ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**(ФИНАНСОВЫЙ УНИВЕРСИТЕТ)**

Департамент анализа данных

и машинного обучения

***Дисциплина: «Технологии анализа данных и машинного обучения»***

*Направление подготовки: «Прикладная информатика»*

*Факультет информационных технологий и анализа больших данных*

*Форма обучения очная*

*Учебный 2022/2023 год, 6 семестр*

**Курсовая работа на тему:**

«Определение эмоциональной окраски тестовых данных с помощью искусственных нейронных сетей»

*Выполнил(а):*

студент(ка) группы ПИ20-3

Казьмин О. А.

*Научный руководитель:*

доцент Кублик Е. И.

**Москва 2022**

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ........................................................................................................................3 Глава1: Подготовка данных.............................................................................................5  
Глава 2: Построение моделей........................................................................................13  
Глава 3: Сравнение моделей..........................................................................................28  
ЗАКЛЮЧЕНИЕ...............................................................................................................32  
СПИСОК ЛИТЕРАТУРЫ И ИНТЕРНЕТ-РЕСУРСОВ...............................................34

ВВЕДЕНИЕ.

Анализ настроений в социальных медиа-сетях является важным исследовательским направлением, которое находит применение в различных сферах, от мониторинга общественного мнения до оценки реакций на новости и продукты. В этом контексте, определение тональности текстов, выражающих мнения и эмоции пользователей, играет ключевую роль.

Существует несколько подходов к анализу настроений, одними из которых являются модели, основанные на рекуррентных нейронных сетях с долгой краткосрочной памятью (LSTM) и сверточных нейронных сетях (CNN). Обе модели успешно применяются в задачах классификации текстов и имеют свои преимущества в обработке и понимании естественного языка.

В данном примере мы представляем две модели для анализа настроений в Twitter: LSTM и CNN. Обе модели обучаются на большом наборе данных с размеченными текстами, отражающими различные тональности - положительные, отрицательные и нейтральные. Затем модели оцениваются на тестовом наборе данных, и их результаты визуализируются для анализа производительности и сравнения эффективности каждой модели. Модели будут созданы с помощью открытой библиотеки глубокого обучения Keras, так как она предоставляет простой и интуитивно понятный интерфейс для создания моделей глубокого обучения, в том числе сверточных и рекурентных моделей. Также, для сравнения, будет представлена модель машинного обучения (логистическая регрессия)

В рамках работы был выполнен следующий план:

* Подготовка датасета: был собран и подготовлен набор данных, содержащий текстовые сообщения из Twitter, размеченные по тональности. Этот этап включал сбор данных, их очистку от шума и нежелательных символов, а также разделение на обучающий и тестовый наборы.
* Токенизация: Текстовые данные были преобразованы в последовательности токенов, где каждый токен представляет отдельное слово или символ. Этот шаг позволяет представить текст в формате, понятном для моделей обработки естественного языка.
* Построение моделей: были разработаны и обучены две модели - LSTM и CNN - для анализа настроений в текстах из Twitter. Каждая модель была настроена с использованием оптимальных гиперпараметров и обучена на обучающем наборе данных.
* Оценка производительности: Обе модели, LSTM и CNN, были оценены на тестовом наборе данных для анализа их производительности. Метрики, такие как точность, полнота и F-мера, были использованы для оценки качества классификации и сравнения результатов двух моделей.
* Сравнение результатов: Результаты работы моделей были сопоставлены и визуализированы, чтобы определить, какая модель лучше справляется с задачей анализа настроений в Twitter. Были проведены сравнения производительности, анализ ошибок и исследование важности различных признаков для классификации.
* Обсуждение и выводы: В заключительной части работы были представлены обсуждение результатов, выявленные трудности и ограничения моделей, а также возможные пути для улучшения и дальнейших исследований. Также были сделаны выводы о применимости LSTM и CNN в задаче анализа настроений в социальных медиа-сетях и их относительной эффективности.

Таким образом, в данной работе мы представляем исследование по анализу настроений в Twitter с использованием моделей LSTM и CNN. Мы описываем подготовку датасета, процесс токенизации, обучение и оценку моделей, а также сравнение и обсуждение результатов. Это исследование поможет расширить понимание методов анализа настроений в социальных медиа-сетях и может быть полезным в различных приложениях, связанных с обработкой и пониманием текстовых данных.

Глава 1: Подготовка данных

В данном разделе мы рассмотрим процесс подготовки данных для задачи анализа настроений в тексте. Подготовка данных является важным этапом, который включает несколько шагов, таких как сбор и предобработка текстовых данных, разделение данных на обучающую и тестовую выборки, а также преобразование текстов в числовые представления, понятные для моделей LSTM и CNN.

1. Сбор данных: первым шагом является сбор необходимых данных для обучения моделей. Для задачи анализа настроений в тексте нам понадобятся текстовые данные, которые содержат информацию о настроении (негативное, позитивное) для каждого текста. Данные могут быть собраны из различных источников, таких как социальные сети, интернет-форумы, новостные статьи и другие. В этой работе используется уже готовый датасет (Рис. 1, 2), собранный из сообщений из соцсети Twitter с оценкой каждого твита.



Рис. 1 - Загрузка датасета

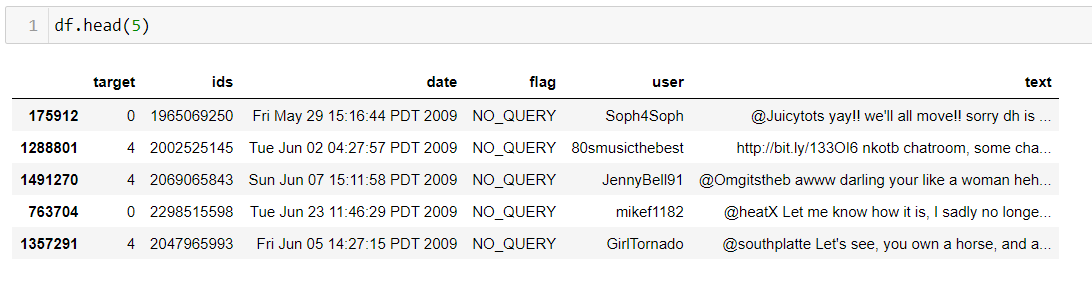


Рис. 2 - Примеры строк

2. Предобработка данных (Рис. 3): после сбора данных следует их предобработка. Этот шаг включает удаление лишних символов, приведение текста к нижнему регистру, удаление стоп-слов (часто встречающихся слов, которые не несут смысловой нагрузки) и другие операции для очистки и стандартизации данных. Предобработка данных помогает улучшить качество моделей и облегчает последующие этапы обработки.



Рис. 3 - Предобработка данных

3. Разделение на обучающую и тестовую выборки (Рис. 4): чтобы оценить производительность моделей, необходимо разделить данные на обучающую и тестовую выборки. Обучающая выборка будет использоваться для обучения моделей, а тестовая выборка - для оценки их точности. Разделение данных позволяет проверить, насколько хорошо модели обобщают информацию и способны делать предсказания на новых, неизвестных данных.

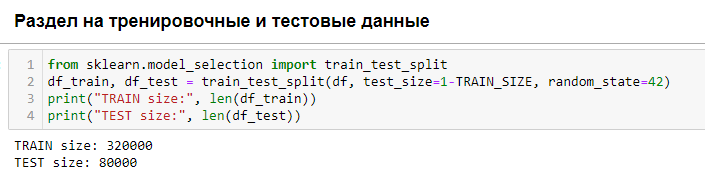


Рис. 4 - Разделение

4. Токенизация и векторизация текста: Модели LSTM и CNN работают с числовыми данными, поэтому необходимо преобразовать текстовые данные в числовые представления. Токенизация является процессом разделения текста на отдельные слова или токены. Затем каждому токену присваивается уникальный числовой идентификатор. Далее, текстовые данные могут быть векторизованы, то есть преобразованы в числовые векторы фиксированной длины. Существуют различные методы векторизации, такие как мешок слов (Bag of Words) и методы на основе эмбеддингов, например, Word2Vec или GloVe. Эти методы позволяют представить слова в виде числовых векторов, учитывая их семантический контекст и отношения между словами. В этой работе используется Word2Vec (Рис. 5).

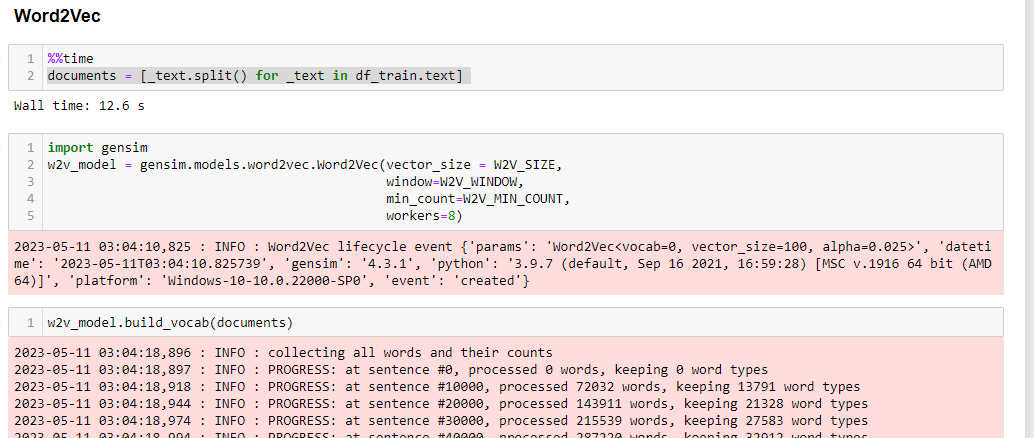


Рис. 5 – Word2Vec

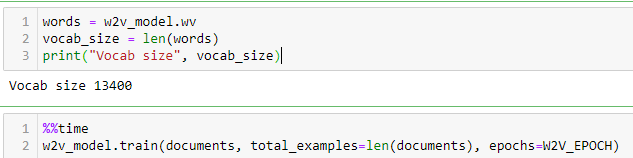


Рис. 6

4.1. Создается список documents (Рис. 5), содержащий предложения, разделенные на отдельные слова. Разделение происходит путем разделения текста (text) из датафрейма (df\_train.text) на отдельные слова с использованием метода split(). В результате каждое предложение становится списком слов.

4.2. Импортируется модуль gensim, который является библиотекой для обработки текстов и моделирования естественного языка.

4.3. Создается экземпляр модели Word2Vec с использованием класса Word2Vec из модуля gensim.models.word2vec. Параметры модели передаются в конструкторе модели:

* vector\_size определяет размерность векторов слов, которые модель будет обучать.
* window задает размер окна, в пределах которого модель анализирует контекст слова.
* min\_count определяет минимальную частоту встречаемости слова, чтобы оно было включено в словарь модели.
* workers определяет количество потоков, используемых для обучения модели.

4.4. С помощью метода build\_vocab() модель строит словарь слов, основанный на documents. Словарь содержит уникальные слова из всех предложений и их частоты встречаемости.

4.5. Получается доступ к векторам слов с помощью w2v\_model.wv (Рис. 6). Векторы слов используются для представления слов в числовом виде, который может быть использован для обучения моделей машинного обучения.

4.6. Вычисляется размер словаря vocab\_size путем получения длины words, который является объектом KeyedVectors и представляет собой словарь слов и соответствующих им векторов.

4.7. И, наконец, метод train() используется для обучения модели на текстовых данных.

4.8. Метод обученной модели most\_similar(\*word\*) приводит самые похожие по смыслу слова (Рис. 7). Как можно заметить - довольно успешно.

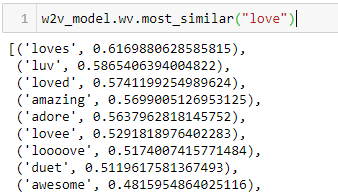


Рис. 7 - метод most\_similar()

4.9. Далее производится токенизация текста (Рис. 8): создается объект класса Tokenizer из модуля keras.preprocessing. С помощью него производится токенизация тестовых данных, строится словарь слов и происходит преобразование текста в числовые последовательности, выровненные до фиксированной длины.

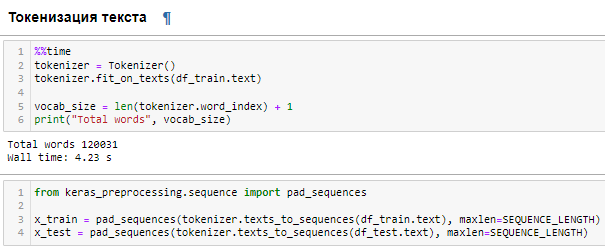


Рис. 8 - Токенизация

4.10. Преобразование целевой переменной происходит с помощью объекта класса LabelEncoder (Рис. 9). Поскольку у нас только два значения целевой переменной (POSITIVE, NEGATIVE) - они будут закодированы как 0 и 1, и представлять собой элементы массива.

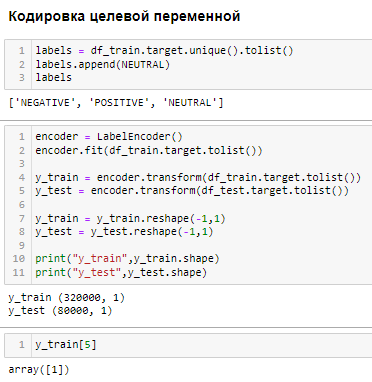


Рис. 9 - LabelEncoder

5. Подготовка данных для моделей: после токенизации и векторизации текста, данные должны быть приведены в удобный формат для моделей LSTM и CNN. Обычно это означает преобразование данных в тензоры, которые являются многомерными массивами чисел. Для модели LSTM данные могут быть представлены в формате последовательностей, где каждая последовательность представляет собой векторизованный текст определенной длины. Для модели CNN данные могут быть представлены в виде матрицы, где каждая строка соответствует вектору представления текста. В данной работе создается матрица векторных представлений слов, которая будет использоваться в слое “Embedding” модели (Рис. 10). Матрица и слой позволят модели учиться и использовать семантическую информацию о словах в задаче классификации текстов.

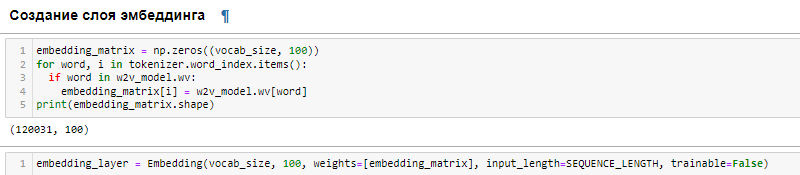


Рис. 10 - Создание слоя эмбеддинга.

Здесь создается нулевая матрица с параметрами размера словаря (уникальные слова) и размерностью векторного представления слов (был уменьшен ради большей производительности. Затем выполняется цикл по парам ‘word’ и ‘i’ из словаря ‘tokenizer.word.index.items() ’. Если слово найдено в модели ‘w2v\_model’, - соответствующая строка матрицы заполняется вектором для данного слова. После этого создается слой ‘embedding\_layer’ с помощью класса ‘Embedding’. Слой используется для преобразования целых числовых идентификаторов слов в их плотные векторные представления. Параметры: размер словаря, размерность векторных представлений слов, матрица векторных представлений слов, длина входных последовательностей и логическая переменная, которая отвечает за неизменость весов слоя во время обучения модели.

6. Обработка несбалансированных данных (при необходимости): В задаче анализа настроений в тексте может возникнуть проблема несбалансированности классов, то есть неравномерного распределения текстов по разным категориям настроений. Если классы не сбалансированы, модель может быть смещена в сторону более представленного класса, что может привести к плохой производительности на менее представленных классах. В таком случае можно применить методы балансировки данных, такие как оверсэмплинг (увеличение числа примеров в менее представленных классах) или андерсэмплинг (уменьшение числа примеров в более представленных классах), чтобы достичь более равномерного распределения данных. В нашем случае такой проблемы нет.

Глава 2: Построение моделей.

В этом разделе мы будем строить модели для анализа настроений в тексте. Мы рассмотрим две основные модели: LSTM (Long Short-Term Memory) и CNN (Convolutional Neural Network).

Часть 1.

Начнем с модели LSTM: модель LSTM является рекуррентной нейронной сетью, способной обрабатывать последовательности данных, такие как текст. Мы определим архитектуру модели LSTM, включая количество слоев, количество нейронов в каждом слое, функции активации и другие параметры. Затем мы обучим модель на обучающей выборке и оценим ее производительность на тестовой выборке. Создание модели показано на рисунке 11:

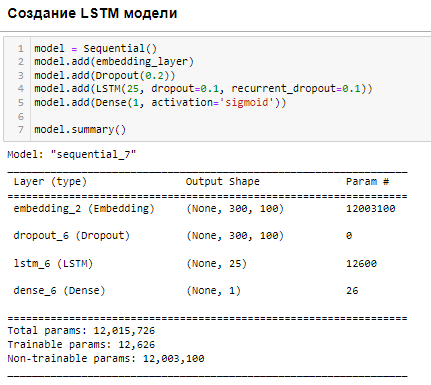


Рис. 11 - Создание LSTM модели

Создается последовательная модель нейронной сети с помощью класса ‘Sequential()’ из библиотеки Keras. Это линейная стековая модель, в которой слои добавляются последовательно друг за другом (Рис. 12).

В модель добавляется слой embedding\_layer, который был определен ранее и содержит предобученные векторные представления слов.

Потом добавляется слой Dropout с коэффициентом 0.2. Dropout - это метод регуляризации, который случайным образом обнуляет некоторые выходы слоя с заданной вероятностью во время обучения, что помогает снизить переобучение модели.

Следующий слой - LSTM с 25 скрытыми единицами. LSTM используется для моделирования последовательных данных, в данном случае текста. Опционально указывается dropout-коэффициент и рекуррентный dropout-коэффициент для регуляризации модели LSTM.

Последний полносвязный слой (Dense) с одним выходным нейроном и активацией 'sigmoid' (Рис. 13). Данный слой используется для получения бинарного вывода (0 или 1) в задачах бинарной классификации.

С помощью метода summary() выводится сводка модели, которая показывает структуру модели, количество параметров и размерность выхода каждого слоя.

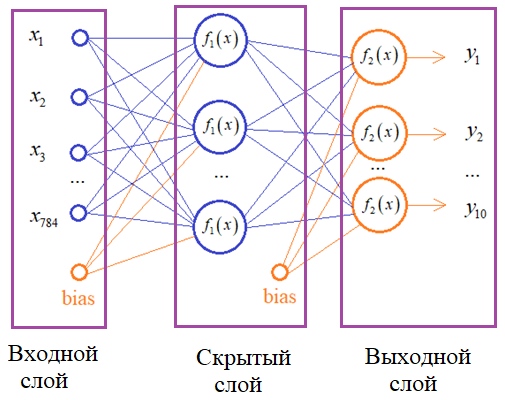


Рис. 12 - последовательная модель

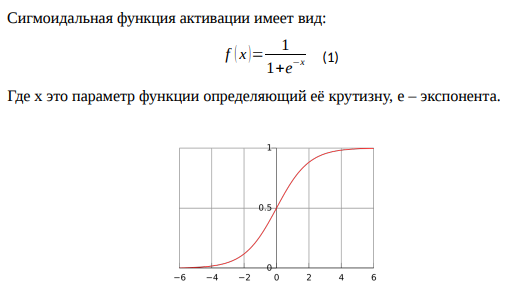


Рис. 13 - Сигмоидальная функция

После создания модели и определения ее архитектуры, следующий шаг - сконфигурировать модель для обучения, задав параметры оптимизации и метрики оценки производительности (Рис. 14).

Model.compile() принимает следующие параметры:

* loss='binary\_crossentropy' указывает функцию потерь, которая будет использоваться во время обучения модели. Здесь используется функция бинарной перекрестной энтропии, которая часто применяется в задачах бинарной классификации.
* optimizer="adam" определяет оптимизатор, который будет использоваться для обновления весов модели во время обучения. Здесь используется оптимизатор Adam, который является распространенным выбором для обучения нейронных сетей.
* metrics=['accuracy'] задает список метрик, которые будут использоваться для оценки производительности модели. В данном случае, используется метрика точности (accuracy), которая измеряет долю правильно классифицированных примеров.

После компиляции модели она будет готова к обучению.

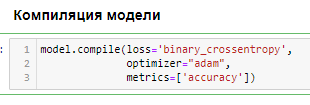


Рис. 14 - Компиляция модели

Теперь нужно определить список обратных вызовов (Рис. 15). Обратные вызовы — это объекты, которые могут быть переданы в метод обучения модели и выполняться в различные моменты обучения, чтобы предоставить дополнительную функциональность и контроль.

В данном коде используются два обратных вызова:

* ReduceLROnPlateau — это обратный вызов, который будет уменьшать скорость обучения (learning rate) в случае, если в течение нескольких эпох значение функции потерь (val\_loss) не улучшается. Он мониторит значение функции потерь на валидационном наборе данных и понижает скорость обучения, если наблюдается замедление улучшения производительности модели. Параметры monitor='val\_loss' указывает, что мониторить нужно значение функции потерь, patience=5 определяет, сколько эпох ждать, прежде чем уменьшать скорость обучения, и cooldown=0 указывает, сколько эпох ожидать после уменьшения скорости обучения, прежде чем возобновить нормальное обучение.
* EarlyStopping — это обратный вызов, который прекращает обучение модели, если значение метрики (val\_accuracy) перестает улучшаться на валидационном наборе данных. Он мониторит значение метрики точности на валидационном наборе данных и останавливает обучение, если наблюдается замедление улучшения производительности модели. Параметры monitor='val\_accuracy' указывает, что мониторить нужно значение метрики точности, min\_delta=1e-4 задает пороговое значение изменения метрики, при котором считается, что улучшение происходит, и patience=5 определяет, сколько эпох ждать, прежде чем прекратить обучение после отсутствия улучшений.

Таким образом, эти обратные вызовы позволяют контролировать процесс обучения модели, управлять скоростью обучения и остановить обучение, если наблюдается замедление улучшения производительности.



Рис. 15 – Callbacks

Теперь можно приступить к обучению модели нейронной сети с помощью метода ‘fit()’ (Рис. 16).

Аргументы метода fit():

* x\_train и y\_train - обучающие данные (входы и соответствующие целевые значения).
* batch\_size=1024 - количество примеров, используемых для одного обновления градиента. Обновление градиента выполняется после каждого мини-батча, содержащего 1024 примера (число выбрано с учетом низкой скорости обучения в процессе выполнения данной работы).
* epochs=EPOCHS - количество эпох обучения, где EPOCHS - предварительно определенное число эпох (в нашем случае - 8).
* validation\_split=0.1 - доля обучающих данных, которая будет использоваться в качестве валидационного набора данных. Здесь 0.1 означает, что 10% обучающих данных будут использоваться в качестве валидационного набора.
* verbose=1 - режим вывода информации о прогрессе обучения. Значение 1 означает, что информация будет выводиться на каждую эпоху обучения.
* callbacks=callbacks - список обратных вызовов, которые были определены ранее.

Метод fit() запускает процесс обучения модели на обучающих данных и выполняет несколько эпох обновления весов на основе заданной функции потерь, оптимизатора и метрик. Во время обучения информация о прогрессе, значения функции потерь и метрик выводятся в соответствии с установленными параметрами.

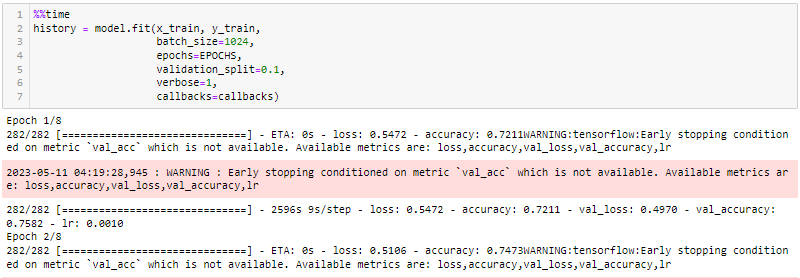
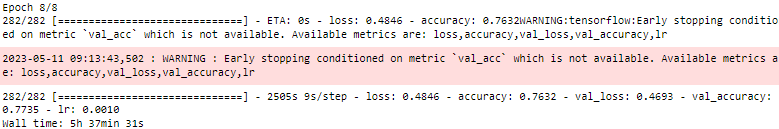


Рис. 16 - Обучение модели

В процессе и конце каждой эпохи можно наблюдать за изменениями метрик ‘loss’ и ‘accuracy’. Общее время обучения составило более 5 часов, что, на самом деле, неоправданно много. (Рис. 17).

Рис. 17

После обучения необходимо выполнить оценку производительности обученной модели на тестовых данных (Рис. 18).

Аргументы метода evaluate():

* x\_test и y\_test - тестовые данные (входы и соответствующие целевые значения).
* batch\_size=BATCH\_SIZE - размер мини-батча, используемого для оценки модели.

Метод evaluate() применяет обученную модель к тестовым данным и вычисляет значение функции потерь и выбранные метрики для оценки производительности модели. В данном коде результаты оценки сохраняются в переменной score, которая является списком, содержащим значения функции потерь и метрик.

Затем, с помощью функции print(), выводятся значения точности (accuracy) и функции потерь (loss) на тестовых данных. score[1] содержит значение точности, а score[0] содержит значение функции потерь, полученные в результате оценки модели на тестовых данных.

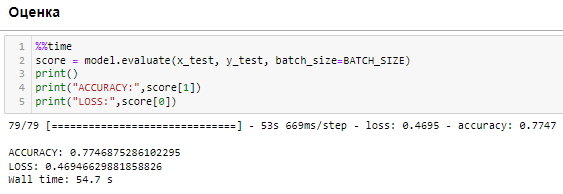


Рис. 18 - Оценка производительности.

Мы можем построить графики точности обучения и валидации и функции потерь обучения и валидации, которые отображают процесс обучения модели (Рис. 19).

График "Training and validation accuracy" показывает изменение точности модели на обучающем и валидационном наборах данных в зависимости от количества эпох. Линия "Training acc" представляет точность на обучающем наборе данных, а линия "Validation acc" - точность на валидационном наборе данных. Этот график позволяет оценить, как точность модели развивается в процессе обучения, а также определить наличие переобучения (если точность на обучающем наборе высока, а на валидационном наборе низкая).

График "Training and validation loss" отображает изменение функции потерь модели на обучающем и валидационном наборах данных в зависимости от количества эпох. Линия "Training loss" представляет функцию потерь на обучающем наборе данных, а линия "Validation loss" - функцию потерь на валидационном наборе данных. Этот график помогает оценить, как функция потерь изменяется в процессе обучения модели и определить, насколько хорошо модель обобщает данные.

Анализ этих графиков позволяет понять эффективность обучения модели и выявить возможные проблемы, такие как переобучение или недообучение. Оптимальным результатом является достижение высокой точности и низкой функции потерь как на обучающем, так и на валидационном наборе данных.

Как можно заметить, функции точности стабильно растут с выходом на “плато”, а функции потерь стабильно уменьшаются. Это говорит нам о том, что модель обучилась корректно.

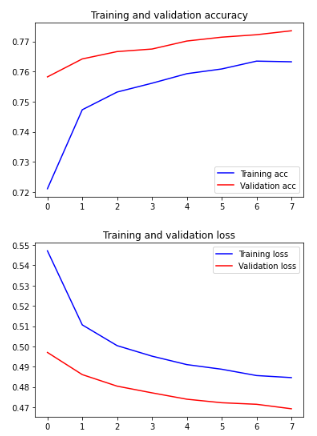


Рис. 19 - Графики

Чтобы проверить работу модели “по факту” созданы функции ‘decode\_sentiment’ и ‘predict’ (Рис. 20). Predict() получает параметр с текстом, который нужно оценить и возвращает метрику ‘score’. Decode\_sentiment() на основе ‘score’ выводит решение.

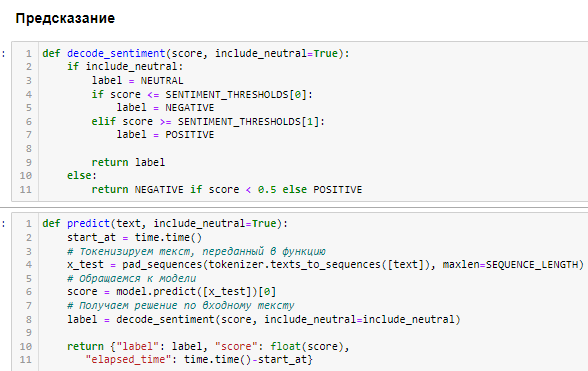


Рис. 20

Посмотрим, какие получим результаты по разным вариантам входного текста на рисунке 21. Можно сделать вывод, что модель работает корректно.

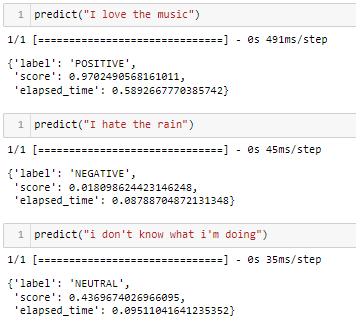


Рис. 21

Теперь перейдем ко второй модели.

Часть 2.

В этом разделе мы рассмотрим построение модели CNN (Convolutional Neural Network) для задачи анализа настроений в тексте. Модель CNN является эффективным инструментом для извлечения признаков из текстовых данных и обработки их в контексте анализа настроений.

Архитектура модели CNN включает несколько ключевых компонентов, включая сверточные слои, слои пулинга и полносвязные слои.

* Сверточные слои: Сверточные слои выполняют операцию свертки между входными данными и набором фильтров (ядер). Каждый фильтр выделяет определенные признаки из текста, например, определенные словосочетания или локальные паттерны. В результате операции свертки получается карта признаков, которая характеризует наличие или отсутствие определенных признаков в тексте.
* Слои пулинга: Слои пулинга выполняют уменьшение размерности карты признаков, удаляя ненужную детализацию и выделяя наиболее значимые признаки. Наиболее распространенный тип пулинга — это операция максимального пулинга, которая выбирает максимальное значение из определенной области карты признаков. Это позволяет снизить размерность данных и сохранить наиболее важные признаки.
* Полносвязные слои: после сверточных слоев и слоев пулинга следует несколько полносвязных слоев. Полносвязные слои обрабатывают высокоуровневые признаки, извлеченные из текста, и выполняют классификацию настроений. В конце модели обычно применяется слой softmax, который преобразует выходные значения в вероятности принадлежности к каждому классу настроения.

Создание модели немного отличается своими гиперпараметрами (Рис. 22).

* model\_cnn = Sequential() - создание последовательной модели.
* model\_cnn.add(embedding\_layer) - добавление слоя эмбеддинга, который был предварительно создан и инициализирован с помощью предобученных весов.
* model\_cnn.add(Dropout(0.5)) - добавление слоя Dropout с коэффициентом отсечения 0.5. Dropout применяется для случайного отключения некоторых нейронов во время обучения, чтобы предотвратить переобучение модели.
* model\_cnn.add(Conv1D(128, 5, activation='relu')) - добавление сверточного слоя с 128 фильтрами и размером окна 5. Слой использует функцию активации ReLU для введения нелинейности.
* model\_cnn.add(Conv1D(64, 5, activation='relu')) - добавление еще одного сверточного слоя с 64 фильтрами и размером окна 5.
* model\_cnn.add(GlobalMaxPooling1D()) - добавление слоя глобального пулинга, который извлекает наиболее значимые признаки из выходов сверточных слоев.
* model\_cnn.add(Dense(1, activation='sigmoid')) - добавление полносвязного слоя с одним нейроном и функцией активации sigmoid. Этот слой принимает извлеченные признаки и выполняет бинарную классификацию.
* model\_cnn.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy']) - компиляция модели, где указывается функция потерь (binary\_crossentropy) для задачи бинарной классификации, оптимизатор (adam) и метрика для оценки производительности модели (accuracy).



Рис. 22 - Создание CNN модели

Список callbacks, обучение модели и оценивание можно сказать не меняются (Рис. 23, 24).

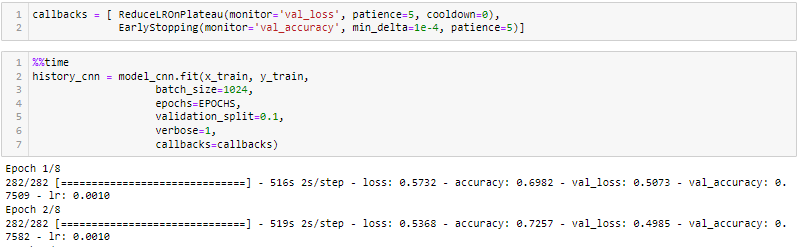


Рис. 23

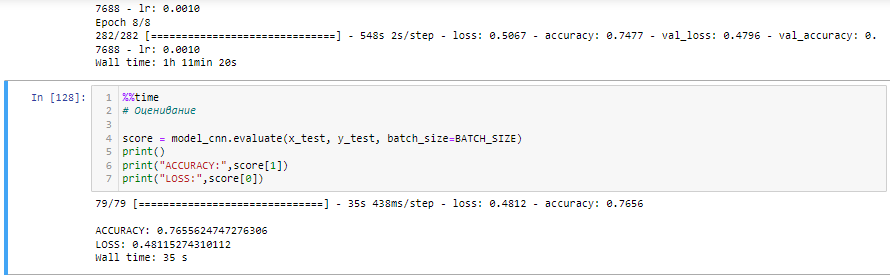


Рис. 24

В верхней части рисунка 24 можно заметить, что модель обучилась практически в 5 раз быстрее, при этом метрики оказались примерно такие же, как и у модели LSTM. Рассмотрим графики точности обучения и валидации и функции потерь обучения и валидации (Рис. 25). Видно, функции точности растут, а функции потерь уменьшаются, что говорит о том, что модель обучилась корректно.

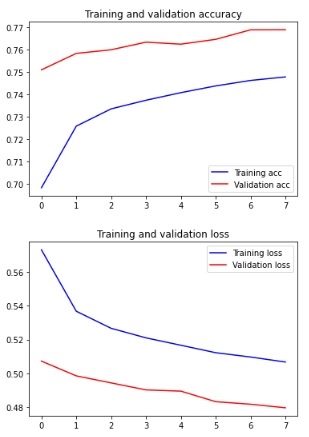


Рис. 25 - Графики

Посмотрим теперь как эта модель покажет себя на практике (Рис. 26). Модель без проблем определяет эмоциональную окраску.

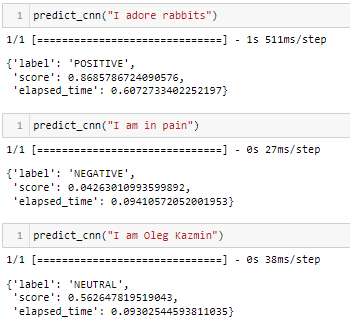


Рис. 26

Глава 3: Сравнение моделей.

После оценки производительности моделей LSTM и CNN, мы можем провести сравнение их результатов для определения, какая модель показывает лучшие результаты в задаче анализа настроений в тексте. Некоторые аспекты, которые можно учесть при сравнении моделей, включают:

* Производительность на метриках (Рис. 27, 28): Мы сравниваем производительность моделей на основе метрик, таких как точность, полнота, F-мера и другие. Модель, которая показывает более высокие значения метрик, считается более эффективной в анализе настроений. Метрики у обоих моделей примерно одинаковые, невозможно точно сказать, какая лучше, основываясь на метриках.

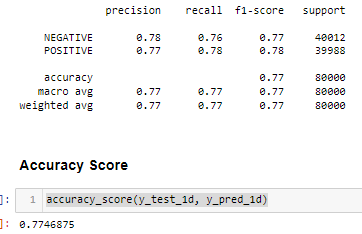


Рис. 27 – LSTM

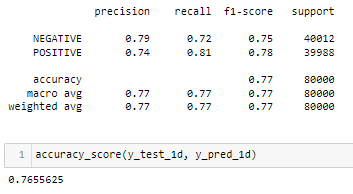


Рис. 28 - CNN

* Распределение ошибок (Рис. 29, 30): Изучение матрицы ошибок поможет понять, в каких случаях каждая модель делает ошибки и с какими классами настроений они связаны. Если одна модель показывает более равномерное распределение ошибок, это может указывать на ее лучшую способность различать разные классы настроений. Здесь тоже результат неоднозначный: LSTM немного лучше предсказывает негативные сообщения, а CNN - позитивные, но отличия не такие большие, чтобы сделать однозначный вывод.

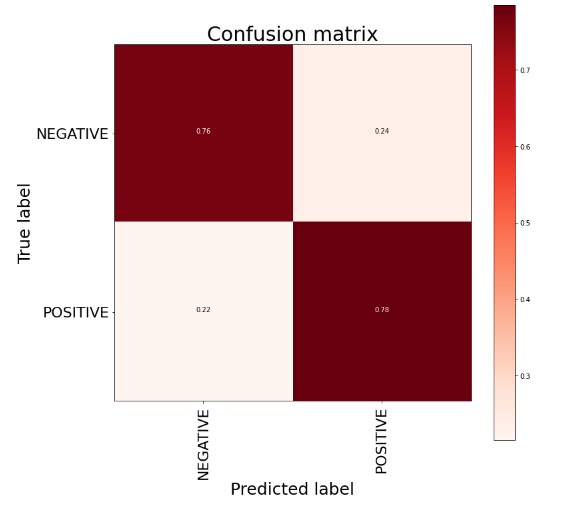


Рис. 29 – LSTM

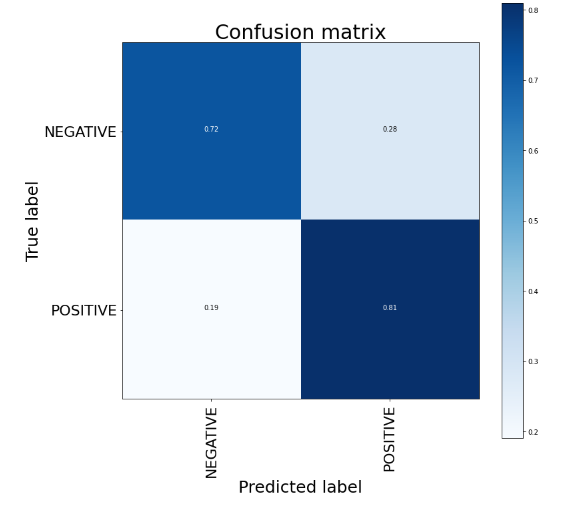


Рис. 30 - CNN

* Скорость обучения и предсказания: можно сравнивать время, необходимое для обучения каждой модели и время, требуемое для предсказания на новых данных. Если одна модель обучается или работает быстрее другой, это может быть важным фактором при выборе модели для конкретной задачи. Здесь однозначно CNN показывает лучшие результаты: в условиях ограниченного времени и ресурсов, модель CNN обучилась почти в 5 раз быстрее, показывая метрики не хуже, чем у LSTM.
* Интерпретируемость результатов: некоторые модели, такие как модель LSTM, могут обеспечить интерпретируемость результатов, позволяя анализировать важность слов или фраз в тексте для принятия решения о настроении. Если интерпретируемость является важным аспектом для решаемой задачи, это также может влиять на выбор модели.

Для перестраховки было принято решение сравнить наши модели нейронных сетей с моделью линейной регрессии (Рис. 31). Часто бывает так, что простые модели машинного обучения справляются лучше, чем сложные нейронные сети, но не в этот раз: на рисунке 31 наглядно показано, что метрики не дотягивают до уровня моделей LSTM и CNN.



Рис. 31 - Логистическая регрессия

Проведя тщательное сравнение моделей между собой и моделью машинного обучения, можно с уверенностью сказать, что в рамках конкретно этой работы лучше всего себя показала модель

ЗАКЛЮЧЕНИЕ.

В рамках данной курсовой работы был проведен анализ и сравнительное исследование различных моделей для задачи классификации текстовых данных. Мы рассмотрели модели на основе LSTM и CNN, а также реализовали более простую модель на основе логистической регрессии.

Предварительная обработка данных включала очистку текста, токенизацию и векторизацию. Был использован алгоритм Word2Vec для получения векторных представлений слов.

В процессе обучения моделей было обнаружено, что более сложные модели, такие как LSTM и CNN, обеспечивают лучшую точность классификации, но требуют больше времени для обучения. Более простая модель на основе логистической регрессии показала более низкую точность, но обучалась значительно быстрее.

При оценке и сравнении моделей мы использовали метрики precision, recall и F1-score. Результаты показали, что модели на основе LSTM и CNN достигли высоких значений этих метрик, в то время как модель на основе логистической регрессии показала нижние значения.

Таким образом, выбор модели зависит от баланса между точностью и временем обучения. Если требуется высокая точность и готовность ждать длительное время обучения, то модели на основе LSTM и CNN могут быть предпочтительными. В случае, когда время обучения является критическим фактором, более простая модель на основе логистической регрессии может быть разумным выбором.

В дальнейшем исследовании можно рассмотреть другие модели и методы для улучшения точности классификации, а также провести дополнительную оптимизацию процесса предобработки данных и обучения моделей.

Однако, необходимо отметить, что результаты данной работы могут зависеть от конкретного набора данных и контекста задачи классификации. Поэтому рекомендуется проводить дополнительные эксперименты и тестирования для подтверждения полученных результатов.

В целом, данная курсовая работа позволила ознакомиться с основными моделями и методами классификации текстовых данных, а также получить практический опыт работы с ними.

СПИСОК ЛИТЕРАТУРЫ (ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ) И ИНТЕРНЕТ-РЕСУРСОВ

1. Иванова, М. (2018). Методы анализа тональности текстов и их применение в задачах обработки естественного языка. Молодой ученый, (11), 363-366.
2. Карпов, А., & Панченко, А. (2019). Модели анализа тональности текстов на русском языке: обзор. Вестник Московского университета. Серия 15: Вычислительная лингвистика и интеллектуальные технологии, (2), 41-56.
3. Нефедова, А., Кузнецов, Д., & Брицын, В. (2017). Анализ тональности текстов на русском языке. Нейрокомпьютеры: разработка, применение, (11), 61-67.
4. Открытая библиотека машинного обучения Keras [Электронный ресурс]. – URL: https://keras.io/
5. Dataset Kaggle Twitter Tweets |Kaggle| [Электронный ресурс]. – URL: https://www.kaggle.com/kazanova/sentiment140
6. Рашка С. Python и машинное обучение / пер. с англ. А. В. Логунова. - М.: ДМК Пресс,2017. - 418 с., ISBN 978-5-97060-409-0. А.Мюллер, C.Гвидо Введение в машинное обучение. Руководство для специалистов по работе с данными: Пер. англ.- СПб: ООО “Альфа-книга”,2017.-480 c., ISBN 978-5-99089-108-1