Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Кафедра математического и компьютерного моделирования

РАБОТА ПРОВЕРЕНА
Рецензент,
ведущий программист
ООО «ЦЕНТР РАЗРАБОТКИ»
_____/ В.Д. Фоменко
«____»_____2022 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой,
д-р физ.-мат. наук, профессор
_____/ С.А. Загребина
«____»_____2022 г.

АВТОМАТИЧЕСКАЯ КАТЕГОРИЗАЦИЯ ТОВАРОВ ПО ИХ НАИМЕНОВАНИЯМ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ–01.03.04.2022.174.ВКР

Руководитель работы,
старший преподаватель
кафедры МиКМ,
_____/ А.К Богушов
«____»_____2022 г.

Автор работы,
студент группы ЕТ-414
_____/ В.Ю. Власов
«____»_____2022 г.

Нормоконтролер,
доцент кафедры МиКМ,
канд. пед. наук
_____/ Н.Н. Овчинникова
«____»_____2022 г.

Челябинск 2022

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Южно-Уральский государственный университет (национальный исследовательский университет)»
Институт естественных и точных наук
Кафедра математического и компьютерного моделирования
Направление подготовки 01.03.04 Прикладная математика

УТВЕРЖДАЮ
Заведующий кафедрой,
д-р физ.-мат. наук, профессор
_____/ С.А. Загребина
_____2022 г.

З А Д А Н И Е

на выпускную квалификационную работу студента
Власова Всеволода Юрьевича,
группа ЕТ-414

1. **Тема работы**
«Автоматическая категоризация товаров по их наименованиям» утверждена приказом по университету от « 25 » апреля 2022 г. № 697-13/12
2. **Срок сдачи студентом законченной работы** «___» _____2022 г.
3. **Исходные данные к работе**
   3.1. Fernández Delgado, M. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems / M. Fernández Delgado, E. Cernadas, S. Barro, D Amorim // The Journal of Machine Learning Research. – 2014.
   3.2. Vaswani, A. Attention is all you need / A. Vaswani, N. Shazeer, N. Parmar, J.
   3.3. Li, Q. A Survey on Text Classification: From Traditional to DeepLearning // Q. Li, H. Peng, J. Li, C. Xia, R. Yang, L. Sun, P. Yu, L. He // Proceedings of Workshop at ACM Transactions on Intelligent Systems and Technology. – 2022.
4. **Перечень вопросов, подлежащих разработке**
   4.1. Формализация задачи классификации текста, определение основных концепций машинного обучения.
   4.2. Обзор основных методов машинного обучения, включая глубокие нейронные сети.
   4.3. Изучение основных метрик качества бинарной и многоклассовой классификации.
   4.4. Построение и обучение модели нейронной сети для решения задачи классификации на основе датасета продуктовых товаров

5. Календарный план

Наименование этапов выпускной квалификационной работы	Срок выполнения этапов работы	Отметка о выполнении руководителя
1. Изучение литературы по тематике выпускной квалификационной работы	05.12.2021 – 31.01.2022	
2. Обзор методов исследования, применяемых по тематике ВКР	10.01.2022 – 28.02.2022	
3. Формулировка целей и задач выпускной квалификационной работы	01.03.2022 – 18.03.2022	
4. Решение основных задач ВКР	11.03.2022 – 28.05.2022	
5. Разработка структуры выпускной квалификационной работы	22.04.2022 – 30.04.2022	
6. Подготовка текста выпускной квалификационной работы	29.04.2022 – 28.05.2022	
7. Проверка работы руководителем, исправление замечаний	01.06.2022 – 15.06.2022	
8. Нормоконтроль и размещение работы в Электронном ЮУрГУ	01.06.2022 – 12.06.2022	
9. Рецензирование, представление зав. кафедрой	13.06.2022 – 18.06.2022	
10. Подготовка иллюстративного материала и доклада	17.06.2022 – 24.06.2022	

6. Дата выдачи задания «___» __декабря 2021 г.

Руководитель работы _____/ А.К. Богушов
(подпись)

Студент _____/ В.Ю. Власов
(подпись)

АННОТАЦИЯ

Власов В.Ю. Автоматическая категоризация товаров по их наименованиям – Челябинск: ЮУрГУ, ЕТ-414, 55 с., 28 ил., библиогр. список – 32 наим., 1 прил.

Выпускная квалификационная работа посвящена разработке модели для автоматической категоризации каталога товаров.

В работе был произведен анализ основных подходов к решению задачи. Описаны главные концепции и определения машинного обучения. Были рассмотрены основные типы классификаторов, формализовано понятие искусственного нейрона. Отдельный раздел посвящен оценке качества моделей при решении задач классификации. Изучены различные решения проблемы векторного представления текстовых данных.

Рассмотрены основные типы архитектур нейронных сетей, использующихся при решении задач обработки естественного языка. Дается анализ использования нейронных сетей для решения задач классификации, описан подход переноса знаний между моделями – Transfer Learning.

Решена практическая задача классификации на примере каталога именованных товаров. Подробно описываются стадии решения задачи, такие как: поиск и обработка датасета, выбор и обучение модели. Произведен анализ результатов полученной модели.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современном мире производится, продается и покупается огромное количество разнообразных товаров. Для удобства поиска каждый товар обычно относят к одной из заранее определенных категорий (классов) – именованных обобщений некоторых свойств группы товаров. Но в такой ситуации каждому товару необходимо вручную соотнести нужную категорию. В случае небольшого числа товаров это не составляет труда, но с ростом числа товаров возникает проблема автоматической категоризации каталога товаров.

Также на данную проблему можно посмотреть с другой стороны. Допустим, что уже имеется некоторая (возможно ручная) категоризация каталога, но необходимо перевести её в другую систему категорий. Данная проблема актуальна для государственных учреждений, в которых государственные закупки делаются в соответствии с общероссийским классификатором продукции по видам экономической деятельности – ОКПД2.

Еще одно возможное применение автоматической категоризации состоит в том, чтобы не непосредственно производить классификацию, но предложить пользователю наиболее вероятные категории товара. Такой подход упрощает категоризацию для пользователя и наиболее актуален при добавлении новых товаров в различные интернет-магазины.

Задача классификации текстов имеет два основных подхода решения. Первый подход состоит в написании правил, по которым определяется категория товара. Данный подход индивидуален для каждой задачи и требует ручного труда для составления правил, которые к тому же впоследствии нужно обслуживать, поэтому большого распространения rule-based методы не получили.

Второй подход состоит в использовании методов машинного обучения. Существует большое количество разнообразных методов обучения, но в последнее время наибольшее распространение получили методы, использующие в своей основе нейронные сети [22].

Среди нейросетевых методов явными фаворитами являются модели, построенные на архитектуре Transformer [16]. Для создания и обучения моделей используются разные языки программирования, но основным языком является Python. Язык Python используется вместе с его библиотеками.

Для анализа данных обычно применяется библиотека pandas, для работы с нейронными сетями – библиотеки tensorflow (keras) и PyTorch. Также существует проект под названием Hugging Face, в котором каждый желающий может выложить свою обученную модель в открытый доступ или использовать модели сообщества для решения своих задач.

1. ОБЗОР МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ

Задача классификации (категоризации) хорошо изучена и имеет множество методов решения. Существует два основных подхода к решению данной задачи:

1) Подход, основанный на правилах (rule-based systems). Данный подход рассмотрен в разделе 1.1.
2) Подход, основанный на использовании методов машинного обучения. Анализ методов машинного обучения осуществлен в разделе 1.2. Особое внимание было уделено нейронным сетям. Модель искусственного нейрона описана в разделе 1.3.

При использовании выбранного подхода получаем классификатор, для оценки качества которого существуют различные метрики, такие как: Accuracy, Precision, Recall и F score. Метрики качества подробно разобраны в разделе 1.4.

Ключевой особенностью задач обработки естественного языка, подмножеством которого является рассматриваемая задача классификации текста, состоит в представлении текста как вектора или набора векторов. Данная проблема является следствием того, что любой классификатор умеет работать только с векторизированными данными.

Существует множество решений данной проблемы. Традиционные решения опираются на использование статистических методов, в то время как современные решения используют в своей основе нейронные сети. Векторные представления получили название эмбеддингов (Word Embeddings). Различные методы векторизации текста рассмотрены в разделе 1.5.

1.1. Rule-based systems

Основная идея данного подхода заключается в составлении правил вида:

Листинг 1.1. – Пример составления правил

```
IF {CONDITION_1} THEN {CLASS_ID}
IF {CONDITION_n} THEN {CLASS_ID}
```

Проходя по правилам, система делает вывод о принадлежности текста к тому или иному классу. Обычно условиями правил является проверка наличия того или иного слова в тексте, например, в задаче определения тональности текста присутствие слова «люблю» может указывать на положительную тональность текста.

Основные преимущества данного подхода:

1) отличная интерпретируемость получаемой модели;
2) неплохая точность при достаточном количестве правил, хорошо описывающих датасет.

Основные недостатки данного подхода:

1) необходимость вручную записывать правила, что влечет за собой проблему человеческого фактора;
2) при большом количестве сильно связанных между собой правил теряется интерпретируемость модели, так же модель становится трудно обслуживать и дополнять.

Учитывая представленные недостатки, использование данного подхода нецелесообразно. Перейдем к следующему подходу, использующему методы машинного обучения.

1.2. Методы машинного обучения

Машинное обучение представляет собой подраздел искусственного интеллекта [**Ошибка! Источник ссылки не найден.**], изучающий методы построения алгоритмов, способных обучаться. Машинное обучение находится на стыке математической статистики, методом оптимизации и классических математических дисциплин, но имеет также и собственную специфику.

Задача классификации текста (Text Classification) в терминах машинного обучения ставится следующим образом:

1) $X = \{x_1, x_2, \dots, x_{|X|}\}$ – множество текстов (документов);
2) $Y = \{y_1, y_2, \dots, y_{|Y|}\}$ – множество меток классов (категорий);
3) $F\colon\ X \to Y$ – неизвестная целевая функция (зависимость).

При этом известно некоторое подмножество объектов $\{x_1, \dots, x_l\} \in X$, для которых известны метки классов $y_i = F(x_i)$, $i = 1, \dots, l$. Пары «объект-ответ» (x_i, y_i) называются примерами или прецендентами. Совокупность пар $X^l = (x_i, y_i)_{i=1}^l$ называется обучающей выборкой или датасетом (training dataset).

Необходимо найти a: $X \to Y$ – решающую функцию, алгоритм, модель, которая приближала бы целевую функцию F на всем множестве X. В случае классификации существуют следующие варианты множества классов Y:

1) $Y = \{-1, +1\}$ – бинарная классификация;
2) $Y = \{1, \dots, K\}$ – многоклассовая классификация;
3) $Y = \{0 \dots 1\}^K$ – классификация на K классов, которые могут пересекаться.

Глобально, методы машинного обучения можно разделить на две категории:

1) Параметрические методы. Предполагается, что алгоритм a имеет определенную функциональную форму и зависит от некоторого набора параметров θ, также называемых весовыми коэффициентами или весами, из пространства параметров Θ. Более формально: F: $X \times \Theta \to Y$. Семейство таких отображений может быть записано как: $F = \{f(x, \theta) \mid \theta \in \Theta\}$, где f – фиксированная функция. Например, в случае линейной зависимости от одной переменной x алгоритм будет иметь вид: $f(x, \theta) = \theta_0 + \theta_1 \cdot t$, где $\{\theta_0, \theta_1\} \in \Theta^2$ – некоторый набор весов.
2) Непараметрические методы. Данные методы не зависят от набора параметров и используют в своей основе свойства пространства объектов. Примеры: kNN [**Ошибка! Источник ссылки не найден.**], Decision trees [**Ошибка! Источник ссылки не найден.**], непараметрическая классификация, непараметрическая регрессия.

В дальнейшем наибольшее внимание будет оказано параметрическим методам, так как именно они получили наибольшее развитие и активно

применяются при решении задач классификации текста. [**Ошибка! Источник ссылки не найден.**, С. 3]

Перед непосредственным использованием, модель необходимо обучить на основе данных из тренировочной выборки. Обучение состоит в настройке весов согласно некоторому методу обучения. Под методом обучения будем подразумевать отображение μ: $(X \times \mathrm{Y})^l \to F$, которое произвольной конечной выборке $X^l = (x_i, y_i)_{i=1}^l$ ставит в соответствие некоторый алгоритм $f \in F$. Также можно сказать, что метод μ строит алгоритм f по выборке X^l.

В процессе обучения необходимо определить, насколько значения модели отличаются от правильных и как необходимо изменить параметры модели для получения лучших результатов. Специально для этих целей вводятся понятия функции потерь и функционала качества.

Функция потерь $L(a, x)$ показывает ошибку модели a на конкретном объекте выборки t. Широко распространенные функции потерь:

1) $L(a, x) = \big(a(x) - F(x)\big)^2$ – квадратичная функция потерь.

Функционал качества $Q(a, T)$ показывает ошибку алгоритма на всем наборе данных. Функционал качества рассчитывается как среднее значение от функции потерь для всех объектов выборки.

$$Q(a, X^l) = \frac{1}{l} \sum_{i=1}^{l} L(a, x_i) ,$$

где l – количество объектов в выборке.

Классический метод обучения, называемый минимизацией эмпирического риска, заключается в том, чтобы найти в заданном множестве алгоритмов F алгоритм f, доставляющий минимальное значение функционалу качества Q на заданной обучающей выборке X^l:

$$\mu(X^l) = arg \min_{f \in F} Q(f, X^l) .$$

Минимизация функционала качества может быть выполнена с помощью градиентного спуска. В начале параметры модели инициализируются небольшими случайными значениями из некоторого распределения.

Градиентный спуск на каждой итерации обучения i изменяет вектор параметров модели в направлении антиградиента функционала качества:

$$w^{[i+1]} = w^{[i]} - \mu \cdot \nabla Q\left(w^{[i]}\right),$$

где $w \in W^n$ – вектор весовых коэффициентов, $\nabla Q(w) = \frac{\partial Q(w)}{\partial w}$ – градиент функционала качества, μ – величина шага в направлении антиградиента, также называемая темпом или скоростью обучения (learning rate).

После разбора основных определений и концепций машинного обучения перейдем к анализу основных моделей, использующихся при решении задач классификации текста.

Существует большое количество различных методов машинного обучения. Например, в статье [**Ошибка! Источник ссылки не найден.**] изучают и тестируют 179 классификаторов из 17 семей на 121 датасете из различных областей. В книге [**Ошибка! Источник ссылки не найден.**] автор предлагает ранжировать классификаторы по следующим категориям:

1) Классификаторы, использующие геометрическую модель. Основные представители: Support Vector Machine (SVM), K-Nearest Neighbors (KNN).
2) Классификаторы, использующие вероятностный подход. Основной представитель – Байесовский классификатор.
3) Классификаторы, использующие логический подход. Основные представители: Decision tree, Random forest.

Также возможно ранжировать классификаторы по линейности:

1) Линейные классификаторы. Основные представители: SVM [**Ошибка! Источник ссылки не найден.**], Логистическая регрессия.
2) Нелинейные классификаторы. Основные представители: KNN, Decision tree, Deep neural network, SVM с нелинейными ядрами.

Выбор нужного классификатора представляет собой довольно трудную задачу, но в последние годы наибольшее распространение получили методы, основанные на нейронных сетях, в частности глубокие нейронные сети.

Данные методы являются наиболее точными в задаче классификации текста. [**Ошибка! Источник ссылки не найден.**, С. 26], поэтому будем использовать их в данной работе. Анализ нейронных сетей стоит начать с модели искусственного нейрона.

1.3. Модель искусственного нейрона

Искусственный нейрон является наименьшим строительным блоком любой нейронной сети. Искусственный нейрон представляет собой упрощенную математическую модель естественного нейрона. Модель искусственного нейрона представлена в рисунке 1.1.

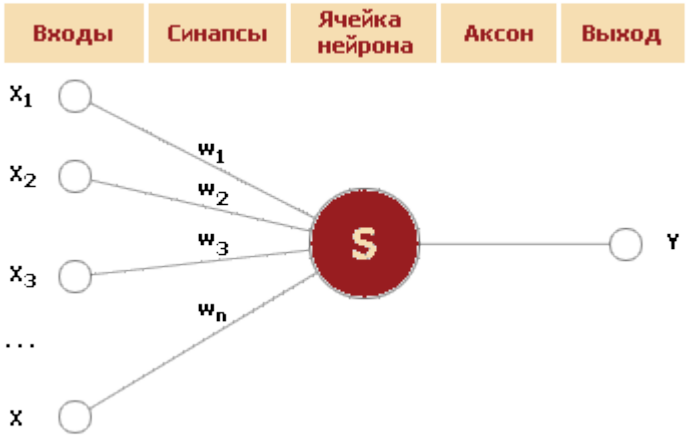

Рисунок 1.1 Модель искусственного нейрона

На вход к нейрону подается n входных сигналов – выходы других нейронов или входные данные. Каждый входной сигнал x_i связывается с ячейкой нейрона посредством синапса с весом w_i. Вес w_i показывает, насколько соответствующий вход влияет на состояние ячейки нейрона. Состояние нейрона S определяется как линейная комбинация входных сигналов и синаптических весов:

$$S = \sum_{i=1}^{n} x_i \, w_i + w_0 \, .$$

Выходной сигнал нейрона – аксон определяется как некоторая функция от его состояния S:

$$Y = f(S) = f\left(\sum_{i=1}^{n} x_i\, w_i + w_0\right).$$

Функцию f называют функцией активации, функцией срабатывания или передаточной функцией. Основные представители функции активации:

1) $f(x) = \frac{1}{1+e^{-x}}$ – сигмоида;
2) $f(x) = tanh(x)$ – гиперболический тангенс;
3) $f(x) = max(0, x)$ – Rectified Linear Unit.

Искусственная нейронная сеть определяется как система соединенных и взаимодействующих между собой искусственных нейронов. Подробно про архитектуры нейронных сетей будет рассказано в главе 2, перед этим же рассмотрим, какие существуют метрики качества моделей.

1.4. Метрики качества

Вначале рассмотрим метрики на примере бинарной классификации (принадлежит ли объект определенному классу или нет). В случае бинарной классификации возможны следующие четыре ситуации:

1) Классификатор верно определил принадлежность объекта классу – истинно положительный прогноз (True Positive);
2) Классификатор определил, что объект принадлежит классу, хотя на самом деле он не принадлежит – ложноположительный прогноз (False Positive) или ошибка первого рода;
3) Классификатор верно определил, что объект не принадлежит классу – истинно отрицательный прогноз (True Negative);
4) Классификатор определил, что объект не принадлежит классу, хотя на самом деле он принадлежит – ложноотрицательный прогноз (False Negative) или ошибка второго рода.

Удобно представить вышестоящие ситуации в виде матрицы несоответствий (confusion matrix), представленной в рисунке 1.2.

		True/Actual	
		Positive ()	Negative
Predicted	Positive ()	5 (TP)	1 (FP)
	Negative	2 (FN)	2 (TN)

Рисунок 1.2 Матрица несоответствий бинарной классификации

Определим основные метрики через матрицу несоответствий бинарной классификации:

1) $Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$ – показывает долю правильно классифицированных объектов по отношению ко всем классифицированным объектам. Недостаток $Accuracy$ заключается в том, что данная метрика не учитывает распределение классов в обучающей выборке, что может привести к ситуации, когда значение $Accuracy$ довольно большое (> 80%), но при этом для отдельных классов значение $Accuracy$ маленькое (< 30%), что дает ложное представление о «хорошем» качестве классификатора.

2) $Precision = \frac{TP}{TP+FP}$ – показывает долю правильно классифицированных объектов в рамках одного класса по отношению ко всем классифицированным объектам этого класса. $Precision$ демонстрирует способность классификатора отличать данный класс от других классов. Недостаток $Precision$ заключается в том, что мы никак не учитываем ложноотрицательные объекты.

3) $Recall = \frac{TP}{TP+FN}$ – показывает долю классифицированных объектов в рамках одного класса относительно всех объектов данного класса в наборе данных. $Recall$ демонстрирует способность алгоритма обнаруживать данный класс вообще. Недостаток $Recall$ заключается в том, что мы никак не учитываем ложноположительные объекты.

Метрики $Precision$ и $Recall$ используют разные подмножества ответов классификатора и их не всегда возможно одновременно максимизировать. Также хотелось бы иметь только одну метрику для сравнения различных классификаторов. Для решения данной проблемы вводится F-мера, которая учитывает $Precision$ и $Recall$.

F-мера представляет собой гармоническое среднее $Precision$ и $Recall$:

$$F = \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} = 2\frac{Precision \cdot Recall}{Precision + Recall}.$$

F-мера одинаково учитывает $Precision$ и $Recall$. Для случаев, когда одна метрика важнее другой, существует параметрическая F-мера:

$$F_\beta = \frac{(1+\beta^2)Precision \cdot Recall}{\beta^2 Precision + Recall}, \qquad \beta \in [0, \infty).$$

При $\beta = 0$: $F_\beta = Precision$, при $\beta = 1$: $F_\beta = F$, при $\beta = \infty$: $F_\beta = Recall$.

От бинарной классификации перейдем к многоклассовой классификации. Для многоклассовой классификации матрица несоответствий имеет следующий вид, представленный в рисунке 1.3.

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Рисунок 1.3 Матрица несоответствий многоклассовой классификации

Строки представляют собой ответы классификатора, столбцы – решения экспертной системы. Пересечение строки и столбца $A_{i,j}$ – число объектов класса j, которым алгоритм классификации присвоил класс i.

Основные метрики для класса c определяются аналогично бинарной классификации:

1) $\mathrm{P}recision_c = \frac{A_{c,c}}{\sum_{i=1}^{n} A_{c,i}}$ – отношение диагонального элемента к сумме элементов строки;

2) $Recall_{\mathrm{c}} = \frac{A_{c,c}}{\sum_{i=1}^{n} A_{i,c}}$ – отношение диагонального элемента к сумме элементов столбца.

$Accuracy$ рассчитывается как отношение суммы диагональных элементов к сумме всех элементов матрицы несоответствий:

$$Accuracy = \frac{\sum_{i=1}^{n} A_{i,i}}{\sum_{i=1}^{n}\sum_{j=1}^{n} A_{i,j}}.$$

$\mathrm{P}recision$ и $Recall$ для всех классов могут быть рассчитаны как средние арифметические значения метрик для каждого класса. Такой подход получил название макро-усреднение (Macro-Averaging):

$$\mathrm{P}recision_{macro} = \frac{\sum_{i=1}^{C} \mathrm{P}recision_i}{C},$$

$$Recall_{macro} = \frac{\sum_{i=1}^{C} Recall_i}{C},$$

где C – количество классов в наборе данных.

Для метрики F_{macro} существуют два распространенных определения:

1) Как среднее гармоническое $\mathrm{P}recision_{macro}$ и $Recall_{macro}$;

$$F_{macro}^{(1)} = 2\frac{\mathrm{P}recision_{macro} \cdot Recall_{macro}}{\mathrm{P}recision_{macro} + Recall_{macro}}.$$

2) Как среднее арифметическое F-меры для каждого класса.

$$F_{macro}^{(2)} = \frac{\sum_{i=1}^{C} F_i}{C},$$

$$F_i = 2\frac{\mathrm{P}recision_{\,i} \cdot Recall_i}{\mathrm{P}recision_i + Recall_i}.$$

Наиболее целесообразно использовать метрику $F_{macro}^{(2)}$, так как она более надежна к распределению ошибок первого и второго рода [31].

Макро-метрики одинаково учитывают каждый класс в выборке, в независимости от распределения классов в датасете. Для несбалансированных

выборок существуют взвешенные варианты $\mathrm{P}recision$, $Recall$ и F-меры, которые учитывают распределение классов в выборке:

$$\mathrm{P}recision_W = \frac{\sum_{i=1}^{C} W_i \cdot \mathrm{P}recision_i}{All},$$

$$Recall_W = \frac{\sum_{i=1}^{C} W_i \cdot Recall_i}{All},$$

$$F_W = \frac{\sum_{i=1}^{C} W_i \cdot F_i}{All},$$

где W_i – количество элементов класса i в выборке, $All = \sum_{i=1}^{C} W_i$ – общее количество элементов в выборке.

Также существует вариант микро-метрик, для которых $\mathrm{P}recision$ и $Recall$ рассчитываются одновременно для всех классов:

$$Precision_{micro} = \frac{\sum_{i=1}^{C} TP}{\sum_{i=1}^{C} TP + \sum_{i=1}^{C} FP},$$

$$Recall_{micro} = \frac{\sum_{i=1}^{C} TP}{\sum_{i=1}^{C} TP + \sum_{i=1}^{C} FN},$$

$$F_{micro} = 2\frac{\mathrm{P}recision_{micro} \cdot Recall_{micro}}{\mathrm{P}recision_{micro} + Recall_{micro}}.$$

Очевидно, что данные метрики равны между собой, а также что они эквивалентны метрике $Accuracy$:

$$Precision_{micro} = Recall_{micro} = F_{micro} = Accuracy\,.$$

1.5. Векторные представления слов и документов

Алгоритмы машинного обучения работают с векторами фиксированной размерности, поэтому существует проблема перевода текстовой информации в вектор или последовательность векторов.

Концептуально текстовую информацию можно рассматривать на трех уровнях:

1) Как последовательность отдельных символов. Данных подход применяется в рекуррентных (RNN) и свёрточных (CNN) сетях.

2) Как последовательность отдельных слов. Данный подход также применяется в рекуррентных сетях (RNN), но основное распространение получил в нейронных сетях, построенных на архитектуре Transformer.
3) Как вектор фиксированной длины. При использовании данного подхода возможно использовать полносвязные нейронные сети или другие классические алгоритмы машинного обучения, не умеющие работать с последовательной информацией.

Первый подход получил довольно ограниченное распространение, так как символы по отдельности несут в себе меньше информации по сравнению с представлением целого слова, поэтому ограничимся обзором векторных представлений второго и третьего подходов.

В подразделе 1.5.1 рассмотрены основные статистические представления слов и документов. В подразделе 1.5.2 произведен анализ алгоритма word2vec. После этого в подразделе 1.5.3 изучена модификация алгоритма word2vec для представления всего абзаца или документа под названием doc2vec. В конце раздела уделено внимание контекстно-зависимым векторным представлениям, как наиболее предпочтительным эмбеддингам.

1.5.1. Статистические методы представления слов и документов

Самым простым способом кодирования слов является one-hot кодировка. Пусть V – словарь всех возможных слов в тексте или документе. Тогда каждому слову $w = V_i$, где i – номер слова в словаре, можно поставить в соответствие вектор $x \in \{0,1\}^{|V|}$. При этом $x_j = \begin{cases} 0, i \neq j \\ 1, i = j \end{cases}$. В итоге получаем пространство $X^{|V|}$ с ортогональными векторами:

$$\forall w = V_i \; \exists x \in X^{|V|}: \; OHE(w) = x, \; x_j = \begin{cases} 0, i \neq j \\ 1, i = j \end{cases}.$$

В качестве примера рассмотрим кодировку слов «a», «abbreviations», «zoology» и «zoom», представленную в рисунке 1.4.

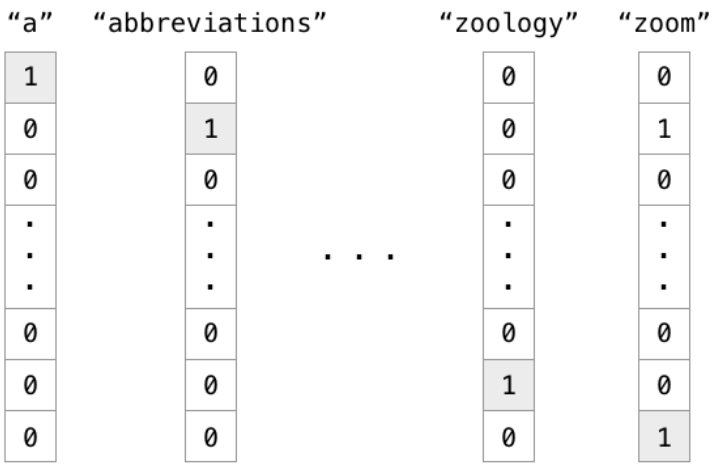

Рисунок 1.4. Пример работы one-hot encoding

К недостаткам one-hot кодировки можно отнести:

1) огромную размерность получаемых векторов при большом размере словаря.
2) каждое слово находится на одинаковом расстоянии друг от друга, то есть не учитывается лексическая и семантическая похожесть слов.

Логическим продолжением one-hot encoding в рамках всего текста или документа является представление Bag of words (Мешок слов). Мешок слов основан на предположении, что похожие тексты будут иметь схожие встречаемости слов в своем составе.

Bag of words ставит в соответствие тексту вектор фиксированной размерности, где i-ая координата соответствует количеству использований в тексте i-го слова. Пусть $\mathrm{X} = \{x_1, x_2, \dots, x_{|X|}\}$ – мультимножество всех слов в тексте или документе. Тогда работа метода Bag of words может быть определена формулой:

$$BoW(X) = \sum_{i=1}^{|X|} ONE(X_i)\,.$$

Представление BoW для некоторого набора документов изображено на рисунке 1.5.

	data	science	...	cambridge	spark
document 1	5	0	...	1	0
document 2	0	10	...	0	3
...	...	...	...	...	...
document 49	20	13	...	5	5
document 50	0	0	...	0	0

Рисунок 1.5 Пример работы представления Bag of words

Недостатки Bag of words вытекают из использования метода One hot encoding: огромная размерность получаемых векторов и их высокая разреженность. Но самый главный недостаток состоит в том, что мы никак не учитываем порядок слов, то есть игнорируем их семантические связи.

Также можно вместо подсчета количества слов в тексте или документе использовать частоту встречаемости каждого слова, для этого разделим вектор $BoW(X)$ на количество слов в тексте – $|X|$. Модифицированный метод получил название tf – term frequency:

$$tf(X) = \frac{1}{|X|} BoW(X) = \frac{1}{|X|} \sum_{i=1}^{|X|} ONE(X_i) .$$

Для отдельного слова t частота может быть выражена формулой:

$$tf(t, d) = \frac{n_t}{\sum_k n_k},$$

где n_t – число вхождений слова t в документ d.

При работе с большим количеством документов наиболее часто будут встречаться широкоупотребительные слова, такие как предлоги или местоимения. Но, так как они часто встречаются во всех документах, мы не получаем от них никакой новой информации о характере конкретного документа. Выдвигается гипотеза, что помимо частоты встречаемости слова в конкретном тексте, важным также является то, насколько редко слово появляется в документе относительно остальных документов.

Для каждого слова t в документе d из множества документов D рассчитывается статистическая мера $tf\text{-}idf$:

$$tf\text{-}idf = tf(t,d) \cdot idf(t,D),$$

где $tf(t,d)$ – частота слова t в документе d, $idf(t,D)$ – inverse document frequency (обратная частота документа) определяется по формуле:

$$idf(t,D) = log\left(\frac{|D|}{|\{d_i \in D \mid t \in d_i\}|}\right),$$

где $|D|$ – количество документов, $|\{d_i \in D | t \in d_i\}|$ – количество документов из D, в которых присутствует слово t.

Высокие значения $tf\text{-}idf$ получают те слова, которые часто появляются в конкретном документе, но при этом редко встречаются среди остальных документов.

Для примера вычислим $tf\text{-}idf$ для двух предложений:

1) The car is driven on the road (предложение A);

2) The truck is driven on the highway (предложение B).

Составим словарь $V = \{\text{The, car, truck, is, driven, on, the, road, highway}\}$. Далее найдем значения tf, idf и $tf\text{-}idf$ для каждого слова в предложениях. Удобно представить результаты вычислений в виде рисунка 1.6.

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

Рисунок 1.6 Пример работы представления tf-idf

Хорошо видно, что значимыми для предложения являются те слова, которые не встречаются в других предложениях. Недостатки представления $tf\text{-}idf$ аналогичны недостаткам представления Bag of words.

1.5.2. Word2Vec

Рассмотренные методы имеют один серьезный недостаток: они никак не учитывают контекст, в котором находится слово. Выдвигается гипотеза, согласно которой слова имеют схожие значения, если они находятся в схожем контексте.

Например, рассмотрим предложение «The fluffy dog barked as it chased a cat» и слово «dog» в нем. Назовем «Окном размера C» – C слов, стоящих справа и слева от интересующего нас слова. «Окно размера C» будет являться контекстом для нашего слова. Окно размера два для слова «dog» будет иметь следующий вид, представленный на рисунке 1.7.

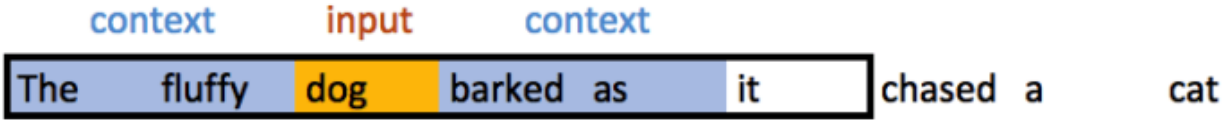

Рисунок 1.7 Пример окна размера 2 для слова «dog»

На основе данного подхода обучим модель, в которую будем подавать на вход C слов слева и справа, а модель будет предугадывать слово посередине. Данный подход получил название Continuous Bag of Words – CBoW [12]:

$$P(w_i|w_{i-\mathrm{C}}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+\mathrm{C}}) .$$

Существует также альтернативный подход, в котором по центральному слову w_i определяют сам контекст $w_{i-N}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+N}$. Данный подход получил название Skip-gram [12]:

$$P(w_{i-\mathrm{C}}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+\mathrm{C}}|w_i) .$$

В модели CBoW на вход подается C «горячих» векторов размера V, где C – размер контекста, V – размер словаря. В модели присутствует один скрытый слой размера N. На выходе модели получаем один вектор размера V – логиты для каждого слова из словаря. То есть фактически решается задача

многоклассовой классификации. Схематически модели CBoW и Skip-gram изображены на рисунке 1.8.

Задача обучения состоит в нахождении весовых коэффициентов двух матриц $W \in R^{V \times N}$ и $W' \in R^{N \times V}$. Обучение происходит с помощью максимизации функции правдоподобия – минимизации эмпирического риска.

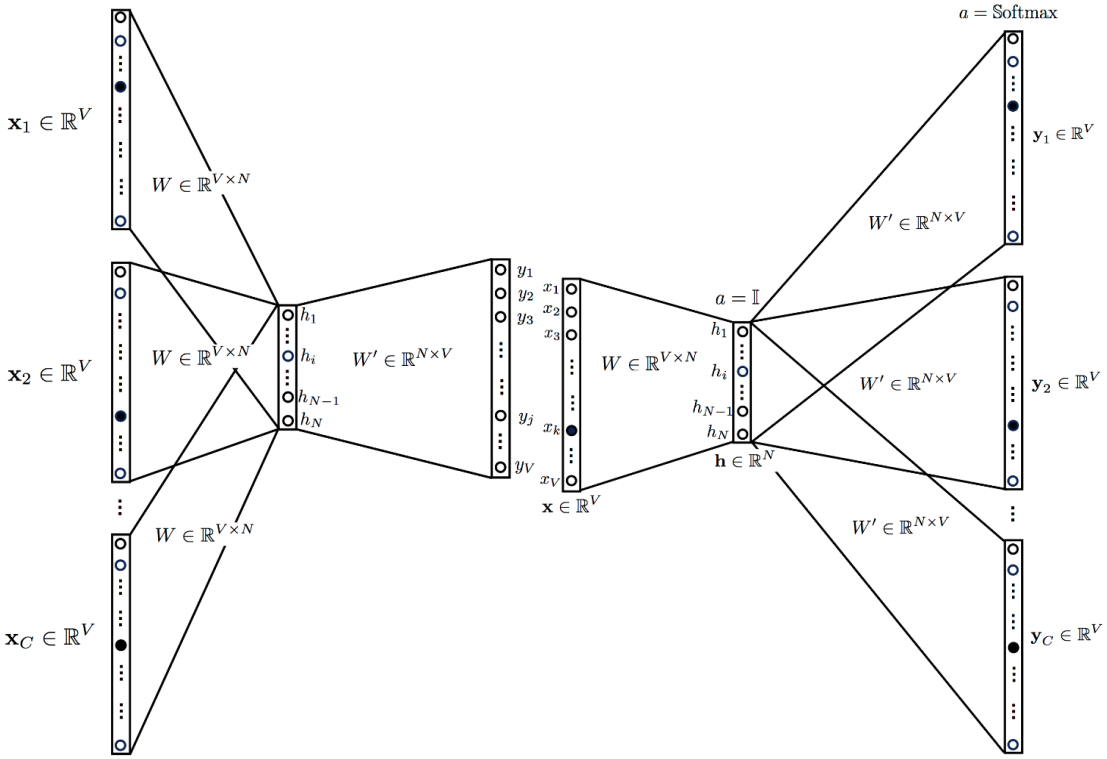

Рисунок 1.8 Схема сети для моделей CBoW и Skip-gram

Векторное представление для i-го слова в словаре является i-ой строкой в матрице $W \in R^{V \times N}$. Обычно гиперпараметр N, отвечающий за размерность полученных векторов, принимает значение в районе 200-300.

Полученные представления уже можно сравнивать между собой и находить ближайших соседей, используя некоторую метрику, например Евклидовое расстояние. Но обычно для сравнения представлений используют коэффициент Отиаи (косинусное расстояние):

$$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}.$$

где θ – угол между векторами.

Чем ближе значение коэффициента к 1, тем более похожи векторы друг на друга. Значение 0 соответствует ортогональным векторам.

В word2vec существуют линейные зависимости между отдельными словами. Самым известным примером этого является следующее выражение:

$$king - man + women \approx queen\,.$$

То есть при вычитании из представления слова «king» вектор «man» и дальнейшим сложении с представлением слова «women» мы получаем вектор, наиболее близкий к слову «queen». В графическом виде данную связь можно представить следующим образом 1.9.

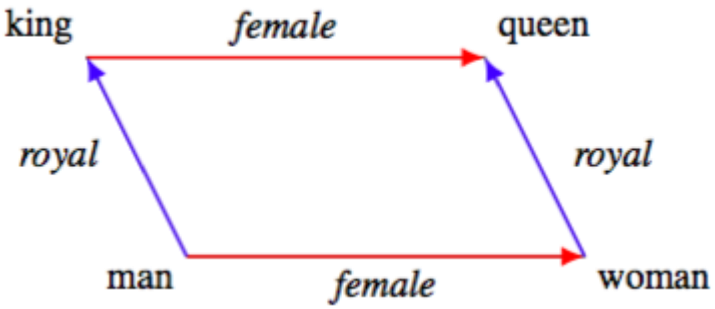

Рисунок 1.9 Пример линейных отношений в word2vec

Но нужно понимать, что в данном признаковом пространстве соседями являются те слова, которые употребляются в похожих контекстах. Так, например, ближайшим соседом к слову «синоним» будет являться слово «антоним» [28, С. 12].

1.5.3. Doc2Vec

Для векторного представления всего документа существует модификация метода word2vec под названием doc2vec [**Ошибка! Источник ссылки не найден.**]. В doc2vec также представлены два варианта обучения представлений:

1) Distributed Memory of Paragraph Vectors (PV-DM);

2) Distributed Bag of Words Paragraph Vector (PV-DBOW).

В каждом из вариантов используется дополнительная матрица параграфов D. Строка в матрице D представляет собой вектор параграфа для

определенного документа или абзаца. При обучении модели вектор параграфа является общим для всех контекстов из одного документа или абзаца.

PV-DM представляет собой модифицированный вариант CBoW. Основное отличие состоит в добавлении к входным данным вектора параграфа.

PV-DBOW представляет собой модифицированный вариант Skip-gram. Основное отличие состоит в том, что вместо входного вектора для предсказания контекста используется вектор параграфа.

Схематически данные модели представлены на рисунке 1.10.

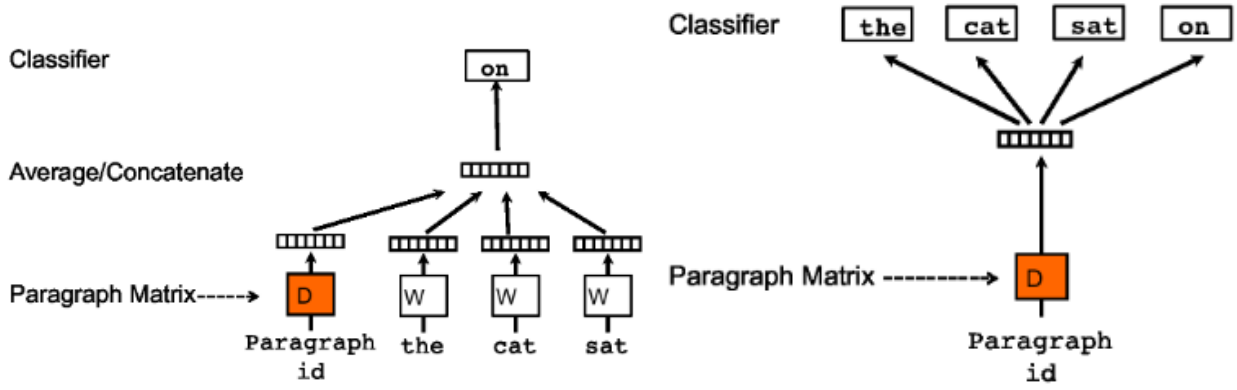

Рисунок 1.10 Упрощенная схема моделей PV-DM и PV-DBOW

Достоинства doc2vec наследуются от достоинств word2vec, но в ряде ситуаций doc2vec показывает себя не лучшим образом.

1.5.4. Контекстно-зависимые векторные представления

Алгоритм word2vec уже можно использовать для векторного представления текста, но у него есть один существенный недостаток. Word2vec ставит в соответствие каждому слову единственный вектор. Это означает, что у каждого слова есть одно смысловое значение. Но некоторые слова в зависимости от контекста могут иметь несколько значений – омонимы.

Пример предложений с омонимом:

1) Жизнь бьет ключом;

2) Отрыть дверь ключом.

Очевидно, что слово «ключом» имеет разное значение в зависимости от предложения.

Данную проблему решают контекстно-зависимые векторные представления. Контекстно-зависимые векторные представления могут быть получены из внутренних состояний в языковых моделях, таких как ELMo или BERT. [26]

Учитывая вышеперечисленное, использование контекстно-зависимых векторных представлений является наиболее предпочтительным при решении задач обработки естественного языка.

1.6. Выводы по первой главе

В первой главе были разобраны основные подходы к решению задачи классификации текста. Было показано, что использование rule-based systems неэффективно и что предпочтительным подходом в решении задачи классификации текста являются методы машинного обучения.

Задача классификации текста была формализована. Определены основные концепции машинного обучения: модель, алгоритм обучения, функция потерь, функционал качества. Рассмотрены основные представители методов машинного обучения для решения задачи классификации текста. Было отмечено, что нейронные сети показывают наилучшие результаты в задаче классификации текста. Для анализа архитектур нейронных сетей, проведенного в главе 2, была рассмотрена модель искусственного нейрона.

Произведен анализ основных метрик качества для бинарной и многоклассовой классификации. Были подробно разобраны метрики $Precision$, $Recall$, $Accuracy$ и F мера в трех вариантах: $micro$, $macro$ и $weighted$. Было показано, что использование макро-варианта F меры наиболее оптимально для оценки общей производительности классификатора в случае сбалансированных выборок, в случае несбалансированных – взвешенный вариант F меры.

Была рассмотрена проблема векторизации текстовых данных. Произведен обзор основных векторных представлений. Среди векторных представлений подробно разобраны алгоритмы word2vec, doc2vec и

контекстно-зависимые векторные представления. Показано, почему использование последних наиболее предпочтительно.

2. ОСНОВНЫЕ АРХИТЕКТУРЫ НЕЙРОННЫХ СЕТЕЙ

Существует большое количество типов нейронных сетей, но для задач классификации текста применяется ограниченное подмножество архитектур. Рассмотрим основные типы архитектур нейронных сетей, использующихся при решении данной задачи.

Исторически первой архитектурой для решения задач классификации текста являются многослойные нейронные сети. Многослойные сети рассмотрены в разделе 2.1. Рекуррентные нейронные сети появились позже. Среди рекуррентных сетей наибольшее распространение получили LSTM сети. В разделе 2.2 произведен анализ рекуррентных сетей и их модификаций.

Но наиболее современной архитектурой нейронных сетей является архитектура Transformer, специально созданная для решения задач обработки естественного языка (NLP). Архитектуре Transformer посвящен раздел 2.3. Яркими экземплярами данной архитектуры являются модели семейства BERT. Модели BERT обучены специальным образом и показывают лучшие результаты в задачах NLP. Подробно про модели семейства BERT рассказано в разделе 2.4.

В конце главы, в разделе 2.5, показано применение нейронных сетей для решения задач классификации.

2.1. Многослойные нейронные сети

Начать рассмотрение многослойных сетей стоит с определения сети прямого распространения. Под сетью прямого распространения подразумевается сеть, в которой сигнал распространяется строго от входного слоя к выходному. В обратном направлении сигнал не распространяется.

Многослойные нейронные сети представляют собой сети прямого распространения, построенные на основе одного или нескольких слоев нейронов. Под слоем подразумевается группа нейронов, не имеющая внутренних связей, но при этом соединенных с нейронами других слоев – предыдущим и следующим.

Архитектуру многослойной сети можно представить как совокупность трех группы слоев:

1) Входной слой. Нейроны данного слоя принимают элементы вектора признаков и распределяют их по нейронам следующего слоя. Обработка данных во входном слое не производится.
2) Скрытые слои. Между входным и выходным слоем может располагаться один или несколько скрытых слоев. Свое название они получили потому, что их входы и выходы неизвестны для внешних по отношению к нейронной сети программам и пользователю.
3) Выходной слой. На выходах нейронов данного слоя формируется результат работы нейронной сети.

Однослойную нейронную сеть можно рассматривать как многослойную сеть без скрытых слоев нейронов. Полносвязными нейронными сетями будем называть сети, в которых все нейроны i-ого слоя соединены со всеми нейронами на $i-1$ слое. Для примера рассмотрим полносвязную сеть с двумя скрытыми слоями, изображенную в рисунке 2.1.

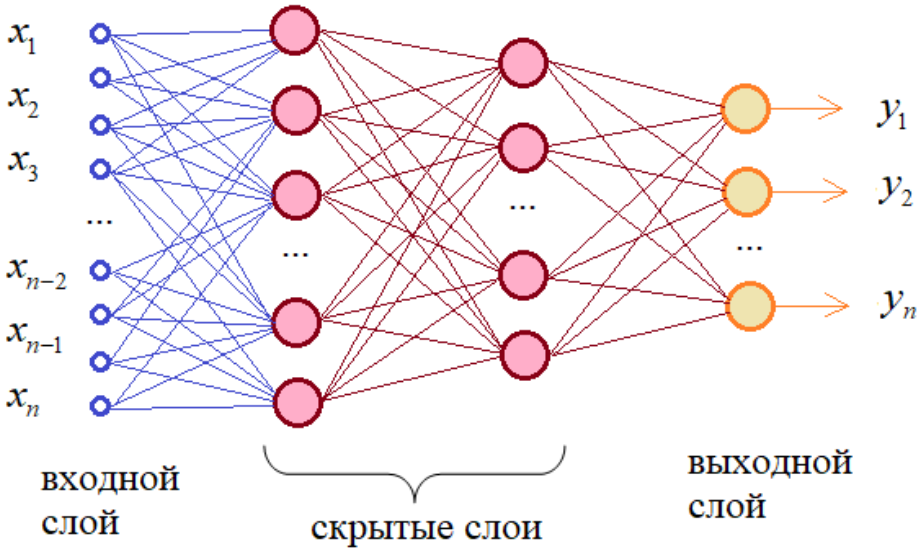

Рисунок 2.1 Пример полносвязной трёхслойной нейронной сети

В частности, полносвязная нейронная сеть с одним скрытым случаем именуется «Перцептроном» Розенблатта [**Ошибка! Источник ссылки не найден.**], а полносвязная сеть с числом скрытых слоев больше двух –

«Многослойным перцептроном» (MLP) Румельхарта. [**Ошибка! Источник ссылки не найден.**]

К недостаткам многослойных полносвязных сетей можно отнести следующее:

1) Большое количество весовых коэффициентов по сравнению с остальными типами нейронных сетей;
2) Невозможность решать определенный круг задач, связанных с последовательной обработкой данных, например подсчет единиц в последовательности.

Для решения проблемы последовательной обработки данных, какими и являются текстовые данные, был изобретен следующий тип сетей, имеющих внутреннюю память – рекуррентные сети.

2.2. Рекуррентные нейронные сети

Рекуррентные нейронные сети – вид нейронных сетей, в которых связи между элементами образуют направленную последовательность. В традиционных нейронных сетях каждое новое выходное значение независимо от предыдущих вычислений. Такой подход не подходит для решения определенного круга задач, например для предсказывания следующего слова в предложении на основе предыдущих слов.

Основным элементом рекуррентной нейронной сети является рекуррентная ячейка. Рекуррентная ячейка имеет скрытое внутреннее состояние h_t. Состояние обновляется каждый раз, когда в RNN отправляются новые входные данные x_t. На основе обновленного внутреннего состояния модель вычисляет выходное значение y_t. Схематически архитектуру RNN удобно представить в следующем виде, показанном на рисунке 2.2.

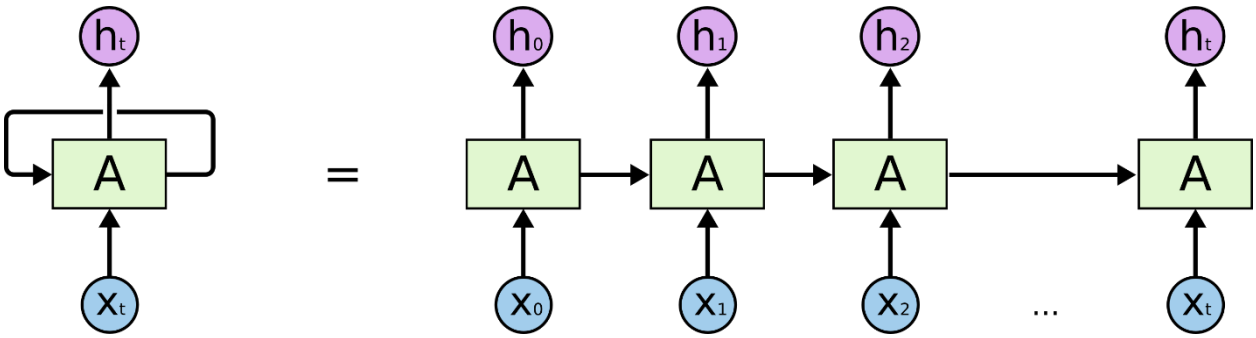

Рисунок 2.2 Схема архитектуры RNN и её развернутое представление

Существует несколько вариантов создания обратных связей между элементами. Одним из вариантов являются сети Элмана [**Ошибка! Источник ссылки не найден.**]. В сетях Элмана рекуррентное соотношение выражается следующими формулами:

$$h_t = f_h(Wh_{t-1} + Ux_t + b_h)\,,$$

$$y_t = f_y\left(Vh_t + b_y\right),$$

$$h_0 = 0\,,$$

где f_h, f_y – функции активации, W, U, V – матрицы весовых коэффициентов, b_h, b_y – векторы параметров, t – номер элемента последовательности.

В зависимости от того, столько входных элементов последовательности подается на вход к рекуррентной сети и то, сколько элементов выходной последовательности требуется получить, различают 4 вида RNN:

1) «Один ко одному». На вход подается последовательность из одного элемента, на выходе также получаем единичную последовательность.
2) «Один ко многим». На вход поступает единичная последовательность, на выходе получаем последовательность значений интересующей длины.
3) «Многие к одному». На вход подается последовательность переменной длины, на выходе получаем единственное значение. Данный тип рекуррентных сетей используется для классификации последовательной информации, в частности, текстовых данных.
4) «Многие ко многим». На вход подается последовательность переменной длины, на выходе также получаем последовательность

переменной длины. Данный тип сетей используется для перевода текста с одного языка на другой.

Графически данные типы RNN можно представить в следующем виде, изображенном на рисунке 2.3.

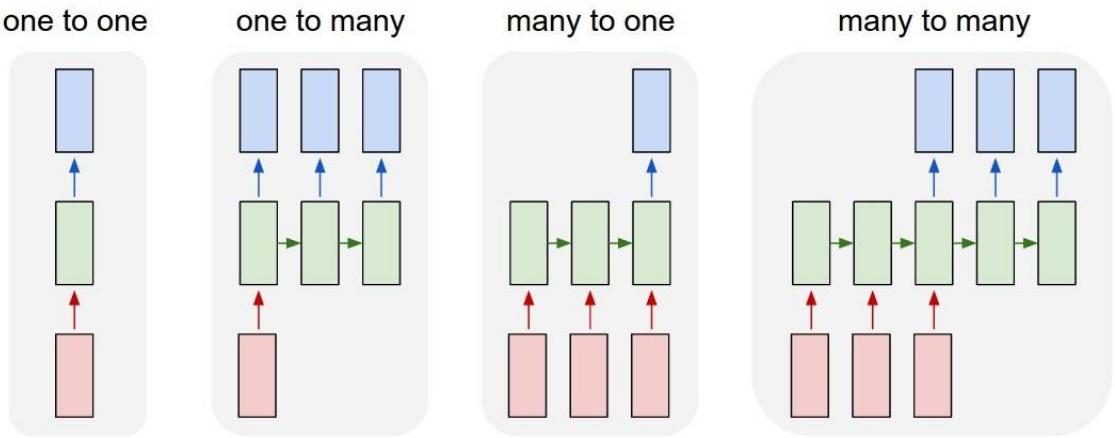

Рисунок 2.3 Различные типы рекуррентных сетей

Рекуррентные нейронные сети хорошо подходят для работы с короткими последовательностями, но при работе с длинными последовательностями сеть может «забыть» начало последовательности, когда дойдет до его конца. Данная проблема связана с затухающими градиентами во время обучения сети. [**Ошибка! Источник ссылки не найден.**]

Для решения этой проблемы существует несколько модификаций рекуррентных сетей, одна из которых – двунаправленные рекуррентные нейронные сети (Bidirectional RNN – BiRNN).

BiRNN представляет собой две однонаправленные рекуррентные нейронные сети, одна из которых обрабатывает входную последовательность в прямом порядке, а другая – в обратном. Таким образом, для каждого элемента входной последовательности вычисляются два вектора скрытых состояний, на основе которых рассчитывается выход сети. Схема архитектуры BiRNN проиллюстрирована в рисунке 2.4.

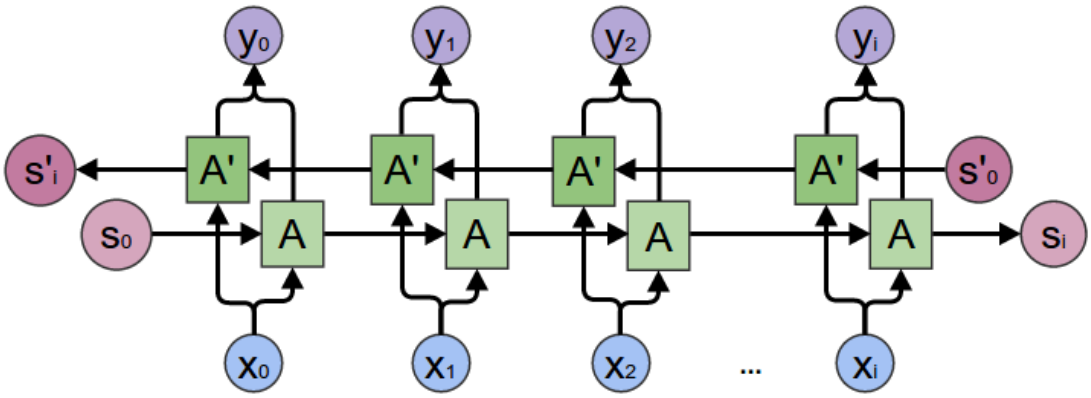

Рисунок 2.4 Двунаправленная рекуррентная нейронная сеть

BiRNN частично решает проблему забывания информации, но не полностью. Другой модификацией рекуррентных сетей являются рекуррентные сети с долгой краткосрочной памятью – LSTM. (Long Short-Term Memory) [**Ошибка! Источник ссылки не найден.**]. Основное достоинство модификации – способность запоминать данные как на короткие, так и на длинные периоды времени.

Ключевой особенностью LSTM является использование различных внутренних механизмов – «фильтров», для регулирования потока информации в LSTM модуле. Состояние LSTM ячейки находится в векторе c_i. На каждом временном шаге с помощью фильтров определяется, какая часть новой информации – i_t будет добавлена в c_t и какая часть старой информации – f_t будет из него удалена. Выходное значение находится в векторе h_t и определяется с помощью вектора состояния c_t, новых входных данных x_t и предыдущего выходного вектора h_{t-1}.

Архитектура ячейки LSTM представлена на рисунке 2.5.

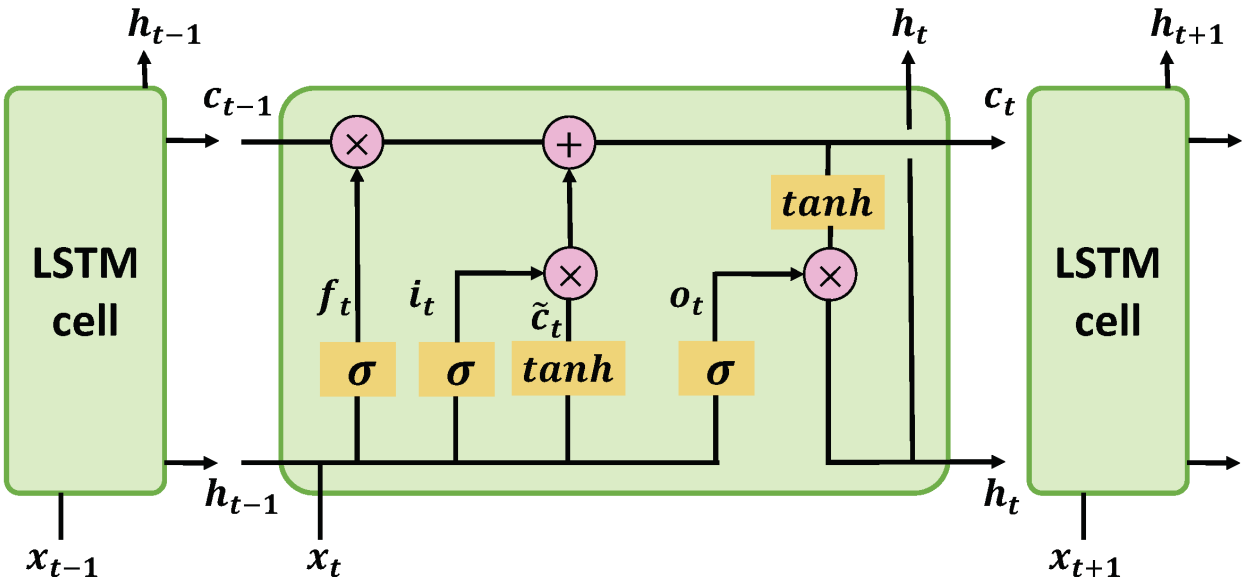

Рисунок 2.5 Схема архитектуры ячейки LSTM

Формальное определение на каждом временном шаге t:

$$f_t = \sigma_g\left(W_f x_t + U_f h_{t-1} + b_f\right),$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i),$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o),$$

$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c),$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t,$$

$$h_t = o_t \circ \sigma_h(c_t),$$

$$h_0 = 0, c_0 = 0.$$

Знак + обозначает сложение векторов, знак × в рисунке и знак ∘ в формулах обозначает произведение Адамара [32].

LSTM сети также могут быть двунаправленными, но наиболее интересной модификацией являются сложенные LSTM сети. (Stacked LSTM). В данной модификации используется не одна ячейка LSTM, а несколько. При этом входом для каждой i-ой ячейки, за исключением первой, является выходное значение $i-1$-ой ячейки:

$$x_m^{[i]} = h_m^{[i-1]}, \qquad i > 2.$$

Stacked LSTM показывают более лучшие результаты, чем обычные LSTM, но при этом их сложнее обучить [**Ошибка! Источник ссылки не**

найден.]. Графически архитектуру Stacked LSTM можно представить следующим образом 2.6.

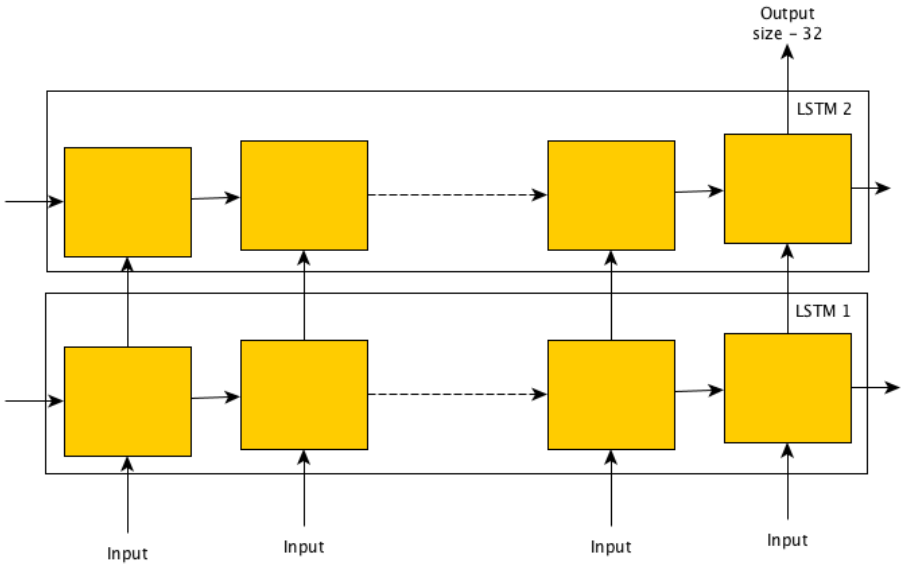

Рисунок 2.6 Схема архитектуры Stacked LSTM

В целом можно сказать, что рекуррентные нейронные сети показывают хорошие результаты в задачах NLP, но использование обратных связей влечет за собой проблемы при обучении, такие как взрывающиеся или исчезающие градиенты. От рекуррентных сетей перейдем к сетям, построенным на архитектуре Transformer.

2.3. Трансформеры

Трансформеры – современная архитектура нейронных сетей, представленная в работе [16]. Схема архитектуры показана в рисунке 2.7. Трансформеры представляют собой encoder-decoder модель с двумя частями: кодировщиком (Encoder) и декодировщиком (Decoder). Кодировщик обрабатывает входные данные и возвращает промежуточное представление, которое в дальнейшем используется как входные данные для декодировщика.

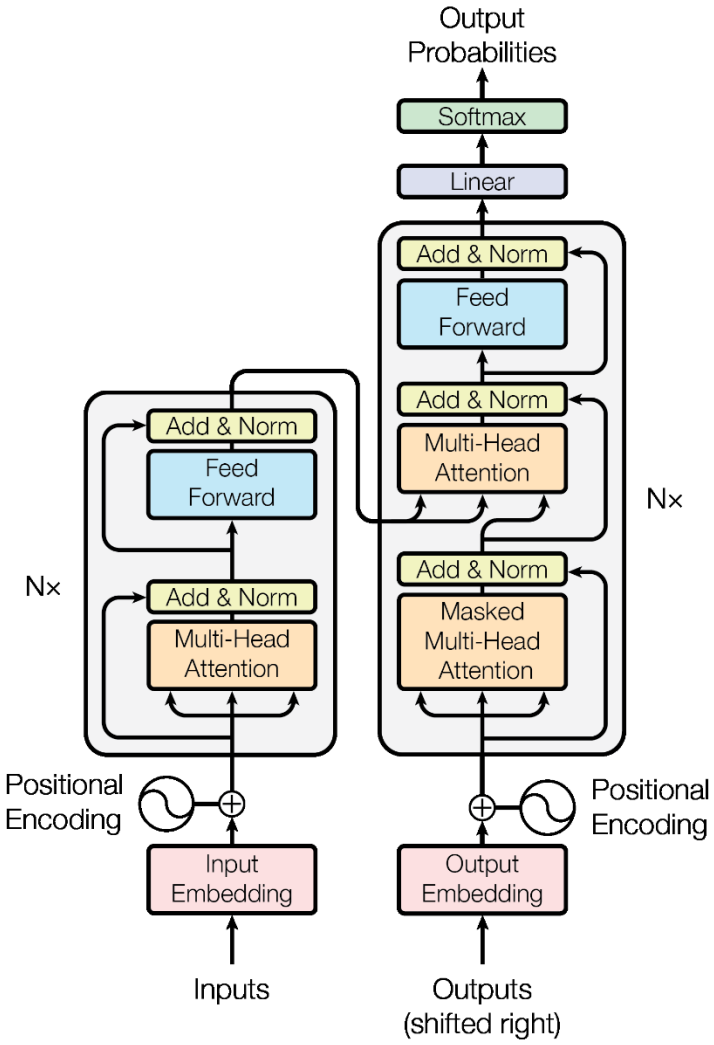

Рисунок 2.7 Архитектура Transformer

Архитектура трансформер принимает на вход последовательность векторов фиксированной размерности. Encoder и Decoder представляют собой последовательность из N блоков. Каждый блок кодировщика и декодировщика использует ключевую особенность архитектуры Transformers – механизм самовнимания (Self-Attention). Self-Attention пытается ответить на вопрос, как относится текущий элемент последовательности ко всем остальным элементам – как текущий элемент «смотрит» на остальные элементы. Для этого каждый входной элемент x_i в блоке Self-Attention представляется в виде трех векторов – запроса (query) q_i, ключа (key) k_i и значения (value) v_i. Значения q_i, k_i, v_i получаются путем умножения на матрицы весов W_q, W_k, W_v.

$$\begin{cases} q_i = W_q x_i \,, \\ k_i = W_k x_i \,, \\ v_i = W_v x_i \,. \end{cases}$$

Или, в матричном виде:

$$\begin{cases} Q = W_q X\,, \\ K = W_k X\,, \\ V = W_v X\,. \end{cases}$$

Далее для каждого запроса q_i и ключа k_j рассчитывается скалярное произведение и производится нормировка:

$$a_{ij} = \frac{q_i^T k_j}{\sqrt{d_k}}, i = 1 \dots n, j = 1 \dots n\,.$$

Или, в матричной форме:

$$A = \frac{QK^T}{\sqrt{d_k}}\,.$$

Пересечения i-ой строки и j-го столбца в матрице A представляют собой отношение i-го элемента к j-ому элементу последовательности, его внимание.

Выходные значения y_i содержат информацию об отношении текущего слова ко всем остальным и вычисляются как выпуклая комбинация значений v_j с коэффициентами $softmax$ преобразования от a_{ij}:

$$y_i = \sum_{j=1}^{n} softmax(a_{ij}) \cdot v_j = \sum_{j=1}^{n} softmax\left(\frac{q_i^T k_j}{\sqrt{d_k}}\right) \cdot v_j\,.$$

Или, в матричном виде:

$$Y = softmax(A) \cdot V = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V\,.$$

Каждый блок Encoder состоит из двух слоев:

1) Multi-Head Attention;

2) слоя проецирования.

Multi-Head Attention представляет собой h блоков Self-Attention, работающих параллельно. Результирующие матрицы $Y^{[i]}$ объединяются в одну и проецируются в нужную размерность с помощью матрицы весов W_O.

$$Z = concat\left(Y^{[1]} \dots Y^{[h]}\right) \cdot W_O\,,$$

$$Y^{[i]} = softmax\left(\frac{Q^{[i]}\left(K^{[i]}\right)^T}{\sqrt{d_k}}\right) \cdot V^{[i]}\,.$$

Multi-Head Attention позволяет модели использовать различные репрезентации отношений между элементами последовательности.

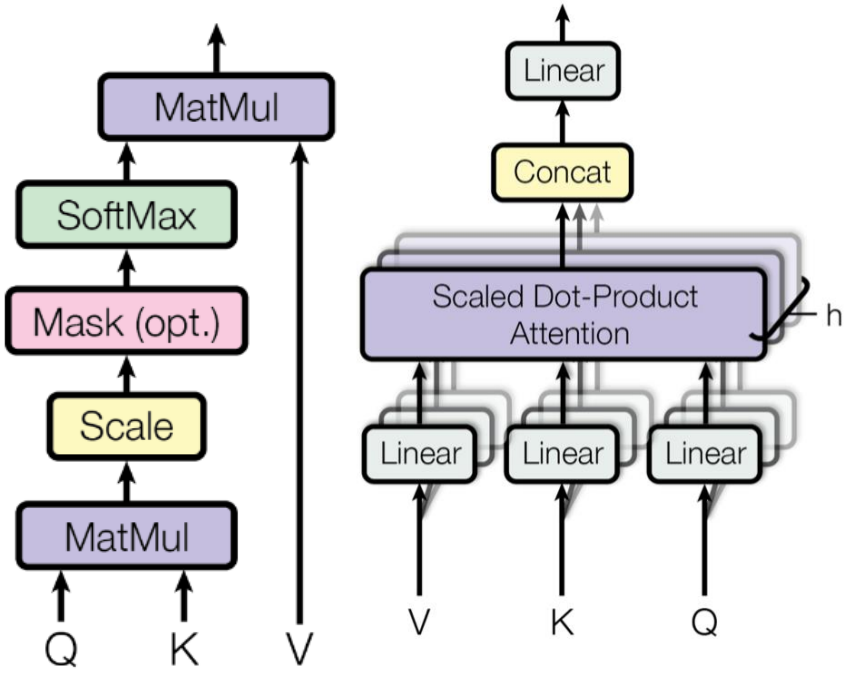

Рисунок 2.8 Схема слоев Attention и Multi-Head Attension

После каждого блока Multi-Head Attension и слоя проецирования находится слой LayerNorm [30], используемый для нормирования данных и устранения проблемы затухающих градиентов при обучении.

$$y = LayerNorm\big(x + Sublayer(x)\big)\,,$$

где x – входные данные, $Sublayer(x)$ – выходное значение предыдущего слоя, y – выходные значения слоя LayerNorm.

Каждый блок Decoder состоит из трех слоев:

1) Masked Multi-Head Attention;

2) Encoder-Decoder Multi-Head Attention;

3) слоя проецирования.

Так как Decoder генерирует слова последовательно, Masked Multi-Head Attention обнуляет коэффициенты внимания для тех элементов, которые еще не были предсказаны.

Encoder-Decoder Multi-Head Attention использует query с предыдущего слоя Decoder, но keys и values берутся с выходного слоя Encoder. Выходные значения Decoder подаются на вход линейному классификатору и последующему $softmax$ преобразованию для получения вероятностей принадлежности классам.

Архитектура Transformer обрабатывает входную последовательность целиком и не учитывает порядок элементов, что является недопустимым при обработке естественного языка. Для решения данной проблемы авторы статьи вводят понятие Positional Embeddings, позиционные представления, которые учитывают позицию элемента в последовательности pos и каждую компоненту i входного векторного представления.

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right),$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right),$$

где d_{model} – размерность входных векторных представлений.

Входные значения вычисляются как сумма векторных представлений и их позиционных представлений.

Модели на основе архитектуры Transformer создавались специально для решения задач обработки естественного языка (NLP), таких как: машинный перевод, классификация текста, генерация новых текстовых данных. И именно в данных задачах модели на основе архитектуры Transformer показали серьезное преимущество по сравнению с остальными архитектурами [13].

Также серьезным достоинством архитектуры Transformer является возможность параллельной обработки элементов последовательности, что может быть эффективно реализовано с использованием графических ускорителей (GPU).

2.4. BERT

Модели семейства BERT представляют собой кодировщик Трансформера, обученный специальным образом [**Ошибка! Источник ссылки не найден.**]. Название BERT расшифровывается как Bidirectional Encoder Representations from Transformers. Схема архитектуры BERT представлена в рисунке 2.9.

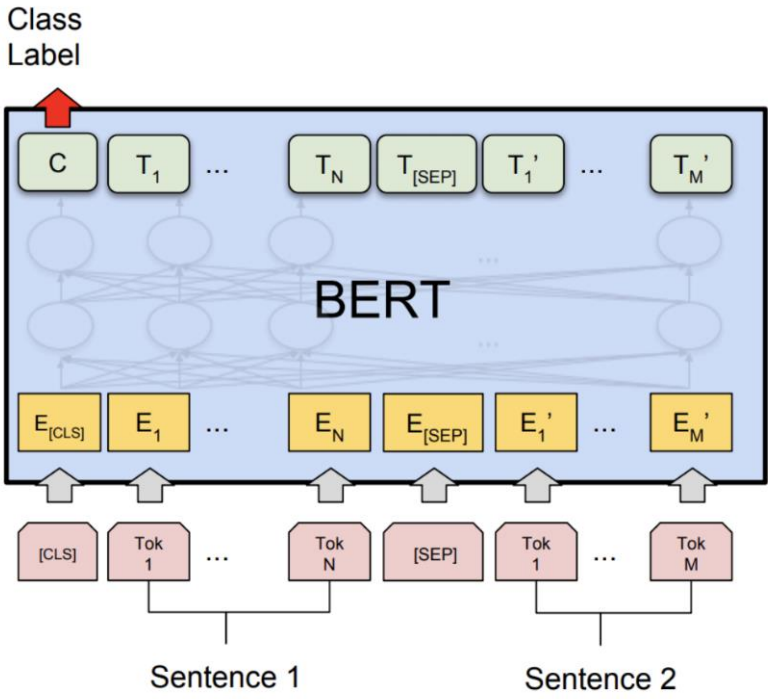

Рисунок 2.9. Архитектура BERT

Модели архитектуры BERT являются двунаправленными в том смысле, что каждое i-ое слово имеет в своем распоряжении левый (от 0 до $i-1$-го слова) и правый (от $i+1$ до n-го слова) контексты предложения.

Модели семейства BERT обучаются на двух задачах:

1) маскированное языковое моделирование (Masked Language Modeling – MLM);
2) предсказание следующего предложения (Next Sentence Prediction – NSP).

В задаче маскированного языкового моделирования некоторые входные токены (в оригинальной работе 15%) заменяются специальным токеном-маской – [MASK]. Модель должна предсказать какие токены были заменены маской.

Пример входных данных для задачи MLM:

1) Язык ##овые модели на базе архитектуры трансформ ##ер.

Входные данные после замены некоторых токенов на токен [MASK]:

1) Язык [MASK] модели на [MASK] архитектуры трансформ ##ер.

В примере видно, что одно слово может быть разбито на несколько токенов (Языковые – Язык, ##овые). Символы ## означают, что данный токен является частью предыдущего токена.

В задаче предсказания следующего предложения на вход к модели подаются два предложения: A и B. В конце каждого предложения идет токен-разделитель [SEP]. Необходимо определить, следует ли предложение B в тексте после предложения A.

Данная задача является задачей бинарной классификации, поэтому необходим еще один специальный токен – [CLS], который будет накапливать в себе информацию о том, насколько похожи предложения A и B. Токен [CLS] всегда находится на первой позиции входных данных. Выходное значение токена [CLS] используется для входа линейного классификатора.

Репрезентация входа для модели BERT представлена в 2.10.

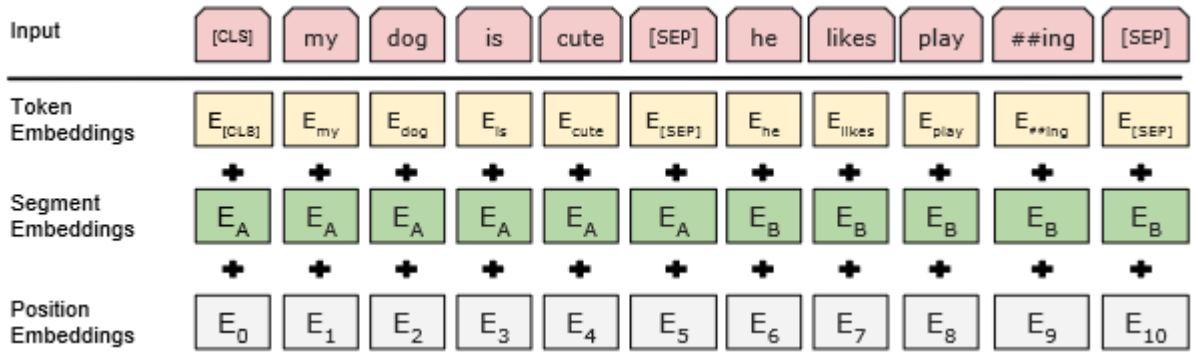

Рисунок 2.10 BERT input representation

В случае задачи классификации текста на вход подается только предложение A, последующие токены после токена-разделителя [SEP] обнуляются с помощью Attention Mask.

2.5. Классификация с помощью нейронных сетей

Стоит отметить, что однослойный перцептрон с одним выходным нейроном является простейшим линейным (бинарным) классификатором, так как его состояние $S = \langle w, x \rangle - b$ определяет гиперплоскость в пространстве входных данных R^n.

Линейный классификатор может использоваться для классификации линейно разделимых данных, в таком случае гиперплоскость будет разделяющей.

$$\begin{cases} \langle w, x \rangle - b < 0, & \forall x \in C_1 , \\ \langle w, x \rangle - b > 0, & \forall x \in C_2 . \end{cases}$$

Разделяющая плоскость в двумерном пространстве имеет следующий вид, представленный в рисунке 2.11.

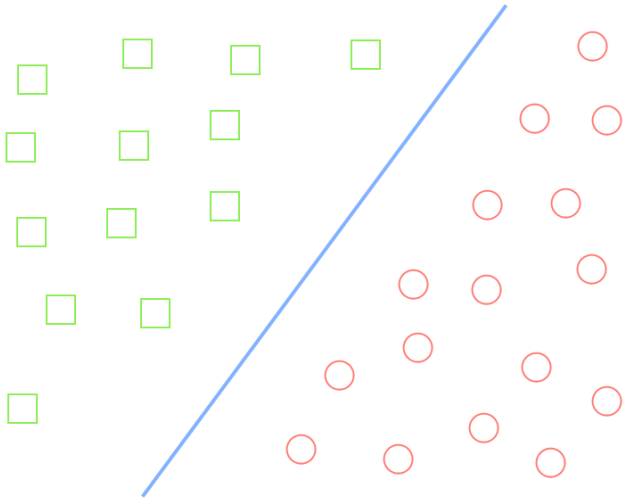

Рисунок 2.11 Разделяющая плоскость в двумерном пространстве

Если пронумеровать классы C_1 и C_2 как $+1$ и -1, то работа линейного классификатора может быть выражена следующей формулой:

$$y = sign\left(\sum_{i=1}^{n} x_i w_i - w_0\right).$$

В свою очередь, перцептроны с несколькими скрытыми слоями могут использоваться для классификации линейно неразделимых данных и, соответственно, являются нелинейными классификаторами.

Перцептрон с K выходными нейронами (логитами) $z_1 \dots z_n$ используется для многоклассовой классификации, где K – число интересующих классов. С

помощью $softmax$ преобразования логиты преобразуются в вероятности каждого из классов:

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}.$$

где $z_k = \sum_{i=1}^{n} x_i^{[k]} w_i^{[k]} - w_0^{[k]}$ – выходное значение k-ого нейрона.

Нужный класс обычно определяется как номер элемента $softmax$ преобразования с наибольшим значением вероятности:

$$y = arg \max_{k=1\ldots K} softmax(z)_k .$$

Работу многоклассового классификатора в случае K классов можно представить следующим образом 2.12.

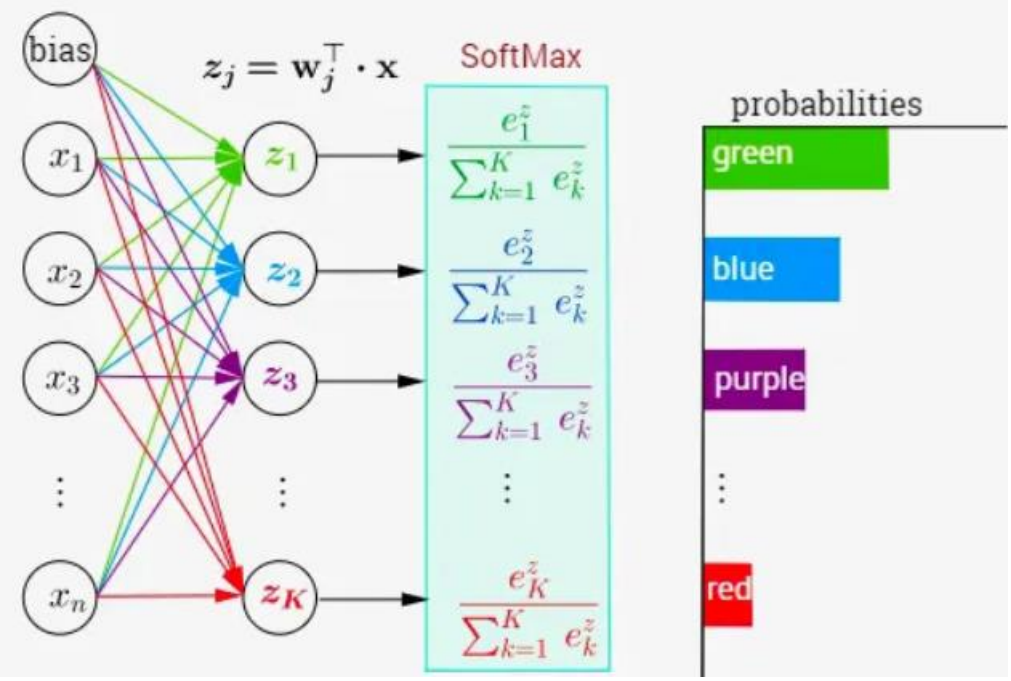

Рисунок 2.12 Многоклассовая классификация многослойным перцептроном

Вернемся к вопросу использования других типов нейронных для решения задачи классификации. Допустим, что используется сеть, которая возвращает выходной вектор y $\in R^n$. Также можно сказать, что сеть по входным данным x получила внутреннее представление y.

В таком случае значения y подается на вход к обычному перцептрону, выступающему в роле классификатора. Обычно используется перцептрон с одним скрытым слоем или без скрытых слоев. Количество нейронов на

выходном слое соответствует количеству классов в задаче. Выходной слой в таком случае называется также слоем проецирования.

Рассмотрим данных подход на примере классификации текста с помощью модели BERT, рисунок 2.13.

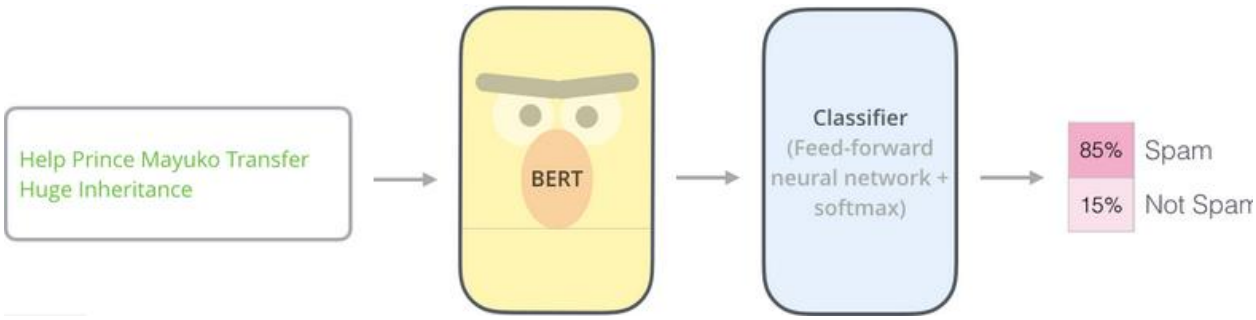

Рисунок 2.13 Classification pipeline

Можно заметить, что при решении конкретной задачи можно взять модель, обученную для решения похожей задачи, заменить классификатор и дообучить модель на своих данных для решения своей задачи. Такой подход получил название Transfer Learning и активно используется при решении задач обработки естественного языка с использованием архитектуры Tranformer.

При этом взятую модель называют предобученной, а процесс дообучения для решения конкретной задачи – тонкой настройкой (fine tuning). Использование Transfer Learning позволяет пользоваться при решении задач моделями с большим числом параметров и при этом не тратить огромное количество времени на обучение модели с нуля.

2.6. Выводы по второй главе

Во второй главе были рассмотрены основные архитектуры нейронных сетей, использующиеся при решении задач классификации текста. В начале главы описаны многослойные полносвязные сети прямого распространения. После многослойных сетей произведен анализ рекуррентных нейронных сетей и их модификаций: двунаправленные рекуррентные сети. Отдельно были рассмотрены сети с долгой краткосрочной памятью (LSTM), и их модификации: Stacked LSTM.

Наибольшее внимание было уделено архитектуре нейронных сетей под названием Transformer. Подробно рассмотрены ключевые особенности архитектуры, механизмы Self-Attention, Multi-Head Attention, Positional Embeddings. Отдельное внимание было уделено семейству моделей BERT и их применению в решении задач обработки естественного языка.

В конце главы описано применение нейронных сетей при решении задач классификации текста, введены такие понятия как: разделяющая гиперплоскость, $softmax$ преобразование. Кратко рассказано про подход к обучению Transfer Learning, какие преимущества дает данный подход, что такое fine tuning.

3. РЕШЕНИЕ ЗАДАЧИ

Перейдем непосредственно к решению задачи автоматической категоризации товаров по их наименованиям на конкретном примере. Решение данной задачи разбивается на решение нескольких подзадач:

1) поиск подходящего датасета для обучения и тестирования модели;

2) обработка датасета, если существует такая необходимость;

3) выбор модели;

4) обучение и тестирование модели;

5) подведение результатов.

3.1. Поиск датасета

Если говорить про поиск нужного датасета, то существует несколько вариантов:

1) найти в открытых источниках размеченный датасет;

2) спарсить каталог товаров с интернет-сайта;

Русскоязычных размеченных датасетов в открытом доступе найти не удалось. Если рассматривать англоязычные датасеты, то среди них можно выделить несколько размеченных датасетов товаров, таких как: Fashion products on Amazon.com и Innerwear Data from Victoria's Secret and Others. Однако стоит отметить, что данные датасеты довольно специфичны и не подходят для построения универсального классификатора.

Следующим вариантом составления датасета является парсинг каталога товаров с интернет-сайта. Данная задача не имеет готовых решений, или не может быть сделана без дополнительных инвестиций. Написание же своего решения выходит за рамки данной работы.

С учетом отсутствия русскоязычных датасетов и специфичности англоязычных, компанией ООО «Наполеон АЙТИ». был предоставлен датасет продуктовых товаров крупного российского ретейлера.

3.2. Обработка датасета

Датасет представляет собой продуктовый каталог, в основном алкогольной продукции. Формат файла данных датасета – csv. Каждый товар имеет две характеристики: текстовое описание и категорию, к которой данный товар принадлежит.

В качестве языка программирования был выбран язык Python 3.7.13 с использованием интерактивной вэб-среды для создания Jupyter блокнотов Google Colaboratory. Для обработки и анализа данных использовалась библиотека Pandas.

Изначально в датасете содержится 8684 записей. Первым делом удалим строки-дубликаты, после этого удалим строки с некорректными данными, такими как: пустые поля, отсутствующие значения. После данных манипуляций получим датасет, имеющий 8354 уникальных записей.

Проанализируем полученный датасет. Для этого построим гистограмму распределения классов в датасете 3.1.

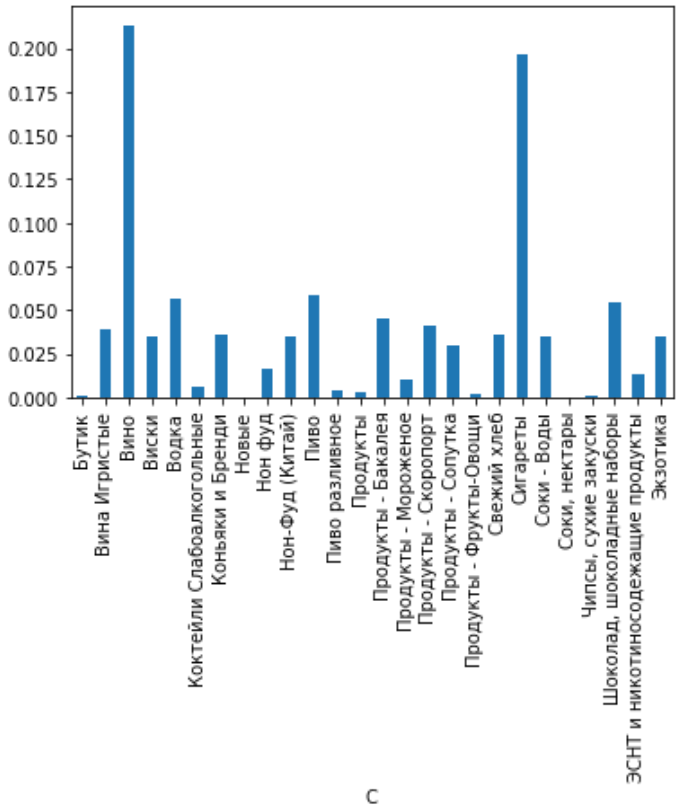

Рисунок 3.1 Диаграмма распределения категорий товаров в датасете

Видно, что категории товаров распределены неравномерно. Большинство товаров в датасете принадлежат двум группам категорий: алкогольной продукции и табачным изделиям. При этом видно, что алкогольная продукция размечена достаточно хорошо, чего нельзя сказать про остальные категории товаров.

Например, присутствуют странные категории, такие как: «Продукты – Скоропорт», «Продукты – Сопутка» и «Новые». Также имеется довольно странное разделение непродовольственных товаров на изготовленные в Китае и остальные. Вызывает вопросы разделение пива на разливное и обычное.

Суммируя вышеперечисленное, можно сделать вывод, что текущая категоризация каталога не соответствует общероссийскому классификатору продукции ОКПД2. Очевидно, что необходимо сделать переразметку датасета. Переразметка датасета в соответствии с ОКПД2 позволит обучить универсальный классификатор, способный переводить различные разметки датасетов в единую систему категорий.

Переразметку датасета разделим на два этапа:

1) автоматическая разметка методами Pandas;

2) ручная разметка оставшихся данных с помощью doccano [27].

Подробно пройдемся по каждому этапу переразметки данных. В качестве референса для определения новых категорий будем использовать уже упомянутый ОКПД2, но при этом по возможности будем объединять схожие категории, имеющие небольшое число товаров, так как в противном случае модель не сможет обучиться на недостаточном количестве данных.

В результате анализа датасета было определено 24 категорий товаров: «Сигареты», «Шоколад и кондитерские изделия», «Водка», «Вино», «Пиво», «Виски», «ЭСНТ и никотиносодежащие продукты», «Изделия хлебобулочные», «Чипсы, сухари, снеки», «Все остальное», «Коньяк», «Бренди», «Мясные изделия», «Вода питьевая», «Соусы», «Соки, Нектары», «Молоко и молочная продукция», «Рыба вяленая, соленая и несоленая»,

«Орехи», «Мороженое», «Семечки», «Овощи свежие», «Фрукты свежие», «Чай и кофе».

Для хорошо размеченных категорий товаров, в основном это касается алкогольной продукции, будем просто менять название соответствующей категории на новое. В остальном случае будем составлять правила перевода товара из одной категории в другую.

Пример замены категории товара представлен в листинге 3.1:

Листинг 3.1. – Пример замены категории товара

```
df3.loc[df3.D.str.contains("виски", case=False) == True, 'C'] = "Виски"
```

В общей сложности с помощью автоматической разметки удалось разметить 5574 элементов. Оставшиеся элементы были размечены вручную. Для ручной разметки использовался doccano – инструмент для текстовых аннотаций с открытым исходным кодом.

В общей сложности было размечено 8354 элемента. Итоговая диаграмма распределения классов имеет следующий вид, представленный в рисунке 3.2.

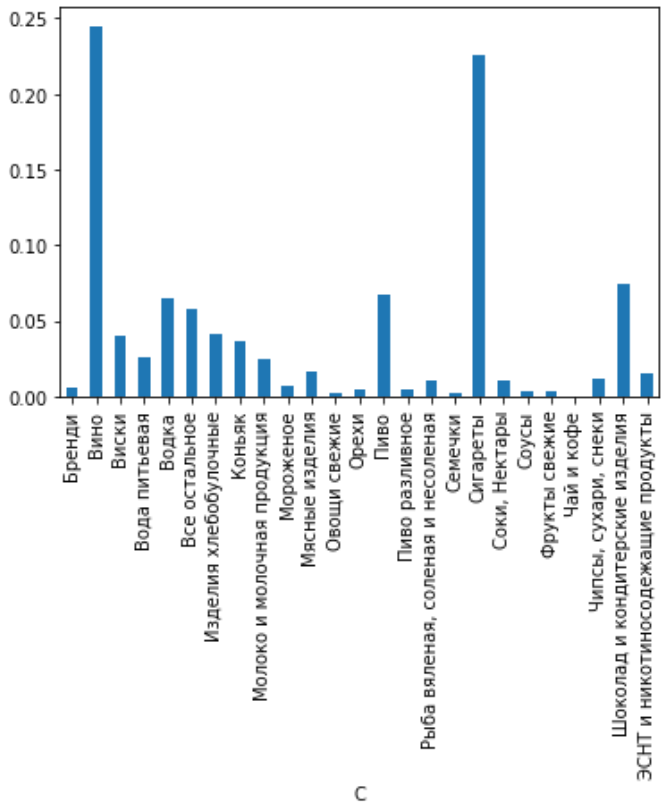

Рисунок 3.2 Диаграмма распределения категорий после переразметки

3.3. Выбор модели

Одним из вариантов, возникающих при выборе модели, является обучение модели с нуля. Такой подход имеет место, если существует большой объем данных для обучения и необходимые вычислительные ресурсы. Но, так как имеющийся датасет довольно маленький и отсутствуют необходимые вычислительные ресурсы, использование данного варианта обучения нецелесообразно. Другим вариантом является использование предобученной модели с последующим дообучением на своих данных. Более подробно данный подход разобран в разделе 2.5.

Существует несколько проектов, занимающихся обучением русскоязычных моделей, такие как DeepPavlov или Sberbank AI. Данные проекты выкладывают в открытый доступ предобученные русскоязычные версии BERT, которые можно использовать для решения задачи классификации.

Но давайте еще раз посмотрим на датасет, точнее, на среднее количество слов в предложении. В среднем каждое описание товара содержит 6 слов. Видно, что контекст маленький и использование больших моделей может быть не очень оптимально. Также стоит отметить, что даже дообучение больших моделей может занимать много времени и требовать большое количество вычислительных ресурсов.

Специально для таких случаев существует несколько tiny-версий BERT, одна из которых предложена в работе [21]. Данная модель называется rubert-tiny и представляет собой уменьшенную версию модели bert-multilingual [**Ошибка! Источник ссылки не найден.**]. Словарь модели был уменьшен с 120K до 30K, размер эмбеддингов с 786 до 312, число слоев с 12 до 3. Обучение происходило путем дистилляции знаний с уже обученных русскоязычных моделей.

Модель выложена в открытый доступ на площадке Hugging Face [**Ошибка! Источник ссылки не найден.**], поэтому будем использовать библиотеки данного проекта для дообучения модели на наших данных.

3.4. Обучение модели

Проект Hugging Face предоставляет две библиотеки: datasets и transformers. С помощью библиотеки datasets мы преобразуем наши данные в форму, подходящую для обучения модели:

Листинг 3.2. – Преобразование данных

```
dataset = Dataset.from_pandas(a.drop("product_id", axis=1).rename(
    columns={"D": "text", "C": "label"
}), preserve_index=False)
```

Перед тем, как дообучить нашу модель, разделим датасет на три выборки в соотношении 0.8 / 0.1 / 0.1: train, validation и test. На выборке train будет обучаться наша модель, на выборке validation будет производиться донастройка гиперпараметров модели после каждой эпохи обучения, на выборке test будет осуществляться проверка качества полученной модели.

Листинг 3.3. – Разделение датасета на три выборки

```
train_test = tokenized_datasets.train_test_split(test_size=0.1)
test_valid = train_test["test"].train_test_split(test_size=0.5)
train, test, valid = train_test["train"], test_valid["train"],
test_valid["test"]
```

Библиотека transformers предоставляет несколько классов, помогающих при обучении моделей. Для задачи классификации существует класс AutoModelForSequenceClassification, который загружает предобученную модель с Hugging Face Hub и добавляет классификатор под нужное число классов. Загрузим выбранную модель и укажем необходимое число категорий классификации num_labels:

Листинг 3.4. – Загрузка предобученной модели

```
model = AutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=len(categories))
```

Токенизация датасета осуществляется с помощью класса AutoTokenizer:

Листинг 3.5. – Токенизация датасета

```
tokenizer = AutoTokenizer.from_pretrained(model_name)

def tokenize_function(batch):
    tokenized_batch = tokenizer(batch["text"], padding="max_length",
truncation=True)
    tokenized_batch["label"] = [str2int[label] for label in batch["label"]]
```

```
    return tokenized_batch
tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

Для настройки гиперпараметров модели используется класс TrainningArguments. С помощью данного класса изменим следующие параметры обучения: output_dir, evaluation_strategy и num_train_epochs.

Значение параметра output_dir обозначает директорию, в которую будут сохраняться промежуточные веса и конфигурация модели во время обучения.

Параметр evaluation_strategy определяет необходимость и стратегию валидации модели по время обучения. Значение «epoch» означает, что валидация будет происходить после каждой эпохи обучения.

Параметр num_train_epochs отвечает за количество прохождений модели по всему датасету – количеству эпох обучения.

Листинг 3.6. – Выбор гиперпараметров обучения модели

```
training_args = TrainingArguments(
    output_dir="test_trainer",
    evaluation_strategy="epoch",
    num_train_epochs=3,
    logging_steps=len(train) // (5 * 8)
)
```

Обучение модели происходит посредством класса Trainer. Класс Trainer принимает на вход модель, параметры обучения, обучающую выборку, валидационный датасет и метрики качества, которые используются при валидации и тестировании модели. Учитывая анализ метрик, проведенный в разделе 1.4, будем использовать метрики F_{macro} и $F_{weighted}$.

Листинг 3.7. – Выбор гиперпараметров обучения модели

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train,
    eval_dataset=valid,
    compute_metrics=compute_metrics,
    callbacks=[PrinterCallback]
)
```

По умолчанию при обучении моделей класса BERT в роли функции потерь используется перекрестная энтропия (Cross Entropy Loss) [17]. Оптимизатором выступает модифицированная версия оптимизатора Adam –

AdamW [11]. Темп обучения составляет вначале составляет 0.001, но меняется во время обучения.

3.5. Результаты

В результате 3 эпох обучения получаем следующие графики функции потерь на тренировочной и валидационной выборках 3.3.

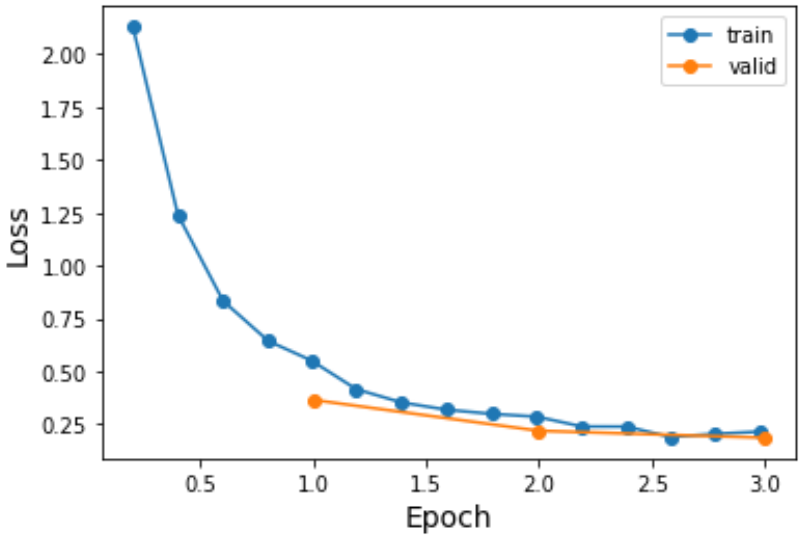

Рисунок 3.3 Графики функции потерь

Отчетливо видно, что значения функции потерь уменьшаются на обоих выборках. Теперь посмотрим на метрики качества модели на валидационном датасете после каждой эпохи обучения 3.4.

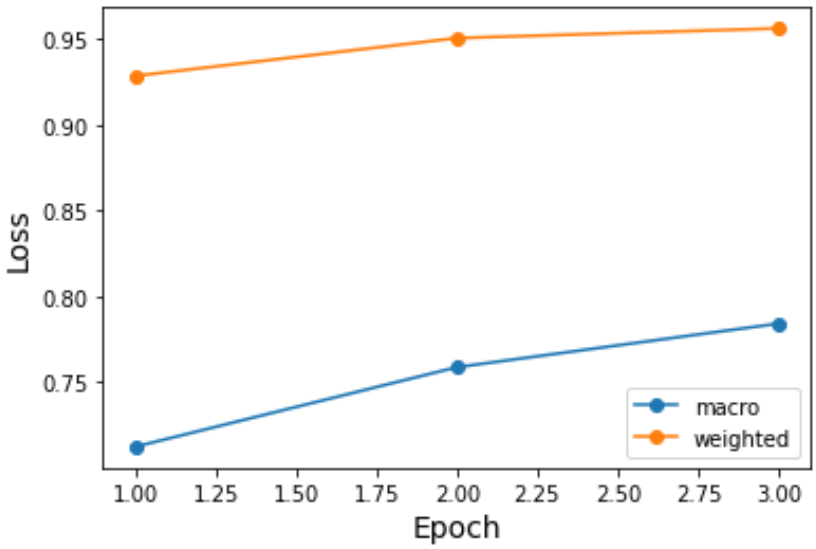

Рисунок 3.4 Графики метрик качества

Метрика F_{macro} показывает значения, близкие к 0.95, что свидетельствует об отличном качестве классификации в рамках всего датасета.

Но при этом значения $F_{weighted}$ не достигает порога 0.8. Данный факт может быть связан с не очень хорошим уровнем классификации для отдельных классов. Чтобы проверить данную гипотезу, посмотрим на матрицу несоответствий для тренировочной выборки 3.5.

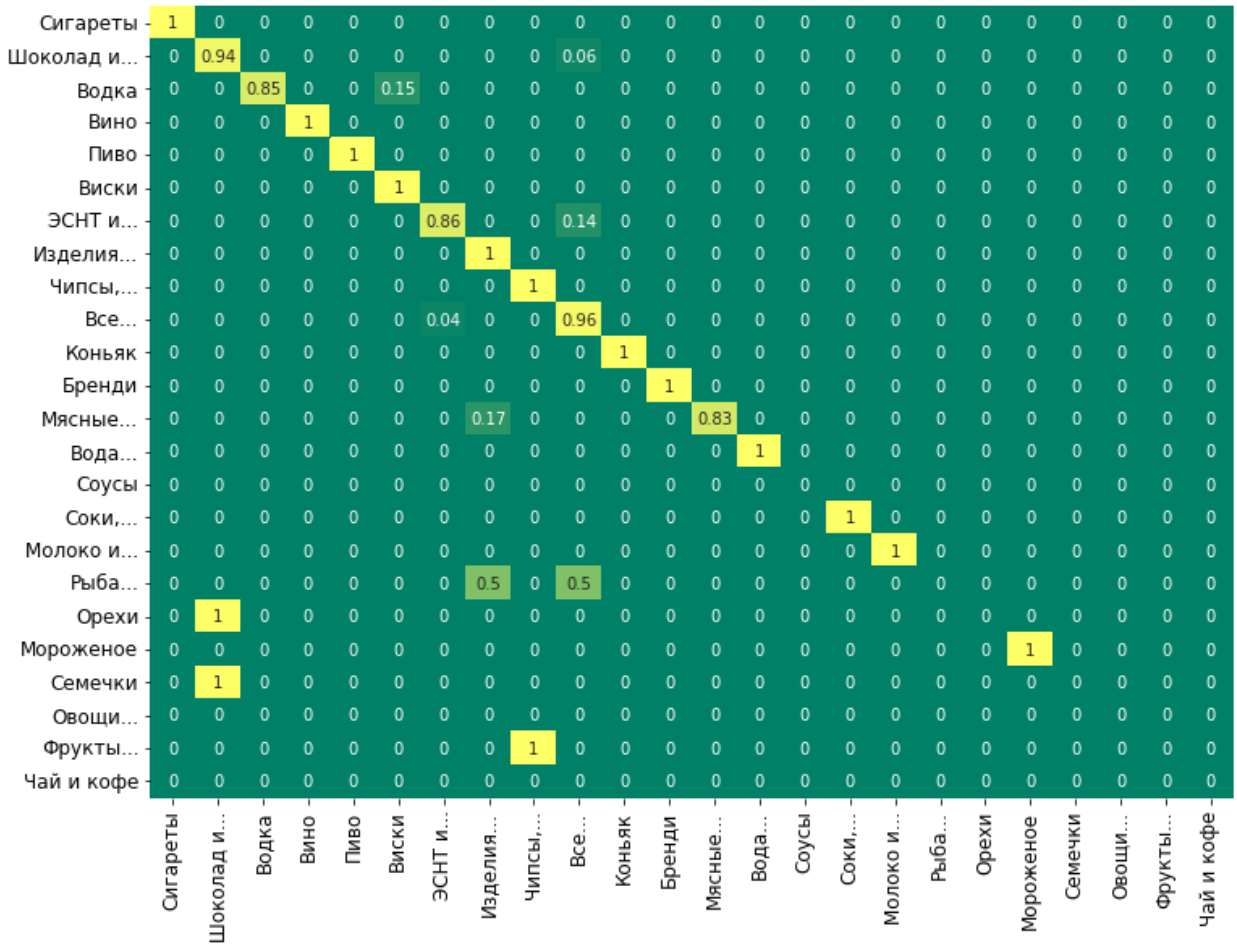

Рисунок 3.5 Матрица несоответствий для тестовой выборки

Анализируя матрицу несоответствий, можно увидеть, что для некоторых классов модель не сделала ни одного правильного предсказывания. Данные классы объединяет одна общая черта – малое количество объектов класса в датасете (< 30).

Можно сделать вывод, что для получения качественной модели необходимо большое число обучающих экземпляров, для каждого класса желательно как минимум 50 объектов в датасете.

3.6. Выводы по третьей главе

В третьей главе была решена практическая задача классификации текста. В качестве датасета выступал продуктовый каталог товаров. Перед тем как обучить модель, датасет был переразмечен в соответствии с ОКПД2.

Был произведен выбор модели. В качестве модели была взята русскоязычная версия BERT, представленная в работе [21]. Был подробно рассмотрен процесс обучения модели. Модель была дообучена на тренировочной выборке датасета.

Результаты обучения модели представлены в виде графиков функции потерь и значения метрик качества при обучении. Для тренировочной выборки представлена матрица несоответствий.

ЗАКЛЮЧЕНИЕ

Выпускная квалификационная работа посвящена разработке модели для автоматической категоризации каталога товаров.

В работе был произведен анализ основных подходов к решению задачи. Описаны главные концепции и определения машинного обучения. Были рассмотрены основные типы классификаторов, формализовано понятие искусственного нейрона. Отдельный раздел посвящен оценке качества моделей при решении задач классификации. Изучены различные решения проблемы векторного представления текстовых данных.

Рассмотрены основные типы архитектур нейронных сетей, использующихся при решении задач обработки естественного языка. Дается анализ использования нейронных сетей для решения задач классификации, описан подход переноса знаний между моделями – Transfer Learning.

Решена практическая задача классификации на примере каталога именованных товаров. Подробно расписаны стадии решения задачи, такие как: поиск и обработка датасета, выбор и обучение модели. Произведен анализ результатов полученной модели.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Розенблатт, Ф. Принципы нейродинамики: Перцептроны и теория механизмов мозга / Ф. Розенблатт – Москва: Мир, 1965.
2. Шитиков, В.К. Классификация, регрессия и другие алгоритмы Data Mining с использованием R / В.К. Шитиков, С.Э. Мастицкий –Лондон: Б.и., 2014.
3. Elman, J. Finding structure in time / J. Elman – San Diego: Cognitive Science, 1990.
4. Rumelhart, D. Parallel Distributed Processing: Explorations in the Microstructures of Cognition / D. Rumelhart – Cambridge, MA: MIT Press
5. Cui, Z. Stacked Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction / R. Ke, Z. Pu, Y. Wang – // Proceedings of Transportation Research Part C Emerging Technologies. – 2020.
6. Devlin, J. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / J. Devlin, K. Lee, K. Toutanova // Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. – 2019. – С. 1-40.
7. Fernández Delgado, M. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems / M. Fernández Delgado, E. Cernadas, S. Barro, D Amorim // The Journal of Machine Learning Research. – 2014.
8. Kowsari, K. Text Classification Algorithms: A Survey / K. Kowsari, K. Meimandi, M. Heidarysafa, S. Mendu, L. Barner, D. Brown // Proceedings of Workshop at MDPI. – 2019.
9. Le, Q. Distributed Representations of Sentences and Documents / Q. Le, T. Mikolov // Proceedings of the 31st International Conference on Machine Learning. – 2014.
10. Li, Q. A Survey on Text Classification: From Traditional to DeepLearning // Q. Li, H. Peng, J. Li, C. Xia, R. Yang, L. Sun, P. Yu, L. He // Proceedings of Workshop at ACM Transactions on Intelligent Systems and Technology. – 2022.

11. Loshchilov, I. Decoupled Weight Decay Regularization / I. Loshchilov, F. Hutter // Proceedings of Workshop at ICLR. – 2019.
12. Mikolov, T. Efficient estimation of word representations in vector space / T. Mikolov, K. Chen. // Proceedings of Workshop at ICLR. – 2013. – С. 1–12.
13. Minaee, S. Deep Learning Based Text Classification: A Comprehensive Review / N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, J. Gao // Proceedings of ACM Computing Surveys. – 2022.
14. Pascanu, R. On the difficulty of training recurrent neural networks / R. Pascanu, T. Mikolov, Y. Bengio // Proceedings of the 30th International Conference on Machine Learning. – 2013.
15. Schmidhuber, J. Long Short-Term Memory / J. Schmidhuber, S. Hochreiter // Neural Computation. – 1997.
16. Vaswani, A. Attention is all you need / A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, I. Polosukhin // Proceedings of the NeurIPS. – 2017. – С. 1-7.
17. Zhang, Z. Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels / Z. Zhang, Mert R. Sabuncu // Proceedings of the NeurIPS. – 2018.
18. Дерево решений // Wikipedia, доступ: https://ru.wikipedia.org/wiki/Дерево_решений (дата обращения – 3 марта 2022).
19. Искусственный интеллект // Wikipedia, доступ: https://en.wikipedia.org/wiki/Artificial_intelligence (дата обращения – 18 мая 2022).
20. Лекции по искусственным нейронным сетям // MachineLearning, доступ: http://www.machinelearning.ru/wiki/images/c/cc/voron-ml-neuralnets.pdf (дата обращения – 5 марта 2022).
21. Маленький и быстрый BERT для русского языка // Habr, доступ: https://habr.com/ru/post/562064/ (дата обращения – 23 апреля 2022).

22. Математические методы обучения по прецендентам // MachineLearning, доступ: http://www.machinelearning.ru/wiki/images/6/6d/Voron-ML-1.pdf (дата обращения – 9 марта 2022).
23. Машина опорных векторов // MachineLearning, доступ: http://www.machinelearning.ru/wiki/index.php?title=Метод_опорных_векторов (дата обращения – 3 марта 2022).
24. Метод ближайших соседей // MachineLearning, доступ: http://www.machinelearning.ru/wiki/index.php?title=Метод_ближайшего_соседа (дата обращения – 12 марта 2022).
25. Учебник по машинному обучению от Школы анализа данных // ml-handbook, доступ: https://ml-handbook.ru/ (дата обращения – 16 апреля 2022).
26. A Survey on Contextual Embeddings // arxiv, доступ: https://arxiv.org/abs/2003.07278 (дата обращения – 25 мая 2022).
27. Doccano // GitHub, доступ: https://github.com/doccano/doccano (дата обращения – 14 марта 2022).
28. Deep Learning for Natural Language Processing // logic.pdmi.ras.ru, доступ: https://logic.pdmi.ras.ru/~sergey/slides/N16_AIUkraineDLNLP.pdf (дата обращения – 28 апреля 2022).
29. cointegrated/rubert-tiny // Hugging Face, доступ: https://huggingface.co/cointegrated/rubert-tiny (дата обращения – 12 мая 2022).
30. Layer normalization // arxiv, доступ: https://arxiv.org/pdf/1607.06450 (дата обращения – 16 мая 2022).
31. Macro F1 and Macro F1 // arxiv, доступ: https://arxiv.org/abs/1911.03347 (дата обращения – 8 апреля 2022).
32. The Hadamard Product // buzzard.ups.edu, доступ: http://buzzard.ups.edu/courses/2007spring/projects/million-paper.pdf (дата обращения – 2 апреля 2022).

ПРИЛОЖЕНИЕ

Листинг П.1 – Jupyter Notebook

```
import csv
import math
import numpy as np
import pandas as pd
from datasets import Dataset, DatasetDict, load_metric
from transformers import (
    AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments,
    Trainer, TrainerCallback
)
from ipywidgets import interact

!gdown https://drive.google.com/uc?id=1nZWeSDHjrKtRliTgtGpr4CGwS9WzTHa7

def write(filename, content):
    with open(filename, "w", encoding="utf-8") as file:
        for i, entry in enumerate(content["D"]):
            if entry is None or isinstance(entry, float):
                continue
            file.write(entry)
            file.write("\n")

def stats(df, column):
    c = df.groupby(column)[column].count()
    c = (c / c.sum())
    #with pd.option_context('display.float_format', '{:0.3f}'.format):
    #    print(c / c.sum())
    c.plot.bar()

df = pd.read_csv("data.csv")
len(df["Description"].unique())

df = df.rename(columns={"Category": "C", "Description": "D"})
df = df.drop_duplicates("D", keep=False)
df = df.replace("None", np.nan)
df = df.dropna()
df

df.head()

len(df["D"].unique())

len(df["C"].unique())

df.to_csv("test.csv", index=False)

vocab = set()
sizes, words = [], []
with open("test.csv", encoding="utf-8") as file:
    reader = csv.reader(file)
    for i, (prod_id, info, class_id) in enumerate(reader):
        vocab.update(info.split())
        words.extend(info.split())
        sizes.append(info)
sizes = list(map(len, sizes))
mean = sum(sizes) / i
std = math.sqrt(sum((x - mean)**2 for x in sizes)/len(sizes))
print(f"vocab len: {len(vocab)}")
print(f"mean: {mean}")
```

Продолжение Листинга П.1

```
print(f"std: {std}")
print(f"cv: {std/mean}")
print(len(words)/i)

@interact(category=df['C'].unique())
def f(category):
    return df.loc[df['C'] == category]

stats(df, "C")

df2 = df[df.C.isin([
    'Бутик', 'Новые', 'Нон фуд', 'Нон-Фуд (Китай)', 'Соки, нектары',
    'Коньяки и Бренди', 'Продукты', 'Продукты - Скоропорт', 'Продукты -
Сопутка',
    'Продукты - Фрукты-Овощи', 'Соки - Воды', 'Чипсы, сухие закуски',
    'Продукты - Мороженое'
])].drop_duplicates(subset="D")
df2.replace("", np.nan).dropna()
df2

#remaining[['D', 'C']].to_csv("remaining.csv", index=False,
quoting=csv.QUOTE_ALL)
df3 = df2.copy()
df3.loc[(df3.C == "Нон фуд") | (df3.C == "Нон-Фуд (Китай)"), 'C'] = "Все
остальное"
df3.loc[(df3.C == "Соки - Воды") & (df3.D.str.contains("сок|нектар",
case=False) == True), 'C'] = "Соки, Нектары"
df3.loc[(df3.C == "Соки - Воды") &
(df3.D.str.contains("вод|напиток|кола|лимонад|квас", case=False) == True),
'C'] = "Вода питьевая"
df3.loc[df3.D.str.contains("мороженое", case=False) == True, 'C'] =
"Мороженое"
df3.loc[df3.D.str.contains("семе", case=False) == True, 'C'] = "Семечки"
df3.loc[df3.D.str.contains("орех|миндаль|фундук|арахис|фисташки", case=False)
== True, 'C'] = "Орехи"
df3.loc[df3.D.str.contains("ж/р|драже|торт|леденцы|карамель|конфет",
case=False) == True, 'C'] = "Шоколад и кондитерские изделия"
df3.loc[df3.D.str.contains("йогурт|сыр|творо|сливки|сметана|молоко|кефир|масл
о", case=False) == True, 'C'] = "Молоко и молочная продукция"
df3.loc[df3.D.str.contains("свин|сосиск|колбас|нарезк|ветчин|шпикачки|пивчик|
грудинка|карбонад|говядина|джерки|сарде|буже", case=False) == True, 'C'] =
"Мясные изделия"
df3.loc[df3.D.str.contains("сушен|вялен|соленая|холодн|кальмар|сельд|скумбри|
горбуш|полосатик", case=False) == True, 'C'] = "Рыба вяленая, соленая и
несоленая"
df3.loc[df3.D.str.contains("чипс|сухар|гренки", case=False) == True, 'C'] =
"Чипсы, сухари, снеки"
df3.loc[df3.D.str.contains("виски", case=False) == True, 'C'] = "Виски"
df3.loc[df3.D.str.contains("коньяк", case=False) == True, 'C'] = "Коньяк"
df3.loc[df3.D.str.contains("бренди|арманьяк", case=False) == True, 'C'] =
"Бренди"
df3.loc[df3.D.str.contains("соус|кетчуп|майо", case=False) == True, 'C'] =
"Соусы"
class_map = [
    "Все остальное", "Соки, Нектары", "Вода питьевая", "Мясные изделия",
"Молоко и молочная продукция", "Виски",
    "Чипсы, сухари, снеки", "Коньяк", "Бренди", "Шоколад и кондитерские
изделия", "Рыба вяленая, соленая и несоленая",
```

Продолжение Листинга П.1

```
    "Орехи", "Мороженое", "Семечки", "Соусы"
]
mapped = df3.loc[df3.C.isin(class_map)]
remaining = df3.loc[~df3.C.isin(class_map)]
write("remaining.txt", remaining)
mapped

remaining

!gdown https://drive.google.com/uc?id=16Y8F2_UGQ19Svr8hZdbSIhil2Dp0287g

labeled = pd.read_csv("vsevolod.csv")
labeled = labeled.rename(columns={"data": "D", "label": "C"})
labeled.drop(labels='id', axis=1, inplace=True)
all = pd.merge(mapped, labeled, how="outer")
all

print(len(all['C'].unique()))
all['C'].unique()

write("data.txt", df2)

normal = df[df.C.isin([
    "Пиво", "Пиво разливное", "Водка", "Виски", "Вино", "Вина игристые",
    "Свежий хлеб", "Сигареты", "ЭСНТ и никотиносодежащие продукты",
    "Шоколад, шоколадные наборы", "Свежий хлеб"
])].replace(
    ["Шоколад, шоколадные наборы", "Свежий хлеб"],
    ["Шоколад и кондитерские изделия", "Изделия хлебобулочные"]
)
normal

#a[a.duplicated("D", keep=False)]
a = pd.merge(normal, all, how="outer")
a.loc[a.C == "Пиво разливное", 'C'] = "Пиво"
a

print(len(a['C'].unique()))

a['C'].unique()

@interact(_df=["a", "df"])
def f(_df):
    return stats({"a":a, "df":df}[_df], "C")

model_name = "cointegrated/rubert-tiny"

tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer

categories = list(a.C.unique())

str2int = {s: i for i, s in enumerate(categories)}
int2str = {i: s for i, s in enumerate(categories)}

dataset = Dataset.from_pandas(a.drop("product_id", axis=1).rename(columns={
    "D": "text", "C": "label"
}), preserve_index=False)
```

Продолжение Листинга П.1

```
def tokenize_function(batch):

    tokenized_batch = tokenizer(batch["text"], padding="max_length",
truncation=True)
    tokenized_batch["label"] = [str2int[label] for label in batch["label"]]
    return tokenized_batch
tokenized_datasets = dataset.map(tokenize_function, batched=True)
train_test = tokenized_datasets.train_test_split(test_size=0.1)
test_valid = train_test["test"].train_test_split(test_size=0.5)
train, test, valid = train_test["train"], test_valid["train"],
test_valid["test"]

@interact(_df=["all", "test"])
def f(_df):
    return stats({
        "all":tokenized_datasets.data.to_pandas(),
        "test":test.data.to_pandas()
    }[_df], "label")

for i, name in int2str.items():
    print(i, name, a[a.C == name].size)

"""## Обучение"""

model = AutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=len(categories))

training_args = TrainingArguments(
    output_dir="test_trainer",
    evaluation_strategy="epoch",
    num_train_epochs=3,
    logging_steps=len(train) // (5 * 8)
)
f1 = load_metric("f1")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    f_micro = f1.compute(predictions=predictions, references=labels,
average='micro')
    f_macro = f1.compute(predictions=predictions, references=labels,
average='macro')
    f_weighted = f1.compute(predictions=predictions, references=labels,
average='weighted')
    f_each = f1.compute(predictions=predictions, references=labels,
average=None)
    return {
        "accuracy | f1_micro": f_micro["f1"],
        "f1_macro": f_macro["f1"],
        "f1_weighted": f_weighted["f1"],
        "f1_each": [f"{int2str[i]}: {f_i:.2f}" for i, f_i in
enumerate(f_each["f1"])]
    }

class PrinterCallback(TrainerCallback):
    training_loss = []
    epoch = []
    validation_loss = []
```

Продолжение Листинга П.1

```
    eval_f1_macro = []
    eval_f1_weighted = []
    #def on_epoch_end(self, args, state, control, **kwargs):
    #    control.should_log = True
    def on_log(self, args, state, control, logs=None, **kwargs):
        #_ = logs.pop("total_flos", None)
        #if state.is_local_process_zero:
        #    print(logs)
        print(logs)
        if "loss" in logs:
            self.training_loss.append(logs["loss"])
            self.epoch.append(logs["epoch"])
        if "eval_loss" in logs:
            self.validation_loss.append(logs["eval_loss"])
            self.eval_f1_macro.append(logs["eval_f1_macro"])
            self.eval_f1_weighted.append(logs["eval_f1_weighted"])

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train,
    eval_dataset=valid,
    compute_metrics=compute_metrics,
    callbacks=[PrinterCallback]
)

trainer.train()

print(PrinterCallback.training_loss)
print(PrinterCallback.epoch)
print(PrinterCallback.validation_loss)
print(PrinterCallback.eval_f1_macro)
print(PrinterCallback.eval_f1_weighted)

import matplotlib.pyplot as plt

fig, ax = plt.subplots()
plt.xlabel("Epoch", fontsize=14)
plt.ylabel("Loss", fontsize=14)
ax.plot(PrinterCallback.epoch, PrinterCallback.training_loss, marker='o')
ax.plot(range(1, 3+1), PrinterCallback.validation_loss, marker='o')
ax.legend(["train", "valid"])
None

fig, ax = plt.subplots()
plt.xlabel("Epoch", fontsize=14)
plt.ylabel("Loss", fontsize=14)
ax.plot(range(1, 3+1), PrinterCallback.eval_f1_macro, marker='o')
ax.plot(range(1, 3+1), PrinterCallback.eval_f1_weighted, marker='o')
ax.legend(["macro", "weighted"])
None

categories

y_true = test["label"]

pred_output = trainer.predict(test)
y_pred = pred_output.predictions.argmax(-1)
```

Продолжение Листинга П.1

```
import seaborn
from sklearn.metrics import confusion_matrix

def draw_matrix(matrix, labels, mask=None, line=0., cmap="inferno",
line_color="black"):
    vmax = np.max(matrix)
    if vmax < 1:
        vmax = 1
    plt.figure(figsize=(12, 9))
    ax = seaborn.heatmap(
        matrix, annot=True, vmin=0, vmax=1, mask=mask, fmt='.5g',
        cmap=cmap, linewidths=line, linecolor=line_color, cbar=False
    )
    ax.set_yticklabels(
        labels, fontsize=12, rotation=0, verticalalignment='center'
    )
    ax.set_xticklabels(
        labels, fontsize=12, rotation=90, horizontalalignment='center'
    )
    plt.show()

def smart_truncate(content, length=15, suffix="..."):
    if len(content) <= length:
        return content
    return ' '.join(content[:length+1].split(' ')[0:-1]) + suffix

matrix = confusion_matrix(y_true, y_pred, labels=list(int2str.keys()),
normalize="true")
matrix = np.around(matrix, decimals=2)
draw_matrix(matrix, [smart_truncate(c) for c in categories], cmap="summer")

path_to_saved_models = "/content/drive/My Drive/diploma"

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

!mkdir "$path_to_saved_models"

from datetime import datetime
now = datetime.now().strftime("%y-%m-%d_%H:%M:%S")

model_path = f"{path_to_saved_models}/model_{now}"
model_path

model.save_pretrained(model_path)

import os
import glob

def get_last_file(folder):
    list_of_files = glob.glob(f"{folder}/*")
    latest_file = max(list_of_files, key=os.path.getmtime)
    return latest_file

last_model_path = get_last_file(path_to_saved_models)
last_model_path
```

Окончание Листинга П.1

```
last_model =
AutoModelForSequenceClassification.from_pretrained(last_model_path)

test_trainer = Trainer(
    model=last_model,
    compute_metrics=compute_metrics
)

pred_output = test_trainer.predict(test)
pred_output

import torch
from torch import LongTensor

def predict(model, item):
    output = model(
        input_ids=LongTensor(item["input_ids"]).unsqueeze(0).to("cuda"),

attention_mask=LongTensor(item["attention_mask"]).unsqueeze(0).to("cuda")
    )
    logits = output["logits"]
    label = logits[0].softmax(0).argmax().item()
    return int2str[label]

example = test[11]
print(example["text"])
print(predict(last_model, example))

@interact(Текст="")
def custom(Текст):
    text_dict = tokenizer(Текст)
    return predict(last_model, text_dict)
```