РЕФЕРАТ

Выпускная квалификационная работа содержит 58 страниц, 0 рисунков, 0 таблиц, 27 использованных источника.

СЕНТИМЕНТ-АНАЛИЗ, АНАЛИЗ ТОНАЛЬНОСТИ, КЛАССИФИКАЦИЯ ТЕКСТОВ, МАШИННОЕ ОБУЧЕНИЕ, РАСПРЕДЕЛЕННЫЕ ПРЕДСТАВЛЕНИЯ СЛОВ, КРАУДСОРСИНГ, НАБОР ДАННЫХ

Данная работа посвящена изучению сентимент-анализа русскоязычных художественных текстов с помощью методов распределенного представления слов и машинного обучения. В ходе исследования был сформирован набор данных для обучения получившихся моделей.

Теоретическая часть работы состоит из обзора и анализа существующих исследований в этой области, обоснования математического аппарата применяемых методов, основных подходов в предобработке русскоязычных текстов и постановки задачи сентимент-анализа.

В практической части разобрана проблема формирования качественного набора данных, реализация и применение математически обоснованных моделей классификации на основе алгоритмов машинного обучения в связке с моделями распределенных представлений слов, а также анализ полученных результатов и итоги исследования.

Данная работа представлена на 19-й международной конференции «Авиация и космонавтика» [1] и на международной молодежной научной конференции XLVII «Гагаринские чтения» [2].

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **4**

ОСНОВНАЯ ЧАСТЬ **5**

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 6

1.1 Обзор предметной области 6

1.2 Задача классификации и сентимент-анализа текстов ... 9

1.3 Предобработка текстов 9

1.4 Представление слов 12

1.5 Классификация 23

2 ПРАКТИЧЕСКАЯ ЧАСТЬ 32

2.1 Используемые инструменты 32

2.2 Сбор данных 32

2.3 Модели классификации 37

2.4 Архитектура моделей классификации 39

2.5 Эксперименты 40

ЗАКЛЮЧЕНИЕ **41**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **42**

ПРИЛОЖЕНИЯ **45**

ПРИЛОЖЕНИЕ 1 46

ПРИЛОЖЕНИЕ 2 52

ВВЕДЕНИЕ

Автоматическая классификация текстов является важной задачей обработки естественного языка. Эта работа посвящена одному из приложений классификации — автоматическому определению эмоциональной окраски русскоязычных художественных текстов. Главная особенность заключается в том, что предсказание базируется на эмоциональных моделях, предложенных Робертом Плутчиком и Полом Экманом.

Мультиклассификатор может интегрироваться в IoT и другие интеллектуальные устройства, чтобы эти устройства могли действовать на основе обнаруженных эмоциональных состояний пользователей. Может быть использован как интеллектуальный ассистент, например, во время психологических консультаций, чтобы лучше отслеживать и понимать состояние пациента и помогать врачам более эффективно оказывать поддержку. Или как инструмент для исследования как собственных эмоций, так и эмоций окружающих.

Основными задачами является:

- изучение существующих работ в данной области;
- формирования набора размеченных данных, содержащих эмотивную лексику;
- математическое обоснование моделей классификации и их реализация на языке программирования Python;
- анализ полученных результатов.

Подавляющее большинство работ на эту тему сводятся к бинарой классификации, модели способны определять только два класса: «положительный» и «отрицательный» или рассматривается похожая задача, но регрессии, в ней появляется промежуточные классы. И почти не существует моделей мультиклассификации по эмоциональным моделям. Основная проблема заключается в отсутствии данных для исследований. В этой работе описан процесс сбора необходимых данных и оценка качества их классификации.

ОСНОВНАЯ ЧАСТЬ

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Обзор предметной области

Классификация текстов — невероятно популярная задача. Мы пользуемся текстовыми классификаторами в почтовом клиенте: он классифицирует письма и фильтрует спам. Другие приложения включают классификацию документов, обзоров и так далее.

Обычно классификация текстов используется не как самостоятельная задача, а является частью более крупного пайплайна. Например, голосовой помощник классифицирует ваше высказывание, чтобы понять, что вы хотите или чувствуете и передать ваше сообщение другой модели в зависимости от решения классификатора. Другой пример — поисковый движок: он может использовать классификаторы для определения языка запроса, или предсказать его тип (например, развлекательный, информационный, навигационный, транзакционный) и понять хотите ли вы увидеть картинки или видео помимо документов или подобрать контент по настроению.

Задачи интеллектуальной обработки текста делятся на три типа: синтаксические, основанные на понимании смысла и третий — не просто понимание написанного, а еще и генерация нового текста. Сентимент-анализ относится ко второму типу.

Сентимент-анализ, как направление компьютерной лингвистики, стал очень популярен последние десятилетие. С появлением больших данных насыщенных эмоциональной составляющей возникла потребность в их обработке. Компании начали устраивать соревнования с внушительными призовыми фондами, а исследователи искать лучшую архитектуру для классификации этих данных. Так в открытом доступе появились большие размеченные датасететы с отзывами, данными из соцсетей и новостями.

Сам термин «сентимент» означает совокупность чувств и взглядов, как основа для действия или суждения; общая эмоциональная установка.

Целью сентимент-анализа является выделении этих тональных компонент из текста. Рассмотрим его применение на разных уровнях [27].

Пусть есть целый документ, тогда, как правило, в нем можно выделить один субъект и один объект, а так же сентимент. Ярким примером такого уровня является отзыв. Здесь автор выступает в качестве субъекта, а предмет отзыва — в качестве объекта. Это уровень документа.

Если документ более сложный, то можно рассматривать его на уровне предложений. В результате можно определить эмоциональную окраску всего документа или предложений по отдельности. Зависит от поставленной задачи.

Также анализ бывает на уровне аспектов. Смысл его в том, что эмоциональная установка определяется не для конкретно объекта, а для его отдельных составляющих — аспектов. Например, для объекта «компьютер» можно выделить аспекты «производительность», «дизайн», «сборка» т.д., другими словами, к аспектам относится все то, к чему могут быть выражены эмоции. Данная задача очень востребована, потому что позволяет точнее определять отношение автора к объекту, а для некоторых задач это очень важный критерий.

Последний вид анализа самый сложный и проводится уровне именованных сущностей (Named Entities). Именованная сущность — это абстрактный или физический объект, который может быть обозначен собственным именем. Сама по себе задача извлечения именованных сущностей (Named Entity Recognition) не из простых, а вкупе с сентимент-анализом становится действительно комплексным решением.

Методы сентимент анализа можно также разделить на несколько основных направлений [27]:

- *метод основанный на правилах (rule-based).* Здесь используются наборы правил классификации, составленные экспертом и эмоционально размеченные словари. Определенный класс присваивается в зависимости от найденных ключевых слов и их использования с другими

ключевыми словами. Такой метод достаточно эффективный, но очень трудоемкий. Неплохих результатов в бинарной классификации добились в работе [6];

- *метод основанный на словарях.* Самый простой метод, основанный на подсчете сентиментных единиц содержащихся в словарях тональности. Очень сильно зависит от размера словаря и не очень точен в разрыве с правилами русского языка. Попытка получения сентимента из текста таким способом описана в этой работе [3];
- *методы основанные на машинном и глубоком обучении.* (machine learning, deep learning) Наиболее популярная группа методов в сентимент-анализе, работающих на способности алгоритмов обобщать выделенные из текста признаки. Их применение позволило получить очень высокие результаты в работах [23, 12, 8, 15];
- *гибридные методы.* Позволяют одновременно использовать несколько подходов. что-нибудь

Разберем подробнее машинное и глубокое обучение. Суть метода в выделении признаков из текста и последующем их обобщении с помощью разнообразных алгоритмов. Для выделения признаков используют, как простые алготитмы по типу мешок слов (Bag of Words) или TF-IDF, так и небольшие нейронные сети для генерирования эмбеддингов, например, Word2Vec [16], GloVe [25], FastText [7]. Существуют и более сложные алгоритмы, которые формируют признаки на уровне предложений, к ним относятся ELMo [13, 19], BERT (Bidirectional Encoder Representations from Transformers) [11] и др.

Чтобы обработать выделенные признаки используют разнообразные алгоритмы. К классическому обучению относятся:

- байессовский классификатор (Naive Bayes classifier);
- дерево решений (Decision Tree);
- логистическая регрессия (Logistic Regression);
- метод опорных векторов (Support Vector Machine).

Среди алгоритмов глубокого обучения можно выделить:

- реккурентные нейронные сети (RNN);
- сверточные нейронные сети (CNN);
- полносвязные нейронные сети (FCNN) и т.д.

1.2 Задача классификации и сентимент-анализа текстов

Что такое естественный язык вообще, и как с ним работают?

- множество допустимых цепочек символов из некоторого алфавита;
- цепочки строятся по некоторым правилам;
- текст — одна такая цепочка
- алфавит — множество символов, из которых строятся тексты языка
- далеко не каждая цепочка несет какую-то информацию

Анализ языка сводится к двум глобальным задачам:

- обучение: понять правила языка;
- применение: для некоторого текста понять по каким правилам он построен.

В этой работе задача классификации сформулирована следующим образом. Пусть есть описание документа $d \in X$, где X — векторное пространство документов и фиксированный набор меток $C = \{c_1, c_2, \ldots, c_n\}$. Из обучающей выборки (множества документов с заранее известными метками — эмоциями) $D = \{\langle d, c\rangle | \langle d, c\rangle \ \in X \times C\}$ и с помощью метода обучения G необходимо получить классифицирующую функцию $G(D) = f$, которая отображает документы в пространство меток $f : X \to C$.

1.3 Предобработка текстов

Представим, что у нас есть набор данных с текстами на русском языке и соответствующие им метки. Для автоматической обработки текста в

рассматриваемой задаче удобно было бы представить слова в начальной форме. Для этого применяется лемматизация текста.

Определение 1.1. Лемматизация — процесс приведения словоформы к лемме — нормальной (словарной) форме.

Квартира простояла пустой и запечатанной только неделю. → квартира простаивать пустой и запечатывать только неделя

Как правило текст хорошо описывает словарь из которого состоит набор данных, и тем как часто они встречаются в тексте, а не структурой фразы. Проведем небольшой эксперимент: возьмем набор данных, который рассматривается в этой работе, посчитаем для каждого слова в этом наборе частоту его встречаемости и отсортируем полученный список по убыванию частот. Получили график распределения Ципфа рис. 1.1 — распределения вероятностей, описывающего взаимоотношения частоты события и количества событий с такой частотой.

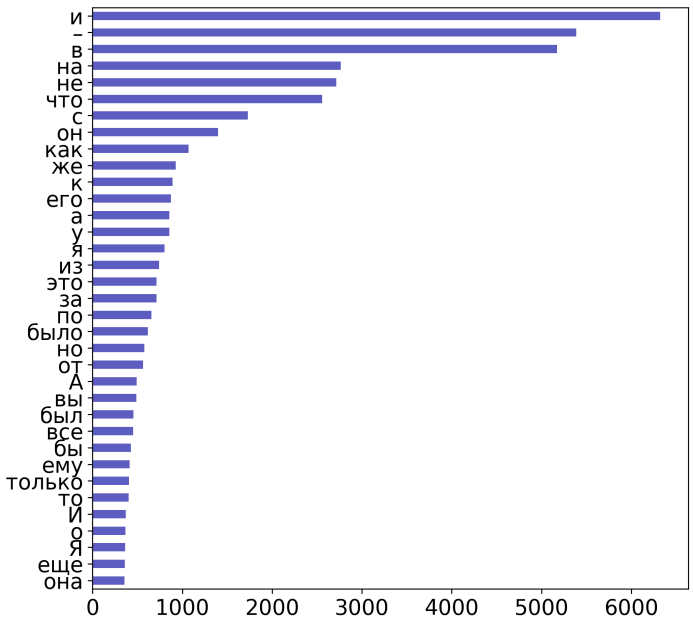

Рис. 1.1. График частоты встречаемости слов и символов в наборе данных

Это распределение представляет собой степенную функцию:

$$f(rank : s, N) = \frac{1}{Z(s, N)rank^s},$$

где $rank$ — порядковый номер слова после сортровки по убыванию частоты, s — коэффициент скорости убывания вероятности, N — количество слов и $Z(s, N) = \sum_(i = 1)^N i^{-s}$ — нормализованная константа.

Из этого графика можно сделать три основных вывода:

- частотных слов мало и они не информативны;
- редких слов много, они информативны, но на них сложно опираться;
- нужен баланс частотности и информативности.

Самые частотные и не значимые слова называются «стоп-словами». Для русского языка определены словари стоп-слов, программная реализация есть в библиотеке «nltk». Обработка предыдущего предложения дает такой результат:

квартира простаивать пустой и запечатывать только неделя → квартира простаивать пустой запечатывать неделя

На рис. 1.2 представлено распределение частотности слов после лемматизации и удаления стоп-слов.

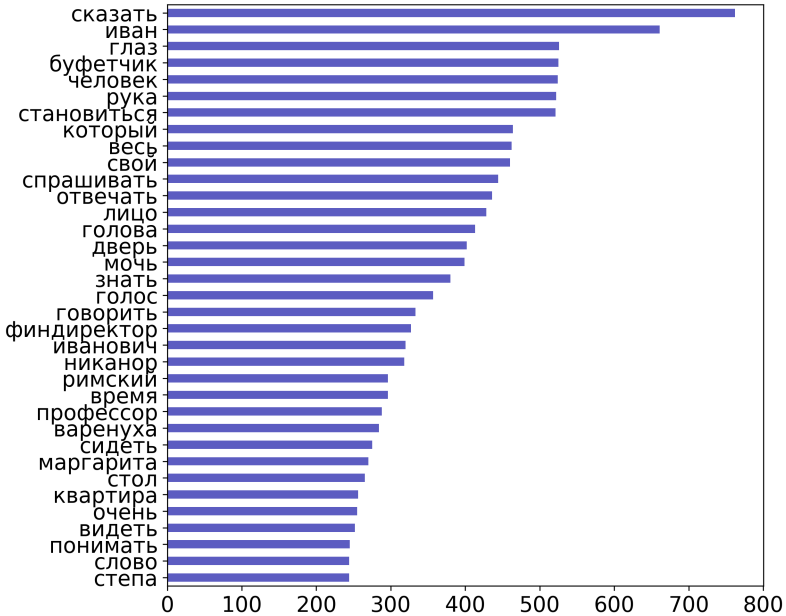

Рис. 1.2. График частоты встречаемости слов в наборе данных после обработки

1.4 Представление слов

То, как модели видят данные отличается от того, как из видят люди. Например, мы легко можем понять предложение «Да будет свет», но модели не могут — им нужны векторы с признаками. Такие векторы являются представлениями слов, которые может обработать наша модель.

Самой простой формой представления слов является дискретное представление, т.е. one-hot представление: все слова представляются в виде вектора, размерность которого совпадает с числом слов словаре. Причем все компоненты кроме i-го равны нулю, а позиция, соответствующая i-му слову равна единице. Очевидно, что такой способ не самый лучший. Во-первых такое представление зависит от положения слов в словаре, а это нежелательно, потому что задает бессмысленные отношения между словами. Во-вторых размеры такого словаря растут прямо пропорционально количеству слов в нем, а это значит размерность его может доходить до сотен тысяч и работать с ними станет очень вычислительно накладно. В-третьих такое представление совершенно не учитывает значение слов, для решения этой проблемы обратимся к дистрибутивной семантике.

1.4.1 Дистрибутивная семантика

Чтобы зафиксировать значение слов в их векторах, нам сначала нужно определить понятие значения, которое можно использовать на практике. Для этого давайте попробуем понять, как мы, люди, узнаем, какие слова имеют схожее значение [22].

Как только мы видим, как неизвестное слово используется в разных контекстах, мы в состоянии понять его значение. Гипотеза состоит в том, что мозг искал другие слова, которые можно использовать в тех же контекстах, нашел некоторые и пришел к выводу, что неизвестное слово имеет значение, подобное этим другим словам. Это гипотеза распределения:

Утверждение 1.1. Слова, которые часто встречаются в схожих контекстах, имеют одинаковое значение.

Это чрезвычайно ценная идея, ее можно использовать на практике, чтобы слова-векторы передавали их значение. Согласно гипотезе распределения, «улавливать смысл» и «улавливать контексты» по своей сути одно и то же. Следовательно, все, что нам нужно сделать, это поместить информацию о контекстах слов в представление слов. В этой части работы мы разберем два способа, как сделать это.

1.4.2 Основная идея архитектуры нейронных сетей

Чтобы объяснить работу неронных сетей нужно опредить из чего они состоят. Для этого рассмотрим простейший линейный бинарный классификатор – перцептрон Розенблата. Пространство данных разделяется на два множества гиперплоскостью, а метка класса будет ставиться в зависимости от значения линейной функции от входных признаков.

$$\text{sign}(w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d), \tag{1.1}$$

где $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$. Мы ищем такие веса $w_0, w_1 \ldots, w_d \in \mathbb{R}^d$, чтобы sign от скалярного произведения признаков и весов $w^\top x$ совпадал с верной меткой $y(x) \in -1, 1$, но для этого добавим фиктивную переменную в вектор $x = (1, x_1, x_2, \ldots, x_d) \in \mathbb{R}^{d+1}$, чтобы размерности сохранялись.

Теперь обучим эту функцию, для этого нам нужна функция ошибки, она называется критерий Перцептрона:

$$L_P(w) = -\sum_{x \in M} y(x)(w^\top x), \tag{1.2}$$

где M — множество неверно классифицируемых примеров. В качестве оптимизатора выберем градиентный спуск. С помощью него мы минимизируем суммарное отклонение предсказаний классификатора от верных, но только в неправильную сторону. Верное предсказание никак не влияет на функцию ошибки. В результате получается кусочно-линейная функция,

которая почти везде дифференцируема и этого достаточно для применения градиентного спуска. Процесс обучение выглядит так: если предсказание верное, то не делаем ничего, если классификатор ошибся, то делаем градиентный шаг.

Такая модель перцептрона линейная и результат ее работы не слишком содержателен. Чтобы из перцепртонов можно было составить сеть, нужно добавить нелинейность. Такой нелинейностью будет функция активации. Они бывают разные, самая распространенная — сигмоида рис. 1.3, она позволяет моделировать вероятность:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.3}$$

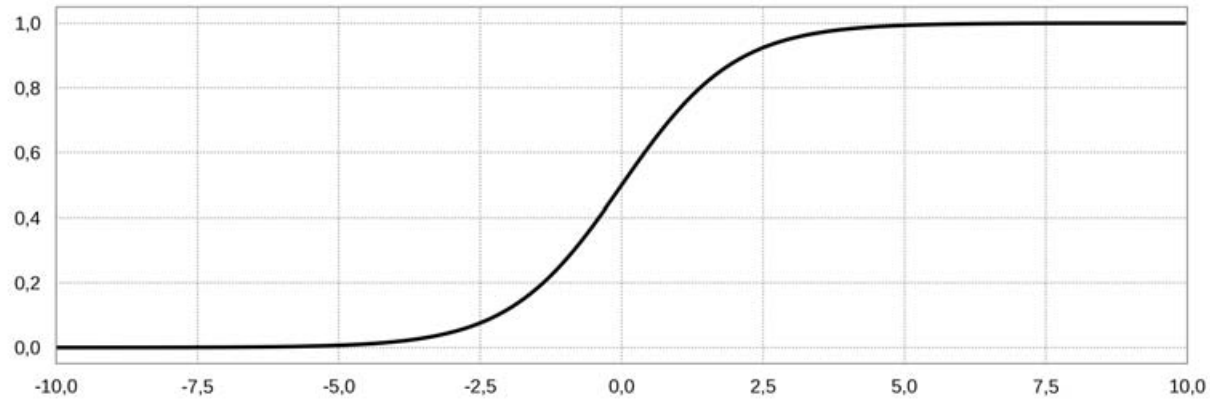

Рис. 1.3. Сигмоида

Обучить этот прецептрон также не составляет труда. Просто теперь мы будем решать задачу бинарной классификации, а функция ошибки будет cross-энтропией:

$$L(w) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log \sigma(w^\top x_i) + (1 - y_i)\log(1 - \sigma(w^\top x_i))). \tag{1.4}$$

Эта функция дифференцируема, значит мы можем сделать градиентный шаг. При этом прецептрон с сигмоидой реализует логистическую регрессию. Обобщение этой модели можно найти в разделе 1.5.3.

Графическое изображение структуры перцепрона представлено на рис. 1.4, *а*.

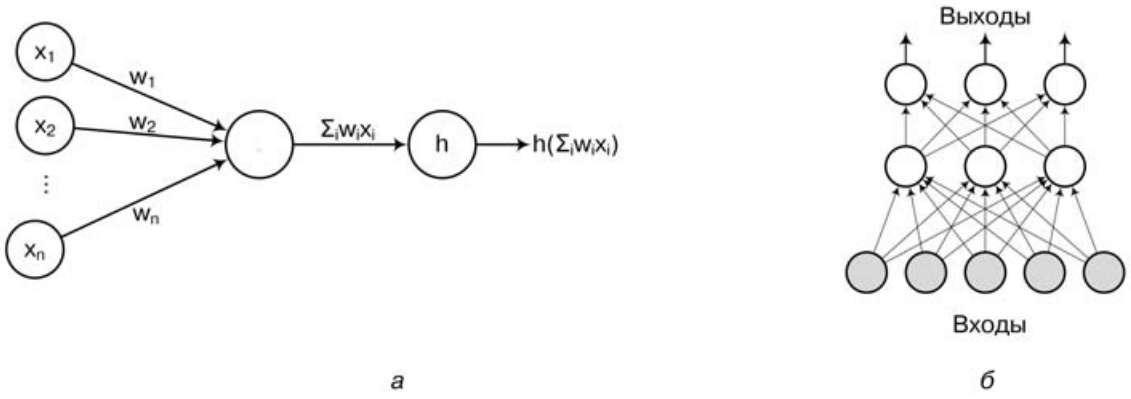

Рис. 1.4. (а) граф вычислений перцептрона; (б) полносвязная нейронная сеть с одним скрытым слоем.

На рис. 1.4, *б* изображена несложная нейронная сеть, на ее примере покажем как можно векторизовать вычисления в слое нейронов с применением функции активации.

Пусть у нас в слое k нейронов с весами $w_1, w_2, \ldots, w_k, \quad w_i = (w_{i1}, \ldots, w_{in})^\top$, на вход вектор $x = (x_1, x_2, \ldots, x_n)^\top$. В результате получим выход $y_i = f(w_i^\top x)$, где f — функция активации. Эти вычисления можно представить в векторной форме:

$$\begin{pmatrix} y_1 \\ \ldots \\ y_k \end{pmatrix} = y = f(Wx) = \begin{pmatrix} f(w_1^\top x) \\ \ldots \\ f(w_k^\top x) \end{pmatrix}, \text{ где } W = \begin{pmatrix} w_1 \\ \ldots \\ w_k \end{pmatrix} = \begin{pmatrix} w_{11} & \ldots & w_{1n} \\ \ldots & & \ldots \\ w_{kk} & \ldots & w_{kn} \end{pmatrix}.$$

Обычно в нелилинейных неронах примемяется сигмоида (1.3), о которой писалось выше. Но существуют и другие функции активации, например, SoftMax или нормализованная экспонента. Она нужна чтобы обобщить функцию ошибки для задач классификации, где классов больше 2-х. Ее удобно использовать на последних слоях нейронной сети, чтобы моделировать из выхода вероятность.

1.4.3 Распределенные представления слов word2vec

Подход к обучению моделей распределенных представлений слов был описан в работе Йошуа Бенджи с соавторами [9], которая была продолже-

на в [5]. Идея подхода описанного в [9] основанна на задаче построения языковой модели, процесс обучения выглядит так:

- всем токенам из словаря $i \in V$ ставят в соответствие вектор признаков w_i размерности d ($w_i \in \mathbb{R}^d$). Стандартным значением d является 300;
- теперь можно определить вероятности для каждого токена i, что он появится в контексте $c_1, \ldots, c_n$. Для этого определим функцию от векторов признаков w:

$$\hat{p}(i|c_1, \ldots, c_n) = f(w_i, w_{c_1}, \ldots, w_{c_n}; \theta), \tag{1.5}$$

  где $w_{c_1}, \ldots, w_{c_n}$ — векторы признаков токенов из контекста, а f — фунция с параметрами θ, которая принимает векторы признаков;
- максимизируя логарифм правдоподобия большого корпуса текстов можно обучить векторы признаков w и параметры θ

$$L(W, \theta) = \frac{1}{K} \sum_t \log f(w_k, w_{k-1}, \ldots, w_{k-n+1}; \theta) + R(W, \theta), \tag{1.6}$$

  где K размер окна контекста, а $R(W, \theta)$ — регуляризация.

Для получения функции f можно использовать нейронную сеть. Модель word2vec строится на описании нейросетевой модели, предложенной в [9]. Она была разработана Томасом Миколовым с соавторами и опубликована в работах [16, 14], причем в двух вариациях:

- CBOW (Continious Bag Of Words) — по контексту восстановить слово;
- skip-gram — восстановить контекст в зависимости от слова;

Архитектура word2vec представляет собой полносвязную нейронную сеть с одним скрытым слоем рис. 1.5.

Принцип работы модели CBOW [4] рис. 1.5, *а* выглядит так:

- на вход сети подаются one-hot вектора размерности V, где V — это размер словаря;

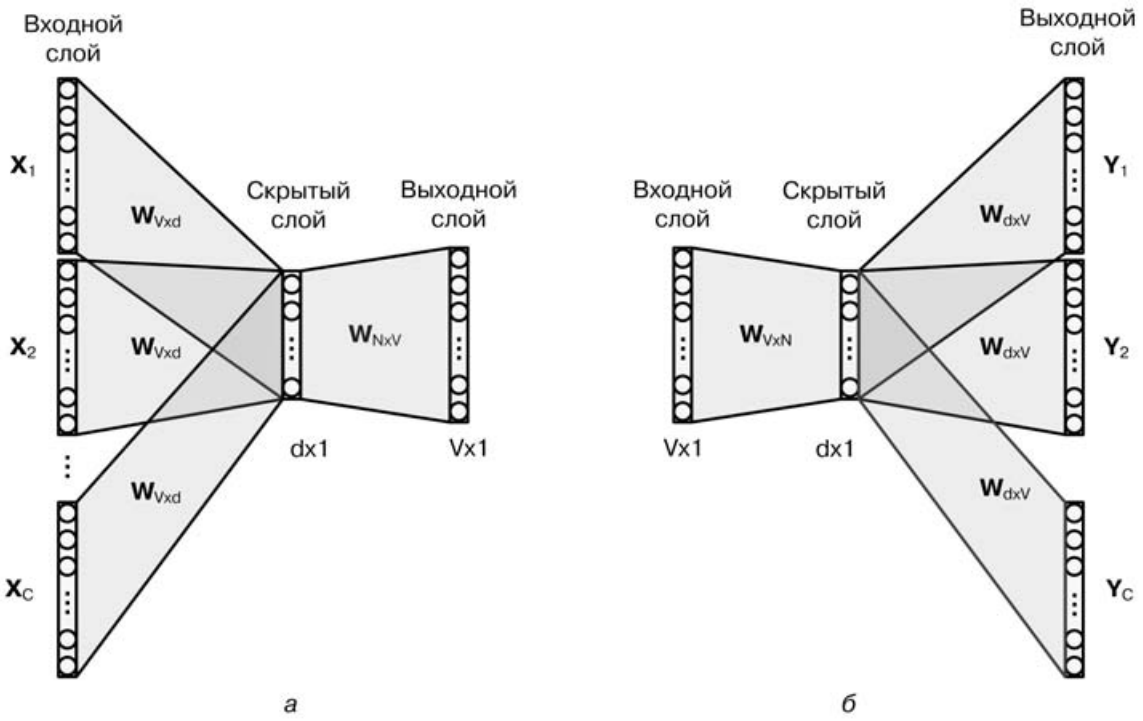

Рис. 1.5. (а) CBOW; (б) skip-gram.

- скрытый слой — это матрица W размерности $V \times d$, которая переводит наши представления слов в d - мерное пространство;
- на выходе для каждого слова в словаре берем среднее всех полученных векторов и получаем оценку u_j, где $j = 1, \ldots, V$.

Чтобы найти апострериорное распределение модели, просто вычисляем softmax:

$$\hat{p}(i|c_1, \ldots, c_n) = \frac{\exp u_j}{\sum_{j'=1}^{V} \exp u_{j'}}. \tag{1.7}$$

Для аппроксимации апостериорным распределением распределения данных используем loss-функцию для одного окна:

$$L = -\log p(i|c_1, \ldots, c_n) = -u_j + \log \sum_{j'=1}^{V} \exp u_{j'}. \tag{1.8}$$

Принцип работы модели skip-gram рис. 1.5, *б* полностью противоположный. До этого мы усредняли контекст, чтобы получить среднее слово в окне, а теперь будем предсказывать слова контекста исходя из центрального слова. На выходе мы получаем $K-1$ мультиномиальных распределений

(центральное слово не учитывается):

$$\hat{p}(c_k|i) = \frac{\exp u_{kc_k}}{\sum_{j'=1}^{V} \exp u_{j'}}, \tag{1.9}$$

loss-функция для окна размера K выглядит так:

$$L = -\log p(c_1, \ldots, c_n|i) = -\sum_{k=1}^{K} u_{kc_k} + n \log \sum_{j'=1}^{V} \exp u_{j'}. \tag{1.10}$$

Возникает вопрос, как же обучить такую модель? Этот процесс хорошо описан в докладе Голдберга и его соавторов [21].

Подробно разберем модель skip-gram для корпуса документов D. Нашей задачей стоит нахождение оптимальных параметров модели θ, чтобы максимизировать функцию правдоподобия:

$$L(\theta) = \prod_{i \in D} \left(\prod_{c \in C(i)} p(c|i;\theta) \right) = \prod_{(i,c) \in D} p(c|i;\theta), \tag{1.11}$$

где $C(i)$ — множество контекстных слов внутри окна вокруг центрального слова i. Вероятность $p(c|i;\theta)$ определяется, как softmax-функция, зависящая от всех возможных векторов контекста.

$$p(c|i;\theta) = \frac{\exp \tilde{w}_c^\top w_i}{\sum_{c'} \exp \tilde{w}_{c'}^\top w_i}, \tag{1.12}$$

где $\tilde{w}_c$ — вектор признаков слова из контекста c, который отличается от w_i. Для каждого слова i надо обучить два вектора признаков w_i и $\tilde{w}_i$, в первом случае это слова выступает в качестве центрального, во втором в качестве контекстного.

Эта особенность обучения, когда мы берем два разных вектора одного и того же слова вместо одного, описана в [21]. И мотивирована тем, что слова редко встречаются в контексте себя самих. Вот, например, слово «мотивация» вряд ли можно встретить в контексте другого слова «мотивация», под это правило попадают почти все слова. Поэтому в процессе обучения модель сведет вероятности $p(i|i,\theta)$ к нулю. А если вектора контекста и

центрального слова будут равны нулю, то норма вектора $|w_i| = w_i^\top w_i$ тоже будет равняться нулю, а это очень не желательно. Поэтому для каждого слова мы обучаем два разных вектора.

Теперь выразим максимум функции правдоподобия для всего корпуса через логарифм (1.11) и (1.12):

$$\begin{aligned} \arg\max_\theta \prod_{(i,c)\in D} p(c|i;\theta) = \arg\max_\theta \sum_{(i,c)\in D} \log p(c|i;\theta) = \\ = \arg\max_\theta \sum_{(i,c)\in D} \left(\exp \tilde{w}_c^\top w_i - \log \sum_{c'} \exp \tilde{w}_{c'}^\top w_i \right). \end{aligned} \tag{1.13}$$

Оптимизируя данную функцию мы получаем хорошее распределенное представление слов. Но для этого нужно решить сложнейшую задачу: суммировать скалярные произведения всех возможных слов и их контекста $\sum_{c'} \tilde{w}_c^\top w_i$ при том, что размер словаря может достигать миллионов.

Чтобы уменьшить количество вычислений Миколов с соавторами [14] предложили элегантный метод: negative sampling. Нам не нужно считать всю сумму $\sum_{c'} \tilde{w}_c^\top w_i$, а только случайно выбрать несколько ее элементов в качестве отрицательных примеров (примеры в которых слово не находится в определенном контексте) и обновить только их. Т.е. теперь нам нужно посчитать только небольшую сумму $\sum_{c'\in D'} \tilde{w}_c^\top w_i$, где D' — случайное подмножество отрицательных примеров.

По сути negative sampling — это тоже правдоподобие, но другого события. Пусть у нас есть слово i и его контекст c, наша задача максимизировать вероятность $p((i,c) \in D;\theta)$, параметризированную вектором θ, т.е. правдоподобие появления пары (i,c):

$$\arg\max_\theta \prod_{(i,c)\in D} p((i,c) \in D;\theta) = \arg\max_\theta \sum_{(i,c)\in D} \log p((i,c) \in D;\theta). \tag{1.14}$$

Выразим $p((i,c) \in D;\theta)$ через softmax. Но так как это бинарное событие, то заменим softmax сигмоидой $\sigma(x) = \frac{1}{1+\exp(-x)}$:

$$p((i,c) \in D; \theta) = \frac{1}{1 + \exp\left(-\tilde{w}_c^\top w_i\right)} \tag{1.15}$$

Максимизируем логарифм правдоподобия:

$$\begin{aligned} \arg\max_\theta \sum_{(i,c)\in D} \log p((i,c) \in D; \theta) = \\ = \arg\max_\theta \sum_{(i,c)\in D} \log \frac{1}{1 + \exp\left(-\tilde{w}_c^\top w_i\right)}. \end{aligned} \tag{1.16}$$

Из (1.16) видно, что оптимальное значение логарифма будет получено при максимальном значении скалярного произведения $\tilde{w}_c^\top w_i$. Сделаем равные векторы с большой нормой и можно без проблем получить правдоподобие почти равное единице. Подвох заключается в том, что модель обучается на данных для бинарной классификации, но мы рассматриваем только набор состоящий из положительных примеров. Классификатор, который всегда предсказывает «да» — плохой. Поэтому имеет смысл добавить отрицательных примеров, просто случайно выбирая слова и контекст, которых нет в данных. После того, как мы получим набор отрицательных данных, максимизация правдоподобия будет выглядеть так:

$$\arg\max_\theta \prod_{(i,c)\in D} p((i,c) \in D; \theta) \prod_{(i,c)\in D'} p((i',c') \notin D; \theta) \tag{1.17}$$

Выразим в (1.17) пару $(i,c) \in D$:

$$\begin{aligned} &\arg\max_\theta \prod_{(i,c)\in D} p((i,c) \in D; \theta) \prod_{(i,c)\in D'} 1 - p((i',c') \in D; \theta) = \\ &= \arg\max_\theta \left[\sum_{(i,c)\in D} \log p((i,c) \in D; \theta) + \sum_{(i,c)\in D'} \log\left(1 - p((i',c') \in D; \theta)\right) \right] = \\ &= \arg\max_\theta \sum_{(i,c)\in D} \left[\log \frac{1}{1 + \exp\left(-\tilde{w}_c^\top w_i\right)} + \sum_{(i,c')\in D'} \log \frac{1}{1 + \exp\left(\tilde{w}_{c'}^\top w_i\right)} \right] = \\ &= \arg\max_\theta \sum_{(i,c)\in D} \left[\log \sigma(\tilde{w}_c^\top w_i) + \sum_{(i,c')\in D'} \log \sigma(-\tilde{w}_{c'}^\top w_i) \right] \end{aligned} \tag{1.18}$$

Получили формулу для negative sampling из [14]. Значит мы для каждого окна случайно берем несколько отрицательных примеров D' и делаем градиентный шаг для loss-функции:

$$L = \log \sigma(\tilde{w}_c^\top w_i) \sum_{(i,c') \in D'} \log \sigma(-\tilde{w}_{c'}^\top w_i) \tag{1.19}$$

Аналогичные рассуждения можно провести для модели CBOW.

1.4.4 ELMo

Создание этой модели породило новую эру распределенных представлений слов. Word2vec показал, что с помощью векторов можно представлять слова в виде, который может передавать их семантику или смысловые отношения (т.е. способность различать схожие и противоположные по смыслу слова или находить параллели в отношениях таких словарных пар, как «Женщина - Мама» и «Транспорт - Повозка»), а также синтаксические или грамматические отношения (например, определять, что отношение между «Красиво и Красивый») [10].

Почему бы не сделать распределенное представление слов на основе контекста, в котором оно стоит? Чтобы передавать не только значение слова, но и контекстуальную информацию. Так появились контекстуализированные распределенные представления слов (contextualized word-embeddings)

Способность понимать язык ELMo получила после обучения предсказыванию нового слова в последовательности слов. Эта задача, называется языковым моделированием (language modeling).

В [20] показано, что двунаправленная языковая модель (biLM) является основой для ELMo. В то время как входные данные представляют собой последовательность n слов $(x_1, \ldots, x_n)$ языковая модель предсказывает условную вероятность следующего слова:

В прямом проходе последовательность содержит слова перед целевым словом:

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1}).$$

При обратном проходе последовательность содержит слова после целевого слова:

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i | x_{i+1}, \ldots, x_n).$$

Предсказания в обоих направлениях моделируются многослойными LSTM ячейками со скрытыми состояниями $\overrightarrow{h}_{i,l}$ и $\overleftarrow{h}_{i,l}$ и для входного слова x_i на уровне слоя $l = 1, \ldots, L$. Скрытое состояние последнего слоя $h_{i,L} = [\overrightarrow{h}_{i,L}; \overleftarrow{h}_{i,L}$ используется для вывода вероятностей по словам после нормализации softmax. Эти скрытые состояния разделяют слой распределенного представления слов и слой softmax, которые параметризованы Θ_e и Θ_s соответственно.

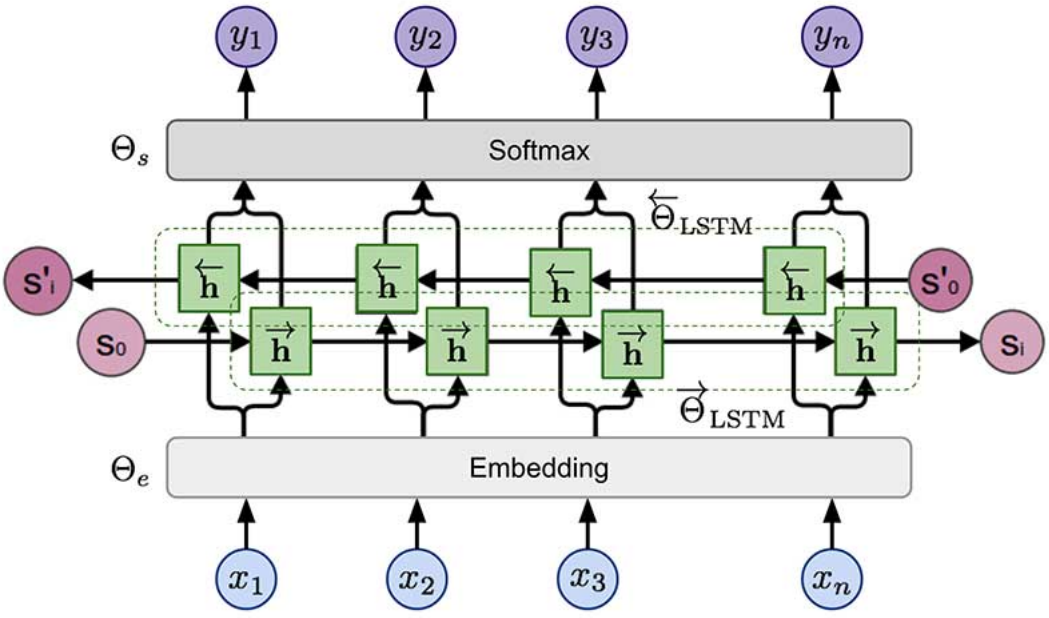

Рис. 1.6. Базовая модель biLSTM ELMo. Источник: [24]

Процесс обучения модели — максимизация логарифма правдоподобия для истиных слов в обоих направлениях:

$$\text{Likelihood}(x) = \arg\max \sum_{i=1}^{n} \Big[\log p(x_i|x_1, \ldots, x_{i-1}|\Theta_e, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \\ + \log p(x_i|x_{i+1}, \ldots, x_n|\Theta_e, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \Big]$$

1.5 Классификация

1.5.1 Анализ качества классификации

Для оценки качества классификации используется Macro F1-мера — нормализованное по всем классам среднее гарманическое метрик P и R:

$$\text{Macro F1} = \frac{1}{n} \sum_{i=1}^{n} 2\frac{P_i \times R_i}{P_i + R_i} = \frac{1}{n} \sum_{i=1}^{n} \text{F1}_i,$$

где P — точность и R — полнота.

Ее преймущество в том, что она не учитывает дисбаланс классов.

Определение 1.2. Точностсь (*precision*) — показывает долю объектов, которые классификатор определил, как принадлежащие действительно правильному классу.

$$P = \frac{TP}{TP + FP},$$

где TP — те метки которые мы ожидали и получили, FP — те метки которые мы не ожидали, но получили на выходе.

Определение 1.3. Полнота (recall) — показывает какую долю объектов из всего множества конкретного класса калассификатор определил верно.

$$R = \frac{TP}{TP + FN},$$

где FN — те метки которые мы ожидали, но не получили на выходе.

1.5.2 Кросс-валидация

Методом оценки модели и ее поведения на тестовых данных является стратифицированая перекрестная проверка stratified k-fold cross validation, при $K = 10$.

Пусть X — множество описаний документов, Y — множество возможных меток. Есть конечная выборка докуметов $X^L = (x_i, y_i)_{i=1}^L \subset X \times Y$. Задан алгоритм обучения — отображение f, которое произвольной конечной выборке X^m ставит в соответствие функцию — алгоритм $f : X \to Y$.

Качество алгоритма a оценивается по произвольной выборке документов X^m с помощью функционала качества $Q(f, X^m)$. Для перекрестной проверки не важно, как именно вычисляется этот функционал:

$$Q(f, X^m) = \frac{1}{m} \sum_{x_i \in X^m} L(f(x_i), y_i),$$

где $L(f(x_i), y_i)$ — неотрицательная функция потерь (loss), возвращающая величину ошибки классификатора $f(x_i)$ при правильном ответе y_i.

Выборка X^L разбивается K способами на две подвыборки: $X^L = X_k^m \cup X_k^n$, где X_n^m — обучающая подвыборка длины m, X_k^n — контрольная подвыборка длины $k = L - m$, $k = 1, \ldots, K$ — номер разбиения.

Для каждого разбиения K строится алгоритм $f_n = \mu(X_n^m)$ и вычисляется значение функционала качества $Q_k = Q(f_k, X_k^n)$. Нормализованное значение Q_k по всем разбиениям называется перекрестной проверкой:

$$CV(\mu, X^L) = \frac{1}{K} \sum_{k=1}^{K} Q(\mu(X_k^m), X_k^n).$$

Стратифицированая выборка позволяет сохранить процентное содержание документов для каждого класса.

1.5.3 Логистическая регрессия

Логистическая регрессия — классическая дискриминативной линейная модель классификации. Дискриминативная значит, что нас интересует $P(y = k|x)$, а не совместное распределение $p(x, y)$. Свое начало она берет из расстояния Кульбака-Лейблера. Оно задается формулой:

$$KL(P||Q) = \int \log \frac{dP}{dQ} dP, \tag{1.20}$$

, где P — истинное распределение, а Q — приближенное. Для дискретного случая:

$$KL(P||Q) = \sum_y p(y) log \frac{p(y)}{q(y)}, \tag{1.21}$$

а если раскрыть получаем:

$$\begin{aligned} KL(P||Q) &= \sum_y p(y) \log \frac{p(y)}{q(y)} = \\ &= \sum_y p(y) \log p(y) - \sum_y p(y) \log q(y) = -H(p) + H(p, q), \end{aligned} \tag{1.22}$$

, где $H(p)$ — энтропия распределения p, а $H(p, q)$ — наша кросс энтропия. Из этой суммы видно, что нам нужно минимизировать $H(p, q)$. Для бинарной классификации loss-функция будет выглядеть так:

$$L(w) = H(p_{data}, q(w)) = -\frac{1}{N} \sum_{i=1}^{N} (y_i \log \hat{y}_i(w) + (1 - y_i) \log(1 - \hat{y}_i(w))) \tag{1.23}$$

где p_{data} — распределение наших данных, $q(w)$ — апостериорное распределение, $\hat{y}_i(w)$ — оценка вероятности при входных параметрах w и y_i — истинное предсказание. Два слагаемых мы получаем, т.к. события несовместные. Например, в тексте говорится о кошечках или о собачках, события появления кошечки или собачки несовместные, т.е. $p(\text{кошечки}) =$

$1 - p$(собачки). Если мы предсказываем кошечку (1), как абсолютный 0 или собачку (0), как 1, то ошибка будет бесконечной из первого и второго слагаемых соответственно – это не допустимо.

Перед тем, как перейти к нескольким классам, рассмотрим сначала задачу классификации с Байейсовской точки зрения: определим для каждого класса C_k плотность $p(x|C_k)$ и какие-то априорные распределения $p(C_k)$ (пускай это будут размеры классов, т.е. мы ничего не знаем о примере, но предполагаем с какой вероятностью он относится к конкретному классу) и найдем $p(C_k|x)$. Для двух классов:

$$\begin{aligned} p(C_1|x) &= \frac{p(x|C_1)p(C_1)}{p(x|C_1)p(C_1) + p(x|C_2)p(C_2)} = \\ &= 1/\frac{(p(x|C_1)p(C_1) + p(x|C_2)p(C_2))}{p(x|C_1)p(C_1)} = \\ &= 1/(1 + \frac{p(x|C_2)p(C_2)}{p(x|C_1)p(C_1)}) = \frac{1}{1+e^{-a}} = \sigma(a), \end{aligned} \tag{1.24}$$

где

$$a = \ln \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)}, \qquad \frac{1}{1+e^{-a}} = \sigma(a). \tag{1.25}$$

Используя логистическую регрессию мы делаем предположение о виде аргумента сигдмоиды a — это будет скалярное произведение вектора признаков на вектор данных: $a = w_\top x$. Сигмоида переводит результат вычисления этой линейной функции на отрезок $[0; 1]$ и как результат мы получаем апостериорную вероятность первого или второго классов:

$$p(C_1|x) = y(x) = \sigma(w_\top x), \qquad p(C_2|x) = 1 - p(C_1|x), \tag{1.26}$$

чтобы обучить эту модель мы можем просто оптимизировать правдоподобие по w.

Для набора x_n, t_n, где x_n – входы, а $t_n \in \{0; 1\}$ – соответствующие метки классов, получается такое правдоподобие:

$$p(t|w) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{1-t_n}, \quad \text{где} \quad y_n = p(C_1|x_n). \tag{1.27}$$

И теперь мы, максимизируя логарифм вероятности, ищем наилучшие параметры функции правдоподобия для этого можно использовать различные оптимизаторы.

$$\text{Likelihood}(w) = -\ln p(t|w) = -\sum_{n=1}^{N}[t_n \ln y_n + (1-t_n)\ln(1-y_n)]. \quad (1.28)$$

Теперь можно легко обобщить задачу на несколько классов. Только вместо сигмоиды будем использовать softmax функцию. Для K классов получаем:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_{j=1}^{K} p(x|C_j)p(C_j)} = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}, \quad (1.29)$$

где количество аргументов $a_k = \ln p(x|C_k)p(C_k)$ равняется количеству классов. Функция правдоподобия почти не изменилась. Пусть на вход метки класса подаются в формате one-hot векторов, тогда для набора векторов $T = t_n$ функция правдоподобия выглядит следующим образом:

$$p(T|w_1, \ldots, w_K) = \prod_{n=1}^{N}\prod_{k=1}^{K} p(C_k|x_n)^{t_{nk}} = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_{nk}}. \quad (1.30)$$

, где $y_{nk} = y_k(x_n)$. Опять переходим к логарифму и получаем функцию максимального правдоподобия для K классов:

$$\text{Likelihood}(w_1, \ldots, w_K) = -\ln p(T|w_1, \ldots, w_K) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}. \quad (1.31)$$

1.5.4 Метод опорных векторов

Главная задача SVM сводится к нахождению разделяющей гиперплоскости в пространстве $\mathbb{R}^n$, то есть этот алгоритм решает задачу бинарной классификации, где объектам из $X \in \mathbb{R}^n$ ставится в соответствие один из двух классов $Y = \{+1, -1\}$.

Чтобы перейти к задаче многоклассовой классификации используется подход one-to-rest, суть что мы используем столько же SWM классификаторов, сколько классов. И классы просто сравниваются между собой.

Но вернемся к объяснению алгоритма. Чтобы найти такую гиперплоскость выборка должна быть линейно разделимой, что на практике встечается очень редко. Обычно данные содержат шум и нечеткие границы между классами. В таком пространстве стандартная задача нахождения гиперплоскости неразрешима. Обойти это ограничение нам поможет замысловатый трюк.

Пусть поставлена задача нелинейного программирования с ограничениями:

$$\begin{cases} f(x) \to \min\limits_{x \in X} \\ g_i(x) \leq 0,\ i = 1 \dots m \\ h_j(x) = 0,\ j = 1 \dots k \end{cases}$$

Если x — точка локального минимума при наложенных ограничениях, то существуют такие множители $\mu_i, i = 1 \dots m,\ \lambda_j, j = 1 \dots k$, что для функции Лагранжа $L(x; \mu, \lambda)$ выполняются условия:

$$\begin{cases} \frac{\partial L}{\partial x} = 0, \quad L(x; \mu, \lambda) = f(x) + \sum\limits_{i=1}^{m} \mu_i g_i(x) + \sum\limits_{j=1}^{k} \lambda_j h_j(x) \\ g_i(x) \leq 0,\ h_j(x) = 0 \quad \text{(исходные ограничения)} \\ \mu_i \geq 0 \quad \text{(двойственные ограничения)} \\ \mu_i g_i(x) = 0 \quad \text{(условие дополняющей нежёсткости)} \end{cases}$$

При этом искомая точка является седловой точкой функции Лагранжа: минимумом по x и максимумом по двойственным переменным μ.

По теореме Каруша—Куна—Таккера, поставленная нами задача минимизации эквивалентна двойственной задаче поиска седловой точки функции Лагранжа:

$$L(\vec{w}, b, \xi; \lambda, \eta) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{\ell} \lambda_i \left(M_i(\vec{w}, b) - 1\right) - \sum_{i=1}^{\ell} \xi_i \left(\lambda_i + \eta_i - C\right)$$

λ_i — переменные, двойственные к ограничениям $M_i \geq 1 - \xi_i$, η_i — переменные, двойственные к ограничениям $\xi_i \geq 0$

Запишем необходимые условия седловой точки функции Лагранжа:

$$\begin{cases} \frac{\partial L}{\partial w} = 0, \quad \frac{\partial L}{\partial b} = 0, \quad \frac{\partial L}{\partial \xi} = 0 \\ \xi_i \geq 0, \quad \lambda_i \geq 0, \quad \eta_i \geq 0, \quad i = 1, \ldots, \ell \\ \lambda_i = 0 \text{ либо } M_i(\vec{w}, b) = 1 - \xi_i, \quad i = 1, \ldots, \ell \\ \eta_i = 0 \text{ либо } \xi_i = 0, \quad i = 1, \ldots, \ell \end{cases} \tag{1.32}$$

Продифференцируем функцию Лагранжа и приравняем к нулю производные. Получим следующие ограничения:

$$\begin{aligned} \frac{\partial L}{\partial w} &= \vec{w} - \sum_{i=1}^{\ell} \lambda_i y_i \vec{x}_i = 0 \;\Rightarrow\; \vec{w} = \sum_{i=1}^{\ell} \lambda_i y_i \vec{x}_i \\ \frac{\partial L}{\partial b} &= -\sum_{i=1}^{\ell} \lambda_i y_i = 0 \;\Rightarrow\; \sum_{i=1}^{\ell} \lambda_i y_i = 0 \\ \frac{\partial L}{\partial \xi_i} &= -\lambda_i - \eta_i + C = 0 \;\Rightarrow\; \eta_i + \lambda_i = C, \quad i = 1, \ldots, \ell \end{aligned}$$

Заметим, что $\eta_i \geq 0$, $\lambda_i \geq 0$, $C > 0$, поэтому из последнего ограничения получаем $0 \leq \eta_i \leq C$, $0 \leq \lambda_i \leq C$.

Диапазон значений λ_i (которые, как указано выше, соответствуют ограничениям на величину отступа) позволяет нам разделить объекты обучающей выборки на три типа:

- $\lambda_i = 0 \;\Rightarrow\; \eta_i = C;\; \xi_i = 0;\; M_i \geq 1$ — периферийные (неинформативные) объекты Эти объекты лежат в своём классе, классифицируются верно и не влияют на выбор разделяющей гиперплоскости (см. уравнение для $\vec{w}$);

- $0 < \lambda_i < C \;\Rightarrow\; 0 < \eta_i < C;\; \xi_i = 0;\; M_i = 1$ — опорные граничные объекты Эти объекты лежат ровно на границе разделяющей полосы на стороне своего класса
- $\lambda_i = C \;\Rightarrow\; \eta_i = 0;\; \xi_i > 0;\; M_i < 1$ — опорные объекты-нарушители Эти объекты лежат внутри разделяющей полосы или на стороне чужого класса

Определение 1.4. Опорный вектор — объект $\vec{x}_i$, соответствующий которому множитель Лагранжа отличен от нуля: $\lambda_i \neq 0$.

Теперь для решения проблемы линейной разделимости применим трюк с ядром (kernel trick) и подставим ограничения, которые мы получили при дифференцировании, в функцию Лагранжа. Получим следующую постановку двойственной задачи:

$$\begin{cases} -L(\lambda) = -\sum\limits_{i=1}^{\ell} \lambda_i + \frac{1}{2} \sum\limits_{i=1}^{\ell} \sum\limits_{j=1}^{\ell} \lambda_i \lambda_j y_i y_j K(\vec{x}_i, \vec{x}_j) \to \min\limits_{\lambda} \\ 0 \leq \lambda_i \leq C, \quad i = 1, \ldots, \ell \\ \sum\limits_{i=1}^{\ell} \lambda_i y_i = 0 \end{cases}$$

Суть, что если выборка объектов с признаковым описанием из $X = \mathbb{R}^n$ не является линейно разделимой, мы можем предположить, что существует некоторое пространство H, вероятно, большей размерности, при переходе в которое выборка станет линейно разделимой. Пространство H здесь называют спрямляющим, а функцию перехода $\psi : X \to H$ — спрямляющим отображением. Построение SVM в таком случае происходит так же, как и раньше, но в качестве векторов признаковых описаний используются векторы $\psi(\vec{x})$, а не $\vec{x}$. Соответственно, скалярное произведение $\langle \vec{x}_1, \vec{x}_2 \rangle$ в пространстве X везде заменяется скалярным произведением $\langle \psi(\vec{x}_1), \psi(\vec{x}_2) \rangle$ в пространстве H. Отсюда следует, что пространство H должно быть гильбертовым, так как в нём должно быть определено скалярное произведение.

Нелинейный классификатор с ядром K:

$$f(x) = sign\left(\sum_{i=1}^{\ell} \lambda_i y_i K(\vec{x}_i, \vec{x}) - b\right)$$

Алгоритмы для нахождения решения: SMO, INCAS. Они хороши тем, что учитывают особенности применяемой SVM архитектуры.

1.5.5 Случайный лес

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Используемые инструменты

- Python 3
- NumPy
- Pandas
- Matplotlib
- Pymorphy2
- Razdel
- Scikit-learn
- TensorFlow
- JavaScript
- HTML и CSS

2.2 Сбор данных

Самым главным этапом перед созданием моделей-классификаторов является сбор данных, этот процесс включает в себя несколько основных этапов:

- обработка и подготовка текстов;
- разметка текстов;
- обработка полученных результатов.

2.2.1 Обработка и подготовка текстов

В качестве основного текста для разметки был взят роман Михаила Афанасьевича Булгакова «Мастер и Маргарита».

Обработка документа:

а) произведение было очищено от нежелательных подстрок регулярными выражениями;

б) разделено на тексты по символу перевода строки «\n»;

в) из полученных текстов восстановлена прямая речь;

г) тексты содержащие больше 52 слов разделены с использованием библиотеки «razdel».

Формирование заданий:

- для разметки выделены тексты от 5 до 52 слов;
- для каждого текста определен контекст: не менее 40 слов перед и не менее 15 после текста.

2.2.2 Разметка текстов

Чтобы приступить к разметке сначала нужно определить множество меток классов, для этого обратимся к истории создания эмоциональных моделей ведушими профессорами в области изучения эмоций. В 1980 году Роберт Плутчик в своей работе [26] определил колесо эмоций рис. 2.1. Данная модель была взята за основу и дополнена моделью Пола Экмана, которую он описал в работе [17] 2004 года и обновил в статье [18] 2011 года.

В результате получилось множество, состоящее из 9 основных эмоций и их производных, дополненное нейтральным классом:

- **Злость** (anger) — желание выразить агрессию или причинить зло, общая для обеих моделей.
  Может проявляться в словах, мимике, поступках. Примеры: злость на оскорбление, на несправедливость, злость на плохое отношение.
  Гнев — более интенсивная эмоция, досада — менее.
- **Интерес** (anticipation) — предчувствие важного события, только в модели Плутчика.

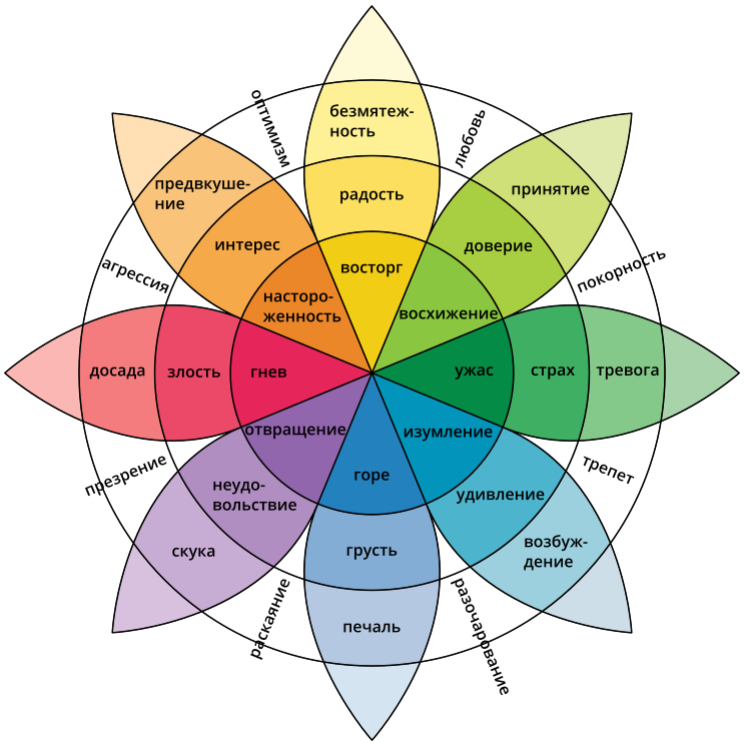

Рис. 2.1. Колесо эмоции Роберта Плутчика

Проявляется в нетерпении, волнении. Примеры: ожидание праздника, ожидание начала каникул, ожидание плохой оценки.

Настороженность — более интенсивная, предвкушение — менее.

- **Радость** (joy) — чувство удовольствия, весёлого настроения и счастья, общая для обеих моделей.

  Проявляется в смехе, улыбке, ласковом обращении к другим. Примеры: радость по поводу подарка, общения с другом.

  Восторг — более интенсивная, безмятежность — менее.

- **Доверие** (trust) — открытое теплое отношение к чему бы то ни было (другу/животному/миру/...), только в модели Плутчика.

  Проявляется в уверенности в положительном исходе. Примеры: доверие другу при встрече с неожиданностями, доверие к собаке, что не укусит, доверие к доктору, что он делает полезные вещи.

  Восхищение — более интенсивная, принятие — менее.

- **Страх** (fear) — состояние перед реальным или предполагаемым бедствием, общая для обеих моделей.

  Проявляется в волнении, напряжении. Пример: страх наказания, страх проигрыша, страх попасть в аварию.

Ужас — более интенсивная, тревога — менее.

- **Удивление** (surprise) — эмоциональная реакция на неожиданную ситуацию, общая для обеих моделей.
  Удивление может проявляться в хороших и плохих ситуациях. Примеры: получил плохой отзыв на работу вместо ожидаемого хорошего, директор школы привел в класс собаку, одноклассник вырос на 10 см за лето.
  Изумление — более интенсивная, возбуждение — менее.
- **Грусть** (sadness) — отсутствие радости, неудовлетворенность происходящим, отстраненность, общая для обеих моделей.
  Проявляется в нежелании веселиться с другими, желании заботы и участия. Примеры: мама уехала в командировку надолго, не покупают собаку или велосипед, никак не дается математика.
  Горе — более интенсивная, печаль — менее.
- **Неудовольствие** (disgust) — эмоциональная реакция на неприятную ситуацию или объект.
  Проявляется в неприятии человека, любых вещей, ситуаций, общая для обеих моделей. Примеры: когда сталкиваешься с неприятным запахом, грязными вещами, плохим поведением.
  Отвращение — более интенсивная, скука — менее.
- **Презрение** (contempt) — пренебрежительное отношение к кому-чему-нибудь морально низкому, недостойному, подлому. Презрение связано с чувством превосходства. Также оно может перейти в безразличное отношение к кому-чему-то. Только в модели Экмана.
- **Нейтральное** (neutral) — безэмоциональное повествование.

Разметка осуществлялась с помощью краудсорсинговой платформы «Яндекс.Толока».

Определение 2.1. Краудсорсинг — это привлечение добровольцев и экспетртов для выполнения определенной работы, действующих на добровольной или комерческой основе.

Для получения более точной разметки данных были использованы встроенные методы и инструменты контроля качества:

- график времени выполнения страницы заданий рис. 2.2 нужен, чтобы видеть на сколько вдумчиво эксперт расставляет метки;
- график выполнения заданий рис. 2.3 показывает сколько выполнено страниц заданий, сколько просрочено и сколько пропущено. По нему можно судить о сложности выполнения задания и использовать эту информацию в процессе формирования новых пулов заданий;
- сформирована система правил, которая позволяет контролировать процесс разметки в автономном режиме:
  - Если пропущенных подряд страниц заданий $\geqslant 10$, то заблокировать на проекте на 2 дня;
  - Если отправленных страниц заданий $\geqslant 50$, то заблокировать на проекте на 2 дня;
  - Минимальное время на страницу заданий — 250 сек. Учитывать последних страниц заданий — 15. Если количество ответов $\geqslant 5$ и количество быстрых ответов $\geqslant 5$, то заблокировать на проекте на 2 дня и т.д.
- размечены контрольные задания, с их помощью можно отслеживать примерную точность (accuracy), как всего набора данных, так и набора, полученного от одного эксперта;
- каждое задание выполняло три различных эксперта (перекрытие x3);
- агрегация результатов производилась методом Дэвида-Скина. Он автоматически оценивает для каждого исполнителя $|L|^2$ параметров, где L — множество возможных различных значений для агрегации и возвращает итоговый ответ и его статистическую значимость.

Была разработана форма задания рис. 2.4. Каждое задание состоит из трех текстовых блоков. Страница заданий состоит из двух не размеченных заданий и одного контрольного, такое разбиение оптимально для получения качественных результатов. Эксперт должен прочитать каждый

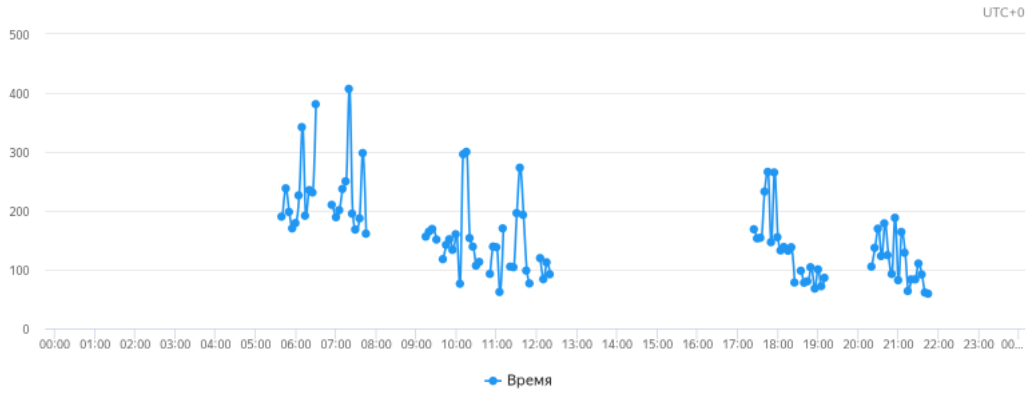

Рис. 2.2. Время выполнения страницы заданий (детализация по 5 минут)

Рис. 2.3. Выполнение страниц заданий (детализация по 5 минут)

текстовый блок и отметить эмоции, которые, по его мнению, описаны в выделенном фрагменте.

В результате был сформирован набор данных с таким распределением классов рис. 2.5.

2.3 Модели классификации

2.3.1 Предобработка текстов

Чтобы работать с текстами, сначала их нужно нормализовать. Этот процесс включает несколько этапов.

а) приводим текст к нижнему регистру;

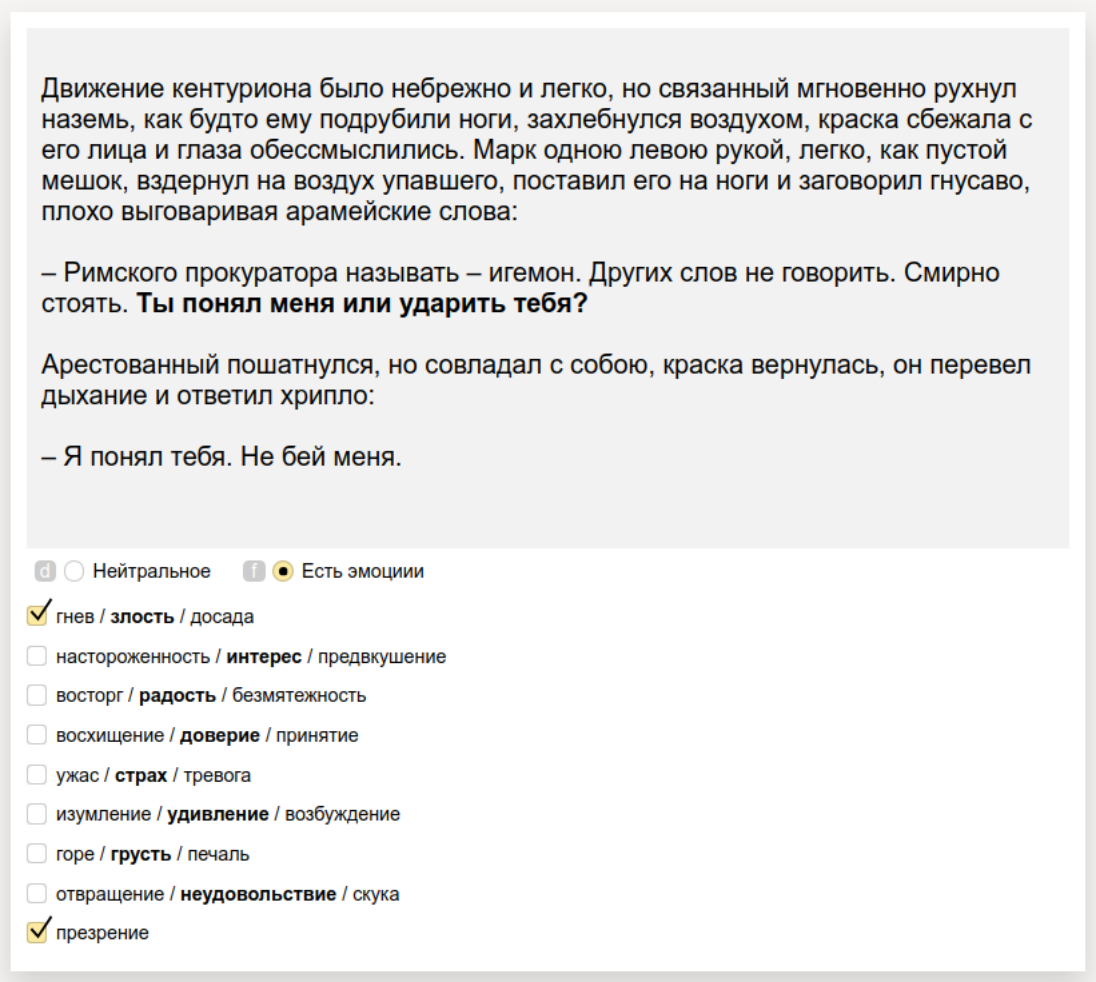

Рис. 2.4. Форма задания в сервисе «Яндекс.Толока»

б) удаляем все «не слова» и «стоп-слова»;

в) лемматизируем текст.

Вот пример обработки небольшого текста:

Квартира простояла пустой и запечатанной только неделю. → квартира простаивать пустой запечатывать неделя

2.3.2 Представление предложений

Теперь текст нужно перевести в векторное пространство R^n, где n — размерность признаково пространства используемой модели. Пусть текст D состоит из слов $d \in D$, f — модель, строящая отображение пространства слов в векторное пространство действительных чисел $f(d) \to R^n$. Тогда

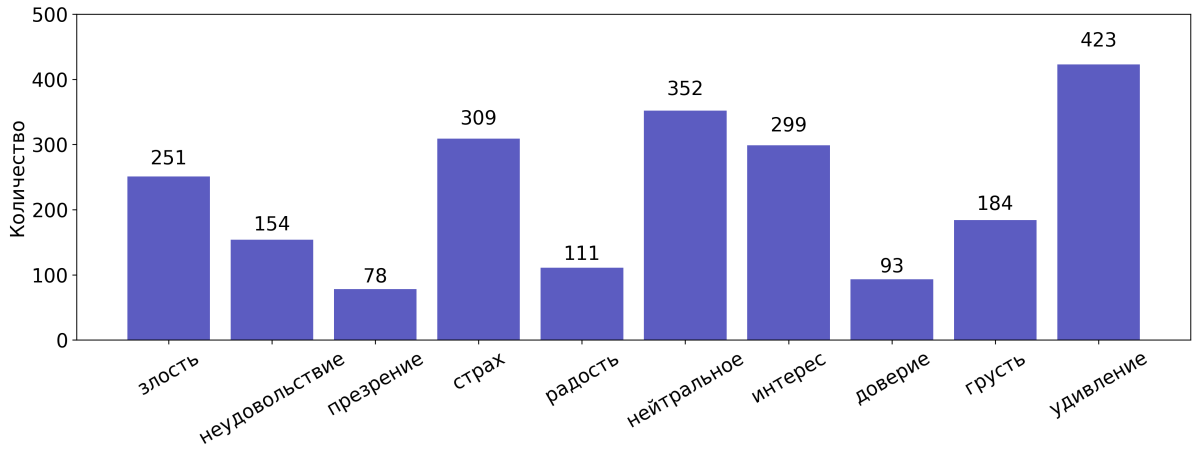

Рис. 2.5. Распределение классов в итоговом наборе данных

текст для классификатора выглядит так:

$$\frac{1}{\#D}\sum_{d\in D} f(d) \in R^n$$

В этой работе использованы модели предобученные на корпусе русскоязычных текстов «Тайга»:

- word2vec & skip-gram: *tayga_upos_skipgram_300_2_2019* ($n = 300$);
- ELMo: *tayga_lemmas_elmo_2048_2019* ($n = 2048$).

Особенность применения ELMo заключается в том, что берется среднее значение всех слоев для каждого слова.

2.4 Архитектура моделей классификации

Для классификации были выбраны алгоритмы классического машинного обучения:

- случайный лес (Random Forest);
- логистическая регрессия (Logistic Regression);
- метод опорных векторов (Support Vector Machine).

Схематично модели классификации представлены на рис. 2.6.

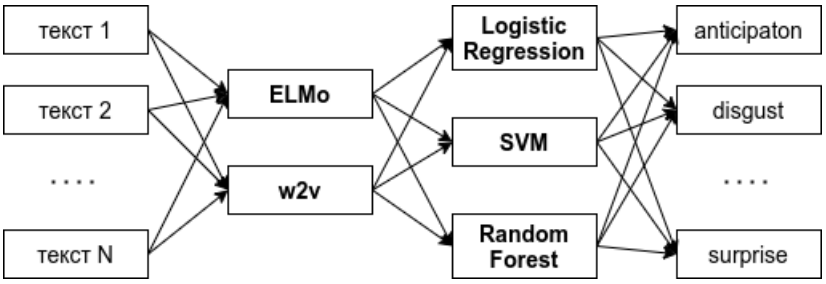

Рис. 2.6. Архитектура моделей классификации

2.5 Эксперименты

Для эмоциональной модели Роберта Плутчика результаты получились следующие:

Таблица 2.1 Значения Macro F1 меры

Macro F1	**log reg**	**SVM**	**random forest**	**tuned random forest**
w2v	0.21519548	0.27992996	0.23415391	0.23932876
ELMO	0.25900211	0.32828541	0.31197309	0.43861588

Для эмоциональной модели Пола Экмана:

Таблица 2.2 Значения Macro F1 меры

Macro F1	**log reg**	**SVM**	**random forest**	**tuned random forest**
w2v	0.25493596	0.18245115	0.21223037	0.37710993
ELMO	0.20242784	0.3007757	0.34183484	0.42405119

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы был сформирован оригинальный набор размеченных данных, содержащих эмотивную лексику. На основе знаний полученных из рассмотренной литературы выбрано несколько архитектур моделей классификации. Был подробно разобран и обоснован их математический аппарат и написана реализация на языке Python с использованием библиотек машинного обучения и предобученых семантических моделей русского языка.

Был проведен сравнительный анализ результатов точности классификации метода опорных векторов (SVM), логистической регрессии (logistic regression) и случайного леса (random forest) в связке с распределенными представлениями слов: word2vec и ELMo.

Согласно результатам исследования лучшие метрики показала модель, основанная на случайном лесе с решающими деревьями и ELMo. Из этого следует вывод, что для задачи сентимент-анализа использование более сложных и тяжеловесных контекстуализированных эмбеддингов — хорошее решение, т.к. они лучше обобщают сравнительно небольние тексты и позволяют классифиакору лучше разделить выборку. Но качество Macro F1 меры все равно не достигло высоких показателей. Возможно это связано с недостаточной величиной собранного набора данных в результате чего того количества признаков, которые выделила модель, не хватило для качественной классификации.

Возможными направлениями для дальнейших исследований могут стать модели для выделения именнованых сущностей. Определение текстов относящихся именно к этим сущностиям и реализация сентимент-анализа с использованием информации от этих моделей в более комплексных архитектурах с механизмом внимания. Также увеличение набора данных поможет улучшить результаты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 19-я Международная конференция «Авиация и космонавтика» //. — 2020.
2. XLVII Международная молодёжная научная конференция «Гагаринские чтения – 2021» //. — 2021.
3. Кирилл И. Анализ эмоциональной тональности литературы с помощью методов машинного обучения. — 2019.
4. Николенко С. И., Кадурин А., Архангельская Е. Глубокое Обучение. Погружение В Мир Нейронных Сетей. Т. 91. — 2018. — С. 480.
5. A neural probabilistic structured-prediction model for transition-based dependency parsing / H. Zhou [и др.] //. Т. 1. — Association for Computational Linguistics (ACL), 2015. — С. 1213—1222.
6. Atex. a rule-based sentiment analysis system processing texts in various topics. Система сентиментного анализа ATEX, основанная на правилах, при обработке текстов различных тематик. — Saint Petersburg State University.
7. Bag of Tricks for Efficient Text Classification / A. Joulin [и др.] // 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017 - Proceedings of Conference. — 2016. — Т. 2. — С. 427—431.
8. Baziotis C., Pelekis N., Doulkeridis C. DataStories at SemEval-2017 Task 4: Deep LSTM with Attention for Message-level and Topic-based Sentiment Analysis //. — Association for Computational Linguistics (ACL), 2018. — С. 747—754.
9. Bengio Y., Ducharme R., Vincent P. A neural probabilistic language model : тех. отч. — 2001. — С. 1137—1155.
10. BERT, ELMO и Ко в картинках (как в NLP пришло трансферное обучение) / Хабр. — URL: `https://habr.com/ru/post/487358/`.

11. BERT: Pre-training of deep bidirectional transformers for language understanding / J. Devlin [и др.] //. Т. 1. — Association for Computational Linguistics (ACL), 10.2019. — С. 4171—4186.

12. Cliche M. BB_twtr at SemEval-2017 Task 4: Twitter Sentiment Analysis with CNNs and LSTMs. — 2018.

13. Deep contextualized word representations / M. E. Peters [и др.] //. Т. 1. — Association for Computational Linguistics (ACL), 2018. — С. 2227—2237.

14. Distributed representations of words and phrases and their compositionality / T. Mikolov [и др.] // Advances in Neural Information Processing Systems. — 2013.

15. Duppada V., Jain R., Hiray S. SeerNet at SemEval-2018 Task 1: Domain Adaptation for Affect in Tweets. — 2018.

16. Efficient estimation of word representations in vector space / T. Mikolov [и др.] //. — International Conference on Learning Representations, ICLR, 2013.

17. Ekman P. Emotions revealed. — 2004. — С. 328.

18. Ekman P., Cordaro D. What is meant by calling emotions basic // Emotion Review. — 2011. — Т. 3, № 4. — С. 364—370.

19. Elmo-Deep Contextualized Word Representations / L. Cassani [и др.] // Food and Bioprocess Technology. — 2017. — Т. 10, № 8. — С. 1454—1465.

20. Generalized Language Models: CoVe, ELMo & Cross-View Training. — URL: `https://www.topbots.com/generalized-language-models-cove-elmo/`.

21. Goldberg Y., Levy O. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method : тех. отч. — 2014.

22. Lena Voita. Word Embeddings. — URL: `https://lena-voita.github.io/nlp_course/word_embeddings.html`.

23. NTUA-SLP at SemEval-2018 Task 3: Tracking Ironic Tweets using Ensembles of Word and Character Level Attentive RNNs / C. Baziotis [и др.]. — 2018.

24. Olah C. Neural Networks, Types, and Functional Programming. — 2015. — URL: https://research.google/pubs/pub45504/.

25. Pennington J., Socher R., Manning C. D. GloVe: Global Vectors for Word Representation : тех. отч.

26. Plutchik R. a General Psychoevolutionary Theory of Emotion // Theories of Emotion. — Elsevier, 1980. — С. 3—33.

27. Semina T. A. Sentiment analysis: Modern approaches and existing problems. // Социальные и гуманитарные науки. Отечественная и зарубежная литература. Серия 6: Языкознание. Реферативный журнал. — 2020. — С. 47—64.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

```
import warnings
def warn(*args, **kwargs):
    pass
warnings.warn = warn
import sys
import gensim
import numpy as np
from scipy import interp
from sklearn.pipeline import Pipeline
from sklearn import metrics, svm
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import SelectPercentile,
    f_classif
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

W2V_FILE = "model.txt"
ELMO_FILE = '199.zip'
FOLDS = 10

class MeanEmbeddingVectorizer(object):
    def __init__(self, word2vec):
        self.word2vec = word2vec
        self.dim = len(word2vec)

    def fit(self, X, y):
        return self

    def transform(self, X):
        return np.array([
```

```
31              np.mean([self.word2vec[w] for w in words if w in
    self.word2vec]
32                      or [np.zeros(self.dim)], axis=0)
33              for words in X
34          ])
35
36 def load_data_and_labels(data):
37      x_text = [t.split() for t in data.sentence.values]
38      labels = data.target.values
39      le = LabelEncoder()
40      labels = le.fit_transform(labels)
41      return x_text,labels
42
43 def preprocess_w2v(X, y):
44      X = np.array(X)
45      y = np.array(y)
46      X_train, X_test, y_train, y_test = [], [], [], []
47      skf = StratifiedKFold(n_splits=FOLDS)
48      for train_index, test_index in skf.split(X, y):
49          X_train_value, X_test_value = X[train_index], X[
    test_index]
50          y_train_value, y_test_value = y[train_index], y[
    test_index]
51          X_train.append(X_train_value)
52          X_test.append(X_test_value)
53          y_train.append(y_train_value)
54          y_test.append(y_test_value)
55      return X_train, X_test, y_train, y_test
56
57 def preprocess_elmo(X, y):
58      emb_model = ElmoModel()
59      emb_model.load(ELMO_FILE)
60      X = np.array(X)
```

```
61      y = np.array(y)
62      X_train, X_test, y_train, y_test = [], [], [], []
63      skf = StratifiedKFold(n_splits=FOLDS)
64      for train_index, test_index in skf.split(X, y):
65          features_train, features_test = X[train_index], X[
    test_index]
66          t_labels_train, t_labels_test = y[train_index], y[
    test_index]
67          features_train = elmo.get_elmo_vector_average(
    features_train)
68          features_test  = elmo.get_elmo_vector_average(
    features_test)
69          selector = SelectPercentile(f_classif, percentile=10)
70          selector.fit(features_train, t_labels_train)
71          X_train.append(selector.transform(features_train).
    toarray())
72          X_test.append(selector.transform(features_test).toarray
    ())
73          y_train.append(t_labels_train)
74          y_test.append(t_labels_test)
75      return X_train, X_test, y_train, y_test
76
77  def train(clf, X_train, X_test, y_train, y_test):
78      headers = 'fl,precision,recall,f1,support,acc,TN, FP, FN, TP
    '
79      print(headers)
80      for fl in range(FOLDS):
81          clf.fit(X_train[fl], y_train[fl])
82          pred = clf.predict(X_test[fl])
83          print(metrics.classification_report(y_test[fl], pred))
84          precision, recall, f1, _ = metrics.
    precision_recall_fscore_support(y_test[fl], pred, average="
    samples")
```

```
85          support = conf_matrix[0][0]+conf_matrix[0][1]+
    conf_matrix[1][0]+conf_matrix[1][1]
86          print(y_test[fl])
87          print(pred)
88          roc_auc = roc_auc_score(y_test[fl], pred, multi_class='
    ovo')
89          print(f'roc_auc: {roc_auc}')
90          print("%s,%s,%s,%s,%s,%s,%s,%s,%s,%s" % (fl, precision,
    recall, f1, support, acc, conf_matrix[0][0], conf_matrix
    [0][1], conf_matrix[1][0], conf_matrix[1][1]))
91
92  def benchmark(input_file):
93      X, y = load_data_and_labels(data)
94
95      clfs = [
96                RandomForestClassifier(class_weight="balanced"),
97                svm.SVC(),
98                LogisticRegression(solver="liblinear")
99
100     ]
101
102     names = ['RandomForest', 'SVM', 'log reg']
103
104     print("WORD2VEC")
105     with open(W2V_FILE, "rb") as lines:
106         w2v = {line.split()[0]: np.array(map(float, line.split()
    [1:])) for line in lines}
107     X_train, X_test, y_train, y_test = preprocess_w2v(X,y)
108     cont = 0
109     sections = ['Word2Vec', 'ELMo']
110
111     results = {}
112     section = sections[cont]
```

```
113     print(section)
114     results[section] = {}
115     for i in range(len(names)):
116         print(section + " " + names[i])
117         clf = Pipeline([
118             (section + " vectorizer", MeanEmbeddingVectorizer(
    w2v)),
119             ("classifier", clfs[i])])
120         results[section][names[i]] = train(clf, X_train, X_test,
     y_train, y_test)
121     cont += 1
122     section = sections[cont]
123     print(section)
124     results[section] = {}
125     X_train, X_test, y_train, y_test = preprocess_elmo(X,y)
126     for i in range(len(names)):
127         print(section + " " + names[i])
128         clf = clfs[i]
129         results[section][names[i]] = train(clf, X_train, X_test,
     y_train, y_test)
130     for name in names:
131         records = []
132         for section in sections:
133             values = results[section][name]
134             values['label'] = section
135             records.append(values)
136
137 def run(params):
138     if params["bench"]:
139         benchmark(params)
140     else:
141         predict(params)
142
```

```
143 def main(argv):
144     params = {
145         "bench": True,
146         "w2v_file": W2V_FILE
147     }
148 
149     run(params)
150 
151 if __name__ == "__main__":
152     main(sys.argv[1:])
```

ПРИЛОЖЕНИЕ 2

```
1  ########################################
2  #                                      #
3  #              html                    #
4  #                                      #
5  ########################################
6
7  <div class="text">
8    {{text1}} <b>{{text2}}</b> {{text3}}
9  </div>
10
11 <div class="first-scale">  {{field type="radio" name="isneutral"
       value="neutral" label="Нейтральное"
12 hotkey="d" class="neutral"}}
13   {{field type="radio" name="isneutral" value="notneutral" label
     ="Есть эмоциии" hotkey="f" class="notneutral"}}
14 </div>
15
16 <div class="second-scale" style="display: none;">
17   {{field type="checkbox" name="anger" label="гнев / <b>злость</
     b> / досада" class="block"}}
18   {{field type="checkbox" name="pl_anticipation" label="насторож
     енность / <b>интерес</b> / предвкушение" class="block"}}
19   {{field type="checkbox" name="joy"  label="восторг / <b>радост
     ь</b> / безмятежность" class="block"}}
20   {{field type="checkbox" name="pl_trust" label="восхищение / <b
     >доверие</b> / принятие" class="block"}}
21   {{field type="checkbox" name="fear" label="ужас / <b>страх</b>
       / тревога" class="block"}}
22   {{field type="checkbox" name="surprise" label="изумление / <b>
     удивление</b> / возбуждение" class="block"}}
23   {{field type="checkbox" name="sadness" label="горе / <b>грусть
```

```
    </b> / печаль" class="block"}}
24  {{field type="checkbox" name="disgust" label="отвращение / <b>
    неудовольствие</b> / скука" class="block"}}
25  {{field type="checkbox" name="ek_contempt" label="презрение"
    class="block"}}
26 </div>
27
28 ########################################
29 #                                      #
30 #              js                      #
31 #                                      #
32 ########################################
33
34 exports.Task = extend(TolokaHandlebarsTask, function(options) {
35     TolokaHandlebarsTask.call(this, options);
36 }, {
37     validate: function(solution) {
38         var isIlliterate = solution.output_values['isneutral']
    === 'notneutral',
39             isMainEmo = solution.output_values['anger'] ||
40                         solution.output_values['fear'] ||
41                         solution.output_values['surprise'] ||
42                         solution.output_values['sadness'] ||
43                         solution.output_values['disgust'] ||
44                         solution.output_values['joy'],
45             errorSelected = isMainEmo ||
46                             solution.output_values['ek_contempt
    '] ||
47                             solution.output_values['
    pl_anticipation'] ||
48                             solution.output_values['pl_trust']
49
50         if (isIlliterate && !errorSelected) {
```

```
51              return {
52                  task_id: this.getTask().id,
53                  errors: {
54                      '__TASK__': {
55                          message: "Пожалуйста, выберите эмоции"
56                      }
57                  }
58              };
59          } else {
60              return TolokaHandlebarsTask.prototype.validate.apply
    (this, arguments);
61          }
62      },
63      setSolution: function(solution) {
64          var secondScale = this.getDOMElement().querySelector('.
    second-scale');
65
66          secondScale.style.display = solution.output_values.
    isneutral === 'notneutral' ? 'block' : 'none';
67
68          TolokaHandlebarsTask.prototype.setSolution.call(this,
    solution);
69      },
70      onRender: function() {
71          // DOM-элемент задания сформирован (доступен через #
    getDOMElement())
72      },
73      onDestroy: function() {
74          // Задание завершено, можно освобождать (если были испол
    ьзованы) глобальные ресурсы
75      }
76  });
77
```

```
78 function extend(ParentClass, constructorFunction, prototypeHash)
       {
79     constructorFunction = constructorFunction || function() {};
80     prototypeHash = prototypeHash || {};
81     if (ParentClass) {
82         constructorFunction.prototype = Object.create(
       ParentClass.prototype);
83     }
84     for (var i in prototypeHash) {
85         constructorFunction.prototype[i] = prototypeHash[i];
86     }
87     return constructorFunction;
88 }
89
90 ########################################
91 #                                      #
92 #               css                    #
93 #                                      #
94 ########################################
95
96 .text {
97   max-width: 690px;
98   padding: 10px;
99   background-color: #F2F2F2;
100  white-space: pre-line;
101 }
102 .block {
103   display: block;
104 }
105
106 ########################################
107 #                                      #
108 #              input 1                 #
```

```
#                                      #
########################################

{
 "text1": {
  "type": "string",
  "hidden": false,
  "required": false
 },
 "text2": {
  "type": "string",
  "hidden": false,
  "required": false
 },
 "text3": {
  "type": "string",
  "hidden": false,
  "required": false
 }
}

########################################
#                                      #
#               input 2                #
#                                      #
########################################

{
 "fear": {
  "type": "boolean",
  "hidden": false,
  "required": false
 },
```

```
"anger": {
 "type": "boolean",
 "hidden": false,
 "required": false
},
"disgust": {
 "type": "boolean",
 "hidden": false,
 "required": false
},
"sadness": {
 "type": "boolean",
 "hidden": false,
 "required": false
},
"surprise": {
 "type": "boolean",
 "hidden": false,
 "required": false
},
"isneutral": {
 "type": "string",
 "hidden": false,
 "required": true
},
"ek_contempt": {
 "required": false,
 "type": "boolean",
 "hidden": false
},
"joy": {
 "required": false,
 "type": "boolean",
```

```
175   "hidden": false
176  },
177  "pl_anticipation": {
178   "required": false,
179   "type": "boolean",
180   "hidden": false
181  },
182  "pl_trust": {
183   "required": false,
184   "type": "boolean",
185   "hidden": false
186  }
187 }
```