Федеральное государственное образовательное бюджетное учреждение
высшего образования

«ФИНАНСОВЫЙ УНИВЕРСИТЕТПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»

(Финансовый университет)

Департамент анализа данных, принятия решений и финансовых технологий

Курсовая работа

на тему

«Машинное обучение в задачах классификации текстов»

Выполнил:

студент группы ПМ19-1

Волненко
А.А.

Научный руководитель:.
Шаталова А.Ю

Москва 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Классификация текстов сейчас является очень популярной задачей в машинном обучении. Самыми простыми и очевидными примерами, в которых необходимо классифицировать текст - это задача определения спама текста, определение темы документа, оценка тональности отзыва о продукте, либо определение его рейтинга и другие задачи.

Однако, в последнее время обычно классификация текста не является отдельно стоящей задачей, а используется как часть какого-то большего процесса. Так, в голосовом помощнике производится классификация произнесенного человеком текста для того, чтобы понять, что требуется сделать и какую задачу необходимо выполнить (поставить будильник, заказать такси и другие) и в зависимости от определенной задачи, данная информация передается в соответствующую модель для дальнейшей работы.

Более того классификация текстов является с помощью машинного обучения является востребованной задачей в бизнесе, потому что может ему сильно помочь в разных его сферах. Так, можно классифицировать проблемы пользователей, с которыми они обращаются в службу поддержки и в зависимости от вопроса давать им релевантные решения, что поможет снизить количество рутинного ручного труда. Также можно анализировать комментарии пользователей, которые они оставляют на различных ресурсах, чтобы вовремя обращать внимание на возможные проблемы и решать их.

Глава 1. Теоретическая справка

1.1 Кодирование слов

Прежде чем говорить об алгоритмах для классификации текстов необходимо разобраться с тем, как собственно кодировать слова и сами предложения, потому что все модели машинного обучения принимают на вход лишь вектора чисел и не умеют работать напрямую со словами.

То, как модели машинного обучения воспринимают данные отличается от того, как делают это люди. Так, человек с легкостью может понять, какой смысл несет в себе предложение, просто прочитав его слова. Однако это задача не так проста для моделей машинного обучение, потому что они не могут работать напрямую со словами и предложениями, то, что они понимаю - вектора, состоящие из чисел, которые называются эмбедингами слов и уже могут быть спокойно поданы на вход в модель.

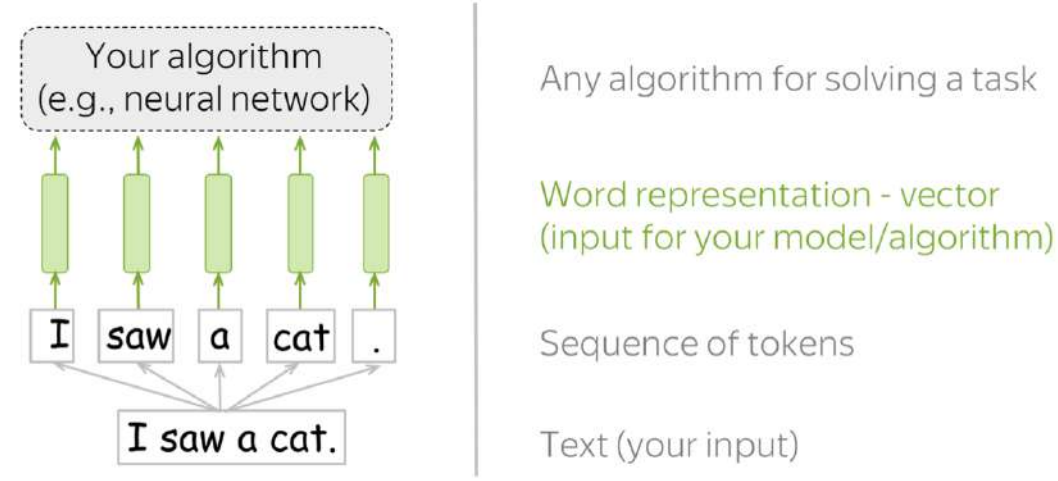


Рисунок 1. Иллюстрация того, что слова необходимо переводить в вектора вещественных чисел перед тем, как подавать их на вход в модель машинного обучения.

На практике у нас обычно есть словарь допустимых слов, который составляется заранее. У всех слов определены их эмбединги, которые хранятся в отдельном списке, а эмбеддинг какого-то определенного слова мы можем получить по его индексу в общем словаре слов.

Рисунок 2. Пример того, как мы получаем эмбеддинг для конкретного слова.

Для того, чтобы учесть неизвестные слова, обычно в словаре присутствует специальный символ UNK, которым заменяются все ранее не встречаемые слова, однако, их можно также пропускать, либо определить равным вектору, состоящим из нулей.

Однако главным вопросом является то, как составлять эмбеддинг для слов?

1.1.1 Использование горячего кодирования

Наиболее простым способом кодирования слов является горячее кодирование (one hot encoding), где для i-го слова в словаре в его соответствующем вектор стоит единица только на i-й позиции, а все остальные равны нулю. В машинном обучении данный способ является наипростейшим для представления категориальных признаков.

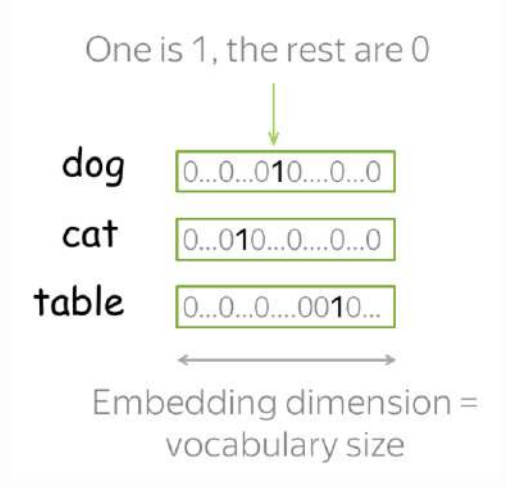


Рисунок 3. Пример использования горячего кодирования

Самой очевидной проблемой данного подхода является то, что для словарей, состоящих из большого числа слов, соответствующие вектора будут иметь очень большую размерность, равную числу слов в словаре. Однако данная проблема не является основной, потому что такие представления совершенно не как не отражают смысл слов, их связь между собой и то, в каком контексте они находятся.

1.1.2 Метод основанный на подсчете контекста слов

Основная идея данного метода заключается в том, что мы создаем матрицу, где по горизонтали и вертикали будут отмечены слова из нашего словаря, тогда каждый элемент данной матрицы будет отражать то, как часто соответствующие слова встречаются в одном контексте. Далее производится понижение размерности данной матрицы, т.к. прежде всего она получается очень большой, а во вторых большинство слов появляются только в определенном контексте, поэтому большое кол-во элементов данной матрицы будут нулями.

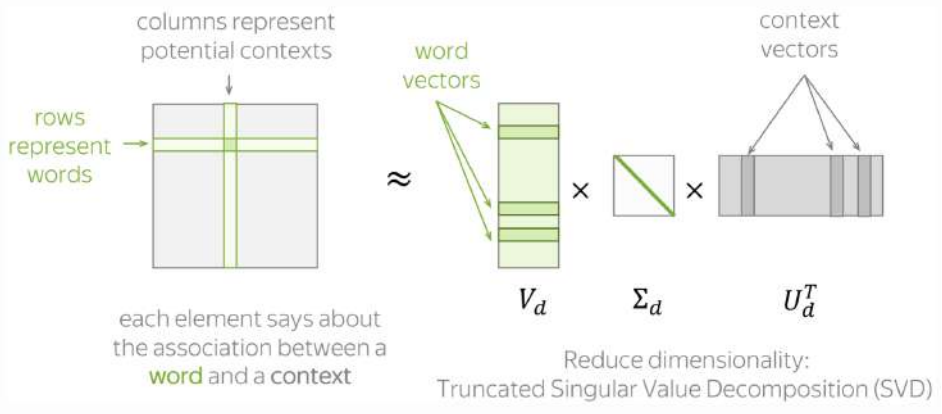


Рисунок 4. Пример матрицы контекста слов.

Самым простым способом определить контекст некоторого слов является

Контекстом слова будем называть все слова, которые попадают в окно заранее определенной размерности L.

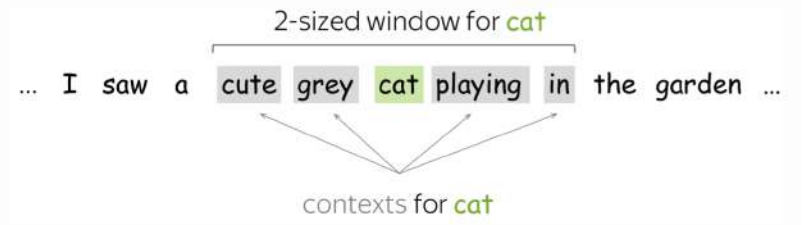


Рисунок 5. Иллюстрация определения контекста слова.

Однако для того, чтобы подсчитывать значения матрицы контекста слов есть два подхода. Первый - значение матрицы на позиции (w, c) будет содержать непосредственно количество раз, которое слово w встретилось в контексте слова c. Данный подход является наиболее базовым для определения эмбедингов слов. Второй же подход является более сложным, но с помощью некого можно получить более хорошо кодирование слов. В данном случае, каждое значение матрицы будет содержать положительную поточечную взаимную информацию (PPMI) слов, которая рассчитывается $PPMI(w,\ c)\ =\ max(0,\ PMI(w,\ c))$, где $PMI(w,c)\ =\ log\frac{P(w,\ c)}{P(w)P(c)}\ =\ log\frac{N(w,\ c)|(w,c)|}{N(w)N(c)}$

.

В отличие от подхода горячего кодирования, в данном подходе уже появляется смысл у появляется смысл у закодированных эмбедингов и слова, которые употребляются примерно в одном контексте будут иметь схожие эмбединги.

1.2 Кодирование предложений

Самый простой способ получения эмбедингов предложений, который не использует никаких нейронных сетей - это просто взять сумму эмбедингов всех слов (Bag of Embeddings - BOE). Использование такого кодирования может

применяться для бейзлайнового решения задачи классификации текстов, т.е. самым простым решением, от которого можно уже будет отталкиваться. Чуть более сложным подходом будет взятие взвешенной суммы эмбедингов слов, например с помощью tf-idf весов.

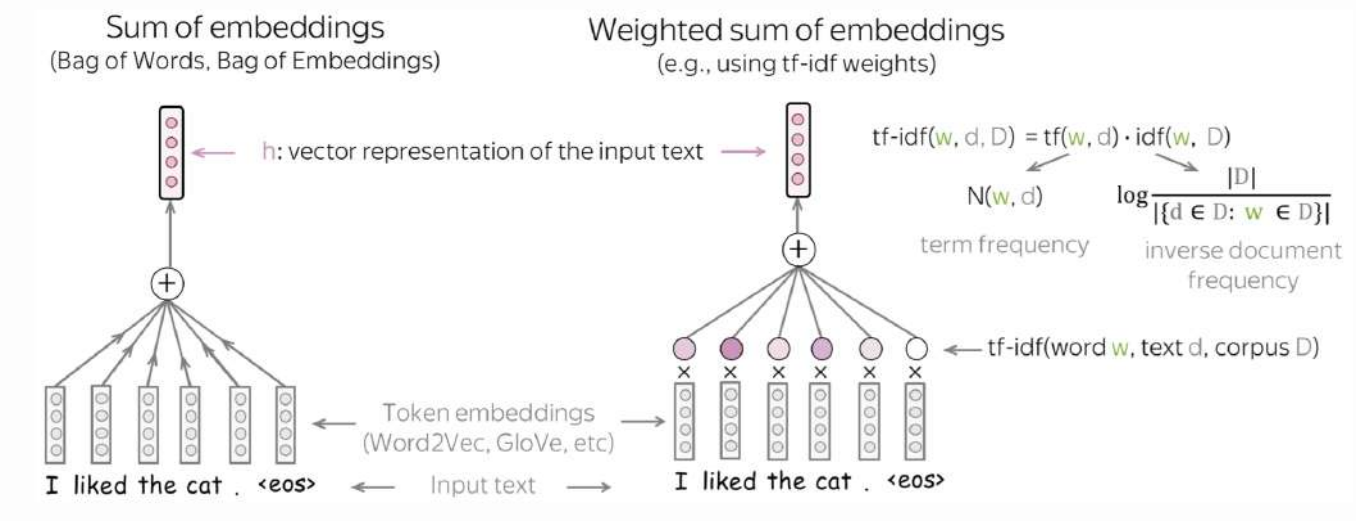


Рисунок 6. Сравнение суммы эмбедингов и взвешенной суммы эмбедингов.

1.3 Классические методы для классификации текстов

В данной части будут рассмотрены классические методы для классификации текстов, которые были придуманы задолго до того, как нейронные сети стали популярны, но даже сейчас для малых датасетов могут показывать сравнимо хорошее качество.

1.3.1 Наивный Байесовский классификатор

Для того, чтобы получить наиболее вероятный ответ на объекте x мы хотели бы смоделировать распределение $P(y|x)$, которую можно оценить по формуле $P(y|\,x) = \frac{P(y,x)}{P(x)}$, однако напрямую это сделать сложно, потому что необходимая вероятность $P(x,\,y)$ обычно стремится к 0. Однако мы можем

воспользоваться правилом Байеса и тогда формула примет следующий вид: $P(y|x) = \frac{P(x|y)P(y)}{P(x)}$. Получается, что задача принимает следующий вид:

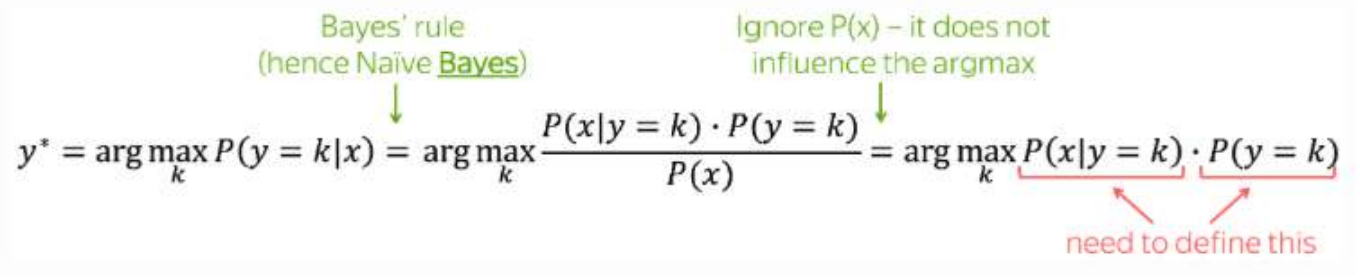


Рисунок 7. Оптимизационная задача Наивного Байеса.

Мы не учитываем P(x), потому что в нашей задаче это является константой, т.е. данная вероятность не зависит от переменной, по которой производится оптимизация.

$P(y = k)$ называется априорной вероятностью и ее можно найти по формуле, которая получается из метода максимального правдоподобия:

$$P(y=k) = \frac{N(y=k)}{\sum_i N(y=i)},$$

где $N(y = k)$ - это кол-во объектов выборки класса k.

Оценка же $P(x|y)$ не будет простой задачей, потому что такое значение тоже может стремиться к 0, и оценка данного значения не будет проще, чем задача оценки $P(y|x)$, поэтому сделаем следующее сильное предположение: будем считать признаки (т.е. слова предложения) независимыми между собой, тогда это позволить разбить $P(x|y)$ произведение отдельных вероятностей:

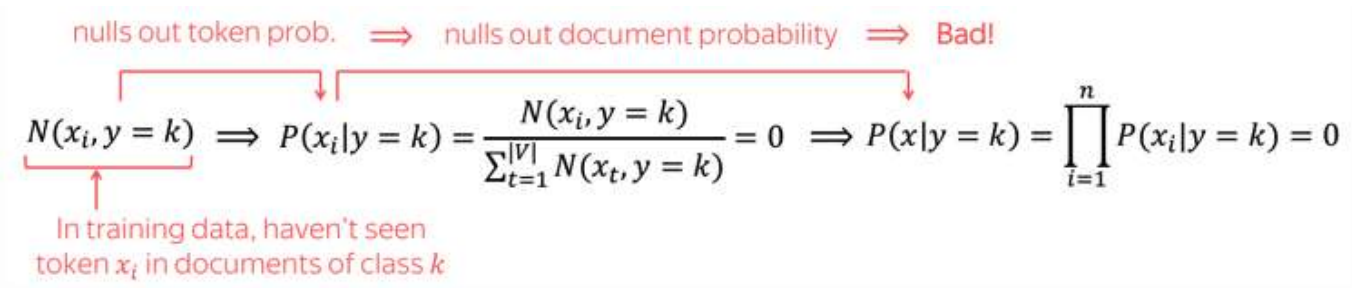
$$P(x|y=k) = P(x_1,\ldots,x_n|y=k) = \prod_{t=1}^{n} P(x_t|y=k).$$

Вероятность одного конкретного слова считается по формуле из метода максимального правдоподобия:

$$P(x_i|y=k) = \frac{N(x_i, y=k)}{\sum_{t=1}^{|V|} N(x_t, y=k)},$$

где $N(x_i, y = k)$ - число раз конкретный токен встречается в документе класса k, а V - кол-во слов в словаре.

Однако может произойти ситуация, когда $N(x_i, y = k) = 0$, т.е. ситуация, в которой в обучающей выборке не присутствовал какой-то токен x_i в документе классаа k, но был в тестовой выборке, что может занулить вероятность общего произведения для конкретного документа:



Для того, чтобы избежать данной ситуации, при подсчете количества слов в выборке делается небольшая добавка δ:

$$P(x_i|y=k) = \frac{\delta + N(x_i, y=k)}{\sum_{t=1}^{|V|}(\delta + N(x_t, y=k))} = \frac{\delta + N(x_i, y=k)}{\delta \cdot |V| + \sum_{t=1}^{|V|} N(x_t, y=k)},$$

данный прием обычно называется сглаживанием по Лапласу.

В итоге получается, что ответом алгоритма Наивного байеса является класс, который максимизирует следующую вероятность:

$$y^* = \arg\max_k P(x, y=k) = \arg\max_k P(y=k) \cdot P(x|y=k).$$

В задаче классификации текстов объектами выборки являются документы, которые необходимо классифицировать. В качестве признакового пространства выбирается мешок слов - сумма векторов слов, которые получаются с помощью горячего кодирования.

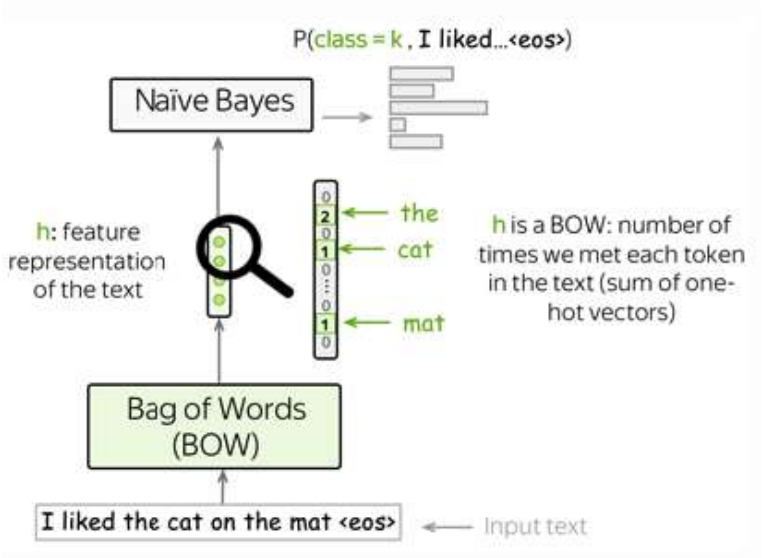


Рисунок 8. Пример признакового описания объект в задаче классификации текстов

1.3.2 Логистическая регрессия

В данном алгоритме также оценивается распределение $P(y|x)$, однако подход другой. Более того, в алгоритме нет ограничений на то, как должно выглядеть признаковое описание объектов, как было в алгоритме Наивного Байеса. Суть данного подхода в том, что каждый объект определяется некоторым вектором признаков $h = (f_1, f_2, \ldots, f_n)$, далее берется некоторый обучаемый вектор весов $w^{(i)} = (w^{(i)}_1, \ldots, w^{(i)}_n)$ и потом берется их скалярное произведение, к которому прибавляется некоторая константа:

$$w^{(k)}h = w_0^{(k)} + w_1^{(k)} \cdot f_1 + \cdots + w_n^{(k)} \cdot f_n, \quad k = 1, \ldots, K.$$

Тогда вероятность класса для конкретного объекта подсчитывается с помощью функции softmax:

$$P(class = k|h) = \frac{\exp(w^{(k)}h)}{\sum_{i=1}^{K}\exp(w^{(i)}h)}.$$

где K - общее кол-во возможных классов.

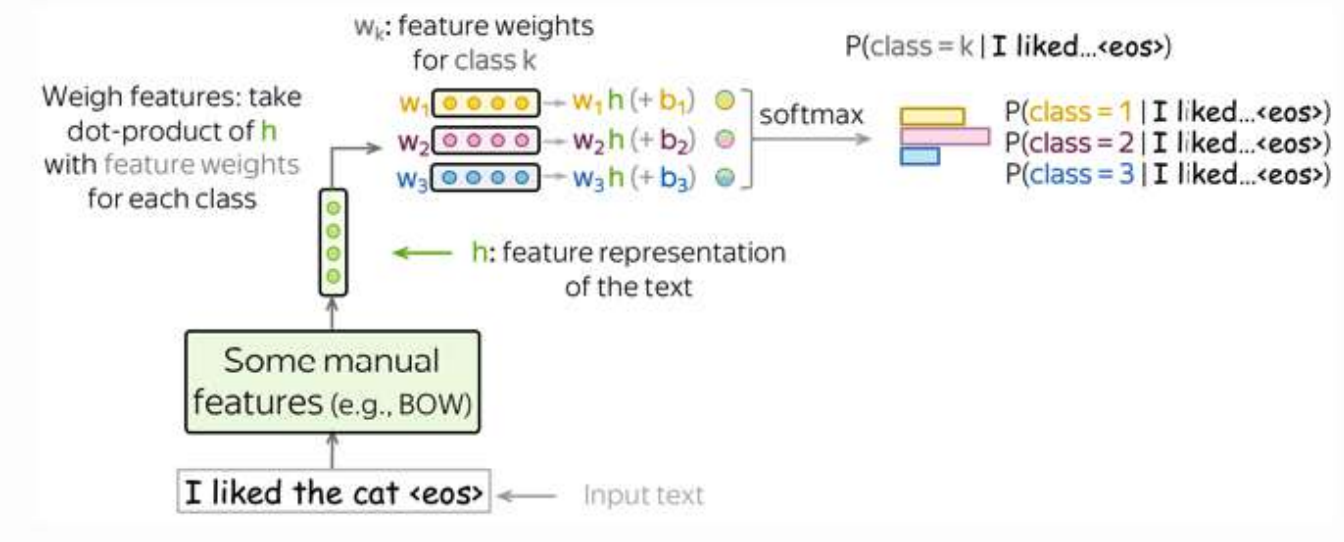


Рисунок 9. Иллюстрация работы логистической регрессии

Для того, чтобы подобрать параметры данной модели (веса) используется метод максимального правдоподобия:

$$w^* = \arg\max_w \sum_{i=1}^{N} \log P(y = y^i|x^i).$$

где x^i - i-й объект выборки с меткой класса равной y^i, где $y^i \in \{1, \ldots, K\}$.

Для того, чтобы найти параметры, которые максимизируют функцию правдоподобия используется градиентный спуск, который постепенно на каждой итерации улучшает значения параметров.

Максимизация функции правдоподобия эквивалентно минимизации функции кросс энтропии между целевым вектором ответов $p^* = (0, \ldots, 0, 1, 0, \ldots)$ (1 - стоит на позиции целевого класса, а 0 на всех остальных) и полученного моделью распределения вероятностей $p = (p_1, \ldots, p_K)$, $p_i = p(i|x)$:

$$Loss(p^*, p) = -p^* \log(p) = -\sum_{i=1}^{K} p_i^* \log(p_i).$$

А так как только одно значение вектора p^*_i равно 1, то формула выше принимает следующий вид:

$$Loss(p^*, p) = -\log(p_k) = -\log(p(k|x)).$$

1.4 Использование нейросетей для классификации текстов

Основным отличие использования нейросетей для классификации текстов состоит в том, что подавая на вход сети некие эмбединги токонов, с ее помощью можно получить признаковое представление всего текста, которое уже потом можно использовать для его классификации. Т.е. получается, что нейросеть сама способна составить необходимое признаковое описание подаваемого ей на вход текста.

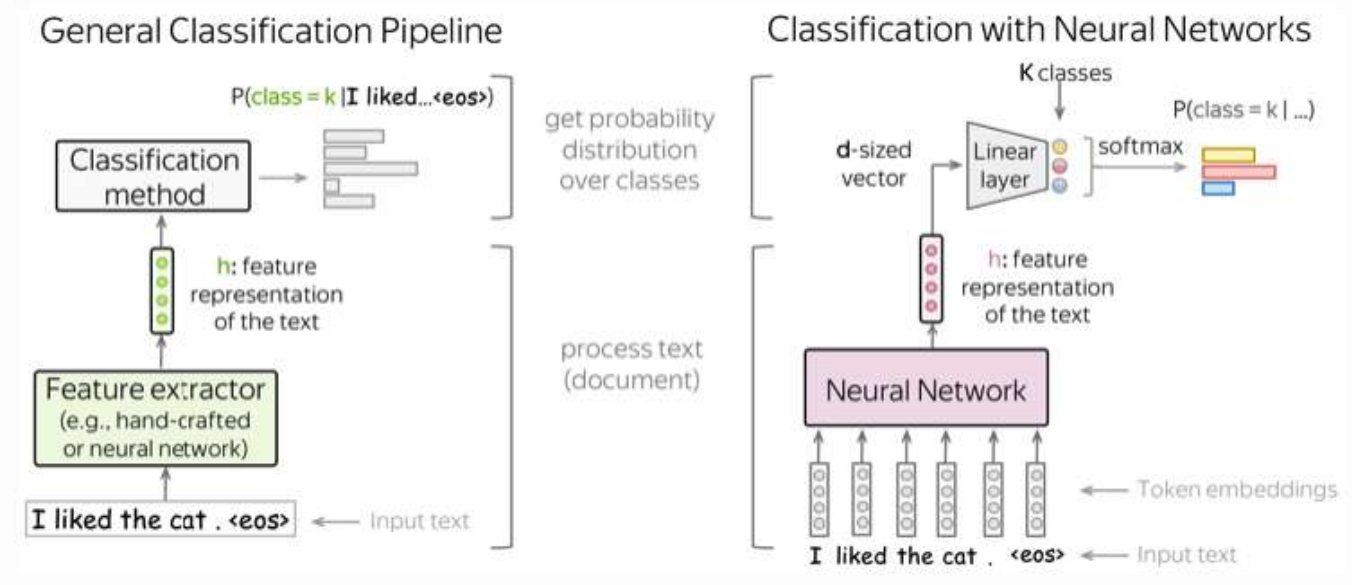


Рисунок 10. Отличие классических методов для классификации текстов от нейросетевых подходов

Векторное представление текста имеет имеет некую размерность d, однако для того, чтобы произвести классификацию размерность вектора должна быть равна K, где K - общее число классов. Для перевода вектора размернсоти d в размерность K используется полносвязный слой. После того, как получен вектор размерности K к нему применяется функция softmax для преобразованию к вектора чисел к распределению вероятностей для каждого класса.

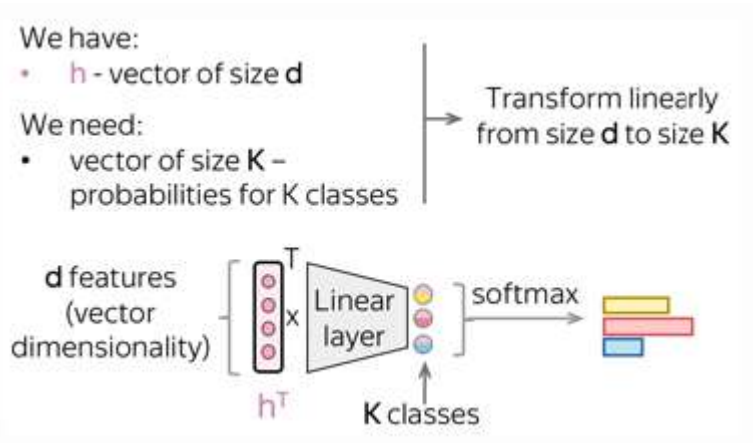


Рисунок 11. Преобразование вектора признакового описания текста в распределение вероятностей

Если присмотреться поближе к тому, как работает полносвязный слой нейросети, который переводит вектор признакового описания текста в распределение вероятностей, то можно понять, что это на самом деле логистическая регрессия, единственное отличие заключается лишь в том, как было получено признаковое описание: определено вручную, либо получено с помощью нейронной сети.

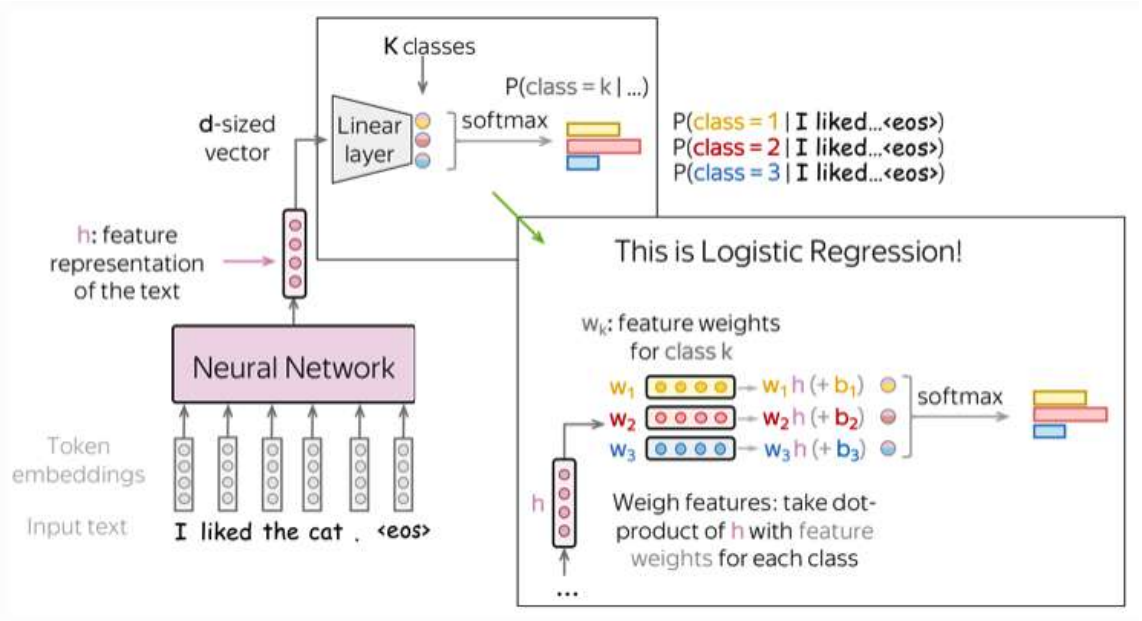


Рисунок 12. Признаковое описание текста переводится в распределение вероятностей с помощью логистической регрессии.

1.4.1 Рекуррентные нейронные сети

Использование рекуррентных нейронных сетей является наиболее естественным способом обработки текста, который схож с тем, как делают это люди: читают токен за токеном и обрабатывают входящую информацию. Предположение заключается в том, что на каждом наше рекуррентная сеть должна помнить все, что она “прочитала” на предыдущих шагах.

На каждом шаге рекуррентная нейронная сеть принимает новый входной вектор, а также скрытое состояние с предыдущего шага. С помощью данных двух векторов на текущем шаге вычисляется текущее скрытое состояние, которое содержит информацию как с предыдущих шагов, так и информацию с текущего входного значения.

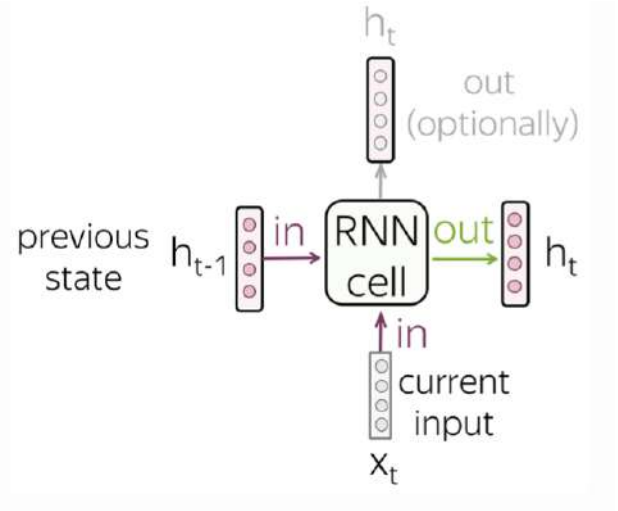


Рисунок 13. Вычисление текущего скрытого состояния

Рекуррентная нейронная сеть обрабатывает входные токены по очереди, вычисляя на каждом шаге текущее скрытое состояние с помощью текущего входного токена и скрытого состояния с предыдущего шага.

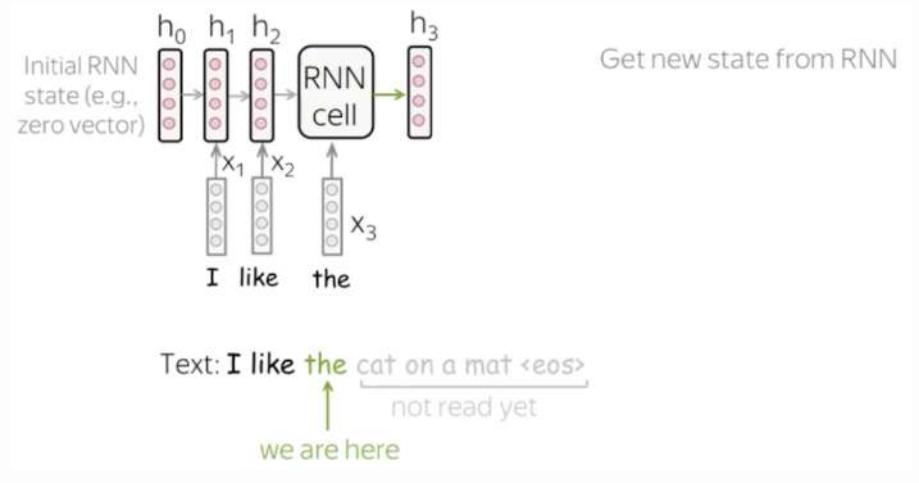


Рисунок 14. Поочередная обработка входных токенов рекуррентной нейронной сетью

Самый простой вариант рекуррентной сети - Vanilla RNN, в котором входной вектор x_t и скрытое состояния с предыдущего шага h_{t-1} преобразуются линейно, а далее к их сумме применяется нелинейная функция - функция тангенса.

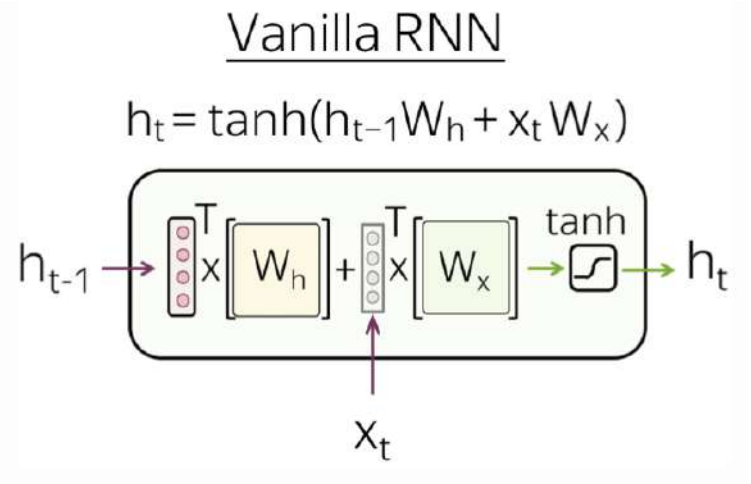


Рисунок 15. Вычисление текущего состояния в Vanilla RNN

Основная проблема Vanilla RNN состоит в том, что данная модель страдает от затухания или взрывания градиентов, что сильно сказывается на качестве ее работы. Для решения данной проблемы были предложены другие вариации рекуррентной сети: LSTM, GRU и другие.

1.4.2 Рекуррентные сети в задаче классификации текстов.

Самый простой вариант использования рекуррентной сети для классификации текста - использовать последний скрытой слой как эмбеддинг прочитанного текста для его классификации.

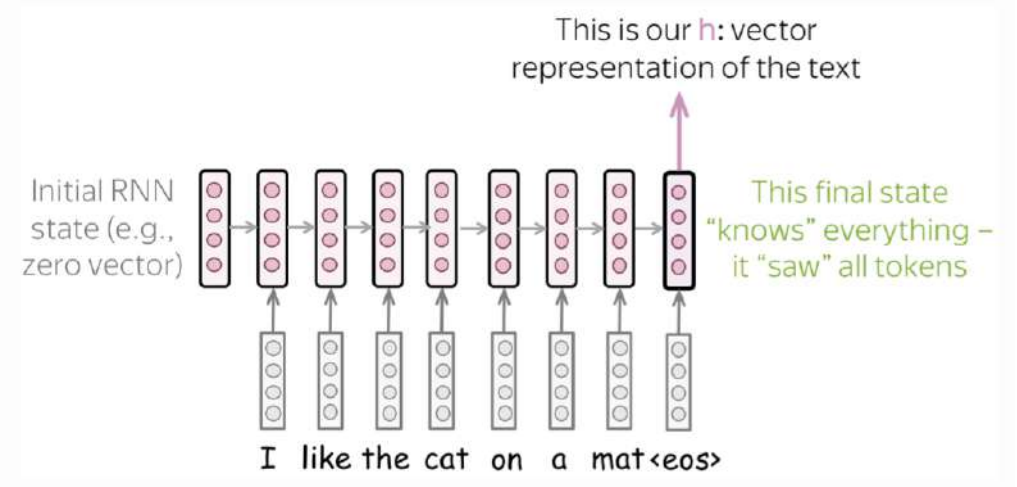


Рисунок 16. Однонаправленная рекуррентная сеть для классификации текста.

Другим возможным вариантом является использование многослойной реккуентой сети, где входами данными каждого последующего слоя на шаге являются скрытые состояния с предыдущего слоя с соответствующих шагов.

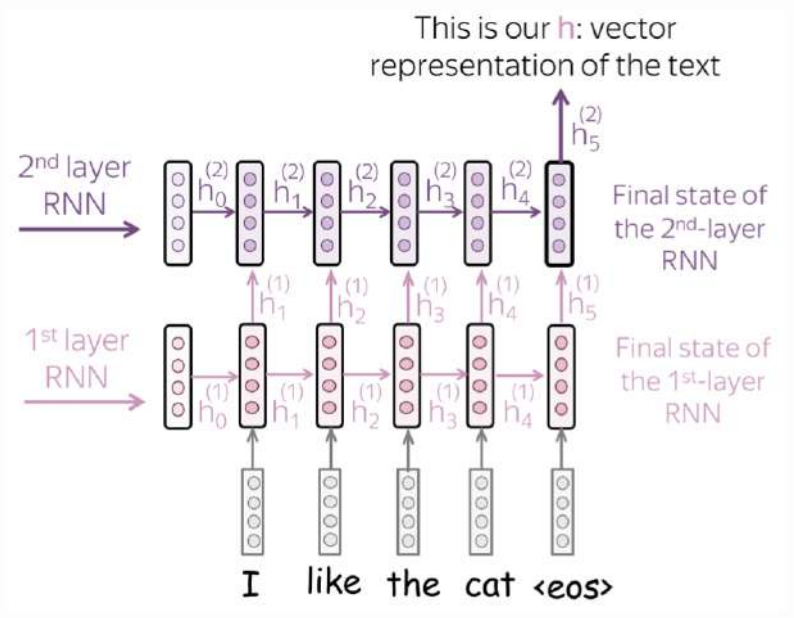


Рисунок 17. Многослойная рекуррентная сеть для классификации текстов.

Проблема двух предыдущих подходов состоит в том, что на конкретном шаге учитывается информация только с предыдущих шагов, хотя информация о том, что идет дальнейшем может быть не менее важна. Для решения данной проблемы можно использовать двунаправленные рекуррентные сети: такие сети состоят из двух рекуррентных сетей, где одна читает текст слева направо, а другая справа налево. Далее берутся конечные состояние обоих сетей и конкатенируются, либо суммируются.

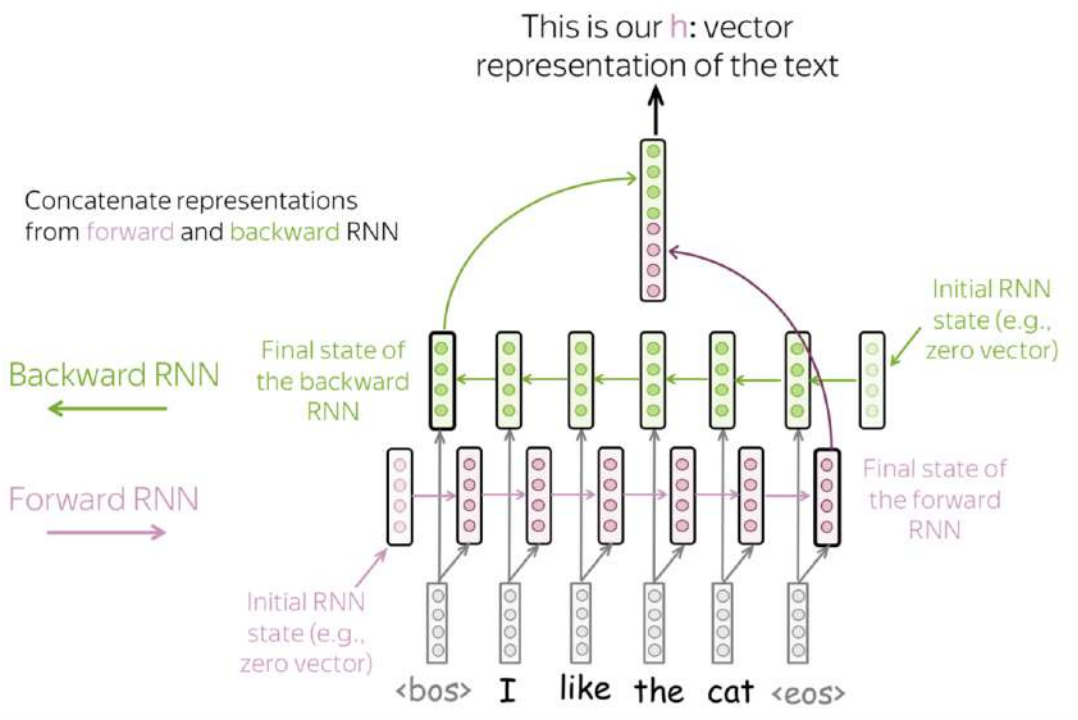


Рисунок 19. Пример двунаправленной рекуррентной сети.

Глава 2. Практическая часть

Для практической части был выбрал датасет IMDB, состоящий из 50 000 положительных и отрицательных отзывов. Данные были взяты с сайта kaggle.com.

2.1 Анализ и предобработка данных

Сначала было проведено предварительное ознакомление с датасетом. Итак, как и было ранее сказано, всего в датасете имеется 50 000 наблюдений, а количество положительных отзывов и отрицательных равно - по 25 000 каждого класса. Пропущенных значений не оказалось, однако после проверки на дубликаты обнаружилось, что в датасете есть 418 повторяющихся наблюдений, которые были в дальнейшем удалены.

Далее каждый отзыв был предобработан, т.е. из него была удалена вся ненужная информацию. Для этого сначала были удалены все html тэги, которые

не несут в себе никакой ценной информации. Далее были заменены некоторые разговорные выражение, для удобства последующей работы, такие как “ain’t”, “she’s”, “we’ll” на “is not”, “she is” и “we will”. Далее все символы были переведены в нижний регистр и были удалены все символы, кроме букв. Далее от каждого слова взяли только его лемму.

Далее был проведен последующий анализ датасета. Распределение количества слов в отзывах выглядело следующим образом:

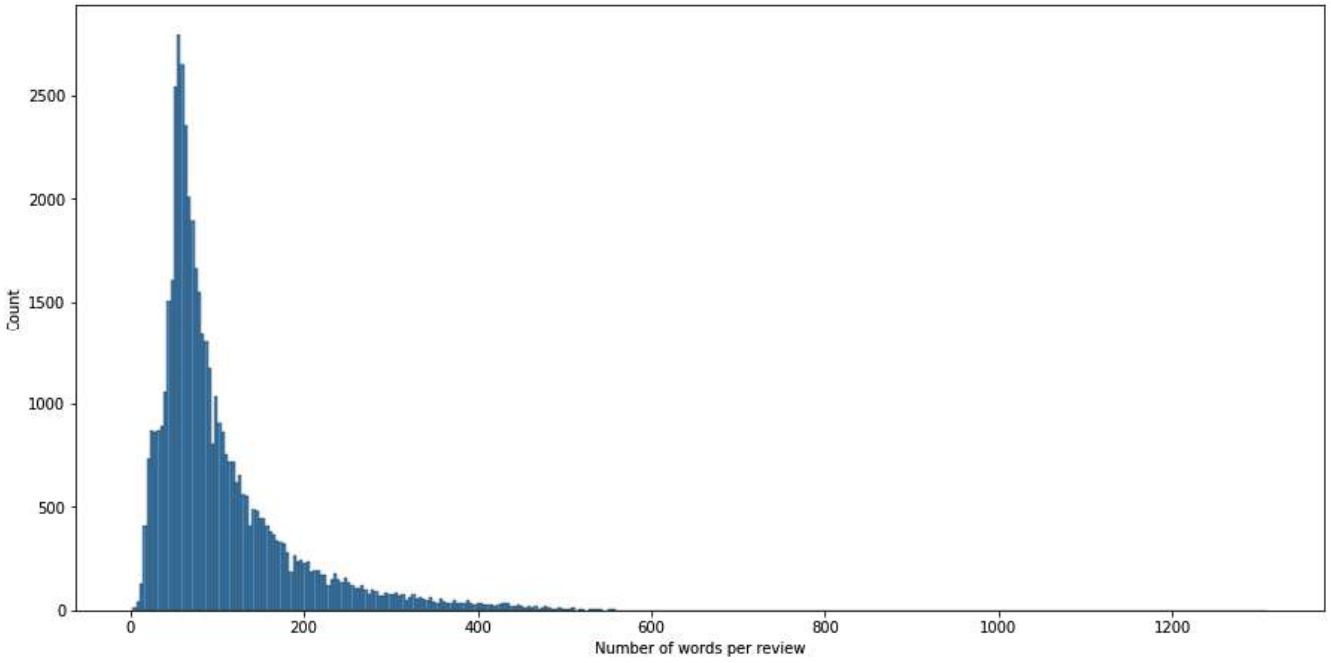


Рисунок 20. Распределение количества в объектах выборки

Со следующими описательными статистиками:

```
count    49582.000000
mean       104.595519
std         80.630538
min          3.000000
25%         55.000000
50%         77.000000
75%        127.000000
max       1310.000000
```

Далее я посмотрел распределения униграмм, биграмм и триграмм. Распределение униграмм выглядит следующим образом:

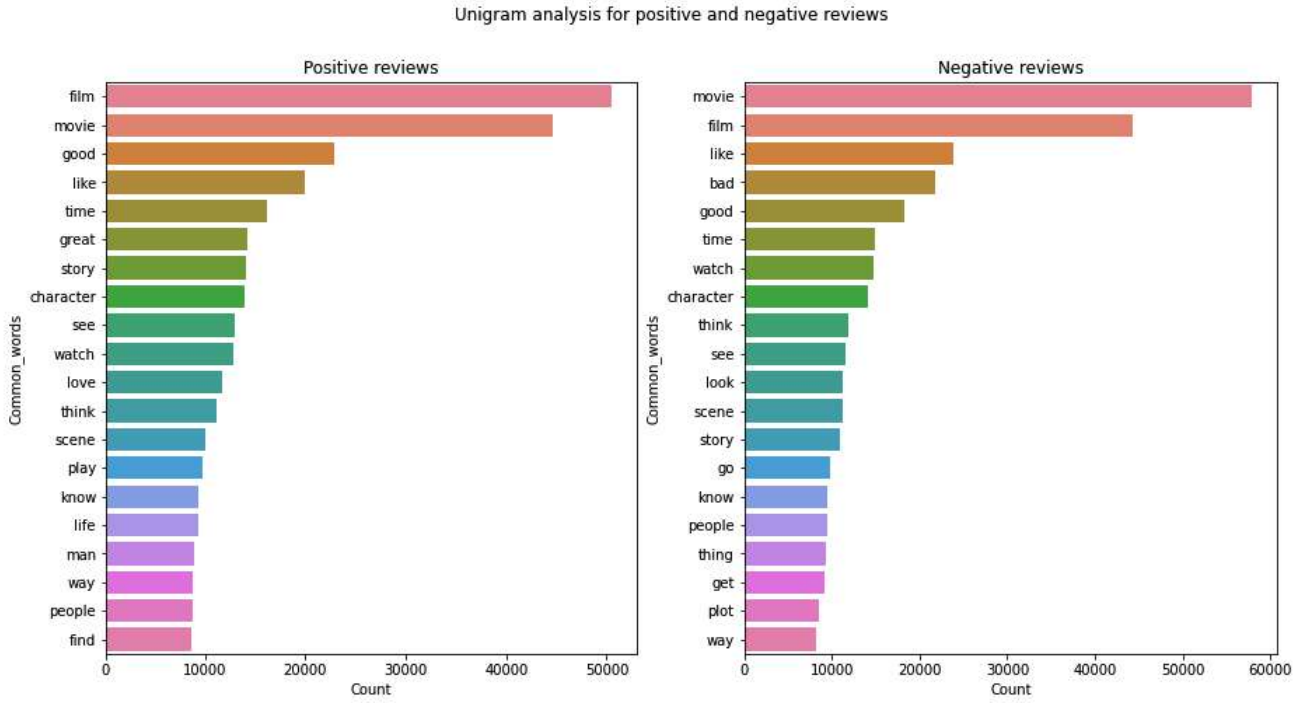


Рисунок 21. Распределение униграмм слов

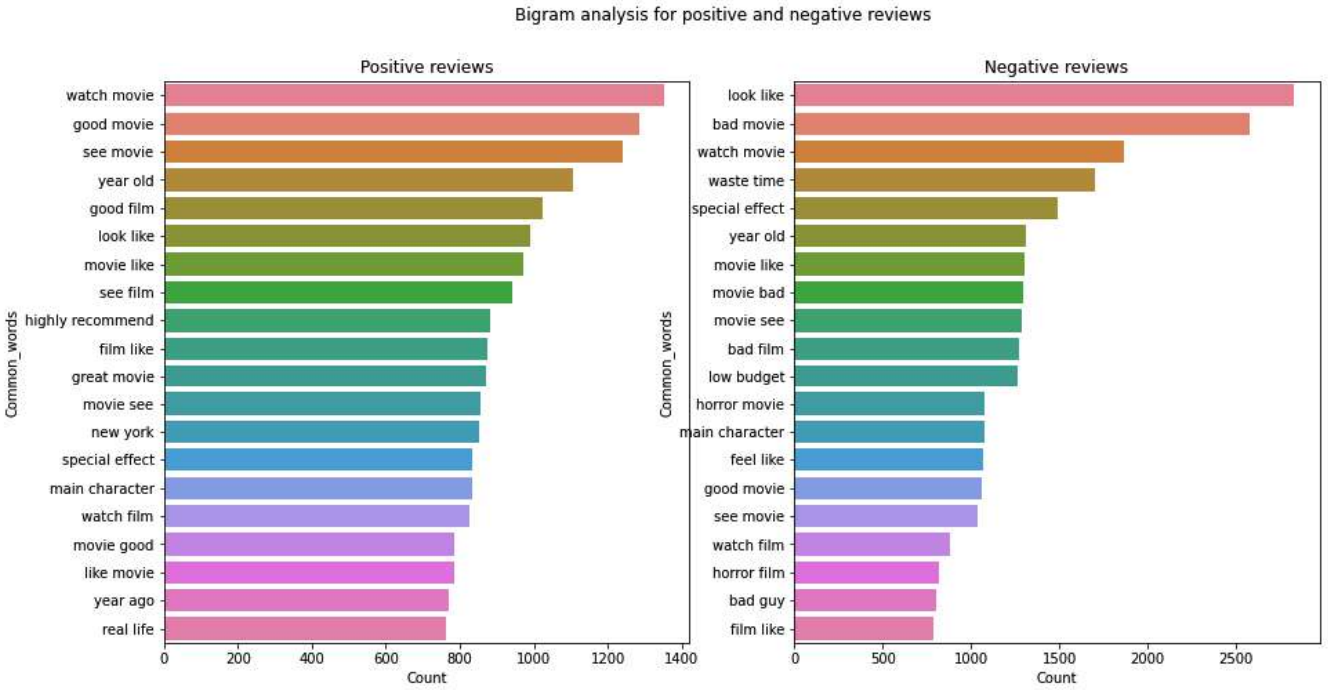


Рисунок 22. Распределение биграмм слов

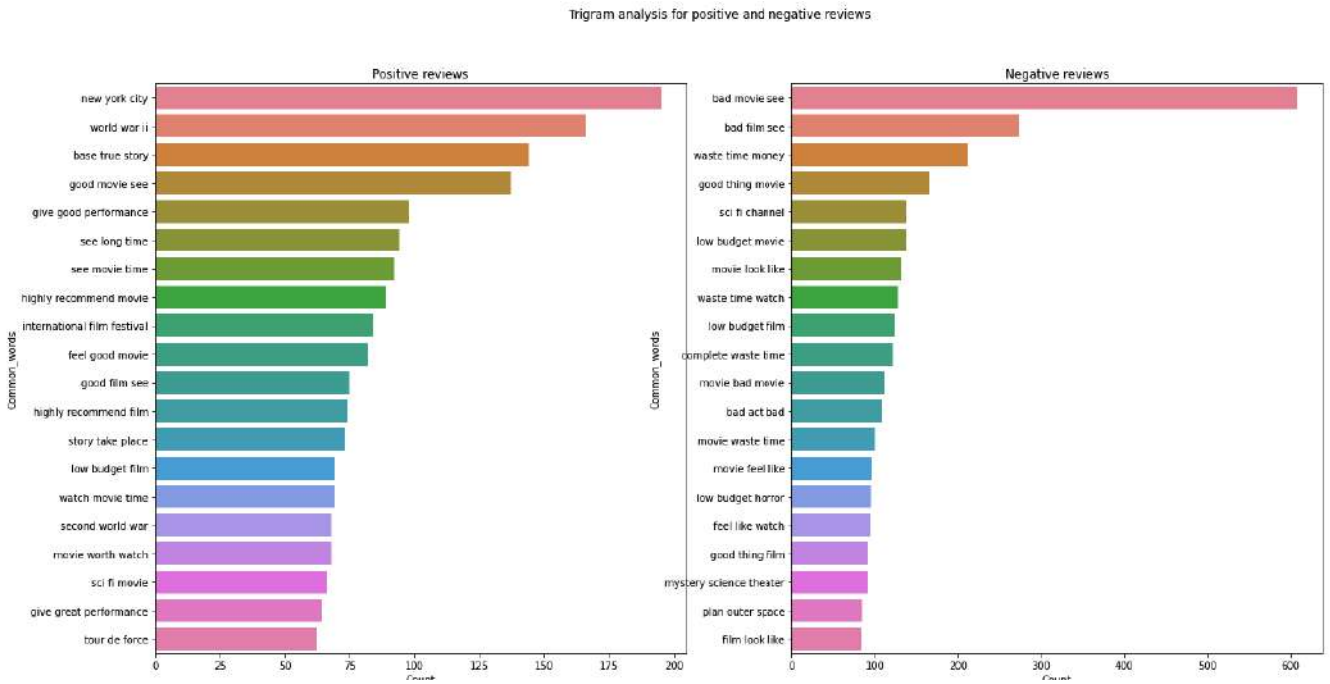


Рисунок 23. Распределение триграмм

2.2 Использование машинного обучения и нейронных сетей

Далее я перешел к построение классических моделей машинного обучения. Для начала я разбил выборку на обучающую, равную 70% от общего объема всех наблюдений и тестовую, состоящую из тех наблюдений, которые не вошли в обучающую.

Первый алгоритм машинного обучения, который я использовал - логистическая регрессия и для того, чтобы ее использовать я закодировал отзывы с помощью метода TF-IDF. Обучив модель ее точность на тестовой выборке составила 88.97%.

Далее я обучил наивный байес, для этого я закодировал отзывы с помощью подсчета. Точность на тестовой выборке составила 85.28%.

Далее я использовал методы глубокого обучения для решения задачи, а именно рекуррентные нейронные сети. Для начала необходимо определить максимальное допустимое количество слов в строке, чтобы все входные данные для сети были одного размера. Однако нельзя просто взять максимальную длину строки, которая есть в обучающем множестве, потому что тогда, в

строки, состоящие из небольшого числа слов придется добавлять много пустых символов, что может повлиять на качество работы сети. Для определения необходимой длины я посчитал описательные статистики длин всех предложений:

```
count    34707.000000
mean       104.692252
std         80.691193
min          3.000000
25%         55.000000
50%         77.000000
75%        127.000000
max       1007.000000
```

В итоге я выбрал максимальную длину строки равную 266 словам, которая была получена как среднее всех длин плюс два стандартных отклонения. Такая длина больше чем 95% длин всех слов из обучающей выборки.

Далее я обучил двухслойную рекуррентную сеть. Лучшее качество было достигнуто при 5-ти эпохах обучения и точность на тестовом множестве равнялась 87.42%. Добавление большего числа слоев или двунаправленности не улучшило итоговое качество модели.

Далее я попробовал использовать предобученные эмбединги слов, полученные с сайта https://nlp.stanford.edu/projects/glove/. Данные эмбеддинг получены с помощью модели glove, всего их 400 000 и размерность каждого беддинга равна 50. Для тех слов, у которых были готовые эмбединги, я использовал их, а для остальных слов, у которых их не оказалось использовал случайную инициализацию их эмбедингов. Во время обучения сети все эмбединги дообучались. Лучшее качество на тестовой выборке было получено при обучении 3-х эпох и составило 88.40%.

В итоге лучшее качество было получено при использовании логистической регрессии с кодированием каждого приложения с помощью с помощью метода TF-IDF. Такой результат может быть объяснен тем, что размер датасета был относительно небольшой, а поставленная задача достаточно простой.

Заключение

Целью данной работы было рассмотрение методов машинного обучения в задаче классификации текстов. В практической части была выполнена классификация отзывов на фильмы из датасета IMDB с помощью различных методов классического машинного обучения и рекуррентных нейронных сетей. В результате было получено, что на данном наборе данных решение с помощью классического машинного обучения не уступает решению с помощью более современных рекуррентных сетей.

В целом задача классификации текстов хорошо решается в данный момент и с появлением все новых методов машинного обучения, а в частности новых архитектур нейронных сетей, качество будет только становится все лучше и лучше. Безусловно, с развитием сферы машинного обучения и нахождением большего количества возможностей для его применений в бизнесе, задача классификации текстов будет становится только более востребованней.

Список литературы

[1] Text Classification by Lena Voita. — URL: https://lena-voita.github.io/nlp_course/text_classification.html (дата обращения: 15.12.2022). — Текст: электронный.

[2] Word Embeddings by Lena Voita. — URL: https://lena-voita.github.io/nlp_course/word_embeddings.html (дата обращения: 15.12.2022). — Текст: электронный.

[3] Pytorch bidirectional LSTM. — URL: https://www.youtube.com/watch?v=jGst43P-TJA (дата обращения: 15.12.2022). — Текст: электронный.

[4] Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *AAAI*, pages 2811–2818. AAAI Press, 2018.

[5] Edwin Chen. Exploring LSTMs. http://blog.echen.me/2017/05/30/exploring-lstms, 2017.

[6] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicoló Cesa-Bianchi, and Roman Garnett, editors, *NeurIPS*, pages 6572–6583, Dec 2018.

[7] Leon O. Chua and Lin Yang. Cellular neural networks: Applications. *IEEE Transactions on Circuits and Systems*, 35:1273–1290, 1988.

[8] Leon O. Chua and Lin Yang. Cellular neural networks: Theory. *IEEE Transactions on Circuits and Systems*, 35:1257–1272, 1988.

[9] Marco Ciccone, Marco Gallieri, Jonathan Masci, Christian Osendorfer, and Faustino J. Gomez. Nais-net: Stable deep networks from non-autonomous differential equations. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicoló Cesa-Bianchi, and Roman Garnett, editors, *NeurIPS*, pages 3029–3039, Dec 2018.

[10] B. de Vries and J. C. Principe. A theory for neural networks with time delays. In R. P. Lippmann, J. E. Moody, and D. S.
Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 162–168. Morgan Kaufmann, 1991.

[11] J.L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[12] Felix Gers. *Long Short-Term Memory in Recurrent Neural Networks*. PhD thesis, École Polytechnique Fédérale de
Lausanne, 2001.

[13] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network
architectures. *Neural Networks*, 18(5-6):602–610, 2005.

[14] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM networks. In *Proc. Int. Joint
Conf. on Neural Networks IJCNN 2005*, 2005.

[15] Alex Graves. *Supervised sequence labelling with recurrent neural networks.* PhD thesis, Technical University Munich,
2008.

Приложения

Загрузка библиотек

```
pip install w3lib
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting w3lib
  Downloading w3lib-2.1.1-py3-none-any.whl (21 kB)
Installing collected packages: w3lib
Successfully installed w3lib-2.1.1
```

```
import re
import nltk

import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.optim as optim

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from nltk.tokenize import word_tokenize
from sklearn.preprocessing import LabelEncoder

# Доп. библиотеки
import numpy as np
import torch.nn.functional as F
import matplotlib.pyplot as plt
import seaborn as sns
import os
from google.colab import drive
import copy
from tqdm import tqdm
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download("stopwords")

# Еще библиотеки
from IPython.display import HTML, IFrame
from w3lib.html import remove_tags
import spacy
from tqdm import tqdm_notebook
from nltk.tokenize.toktok import ToktokTokenizer

from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import GradientBoostingClassifier
```

```
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix, plot_confusion_matrix, plot_roc_curve,
plot_precision_recall_curve
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
drive.mount('/content/drive/')
main_path = "/content/drive/MyDrive/Университет/Курсовые/Курсовая.
Классификация текстов"
os.chdir(main_path)
```

```
Mounted at /content/drive/
```

EDA и предобработка данных

Первоначальный обзор данных

```
data = pd.read_csv("data/IMDB Dataset.csv")
data.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

```
data.tail()
```

	review	sentiment
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

```
# Смотрим на размерность нашего датасета
data.shape
```

```
(50000, 2)
```

```
# Смотрим на распределение классов целевого значения
data.sentiment.value_counts()
```

```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

```
# Проверяем пропущенные значения
data.isna().sum()
```

```
review       0
sentiment    0
dtype: int64
```

```
# Проверим, если повторяющиеся значения
(data
 .duplicated(keep='last')
 .sum())
```

```
418
```

```
# Удаление дубликатов
data.drop_duplicates(keep='first', inplace=True)
```

Предобработка

```
mapping = {"ain't": "is not", "aren't": "are not","can't": "cannot",
           "'cause": "because", "could've": "could have", "couldn't":
"could not",
           "didn't": "did not",  "doesn't": "does not", "don't": "do
not", "hadn't": "had not",
           "hasn't": "has not", "haven't": "have not", "he'd": "he
would","he'll": "he will",
           "he's": "he is", "how'd": "how did", "how'd'y": "how do
you", "how'll": "how will",
           "how's": "how is",  "I'd": "I would", "I'd've": "I would
have", "I'll": "I will",
           "I'll've": "I will have","I'm": "I am", "I've": "I have",
"i'd": "i would",
           "i'd've": "i would have", "i'll": "i will",  "i'll've": "i
will have",
           "i'm": "i am", "i've": "i have", "isn't": "is not", "it'd":
"it would",
           "it'd've": "it would have", "it'll": "it will", "it'll've":
"it will have",
           "it's": "it is", "let's": "let us", "ma'am": "madam",
"mayn't": "may not",
           "might've": "might have","mightn't": "might
not","mightn't've": "might not have",
           "must've": "must have", "mustn't": "must not",
"mustn't've": "must not have",
           "needn't": "need not", "needn't've": "need not
have","o'clock": "of the clock",
           "oughtn't": "ought not", "oughtn't've": "ought not have",
"shan't": "shall not",
```

```
            "sha'n't": "shall not", "shan't've": "shall not have",
"she'd": "she would",
            "she'd've": "she would have", "she'll": "she will",
"she'll've": "she will have",
            "she's": "she is", "should've": "should have", "shouldn't":
"should not",
            "shouldn't've": "should not have", "so've": "so
have","so's": "so as", "this's": "this is",
            "that'd": "that would", "that'd've": "that would have",
"that's": "that is",
            "there'd": "there would", "there'd've": "there would have",
"there's": "there is",
            "here's": "here is","they'd": "they would", "they'd've":
"they would have",
            "they'll": "they will", "they'll've": "they will have",
"they're": "they are",
            "they've": "they have", "to've": "to have", "wasn't": "was
not", "we'd": "we would",
            "we'd've": "we would have", "we'll": "we will", "we'll've":
"we will have",
            "we're": "we are", "we've": "we have", "weren't": "were
not",
            "what'll": "what will", "what'll've": "what will
have","what're": "what are",
            "what's": "what is", "what've": "what have", "when's":
"when is", "when've": "when have",
            "where'd": "where did", "where's": "where is", "where've":
"where have", "who'll": "who will",
            "who'll've": "who will have", "who's": "who is", "who've":
"who have", "why's": "why is",
            "why've": "why have", "will've": "will have", "won't":
"will not", "won't've": "will not have",
            "would've": "would have", "wouldn't": "would not",
"wouldn't've": "would not have",
            "y'all": "you all", "y'all'd": "you all
would","y'all'd've": "you all would have",
            "y'all're": "you all are","y'all've": "you all
have","you'd": "you would",
            "you'd've": "you would have", "you'll": "you will",
"you'll've": "you will have",
            "you're": "you are", "you've": "you have" }

#Spacy lemmatization
spacy_model = spacy.load("en_core_web_sm", disable = ['parser','ner'])

t = "you'reu"
mapping.get(t,t)

{"type":"string"}
```

```
def preprocess_text(text):
  text = remove_tags(text)
  text = text.lower()
  text = text = ' '.join([mapping.get(t,t) for t in text.split(" ")])
#Заменяем разгвоворные обороты речи
  text = re.sub("[^a-z]", " ", text)
  text = re.sub(r"\s+", ' ', text)
  doc = spacy_model(text)
  text = " ".join([token.lemma_ for token in doc if not
token.is_stop])
  return text
```

```
HTML(preprocess_text(data.review.iloc[1]))
```

```
<IPython.core.display.HTML object>
```

```
data["clean_review"] = data.review.apply(preprocess_text)
data.to_csv("./data/dataset_clean_4", index = False)
```

Последующий обзор данных

```
cdata = pd.read_csv("./data/dataset_clean_4")
cdata.sentiment = cdata.sentiment.map({"positive": 1,
                                       "negative": 0})
cdata = cdata.iloc[:, [0, 2, 1]]
cdata.head()
```

```
                                              review  \
0  One of the other reviewers has mentioned that ...
1  A wonderful little production. <br /><br />The...
2  I thought this was a wonderful way to spend ti...
3  Basically there's a family where a little boy ...
4  Petter Mattei's "Love in the Time of Money" is...

                                        clean_review  sentiment
0  reviewer mention watch oz episode hook right e...          1
1  wonderful little production film technique una...          1
2  think wonderful way spend time hot summer week...          1
3  basically family little boy jake think zombie ...          0
4  petter mattei s love time money visually stunn...          1
```

```
HTML(cdata.review.iloc[1])
```

```
<IPython.core.display.HTML object>
```

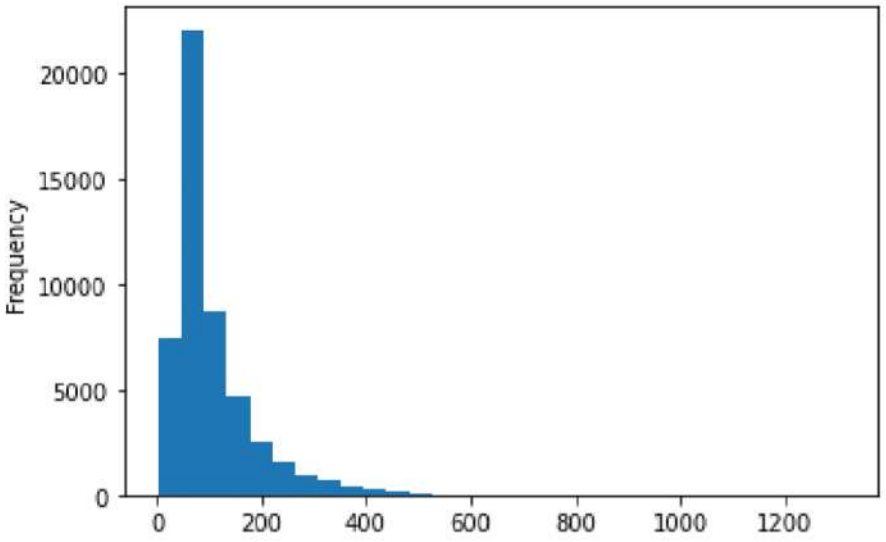
```
HTML(cdata.clean_review.iloc[1])
```

```
<IPython.core.display.HTML object>
```

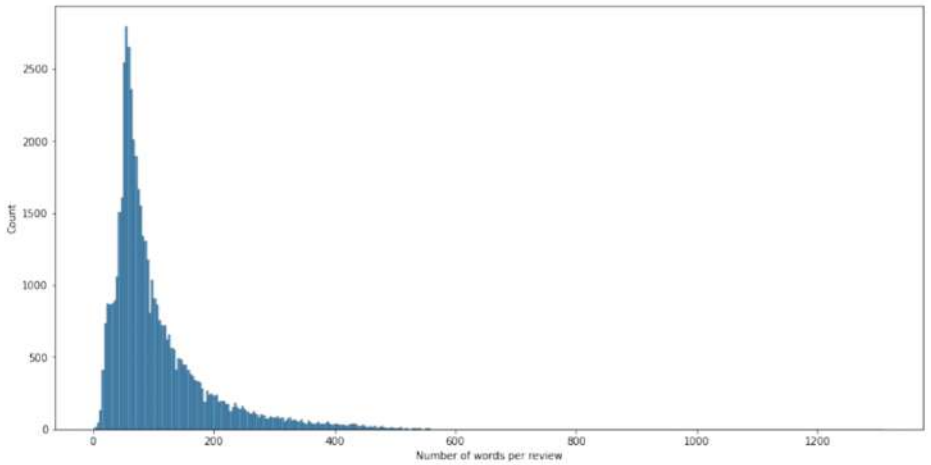
```
cdata.drop("review", axis=1, inplace=True)
```

```
cdata.clean_review.str.split().apply(lambda x:
len(x)).plot.hist(bins=30)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f079ed019a0>
```



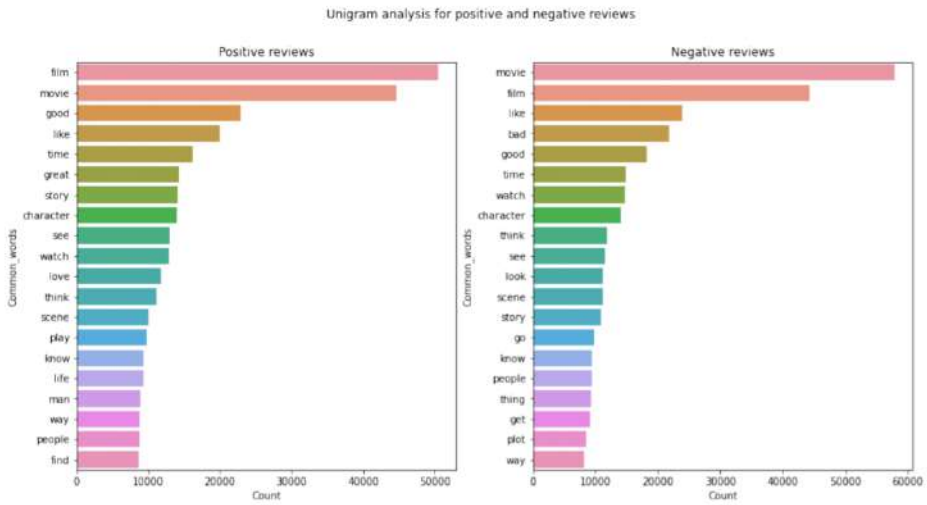

```
# Распределение количества слов в отзыве
scale = 0.5
fig, ax = plt.subplots(figsize=(30*scale,15*scale))
sns.histplot(cdata.clean_review.str.split().apply(lambda x:
len(x)).tolist(), ax=ax)
ax.set_xlabel('Number of words per review')
plt.show()
```



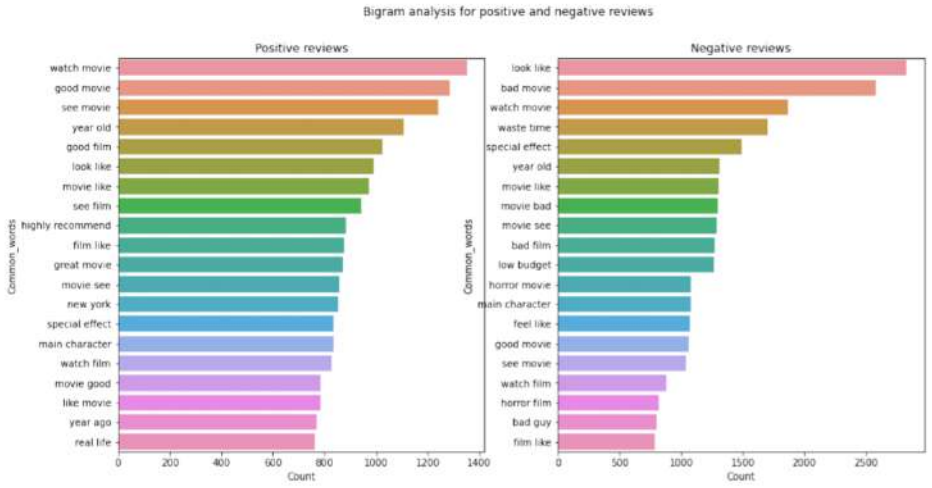

```
rev_len = cdata.clean_review.str.split().apply(lambda x: len(x))
rev_len.describe()
```

```
count    49582.000000
mean       104.595519
std         80.630538
min          3.000000
25%         55.000000
50%         77.000000
75%        127.000000
max       1310.000000
Name: clean_review, dtype: float64
```

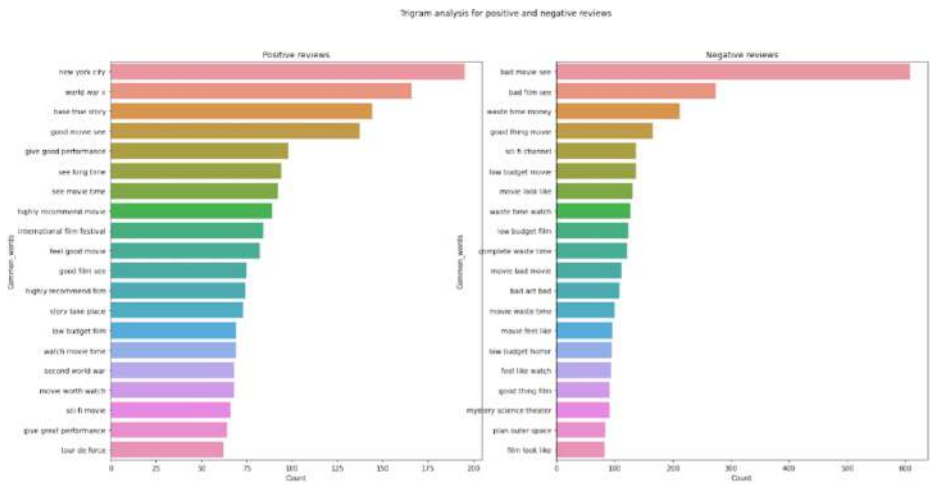
```
def get_ngrams(review, n, g):
    vec = CountVectorizer(ngram_range=(g, g)).fit(review)
    bag_of_words = vec.transform(review) #sparse matrix of
count_vectorizer
    sum_words = bag_of_words.sum(axis=0) #total number of words
    sum_words = np.array(sum_words)[0].tolist() #convert to list
    words_freq = [(word, sum_words[idx]) for word, idx in
vec.vocabulary_.items()] #get word freqency for word location in count
vec
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
#key is used to perform sorting using word_freqency
    return words_freq[:n]

pos_data = cdata[cdata.sentiment==1].clean_review
neg_data = cdata[cdata.sentiment==0].clean_review

#unigrams
scale = 0.5
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(30*scale,15*scale))
uni_positive = get_ngrams(pos_data,20,1)
uni_positive = dict(uni_positive)
temp = pd.DataFrame(list(uni_positive.items()), columns =
["Common_words" , 'Count'])
sns.barplot(data = temp, x="Count", y="Common_words", orient='h',ax =
ax1)
ax1.set_title('Positive reviews')
uni_negative = get_ngrams(neg_data,20,1)
uni_negative = dict(uni_negative)
temp = pd.DataFrame(list(uni_negative.items()), columns =
["Common_words" , 'Count'])
sns.barplot(data = temp, x="Count", y="Common_words", orient='h',ax =
ax2)
ax2.set_title('Negative reviews')
fig.suptitle('Unigram analysis for positive and negative reviews')
plt.show()
```




```
# bigrams
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(30*scale,15*scale))
bi_positive = get_ngrams(pos_data,20,2)
bi_positive = dict(bi_positive)
temp = pd.DataFrame(list(bi_positive.items()), columns =
["Common_words" , 'Count'])
sns.barplot(data = temp, x="Count", y="Common_words", orient='h',ax =
ax1)
ax1.set_title('Positive reviews')
bi_negative = get_ngrams(neg_data,20,2)
bi_negative = dict(bi_negative)
temp = pd.DataFrame(list(bi_negative.items()), columns =
["Common_words" , 'Count'])
sns.barplot(data = temp, x="Count", y="Common_words", orient='h',ax =
ax2)
ax2.set_title('Negative reviews')
fig.suptitle('Bigram analysis for positive and negative reviews')
plt.show()
```




```
# Trigrams
scale = 0.7
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(30*scale,15*scale))
tri_positive = get_ngrams(pos_data,20,3)
tri_positive = dict(tri_positive)
temp = pd.DataFrame(list(tri_positive.items()), columns =
["Common_words" , 'Count'])
sns.barplot(data = temp, x="Count", y="Common_words", orient='h',ax =
ax1)
ax1.set_title('Positive reviews')
tri_negative = get_ngrams(neg_data,20,3)
tri_negative = dict(tri_negative)
temp = pd.DataFrame(list(tri_negative.items()), columns =
["Common_words" , 'Count'])
sns.barplot(data = temp, x="Count", y="Common_words", orient='h',ax =
ax2)
ax2.set_title('Negative reviews')
fig.suptitle('Trigram analysis for positive and negative reviews')
plt.show()
```



ML

```
#splitting into train and test
train, test= train_test_split(cdata, test_size=0.3, random_state=42)
Xtrain, ytrain = train['clean_review'], train['sentiment']
Xtest, ytest = test['clean_review'], test['sentiment']

#Vectorizing data

tfidf_vect = TfidfVectorizer() #tfidfVectorizer
Xtrain_tfidf = tfidf_vect.fit_transform(Xtrain)
Xtest_tfidf = tfidf_vect.transform(Xtest)


count_vect = CountVectorizer() # CountVectorizer
Xtrain_count = count_vect.fit_transform(Xtrain)
Xtest_count = count_vect.transform(Xtest)

res = pd.DataFrame({"Model": [], "Accuracy": []})
res
```

```
Empty DataFrame
Columns: [Model, Accuracy]
Index: []
```

Logistic Regression

```
lr = LogisticRegression()
lr.fit(Xtrain_tfidf,ytrain)
p1=lr.predict(Xtest_tfidf)
s1=accuracy_score(ytest,p1)
print("Logistic Regression Accuracy :", "{:.2f}%".format(100*s1))
```

```
Logistic Regression Accuracy : 88.97%
```

```
res = res.append({"Model": "Log Regression",
                  "Accuracy": "{:.2f}%".format(100*s1)},
ignore_index=True)
```

Naive Bayes

```
mnb= MultinomialNB()
mnb.fit(Xtrain_count,ytrain)
p2=mnb.predict(Xtest_count)
s2=accuracy_score(ytest,p2)
print("Multinomial Naive Bayes Classifier Accuracy :", "{:.2f}
%".format(100*s2))
```

```
Multinomial Naive Bayes Classifier Accuracy : 85.28%
```

```
res = res.append({"Model": "Naive Bayes",
                  "Accuracy": "{:.2f}%".format(100*s2)},
ignore_index=True)

res.style.hide_index()
```

```
<pandas.io.formats.style.Styler at 0x7f704d8ffc70>
```

Deep learning

```
Xtrain_len = Xtrain.str.split().apply(lambda x: len(x))
Xtrain_len.describe()
```

```
count    34707.000000
mean       104.692252
std         80.691193
min          3.000000
25%         55.000000
50%         77.000000
75%        127.000000
max       1007.000000
Name: clean_review, dtype: float64
```

```
max_seq_len = int(Xtrain_len.mean() + 2 * Xtrain_len.std())
max_seq_len
```

```
266
```

```
perc_covered = np.sum(Xtrain_len < max_seq_len) / len(Xtrain_len)*100
print('The above calculated number coveres approximately {} % of
data'.format(np.round(perc_covered,2)))
```

```
The above calculated number coveres approximately 94.49 % of data
```

```
class Vocab:
  def __init__(self, data: list, max_seq_len):
    st = set()
    self.max_seq_len = max_seq_len
    for x in data:
```

```
      rev = nltk.word_tokenize(x)
      st.update(rev)
    self.unk_token = '<UNK>'
    self.pad_token = '<PAD>'
    #unique_words = [self.pad_token, self.unk_token] + sorted(list(st
- set(stopwords.words('english'))))
    unique_words = [self.pad_token, self.unk_token] + sorted(list(st))
    word_ind = list(np.arange(len(unique_words)))
    self.idx_to_token = dict(zip(word_ind, unique_words))
    self.token_to_idx = dict(zip(unique_words, word_ind))
    self.vocab_len = len(unique_words)


class ReviewDataset(Dataset):
  def __init__(self, X, y, vocab: Vocab):
    self.X = X
    self.y = y
    self.vocab = vocab

  def get_idx(self, review):
    review = nltk.word_tokenize(review)
    pad_idx = self.vocab.token_to_idx[self.vocab.pad_token]
    unk_idx = self.vocab.token_to_idx[self.vocab.unk_token]
    idx = [pad_idx] * self.vocab.max_seq_len
    n = min(len(review), self.vocab.max_seq_len)
    for i in range(n):
      idx[i] = self.vocab.token_to_idx.get(review[i], unk_idx)


    return torch.tensor(idx, dtype=torch.int64)


  def __len__(self):
    return len(self.X)

  def __getitem__(self, idx):
    #return self.X.iloc[idx], self.get_idx(self.X.iloc[idx]),
self.y.iloc[idx]
    return self.get_idx(self.X.iloc[idx]), self.y.iloc[idx]

vocab = Vocab(Xtrain.tolist(), max_seq_len)
review_train = ReviewDataset(Xtrain, ytrain, vocab)
review_test = ReviewDataset(Xtest, ytest, vocab)

batch_size = 32
train_iter = DataLoader(review_train, batch_size, shuffle=True)
test_iter = DataLoader(review_test, batch_size, shuffle=False)

vocab.vocab_len
```

73569

```
# v2

def number_of_correct_ans(y_hat, y):
  # Считаем кол-во верных ответов (работает для ответов - векторов)
  y_hat = y_hat.argmax(axis=1)
  correct_mask = y_hat == y
  return correct_mask.sum()


def evaluate_accuracy(net, test_iter):

  # Выставляем режим оценки модели
  net.eval()

  total_samples = len(test_iter.dataset) #Общее кол-во элементво в
выборке
  correct_ans = 0

  with torch.no_grad():
    for X, y in test_iter:
      X, y = X.to(device), y.to(device)

      y_hat = net(X)
      correct_ans += number_of_correct_ans(y_hat, y)

  return correct_ans / total_samples


def train_epoch(net, train_iter, loss, updater, clip_value):

  # Выставляем режим обучения модели
  net.train()

  total_samples = len(train_iter.dataset)
  correct_samples = 0
  total_loss = 0

  for X, y in train_iter:
    X, y = X.to(device), y.to(device)

    y_hat = net(X)
    l = loss(y_hat, y)

    updater.zero_grad()
    l.backward()
    if clip_value:
      torch.nn.utils.clip_grad_norm_(net.parameters(), clip_value)
```

```
      updater.step()

      total_loss += l
      correct_samples += number_of_correct_ans(y_hat, y)

    return total_loss / total_samples, correct_samples / total_samples


class Model_metrics:
  def __init__(self):
    self.train_loss = []
    self.train_accuracy = []
    self.test_accuracy = []

  def add(self, train_loss, train_accuracy, test_accuracy):
      self.train_loss.append(train_loss.cpu().item())
      self.train_accuracy.append(train_accuracy.cpu().item())
      self.test_accuracy.append(test_accuracy.cpu().item())

  def plot(self):
      plt.rcParams.update({'font.size': 12})
      fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))
      epochs = np.arange(len(self.train_loss))

      # График функции потерь
      ax[0].plot(epochs, self.train_loss, label="train loss")
      ax[0].set_xticks(epochs)
      ax[0].grid()
      ax[0].legend()

      # График точности на обучении и тесте
      ax[1].plot(epochs, self.train_accuracy, label="train accuracy")
      ax[1].plot(epochs, self.test_accuracy, label="test accuracy")
      ax[1].set_xticks(epochs)
      ax[1].grid()
      ax[1].legend()

      plt.show()


  def return_dict(self):
    res = {
        "train_loss": self.train_loss,
        "train_accuracy": self.train_accuracy,
        "test_accuracy": self.test_accuracy
    }
    return res
```

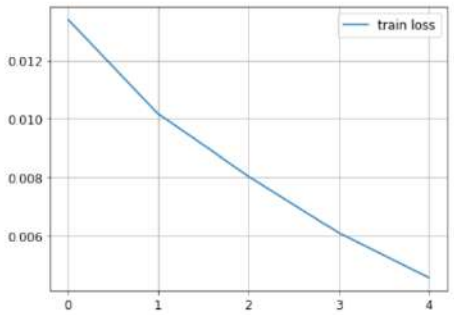
```
def saving(net, model_metrics, path):
  state = {"state_dict": net.state_dict()}
  metrics_dict = model_metrics.return_dict()
  state = {**state, **metrics_dict}
  torch.save(state, path)


def train_rnn_model(net, train_iter, test_iter, loss,
                num_epochs, updater, clip_value = None, path=None):

  model_metrics = Model_metrics()
  for epoch in tqdm(range(num_epochs)):
    train_loss, train_accuracy = train_epoch(net, train_iter, loss,
updater, clip_value)
    test_accuracy = evaluate_accuracy(net, test_iter)
    model_metrics.add(train_loss, train_accuracy, test_accuracy)
    print (f': Loss: {train_loss:.4f}, Train acc:
{train_accuracy:.4f}, Test acc: {test_accuracy:.4f}')
  model_metrics.plot()
  if path:
    saving(net, model_metrics, path)
```

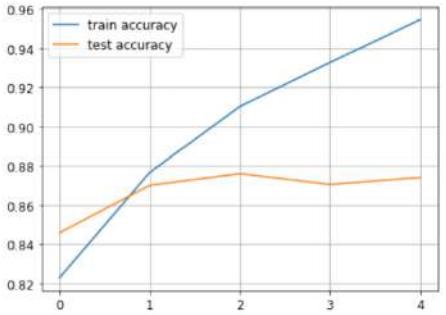
Two layer GRU

```
class GRU_model(nn.Module):
  def __init__(self, vocab_len, embedding_size, rnn_hidden_size,
               num_classes, seq_len, num_layers=1, bidir=False):
    super().__init__()
    self.hidden_dim = rnn_hidden_size
    self.seq_len = seq_len
    self.num_layers = num_layers
    self.bidir = bidir
    self.embedding = nn.Embedding(vocab_len, embedding_size,
padding_idx=0)
    self.rnn = nn.GRU(embedding_size, rnn_hidden_size,
                      num_layers = num_layers, batch_first=True,
bidirectional = bidir)
    self.classifier = nn.Sequential(nn.Linear(rnn_hidden_size  * (1 +
self.bidir),

rnn_hidden_size),nn.ReLU(), nn.Dropout(0.5),
                                    nn.Linear(rnn_hidden_size,
num_classes))
    self.dropout = nn.Dropout(0.5)

  def forward(self, X):
    #h0 = torch.zeros((1 + self.bidir) * numm, X.size(0),
self.hidden_dim).to(device)
    batch_size = X.shape[0]

    emb = self.embedding(X) # batch x seq x emb
```

```
        out, hidden = self.rnn(emb)
        out = out.sum(axis=1)
        out = self.dropout(out) # batch x rnn_hidden
        out = self.classifier(out) # batch x num_classes
        return out


vocab_len = vocab.vocab_len
embedding_size = 64
rnn_hidden_size = 256
num_classes = 2
seq_len = vocab.max_seq_len
net = GRU_model(vocab_len, embedding_size, rnn_hidden_size,
                    num_classes, seq_len, num_layers = 2)
net.to(device)

loss = nn.CrossEntropyLoss()
lr = 0.001
updater = optim.Adam(net.parameters(), lr=lr)
clip_value = 5

num_epochs = 5

#Обучение сети
path = "./Saved models/GRU_code_2_1.pt"
train_rnn_model(net, train_iter, test_iter, loss, num_epochs, updater,
clip_value, path)
```

```
 20%|██        | 1/5 [01:02<04:11, 62.98s/it]
: Loss: 0.0134, Train acc: 0.8229, Test acc: 0.8458
  40%|████      | 2/5 [02:00<02:59, 59.97s/it]
: Loss: 0.0102, Train acc: 0.8766, Test acc: 0.8701
  60%|██████    | 3/5 [02:59<01:59, 59.58s/it]
: Loss: 0.0080, Train acc: 0.9103, Test acc: 0.8761
  80%|████████  | 4/5 [03:57<00:58, 58.72s/it]
: Loss: 0.0061, Train acc: 0.9328, Test acc: 0.8706
100%|██████████| 5/5 [04:55<00:00, 59.08s/it]
: Loss: 0.0045, Train acc: 0.9547, Test acc: 0.8742
```






```
state = torch.load(path, map_location=device)
net.load_state_dict(state["state_dict"])
acc = evaluate_accuracy(net, test_iter)
print("Точность на тестовом множестве:", "{:.2f}%".format(100*acc))
```

```
Точность на тестовом множестве: 87.42%
```

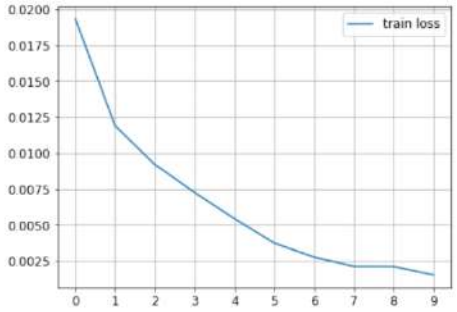
```
res = res.append({"Model": "Two layer gru",
                  "Accuracy": "{:.2f}%".format(100*acc)},
ignore_index=True)
```

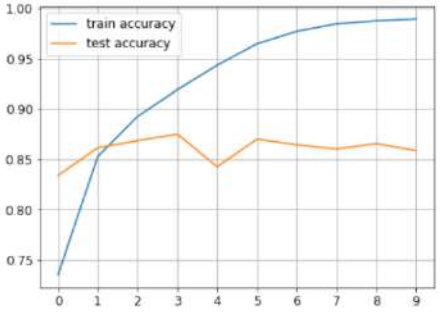
3 layer gru

```
vocab_len = vocab.vocab_len
embedding_size = 64
rnn_hidden_size = 256
num_classes = 2
seq_len = vocab.max_seq_len
net = GRU_model(vocab_len, embedding_size, rnn_hidden_size,
                num_classes, seq_len, num_layers = 3)
net.to(device)

loss = nn.CrossEntropyLoss()
lr = 0.001
updater = optim.Adam(net.parameters(), lr=lr)
clip_value = 5

path = "./Saved models/GRU_code_2_3.pt"
num_epochs = 10

#Обучение сети
train_rnn_model(net, train_iter, test_iter, loss, num_epochs, updater,
clip_value, path)
```

```
 10%|          | 1/10 [01:13<11:00, 73.37s/it]

: Loss: 0.0193, Train acc: 0.7351, Test acc: 0.8337

  20%|          | 2/10 [02:24<09:35, 71.88s/it]

: Loss: 0.0119, Train acc: 0.8526, Test acc: 0.8614
```

```
  30%|███       | 3/10 [03:34<08:17, 71.10s/it]
: Loss: 0.0092, Train acc: 0.8925, Test acc: 0.8684
  40%|████      | 4/10 [04:46<07:08, 71.38s/it]
: Loss: 0.0072, Train acc: 0.9189, Test acc: 0.8749
  50%|█████     | 5/10 [05:56<05:54, 70.92s/it]
: Loss: 0.0054, Train acc: 0.9433, Test acc: 0.8424
  60%|██████    | 6/10 [07:06<04:43, 70.77s/it]
: Loss: 0.0037, Train acc: 0.9646, Test acc: 0.8700
  70%|███████   | 7/10 [08:18<03:33, 71.17s/it]
: Loss: 0.0028, Train acc: 0.9770, Test acc: 0.8643
  80%|████████  | 8/10 [09:29<02:21, 70.98s/it]
: Loss: 0.0021, Train acc: 0.9845, Test acc: 0.8602
  90%|█████████ | 9/10 [10:46<01:12, 72.99s/it]
: Loss: 0.0021, Train acc: 0.9874, Test acc: 0.8655
100%|██████████| 10/10 [12:04<00:00, 72.45s/it]
: Loss: 0.0015, Train acc: 0.9893, Test acc: 0.8585
```





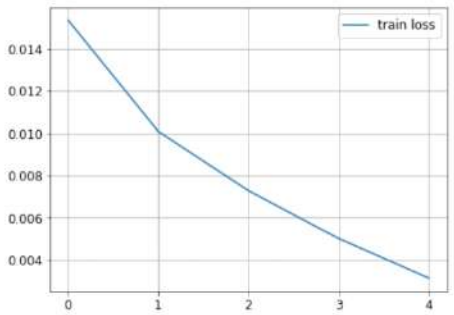
Concat GRU

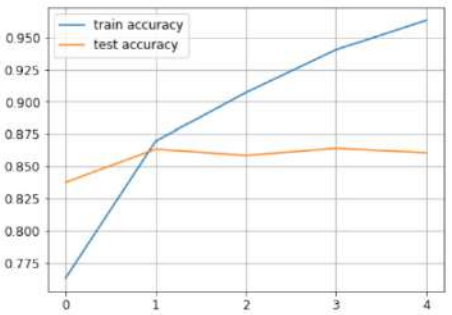
```python
class GRU_model(nn.Module):
  def __init__(self, vocab_len, embedding_size, rnn_hidden_size,
               num_classes, seq_len, num_layers=1, bidir=False):
    super().__init__()
    self.hidden_dim = rnn_hidden_size
    self.seq_len = seq_len
    self.num_layers = num_layers
```

```
    self.bidir = bidir
    self.embedding = nn.Embedding(vocab_len, embedding_size,
padding_idx=0)
    self.rnn = nn.GRU(embedding_size, rnn_hidden_size,
                      num_layers = num_layers, batch_first=True,
bidirectional = bidir)
    self.classifier = nn.Sequential(nn.Linear(rnn_hidden_size  * (1 +
self.bidir) * self.seq_len,

rnn_hidden_size),nn.ReLU(), nn.Dropout(0.5),
                                    nn.Linear(rnn_hidden_size,
num_classes))
    self.dropout = nn.Dropout(0.5)

  def forward(self, X):
    #h0 = torch.zeros((1 + self.bidir) * numm, X.size(0),
self.hidden_dim).to(device)
    batch_size = X.shape[0]

    emb = self.embedding(X) # batch x seq x emb
    out, hidden = self.rnn(emb)
    out = out.reshape(batch_size, self.hidden_dim * self.seq_len)
    out = self.dropout(out) # batch x rnn_hidden
    out = self.classifier(out) # batch x num_classes
    return out


vocab_len = vocab.vocab_len
embedding_size = 64
rnn_hidden_size = 256
num_classes = 2
seq_len = vocab.max_seq_len
net = GRU_model(vocab_len, embedding_size, rnn_hidden_size,
                num_classes, seq_len, num_layers = 2)
net.to(device)

loss = nn.CrossEntropyLoss()
lr = 0.001
updater = optim.Adam(net.parameters(), lr=lr)
clip_value = 5

num_epochs = 5

#Обучение сети
path = "./Saved models/GRU_code_2_concat.pt"
train_rnn_model(net, train_iter, test_iter, loss, num_epochs, updater,
clip_value, path)
```

```
 20%|██        | 1/5 [00:57<03:50, 57.61s/it]
```

```
: Loss: 0.0154, Train acc: 0.7631, Test acc: 0.8376
  40%|████      | 2/5 [01:55<02:53, 57.76s/it]
: Loss: 0.0101, Train acc: 0.8694, Test acc: 0.8633
  60%|██████    | 3/5 [02:54<01:56, 58.30s/it]
: Loss: 0.0073, Train acc: 0.9071, Test acc: 0.8584
  80%|████████  | 4/5 [03:51<00:58, 58.00s/it]
: Loss: 0.0050, Train acc: 0.9403, Test acc: 0.8640
100%|██████████| 5/5 [04:50<00:00, 58.03s/it]
: Loss: 0.0031, Train acc: 0.9632, Test acc: 0.8606
```





Pre-trained deep learning

```
words_path = "../../ТОБД/2 семестр/Задание
7/embeddings/glove.6B.50d.txt"
with open(words_path) as f:
    lines = f.readlines()

words = set()
emb_dim = 50
embeddings = np.empty((len(lines), emb_dim))
word2idx = dict()
for idx, line in enumerate(lines):
  line = line.split()
  word = line[0]
  words.add(word)
  embeddings[idx] = np.array(line[1:]).astype(float)
  word2idx[word] = idx

len(words)
```

```
400000
```

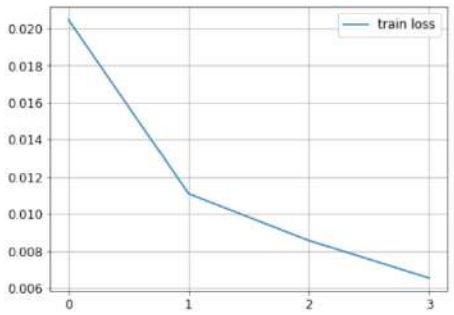
```
vocab_emb = np.empty((vocab.vocab_len, emb_dim))
missed_words = 0
for idx, word in enumerate(vocab.idx_to_token.values()):
  if word in words:
    word_idx = word2idx[word]
    vocab_emb[idx] = embeddings[word_idx]
  else:
    vocab_emb[idx] = np.random.normal(scale=0.6, size=(emb_dim, ))
    missed_words += 1

missed_words
```

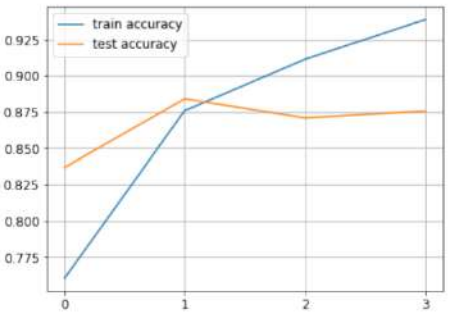
2 layer GRU with pretrained word embeddings

```
class GRU_model(nn.Module):
  def __init__(self, vocab_len, embedding_size, rnn_hidden_size,
               num_classes, seq_len, num_layers=1, bidir=False,
pr_embs=None):
    super().__init__()
    self.hidden_dim = rnn_hidden_size
    self.seq_len = seq_len
    self.num_layers = num_layers
    self.bidir = bidir
    if pr_embs.any():
      self.embedding =
torch.nn.Embedding.from_pretrained(torch.from_numpy(pr_embs).float(),

freeze=False, padding_idx=0)
    else:
      self.embedding = nn.Embedding(vocab_len, embedding_size,
padding_idx=0)
    self.rnn = nn.GRU(embedding_size, rnn_hidden_size,
                      num_layers = num_layers, batch_first=True,
bidirectional = bidir)
    self.classifier = nn.Sequential(nn.Linear(rnn_hidden_size  * (1 +
self.bidir),

rnn_hidden_size),nn.ReLU(), nn.Dropout(0.5),
                                    nn.Linear(rnn_hidden_size,
num_classes))
    self.dropout = nn.Dropout(0.5)

  def forward(self, X):
    #h0 = torch.zeros((1 + self.bidir) * numm, X.size(0),
self.hidden_dim).to(device)
    batch_size = X.shape[0]

    emb = self.embedding(X) # batch x seq x emb
    out, hidden = self.rnn(emb)
    out = out.sum(axis=1)
    out = self.dropout(out) # batch x rnn_hidden
```

```
    out = self.classifier(out) # batch x num_classes
    return out


vocab_len = vocab.vocab_len
embedding_size = 50
rnn_hidden_size = 256
num_classes = 2
seq_len = vocab.max_seq_len
net = GRU_model(vocab_len, embedding_size, rnn_hidden_size,
                num_classes, seq_len, num_layers = 2,
pr_embs=vocab_emb)
net.to(device)

loss = nn.CrossEntropyLoss()
lr = 0.001
updater = optim.Adam(net.parameters(), lr=lr)
clip_value = 5

num_epochs = 4

#Обучение сети
path = "./Saved models/GRU_code_2_pre_2.pt"
train_rnn_model(net, train_iter, test_iter, loss, num_epochs, updater,
clip_value, path)
```

```
 25%|██        | 1/4 [00:57<02:53, 57.80s/it]

: Loss: 0.0205, Train acc: 0.7605, Test acc: 0.8368

  50%|█████     | 2/4 [01:53<01:53, 56.59s/it]

: Loss: 0.0111, Train acc: 0.8759, Test acc: 0.8840

  75%|███████   | 3/4 [02:49<00:56, 56.39s/it]

: Loss: 0.0086, Train acc: 0.9114, Test acc: 0.8709

100%|██████████| 4/4 [03:46<00:00, 56.72s/it]

: Loss: 0.0065, Train acc: 0.9389, Test acc: 0.8756
```






```
state = torch.load(path, map_location=device)
net.load_state_dict(state["state_dict"])
acc = evaluate_accuracy(net, test_iter)
print("Точность на тестовом множестве:", "{:.2f}%".format(100*acc))
```

```
Точность на тестовом множестве: 87.56%
```