остаточная сеть

Как показала практика, обучить «остаточную» функцию гораздо проще, чем исходную. Это также связано с тем, что градиент для остаточного блока принимает вид:

Благодаря единичке, которая появляется за счет «обходного» пути, градиент не будет затухать при распространении от последних слоев к начальным и сеть, сколь большой она не была бы, будет обучаться равномерно по всем слоям.

Архитектура ResNet34 (рисунок 32): состоит из 34 слоев сверточных нейронных сетей, образующих блоки с «остаточными соединениями», после каждой свертки изображение проходит слой нормализации и нелинейную функцию активации ReLU (max{0, x}).

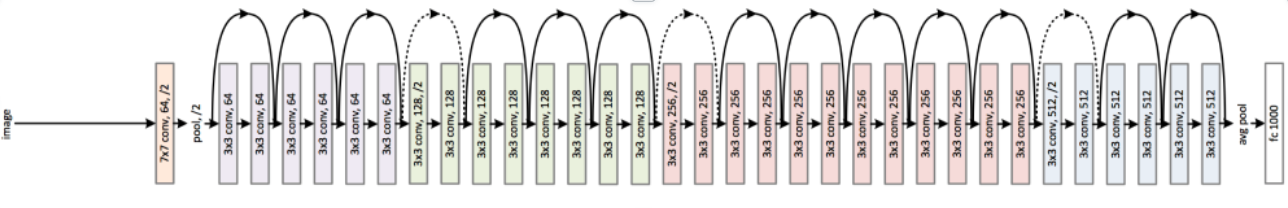


Рисунок 32 – Модель ResNet34

## 2.6 ArcFace

Логитовый слой – последний слой нейронной сети задач классификации, принимающий действительные значения в диапазоне (-∞, +∞).

Функция «софтмакс» (softmax) превращает логиты в вероятности. Функция выводит в качестве результата вектор, представляющий распределения вероятностей списка потенциальных результатов.

Когда логиты отрицательны или обладают разными знаками, правильной нормализации выполнить не удастся, поэтому возведение в степень решает эту проблему, а выбор экспоненты упрощает дальнейшее вычисление функции потерь.

Но функция потерь softmax не обеспечивает более высокого сходства для выборок внутри класса и разнообразия для выборок между классами, поэтому были предложены другие функции, например, ArcFace [18] (рисунок 34).

Как было уже сказано, наиболее популярная классификационная функция «софтмакс»:

где обозначает признак -го образца, принадлежащий -му классу. обозначает -й столбец веса — член смещения, а номер класса равен.

Для простоты смещение фиксируется . Затем преобразуется логит: , где — угол между весом и элементом . Вес фиксируется . Также признак встраивания масштабируется до . Шаг нормализации признаков и весов делает прогнозы зависящими только от угла между признаком и весом. Таким образом, изученные функции встраивания распределяются по гиперсфере радиусом .

Добавляется угловой запас между и , чтобы одновременно повысить компактность внутри класса и несоответствие между классами.

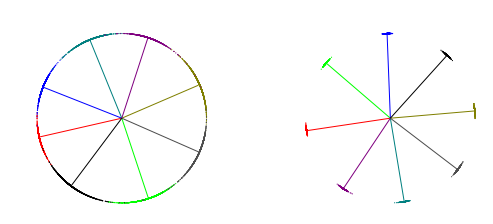


Рисунок 33 – Расположение классов на окружности после применения функции «софтмакс» и ArcFace

Изначально предполагалось, что данные чистые и без шума, однако это не так, особенно когда набор данных имеет большой масштаб. Что бы сделать потери «софтмакс» на основе маржи устойчивыми к шуму, предлагается идея использования субцентров для каждого класса, которая значительно повысит надежность ArcFace (рисунок 36).

Устанавливаеся K субцентров [19] для каждого класса (обучающие выборки должны быть близки только к одному любому из них). На основе 2-го шага нормализации для признака встраивания и для всех субцентров получаются оценки сходства по подклассам путем матричного умножения . Затем берется максимальное значение (maxpooling) по каждому классу для оценки сходства по субцентрам , чтобы получить оценку сходства по классам . На следующих шагах вычисляется угол между векторами, к нему прибавляется угловой запас , считается косинусное расстояние, масштабируется на окружность с радиусом и считается обычная функция «софтмакс». Получив распределение вероятностей на выходе, считается функция потерь кросс-энтропия.

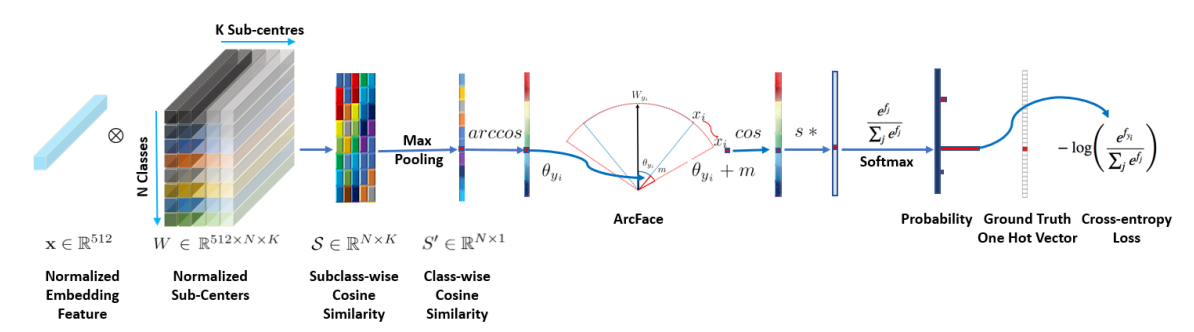


Рисунок 35 – Архитектура ArcFace

## 2.7 Кросс-энтропия

Кросс-энтропия – это [функция потерь](https://dzen.ru/away?to=https%3A%2F%2Fwww.helenkapatsa.ru%2Ffunktsiia-potieri%2F), которая используется для количественной оценки разницы между двумя распределениями вероятностей. Позволяет оценить насколько хорошо функционирует модель классификации в машинном обучении. Принимает значения в диапазоне [0, 1], где 0 – «идеальная модель».

где - количество классов, – истинная вероятность класса , – предсказанная моделью вероятность класса .

Процесс обучения модели подразумевает использование градиентных оптимизационных методов, позволяющих приблизить значения функции потерь к нулю.

# 3 ПРОГРАММНАЯ ЧАСТЬ

## 3.1. Данные

Данные были предоставлены компанией Sarawan (рисунок 36), собранные с различных интернет-площадок, предлагающих в основном продукты питания и товары личной гигиены. По каждому товару имеется следующая информация:

Название. Причем большинство продуктов содержит в названии не только наименование, но и численные характеристики (например, «Творог Авида, 180г 15%»).

Изображение. Изображения хранятся в отдельной папке, в таблице содержатся лишь их имена, поэтому в процессе работы с данными также прописан модуль, который подгружает изображения нужного размера.

Атрибуты. Параметры товаров категориального и численного характера, представленные в виде python-словаря (например, [«бренд» : «Авида», «вес» : 180г, «жирность» : 15%, «упаковка» : «пластик», «страна производства» : «Россия»])

Категории. Все продукты можно разделить на большие классы (кат.1) и более мелкие (кат.2 и самая маленькая кат.3). У многих товаров отсутствует категория 3, поэтому было принято решение не использовать ее в процессе обучения.

Всего обработанные данные содержат информацию о 31786 товарах.

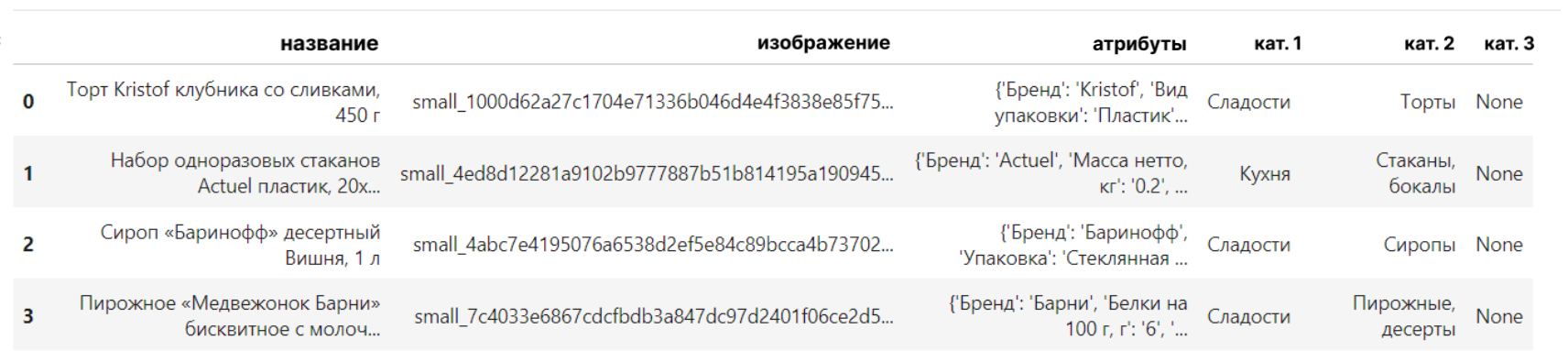


Рисунок 36 – Пример данных предоставленных компанией Sarawan

Для обучения модели машинного обучения было необходимо предобработать текст. Для этого был написан модуль, который вырезает знаки препинания, приводит текст к единому регистру и к лемме (нормальной словарной форме, что решает проблему окончаний в русском языке). Также из названий удаляется информация, содержащаяся в атрибутах товара, оставляя лишь наименование и бренд, аналогично из атрибутов удаляется бренд и параметры, содержащие большое количество пропусков. Изображения приводятся к единому размеру.

## 3.2. Модель

В разделе анализа литературы были рассмотрены существующие идеи и разработки в открытом доступе, там же были рассмотрены их недостатки и преимущества. После анализа было принято решение остановиться на модели, предложенной компанией OZON.

Архитектура OZON (рисунок 37) состоит из трех моделей, на вход каждой из которых подаются данные о товаре. ResNet34 принимает изображения, два Rubert-Tiny - названия и атрибуты. На выходе получаются три вектора. Далее каждый из них используется для вычисления кросс-энтропии по высокоуровневой категории 1 (), и применяется обычная функция «софтмакс», а также они объединяются и идут в следующий линейный слой, после которого используется уже функция активации ArcFace. На выходе последнего слоя считается уже кросс-энтропия по категории 3 (). Для обучения последнего линейного слоя используется функция, состоящая из четырех функций потерь, вычисленных на выходе каждой модели, и посчитанная по категории 3:

где a = , - эпоха обучения. То есть сначала больший вес в значение функции потерь будут вкладывать кросс-энтропии, посчитанные на более крупной категории 1 (), а с увеличением эпох – на более мелкой категории 3 ().

После обучения моделей с помощью градиентов (использовался оптимизатор Адам [20]), считается косинусное сходство векторов товаров на выходе последнего слоя. Опытным путем подбирается порог, который определяет идентичность товаров.

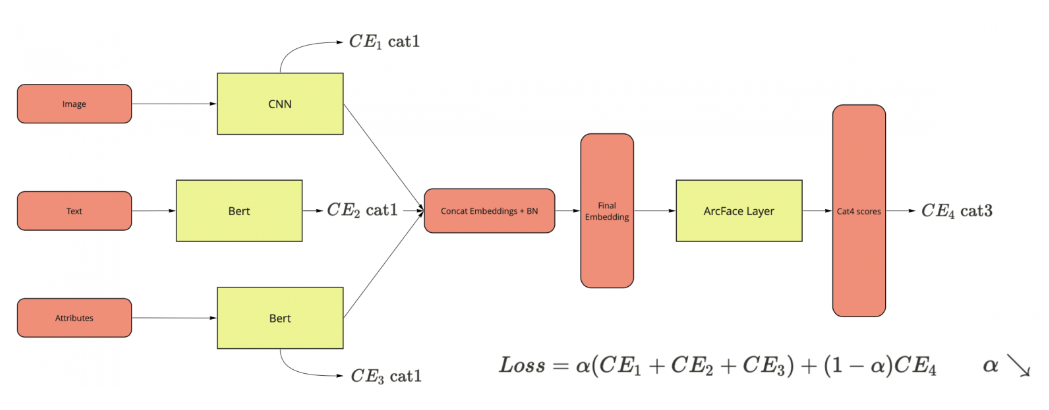


Рисунок 37 – Архитектура модели компании OZON

Для обучения компания OZON использовала около 5 миллионов товаров с 1300 категориями кат.3 и минимум 500 товаров на категорию.

После реализации данной модели на собственных данных были получены неплохие результаты: модель определяла идентичные продукты в 80% случаях. Проблемы возникали только в сложных ситуациях, когда два продукта имели абсолютно одинаковые данные за исключением лишь различий в одном из атрибутов. Например, два одинаковых карандаша для губ, отличающиеся лишь номером оттенка, а это весьма важный параметр.

Тогда пришлось изменить архитектуру следующим образом (рисунок 38). Было принято решение обучить «атрибутную» модель отдельно и подобрать для ее выходных векторов свой собственный порог. Так на первом шаге с помощью порога 1 отсеиваются разные продукты на основе лишь информации об изображениях и названиях товаров, а затем продукты, прошедшие отбор, проходят сортировку уже по порогу 2 на основе атрибутов. Это помогает определить более четкую грань для товаров, которые схожи практически во всем за исключением самых мелких деталей.

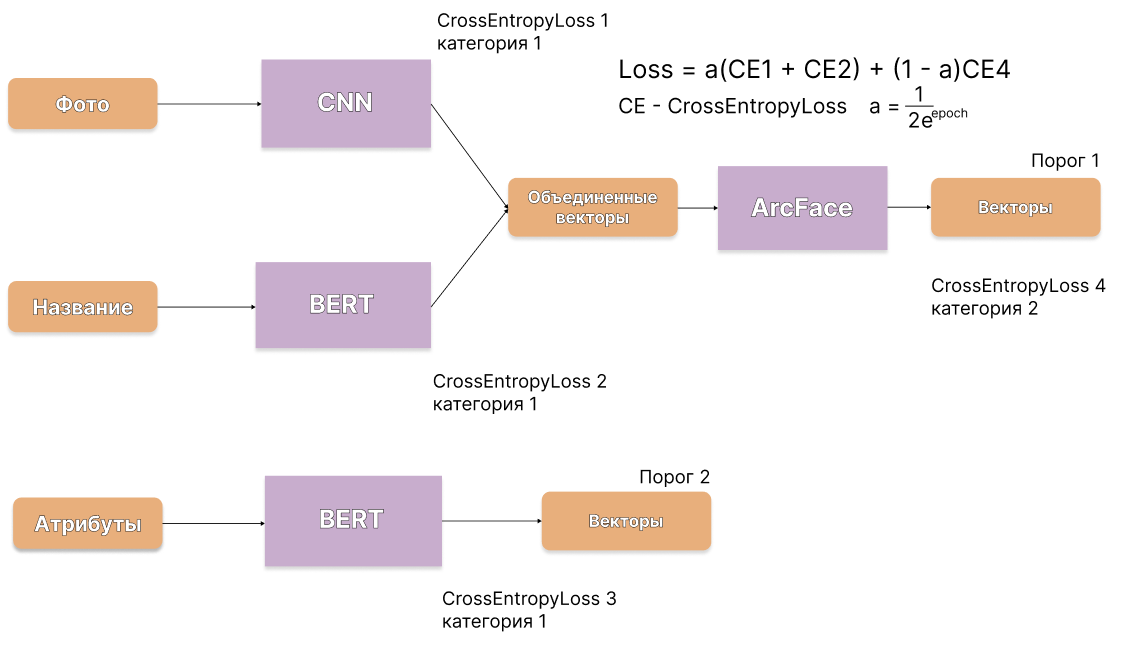


Рисунок 38 – Новая архитектура модели поиска идентичных продуктов для сайта сравнения цен

## 3.3 Программная реализация

В данном разделе описана реализация программного обеспечения, направленного на поиск идентичных товаров в электронной коммерции с использованием моделей машинного обучения. Представленный код включает в себя подготовку данных, создание кастомного датасета, реализацию моделей, обучение и предсказание схожести товаров. Ключевые файлы кода включают dataset.py, models.py и pipeline.py.

В файле dataset.py описан класс CustomMatchingDataset (рисунок 39), который отвечает за подготовку данных. Класс создаёт кастомный датасет для задач машинного обучения. При инициализации происходит кодирование категориальных признаков с помощью LabelEncoder [21]. Данные преобразуются и готовятся для подачи в модели. Так же написан большой модуль по предобработке данных, который можно посмотреть в приложении.



Рисунок 39 - класс CustomMatchingDataset

В файле models.py представлены различные модели, использованные для создания векторов (embeddings) товаров, и некоторые вспомогательные функции (например, объединение векторов и сложная функция на основе четырех кросс-энтропий)

ArcMarginProduct [22] (рисунок 40) - модуль, реализующий слой ArcFace, позволяющий учесть схожесть объектов внутри класса и различия в разных, что улучшает свойства векторов.

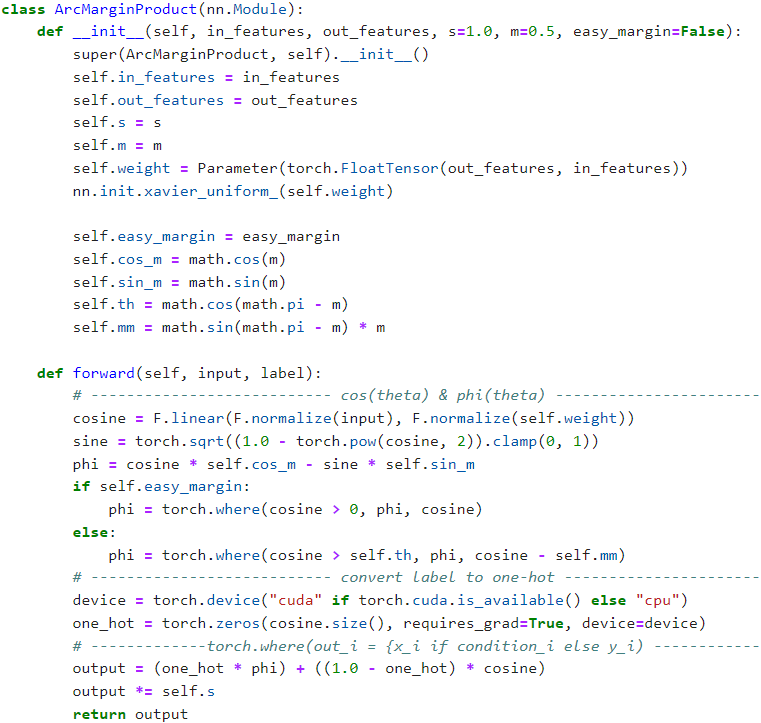


Рисунок 40 – Класс ArcMarginProduct

NameBert [23] (рисунок 41) и AttributeBert (рисунок 42) – модели Rubert-Tiny, предназначенные для построения векторов текстовых данных. Все слои моделей замораживаются, кроме последнего, который и участвует в процессе обучения.

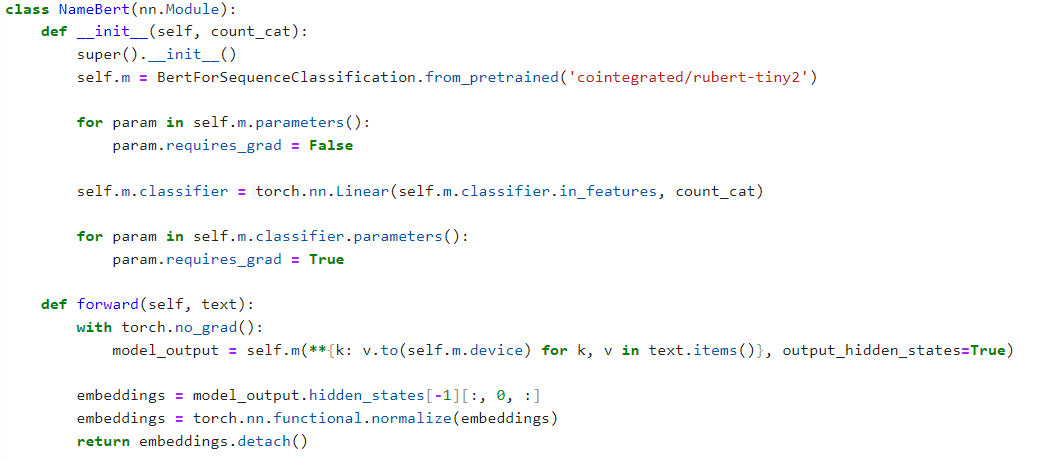


Рисунок 41 – Класс NameBert

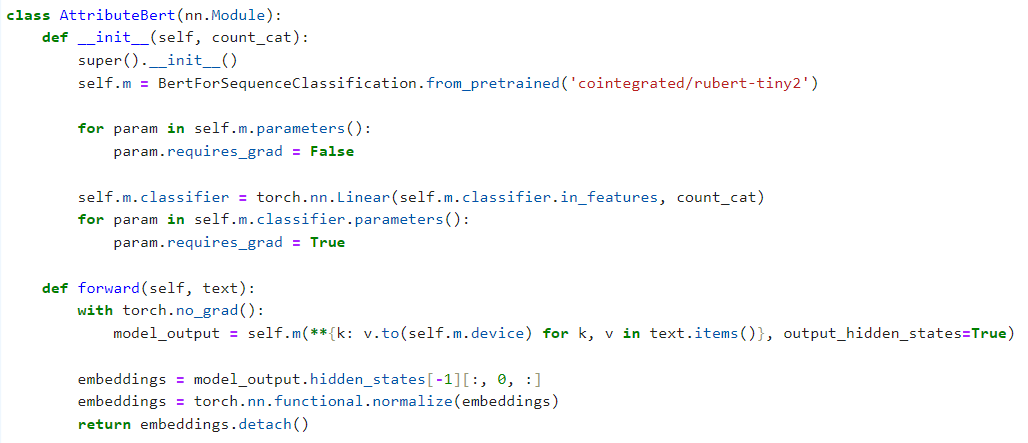


Рисунок 42 - AttributeBert

CNN (рисунок 43) – модель ResNet34 [24] для извлечения векторов изображений товаров.

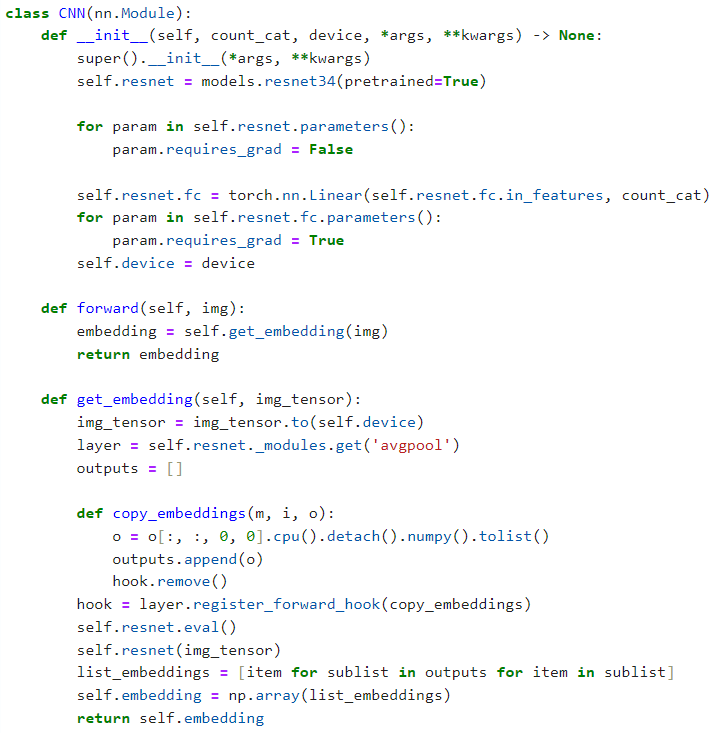


Рисунок 43 – Класс CNN

Файл pipeline.py отвечает за создание пайплайна, обучение моделей и предсказание схожих товаров.

Pipeline - основной класс, управляющий процессом обучения и предсказания. В методе train происходит обучение моделей на основе переданных данных, а в методе predict создаются эмбеддинги для новых данных и осуществляется поиск схожих товаров. Данный модуль достаточно большой, поэтому ознакомиться с ним можно в приложении.

Для написания кода использовался язык программирования python и его библиотеки: torch, torchvision, sklearn, matplotlib, seaborn, pandas, numpy, transformers, math, statistics, tqdm и другие. Текстовые модели обучались на оптимизаторе Adam, изображения на SGD. Обучение проходило 25 эпох, так как после данного числа значения функции потерь начинают расти (рисунок 44).

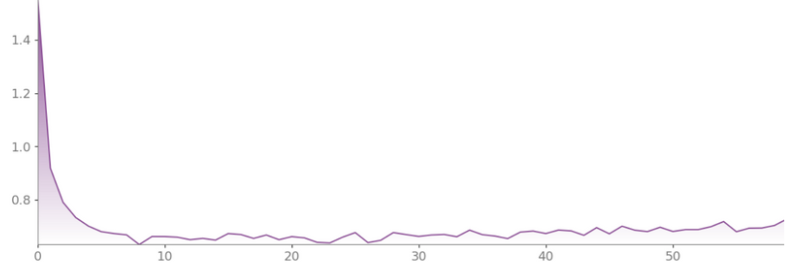


Рисунок 44 – График кросс-энтропии 60 эпох

## 3.4. Результаты

Ниже можно увидеть обучение каждой из моделей и значения функции потерь кросс-энтропии на каждой эпохе на валидационных данных (рисунок 45, 46, 47, 48).

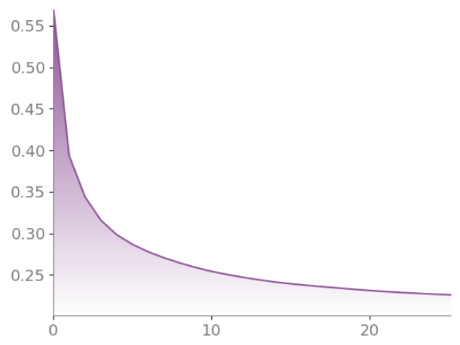


Рисунок 45 – График кросс-энтропии ResNet34 для изображений товара

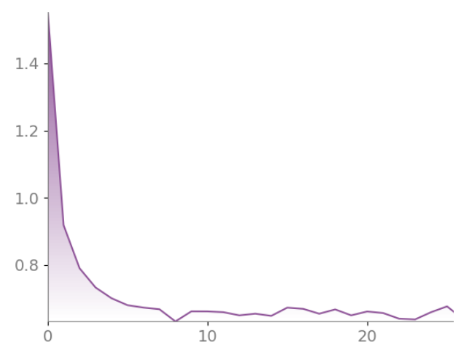


Рисунок 46 – График кросс-энтропии Rubert-Tiny для названий товара

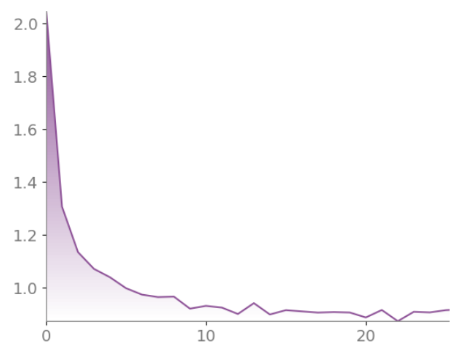


Рисунок 47 – График кросс-энтропии Rubert-Tiny для атрибутов товара

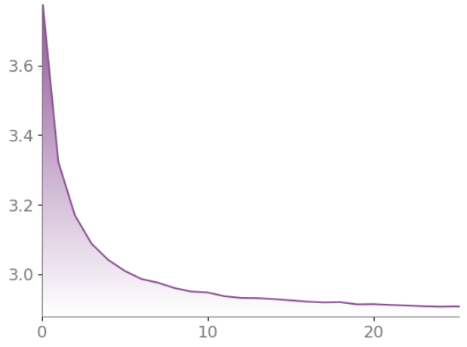


Рисунок 48 – График кросс-энтропии линейного слоя ArcFace для общего вектора товара

После обучения моделей проанализировав полученные коэффициенты сходства, были подобраны соответствующие пороги на первом этапе (на основе изображений и названий) – 0,65 и на втором этапе (на основе атрибутов) - 0,999.

До создания моделей машинного обучения на сайте сравнения цен Sarawan идентичность продуктов определялась обычным сравниванием символов названий продуктов. Для того, чтобы определить превосходство моделей, было проведено сравнение товаров двумя способами: исходным алгоритмом и с помощью новой модели. В итоге были получены следующие результаты: обычный поиск сравнивания названий по символам обнаружил 3953 пар (рисунок 49 и рисунок 50), в то же время как модель машинного обучения смогла определить их все и еще предоставила 506 идентичных товаров (рисунок 51), среди которых всего 11 оказались ошибочными.



Рисунок 49 – Пример полностью идентичных товаров, которые нашли и алгоритм, и модель (Название продукта: «Соус Uni Dan брусничный для вторых обеденных блюд, 270мл»)



Рисунок 50 – Пример полностью идентичных товаров, которые нашли и алгоритм, и модель (Название продукта: «Горбуша + скумбрия» Европром рубленая с чесноком, 180г)



Рисунок 51 – Пример товаров, когда модель верно определила идентичность товаров, а алгоритм ошибся (Название товара отличается лишь буквой «ё», название: «Горбуша + скумбрия» Европром рубленая с чесноком, 180г)

В дальнейшем можно улучшить показатели модели следующими способами:

1. Увеличить количество товаров
2. Восполнить пропущенную информацию в данных
3. Предварительно разметить данные парами (идентичные/разные)
4. Взять более сложные архитектуры BERT (увеличит время поиска, но повысит качество результата)

# ЗАКЛЮЧЕНИЕ

В данной работе была рассмотрена актуальная проблема поиска идентичных товаров в электронной коммерции, а также определена основная цель работы. Для ее достижения были проанализированы существующие решения в сфере «матчинга» продуктов, выбрана подходящая архитектура на основе возможностей компании, а также были обработаны данные, и в полной мере реализован программный код с использованием современных моделей машинного обучения. В результате были получены графики, отображающие рост точности предсказаний, а также были приведены результаты сравнения новой архитектуры со старым решением, уже существующем на сайте сравнения цен. На основе полученных результатов можно сделать вывод, что разработанная модель машинного обучения отлично справилась с поставленной задачей, и превзошла показатели предыдущего алгоритма. Подводя итоги, можно с точностью утверждать, что поставленная цель достигнута, а задачи решены, и разработанная модель машинного обучения нашла свое практическое применение в сфере электронной коммерции. Дальнейшие исследования могут быть направлены на улучшение точности предсказаний реализованной математической модели.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. [Szymon Łukasik](https://www.researchgate.net/profile/Szymon-Lukasik?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InB1YmxpY2F0aW9uIn19), Text-Based Product Matching with Incomplete and Inaccurate Items Descriptions. – Текст: электронный // Research.net: [сайт]. – 2021. – URL: https://www.researchgate.net/publication/352318662\_Text-Based\_Product\_Matching\_with\_Incomplete\_and\_Inconsistent\_Items\_Descriptions (дата обращения: 09.10.2023)
2. [Jeremy Foxcroft](https://www.researchgate.net/scientific-contributions/Jeremy-Foxcroft-2148318352?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InB1YmxpY2F0aW9uIn19), Product Matching Lessons and Recommendations from a Real World Application. – Текст: электронный // ResearchGate.net: [сайт]. – 2021. – URL: https://www.researchgate.net/publication/352726054\_Product\_Matching\_Lessons\_and\_Recommendations\_from\_a\_Real\_World\_Application (дата обращения: 11.10.2023)
3. [Ajinkya More](https://medium.com/@ajinkya.more?source=post_page-----4f19b6aebaca--------------------------------), Product Matching in eCommerce using deep learning. – Текст: электронный // medium.com: [сайт]. – 2017. – URL: https://medium.com/walmartglobaltech/product-matching-in-ecommerce-4f19b6aebaca (дата обращения: 12.10.2023)
4. Data Science Team, Обзор ResNet и его вариантов. – Текст: электронный // datascience.eu: [сайт]. – 2020. – URL: https://datascience.eu/ru/машинное-обучение/обзор-resnet-и-его-вариантов/ (дата обращения: 15.10.2023)
5. Компания OZON, Векторное представление товаров Prod2Vec: как мы улучшили матчинг и избавились от кучи эмбеддингов. – Текст: электронный // habr.com: [сайт]. – 2022. – URL: https://habr.com/ru/companies/ozontech/articles/648231/ (дата обращения: 20.10.2023)
6. Alberto Rizzoli, Cross Entropy Loss: Intro, Applications, Code. – Текст: электронный // medium.com: [сайт]. – 2023. – URL: https://medium.com/v7-labs/cross-entropy-loss-intro-applications-code-1225f301d10f (дата обращения: 22.10.2023)
7. Softmax в машинном обучении простыми словами. – Текст: электронный // neurohive.io: [сайт]. – 2021. – URL: https://dzen.ru/a/YR-pK4RFbzzljUZF (дата обращения: 25.10.2023)
8. ResNet - революция глубокого обучения. Исчезающие и взрывающиеся градиенты. – Текст: электронный // proproprogs.ru: [сайт]. – 2023. – URL: https://proproprogs.ru/tensorflow/keras-resnet-revolyuciya-glubokogo-obucheniya-ischezayushchie-vzryvayushchiesya-gradienty (дата обращения: 11.11.2023)
9. Давид Дале, Маленький и быстрый BERT для русского языка. – Текст: электронный // habr.com: [сайт]. – 2021. – URL: https://habr.com/ru/articles/562064/ (дата обращения: 15.11.2023)
10. [Felipe Almeida](https://arxiv.org/search/cs?searchtype=author&query=Almeida,+F), Word Embeddings: A Survey. – Текст: электронный // arXiv.org: [сайт]. – 2023. – URL: https://arxiv.org/abs/1901.09069 (дата обращения: 17.11.2023)
11. Ashish Vaswani, Attention Is All You Need. – Текст: электронный // arXiv.org: [сайт]. – 2023. – URL: https://arxiv.org/pdf/1706.03762 (дата обращения: 10.12.2023)
12. Jay Alammar, Transformer в картинках. – Текст: электронный // habr.com: [сайт]. – 2020. – URL: https://habr.com/ru/articles/486358/ (дата обращения: 22.12.2023)
13. Alina Kolesnikova, Knowledge Distillation of Russian Language Models with Reduction of Vocabulary. – Текст: электронный // arXiv.org: [сайт]. – 2022. – URL: https://arxiv.org/abs/2205.02340 (дата обращения: 10.01.2024)
14. Иванов Григорий, Сверточные нейронные сети. – 2023. – URL: https://education.yandex.ru/handbook/ml/article/svyortochnye-nejroseti (дата обращения: 15.01.2024)
15. Kevin P. Murphy Machine Learning: A Probabilistic Perspective / P. Kevin Murphy. - The MIT Press Cambridge, MA. Copyright, 2021. – 939 c.
16. Kaiming He, Deep Residual Learning for Image Recognition. – Текст: электронный // arXiv: [сайт]. – 2015. – URL: https://arxiv.org/pdf/1512.03385 (дата обращения: 18.01.2024)
17. Станислав Литвинов, ResNet (34, 50, 101): «остаточные» CNN для классификации изображений. – Текст: электронный // neurohive.io: [сайт]. – 2019. – URL: https://neurohive.io/ru/vidy-nejrosetej/resnet-34-50-101/ (дата обращения: 15.02.2024)
18. Jiankang Deng, ArcFace: Additive Angular Margin Loss for Deep Face Recognition. – Текст: электронный // arXiv: [сайт]. – 2019. – URL: https://arxiv.org/abs/1801.07698v3 (дата обращения: 20.02.2024)
19. Hao Wang, CosFace: Large Margin Cosine Loss for Deep Face Recognition. – Текст: электронный // arXiv: [сайт]. – 2018. – URL: https://arxiv.org/pdf/1801.09414 (дата обращения: 05.03.2024)
20. [Diederik P. Kingma](https://arxiv.org/search/cs?searchtype=author&query=Kingma,+D+P), Adam: A Method for Stochastic Optimization. – Текст: электронный // arXiv: [сайт]. – 2017. – URL: https://arxiv.org/abs/1412.6980 (дата обращения: 09.03.2024)
21. [Deval Shah](https://arxiv.org/search/cs?searchtype=author&query=Shah,+D), Label Encoding for Regression Networks. – Текст: электронный // arXiv: [сайт]. – 2022. – URL: https://arxiv.org/abs/2212.01927 (дата обращения: 15.03.2024)
22. Trevor Hastie The Elements of Statistical Learning: Data Mining. Inference, and Prediction / Trevor Hastie. – Springer. – 745 c.
23. Вьюгин В. В. Математические основы теории машинного обучения и прогнозирования / В. В. Вьюгин. – М.: 2013. - 387 с.
24. Ричард С. Саттон Обучение с подкреплением введение / С. Ричард Саттон. - М.: 2020. - 551 с.

# ПРИЛОЖЕНИЕ А

Листинг А.1 – программная реализация модели машинного обучения для поиска идентичных товаров

dataset.py

from sklearn.preprocessing import LabelEncoder

from torchvision.io import read\_image

from torch.utils.data import Dataset

from torchvision import transforms

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

import os

class CustomMatchingDataset(Dataset):

def \_\_init\_\_(self, data, img\_dir):

self.le\_cat1 = LabelEncoder()

self.le\_cat2 = LabelEncoder()

data['cat1\_enc'] = self.le\_cat1.fit\_transform(data['cat1'].values)

data['cat2\_enc'] = self.le\_cat2.fit\_transform(data['cat2'].values)

self.data = data

self.images = data['image']

self.img\_dir = img\_dir

self.transform = transforms.Compose([

transforms.ToPILImage(),

transforms.Resize((380, 380)),

transforms.ToTensor()])

self.names = data['name\_prepared']

self.attributes = data['attributes\_prepared']

self.labels\_cat1 = data['cat1\_enc']

self.labels\_cat2 = data['cat2\_enc']

def \_\_len\_\_(self):

return len(self.images)

def \_\_getitem\_\_(self, idx):

label\_cat1 = self.labels\_cat1.iloc[idx]

label\_cat2 = self.labels\_cat2.iloc[idx]

img\_path = os.path.join(self.img\_dir, self.images.iloc[idx])

image = read\_image(img\_path)

if self.transform:

image = self.transform(image).float()

name = self.names.iloc[idx]

attribute = self.attributes.iloc[idx]

return name, attribute, image, idx, label\_cat1, label\_cat2

models.py

from transformers import BertForSequenceClassification

from sklearn.metrics import top\_k\_accuracy\_score

import torchvision.models as models

from torch.nn import BatchNorm1d

import torch.nn.functional as F

from torch.nn import Parameter

import torch.nn as nn

import numpy as np

import torch

import math

class ArcMarginProduct(nn.Module):

def \_\_init\_\_(self, in\_features, out\_features, s=30.0, m=0.50, easy\_margin=False):

super(ArcMarginProduct, self).\_\_init\_\_()

self.in\_features = in\_features

self.out\_features = out\_features

self.s = s

self.m = m

self.weight = Parameter(torch.FloatTensor(out\_features, in\_features))

nn.init.xavier\_uniform\_(self.weight)

self.easy\_margin = easy\_margin

self.cos\_m = math.cos(m)

self.sin\_m = math.sin(m)

self.th = math.cos(math.pi - m)

self.mm = math.sin(math.pi - m) \* m

def forward(self, input, label):

# --------------------------- cos(theta) & phi(theta) ---------------------------

cosine = F.linear(F.normalize(input), F.normalize(self.weight))

sine = torch.sqrt((1.0 - torch.pow(cosine, 2)).clamp(0, 1))

phi = cosine \* self.cos\_m - sine \* self.sin\_m

if self.easy\_margin:

phi = torch.where(cosine > 0, phi, cosine)

else:

phi = torch.where(cosine > self.th, phi, cosine - self.mm)

# --------------------------- convert label to one-hot ---------------------------

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

one\_hot = torch.zeros(cosine.size(), requires\_grad=True, device=device)

# -------------torch.where(out\_i = {x\_i if condition\_i else y\_i) -------------

output = (one\_hot \* phi) + ((1.0 - one\_hot) \* cosine) # you can use torch.where if your torch.\_\_version\_\_ is 0.4

output \*= self.s

return output

class NameBert(nn.Module):

def \_\_init\_\_(self, count\_cat):

super().\_\_init\_\_()

self.m = BertForSequenceClassification.from\_pretrained('cointegrated/rubert-tiny2')

for param in self.m.parameters():

param.requires\_grad = False

self.m.classifier = torch.nn.Linear(self.m.classifier.in\_features, count\_cat)

for param in self.m.classifier.parameters():

param.requires\_grad = True

def forward(self, text):

# на вход подаётся уже токенизированный текст

with torch.no\_grad():

model\_output = self.m(\*\*{k: v.to(self.m.device) for k, v in text.items()}, output\_hidden\_states=True)

embeddings = model\_output.hidden\_states[-1][:, 0, :]

embeddings = torch.nn.functional.normalize(embeddings)

return embeddings.detach()

class AttributeBert(nn.Module):

def \_\_init\_\_(self, count\_cat):

super().\_\_init\_\_()

self.m = BertForSequenceClassification.from\_pretrained('cointegrated/rubert-tiny2')

for param in self.m.parameters():

param.requires\_grad = False

self.m.classifier = torch.nn.Linear(self.m.classifier.in\_features, count\_cat)

for param in self.m.classifier.parameters():

param.requires\_grad = True

def forward(self, text):

with torch.no\_grad():

model\_output = self.m(\*\*{k: v.to(self.m.device) for k, v in text.items()}, output\_hidden\_states=True)

embeddings = model\_output.hidden\_states[-1][:, 0, :]

embeddings = torch.nn.functional.normalize(embeddings)

return embeddings.detach()

class CNN(nn.Module):

def \_\_init\_\_(self, count\_cat, device, \*args, \*\*kwargs) -> None:

super().\_\_init\_\_(\*args, \*\*kwargs)

self.resnet = models.resnet34(pretrained=True)

for param in self.resnet.parameters():

param.requires\_grad = False

self.resnet.fc = torch.nn.Linear(self.resnet.fc.in\_features, count\_cat)

for param in self.resnet.fc.parameters():

param.requires\_grad = True

self.device = device

def forward(self, img):

embedding = self.get\_embedding(img)

return embedding

def get\_embedding(self, img\_tensor):

img\_tensor = img\_tensor.to(self.device)

layer = self.resnet.\_modules.get('avgpool')

outputs = []

def copy\_embeddings(m, i, o):

o = o[:, :, 0, 0].cpu().detach().numpy().tolist()

outputs.append(o)

hook.remove()

hook = layer.register\_forward\_hook(copy\_embeddings)

self.resnet.eval()

self.resnet(img\_tensor)

list\_embeddings = [item for sublist in outputs for item in sublist]

self.embedding = np.array(list\_embeddings)

return self.embedding

class Embedding(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def forward(self, name\_bert, cnn, name, img, num\_features, device):

name = name\_bert(name)

img = torch.from\_numpy(cnn(img)).to(device)

self.BN = BatchNorm1d(num\_features)

embedding\_raw = torch.cat((name, img), dim=1)

self.BN.to(device, dtype=float)

embedding = self.BN(embedding\_raw.to(device, dtype=float))

return embedding

class CustomLoss(nn.Module):

def \_\_init\_\_(self):

super(CustomLoss, self).\_\_init\_\_()

self.alpha = lambda x: 0.5 / np.exp(x/np.exp(1)) # [0, 0.5]?? or [0, 1]

def forward(self, cnn\_loss, names\_bert\_loss, arc\_loss, epoch):

main\_loss = self.alpha(epoch)\*(cnn\_loss + names\_bert\_loss) + (1 - self.alpha(epoch)) \* arc\_loss

return main\_loss

class Losses:

def \_\_init\_\_(self) -> None:

self.losses = [[], [], [], [], []]

self.losses\_val = [[], [], [], [], []]

self.Loss\_history = [[], [], [], [], []]

self.Loss\_history\_val = [[], [], [], [], []]

self.acc = [[], [], [], [], []]

self.acc5 = [[], [], [], [], []]

self.acc\_val = [[], [], [], [], []]

self.Accuracy\_history = [[], [], [], [], []]

self.Accuracy\_history\_val = [[], [], [], [], []]

def append(self, cnn\_loss, names\_bert\_loss, attr\_bert\_loss, arc\_loss, loss, cnn\_outputs, names\_bert\_outputs, attr\_bert\_outputs, output, labels\_cat1, cat1\_list, labels\_cat2, cat2\_list):

self.losses[0].append(cnn\_loss.item())

self.losses[1].append(names\_bert\_loss.item())

self.losses[2].append(attr\_bert\_loss.item())

cnn\_accuracy1 = top\_k\_accuracy\_score(labels\_cat1.cpu().detach().numpy(), cnn\_outputs.cpu().detach().numpy(), k = 1, labels=cat1\_list)

names\_bert\_accuracy1 = top\_k\_accuracy\_score(labels\_cat1.cpu().detach().numpy(), names\_bert\_outputs.logits.cpu().detach().numpy(), k = 1, labels=cat1\_list)

attr\_bert\_accuracy1 = top\_k\_accuracy\_score(labels\_cat1.cpu().detach().numpy(), attr\_bert\_outputs.logits.cpu().detach().numpy(), k = 1, labels=cat1\_list)

# считаем acc@5 на кат1

cnn\_accuracy5 = top\_k\_accuracy\_score(labels\_cat1.cpu().detach().numpy(), cnn\_outputs.cpu().detach().numpy(), k = 5, labels=cat1\_list)

names\_bert\_accuracy5 = top\_k\_accuracy\_score(labels\_cat1.cpu().detach().numpy(), names\_bert\_outputs.logits.cpu().detach().numpy(), k = 5, labels=cat1\_list)

attr\_bert\_accuracy5 = top\_k\_accuracy\_score(labels\_cat1.cpu().detach().numpy(), attr\_bert\_outputs.logits.cpu().detach().numpy(), k = 5, labels=cat1\_list)

self.acc[0].append(cnn\_accuracy1.item())

self.acc[1].append(names\_bert\_accuracy1.item())

self.acc[2].append(attr\_bert\_accuracy1.item())

self.acc5[0].append(cnn\_accuracy5.item())

self.acc5[1].append(names\_bert\_accuracy5.item())

self.acc5[2].append(attr\_bert\_accuracy5.item())

arc\_accuracy1 = top\_k\_accuracy\_score(labels\_cat2.cpu().detach().numpy(), output.cpu().detach().numpy(), k = 1, labels=cat2\_list)

arc\_accuracy5 = top\_k\_accuracy\_score(labels\_cat2.cpu().detach().numpy(), output.cpu().detach().numpy(), k = 5, labels=cat2\_list)

self.losses[3].append(arc\_loss.item())

self.losses[4].append(loss.item())

self.acc[3].append(arc\_accuracy1.item())

self.acc5[3].append(arc\_accuracy5.item())

pipeline.py

from \_\_future\_\_ import print\_function

from \_\_future\_\_ import division

from sklearn.metrics.pairwise import cosine\_similarity

from transformers import AutoTokenizer

import matplotlib.image as mpimg

import matplotlib.pyplot as plt

import plotly.express as px

import torch.optim as optim

from statistics import mean

from tqdm import tqdm

from umap import UMAP

import pandas as pd

import numpy as np

import zipfile

import dataset

import shutil

import pickle

import models

import torch

import ast

import os

from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter("./log/")

class Pipeline():

DATA = './data/'

MODELS = './models/'

ZIP\_PHOTO\_DIR = './all\_photos.zip'

def \_\_init\_\_(self, data, img\_dir):

self.device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

self.data = data

self.img\_dir = img\_dir

self.data = self.clear\_images()

def train(self,

num\_epoch=60,

momentum=0.9,

batch\_size=10,

learning\_rate=1e-2, s=30, m=0.5, change\_lr=True):

custom\_dataset = dataset.CustomMatchingDataset(data=self.data, img\_dir=self.img\_dir)

data\_loader = torch.utils.data.DataLoader(custom\_dataset, batch\_size=batch\_size, shuffle=False)

cnn = models.CNN(len(custom\_dataset.le\_cat1.classes\_), self.device)

names\_bert = models.NameBert(len(custom\_dataset.le\_cat1.classes\_),)

attr\_bert = models.AttributeBert(len(custom\_dataset.le\_cat1.classes\_),)

emb = models.Embedding()

tokenizer = AutoTokenizer.from\_pretrained("cointegrated/rubert-tiny2")

main\_loss = models.CustomLoss()

loss\_monitor = models.Losses()

final\_embedding\_lenght = cnn.resnet.fc.in\_features + names\_bert.m.classifier.in\_features

metric\_fc = models.ArcMarginProduct(final\_embedding\_lenght, len(custom\_dataset.le\_cat2.classes\_), s=s, m=m) # cat1\_num\_classes or cat2\_num\_classes

cnn\_criterion = torch.nn.CrossEntropyLoss()

names\_bert\_criterion = torch.nn.CrossEntropyLoss()

attr\_bert\_criterion = torch.nn.CrossEntropyLoss()

arcface\_criterion = torch.nn.CrossEntropyLoss()

# оптимизаторы

optimizer1 = optim.SGD(cnn.resnet.fc.parameters(), lr=learning\_rate)

optimizer2 = optim.Adam(names\_bert.m.classifier.parameters(), lr=learning\_rate)

optimizer3 = optim.Adam(attr\_bert.m.classifier.parameters(), lr=learning\_rate)

optimizer = optim.SGD(metric\_fc.parameters(), lr=learning\_rate)

cnn.resnet = cnn.resnet.to(self.device)

names\_bert.m = names\_bert.m.to(self.device)

attr\_bert.m = attr\_bert.m.to(self.device)

emb.to(self.device)

metric\_fc.to(self.device)

n\_batches = round(len(self.data) / batch\_size)

cat1\_list = list(custom\_dataset.data['cat1\_enc'].unique())

cat1\_list.sort()

cat2\_list = list(custom\_dataset.data['cat2\_enc'].unique())

cat2\_list.sort()

for epoch in range(num\_epoch):

cnn.resnet.train()

names\_bert.m.train()

attr\_bert.m.train()

embbb = None

cat1 = None

cat2 = None

names\_store = None

data\_with\_embeddings = None

if change\_lr == True and epoch == num\_epoch // 2:

learning\_rate = 1e-2

for names, attributes, images, index, labels\_cat1, labels\_cat2 in tqdm(data\_loader):

names = tokenizer(names, padding=True, truncation=True, return\_tensors='pt')

attributes = tokenizer(attributes, padding=True, truncation=True, return\_tensors='pt')

names = names.to(self.device)

attributes = attributes.to(self.device)

images = images.to(self.device)

index = index.to(self.device)

labels\_cat1 = labels\_cat1.to(self.device)

labels\_cat2 = labels\_cat2.to(self.device)

optimizer1.zero\_grad(set\_to\_none=True)

optimizer2.zero\_grad(set\_to\_none=True)

optimizer3.zero\_grad(set\_to\_none=True)

optimizer.zero\_grad(set\_to\_none=True)

# получаем предсказания

cnn\_outputs = cnn.resnet(images.to(self.device))

names\_bert\_outputs = names\_bert.m(\*\*{k: v.to(self.device) for k, v in names.to(self.device).items()})

attr\_bert\_outputs = attr\_bert.m(\*\*{k: v.to(self.device) for k, v in attributes.to(self.device).items()})

# считаем лоссы на кат1

cnn\_loss = cnn\_criterion(cnn\_outputs.to(self.device), labels\_cat1.to(self.device))

names\_bert\_loss = names\_bert\_criterion(names\_bert\_outputs.logits.to(self.device), labels\_cat1.to(self.device))

attr\_bert\_loss = attr\_bert\_criterion(attr\_bert\_outputs.logits.to(self.device), labels\_cat1.to(self.device))

cnn\_loss.backward(retain\_graph=True)

names\_bert\_loss.backward(retain\_graph=True)

attr\_bert\_loss.backward(retain\_graph=True)

# Добавление L2-нормы весов модели по слоям

for name, param in cnn.resnet.named\_parameters():

if 'weight' in name:

writer.add\_scalar(f'L2\_norm\_weights/{name}', torch.norm(param, p=2).item(), global\_step=epoch)

# Расчет и добавление L2-нормы градиентов

for name, param in cnn.resnet.named\_parameters():

# print(name)

# print(param.grad)

# print()

if 'weight' in name:

try:

writer.add\_scalar(f'L2\_norm\_gradients/{name}', torch.norm(param.grad, p=2).item(), global\_step=epoch)

except:

pass

# получаем ембеддинг от трёх моделей

embedding = emb(names\_bert, cnn, names.to(self.device), images.to(self.device), final\_embedding\_lenght, self.device)

# получаем предсказание кат2 от АркФейса

output = metric\_fc(embedding.float(), labels\_cat2)

# считаем лосс по кат2

arc\_loss = arcface\_criterion(output, labels\_cat2.to(self.device))

# считаем финальный лосс

loss = main\_loss(cnn\_loss, names\_bert\_loss, arc\_loss, epoch)

arc\_loss.backward(retain\_graph=True)

loss.backward(retain\_graph=True)

loss\_monitor.append(cnn\_loss, names\_bert\_loss, attr\_bert\_loss, arc\_loss, loss,

cnn\_outputs, names\_bert\_outputs, attr\_bert\_outputs, output,

labels\_cat1, cat1\_list,

labels\_cat2, cat2\_list)

optimizer1.step()

optimizer2.step()

optimizer3.step()

optimizer.step()

data = pd.DataFrame(self.data.loc[index.cpu().detach().numpy(), :])

# записываем

if (epoch+1) % 5 == 0 or epoch == 0:

if data\_with\_embeddings is None:

embedding = embedding.cpu().detach().numpy()

emb\_attr = attr\_bert\_outputs.logits.cpu().detach().numpy()

data['emb\_main'] = [embedding[i] for i in range(len(embedding))]

data['emb\_attr'] = [emb\_attr[i] for i in range(len(emb\_attr))]

data\_with\_embeddings = data

else:

embedding = embedding.cpu().detach().numpy()

emb\_attr = attr\_bert\_outputs.logits.cpu().detach().numpy()

data['emb\_main'] = [embedding[i] for i in range(len(embedding))]

data['emb\_attr'] = [emb\_attr[i] for i in range(len(emb\_attr))]

data\_with\_embeddings = pd.concat((data\_with\_embeddings, data), axis=0)

with open('loss.txt', 'a') as f:

f.write(f"Epoch [{epoch}], img\_loss: {mean(loss\_monitor.losses[0][-n\_batches:])}, attr\_loss: {mean(loss\_monitor.losses[2][-n\_batches:])}, names\_bert\_loss: {mean(loss\_monitor.losses[1][-n\_batches:])}, arc\_loss: {mean(loss\_monitor.losses[3][-n\_batches:])}, main\_loss: {mean(loss\_monitor.losses[4][-n\_batches:])}\n")

f.write(f"Epoch [{epoch}], img\_accuracy1: {mean(loss\_monitor.acc[0][-n\_batches:])}, attr\_accuracy1: {mean(loss\_monitor.acc[2][-n\_batches:])}, names\_bert\_accuracy1: {mean(loss\_monitor.acc[1][-n\_batches:])}, arc\_accuracy1: {mean(loss\_monitor.acc[3][-n\_batches:])}\n")

f.write(f"Epoch [{epoch}], img\_accuracy5: {mean(loss\_monitor.acc5[0][-n\_batches:])}, attr\_accuracy5: {mean(loss\_monitor.acc5[2][-n\_batches:])}, names\_bert\_accuracy5: {mean(loss\_monitor.acc5[1][-n\_batches:])}, arc\_accuracy5: {mean(loss\_monitor.acc5[3][-n\_batches:])}\n\n")

if (epoch+1) % 5 == 0 or epoch == 0:

self.\_save\_everything(epoch, data\_with\_embeddings)

torch.save(cnn.resnet.state\_dict(), './models/resnet34.pth')

torch.save(names\_bert.m.state\_dict(), './models/names\_bert.pth')

torch.save(attr\_bert.m.state\_dict(), './models/attr\_bert.pth')

print(f"Epoch [{epoch}], img\_loss: {mean(loss\_monitor.losses[0][-n\_batches:])}, attr\_loss: {mean(loss\_monitor.losses[2][-n\_batches:])}, names\_bert\_loss: {mean(loss\_monitor.losses[1][-n\_batches:])}, arc\_loss: {mean(loss\_monitor.losses[3][-n\_batches:])}, main\_loss: {mean(loss\_monitor.losses[4][-n\_batches:])}\n")

print(f"Epoch [{epoch}], img\_accuracy1: {mean(loss\_monitor.acc[0][-n\_batches:])}, attr\_accuracy1: {mean(loss\_monitor.acc[2][-n\_batches:])}, names\_bert\_accuracy1: {mean(loss\_monitor.acc[1][-n\_batches:])}, arc\_accuracy1: {mean(loss\_monitor.acc[3][-n\_batches:])}\n")

print(f"Epoch [{epoch}], img\_accuracy5: {mean(loss\_monitor.acc5[0][-n\_batches:])}, attr\_accuracy5: {mean(loss\_monitor.acc5[2][-n\_batches:])}, names\_bert\_accuracy5: {mean(loss\_monitor.acc5[1][-n\_batches:])}, arc\_accuracy5: {mean(loss\_monitor.acc5[3][-n\_batches:])}\n\n")

def predict(self, data=None, batch\_size=10, first\_threshold=0.65, last\_threshold=0.99):

if data is None:

data = self.data

custom\_dataset = dataset.CustomMatchingDataset(data=data, img\_dir=self.img\_dir)

data\_loader = torch.utils.data.DataLoader(custom\_dataset, batch\_size=batch\_size, shuffle=False)

cnn = models.CNN(len(custom\_dataset.le\_cat1.classes\_), self.device)

name\_bert = models.NameBert(len(custom\_dataset.le\_cat1.classes\_),)

attr\_bert = models.AttributeBert(len(custom\_dataset.le\_cat1.classes\_),)

emb = models.Embedding()

tokenizer = AutoTokenizer.from\_pretrained("cointegrated/rubert-tiny2")

final\_embedding\_lenght = cnn.resnet.fc.in\_features + name\_bert.m.classifier.in\_features

name\_bert.to(self.device)

attr\_bert.to(self.device)

cnn.to(self.device)

emb.to(self.device)

try:

cnn.resnet.load\_state\_dict(torch.load(self.MODELS + "resnet34.pth"))

name\_bert.m.load\_state\_dict(torch.load(self.MODELS + "names\_bert.pth"))

attr\_bert.m.load\_state\_dict(torch.load(self.MODELS + "attr\_bert.pth"))

except Exception as e:

print(e, f'\n\n Сначала обучите или поместите модели в {self.MODELS}')

return

self.data\_with\_embeddings = None

for name, attribute, image, index, \_, \_ in tqdm(data\_loader):

"""создание ембеддингов"""

torch.cuda.empty\_cache()

name = tokenizer(name, padding=True, truncation=True, return\_tensors='pt')

attribute = tokenizer(attribute, padding=True, truncation=True, return\_tensors='pt')

emb\_main = emb(name\_bert, cnn, name.to(self.device), image.to(self.device), final\_embedding\_lenght, self.device)

attr\_bert\_outputs = attr\_bert.m(\*\*{k: v.to(self.device) for k, v in attribute.to(self.device).items()})

emb\_main = emb\_main.cpu().detach().numpy()

emb\_attr = attr\_bert\_outputs.logits.cpu().detach().numpy()

data = pd.DataFrame(self.data.loc[index.cpu().detach().numpy(), :])

if self.data\_with\_embeddings is None:

data['emb\_main'] = [emb\_main[i] for i in range(len(emb\_main))]

data['emb\_attr'] = [emb\_attr[i] for i in range(len(emb\_attr))]

self.data\_with\_embeddings = data

else:

data['emb\_main'] = [emb\_main[i] for i in range(len(emb\_main))]

data['emb\_attr'] = [emb\_attr[i] for i in range(len(emb\_attr))]

self.data\_with\_embeddings = pd.concat((self.data\_with\_embeddings, data), axis=0)

self.data\_with\_embeddings.to\_json(self.DATA + 'data\_with\_embeddings.json')

similar\_products, count\_in\_cats = self.look\_for\_identical\_products\_main(self.data\_with\_embeddings, first\_threshold)

identical\_products = self.attribute\_identical\_search(similar\_products, last\_threshold)

print('\n\nНайденные идентичные продукты на 1-ом уровне с threshold = ' + str(first\_threshold) + ': ' + str(count\_in\_cats))

print('Найденные идентичные продукты на 2-ом уровне с threshold = ' + str(last\_threshold) + ': ' + str(len(identical\_products)))

return identical\_products

def \_prepare\_data(self, data):

# Подготовка данных в init

tranformer = dataset.DataPreparation(data)

prepared = tranformer.prepare()

pass

def extract\_zip(self, zip\_path):

"метод для извлечения фотографий из архива"

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall('./')

def umap\_vector\_visualization(self, dataframe:pd.DataFrame, show=None, color=None, dot\_size=3, title=None, text=None, write\_to\_html\_path=None):

reducer = UMAP(n\_components=3)

vector = reducer.fit\_transform(dataframe) # понижаем размерность

fig = px.scatter\_3d(x=vector[:, 0],

y=vector[:, 1],

z=vector[:, 2],

color=color,

title=title,

text=text

)

fig.update\_traces(marker=dict(size=dot\_size)) # задаём размер точек

if write\_to\_html\_path:

fig.write\_html(write\_to\_html\_path) # если передан путь к html, файл будет записан в него

if show:

fig.show() # отображение в jupiter notebook

def attribute\_identical\_search(self, identical\_products\_first, threshold\_2):

identical\_products\_second = []

for sim\_main, i, j in identical\_products\_first:

similarity = cosine\_similarity(np.array(self.data\_with\_embeddings['emb\_attr'][i]).reshape(1, -1), np.array(self.data\_with\_embeddings['emb\_attr'][j]).reshape(1, -1))

if similarity > threshold\_2:

identical\_products\_second.append([sim\_main, similarity, i, j])

return identical\_products\_second

def look\_for\_identical\_products\_main(self, data, threshold):

"""поиск идентичных товаров по основным ембеддингам"""

count\_in\_cats = []

similar\_products = []

cats = list(set(data['cat1']))

for cat in tqdm(cats):

count = 0

indexes = data[data['cat1'] == cat].index

emb = pd.DataFrame(np.vstack(data.loc[indexes, 'emb\_main'].values))

emb.set\_index(indexes, inplace=True)

similarity = pd.DataFrame(cosine\_similarity(emb))

similarity.set\_index(indexes, inplace=True)

similarity.columns = similarity.index

for row in indexes:

for col in indexes:

if row != col and [similarity[row][col], col, row] not in similar\_products and similarity[row][col] >= threshold:

similar\_products.append([similarity[row][col], row, col])

count += 1

count\_in\_cats.append([cat, count])

return similar\_products, count\_in\_cats

def \_look\_for\_identical\_products\_main\_optimized\_experimental(self, data, threshold):

count\_in\_cats = []

similar\_products = []

cats = data['cat1'].unique()

for cat in cats:

d = data[data['cat1'] == cat]

emb = np.vstack(d['emb\_main'].values)

similarity = cosine\_similarity(emb)

np.fill\_diagonal(similarity, 0) # диагональные элементы в 0

similar\_indices = np.argwhere(similarity >= threshold)

similar\_indices = [(row, col) for row, col in similar\_indices if row < col]

similar\_products.extend([(row, col) for row, col in similar\_indices])

count\_in\_cats.append([cat, len(similar\_indices)])

return similar\_products, count\_in\_cats

def \_save\_everything(self, epoch, data\_with\_embeddings):

shutil.copy2('./loss.txt', f'./objs/loss.txt')

data\_with\_embeddings.to\_json(f'./objs/data\_with\_embeddings\_{epoch}.json')

self.umap\_vector\_visualization(np.vstack(data\_with\_embeddings['emb\_main'].values), color=data\_with\_embeddings['cat1'], write\_to\_html\_path=f'./objs/cat1\_{epoch}.html')

def \_show\_imgs(self, data, index1, index2):

"""вывод изображений"""

plt.figure(figsize=(20, 10))

photo1 = data.loc[index1, 'image']

photo2 = data.loc[index2, 'image']

# Путь к файлам с изображениями

path\_to\_image1 = f'{self.img\_dir}{photo1}'

path\_to\_image2 = f'{self.img\_dir}{photo2}'

# Загрузка изображений

image1 = mpimg.imread(path\_to\_image1)

image2 = mpimg.imread(path\_to\_image2)

# Создание нового графика

plt.figure()

# Первое изображение

plt.subplot(1, 2, 1)

plt.imshow(image1)

plt.title(data.loc[index1, 'name'], fontsize=8)

plt.axis('off') # Отключение осей

# Второе изображение

plt.subplot(1, 2, 2)

plt.imshow(image2)

plt.title(data.loc[index2, 'name'], fontsize=8)

plt.axis('off')

plt.show('image.png')

def clear\_images(self):

image\_folder = os.listdir(self.img\_dir)

bad\_df = pd.DataFrame(columns = self.data.columns)

print('Исходный датасет: ', self.data.shape)

for i in range(len(self.data)):

try:

if not self.data.loc[i, 'image'] in image\_folder:

bad\_df.loc[i, :] = self.data.loc[i, :]

except:

print(self.data.loc[i, 'image'])

break

self.data.drop(index=bad\_df.index, inplace=True)

self.data.reset\_index(drop=True, inplace=True)

print('Очищенный:', self.data.shape)

return self.data

def compare\_names(self, data=None):

"алгоритм поиска идентичных товаров по именам"

if data is None:

data = self.data

similarity = []

cats = list(set(data['cat1']))

for cat in tqdm(cats):

d = data[data['cat1'] == cat]

indexes = d.index

for idx, i in enumerate(indexes[:-1]):

for j in indexes[idx + 1:]:

if d['name'][i].lower() == d['name'][j].lower():

similarity.append([i, j])

return similarity

pipeline\_test.py

!pip install transformers

!pip install umap-learn

import os

os.environ["CUDA\_VISIBLE\_DEVICES"] = "1"

import pandas as pd

from pipeline import Pipeline

data = pd.read\_json('./data/cleared\_data.json')

data.reset\_index(drop= True , inplace= True )

data.tail(5)

pipeline = Pipeline(data, './all\_photos/')

pipeline.extract\_zip('./all\_photos.zip') # разархивировать фотки в локальную папку

pipeline.train(num\_epoch=25)

pairs = pipeline.predict() # возвращает пары [[косинус на 1 этапе, косинус на 2 этапе], индекс1, индекс2]

pairs

with open('pairs.txt', 'a') as f:

for i in pairs:

f.write(str(i[0]) + ' ' + str(i[1][0][0]) + ' ' + str(i[2]) + ' '+ str(i[3]) + '\n')

pipeline.\_show\_imgs(data, 20844, 20851)

data\_preparation.py

!pip install transformers

!pip install pymorphy3

!python -m spacy download ru\_core\_news\_sm

!python -m spacy download ru\_core\_news\_md

import os

import re

import nltk

import json

import spacy

import pymorphy3

import numpy as np

import transformers

import pandas as pd

from tqdm import tqdm

import ru\_core\_news\_md

import ru\_core\_news\_sm

from pprint import pprint

from spacy import displacy

from transformers import pipeline

from nltk.tokenize import word\_tokenize

morph = pymorphy3.MorphAnalyzer()

nltk.download('punkt')

COLAB = '/content/drive/MyDrive/sarawan\_data/'

DATA = COLAB + 'data/'

PHOTO = COLAB + 'data/photo/'

OBJS = COLAB + 'objs/'

MODELS = COLAB + 'models/'

nlp = spacy.load('ru\_core\_news\_sm')

#google colab

# colab drive

from google.colab import drive

drive.mount('/content/drive')

# download photos

import zipfile

with zipfile.ZipFile(PHOTO + 'auchan\_perekrestok.zip', 'r') as zip\_ref:

zip\_ref.extractall('./')

ALL\_PHOTOS = 'auchan\_perekrestok'

# load stop\_words and datasets on google colab

stop\_words = []

with open(DATA + 'utils/stop\_words.txt', encoding='utf-8') as f:

for i in f.readlines():

stop\_words.append(i[:-1])

a = pd.read\_json(DATA + 'original/auchan.json')

p = pd.read\_json(DATA + 'original/perek.json')

#обработка сырых датасетов

def is\_exception(a, b):

"""вытаскиваем имена файлов фоток"""

try:

url = a[b]

if isinstance(url, list):

url = url[0]

url = url.split('/')

return url[-1]

except:

return np.nan

def unpack\_perekrestok\_attributes\_dict(attributes\_column:pd.Series):

"""вытаскиваем вложенные друг в друга словари атрибутов"""

attributes\_column\_list = []

for i in range(len(attributes\_column)):

info = attributes\_column[i]['Информация']

try:

features = attributes\_column[i]['Свойства товара']

info.update(features)

except: pass

volume = attributes\_column[i]['volume']

info['volume'] = volume

attributes\_column\_list.append(info)

return attributes\_column\_list

perek = pd.DataFrame()

auchan = pd.DataFrame()

# берем важные колонки

auchan['name'] = a['product\_name']

auchan['image'] = a['product\_img\_links']

auchan['volume'] = a['product\_human\_volume']

auchan['attributes'] = a['specification']

auchan['cat1'] = a['main\_category\_name']

auchan['cat2'] = a['category\_name']

perek['name'] = p['product\_name']

perek['image'] = p['product\_img\_link']

perek['volume'] = p['product\_volume'].apply(lambda x: str(x) + ' ') + p['product\_volume\_type']

perek['attributes'] = p['features']

perek['cat1'] = p['main\_category\_name']

perek['cat2'] = p['primary\_category']

perek['cat3'] = p['category\_name']

# добавляем в атрибуты volume

for i in range(len(perek)):

perek.loc[i, 'attributes']['volume'] = perek.loc[i, 'volume']

for i in range(len(auchan)):

auchan.loc[i, 'attributes']['volume'] = auchan.loc[i, 'volume']

perek.drop(columns=['volume'], inplace=True)

auchan.drop(columns=['volume'], inplace=True)

# достаем вложенные словари

perek['attributes'] = unpack\_perekrestok\_attributes\_dict(perek['attributes'])

# вытаскиваем имена файлов фоток

auchan['image'] = auchan['image'].apply(lambda x: is\_exception(x, 'preview\_url'))

perek['image'] = perek['image'].apply(lambda x: is\_exception(x, 'small\_url'))

#приведение имен и атрибутов к хорошему виду

def each\_attribute\_pocessing(product):

"""очищает атрибуты товара от атрибутов bad\_parameters"""

bad\_parameters = ['рекомендац', 'срок', 'гаранти', 'измерени', 'цена','условия', 'хранения',

'isbn', 'дополнени', 'регион', 'время варки', 'не содержит', 'температур',

'подача', 'может содержать', 'адрес', 'сочетание', 'тип потребления', 'углевод',

'стандарт', 'предосторожност', 'гарантия', 'артикул товара', 'калорийность',

'аннотац', 'органы', 'погрешность', 'сторона', 'сертифик', 'cпособ', 'дополнени', 'дополнит', 'бренд', 'состав']

bad\_parameters\_for\_values = ['да', 'нет', '-']

attr\_names = list(product.keys())

attr\_values = list(product.values())

brand = ''

for name, val in zip(attr\_names, attr\_values):

flag = 0

for p in bad\_parameters:

if 'бренд' in name.lower():

brand = val.lower()

if p in name.lower():

del product[name]

flag = 1

break

if flag == 0:

if val.lower() in bad\_parameters\_for\_values:

del product[name]

flag = 1

return product, brand

def attributes\_processing(attr\_column: pd.Series):

"""убирает плохие атрибуты, приводит к надлежащему виду атрибуты"""

attr = attr\_column.copy(deep=True)

brand = attr\_column.copy(deep=True)

for idx in tqdm(attr.index):

attr[idx], brand[idx] = each\_attribute\_pocessing(attr[idx])

attr[idx] = ' '.join(attr[idx].values())

attr[idx] = ' '.join([morph.parse(i)[0].normal\_form.lower() for i in re.findall("[a-zA-Zа-яА-Яё]+|[0-9.%]+", attr[idx]) if i != '' and i not in '.,:;\"\'@!#№$^?\*\(\)\\/' and i.lower() not in stop\_words])

return attr, brand

auchan['attributes\_prepared'], auchan['brand'] = attributes\_processing(auchan['attributes'])

perek['attributes\_prepared'], perek['brand'] = attributes\_processing(perek['attributes'])

#убираем пропуски в атрибутах

# вспомогательная функция проверки на число.

def is\_num(num:str):

try:

float(num)

return True

except ValueError:

return False

def clean\_attributes(data:str, clean\_all=False):

clean\_list = ['вес', 'змж', 'вт','г','кг','л', 'мл', 'лист','листа',\

'листов', 'элементов', 'элемент', 'файлами', 'файлов',\

'м','мл', 'мм','наб','пара', 'лет', 'см','упак', 'цв',\

'цвет', 'цветов', 'цвета', 'шт', 'kg', 'g', 'mg', 'ml', 'l']

data = data.split()

# убираем лишние пробелы

data = [i.strip() for i in data]

# приклеиваем весовые / обьемные характеристики к их числовым выражениям

volume\_features = {}

for ind, el in enumerate(data):

if ind < len(data) - 1:

if is\_num(el) and data[ind + 1] in clean\_list:

new = el + data[ind + 1]

data.pop(ind)

data.insert(ind, new)

data.pop(ind + 1)

if new in [k for k, v in volume\_features.items()]:

volume\_features[new].append(ind)

else:

volume\_features[new] = [ind]

data = [i for i in data if i not in clean\_list]

for k, v in volume\_features.items():

if len(v) > 1:

for index in v[1:]:

data.pop(index)

for el in data:

if not is\_num(el) and len(el) < 2:

data.remove(el)

if clean\_all:

final\_data = []

for el in data:

if is\_num(el):

final\_data.append(el)

elif el not in final\_data:

final\_data.append(el)

return ' '.join(final\_data)

else:

return ' '.join(data)

def atribute\_duplicate\_cleaning(dataframe, clean\_all=False):

for i in tqdm(range(len(dataframe))):

dataframe.loc[i, 'attributes\_prepared'] = clean\_attributes(dataframe.loc[i, 'attributes\_prepared'], clean\_all

return dataframe

auchan = atribute\_duplicate\_cleaning(auchan, True)

perek = atribute\_duplicate\_cleaning(perek, True)

def name\_preprocessing(data):

"""приводит имена к надлежащему виду"""

symbols = ['вес', 'змж', 'вт','г','кг','л', 'мл', 'лист','листа', 'листов', 'элементов', 'элемент', 'файлами', 'файлов', 'м','мл', 'мм','наб','пара', 'лет', 'см','упак', 'цв', 'цвет', 'цветов', 'цвета', 'шт', 'предмета', 'месяцев','месяца', 'год', 'лет', 'сорт', 'вида', 'вид', 'тон', 'смотреть', 'размер', 'без', 'сахара', 'сахар',

'caramello','daino','nero','айворя','алый','ассорти','бежевый','белые','белый','берюзовый','бирюзовый','бледно','бордо','бордовый','бронза','бронзовый','бурый','голубой','дымка','дымчатый','жёлтый','зелёный','золотистый','золотой','изумрудный','капучиный','карамель','комбинированный','коралл','коралловый',

'коричневый', 'шоколадный', 'коричнеый','кость','кофейный','красный','кремовый','лаванда','лиловый','лимонный','малиновый','медный','металлик','многоцветный','молочный','морской','мультиколор','мультицвет','мятный','натуральный',

'нефрит','оливковый','оранжевый','ореховый','оттенок','охра','пастельный','персиковый','прозрачный','пудровыя','пурпурный','разгоцветный','разноцвентный','разноцветеый','разноцветный','разноцветь','рисунок','розовый','рубиновый','салатовый','светло','светлый','серебристый','серебряный','серо','серый','синий',

'сиреневый','слоновый','сталь','стальной','телесный','темно','терракотовый','тёмный','узор','фиолетовый','фисташковый','фуксия','хаки','хром','цвет','цветной','черно','чёрный','шампаня','янтарный','ярко','white','black','red','yellow', 'orange','green','blue','purple','pink','brown','grey',

'xxxxl', 'xxxl', 'xxl', 'xl', 'l','xxxxs', 'xxxs', 'xxs', 'xs', 's', 'xxxxm', 'xxxm', 'xxm', 'xm', 'm', 'холодный', 'теплый', 'тёплый', 'газированный', 'негазированный']

for idx in tqdm(data.index):

name = ' '

for word in word\_tokenize(data['name'][idx]):

word = morph.parse(word.replace(',', '.'))[0].normal\_form.lower()

if word not in stop\_words and word not in ' \-\|\(\)\.,;:@\"\'\`~$^&\*\_-+=|/\\\+\!"\'«»':

name += ' ' + word

if word in symbols or bool(re.search(r'\d', word)):

data['attributes\_prepared'][idx] += ' ' + word

data.loc[idx, 'name\_prepared'] = name.strip()

return data

auchan = name\_preprocessing(auchan)

perek = name\_preprocessing(perek)

#соединение магазинов

data\_concat = pd.concat((auchan, perek), axis=0, ignore\_index=True)

data\_concat.head()

def extract\_attr\_pos\_tagging(data):

features = []

for index, name, attr, brand in tqdm(zip(data.index, data.name, data.attributes, data.brand)):

name = ' '.join(re.split(r'\W', name.lower()))

try:

if brand in name.lower():

name = name.lower().replace(brand.lower(), brand.title())

except KeyError:

pass

ner = nlp(name)

attr = []

for token in ner:

if (token.pos\_ in ('NOUN', 'NUM', 'VERB', 'ADV', 'PROPN', 'X') and token.dep\_ not in ('ROOT', 'nsubj') or (token.pos\_ == 'VERB' and token.dep\_ == 'ROOT') or token.pos\_ == 'ADJ') and str(token).istitle() == False and morph.parse(str(token))[0].normal\_form not in data['attributes\_prepared'][index]:

attr.append(morph.parse(str(token))[0].normal\_form)

data['attributes\_prepared'][index] = data['attributes\_prepared'][index] + ' ' + ' '.join(attr)

return data

data\_concat['before\_pos\_tagging'] = data\_concat['attributes\_prepared']

data\_concat = extract\_attr\_pos\_tagging(data\_concat)

data\_concat.head(5)

# до и после pos\_tagging

for k, i, j in zip(data\_concat['name'], data\_concat['before\_pos\_tagging'], data\_concat['attributes\_prepared']):

print(f'Имя:{k} \nДо: {i}\nПосле: {j}\n\n')

#проверка валидности фото

def clear\_images(dataframe, photo\_dir='all\_photos'):

"""убирает из датафрейма товары, если их фоток нет в папке photo\_dir"""

image\_folder = os.listdir(photo\_dir)

bad\_df = pd.DataFrame(columns = dataframe.columns)

print(dataframe.shape)

for i in range(len(dataframe)):

try:

if not dataframe.loc[i, 'image'] in image\_folder:

bad\_df.loc[i, :] = dataframe.loc[i, :]

except:

print(dataframe.loc[i, 'image'])

break

dataframe.drop(index=bad\_df.index, inplace=True)

dataframe.reset\_index(drop=True, inplace=True)

print(dataframe.shape)

return dataframe

data\_concat = clear\_images(data\_concat)

data\_concat.reset\_index(drop=True, inplace=True)

#обработка категорий

data\_concat = data\_concat.replace({'cat1' : {'Алкоголь (самовывоз)' : 'Алкогольные напитки',

'Соки, воды, напитки' : 'Вода, соки, напитки',

'Кофе, чай, какао, сахар' : 'Чай, кофе, горячие напитки',

'Шоколад, конфеты, сладости' : 'Сладости',

'Чипсы, снеки, попкорн' : 'Снеки',

'Чипсы, снеки, сухофрукты' : 'Снеки',

'Орехи, семечки, сухофрукты' : 'Снеки',

'Овощи, фрукты, грибы' : 'Овощи, фрукты, ягоды, грибы',

'Овощи, фрукты, орехи' : 'Овощи, фрукты, ягоды, грибы',

'Уборка и бытовая химия' : 'Уборка, товары для дома',

'Уборка' : 'Уборка, товары для дома',

'Товары для дома' : 'Уборка, товары для дома',

'Для дома и дачи' : 'Уборка, товары для дома',

'Рыба' : 'Морепродукты',

'Рыба, икра, морепродукты' : 'Морепродукты',

'Красота и гигиена' : 'Красота, гигиена, аптека',

'Отдых на даче' : 'Сад и огород',

'Молоко, сыр, яйца' : 'Молочные продукты',

'Молочные продукты, яйца' : 'Молочные продукты',

'От Перекрёстка' : 'Собственные торговые марки',

'Посуда' : 'Кухня',

'Все для детей' : 'Игрушки',

'Сухие завтраки, мюсли' :'Макароны, крупы',

'Макароны, крупы, масло, специи' : 'Макароны, крупы',

'Продукты быстрого приготовления' : 'Готовая еда'}})

data\_concat = data\_concat[(data\_concat['cat1'] != 'Зоотовары') &

(data\_concat['cat1'] != 'Игрушки') &

(data\_concat['cat1'] != 'Канцтовары') &

(data\_concat['cat1'] != 'Книги, журналы') &

(data\_concat['cat1'] != 'Сад и огород') &

(data\_concat['cat1'] != 'Системы нагревания, табак') &

(data\_concat['cat1'] != 'Спорт и отдых') &

(data\_concat['cat1'] != 'Сад и огород')]