МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра дискретной математики и информационных технологий

АНАЛИЗ МНЕНИЙ МЕТОДАМИ МАШИННОГО ОБУЧЕНИЯ

БАКАЛАВРСКАЯ РАБОТА

студентки 4 курса 421 группы
направления 09.03.01 — Информатика и вычислительная техника
факультета КНиИТ
Сапаевой Иоанны Викторовны

Научный руководитель
Профессор кафедры ДМиИТ _____ Л. В. Кальянов

Заведующий кафедрой
доцент, к. ф.-м. н. _____ Л. Б. Тяпаев

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С развитием интернета в современном мире стало возможно многое, недоступное раньше. Например, покупка товаров онлайн, не выходя из дома. Для этого были созданы различные маркетплейсы, предоставляющие услуги по продаже всего, от продуктов до сложной техники. И, конечно же, для удобства покупателей была создана система отзывов и оценивания товаров. С каждым днём количество выставляемых и продаваемых товаров и услуг растёт, а значит, растёт и количество отзывов. И ориентироваться в этом огромном количестве информации становится всё сложнее. Отзывы стали неотъемлемой частью процесса принятия решений о покупке, а их влияние на поведение потребителей с каждым днём становится все более значительным. В свете этих изменений, анализ мнений стал одной из самых важных задач для бизнеса и общества в целом. Сегодня компании стремятся понять предпочтения и потребности своих клиентов, чтобы оптимизировать свои продукты, услуги и стратегии маркетинга.

Однако, с увеличением объёма доступной информации, ручной анализ мнений становится неэффективным и непрактичным, а порой и попросту невозможным. В связи с этим возникает потребность в автоматизации процесса анализа мнений, что приводит к использованию методов машинного обучения. Машинное обучение предоставляет возможность обрабатывать и анализировать большие объёмы данных, выделять основные темы, определять эмоциональную окраску и делать выводы на основе этих данных. Это открывает новые перспективы для компаний, продвигающих свой бизнес, помогая им принимать обоснованные решения и улучшать качество предлагаемых продуктов и услуг.

Целью данной выпускной квалификационной работы бакалавра является исследование и анализ методов машинного обучения для анализа мнений на основе датасета отзывов с сайта Амазон.

Амазон является одним из крупнейших онлайн-ритейлеров, предоставляющих широкий спектр товаров и услуг, и каждый день на его сайте публикуются тысячи отзывов от пользователей. Использование датасета отзывов с сайта Амазон представляет собой ценный ресурс для анализа мнений и оценки качества предлагаемых продуктов.

Основной целью данной работы является разработка и применение моделей машинного обучения для классификации отзывов на основе их тональности,

то есть определение, является ли отзыв положительным, или отрицательным. Для достижения этой цели необходимо решить следующие задачи:

Сбор и предобработка данных: необходимо собрать достаточно большой датасет отзывов с сайта Амазон и провести их предварительную обработку, включающую очистку данных, токенизацию и удаление стоп-слов.

Выбор и обучение моделей: необходимо выбрать подходящие модели машинного обучения для анализа мнений и обучить их на предварительно обработанных данных.

Оценка производительности моделей: необходимо провести эксперименты для оценки производительности различных моделей.

Анализ результатов: необходимо проанализировать полученные результаты и сделать выводы о применимости и эффективности выбранных моделей машинного обучения для анализа мнений на основе датасета отзывов с сайта Амазон.

Ожидается, что результаты данной работы помогут в создании инструмента для автоматического анализа мнений на основе отзывов с сайта Амазон. Этот инструмент может быть полезен для бизнеса в плане понимания общественного мнения о продуктах и услугах, выявления проблем и улучшения качества предлагаемых товаров. Кроме того, результаты исследования могут быть использованы для дальнейших исследований в области анализа мнений и машинного обучения.

Актуальность данной работы заключается в том, что результаты работы могут быть использованы для принятия решений в области управления качеством продуктов и услуг, а также для улучшения взаимодействия с клиентами. К примеру, возможна реализация оценки негативного отзыва и выяснения причины недовольства клиента с дальнейшей рекомендацией ему по исправлению причинённых неприятностей. Допустим, рекомендация купить схожий по цене и предназначению продукт лучшего качества, связь с техническим специалистом или советы по исправлению недочётов уже приобретённого продукта.

В следующих разделах работы будут представлены описание датасета и методики его предобработки, описание используемых методов и алгоритмов, подробные результаты экспериментов и их анализ, а также обзор литературы.

1 Анализ мнений как теоретическая задача

Так как темой данной работы является анализ мнений, рассмотрим этот термин подробнее.

Анализ мнений или анализ тональности текстов - это спектр задач из области компьютерной лингвистики, решающий проблемы автоматизированного распознавания и изучения эмоциональной оценки авторов с использованием эмоционально окрашенной лексики по отношению к объектам, о которых идёт речь в конкретном тексте. Подобные задачи стали актуальны в начале этого века и стали развиваться активнее по причине большого количества возможностей практического их применения в различных областях жизни. К примеру, в мониторинге различных брендов, при прогнозировании рынка, выявлении общественных настроений и др. Говоря подробнее, анализ мнений определяется как вычислительное исследование, анализирующее отношение, волнение, эмоции, выражения, точки зрения и мнения людей по отношению к определённому объекту. Анализ мнений - одно из самых обширных применений текстовой аналитики. Он фокусируется на анализе настроений различных текстовых источников, таких как государственные службы, интеллектуальные приложения, корпоративные опросы, обзоры продуктов, рецензии на фильмы, отзывы о ресторанах и др. Также подобный вид анализа используется для извлечения мнения клиентов из их текстовых отзывов об услуге или продукте. Государственные и частные организации стремятся узнать мнение людей о своих услугах и продуктах. Область анализа мнений привлекает внимание исследователей и вызывает у них ажиотаж из-за расширяющихся реальных деловых и социальных приложений. Выяснение мнения людей считается важным фактором для принятия ими решений. Важно узнать, что люди думают о конкретном продукте или услуге в соответствии со своим опытом. Развивающийся интернет позволил пользователям выражать своё мнение и обеспечил быстрый рост пользовательского контента в Интернете.

Мнения в социальных сетях играют неотъемлемую роль в повседневной жизни. Комментарии и отзывы в социальных сетях и Интернете считаются одним из самых надёжных и мощных источников информации. Анализ постов в социальных сетях и веб-контента очень важен [1]. Поэтому анализ мнений или Sentiment Analysis может помочь понять мысли и чувства людей по отношению к определённой теме, принять решение о рецензии на фильм, узнать мнение лю-

дей о кандидате, предсказать фондовые рынки, извлечь аспекты или атрибуты услуг или продуктов, а также автоматически определить важные аспекты и суммировать различные положительные и отрицательные отзывы в соответствии с этими аспектами или атрибутами и т.д.

Обширное количество информации привело к тому, что государственный и частный секторы стали уделять все больше внимания изучению и учёту отзывов потребителей в процессе постоянного развития своих услуг и продуктов. Различные организации ищут подходы, ориентированные на клиента, чтобы узнать мнение потребителей об их услугах. И наоборот, клиенты пытаются узнать мнение существующих пользователей о конкретной услуге, прежде чем воспользоваться ею. "Что другие люди думают об этом? важный вопрос, который всегда приходит на ум перед принятием любого решения.

В своё время проводилось множество опросов с целью получения представления о том, как люди относятся к использованию различных мобильных приложений и маркетплейсов, и выяснить, что влияет на их решение о покупке того или иного продукта. Результаты подобных опросов каждый раз показывают значительное влияние рейтинга товара на решение о его приобретении.

Анализ тональности текста позволяет извлечь мнение автора из текста по отношению к заданному объекту. Это отношение может содержать в себе суждение, оценку автора, его эмоциональное состояние или мнение.

Анализ мнений или настроений сталкивается, в первую очередь, с тем же набором проблем, что и распознавание эмоций. Необходимо, в первую очередь, что такое «настроение», прежде чем его определять. К примеру, можно ли оценивать чувства согласно некоторой фиксированной шкале, скажем, от 1 до 5, где 1 - это резко негативное отношение, а 5 - строго позитивное.

К тому же люди часто используют сложные способы для выражения своего мнения: риторические конструкции, иронию, сарказм и подразумевающееся значение. Это может ввести алгоритм анализа в заблуждение и привести к неверному результату. И единственный способ избежать этого и понять реальное значение - это через контекст. Начало абзаца или предложения может существенно повлиять на восприятие настроения последующего текста.

Сейчас в современных алгоритмах и способах оценки настроения осуществляется в категориальной структуре. То есть, настроения анализируются относительно их принадлежности к определённой группе. К примеру, «радость»

- 30%, «возбуждение» - 70%, «счастье» - 65%. Эти цифры не суммируются до ста процентов, так как они являются индивидуальными признаками того насколько то или иное состояние относится к предложению.

Имеются различные типы классификации найденных в тексте тональностей. Самым часто встречающимся и основным из них является стандартное деление на «негативные» и «позитивные» настроения.

В дальнейшем в работе будет реализована бинарная классификация при помощи методов логистической регрессии и свёрточных нейронных сетей.

Рассмотрим эти понятия подробнее.

Задача классификации - это подкатегория методов машинного обучения с учителем, суть которой заключается в идентификации категориальных меток классов для новых экземпляров на основе предыдущих наблюдений, как показано на рисунке 1. Метка класса представляет собой дискретное, неупорядоченное значение, которое может пониматься как принадлежность группе экземпляров [2].

Рисунок 1 – Задача классификации

Бинарная классификация - это один из типов задач классификации в машинном обучении, когда необходимо классифицировать два взаимоисключающих класса, как показано на рисунке 2 [3]. То есть, это классификация с бинарной переменной класса, или категориальной выходной переменной, которая может принимать только два значения. Бинарные классификационные модели являются более понятными и интерпретируемыми.

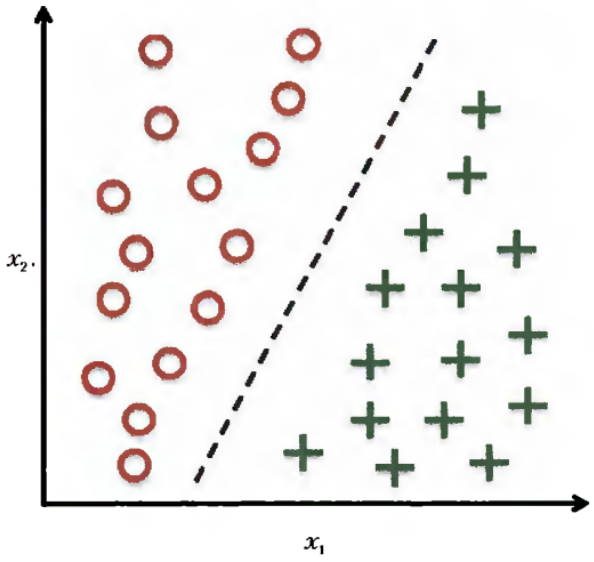

Рисунок 2 – Бинарная классификация

Для их построения широко распространены как раз таки такие методы, как логистическая регрессия и свёрточные нейронные сети [4].

Логистическая регрессия - это полезный инструмент для решения задач регрессии и классификации, представляет собой разновидность множественной регрессии, общее назначение которой состоит в анализе связи между несколькими независимыми переменными, называемыми также предикторами или регрессорами, и зависимой переменной. Бинарная логистическая регрессия применяется в том случае, когда когда зависимая переменная является бинарной, то есть принимающей только два значения, к примеру, 0 или 1. С помощью логистической регрессии можно оценивать вероятность того, что событие наступит для конкретного испытуемого объекта [5].

Все регрессионные модели могут быть записаны в виде формулы:

$$y = F(x_1, x_2, \ldots, x_n)$$

В множественной линейной регрессии предполагается, что зависимая переменная является линейной функцией независимых переменных, то есть:

$$y = a + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n$$

Её вполне можно использовать для задачи оценки вероятности исхода события, вычислив стандартные коэффициенты регрессии. То есть, коэффициенты уравнения регрессии, показывающие силу и характер влияния независимых

переменных на зависимую и характеризующие степень значимости отдельных переменных для повышения точности модели.

Такими коэффициентами являются параметры a и b в уравнении линейной регрессии. $y = ax + b$ Они выбираются таким образом, чтобы сумма квадратов отклонений точек, соответствующих реальным наблюдениям данных, от линии регрессии была минимальной. Подбор коэффициентов производится по методу наименьших квадратов.

Метод наименьших квадратов - это математический подход для оценки параметров моделей на основании экспериментальных данных, содержащих случайные ошибки.

Если данные известны с некоторой погрешностью, то вместо неизвестного точного значения параметра модели используется приближенное. Поэтому параметры модели должны быть рассчитаны так, чтобы минимизировать разницу между экспериментальными данными и теоретическими.

Мерой рассогласования между фактическими значениями и значениями, оценёнными моделью в методе наименьших квадратов, служит сумма квадратов разностей между ними, то есть:

$$\sum_{i=1}^{N}(y' - y)^2$$

где y'? оценка, полученная с помощью модели, y - фактическое наблюдаемое значение. Очевидно, что лучшей будет считаться та модель, которая минимизирует данную сумму [6].

Однако, после подбора коэффициентов, может возникнуть проблема: множественная регрессия не «знает», что переменная отклика бинарна по своей природе. Это неизбежно приведёт к модели с предсказываемыми значениями, большими 1 или меньшими 0. Но такие значения вообще не допустимы для первоначальной задачи. Таким образом, множественная регрессия просто игнорирует ограничения на диапазон значений для y.

Для решения проблемы задача регрессии может быть сформулирована иначе: вместо предсказания бинарной переменной, мы предсказываем непрерывную переменную со значениями на отрезке $[0, 1]$, при любых значениях независимых переменных. Это достигается применением следующего регрессионного уравнения (логит-преобразование):

$$P = \frac{1}{1 + e^{-1}}$$

где P ? вероятность того, что произойдёт интересующее событие, e - основание натуральных логарифмов, y ? стандартное уравнение регрессии.

Зависимость между вероятностью события и величиной y можно наблюдать на следующем графике, показанном на рисунке 3:

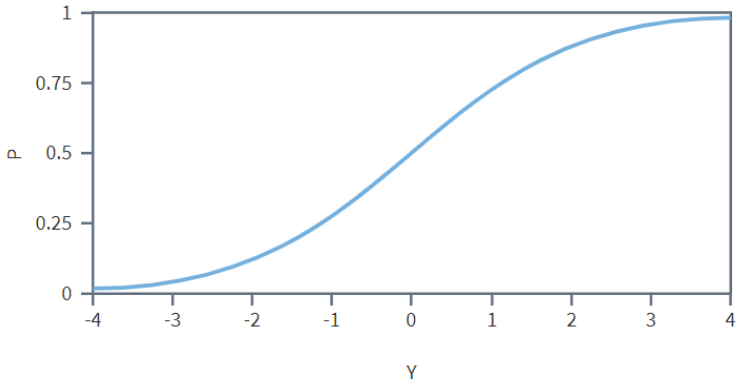

Рисунок 3 – Зависимость между вероятностью события P и величиной y

Следует пояснить необходимость преобразования. Можно предположить, что рассуждение ведётся о зависимой переменной в терминах основной вероятности P, лежащей между 0 и 1. Тогда преобразуем эту вероятность P:

$$P' = \log_e(\frac{P}{1 - P})$$

Это преобразование обычно называют логистическим или логит-преобразованием Теоретически, P' может принимать любое значение. Поскольку логистическое преобразование решает проблему об ограничении на 0-1 границы для первоначальной зависимой переменной (вероятности), то эти преобразованные значения можно использовать в обычном линейном регрессионном уравнении. А именно, если произвести логистическое преобразование обеих частей описанного выше уравнения, мы получим стандартную модель линейной регрессии.

Существует несколько способов нахождения коэффициентов логистической регрессии. На практике часто применяется метод максимального правдоподобия. Он применяется в статистике для получения оценок параметров генеральной совокупности по данным выборки.

Генеральная совокупность - это совокупность всех объектов или наблюдений, относительно которых исследователь намерен делать выводы при решении

конкретной задачи. В её состав включаются все объекты, которые подлежат изучению.

Объём генеральной совокупности может быть очень велик, и на практике рассмотреть все её элементы не представляется возможным. Поэтому обычно из генеральной совокупности извлекаются выборки, на основе анализа которых делаются уже выводы о свойствах всей совокупности, скрытых в ней закономерностях, действующих правилах и т.д. При этом выборки должны быть репрезентативными [7].

Основу же метода максимального правдоподобия составляет функция правдоподобия (likehood function), выражающая плотность вероятности (вероятность) совместного появления результатов выборки:

$$L(Y_1, Y_2, \ldots, Y_k; \theta) = p(Y_1; \theta) \cdot \cdots \cdot p(Y_k; \theta)$$

Согласно методу максимального правдоподобия в качестве оценки неизвестного параметра принимается такое значение $\theta = \theta(Y_1, \ldots, Y_k)$, которое максимизирует функцию L.

Нахождение оценки упрощается, если максимизировать не саму функцию L, а натуральный логарифм $\ln(L)$, поскольку максимум обеих функций достигается при одном и том же значении θ:

$$L * (Y; \theta) = \ln(L(Y; \theta)) \rightarrow max$$

В случае бинарной независимой переменной, которую мы имеем в логистической регрессии, выкладки можно продолжить следующим образом. Обозначим через P_i вероятность появления единицы: $P_i = Prob(Y_i = 1)$. Эта вероятность будет зависеть от $X_i W$, где X_i - строка матрицы регрессоров, W - вектор коэффициентов регрессии:

$$P_i = F(X_i W), F(z) = \frac{1}{1 + e^{-z}}$$

Логарифмическая функция правдоподобия равна:

$$L^* = \sum_{i \in I_1} \ln P_i(W) + \sum_{i \in I_0} \ln(1 - P_i(W) = \sum_{i=1}^{k} [Y_i \ln P_i(W) + (1 - Y_i) \ln(1 - P_i(W))])$$

где I_0, I_1- множества наблюдений, для которых $Y_i = 0$ и $Y_i = 1$ соответственно.

Можно показать, что градиент g и гессиан H функции правдоподобия равны:

$$g = \sum_i (Y_i - P_i) X_i$$

$$H = -\sum_i P_i (1 - P_i) X_i^T \; X_i \leq 0$$

Гессиан всюду отрицательно определённый, поэтому логарифмическая функция правдоподобия всюду вогнута. Для поиска максимума можно использовать метод Ньютона, который здесь будет всегда сходиться (выполнено условие сходимости метода):

$$W_{t+1} = W_t - (H(W_t))^{-1} g_t(W_t) = W_t - \triangle W_t$$

Логистическую регрессию можно представить в виде однослойной нейронной сети с сигмоидальной функцией активации, веса которой есть коэффициенты логистической регрессии, а вес поляризации - константа регрессионного уравнения [8] [9].

Рассмотрим также свёрточные нейронные сети, которые будут использоваться в дальнейшем в практической части работы.

Как известно, глубокое обучение сейчас привлекает к себе много внимания и является самой горячо обсуждаемой темой в сфере машинного обучения. Глубокое обучение может пониматься, как набор алгоритмов, разработанных для наиболее эффективной тренировки искусственных нейронных сетей, состоящих из множества слоёв.

Искусственные нейроны составляют базовые элементы многослойных искусственных нейронных сетей. Ключевая идея, лежащая в основе искусственных нейроных сетей, опирается на гипотезы и модели того, каким образом человеческий мозг работает для решения сложных практических задач. Несмотря на то, что в последние годы искусственные нейронные сети получили большую популярность, ранние исследования нейронных сетей восходят к 1940-м, когда Уоррен Маккалок и Уолтер Питт впервые дали описание того, как могли работать нейроны. Однако в последующие десятилетия вслед за первой реализацией модели нейрона Маккалока-Питта, персептрона Розенблатта в 1950-х, многие

научные исследователи и практики в области машинного обучения постепенно стали терять интерес к нейронным сетям, поскольку никто не мог предложить хорошего решения задачи тренировки многослойной нейронной сети. Позже интерес к нейронным сетям появился вновь, когда в 1986 г. Д. Румелхарт, Г. Хинтон и Р. Дж. Уильямс повторно открыли и популяризировали алгоритм обратного распространения ошибки (backpropagation) для наиболее эффективной тренировки нейронных сетей.

В течение длительного периода многие другие важные прорывные работы привели к тому, что теперь мы называем алгоритмами глубокого обучения. Они используются для создания из немаркированных данных детекторов признаков (feature detectors), которые применяются для предварительной тренировки глубоких нейронных сетей ? нейронных сетей, скомпонованных из из многих слоёв. Нейронные сети ? горячая тема не только в научных кругах, но и в крупных технологических компаниях, таких, как Google, Microsoft, Facebook, которые выделяют огромные инвестиции на искусственные нейронные сети и научные исследования в области глубокого обучения. На данный момент сложные нейронные сети, приводимые в действие алгоритмами глубокого обучения, рассматриваются как передовая технология, когда заходит речь о решении таких сложных задач, как распознавание изображений и голоса. Популярными примерами продуктов из нашей повседневной жизни, которые приводятся в действие технологиями глубокого обучения можно считать службы Google по поиску различных изображений на основе исходного и переводу, а также приложение для смартфонов, способное автоматически распознавать текст в изображениях с его последующим переводом на 20 языков в реальном времени.

В крупнейших технологических компаниях существуют и ещё много других интересных приложений, находящихся в процессе активной разработки, к примеру приложение DeepFace от компании Facebook для маркировки изображений. Кроме того, фармацевтическая промышленность недавно начала использовать методы глубокого обучения для изобретения лекарственных препаратов и предсказания токсичности; исследование показало, что эти новые методы существенно превышают качество традиционных методов виртуального скрининга, то есть выбора соединений путём оценки их желательности в вычислительной модели.

Нейронные сети работают, выполняя некоторые вычисления над заданны-

ми данными, а затем выдавая результат. Процесс выглядит следующим образом:

1. Веса и смещения инициализируются небольшими числами, близкими к нулям. Существует несколько способов их инициализации, например, с помощью нормального или равномерного распределения.
2. Затем наблюдения поступают на входной слой.
3. Затем эти наблюдения распространяются по сети, и получаются прогнозы. Это известно как прямое распространение. Когда данные проходят через скрытые слои, применяется функция активации. Чаще всего в этих слоях используется функция активации ReLu. Она обнуляет все числа до 0 и выше.
4. Перед получением конечного результата ‘y‘ результат проходит через другую функцию активации в выходном слое. Функция активации гарантирует, что сеть выдаёт ожидаемый результат. Например, в задаче классификации вы хотите, чтобы конечным выходом вашей модели была вероятность того, что вход принадлежит к определённому классу. Некоторые распространённые функции активации включают сигмоид, используемый для бинарной классификации, и softmax, используемый для многоклассовых задач, где метки являются взаимоисключающими.
5. Следующим шагом является сравнение предсказанного результата с истинным. Разница между ними и есть ошибка. Сеть обучается, корректируя веса в зависимости от ошибки. Для этого используется функция потерь, также известная как функция затрат. Цель состоит в том, чтобы минимизировать эту функцию затрат. Выбор функции затрат зависит от задачи, например, для задач классификации вы будете использовать функции потерь классификации.
6. Следующим шагом является передача этих ошибок обратно в сеть, чтобы она могла скорректировать веса. Так сеть обучается. Это известно как обратное распространение. Это делается с помощью градиентного спуска. Градиентный спуск - это стратегия оптимизации, направленная на получение точки с наименьшей ошибкой. Существует несколько вариантов градиентного спуска. К распространенным относятся пакетный градиентный спуск, стохастический градиентный спуск и градиентный спуск с использованием мини-батчей. Градиентный спуск с использованием батчей использует весь набор данных для вычисления градиента функции стои-

мости. Обычно он работает медленно, так как перед выполнением одного обновления необходимо вычислить градиент всего набора данных. При стохастическом градиентном спуске градиент функции стоимости вычисляется по одному обучающему примеру на каждой итерации. В результате он работает быстрее, чем градиентный спуск с батчами. При градиентном спуске с мини-батчами для вычисления градиента функции стоимости используется выборка из обучающих данных.

Одним из видов нейронных сетей являются свёрточные нейронные сети (convolutional neural network, CNN или ConvNet), которые завоевали популярность в компьютерном зрении из-за их экстраординарно хорошего качества работы на задачах классификации изображений, аудиофайлов, в частности речи и анализа тональности текстов. На сегодняшний день CNN являются одной из самых популярных архитектур нейронных сетей в глубоком обучении. Лежащая в основе свёрточных нейронных сетей ключевая идея состоит в создании множества слоёв детекторов признаков (feature detectors), чтобы, к примеру, учитывать пространственное расположение пикселов во входном изображении.

Свёрточная нейронная сеть состоит из двух основных слоёв:

1. Слой свёртки для получения признаков из данных.
2. Слой объединения для уменьшения размера карты признаков.

Слой свёртки и объединяющий слой являются основными составляющими свёрточной нейронной сети.

Свёртка - это процесс, в ходе которого получаются признаки. Затем эти признаки поступают в CNN. Операция свёртки отвечает за выявление наиболее важных признаков. Выход операции свёртки известен как карта признаков, свёрнутая функция или карта активации. Карта признаков вычисляется путём применения детектора признаков к входным данным. В некоторых системах глубокого обучения детектор признаков также называют ядром или фильтром. Например, этот фильтр может быть матрицей 3 на 3. Но для конкретной задачи необходимо подбирать фильтр уже исходя из самой задачи.

Карта признаков вычисляется путем поэлементного умножения ядра и матричного представления входных данных. Это гарантирует, что карта признаков, передаваемая в CNN, будет меньше, но будет содержать все важные признаки. Фильтр делает это, проходя шаг за шагом по каждому элементу входных данных. Эти шаги называются страйдами и могут быть определены при

создании CNN.

При построении свёрточных нейронных сетей фреймворки глубокого обучения обычно позволяют настраивать некоторые из рассмотренных выше элементов. К ним относятся:

— размер ядра,
— количество фильтров,
— высота и ширина полос,
— используемая функция активации,
— как должно быть инициализировано ядро,
— способ инициализации смещения [10].

Также, как уже упоминалось выше, в работе будет использована батч-нормализация. Рассмотрим её подробнее.

В 2015 году два сотрудника корпорации Google, Sergey Ioffe и Christian Szegedy, ввели понятие внутреннего ковариационного сдвига (internal covariance shift), решая проблему ускорения обучения нейронной сети.

Рассмотрим для начала, что такое ковариационный сдвиг.

Допустим, существует некая наблюдаемая величина x, и есть зависящая от x величина y. Необходимо выяснить эту зависимость и научиться по x находить y. Зависимость будет вероятностной, то есть, существует некая условная вероятность, которая определяется при помощи плотности $q(y|x)$. А мы собственно и хотим восстановить эту плотность в виде функции $p(y|x, \theta)$, где θ - это параметры, которые следует подобрать.

Для решения данной задачи выдано некоторое количество пар $(X_n, Y_n)|n = 1, \ldots,$

в качестве тренировочного набора. Чтобы сформировать этот набор, x_n выбирались случайным образом из распределения с плотностью вероятности $q_0(x)$, а y_n из распределения с плотностью вероятности $q(y|x_n)$. Каким-то образом «натренировав» параметры θ по этому тренировочному набору, можно перейти к проверке, что же получилось. Для этого можно взять второй (проверочный) набор пар $(X_m^{'}, Y_m^{'})|m = 1, \ldots, M$, который формируется аналогично первому, вот только $x_m^{'}$ выбираются из распределения с плотностью $q_1(x)$. Ситуацию, когда $q_0(x) \neq q_1(x)$, называют covariate shift в распределении. При определённых обстоятельствах такого рода изменение может дать серьёзную ошибку на проверочных данных у модели, которая, вообще говоря, очень хорошо работала на тренировочных.

В оригинальной статье, однако, Иоффе и Сегеди приводят понятие именно Internal Covariate Shift (ICS). В ней он определяется как изменение распределения входных данных слоя нейронной сети в результате изменения параметров модели в процессе тренировки. То есть, каждый слой нейронной сети представляет собой некую функцию, переводящую один случайный вектор в другой, на вход случайный вектор приходит из предыдущего слоя нейронной сети, соответственно, в процессе тренировки веса предыдущего слоя меняются - значит может поменяться и распределение векторов, выданных предыдущим слоем. Идея батч-нормализации в том, чтобы погасить этот внутренний ковариационный сдвиг и за счёт этого ускорить тренировку.

Если теоретическое обоснование батч-нормализации выглядит неубедительно, то практические соображения, лежащие в основе метода, описаны авторами в статье очень чётко.

В статье начинается всё с известного метода преобразования случайных векторов, который называется whitening (после применения которого данные становятся похожи на «белый шум») и формально определяется так: whitening - это такое линейное преобразование, которое переводит случайный вектор X в случайный вектор Y (той же размерности), таким образом, чтобы у получившегося вектора Y все компоненты имели нулевое математическое ожидание, были некоррелированы и их дисперсии равнялись единице (иначе говоря, у вектора Y должна быть ковариационная матрица).

Whitening хорошо зарекомендовал себя в том числе и при тренировке нейронных сетей, но обычно, при тренировке нейронных сетей он используется на исходных данных. Иоффе и Сегеди в своей статье резонно замечают, что также хорошо было бы применить нечто похожее и на промежуточных данных, которые приходят на вход слоёв нейронной сети. Однако, возникают две проблемы. Первая, и она же основная, - это крайне дорогое удовольствие, вычислять и применять whitening-преобразование для всех слоёв. Вторая, хотелось бы, чтобы рассматриваемая сеть была дифференцируемой функцией (а иначе оптимизировать недифференцируемую функцию будет достаточно проблематично), но введение whitening-преобразования дифференцируемость разрушит (может и не везде, но и в одном месте будет достаточно). Поэтому предлагается несколько упростить подход и просто нормализовывать данные по каждой компоненте вектора отдельно. То есть, для вектора $X = (x_1, \ldots, x_d)$ посчитать математиче-

ское ожидание и дисперсию для каждой компоненты отдельно и нормализовать его в вектор $\widehat{X} = (\widehat{x}_1, \ldots, \widehat{x}_d)$, используя следующее преобразование:

$$\widehat{x}_k = \frac{(x_k - E(x_k))}{\sqrt{Var(x_k)}}$$

Однако, если просто нормализовать данные, подаваемые на вход слоя (некоторой функции), то сузится возможность того, что слой может представлять. Например, если взять в качестве функции сигмоид, то нормализация данных, подаваемых на вход, приводит в область, где сигмоид почти линейная функция. Это плохо, поэтому надо ввести два дополнительных параметра γ_k и β_k, которые будут тренироваться вместе с остальными параметрами сети, и окончательное преобразование будет выглядеть следующим образом:

$$y_k = \gamma_k \cdot \widehat{x}_k + \beta_k = \gamma_k \cdot \frac{(x_k - E(x_k))}{\sqrt{Var(x_k)}} + \beta_k$$

Стоит отметить ещё и замечание. Если положить $\gamma_k = \sqrt{Var(x_k)}$ и $\beta_k = E(x_k)$, то получится тождественное преобразование, то есть, возможна ситуация, когда вводимая трансформация не изменит ничего вообще.

По итогу, Иоффе и Сегеди определили преобразование $BN_{\gamma,\beta}(x)$, которое добавляется перед каждым слоем нейронной сети (в зависимости от подбора структуры сети возможен вариант, что и не перед каждым). При этом γ, β? тренируемые параметры сети, и их чрезвычайно много, по паре на каждую компоненту вектора на входе каждого слоя сети.

Остаётся ещё пара шагов.

Первый, во время тренировки будем работать не со всеми тренировочными данными разом, а с данными, разделёнными на минибатчи. Поэтому во время тренировки мы нормализуем данные в минибатче, то есть, аппроксимируем математическое ожидание и дисперсии в преобразовании $\widehat{x}$ по каждому минибатчу отдельно.

Второй, когда сеть используется для вывода, обычно нет ни батчей, ни минибатчей. Поэтому среднее и дисперсию сохраняются из тренировки, для этого для каждой компоненты вектора входа каждого слоя запоминаются аппроксимированные математическое ожидание и дисперсия, считая скользящую среднюю математического ожидания и дисперсии по минибатчам в процессе тренировки.

Теперь стоит разобраться, как применять батч-нормализацию. Чаще всего, и полносвязный и свёрточный слой можно представить в виде:

$$z = g(W \cdot x + b)$$

W и b - это параметры линейного преобразования, которые тренируются, $g(\cdot)$? некоторая нелинейная функция (например, сигмоид, $ReLu(\cdot)$ и тому подобное). Необходимо вставить батч-нормализацию перед применением нелинейного преобразования. В таком случае получится:

$$z = g(BN(W \cdot x))$$

Был убран сдвиг b, потому что, когда будет нормализовываться распределение, то b все равно потеряется, а затем его место займёт сдвиг β - параметр батч-нормализации. Также в оригинальной статье Иоффе и Сегеди объясняют, почему батч-нормализация не применяется непосредственно к x. Обычно x - это результат нелинейности с предыдущего слоя, форма распределения x скорее всего меняется в процессе тренировки и ограничение от батч-нормализации не помогут убрать covariate shift.

Чтобы получить максимальный эффект от использования батч-нормализации, в оригинальной статье предлагается ещё несколько изменений в процессе тренировки сети:

1. Увеличить learning rate. Когда в сеть внедрена батч-нормализация, можно увеличить learning rate и таким образом ускорить процесс оптимизации нейронной сети, при этом избежать расходимости. В сетях без BN при применении больших learning rate за счёт увеличения амплитуды градиентов возникает расходимость.
2. Не использовать dropout слои. Батч-нормализация решает в том числе те же задачи, что и dropout слои, поэтому в сетях где мы применяем эту нормализацию, можно убрать dropout слои - это ускоряет процесс тренировки при этом переобучения сети не возникает.
3. Уменьшить вес L_2 регуляризации. L_2 регуляризация обычно используется при тренировке сетей, чтобы избежать переобучения. Иоффе и Сегенди уменьшили в пять раз вес L_2 регуляризации в штрафной функции, при этом точность на валидационных данных увеличилась.

4. Ускорить убывание learning rate. Поскольку при добавлении в сеть батч-нормализации скорость сходимости увеличивается, то можно уменьшать learning rate быстрее. (Обычно используется либо экспоненциальное уменьшение веса, либо ступенчатое, и то и другое регулируется двумя параметрами: во сколько раз и через сколько итераций уменьшить learning rate, при применении батч-нормализации можно сократить число итераций).
5. Более тщательно перемешивать тренировочные данные, чтобы минибатчи не собирались из одних и тех же примеров.

одводя итог по батч-нормализации, можно сказать, что Иоффе и Сегенди предложили методику ускорения сходимости нейронной сети. И эта методика вполне доказала свою эффективность. И более того, она позволяет использовать более широкий интервал метапараметров тренировки и с меньшей аккуратностью подходить к инициализации весов сети [11] [12].

Также при построении свёрточной нейронной сети будут использованы эмбеддинги. Расмотрим подробнее, что это.

В контексте обработки естественного языка (Natural Language Processing, NLP) эмбеддинги (embeddings) представляют собой векторные представления слов или текстовых фрагментов. Эмбеддинги используются для преобразования текстовых данных в числовые векторы, которые могут быть использованы алгоритмами машинного обучения.

Эмбеддинги позволяют модели машинного обучения учитывать семантический и синтаксический контекст слов или текстовых фрагментов. Вместо представления слов в виде разреженных векторов, где каждое измерение соответствует отдельному слову из словаря, эмбеддинги представляют слова в непрерывном пространстве, где близкие по смыслу слова имеют схожие векторные представления.

Обучение эмбеддингов может быть выполнено с использованием различных методов, таких как Word2Vec, GloVe, FastText и других. Эти методы обучаются на больших текстовых корпусах и пытаются уловить семантические и синтаксические свойства слов на основе их соседей в тексте.

Применение эмбеддингов в NLP позволяет моделям машинного обучения улучшить качество работы на задачах, таких как классификация текстов, машинный перевод, определение тональности текста и других. Эмбеддинги позволяют моделям лучше понимать семантическую близость и отношения меж-

ду словами, что помогает уловить более высокий уровень информации в тексте [13] [14].

2 Реализация практической части работы

Практическая работа была выполнена на языке Python, поскольку он поддерживает большое количество библиотек, удобных для использования при реализации машинного обучения, в частности, библиотеки scikit-learn, TensorFlow, Keras, NLTK [15] [16] [17]. Для реализации практической части данной выпускной квалификационной работы был выбран датасет отзывов к товару Alexa, продающемуся на интернет-площадке Amazon. Сам датасет был выбран на сайте kaggle.com [18]. В датасете представлены около 3000 отзывов клиентов Amazon, звёздный рейтинг и дата оставленного отзыва. Был взят англоязычный датасет, поскольку использованная в работе библиотека NLTK плохо поддерживает русский язык. В данной библиотеке есть только корпус русскоязычных стоп-слов. Однако всё остальное поддерживаться не будет.

Также вся работа выполнялась в бесплатной интерактивной облачной среде для работы с кодом Google Colab.

Для начала установим необходимые нам библиотеки Python, как показано на рисунках 4 и 5:

```
import pandas as pd
import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
import re
import string
import sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
import seaborn as sns
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from nltk.stem import WordNetLemmatizer
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from tensorflow.keras.preprocessing.text import Tokenizer
import pad_sequences
from keras.layers import Flatten
from keras.layers import Embedding
```

Рисунок 4 – Установка необходимых библиотек

```
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from gensim.models import Word2Vec
from numpy import asarray
from numpy import zeros
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from nltk.tokenize import RegexpTokenizer
import plotly
import plotly.graph_objs as go
import matplotlib.pyplot as plt
from wordcloud import WordCloud,STOPWORDS
# general imports
import math
from bs4 import BeautifulSoup
import tensorflow as tf
import numpy as np
import skimage
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,f1_score
import missingno as msno
```

Рисунок 5 – Установка необходимых библиотек

После этого загрузим данные, посмотрим их размерность и выведем первые строки, как продемонстрировано на рисунке 6:

Как можно увидеть, размер датасета будет составлять 3150 строк на 5 столбцов.

```
[ ] df = pd.read_csv('amazon_alexa.tsv', sep='\t')

[ ] df.shape

    (3150, 5)

[ ] df.head()
```

	rating	date	variation	verified_reviews	feedback
0	5	31-Jul-18	Charcoal Fabric	Love my Echo!	1
1	5	31-Jul-18	Charcoal Fabric	Loved it!	1
2	4	31-Jul-18	Walnut Finish	Sometimes while playing a game, you can answer...	1
3	5	31-Jul-18	Charcoal Fabric	I have had a lot of fun with this thing. My 4 ...	1
4	5	31-Jul-18	Charcoal Fabric	Music	1

Рисунок 6 – Загрузка данных

Соберём некоторые статистические данные выбранного датасета при помощи функции describe(), как показано на рисунке 7:

```
[ ] df.describe()
```

	rating	feedback
count	3150.000000	3150.000000
mean	4.463175	0.918413
std	1.068506	0.273778
min	1.000000	0.000000
25%	4.000000	1.000000
50%	5.000000	1.000000
75%	5.000000	1.000000
max	5.000000	1.000000

Рисунок 7 – Статистические данные по датасету

Из этой информации в общем случае можно делать выводы, есть ли пропущенные данные в датасете, какой разброс данных, каков их диапазон и как можно разделить на квартили.

В данном случае можно сделать вывод, что пропущенных данных в столбцах нет, поскольку значение параметра count совпадает со значением, полученным ранее при помощи функции shape().

Проверим датафрейм на наличие нулевых значений, как показано на рисунке 8:

```
[ ]  # Find missing values
     print("Colums with Null values: \n",df.isnull().sum())
```

```
Colums with Null values:
 rating              0
date                 0
variation            0
verified_reviews     0
feedback             0
dtype: int64
```

Рисунок 8 – Анализируем датафрейм на нулевые значения

Проверим наличие дубликатов в датафрейме с помощью диаграммы, как показано на рисунке 9:

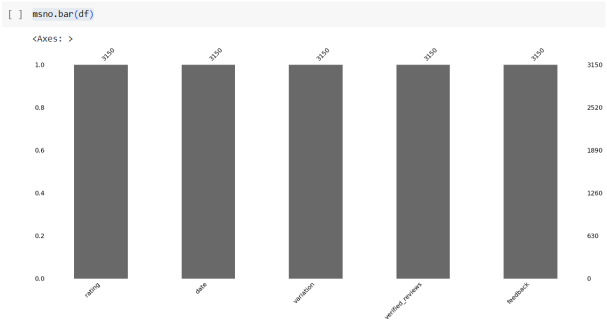

Рисунок 9 – Диаграмма наличия дубликатов

Как видно из диаграммы, дубликатов в датафрейме нет, нулевых значений тоже.

Оценка данных завершена, мы убедились, что нет ни нулевых значений, ни дубликатов в данном датасете, а значит можно перейти к предобработке информации.

Далее начнём очищать текст, на основе которого и будет обучаться наш алгоритм. Для этого необходимо привести текст отзывов к одному регистру, убрать стоп-слова и знаки препинания, лемматизировать текст и составить мешок слов.

Начнём с приведения текста к одному регистру, как показано на рисунке 10:

```
[ ] #Step 1: transform to lowercase
    df['review_lw'] = df['verified_reviews'].str.lower()
    df[['verified_reviews','review_lw']].head(10)
```

	verified_reviews	review_lw
0	Love my Echo!	love my echo!
1	Loved it!	loved it!
2	Sometimes while playing a game, you can answer...	sometimes while playing a game, you can answer...
3	I have had a lot of fun with this thing. My 4 ...	i have had a lot of fun with this thing. my 4 ...
4	Music	music
5	I received the echo as a gift. I needed anothe...	i received the echo as a gift. i needed anothe...
6	Without having a cellphone, I cannot use many ...	without having a cellphone, i cannot use many ...
7	I think this is the 5th one I've purchased. I'...	i think this is the 5th one i've purchased. i'...
8	looks great	looks great
9	Love it! I've listened to songs I haven't hear...	love it! i've listened to songs i haven't hear...

Рисунок 10 – Приведение текста отзывов к одному регистру

Затем требуется убрать стоп-слова и пунктуацию, что приведено на рисунках 11 и 12:

```
[ ]  def transform_text(s):

        # remove html
        html=re.compile(r'<.*?>')
        s = html.sub(r'',s)

        # remove numbers
        s = re.sub(r'\d+', '', s)

        # remove punctuation
        # remove stopwords
        tokens = nltk.word_tokenize(s)

        new_string = []
        for w in tokens:
            # remove words with len = 2 AND stopwords
            if len(w) > 2 and w not in sw:
                new_string.append(w)

        s = ' '.join(new_string)
        s = s.strip()

        exclude = set(string.punctuation)
        s = ''.join(ch for ch in s if ch not in exclude)

        return s.strip()
```

Рисунок 11 – Программная реализация очистки текста

```
[ ] df['review_sw'] = df['review_lw'].apply(transform_text)
    df[['verified_reviews','review_lw', 'review_sw']].head(10)
```

	verified_reviews	review_lw	review_sw
0	Love my Echo!	love my echo!	love echo
1	Loved it!	loved it!	loved
2	Sometimes while playing a game, you can answer...	sometimes while playing a game, you can answer...	sometimes playing game answer question correct...
3	I have had a lot of fun with this thing. My 4 ...	i have had a lot of fun with this thing. my 4 ...	lot fun thing old learns dinosaurs control lig...
4	Music	music	music
5	I received the echo as a gift. I needed anothe...	i received the echo as a gift. i needed anothe...	received echo gift needed another bluetooth so...
6	Without having a cellphone, I cannot use many ...	without having a cellphone, i cannot use many ...	without cellphone use many features ipad see u...
7	I think this is the 5th one I've purchased. I'...	i think this is the 5th one i've purchased. i'...	think one ve purchased working getting one eve...
8	looks great	looks great	looks great
9	Love it! I've listened to songs I haven't hear...	love it! i've listened to songs i haven't hear...	love listened songs heard since childhood get ...

Рисунок 12 – Очистка содержания датафрейма от стоп-слов и пунктуации

Далее проведём лемматизацию текста, как представлено на рисунке 13:

```
[ ] def lemmatizer_text(s):
        tokens = nltk.word_tokenize(s)

        new_string = []
        for w in tokens:
            lem = lemmatizer.lemmatize(w, pos="v")
            # exclude if lenght of lemma is smaller than 2
            if len(lem) > 2:
                new_string.append(lem)

        s = ' '.join(new_string)
        return s.strip()
```

```
[ ] df['review_lm'] = df['review_sw'].apply(lemmatizer_text)
    df[['verified_reviews','review_lw', 'review_sw', 'review_lm']].head(20)
```

	verified_reviews	review_lw	review_sw	review_lm
0	Love my Echo!	love my echo!	love echo	love echo
1	Loved it!	loved it!	loved	love
2	Sometimes while playing a game, you can answer...	sometimes while playing a game, you can answer...	sometimes playing game answer question correct...	sometimes play game answer question correctly ...
3	I have had a lot of fun with this thing. My 4 ...	i have had a lot of fun with this thing. my 4 ...	lot fun thing old learns dinosaurs control lig...	lot fun thing old learn dinosaurs control ligh...
4	Music	music	music	music
5	I received the echo as a gift. I needed anothe...	i received the echo as a gift. i needed anothe...	received echo gift needed another bluetooth so...	receive echo gift need another bluetooth somet...

Рисунок 13 – Лемматизация текста

Создадим мешок слов и уберём редкие слова из датасета, как показано на рисунках 14 и 15:

```
[ ]  # Creating Frequency
     text_series = pd.Series(text.split())
     freq_comm = text_series.value_counts()
     freq_comm
```

```
love           1060
echo            858
great           720
use             681
work            648
                ...
spectacular       1
feed              1
argument          1
consumer          1
adept             1
Length: 3222, dtype: int64
```

```
[ ]  rare_words = freq_comm[-1266:-1]
     'rattlecrackle' in rare_words
```

```
True
```

Рисунок 14 – Создаём мешок слов

```
[ ]  rare_words
```

```
mint          1
revise        1
successful    1
pray          1
devicei       1
             ..
stellar       1
spectacular   1
feed          1
argument      1
consumer      1
Length: 1265, dtype: int64
```

```
[ ]  # Removing 1266 rare occuring words
     df['review_lm'] = df['review_lm'].apply(lambda x: ' '.join([word for word in x.split() if word not in rare_word
     df['review_lm'].sample(5)
```

```
19      like original echo shorter greater fabriccolor...
876                                          alexa rock
2284    work great lag much better gen devices use wou...
432                                             thank
2202                              perfect work like charm
Name: review_lm, dtype: object
```

Рисунок 15 – Убираем редкие слова

Создадим облако слов для положительных отзывов, как представлено на рисунках 16 и 17:

Создадим так же облако слов для негативных отзывов. В этом облаке можно заметить такие характерные слова, как «bad», «horrible», «terrible» и другие, что показано на рисунке 18:

```
[ ]  df_pos = df[df['feedback']==1]['review_lm']

     wordcloud1 = WordCloud(stopwords=STOPWORDS,
                            background_color='white',
                            width=2500,
                            height=2000
                            ).generate(" ".join(df_pos))

     plt.figure(1,figsize=(12, 12))
     plt.imshow(wordcloud1)
     plt.axis('off')
     plt.show()
```

Рисунок 16 – Программная реализация создания облака слов

Рисунок 17 – Облако слов для положительных отзывов

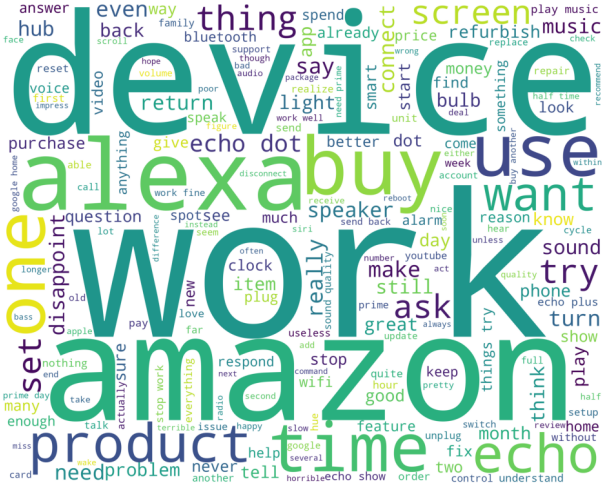

Рисунок 18 – Облако слов для негативных отзывов

Составим частоту встречаемости слов в положительных и негативных отзывах, как показано на рисунках 19, 20 и 21:

```
def plot_ngram(sentiment, n):
    print("sentiment",sentiment)
    temp_df = df[df['feedback'] == sentiment]

    word_vectorizer = CountVectorizer(ngram_range=(n, n), analyzer='word')
    sparse_matrix = word_vectorizer.fit_transform(temp_df['review_lm'])

    frequencies = sum(sparse_matrix).toarray()[0]

    return pd.DataFrame(frequencies, index=word_vectorizer.get_feature_names_out(), columns=['frequency'])\
            .sort_values(by='frequency', ascending=False) \
            .reset_index() \
            .head(10)
```

Рисунок 19 – Программная реализация вывода встречаемости слов

```
# For Feedback=1
plot_ngram(1, 1)
```

```
sentiment 1
```

	index	frequency
0	love	1051
1	echo	766
2	great	702
3	use	633
4	alexa	572
5	work	546
6	music	505
7	like	483
8	get	429
9	sound	390

Рисунок 20 – Частота встречаемости слов в положительных отзывах

```
plot_ngram(0, 1)
```

sentiment 0

	index	frequency
0	work	102
1	echo	92
2	get	68
3	amazon	61
4	would	58
5	buy	57
6	alexa	54
7	device	54
8	time	50
9	use	48

Рисунок 21 – Частота встречаемости слов в негативных отзывах

Разделим наш датасет на тренировочную и тестовую выборки в процентном соотношении 70 к 30 соответственно, как представлено на рисунке 22:

```
# Train dataset
pos_train = df[df['feedback']==1][['review_lm', 'feedback']].head(2025)
neg_train = df[df['feedback']==0][['review_lm', 'feedback']].head(180)


# Test dataset
pos_test = df[df['feedback']==1][['review_lm', 'feedback']].tail(868)
neg_test = df[df['feedback']==0][['review_lm', 'feedback']].tail(77)
```

```
# put all toghether
train_df = pd.concat([pos_train, neg_train]).sample(frac = 1).reset_index(drop=True)
test_df = pd.concat([pos_test, neg_test]).sample(frac = 1).reset_index(drop=True)
```

Рисунок 22 – Деление датасета на тренировочную и тестовую выборки

Векторизируем слова нашего датасета. Векторизация - это процесс обработки естественного языка. В процессе используются языковые модели для сопоставления слов с пространством векторов. Векторное пространство представляет каждое слово с помощью вектора. Этот метод также позволяет словам

с аналогичным значением иметь похожие представления [19] [20]. Для векторизации применим функцию CountVectorizer библиотеки scikit-learn, как показано на рисунке 23:

```
vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train)
X_train_onehot = vectorizer.transform(X_train)
X_test_onehot = vectorizer.transform(X_test)

#print shape post encoding
print(X_train_onehot.shape)
print(X_test_onehot.shape)
```

```
(2205, 1760)
(945, 1760)
```

Рисунок 23 – Проведём векторизацию слов

Реализуем функцию обучения классификатора classifier() и оценки его производительности на тестовых данных, в качестве классификатора будем использовать логистическую регрессию, как показано на рисунке 24:

```
# Generic function for model building
def fit_and_test(classifier, X_train, y_train, X_test, y_test, only_return_accuracy=False):
  classifier.fit(X_train, y_train)
  y_hat = classifier.predict(X_test)
  print('accuracy:', accuracy_score(y_test, y_hat))
  if not only_return_accuracy:
    print('f1_score:', f1_score(y_test, y_hat))
```

```
#Logistic Regression
#grid search over regularisation hyperparameter 'c'
for c in [0.01, 0.02, 0.05, 0.25, 0.5, 0.75, 1,]:
  lr = LogisticRegression(C=c, max_iter=1000) # 92.91%
  print (f'At C = {c}:-', end=' ')
  fit_and_test(lr, X_train_onehot, y_train, X_test_onehot, y_test, True)
```

```
At C = 0.01:- accuracy: 0.9185185185185185
At C = 0.02:- accuracy: 0.9185185185185185
At C = 0.05:- accuracy: 0.9185185185185185
At C = 0.25:- accuracy: 0.9227513227513228
At C = 0.5:- accuracy: 0.9227513227513228
At C = 0.75:- accuracy: 0.926984126984127
At C = 1:- accuracy: 0.9291005291005291
```

Рисунок 24 – Реализация классификатора Logistic regression

Как видно из результата работы кода, при значении гиперпараметра C = 1, алгоритм выдаёт наилучшие значения, запустим его от этого значения гиперпараметра ещё раз, как показано на рисунке 25:

```
#Make an instance of the model
logistic = LogisticRegression(C=1,max_iter=10000)
#fitting the values for x and y
logistic.fit(X_train_onehot,y_train)
```

```
LogisticRegression
LogisticRegression(C=1, max_iter=10000)
```

Рисунок 25 – Запускаем логистическую регрессию при C = 1

Построим тепловую карту матрицы ошибок, визуализирующую отношение между истинным и предсказанным значениями классов модели. Она позволяет оценить наглядно, в каких случаях модель делает верные или неверные предсказания для каждого класса. Программная реализация представлена на рисунках 26 и 27:

```
#predictions from test data
prediction = logistic.predict(X_test_onehot)
```

```
###### confusion matrix  starts ######
from sklearn.metrics import accuracy_score, confusion_matrix
cm_lgr1 = confusion_matrix(y_test,prediction)
names = np.unique(prediction)
sns.heatmap(cm_lgr1, square=True, annot=True, cbar=False,xticklabels=names, yticklabels=names, cmap="YlGnBu" ,f
plt.xlabel('Truth')
plt.ylabel('Predicted')
```

Рисунок 26 – Программная реализация построения тепловой карты

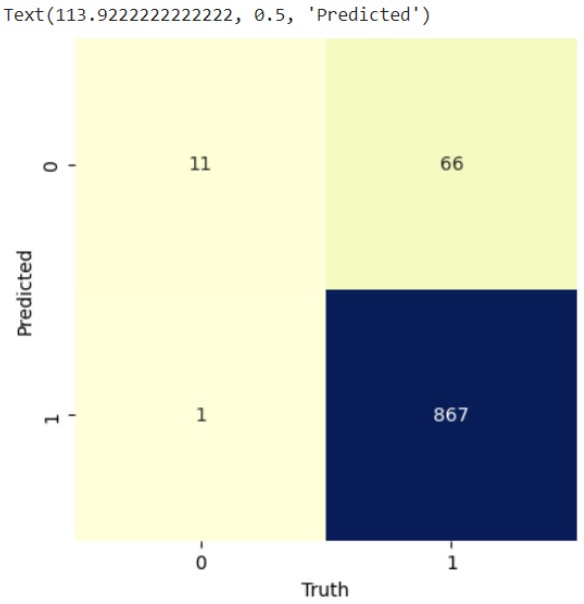

Рисунок 27 – Тепловая карта матрицы ошибок

Подсчитаем точность логистической регрессии, как показано на рисунке 28:

```
#calculating the accuracy
accuracy_score = accuracy_score(y_test,prediction)
print("Accuracy of  Logistic Regression :",accuracy_score)
```

```
Accuracy of  Logistic Regression : 0.9291005291005291
```

Рисунок 28 – Подсчёт точности логистической регрессии

Можем сказать, что простая логистическая регрессия достигает точности в 93% при значении гиперпараметра C = 1 (только 67 примеров из 945 были определены ошибочно).

Составим график частоты встречаемости n-грамм, как показано на рисунках 29 и 30:

```
################ Print n-grams and frequency starts #################
vectorizer = CountVectorizer()
vect_texts = vectorizer.fit_transform(list(df['review_lm']))

all_ngrams = vectorizer.get_feature_names_out()
#print(all_ngrams[:5])
num_ngrams = min(50, len(all_ngrams))
all_counts = vect_texts.sum(axis=0).tolist()[0]

all_ngrams, all_counts = zip(*[(n, c) for c, n in sorted(zip(all_counts, all_ngrams), reverse=True)])
ngrams = all_ngrams[:num_ngrams]
counts = all_counts[:num_ngrams]

idx = np.arange(num_ngrams)
```

```
# Let's plot a frequency distribution plot for the most seen words
plt.figure(figsize=(18, 18))
plt.bar(idx, counts, width=0.8)
plt.xlabel('N-grams')
plt.ylabel('Frequencies')
plt.title('Frequency distribution of ngrams')
plt.xticks(idx, ngrams, rotation=45)
plt.show()
```

Рисунок 29 – Программная реализация построения графика

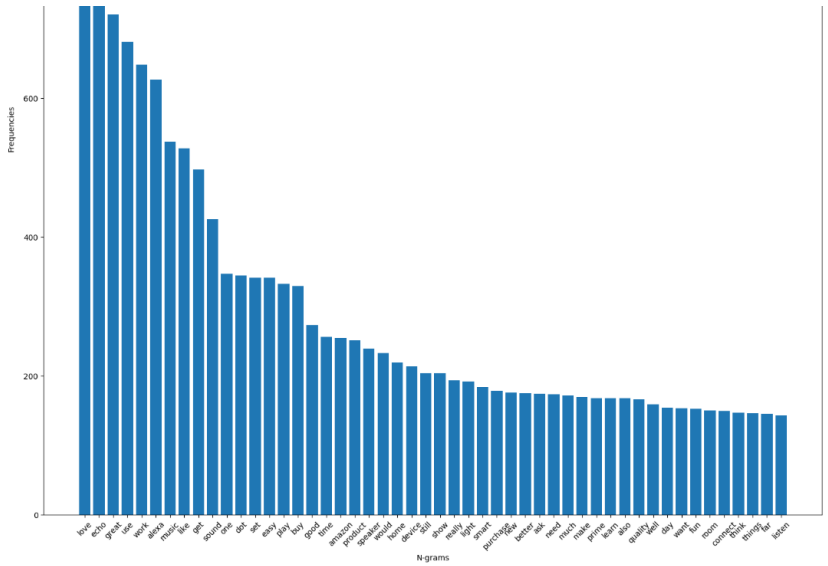

Рисунок 30 – График частоты встречаемости n-грамм

Реализуем модель глубокого обучения при помощи библиотеки Keras с ис-

пользованием батч-нормализации и обучим её, как показано на рисунках 31, 32 и 33:

```
model = tf.keras.models.Sequential([

    tf.keras.layers.Dense(256, input_shape=(n_words1,), activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),


    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model.compile(loss='binary_crossentropy',
              optimizer=Adam(0.0001),
              metrics=['acc'])
```

Рисунок 31 – Определим структуру модели нейронной сети и опишем компиляцию этой модели

```
callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)]


history = model.fit(Xtrain, y_train,
                    epochs=20,
                    validation_data=(Xtest, y_test,),
                    verbose=1,
                    callbacks=callbacks,
                   )
```

Рисунок 32 – Программная реализация обучения модели с использованием callbacks и сохранением истории обучения в history

```
Epoch 1/20
69/69 [==============================] - 9s 9ms/step - loss: 0.9695 - acc: 0.5224 - val_loss: 0.6158 - val_acc: 0.9079
Epoch 2/20
69/69 [==============================] - 0s 6ms/step - loss: 0.8576 - acc: 0.5755 - val_loss: 0.5548 - val_acc: 0.9185
Epoch 3/20
69/69 [==============================] - 1s 8ms/step - loss: 0.7808 - acc: 0.6181 - val_loss: 0.5093 - val_acc: 0.9196
Epoch 4/20
69/69 [==============================] - 1s 8ms/step - loss: 0.7120 - acc: 0.6440 - val_loss: 0.4747 - val_acc: 0.9175
Epoch 5/20
69/69 [==============================] - 0s 6ms/step - loss: 0.6614 - acc: 0.6658 - val_loss: 0.4519 - val_acc: 0.9185
Epoch 6/20
69/69 [==============================] - 1s 8ms/step - loss: 0.6331 - acc: 0.6834 - val_loss: 0.4381 - val_acc: 0.9122
Epoch 7/20
69/69 [==============================] - 1s 9ms/step - loss: 0.5910 - acc: 0.7229 - val_loss: 0.4232 - val_acc: 0.9069
Epoch 8/20
69/69 [==============================] - 1s 10ms/step - loss: 0.5349 - acc: 0.7478 - val_loss: 0.4154 - val_acc: 0.9005
Epoch 9/20
69/69 [==============================] - 1s 10ms/step - loss: 0.5166 - acc: 0.7683 - val_loss: 0.4090 - val_acc: 0.8921
Epoch 10/20
69/69 [==============================] - 1s 10ms/step - loss: 0.4926 - acc: 0.7823 - val_loss: 0.4011 - val_acc: 0.8921
Epoch 11/20
69/69 [==============================] - 1s 11ms/step - loss: 0.4476 - acc: 0.8104 - val_loss: 0.3917 - val_acc: 0.8921
Epoch 12/20
69/69 [==============================] - 1s 11ms/step - loss: 0.4325 - acc: 0.8195 - val_loss: 0.3848 - val_acc: 0.8995
Epoch 13/20
69/69 [==============================] - 0s 7ms/step - loss: 0.3972 - acc: 0.8463 - val_loss: 0.3684 - val_acc: 0.8995
Epoch 14/20
69/69 [==============================] - 0s 6ms/step - loss: 0.3869 - acc: 0.8422 - val_loss: 0.3563 - val_acc: 0.9037
Epoch 15/20
69/69 [==============================] - 1s 7ms/step - loss: 0.3622 - acc: 0.8649 - val_loss: 0.3401 - val_acc: 0.9090
Epoch 16/20
69/69 [==============================] - 0s 6ms/step - loss: 0.3363 - acc: 0.8753 - val_loss: 0.3333 - val_acc: 0.9111
Epoch 17/20
69/69 [==============================] - 0s 6ms/step - loss: 0.3190 - acc: 0.8853 - val_loss: 0.3250 - val_acc: 0.9164
Epoch 18/20
69/69 [==============================] - 0s 6ms/step - loss: 0.2907 - acc: 0.9025 - val_loss: 0.3095 - val_acc: 0.9175
Epoch 19/20
69/69 [==============================] - 0s 6ms/step - loss: 0.2794 - acc: 0.9120 - val_loss: 0.2990 - val_acc: 0.9206
Epoch 20/20
69/69 [==============================] - 0s 6ms/step - loss: 0.2638 - acc: 0.9138 - val_loss: 0.2898 - val_acc: 0.9249
```

Рисунок 33 – Результат выполнения программы обучения с использованием callbacks

Реализуем предсказания для тестовой выборки X_test с использованием модели классификатора clf_object и вычислим метрику accuracy_score(), как показано на рисунке 34:

```
# Function to make predictions
def prediction(X_test, clf_object):

    y_pred = clf_object.predict(X_test)
    return y_pred
```

```
predictions_keras = prediction(Xtest, model)
```

```
30/30 [==============================] - 0s 2ms/step
```

```
pred_ann = [ 1 if y>=0.5 else 0 for y in predictions_keras]
```

```
#calculating the accuracy
accuracy_score = accuracy_score(y_test, pred_ann)
print("Accuracy score with Keras:",accuracy_score)
```

```
Accuracy score with Keras: 0.9248677248677248
```

Рисунок 34 – Программная реализация предсказания и вычисления метрики accuracy_score()

Построим графики для визуализации метрик обучения и валидации (точности и потерь), как показано на рисунках 35, 36 и 37:

```
def plot_history(history):
    accuracy = history.history['acc']
    val_accuracy = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(1,len(accuracy) + 1)

    # Plot accuracy
    plt.figure(1)
    plt.plot(epochs, accuracy, 'b', label='Training accuracy')
    plt.plot(epochs, val_accuracy, 'g', label='Validation accuracy')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # Plot loss
    plt.figure(2)
    plt.plot(epochs, loss, 'b', label='Training loss')
    plt.plot(epochs, val_loss, 'g', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

plot_history(history)
```

Рисунок 35 – Программная реализация построения графиков

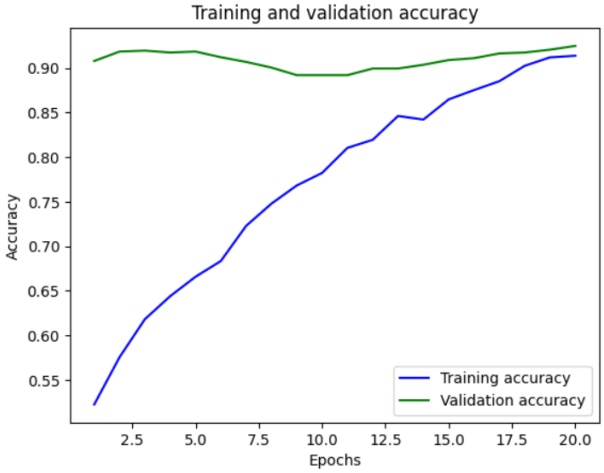

Рисунок 36 – График точности

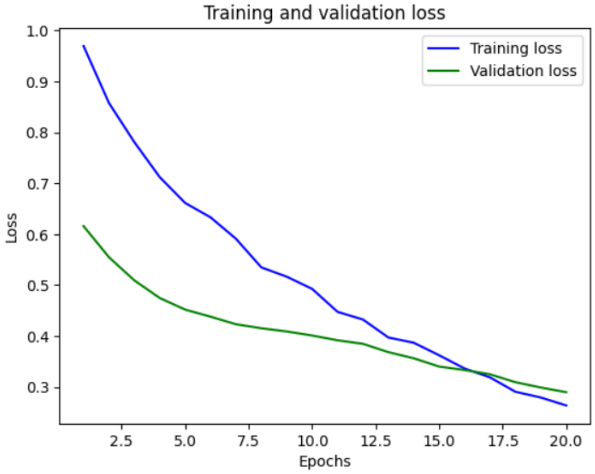

Рисунок 37 – График потерь

Построим тепловую карту матрицы ошибок для данной модели, как показано на рисунке 38:

Рисунок 38 – Тепловая карта матрицы ошибок

Данная модель определила правильно 854 отзыва и ошиблась в 71. Модель

на основе логистической регрессии справилась лучше с данной задачей.

Построим свёрточную нейронную сеть с использованием эмбеддингов, как показано на рисунке 39:

```
model = Sequential()

model.add(Embedding(vocab_size, 100, input_length=max_length))
model.add(Conv1D(filters=32, kernel_size=8, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 230, 100)          176300

 conv1d (Conv1D)             (None, 223, 32)           25632

 max_pooling1d (MaxPooling1D  (None, 111, 32)          0
 )

 flatten (Flatten)           (None, 3552)              0

 dense_3 (Dense)             (None, 10)                35530

 dense_4 (Dense)             (None, 1)                 11

=================================================================
Total params: 237,473
Trainable params: 237,473
Non-trainable params: 0
_________________________________________________________________
None
```

Рисунок 39 – Построение CNN с использованием эмбеддингов

Вычислим точность построенной модели, как представлено на рисунке 40:

```
# evaluate
loss, acc = model.evaluate(Xtest, y_test, verbose=0)
print('Test Accuracy: %f' % (acc*100))
```

```
Test Accuracy: 92.910051
```

Рисунок 40 – Вычисление метрики точности CNN

Данная реализованная модель работает с точностью в 93%. Это довольно-таки неплохой результат из всех трёх. Можно сделать вывод, что из логистической регрессии, модели глубокого обучения с использованием батч-нормализации на основе библиотеки Keras и свёрточной нейронной сети с использованием эмбеддингов лучше всего отработала модель свёрточной нейронной сети, ближе всего к её результату был получен результат модели логистической регрессии и худший результат был у модели глубокого обучения на основе библиотеки Keras.

Однако, если изменить модель глубокого обучения с использованием фреймворка Keras, то можно улучшить результат до 94%. В предыдущей модели содержались следующие слои:

1. Embedding слой: выполняет преобразование индексов слов в плотные векторные представления (эмбеддинги) размерности 100.
2. Conv1D слой: выполняет свертку одномерного входного сигнала с фильтрами размера 8 и активацией ReLU.
3. MaxPooling1D слой: выполняет операцию максимальной пулинга с размером пула 2.
4. Flatten слой: преобразует выходные данные в плоский вектор.
5. Dense слой: полносвязный слой с 10 нейронами и активацией ReLU.
6. Dense слой: полносвязный слой с 1 нейроном и активацией sigmoid.

Изменим состав модели на следующий:

1. Embedding слой: выполняет преобразование индексов слов в плотные векторные представления (эмбеддинги) размерности 100.
2. Dropout слой: применяет регуляризацию Dropout с коэффициентом 0.2, чтобы случайным образом обнулить 20% входных единиц на каждом обновлении во время обучения.
3. LSTM слой: рекуррентный слой LSTM с 200 скрытыми единицами.
4. Dropout слой: применяет регуляризацию Dropout с коэффициентом 0.2.
5. Dense слой: полносвязный слой с 1 нейроном и активацией sigmoid.

В отличие от предыдущей модели, в новой будут использоваться рекуррентные слои для учёта последовательности данных вместо свёрточных слоёв для обнаружения локальных паттернов в данных.

Реализуем новую модель, как показано на рисунке 41:

```
model = Sequential([
    Embedding(vocab_size,100,input_length=maxlen),
    Dropout(0.2),
    LSTM(200),
    Dropout(0.2),
    Dense(1,activation='sigmoid')
])
```

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['acc'])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 30, 100)           382600

 dropout (Dropout)           (None, 30, 100)           0

 lstm (LSTM)                 (None, 200)               240800

 dropout_1 (Dropout)         (None, 200)               0

 dense (Dense)               (None, 1)                 201

=================================================================
Total params: 623,601
Trainable params: 623,601
Non-trainable params: 0
```

Рисунок 41 – Программная реализация новой модели

И обучим её, как показано на рисунке 42:

```
history  = model.fit(x_train,y_train,epochs=10,batch_size=32,validation_data=(x_test,y_test))
```

```
Epoch 1/10
89/89 [==============================] - 26s 192ms/step - loss: 0.2999 - acc: 0.9122 - val_loss: 0.2535 - val_acc: 0.9175
Epoch 2/10
89/89 [==============================] - 9s 98ms/step - loss: 0.1722 - acc: 0.9411 - val_loss: 0.1514 - val_acc: 0.9333
Epoch 3/10
89/89 [==============================] - 4s 44ms/step - loss: 0.0813 - acc: 0.9746 - val_loss: 0.1577 - val_acc: 0.9333
Epoch 4/10
89/89 [==============================] - 4s 49ms/step - loss: 0.0583 - acc: 0.9810 - val_loss: 0.2100 - val_acc: 0.9397
Epoch 5/10
89/89 [==============================] - 2s 23ms/step - loss: 0.0409 - acc: 0.9880 - val_loss: 0.2668 - val_acc: 0.9460
Epoch 6/10
89/89 [==============================] - 3s 30ms/step - loss: 0.0258 - acc: 0.9926 - val_loss: 0.2291 - val_acc: 0.9429
Epoch 7/10
89/89 [==============================] - 2s 22ms/step - loss: 0.0231 - acc: 0.9933 - val_loss: 0.2901 - val_acc: 0.9460
Epoch 8/10
89/89 [==============================] - 2s 19ms/step - loss: 0.0206 - acc: 0.9933 - val_loss: 0.2535 - val_acc: 0.9429
Epoch 9/10
89/89 [==============================] - 2s 18ms/step - loss: 0.0182 - acc: 0.9933 - val_loss: 0.2869 - val_acc: 0.9429
Epoch 10/10
89/89 [==============================] - 2s 26ms/step - loss: 0.0181 - acc: 0.9940 - val_loss: 0.2891 - val_acc: 0.9397
```

Рисунок 42 – Обучение модели

Вычислим метрику точности для новой модели, как на рисунке 43:

```
loss,acc =  model.evaluate(x_test,y_test)
acc
```

```
10/10 [==============================] - 0s 4ms/step - loss: 0.2891 - acc: 0.9397
0.9396825432777405
```

Рисунок 43 – Вычисление точности новой модели

Попробуем реализовать предсказание на основе новой модели для случайно составленного отзыва. Пусть это будет негативный отзыв, как продемонстрировано на рисунке 44:

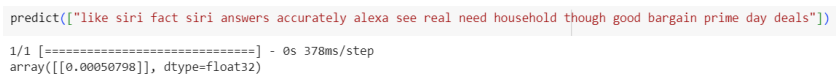

Рисунок 44 – Предсказание принадлежности самостоятельно сгенерированного отзыва на основе обученной модели

Как видно из рисунка, модель отнесла отзыв к негативным. И это действительно так. Модель работает корректно с 94% точностью.

ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе был проведён анализ мнений с использованием методов машинного обучения на основе датасета отзывов с сайта Амазон. В процессе исследования были реализованы несколько моделей машинного обучения с разной точностью.

Вначале был проведён предварительный анализ датасета, включающий обработку текстовых данных, удаление лишних символов, токенизацию и векторизацию текстов. Затем было проведено разделение данных на обучающую и тестовую выборки с использованием функции train_test_split, учитывая стратификацию для сохранения баланса классов.

Для обучения моделей были применены различные алгоритмы машинного обучения, включая логистическую регрессию, модель глубокого обучения с использованием библиотеки Keras и нейронные сети. Каждая модель была обучена на обучающей выборке и оценена на тестовой выборке с использованием метрик точности.

Результаты экспериментов показали, что различные модели машинного обучения демонстрируют разную точность в анализе мнений. Логистическая регрессия показала хорошую точность, достигая практически 93%, в то время как модель на основе Keras достигла 92% точности. Свёрточные нейронные сети также показали неплохие результаты, достигая 93% точности.

На второй модели с использованием Keras, где свёрточные слои были заменены на рекуррентные слои, была достигнута наивысшая в данной работе точность в 94%.

Таким образом, на основе проведённого исследования можно сделать вывод, что анализ мнений методами машинного обучения на основе датасета отзывов с сайта Амазон является эффективным инструментом для определения тональности отзывов. Однако, точность классификации может различаться в зависимости от выбранной модели машинного обучения. Результаты работы могут быть использованы для принятия решений в области управления качеством продуктов и услуг, а также для улучшения взаимодействия с клиентами. К примеру, возможна реализация оценки негативного отзыва и выяснения причины недовольства клиента с дальнейшей рекомендацией ему по исправлению причинённых неприятностей. Допустим, рекомендация купить схожий по цене и предназначению продукт лучшего качества, связь с техническим специалистом

или советы по исправлению недочётов уже приобретённого продукта.

Дальнейшие исследования могут быть направлены на улучшение точности моделей, включая оптимизацию параметров, использование более сложных архитектур нейронных сетей и обработку текста с использованием методов обработки естественного языка. Также можно провести анализ на других датасетах и сравнить результаты с данной работой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Котельников Е.В. Клековкина М.В. Автоматический анализ тональности текстов на основе методов машинного обучения [Электронный ресурс] URL: https://www.dialog-21.ru/media/1380/105.pdf (дата обращения: 02.05.2023) - Яз. рус.

2 Рашка С. Python и машинное обучение / пер. с англ. А. В. Логунова. ? М.: ДМК Пресс, 2017. ? 418 с.: ил.

3 Алгоритмы бинарной классификации в машинном обучении. [Электронный ресурс] URL: https://biconsult.ru/products/algoritmy-binarnoy-klassifikacii-v-mashinnom-obuchenii (дата обращения: 06.05.2023) - Яз. рус.

4 Бинарная классификация (Binary classification) [Электронный ресурс] URL: https://wiki.loginom.ru/articles/binary-classification.html (дата обращения: 03.05.2023) - Яз. рус.

5 Самигулин Т.Р., Джурабаев А.Э.У. Анализ тональности текста методами машинного обучения [Электронный ресурс] URL: https://cyberleninka.ru/article/n/analiz-tonalnosti-teksta-metodami-mashinnogo-obucheniya/viewer (дата обращения: 04.05.2023) - Яз. рус.

6 Метод наименьших квадратов (Least-Squares method) [Электронный ресурс] URL: https://wiki.loginom.ru/articles/least-squares-method.html (дата обращения: 03.05.2023) - Яз. рус.

7 Генеральная совокупность (General population) [Электронный ресурс] URL: https://wiki.loginom.ru/articles/general-population.html (дата обращения: 03.05.2023) - Яз. рус.

8 Логистическая регрессия и ROC-анализ ? математический аппарат [Электронный ресурс] URL: https://loginom.ru/blog/logistic-regression-roc-auc (дата обращения: 04.05.2023) - Яз. рус.

9 Воронцов К. В. Метрические методы классификации и регрессии [Электронный ресурс] URL: https://www.youtube.com/watch?v=GyOxB2itxncab_channel=MachineLearningPhyste (дата обращения: 05.03.2023) - Яз. рус.

10 Derrick Mwiti. Convolutional Neural Network for Sentence Classification [Электронный ресурс] URL: https://cnvrg.io/cnn-sentence-classification/ (дата обращения: 21.05.2023) - Яз. англ.

11 Batch Normalization. Основы. [Электронный ресурс] URL: http://vbystricky.ru/2020/05/batch_normalization.html (дата обращения: 22.05.2023) - Яз. рус.

12 S. Ioffe, C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Электронный ресурс] URL: https://arxiv.org/abs/1502.03167 (дата обращения: 22.05.2023) - Яз. англ.

13 Что такое эмбеддинги и как они помогают искусственному интеллекту понять мир людей [Электронный ресурс] URL: https://www.nkj.ru/open/36052/ (дата обращения: 22.05.2023) - Яз. рус.

14 ML: Embedding слов [Электронный ресурс] URL: https://qudata.com/ml/ru/NN_Embedding.html (дата обращения: 23.05.2023) - Яз. рус.

15 Keras: описание и особенности [Электронный ресурс] URL: https://learn.microsoft.com/ru-ru/azure/machine-learning/component-reference/convert-word-to-vector (дата обращения: 12.05.2023) - Яз. рус.

16 Natural Language Processing With Python's NLTK Package [Электронный ресурс] URL: https://realpython.com/nltk-nlp-python/ (дата обращения: 12.05.2023) - Яз. англ.

17 scikit-learn Tutorials [Электронный ресурс] URL: https://scikit-learn.org/stable/tutorial/index.html (дата обращения: 12.05.2023) - Яз. англ.

18 Amazon Alexa Reviews [Электронный ресурс] URL: https://www.kaggle.com/datasets/sid321axn/amazon-alexa-reviews?select=amazon_alexa.tsv (дата обращения: 10.05.2023) - Яз. англ.

19 Компонент преобразования слов в векторы [Электронный ресурс] URL: https://learn.microsoft.com/ru-ru/azure/machine-learning/component-

reference/convert-word-to-vector (дата обращения: 12.05.2023) - Яз. рус.

20 Yorko / mlcourse.ai [Электронный ресурс] URL: (дата обращения: 18.05.2023) - Яз. рус.

ПРИЛОЖЕНИЕ А

Программная реализация модели с 93% точностью

```
# -*- coding: utf-8 -*-
"""Диплом.ipynb
Automatically generated by Colaboratory.
Original file is located at
https://colab.research.google.com/drive/1aI8OdY8PtnBzdc2oV6iBDYTyKi-ggZc

"""
pip install opendatasets
! pip install kaggle

import opendatasets as od

from google.colab import drive
drive.mount('/content/drive')

import os
os.environ['KAGGLE_CONFIG_DIR'] = "/content/drive/MyDrive/Kaggle"

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/drive/MyDrive/Kaggle/
pwd
pip install pad_sequences

import pandas as pd
import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
import re
import string
import sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
import seaborn as sns
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from nltk.stem import WordNetLemmatizer
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from tensorflow.keras.preprocessing.text import Tokenizer
import pad_sequences
from keras.layers import Flatten
from keras.layers import Embedding
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from gensim.models import Word2Vec
from numpy import asarray
from numpy import zeros
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from nltk.tokenize import RegexpTokenizer
import plotly
```

```
import plotly.graph_objs as go
import matplotlib.pyplot as plt
from wordcloud import WordCloud,STOPWORDS

# general imports
import math
from bs4 import BeautifulSoup
import tensorflow as tf
import numpy as np
import skimage
from sklearn.metrics import classification_report,confusion_matrix,accur
import missingno as msno
od.download("https://www.kaggle.com/datasets/sid321axn/amazon-alexa-revi
os.listdir()

# Commented out IPython magic to ensure Python compatibility.

# %cd ./amazon-alexa-reviews/

pwd
ls
"""загружаем данные"""

df = pd.read_csv('amazon_alexa.tsv', sep='\t')
df.shape
df.head()
df.describe()

#Duplicate Analysis
cols=df.columns
#print(list(cols))
dups_df = df.pivot_table(index=list(cols), aggfunc='size')
print (dups_df)
```

```
#Explore Categorical Variables
cat_col = ['rating','feedback']
plt.figure(figsize=(10, 12))
count = 1
for cols in cat_col:
plt.subplot(2, 2, count)
df[cols].value_counts().plot.pie(shadow=True,autopct='%1.1f%%')
count +=1
"""Дубликатов не обнаружено"""

# Find missing values
print("Colums with Null values: \n",df.isnull().sum())
"""Нулевых значений также нет"""

msno.bar(df)
#Variable - variation
#Find unique variation
len(df['variation'].unique())

#Plotting Variation
plt.figure(figsize=(18,7))
sns.countplot(x='variation', data=df, palette="hls")
plt.xlabel("variation", fontsize=15) #seting the xtitle and size
plt.ylabel("Count", fontsize=15) # Seting the ytitle and size
plt.title("variation Count", fontsize=15)
plt.xticks(fontsize=10)
plt.yticks(fontsize=12)
plt.xticks(rotation=45)
plt.show()

"""Больше всего Black Dot и меньше всего Walnut finish"""

# Lets check the reviews again
df['verified_reviews'].head(5)
```

```
#Step 1: transform to lowercase
df['review_lw'] = df['verified_reviews'].str.lower()
df[['verified_reviews','review_lw']].head(10)

#Step 2: Removing HTML Tags
df['review_lw'] = df['review_lw'].apply(lambda x: BeautifulSoup(x, 'html
df[['verified_reviews','review_lw']].head(10)

#Step 3: remove stopwords 'n punctuation
sw = stopwords.words('english')
def transform_text(s):

# remove html
html=re.compile(r'<.*?>')
s = html.sub(r'',s)

# remove numbers
s = re.sub(r'\d+', '', s)

# remove punctuation

# remove stopwords
tokens = nltk.word_tokenize(s)

new_string = []
for w in tokens:

# remove words with len = 2 AND stopwords
if len(w) > 2 and w not in sw:
new_string.append(w)
s = ' '.join(new_string)
s = s.strip()
exclude = set(string.punctuation)
```

```
s = ''.join(ch for ch in s if ch not in exclude)
return s.strip()
df['review_sw'] = df['review_lw'].apply(transform_text)
df[['verified_reviews','review_lw', 'review_sw']].head(10)

#Step 4: lemmatizer

lemmatizer = WordNetLemmatizer()
def lemmatizer_text(s):
tokens = nltk.word_tokenize(s)
new_string = []
for w in tokens:
lem = lemmatizer.lemmatize(w, pos="v")

# exclude if lenght of lemma is smaller than 2
if len(lem) > 2:
new_string.append(lem)
s = ' '.join(new_string)
return s.strip()

df['review_lm'] = df['review_sw'].apply(lemmatizer_text)
df[['verified_reviews','review_lw', 'review_sw', 'review_lm']].head(20)

#Step 5: Removing Rare Words
text = ' '.join(df['review_lm'])
len(text)

# Creating Frequency
text_series = pd.Series(text.split())
freq_comm = text_series.value_counts()
freq_comm
rare_words = freq_comm[-1266:-1]
'rattlecrackle' in rare_words
rare_words
```

```
# Removing 1266 rare occuring words

df['review_lm'] = df['review_lm'].apply(lambda x: ' '.join([word for wor
df['review_lm'].sample(5)
text = ' '.join(df['review_lm'])
len(text)

"""Wordcloud for feedback == 1"""

df_pos = df[df['feedback']==1]['review_lm']
wordcloud1 = WordCloud(stopwords=STOPWORDS,
background_color='white',
width=2500,
height=2000
).generate(" ".join(df_pos))

plt.figure(1,figsize=(12, 12))
plt.imshow(wordcloud1)
plt.axis('off')
plt.show()

"""Wordcloud for feedback == 0"""
df_neg = df[df['feedback']==0]['review_lm']
wordcloud1 = WordCloud(stopwords=STOPWORDS,
background_color='white',
width=2500,
height=2000
).generate(" ".join(df_neg))

plt.figure(1,figsize=(12, 12))
plt.imshow(wordcloud1)
plt.axis('off')
plt.show()
```

```
def plot_ngram(sentiment, n):
print("sentiment",sentiment)
temp_df = df[df['feedback'] == sentiment]
word_vectorizer = CountVectorizer(ngram_range=(n, n), analyzer='word')
sparse_matrix = word_vectorizer.fit_transform(temp_df['review_lm'])
frequencies = sum(sparse_matrix).toarray()[0]
return pd.DataFrame(frequencies, index=word_vectorizer.get_feature_names
.sort_values(by='frequency', ascending=False) \
.reset_index() \
.head(10)

# For Feedback=1
plot_ngram(1, 1)
plot_ngram(0, 1)

#Feedback=1 (i.e. +ve)
df[df['feedback']== 1]['review_lm'].count()

#Feedback=0 (i.e. -ve)
df[df['feedback']== 0]['review_lm'].count()

"""Train and test dataset
Feeedback=1 : Total 2893
we can use the first 2025(70%) for training and 868(30%) remaining for t
i.e. 2893=2025+868
Feeedback=0 : Total 257
we can use the first 180(70%) for training and 77(30%) remaining for tes
i.e. 257=180+77
"""

# Train dataset
pos_train = df[df['feedback']==1][['review_lm', 'feedback']].head(2025)
neg_train = df[df['feedback']==0][['review_lm', 'feedback']].head(180)
```

```
# Test dataset
pos_test = df[df['feedback']==1][['review_lm', 'feedback']].tail(868)
neg_test = df[df['feedback']==0][['review_lm', 'feedback']].tail(77)

# put all toghether
train_df = pd.concat([pos_train, neg_train]).sample(frac = 1).reset_inde
test_df = pd.concat([pos_test, neg_test]).sample(frac = 1).reset_index(d
train_df.head()
test_df.head()
X_train = train_df['review_lm']
X_test  = test_df['review_lm']
y_train = train_df['feedback']
y_test  = test_df['feedback']
vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train)
X_train_onehot = vectorizer.transform(X_train)
X_test_onehot = vectorizer.transform(X_test)

#print shape post encoding
print(X_train_onehot.shape)
print(X_test_onehot.shape)

#Let's have a look at 10 words from the word vocabulary
#with their corresponding indices in the vocabulary.
word_dict = vectorizer.vocabulary_
print({k: word_dict[k] for k in list(word_dict)[:20]})

# Generic function for model building
def fit_and_test(classifier, X_train, y_train, X_test, y_test, only_retu
classifier.fit(X_train, y_train)
y_hat = classifier.predict(X_test)
print('accuracy:', accuracy_score(y_test, y_hat))
if not only_return_accuracy:
```

```
print('f1_score:', f1_score(y_test, y_hat))

#Logistic Regression
#grid search over regularisation hyperparameter 'c'
for c in [0.01, 0.02, 0.05, 0.25, 0.5, 0.75, 1,]:
lr = LogisticRegression(C=c, max_iter=1000) # 92.91%
print (f'At C = {c}:-', end=' ')
fit_and_test(lr, X_train_onehot, y_train, X_test_onehot, y_test, True)

"""Запустим логистическую регрессию с гиперпараметром C = 1, так как при

#Make an instance of the model
logistic = LogisticRegression(C=1,max_iter=10000)
#fitting the values for x and y
logstic.fit(X_train_onehot,y_train)

#predictions from test data
prediction = logistic.predict(X_test_onehot)

###### confusion matrix  starts ######
from sklearn.metrics import accuracy_score, confusion_matrix
cm_lgr1 = confusion_matrix(y_test,prediction)
names = np.unique(prediction)
sns.heatmap(cm_lgr1, square=True, annot=True, cbar=False,xticklabels=nam
plt.xlabel('Truth')
plt.ylabel('Predicted')

###### Confusion matrix ends ########
#calculating the accuracy
accuracy_score = accuracy_score(y_test,prediction)
print("Accuracy of  Logistic Regression :",accuracy_score)

"""Простая логистическая регрессия достигает 93% точности при значении г
Выведем n-граммы и frequency
```

```
"""

################ Print n-grams and frequency starts ################
vectorizer = CountVectorizer()
vect_texts = vectorizer.fit_transform(list(df['review_lm']))
all_ngrams = vectorizer.get_feature_names_out()
#print(all_ngrams[:5])
num_ngrams = min(50, len(all_ngrams))
all_counts = vect_texts.sum(axis=0).tolist()[0]
all_ngrams, all_counts = zip(*[(n, c) for c, n in sorted(zip(all_counts,
ngrams = all_ngrams[:num_ngrams]
counts = all_counts[:num_ngrams]
idx = np.arange(num_ngrams)

# Let's plot a frequency distribution plot for the most seen words
plt.figure(figsize=(18, 18))
plt.bar(idx, counts, width=0.8)
plt.xlabel('N-grams')
plt.ylabel('Frequencies')
plt.title('Frequency distribution of ngrams')
plt.xticks(idx, ngrams, rotation=45)
plt.show()

"""Keras with batch normalisation with mode=count"""

from tensorflow.keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
import tensorflow as tf
from sklearn.metrics import accuracy_score, confusion_matrix
from keras.layers import Embedding

# create the tokenizer
```

```
tokenizer = Tokenizer()
# fit the tokenizer on the documents
tokenizer.fit_on_texts(X_train)
# encode training data set
Xtrain = tokenizer.texts_to_matrix(X_train, mode='count')

# encode training data set
Xtest = tokenizer.texts_to_matrix(X_test, mode='count')
n_words1 = Xtest.shape[1]
print("n_words1", n_words1)
model = tf.keras.models.Sequential([
tf.keras.layers.Dense(256, input_shape=(n_words1,), activation='relu'),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',
optimizer=Adam(0.0001),
metrics=['acc'])
callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_loss', patien
history = model.fit(Xtrain, y_train,
epochs=20,
validation_data=(Xtest, y_test,),
verbose=1,
callbacks=callbacks,
)

# Function to make predictions
def prediction(X_test, clf_object):
y_pred = clf_object.predict(X_test)
return y_pred
```

```
predictions_keras = prediction(Xtest, model)
pred_ann = [ 1 if y>=0.5 else 0 for y in predictions_keras]
#calculating the accuracy
accuracy_score = accuracy_score(y_test, pred_ann)
print("Accuracy score with Keras:",accuracy_score)
def plot_history(history):
accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1,len(accuracy) + 1)

# Plot accuracy
plt.figure(1)
plt.plot(epochs, accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'g', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss
plt.figure(2)
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plot_history(history)

###### confusion matrix  starts ######
```

```
from sklearn.metrics import accuracy_score, confusion_matrix
cm_ann = confusion_matrix(y_test,pred_ann)
names = np.unique(pred_ann)
sns.heatmap(cm_ann, square=True, annot=True, cbar=False,xticklabels=name
plt.xlabel('Truth')
plt.ylabel('Predicted')
###### Confusion matrix ends ########

"""Embedding Layer with CNN"""
# create the tokenizer
tokenizer = Tokenizer()

# fit the tokenizer on the train documents
tokenizer.fit_on_texts(X_train)

# sequence encode
encoded_docs = tokenizer.texts_to_sequences(X_train)

# pad sequences
max_length = max([len(s.split()) for s in X_train])
Xtrain = tf.keras.preprocessing.sequence.pad_sequences(encoded_docs, max

# fit the tokenizer on the test documents
# sequence encode
encoded_docs = tokenizer.texts_to_sequences(X_test)

# pad sequences
Xtest = tf.keras.preprocessing.sequence.pad_sequences(encoded_docs, maxl

# define vocabulary size (largest integer value)
vocab_size = len(tokenizer.word_index) + 1
print("vocab_size = ",vocab_size)

# define model
```

```
model = Sequential()
model.add(Embedding(vocab_size, 100, input_length=max_length))
model.add(Conv1D(filters=32, kernel_size=8, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
print(model.summary())

# compile network
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac

# fit network
model.fit(Xtrain, y_train, epochs=10, verbose=1)

# evaluate
loss, acc = model.evaluate(Xtest, y_test, verbose=0)
print('Test Accuracy: %f' % (acc*100))

token = Tokenizer(num_words=5000)

##### Hyperparamaters
vocab_size = len(token.word_index) +  1
maxlen = 230

def preprocess(text):
text = token.texts_to_sequences(text)
text =  tf.keras.preprocessing.sequence.pad_sequences(text,padding='pre'
return text

def predict(text):
text = preprocess(text)
pred = model.predict(text)
return pred
```

ПРИЛОЖЕНИЕ Б

Программная реализация модели с 94% точностью

```
# -*- coding: utf-8 -*-
"""Диплом_2.ipynb
Automatically generated by Colaboratory.
Original file is located at
https://colab.research.google.com/drive/16nHTomRCeejtoWzcVrm-zAsOrDmOGKg
"""
pip install opendatasets
! pip install kaggle
import opendatasets as od

from google.colab import drive
drive.mount('/content/drive')
import os
os.environ['KAGGLE_CONFIG_DIR'] = "/content/drive/MyDrive/Kaggle"

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/drive/MyDrive/Kaggle/
pwd

pip install pad_sequences

import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,LSTM,Embedding
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import nltk
nltk.download('stopwords')
import re
import matplotlib.pyplot as plt
```

```
od.download("https://www.kaggle.com/datasets/sid321axn/amazon-alexa-revi
os.listdir()
# Commented out IPython magic to ensure Python compatibility.
# %cd ./amazon-alexa-reviews/
pwd
ls
df = pd.read_csv('amazon_alexa.tsv', sep='\t')
df.head()
df['rating'] = df['rating'].apply(lambda x: 1 if x >= 3 else 0)
df = df[['rating','verified_reviews']]
df.head()
df['rating'].value_counts()
plt.hist(df['rating'])
from nltk.corpus import stopwords
STOPWORDS = stopwords.words("english")

def clean(x):
x = x.lower()
x = re.sub("[^\w\d]"," ",x)
x = " ".join([y for y in x.split() if y not in STOPWORDS])
return x

df['text'] =  df['verified_reviews'].apply(lambda x :  clean(x))
df.head()
x=  np.array(df['text'])
y = np.array(df['rating'])
df['rating'].shape
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.1,shuff
token = Tokenizer(num_words=5000)
token.fit_on_texts(x_train)
x_train = token.texts_to_sequences(x_train)
x_test = token.texts_to_sequences(x_test)
vocab_size = len(token.word_index) +  1
maxlen = 30
```

```
x_train = pad_sequences(x_train,padding='pre',maxlen=maxlen)
x_test = pad_sequences(x_test,padding='pre',maxlen=maxlen)

model = Sequential([
Embedding(vocab_size,100,input_length=maxlen),
Dropout(0.2),
LSTM(200),
Dropout(0.2),
Dense(1,activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['acc'
model.summary()
history  = model.fit(x_train,y_train,epochs=10,batch_size=32,validation_
loss,acc =  model.evaluate(x_test,y_test)
acc
history_df = pd.DataFrame(history.history)
history_df[['loss','val_loss']].plot(title='LOSS')
history_df[['acc','val_acc']].plot(title='Accuracy')

def preprocess(text):
text = token.texts_to_sequences(text)
text =  pad_sequences(text,padding='pre',maxlen=maxlen)
return text

def predict(text):
text = preprocess(text)
pred = model.predict(text)
return pred

predict(["like siri fact siri answers accurately alexa see real need hou
```