СОДЕРЖАНИЕ

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор аналогов

Крупнейшими продуктами на рынке анализа тональности текста для множества естественных языков являются: Natural Language API, поставляемый на Google Cloud Platform; Text Analysis API, поставляемый на платформе Microsoft Azure; IBM Watson; WIT.AI принадлежащий Facebook. Данные решения представляют собой облачные сервисы, предоставляющие API пользователям, которое позволяет загружать тексты на естественных языках и возвращает их анализ. Кроме анализа тональности эти сервисы предоставляют и другие возможности, как определение частей речи (Part of Speech Tagging, PoS Tagging), морфологический разбор слов и синтаксический разбор предложений. Такие системы носят название NLP-конвейера (Natural Language Processing). К сожалению исходные коды этих систем закрыты, а применяемые в них принципы если и распространяются, то описаны в платных научных изданиях. Однако можно уверенно предположить, что Google применяет Globally Normalized Transition-Based Neural Networks[1]. Так как реализацию данного подхода Google анонсировал в виде мощного и высокопроизводительного NLP инструмента SyntaxNet.

Так же стоит обратить внимание на крупный академический проект от Стенфодрского университета — CoreNLP. CoreNLP — это открытый NLP конвейер, демонстрирующий практические возможности методов, изобретенных в результате широкого ряда исследований сотрудников Стенфорда.

Итак, вне зависимости от огласки принципов работы NLP-конвейеров, все ещё можно сравнить их бизнес-логику. Сравнение возможностей описанных выше закрытых коммерческих проектов друг с другом возможно только в ходе мощного маркетингово исследования. Но если сравнить их с некоммерческим CoreNLP, то станет очевидно, что CoreNLP, являясь лишь демонстрацией, очень отстает по производительности, но предоставляет более детальную визуализацию анализа тональности. CoreNLP в результате анализа предложения выдает синтаксическое дерево, узлами которого являются слова, а ветвями — синтаксические связи между словами. Таким образом каждое поддерево представляет собой фразу. И оценка тональности указывается для каждого узла. То есть, потенциальный пользователь может легко понять каким образом давалась оценка тональности всего предложения, глядя на оценки отдельных фраз. Коммерческие же продукты такой возможности не предоставляют, так как дают оценку всему предложению целиком в силу своей структуры. Таким образом, задача проектирования в создании модели с высокими возможностями визуализации, с использованием высокопроизводительных технологий.

1.2 Процесс выделения особенностей

Обычно задачи, связанные с классификацией текстов, можно разделить на три этапа: выделение особенностей, сжатие предложений и классификация. На первом этапе из текста выбирают слова для обработки и выбирают соответствующие им векторы. Затем производят сжатие набора векторов соответствующих словам в предложении в один вектор, который будет представлять все предложение. И далее полученные векторы предложений классифицируют[2]. Для классификации предложений выделение особенностей начинают со статической обработки текста, например убирают символы переноса строки, или заменяют символы, которые могут использоваться в формате представления набора данных, на аналоги или ключевые слова. Часто используется замена символов круглых скобок на сокращения “–LRB–” и “–RRB–”. С этим легко справится механизм регулярных выражений. Затем необходимо разбить предложения на единицы языка, несущие независимое семантическое значение — произвести токенизацию. Для большинства языков регулярные выражения так же справятся с задачей. Для русского языка хватит разбиения по символу пробела, в английском надо будет учитывать ещё и апострофы (“It’s” разобьется на “It” и “’s”). Однако для обработки большинства восточных языков так же приходится прибегать к машинному обучению, так как синтаксическое разделение слов на письме часто отсутствует, и токенизацию можно произвести только анализируя семантическое значение предложения. Для ограничения масштаба в данной работе методы токенизации рассматриваться не будут[2]. В работе для токенизации и синтаксического разбора будут использованы средства CoreNLP встроенные в библиотеку NLTK для Python.

Итак, после токенизации предложения представляют из себя набор единиц языка в строковом формате. Но нейронные сети работают только с числами. Поэтому необходимо представить слова в векторном виде. Данный процесс называется *встраивание слов*. Классический метод one-hot-encoding предлагает представлять слова в виде позиционных кодов. Для всех уникальных слов в корпусе строится словарь, где каждому слову соответствует вектор заполненный нулями и одной единицей, соответствующей позиции слова в словаре. Очевидно, это очень производительный метод встраивания слов, так как алгоритм кодирования слов в новом корпусе имеет линейную сложность. Однако прямая зависимость размера вектора от количества уникальных слов в корпусе может вызвать проблемы с хранением этих векторов. Например, корпус книг от Google содержит 1 миллион уникальных слов, и с one-hot-encoding каждое слово будет занимать 30 мегабайт памяти, если пользоваться в вычислениях хотя бы 32-битными числами. Так же семантическое значение слов в данном методе теряется — сравнение синонимов, антонимов и никак не связанных между собой по значению слов даст один и тот же результат. Метод сжатия векторов слов в единый век-

Term	Document 1	Document 2
aid	0	1
all	0	1
back	1	0
brown	1	0
come	0	1
dog	1	0
fox	1	0
good	0	1
jump	1	0
lazy	1	0
men	0	1
now	0	1
over	1	0
party	0	1
quick	1	0
their	0	1
time	0	1

Рисунок 1.1 – Пример мешков для двух предложений[2]

тор, соответствующий предложению, заключается в простом суммировании векторов слов и носит название *Мешок слов*.[2]. Пример мешков слов представлен на рисунке 1.1.

Скалярное произведение двух таких векторов предложений даст степень схожести этих предложений, основывающейся только на количестве одинаковых слов встречающихся в обоих предложениях. Таким образом данный метод для трех предложений “Мальчик ударил мяч.”, “Мальчик ударился в учебу” и “Дети играют в футбол” сделает вывод о том, что схожи первые два предложения, хотя очевидно что по смыслу больше связаны первое и третье. Обучение нейронной сети на выборке из мешков слов не даст никаких результатов для задач классификации по значению не только по причине отсутствия семантики в векторных представлениях слов и фраз, но и из-за представления в виде позиционных кодов, когда большая часть вектора не несет никакой нагрузки, а содержит лишь ноли[2].

1.3 Технология word2vec

Существует ряд методов встраивания слов, в которых векторы сохраняют семантическое значение обрабатываемых слов. В основе этих методов лежит идея о том, что семантика слова заключена в контексте его применения. Самым значимым результатом исследований на почве этой идеи является технология word2vec. На сегодняшний день это самый распростра-

ненный и эффективный метод встраивания слов. Качественно обученная модель word2vec представляет собой словарь, в котором словам соответствуют так называемые плотные вектора[3].

Процесс обучения модели word2vec начинается с генерации произвольных значений векторов для изучаемых слов. Каждому слову будет соответствует два плотных вектора, так как слово в процессе обучения может участвовать в роли центрального слова, и в роли слова из контекста центрального слова. На каждом шаге обучения в тексте последовательно выбирается центральное слово и его контекст — слова которые отстоят от центрального на m слов слева и справа. Для каждого центрального слова t делается предсказание слов в контексте[3]. Целевой функцией оптимизации в данном случае будет

$$J'(\Theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \geq m \\ j \neq 0}} p(w_{t+j}|w_t; \Theta), \tag{1.1}$$

где Θ — это параметры модели, изменяемые в ходе обучения.

Тогда отрицательная логарифмическая функция максимального правдоподобия $J(\Theta)$ будет равна:

$$J(\Theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \geq m \\ j \neq 0}} \log(p(w_{t+j}|w_t)). \tag{1.2}$$

Для предсказания вероятности слова в контексте применяется функция softmax, описанная в общем виде в выражении

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}. \tag{1.3}$$

Вероятность нахождения слова o в контексте слова c — это softmax функция для c по o

$$p(o|c) = \frac{\exp(u_o^T \cdot v_c)}{\sum_{w=1}^{V} \exp(u_w^T \cdot v_c)}. \tag{1.4}$$

Матрица U хранит вектора в для слов из контекста, а V — для центральных слов. Градиент для обратного прохода для 1.2 и 1.4 будет равен:

$$\frac{\partial}{\partial v_c} \frac{\exp(u_o^T \cdot v_c)}{\sum_{w=1}^{V} \exp(u_w^T \cdot v_c)} = u_o - \sum_{x=1}^{V} p(x|c) \cdot u_x. \tag{1.5}$$

После того, как градиент рассчитан значение градиента отнимается от всех обучаемых параметров модели, то есть от матриц U и V.

В результате множества итераций обучения будет получена простейшая softmax модель word2vec, которая будет представлять из себя матрицу V. Полученные вектора хранят семантическое значение слов, таким обра-

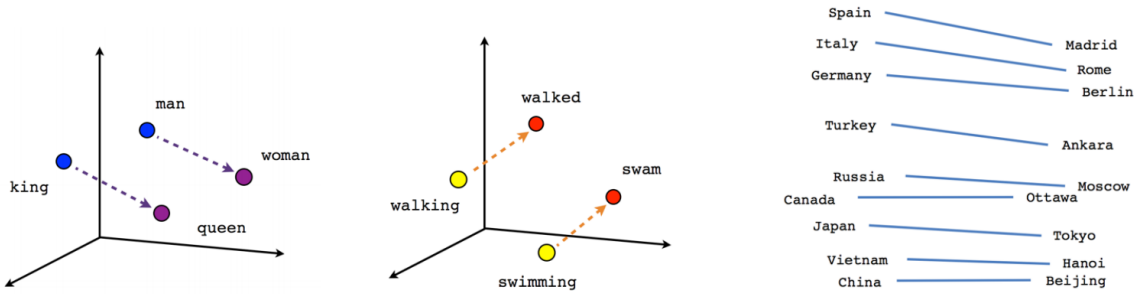

Рисунок 1.2 – Отношение между векторами в примере word2vec[3]

зом близкие по смыслу слова и синонимы будут располагаться близко друг от друга. А математические операции над этими векторами будут давать интересные результаты. Например если отнять от вектора “Король” вектор “Мужчина” и добавить вектор “Женщина” то будет получен вектор слова “Королева”. На рисунке 1.2 видно, что вектора располагаются параллельно вдоль некоторых осей, выученных моделью, по признакам пола, части речи и географического положения. Реализовать статическую модель распознающую подобные признаки невероятно трудно[3].

1.4 Классическая рекурсивная нейронная сеть

Когда каждое предложение представляет собой набор векторов, соответствующих языковым единицам, необходимо провести сжатие — обработать вектора слов таким образом, чтобы каждому предложению соответствовал один плотный вектор. Простейшая нейронная сеть, которая может быть применена в данной задаче — это рекурсивная нейронная сеть (RNN). Данная сеть имеет два входа: x_t — входной вектор, s_{t-1} — вектор состояния с прошлой итерации. Сеть последовательно применяется к векторам в предложении слева направо, и выдает на каждой итерации выходной вектор o_t и вектор состояния s_t.

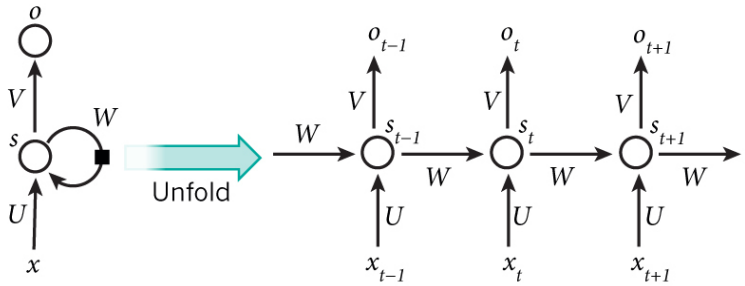

Рисунок 1.3 – Рекурсивная нейронная сеть в развертке во времени[2]

Состоит сеть из трех матриц весов: входная матрица U, матрица состояний W, матрица выходов V. В описывается двумя функциями 1.6 и 1.7.

Softmax функция описана в 1.3. Пример рекурсивной нейронной сети и её развертки во времени представлен на рисунке 1.3.

$$s_t = f(U \cdot x_t + W \cdot s_{t-1}), \tag{1.6}$$

$$o_t = softmax(V s_t), \tag{1.7}$$

где f — это функция активации, обычно выбирают тангенс.

После обхода всего предложения, получается выходной вектор, и появляется возможность посчитать функцию потерь, и вычислить значение градиента. На рисунке fig:overview:rnn показана развертка во рекурсивной сети во времени. Таким образом модель можно представить в виде многослойной сети. А значит модель страдает от проблемы затухающего градиента, когда градиентный спуск необходимо произвести на множество слоев вниз, и значение функции ошибки может очень сильно отличаться от реального её значения. Но так как применяется одна и та же сеть на каждом слое, то проблема затухающего градиента для рекурсивной сети усиливается. Поэтому с помощью RNN не было получено высоких результатов в классификации предложений. Так же рекурсивная сеть сталкивается с проблемой отдаленных зависимостей. Суть проблемы в том, что семантически связанные слова, которые имеют главную роль в понимании предложения, могут находиться удаленно друг от друга в предложении. И так как RNN обходит предложение слева на право, то информация о том, что первое слово встречалось в предложении может уже быть потеряно к тому моменту, как на вход придет второе, и модель сделает неверный вывод.

1.5 Long Short Term Memory

Проблему отдаленных зависимостей решает нейронная сеть под названием Long Short Term Memory (LSTM). Это особый вид рекурсивной нейронной сети, способная выучить удаленные зависимости. Данное свойство было получено за счет усложнения структуры сети. Принцип применения и обучения остался таким же: предложение обходится слева направо последовательно подавая на вход сети плотные вектора соответствующие словам. Схема сети представлена на рисунке 1.4.

Внутреннее состояние сети описывается вектором состояния ячейки C_t и вектором скрытого слоя h_t. На вход принимает входной вектор x_t и состояние с прошлой итерации: C_{t-1} и h_{t-1}. Выходной вектор o_t равен вектору скрытого слоя h_t. Введена система врат. Врата блокируют или пропускают векторы между различными состояниями нейронной сети. f_t — сигнал забывания, сбрасывает состояния ячейки. i_t — входные врата, блокируют или пропускают входной вектор. И o_t — выходные врата, блокируют или пропускают выходной вектор[4].

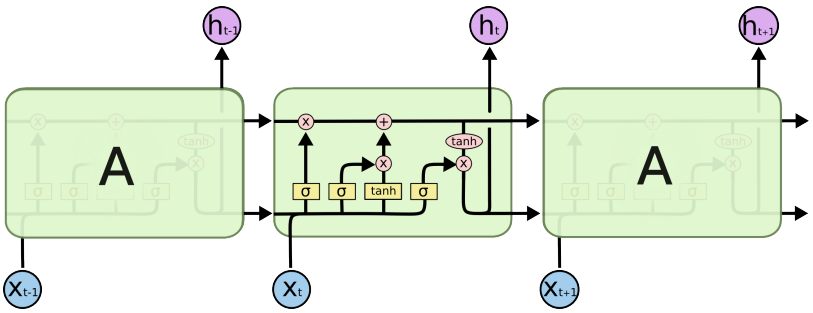

Рисунок 1.4 – Схематическое изображение LSTM-сети[4]

Состояния врат описаны выражениями 1.9, 1.11 и 1.10.

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{1.8}$$

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i), \tag{1.9}$$

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f), \tag{1.10}$$

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o), \tag{1.11}$$

где σ это функция сигмоида описанная 1.8;

W и U — это тензоры весов LSTM.

Состояния ячейки вычисляются согласно выражениям 1.13, 1.14 и 1.15.

$$(A \odot B)_{i,j} = (A)_{i,j} \cdot (B)_{i,j}, \tag{1.12}$$

$$\tilde{C}_t = \tan(W_C \cdot x_t + U_c \cdot h_{t-1} + b_c), \tag{1.13}$$

$$C_t = f_t \odot c_{t-1} + i_t \odot \tilde{C}_t, \tag{1.14}$$

$$h_t = o_t \odot \tan(c_t), \tag{1.15}$$

где $\tilde{C}_t$ носит название кандидата в состояние ячейки;

$\odot$ — это операция произведения Адамара, описанная в 1.12 для двух матриц A и B.

LSTM на сегодняшний день лежит в основе самой эффективной модели анализа тональности предложений Sentiment Neuron от сообщества OpenAi. Помимо лидерства на текущий момент, Sentiment Neuron обучается без учителя — это единственная успешная модель способная сжимать вектора слов в плотный вектор предложения и обучающая без учителя. Однако обучение этой модели крайне дорого. OpenAi обучали её на четырех Nvidia Pascal Titan X и обучение заняло приблизительно один месяц[5].

1.6 Рекурсивная тензорная нейронная сеть

Проблема отложенных связей может решаться иначе. Очевидно, что порядок слов в предложении редко совпадает с нитью размышлений автора. Обход предложения слева направо не может запомнить все семантически значимые последовательности, особенно в языках со специфической грамматикой. Поэтому исследователи решили изменить порядок обхода предложений. Один из итогов исследований — это модель рекурсивной нейронной тензорной сети (RNTN), которая легла в основу CoreNLP. Обход предложения производится по синтаксическому дереву, построенному согласно генеративной грамматике Хомского[6]. Итак, в процесс выделения особенностей добавляется ещё один шаг — синтаксический анализ предложения. Для RNTN необходимо на входе иметь синтаксическое дерево составляющих — один из видов синтаксических деревьев. Это дерево удобно тем, что его можно нормализовать, то есть привести к виду бинарного дерева. На рисунке 1.5 показан пример синтаксического дерева составляющих[6].

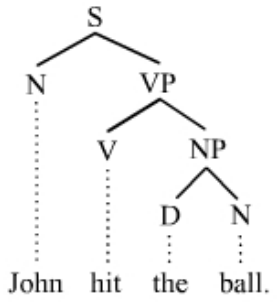

Рисунок 1.5 – Пример синтаксического дерева составляющих[7]

Как видно из рисунка 1.5, слова представлены листьями дерева, а его узлы — это различные виды составляющий в предложении. Типы связей не нужны в модели RNTN, интересен только сам факт их наличия и какие слова и фразы они объединяют. Для того, чтобы эффективно обрабатывать дерево составляющих, RNTN модифицирована и имеет два входа, и один выход. То есть она принимает два вектора с нижних узлов и передает верхнему, и т.д., пока в результате обработки всего дерева не будет получен один вектор, соответствующий всему предложению. Так же модель выдает вектор для каждого узла в дереве, что соответствует фразам в предложении. Это свойство эффективно применяется при обучении. Специально для обучения RNTN был создан Stanford Sentiment Treebank (SST) — Стенфордский набор деревьев тональности. Это набор более чем из десяти тысяч синтаксических деревьев, все узлы которых оценены по тональности носителями языка. SST стал очень популярен и вне исследований RNTN и является классическим набором для измерения эффективности метода оценки тональности[8].

Пусть $\begin{bmatrix} a \\ b \end{bmatrix}$ — два вектора размерности d объединенные в один размерностью $2d$. Тогда значение вектора результата p для входных векторов a и b высчитывается по формуле 1.16[8].

$$p = f(\begin{bmatrix} a \\ b \end{bmatrix}^T \cdot V \cdot \begin{bmatrix} a \\ b \end{bmatrix} + W \cdot \begin{bmatrix} a \\ b \end{bmatrix}), \tag{1.16}$$

1.7 Tree LSTM

На следующий год после релиза RNTN, изменяется подход в синтаксическом разборе предложений, так как выходит работа с описанием алгоритма синтаксического разбора с линейной сложностью. Однако этот алгоритм возвращает дерево зависимостей, а RNTN работает с деревом составляющих. Пример дерева зависимостей представлен на рисунке 1.6. Дерево зависимостей несет в себе меньше информации, чем дерево составляющих, так как дерево составляющих можно сконвертировать в зависимости без потерь, а обратный процесс без потерь невозможен[6].

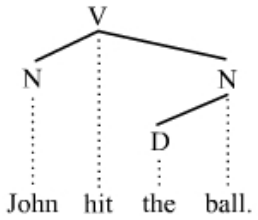

Рисунок 1.6 – Пример синтаксического дерева зависимостей[7]

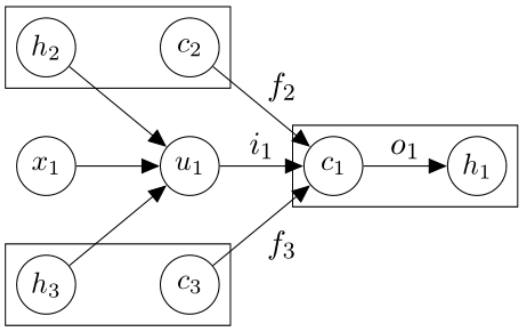

Рисунок 1.7 – Пример подключения узла Tree LSTM к дочерним узлам. h_2, c_2, h_3, c_3 — состояния двух дочерних узлов, h_1, c_1 — состояние узла родителя. x_1 — входной вектор узла.[9]

Дерево зависимостей для задачи классификации отличается тем, что каждый узел дерева содержит вектор слова. Так же дерево не является би-

нарным — каждый узел может иметь произвольное количество детей. Для обработки подобных деревьев была разработана модель Tree LSTM, которая существует в двух модификациях: N-арная Tree LSTM, для деревьев составляющих и Child-Sum Tree LSTM, для работы с деревьями зависимостей. В данной работе была реализована модель Child-Sum Tree LSTM[9].

Итак, ячейка Child-Sum Tree LSTM в некотором узле дерева принимает на вход состояния дочерних узлов. Пример подключения показан на рисунке 1.7. Состояние ячейки Tree LSTM, как и в обычной LSTM, описано двумя векторами: h_j и c_j — значение скрытого слоя и внутреннего состояния соответственно. Состояние для узла j, имеющего множество детей $C(j)$ можно выразить следующим образом:

$$\tilde{h_j} = \sum_{k \in C(j)} h_k, \tag{1.17}$$

$$i_j = \sigma(W^{(i)} \cdot x_j + U^{(i)} \cdot \tilde{h_j} + b^{(i)}), \tag{1.18}$$

$$f_{jk} = \sigma(W^{(f)} \cdot x_j + U^{(f)} \cdot h_k + b^{(f)}), \tag{1.19}$$

$$o_j = \sigma(W^{(o)} \cdot x_j + U^{(o)} \cdot \tilde{h_j} + b^{(o)}), \tag{1.20}$$

$$u_j = \tan(W^{(u)} \cdot x_j + U^{(u)} \cdot \tilde{h_j} + b^{(u)}), \tag{1.21}$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k, \tag{1.22}$$

$$h_j = o_j \odot \tan(c_j), \tag{1.23}$$

где $k \in C(j)$ для выражения 1.19.

1.8 Классификация

Последним этапом NLP-конвейера будет классификация плотных векторов предложений, полученных в результате сжатия. Стоит отметить, что плотные вектора предложений кодируют различные семантические свойства этих предложений, а значит с их помощью можно решать различные задачи классификации. Например сравнивать предложения по значению, похожи ли они, или противоположны, либо же нейтральны. Однако в данной работе интересная задача оценки тональности текста[2].

Для оценки тональности обычно обучают простейшую регрессионную модель

$$p_\Theta(y|x_j) = softmax(W \cdot h_j + b), \tag{1.24}$$

$$y_j = \arg\max_y(p_\Theta(y|x_j)), \tag{1.25}$$

где h_j — это плотный вектор предложения;

$\arg\max_y$ — функция, которая вернет индекс максимального элемента вектора.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Изучив теоретические аспектыn разрабатываемого приложения и выработав список требований необходимых для разработки модуля, разбиваем систему на компоненты. В разрабатываемом приложении можно выделить следующие блоки:

- модуль сбора статистики;
- модуль объектно-реляционного отображения;
- база данных объектов для анализа;
- модуль развертки;
- тренировочный набор Стенфорда;
- модуль выделения особенностей;
- модуль модели анализатора;
- модуль ячейки Tree LSTM;
- визуализации CoreNLP;
- модуль визуализации TensorBoard.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.009 С1.

2.1 Подготовка данных

Чтобы начать работу модели распознования тональностей, необходимы входные данные. Первый тип входных данных — это тренировочный набор Стенфорда. Он необходим для обучения модели распознованию тональностей. Стенфордский набор Представляет собой выборку синтаксических деревьев составляющих. Однако модель должна работать с синтаксическими деревьями зависимостей. Поэтому необходимо привести его к виду синтаксических деревьев зависимостей. Так как исследование синтаксического разбора не входит в задачи проектирования, то для этой цели будет использоваться синтаксический анализатор из CoreNLP. Он не является самым производительным, но самым простым в применении. За несколько секунд он обрабатывает более 10000 предложений, составляющих Стенфордский тренировочный набор. Обработать тренировочный набор необходимо всего один, при устafновке программы, поэтому скорость в несколько секунд вполне удовлетворительна. Объем анализатора вместе с моделью нейросети, обученной синтаксическому разбору, составляет более 350 мегабайт. Далее нужен предобученный набор встроенных векторов GloVe. Он содержит в себе 5 гигабайт слов и соответствующих им векторам размерностью в 300 состовляющих каждый. Чтобы сократить размер, занимаемый набором GloVe, он будет отфильтрован: будут выброшены все слова, которые не встречаются в Стенфордском тренировачном наборе. После такой фильтрации набор встраиваемых векторов будет занимать около 30 мегабайт дискового пространства. Фильтрацию так же необходимо про-

извести только один раз при установке программы. То есть, если перед началом работы программы произвести предобработку входных данных, то это повысит производительность и сократить объем, занимаемый системой на диске. Итак, чтобы решать проблему управления зависимостями, введен модуль развертки. В его задачи входит:

- проверить наличие отфильтрованного набора GloVe;
- проверить наличие Стенфорского тренировочного набора в виде синтаксических деревьев зависимости;
- проверить наличие синтаксического анализатора и скачать его, если отсутствует;
- при необходимости скачать оригинальные набор GloVe и тренировочный набор;
- при необходимости отфильтровать GloVe и удалить оригинальную версию;
- при необходимости обработать оригинальный Стенфордский тренировочный набор с помощью синтаксического анализатора;
- проверить наличиек необходимых библиотек и установить отсутствующие.

В результате работы модуля развертки, модель будет способна обучаться. Для того чтобы расширить возможности исследования возможностей модели, введен модуль сбора статистики. Его задачей будет сбор данных с сайта rottentomatoes.com, популярного форума о кино. Там собрано множество отзывов о фильмах последних десятилетий. Так же он предоставляет API для разработчиков, что поможет в сборе данных с ресурса. Для демонстрации хватит отзывов профессиональных критиков и обычных пользователей, и рейтинга фильма. Собранные через API данные необходимо хранить в базе данных, так как сбор данных для анализа может занять значительно больше времени, чем сам анализ. А база данных поможет сохранить результаты, и продолжить сбор информации в любое удобное время.

Так как сбор данных через API — задача достаточно тривиальная, в которой производительность зависит от скорости обмена данными с сервером, то производительность языка Python даст удовлетворительные результаты. Так же Python скриптовый язык, что упростит разработку. Для работы с REST API отлично подойдет библиотека requests для Python, которая просто представяет собой реализацию REST-интерфеса и довольно проста в использовании.

В качестве базы данных будет использована Mysql, как стабильное и надежное решение. Для удобства работа с базой данных будет организована по принципу объектно-реляционного отображения. Данный принцип предполагает абстракцию модели транзакций и предлагает программную модель — объекты базы данных представляются в виде объектов в модели

объектно-ориентированного программирования. Для реализации объектно-реляционного отображения будет использована библиотека sqlalchemy, которая позволяет используя возможности языка Python максимально абстрагироваться от SQL. В объектах sqlalchemy будут хранится данные из базы данных. И любые изменения этих объектов sqlalchemy представляет в виде транзакции в базу, таким образом поддерживая актуальность данных в базе и в памяти программы.

Таким образом сбор данных для NLP-модели будет осуществлятся модулем сбора статистики через API ресурса rottentomatoes.com. Полученные от сайта объекты REST будут сохраняться в объектах Python, которые в свою очередь, являясь объекто-реляционным отображениями базы данных, будут управлять данными в базе. И затем, когда данные из базы данных потребуется проанализировать, то они будут загруженны в виде Python-объектов в рамках той же объектно-реляционной модели.

2.2 Анализ данных

В модели нейронной сети может протекать два процесса: обучение и анализ. Обучение сети выполняется с помощью тренировочного набора Стенфорда. А анализ происходит на обученной сети с данными, собранными модулем сбора статистики. Процесс обучения очень сложен в плане вычислений и требует много времени и ресурсов. Поэтому существует возможность сохранения обученной сети в файл и последующей загрузки из файла. Однако для исследовательских целей контроль обучения модели очень важен. К тому же обучение модели Tree LSTM, за счет сложности архитектуры, происходит намного быстрее множества других нейронных сете. Процесс анализа же очень прост, так так математически нейронная сеть это линейная функция, и алгоритм анализа имеет линейную сложность.

Модуль выделения особенностей в процессе обучения модели выполняет синтаксический разбор тренировочного набора Стенфорда, что описано выше. В результате полученные деревья в виде списков Python сохраняются в формате JSON. В результате получается три набора: тренировочный, тестовый и набор разработки, каждый из которых составляет соответственно 10%, 10% и 80% от всего набора Стенфорда. В процессе анализа данный модуль производит синтаксический разбор предложений, подлежащий обработке. И точно так же в JSON формате подается в модуль модели анализатора.

Модуль анализатора и модуль ячейки Tree LSTM представляют собой непосредственно саму нейронную сеть как математический алгоритм. В качестве основного инструмента реализации сети выбор пал на Tensorflow. В машинном обучении Tensorflow является стандартом на сегодняшний день. Данный фреймворк имеет API различных уровней абстракций: с использованием низкого уровня можно реализовывать модель в виде последователь-

ности операций линейной алгебры, а на высоком уровне составными частями алгоритма являются слои нейронных сетей. Так же tensorflow предоставляет возможности выполнения вычислений на графических процессорах, и так же на тензорных процессорах (TPU). Кроме того, tensorflow представил в конце 2017го технологию *горячего выполнения*. Так как в классической архитектуре ПК оперативная память и память графического процессора разделены, и центральный процессор не имеет прямого доступа к видеопамяти, то программирование алгоритмов на графических процессорах вызывает сложности. Процесс написания кода на GPU с использованием популярных технологий, таких как CUDA, Torch, OpenCV и Tensorflow в том числе, заключается в предварительном построении графа выполнение, узлы которого — это операции GPU, а грани — это движение данных. И результат промежуточных вычислений недоступен, только выход всего графа целиком. Горячее вычисление предоставляет возможность получить промежуточные результаты без серьезных потерь производительности. Это во многом упрощает процесс отладки. В рамках данного проекта была построена модель с использованием этой техники. На данный момент в открытом доступе очень немного реализаций алогоритмов машинного обучения с горячими вычислениями.

Итак, модуль ячейки Tree LSTM представляет собой реализацию ячейки Child Sum Tree LSTM. Реализована она в виде класса наследника tensorflow.keras — базового элемента высокоуровнего API keras. То есть, ячейка Tree LSTM является полноценным слоем в рамках tensorflow, и, что интересно в данной работе, поддерживает автоматический рассчет градиента.

Модуль анализатора же не является классом keras, так как принимает на вход синтасические деревья. Объекты tensorflow созданы для того, чтобы работать и на CPU и на GPU, поэтому обрабатывают они только тензоры, которые легко хранить в графической памати. Дерево же довольно сложная структура, чтобы представить ее в виде тензора. Таким образом, модуль анализатора содержит в себе множество слоев нейронных сетей:

- слой проекции встроенных векторов;
- слой исключения;
- слой Child Sum Tree LSTM;
- слой линейной регрессии.

Каждый узел синтаксического дерева будет проходить последовательно все эти слои в процессе обучения модели. В процессе выполнения слой регрессии будет выполнятся только в корне дерева.

Процесс анализа обученной моделью некоторого дерева начинается с подачи дерева на вход модели. Модель рекурсивно спускается к листьям дерева и выполняет последовательно слои модели на каждом узле. В результате работы на выходе линейной регрессии получается вектор, элементами которого являются вероятности принадлежности входного дерева каждому

из целевых классов. Индекс элемента с наибольшей вероятностью и будет намером класса, в чью пользу модель сделала выбор.

Процесс обучения включает в себя многократное выполнение процесса анализа. Для обучения сети нужны тренировочный набор деревьев и набор разработки. Сам процесс обучения делится на эпохи. В ходе эпохи модель обрабатывает тренировочный набор целиком. В начале каждой эпохи элементы тренировочного набора перемешиваются в случайном порядке. Далее набор делится на равные группы, за исключением последней, если количество деревьев в наборе не кратно размеру группы. Каждая группа последовательно анализируется моделью, для каждого анализа по принципу обратной связи вычисляется функция потерь. Потери для деревьев в группе суммируются и после того, как группа целиком обработана, вычисляется градиент сети относительно каждого обучаемого параметра модели, куда в качестве аргумента передается суммарная потеря. Обучаемыми параметрами являются коэффициенты функций, которыми описаны различные слои нейронной сети. После этого, результаты градиента относительно каждого из обучаемых параметров сети добавляются к значениям этого же параметра.

После того, как обработаны все группы, эпоха считается законченой. Однако, помимо обучаемых параметров, модель имеет ещё и гипер-параметры — параметры, которые не возможно эффективно обучить методом градиентного спуска: например количество эпох, размер группы, количество элементов скрытого слоя того или иного слоя нейронной сети. Однако данные параметры играют очень важную роль в обучении сети и их успешный подбор зависит от опыта инженера. Для оценки успешности выбранных гипер-параметров, в конце каждой эпохи обучения производится анализ деревьев из набора разработки, и высчитывается оценка точности. Гипер-параметры изменяются для получения наилучшего результата точности на наборе разработки.

После того, как все эпохи обучения пройдены, модель анализирует деревья из тестового набора. Результирующая точность является качественной оценкой нейронной сети. Подбор гипер-параметров по тестовой выборке, вместо выборки разработки крайне не рекомендуется, так как может привести к переобучению сети, и результаты точности, полученные на тестовой выборке, могут очень сильно отличатся от результатов полученных в ходе практического примеенения сети.

Модуль визуализации CoreNLP представляет собой небольшой скрипт на JavaScript, который наглядно показывает, как считалась оценка тональности, и поможет понять, почему именно алгоритм сделал такой вывод. Это позволит выявить проблемы отсутствия какого-то ключевого слова в наборе векторов GloVe.

Модуль визуализации TensoBorad демонстрирует процесс изменения

весов во время обучения, что является очень интересной информацией для исследований.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Globaly Normalized Transition-Based Neural Networks / Daniel Andor [et al.].

2 Goodfellow, Ian. Deep Learning / Ian Goodfellow, Yoshua Bengio, Aaron Courville. — MIT Press, 2016. — http://www.deeplearningbook.org.

3 Distributed Representations of Words and Phrases and their Compositionality / Tomas Mikolov [et al.].

4 Hochreiter, Sepp. Long Short-Term Memory / Sepp Hochreiter, Jürgen Schmidhuber // Neural Computation. — 1997. — Vol. 9, no. 8. — Pp. 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

5 Radford, Alec. Learning to Generate Reviews and Discovering Sentiment / Alec Radford, Rafal Józefowicz, lya Sutskever.

6 Chomsky, Naom. Three models for the description of language / Naom Chomsky. — IRE Transactions on Information Theory, 1956.

7 Wikipedia contributors. Dependency grammar — Wikipedia, The Free Encyclopedia. — 2004. — [Online; accessed 11-April-2018]. https://en.wikipedia.org/wiki/Dependency_grammar.

8 Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank / Richard Socher [et al.]. — 2013.

9 Tai, Kai Sheng. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks / Kai Sheng Tai, Richard Socher, Christopher D. Manning.