ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ

ВЫСШАЯ ШКОЛА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ И СУПЕРКОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Отчет о прохождении стационарной учебной практики (научно-исследовательской работа (получение первичных навыков научно-исследовательской работы)) на тему:

«Разработка алгоритма для анализа атрибутивной тональности текста»

Покровского Валерия Дмитриевича, гр. 3530203/90002

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Место прохождения практики: СПбПУ, ИКНТ, ВШИСиСТ

(указывается наименование профильной организации или наименование структурного подразделения

ФГАОУ ВО «СПбПУ», фактический адрес)

Сроки практики: с 20.06.2020 по 15.07.2020.

Руководитель практики от ФГАОУ ВО «СПбПУ»:

Резединова Евгения Юрьевна, ст. преподаватель

(Ф.И.О., уч. степень, должность)

Консультант практики от профильной организации:

Нет

(Ф.И.О., должность)

Оценка:

Руководитель практики
от ФГАОУ ВО «СПбПУ» Резединова Е.Ю.

Консультант практики
от ФГАОУ ВО «СПбПУ» Туральчук К.А.

Обучающийся Покровский В.Д.

Дата:

ВВЕДЕНИЕ

Идея анализа тональности того или иного текста появилась еще во второй половине XX века. На тот момент использовались алгоритмы, основанные на лингвистических правилах и словарях. Однако с появлением машинного обучения и глубоких нейронных сетей (Deep neural networks) появились гораздо более сильные и мощные инструменты, дающие высокий результат классификации. На данный момент такие модели широко используются в бизнесе: определение тональности отзывов, комментариев, статей. Однако очевидный недостаток нейросетевых моделей – интерпретация алгоритма. Так как параметры модели – числовые матрицы часто большого размера, мы не можем сказать, что побудило сеть предсказать тот или иной класс. Поэтому целесообразно придумать более информативный алгоритм, который позволит находить в тексте эмоционально окрашенные слова, чтобы дать пользователю более интерпретируемый результат. Таким алгоритмом является анализ атрибутивной тональности текста. Идея заключается в нахождении пар «объект – характеристика». Такой подход делает возможным более детальный анализ.

Работа в первую очередь нацелена на атрибутивный анализ текста, также рассматривалась задача анализа общей тональности всего текста в целом.

1. ЗАДАЧА АНАЛИЗА АТРИБУТИВНОЙ ТОНАЛЬНОСТИ ОТЗЫВОВ С САЙТА KINOPOISK.RU

1.1. Постановка задачи

Сформулируем задачу и переведем ее в более формальный вид. Пусть имеется отзыв к фильму A. Задача заключается в том, чтобы, во-первых, определить его тональность в общем и, во-вторых, определить разные аспекты фильма и определить их эмоциональную окраску.

Более формально: для каждого отзыва (набора слов-токенов) поставить в соответствие один из классов (позитивный, нейтральный, негативный). Далее необходимо выделить токены, являющиеся атрибутами фильма и найти токен эмоциональной окраски для этого атрибута. Для каждого найденного эмоционально окрашенного поставить в соответствие один из классов (позитивный, нейтральный, негативный).

1.2. Данные для обучения

Данные для обучения были получены на платформе Kaggle [1]. В них содержится 131583 отзыва, разделенных на 3 класса (позитивные, нейтральные и негативные). Для стабильности было решено оставить только отзывы, в которых не более 700 слов. Распределение количества слов представлено на рис.1.1.

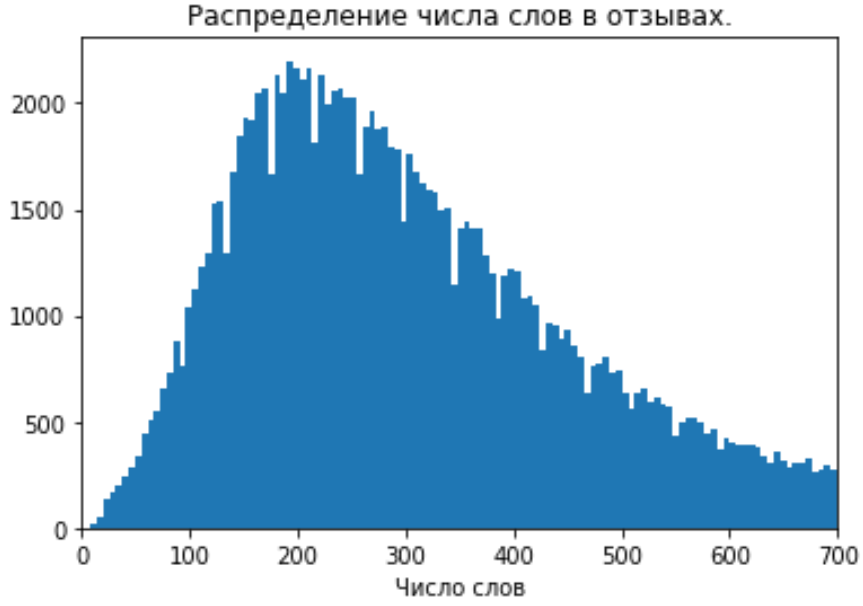

Рис 1.1. Распределение количества слов в отзывах.

2. РЕШЕНИЕ ЗАДАЧИ АНАЛИЗА ТОНАЛЬНОСТИ ОТЗЫВА

2.1. Существующие подходы к решению задачи анализа тональности

На данный момент наиболее перспективными являются нейросетевые подходы. Часто используются так называемые рекуррентные слои (RNN, LSTM, GRU) [2]. Их главной особенностью является возможность обработки последовательных данных разной длины в вектор фиксированной длины. Однако существенным недостатком является очень быстрое время «забывания» информации. Другими словами, слой хранит больше информации из конца последовательности, чем из начала. Идея алгоритма, использующего такой подход: разбить отзыв на слова-токены (используя какое-то промежуточное представление), свернуть их в вектор и дальше работать с вектором постоянной длины.

Другой популярной идеей является использование трансформеров (моделей семейства BERT) [3]. Не будем останавливаться подробно, отметим лишь, что идея этого семейства алгоритмов заключается в том, чтобы обучить сеть понимать, какие слова связаны друг с другом, а какие-нет. Обучение таких сетей занимает много времени и памяти.

2.2 Решение задачи анализа тональности

Для решения задачи была выбрана следующая архитектура сети (рис.2.1).

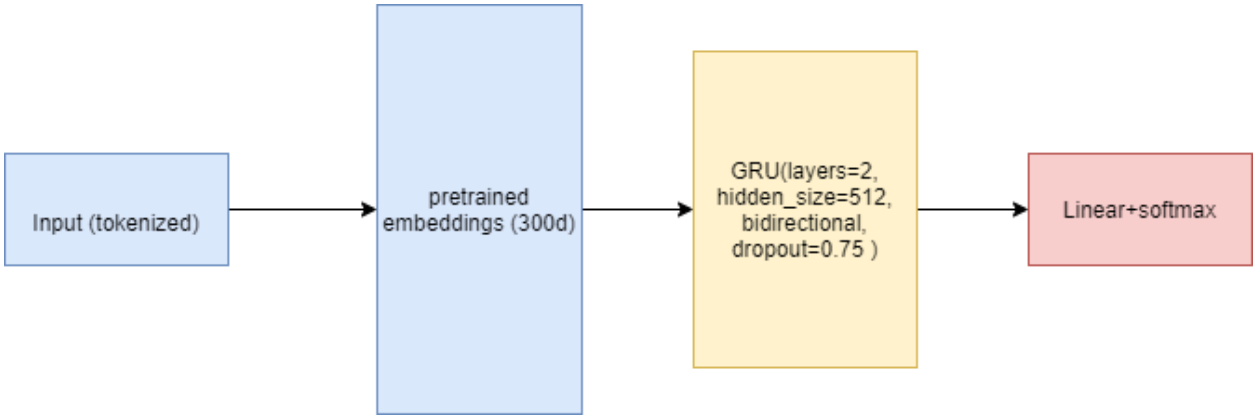

Рис. 2.1. Архитектура сети.

Сначала отзыв разделялся по словам, далее каждое слово кодировалось предобученной сетью в 300-мерное пространство. Далее форматированный

отзыв попадал в GRU сеть и, полученный на выходе вектор фиксированной длины с помощью полносвязного слоя и функции активации SoftMax становится распределением вероятностей принадлежности к тому или иному классу. Для функции ошибок будем использовать функцию кросс-энтропии:

$$Cross\text{-}Entropy = \Sigma_{i=1}^{n} p_i \, log\sigma_{z_i},$$

где (p_i – истинная вероятность принадлежности к i классу, σ_{z_i}-посчитанная вероятность принадлежности к i классу).

Для преобразования текстов в числовые вектора использовался проект natasha [4]. Проект направлен на упрощение работы с русским языком в контексте nlp задач. Наибольший интерес представляет векторное пространство слов. Оно обладает свойствами, близкими к линейным, так как было получено в результате обучения нейронной сети (в проекте natasha это модель из семейства трансформеров) поиску контекстуальных синонимов (модель word2vec).

Для тестирования было отложено 20% выборки. В результате была выбрана лучшая модель, которая добилась 80% точности (accuracy). График ошибки в зависимости от эпохи представлен на рис.2.2. Решение было написано с помощью фреймворка глубокого обучения pyTorch и языка python 3.7. Написание и запуск кода осуществлялся в среде Google Collaboratory.

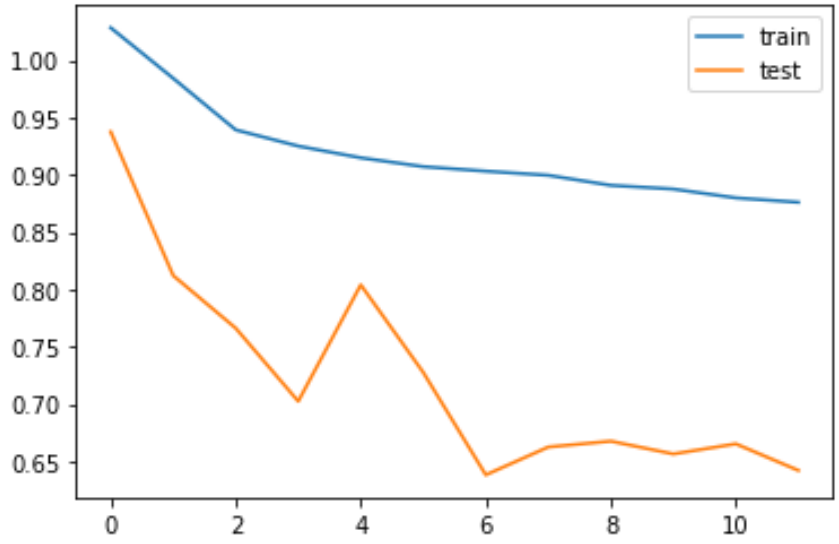

Рис. 2.2. Зависимость величины ошибки от эпохи обучения.

3. РЕШЕНИЕ ЗАДАЧИ АНАЛИЗА АТРИБУТИВНОЙ ТОНАЛЬНОСТИ ОТЗЫВА

3.1. Существующие подходы к решению задачи анализа атрибутивной тональности

На сегодняшний день существуют различные решения данной задачи, основанные, например, на трансформерах [5]. Однако такой алгоритм не походит к данной задаче, поскольку у нас нет размеченных примеров и нам необходимо решать unsupervised-learning задачу. Решение для такой задачи было представлено [6]. Идея данного подхода заключается в морфологическом анализе текста, поиске аспектов (частей речи) и их эмоциональной окраски. В статье используется модель word2vec для поиска контекстуальных синонимов и синтаксический парсер для поиска связи между словами. На основе этого алгоритма и была построена итоговая модель. На рис. 3.1 изображена визуализация свойств пространства, полученного с помощью word2vec.

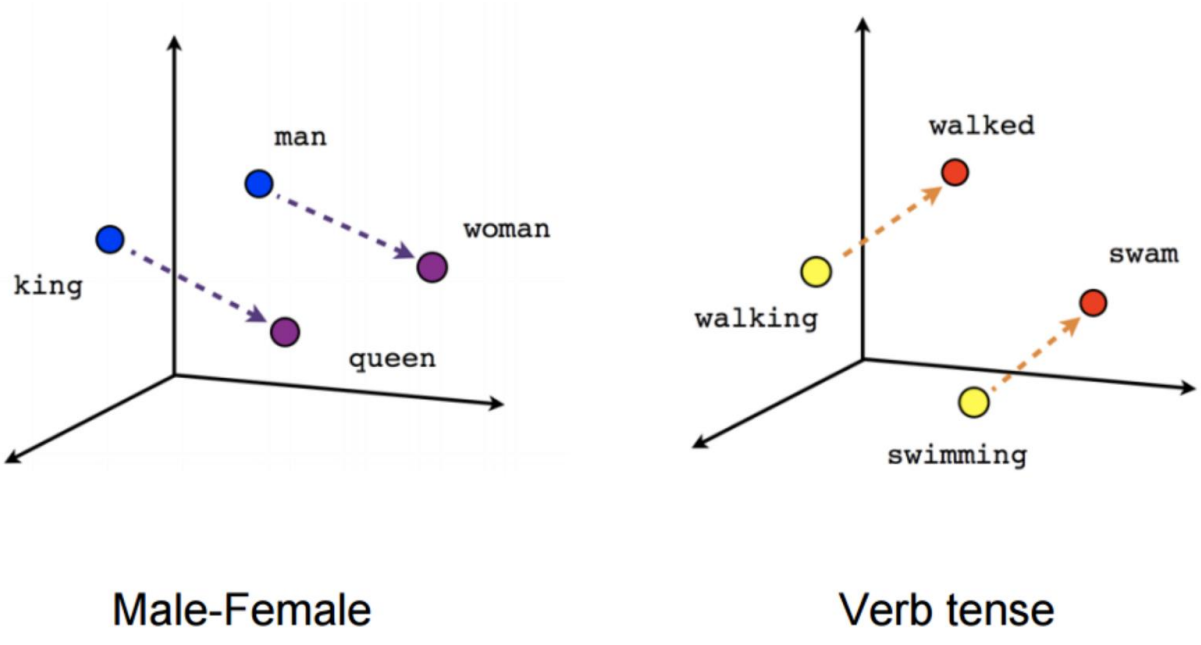

Рис. 3.1. Демонстрация линейных свойств word2vec.

3.2. Решение задачи анализа атрибутивной тональности

Сформулируем задачу более конкретно: необходимо для каждого отзыва определить множество пар (аспект, эмоциональная окраска). В качестве приближения предположим, что аспектами фильма являются имена существительные, а их окраской – имена прилагательные. Тогда задача сводится к поиску имен существительных и прилагательных и классификации прилагательного на тип эмоциональной окраски (позитивный, нейтральный, отрицательный). Недостатком такого подхода является то, что, некоторые существительные не являются аспектами фильма, а являются упоминанием героев, событий, что, возможно, не несет большой информации. Другой проблемой является то, что синтаксический парсер не может со стопроцентной точностью найти все пары имен существительных и имен прилагательных. К тому же сейчас не существует большого разнообразия модулей для обработки русского языка. Несмотря на это, агрегируя анализы разных отзывов к одному фильму, можно получить более целостную картину. Структура решения представлена рис. 3.2.

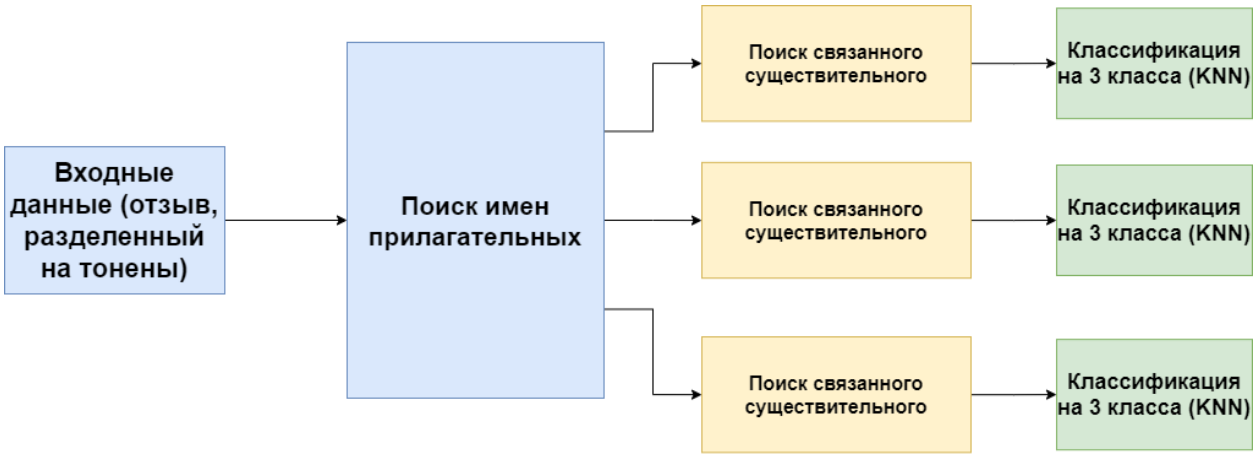

Рис. 3.2. Структура решения.

Теперь рассмотрим решение более подробно: после разделения на слова для поиска прилагательных в отзыве используется лингвистический модуль pymorphy2. Он позволяет производить морфологический разбор слов в контексте и без. Его работа основана на заранее прописанных правилах и словарях. На данный момент доступна работа с тремя языками: русским,

украинским и английским. Это один из самых эффективных модулей для морфологического анализа русского языка.

Далее для каждого найденного прилагательного необходимо найти слово в предложении, от которого оно зависит. Для этого используется модуль slovnet.syntax из модуля natasha. Он позволяет с легкостью сделать синтаксический разбор предложения, что позволяет найти нужное слово. Далее, если главное слово – имя существительное, сохраняем эту пару слов.

Теперь необходимо классифицировать прилагательное как позитивное, нейтральное или негативное. Для этого, во-первых, нужны предобученные эмбеддинги слов, во-вторых, словарь каких-либо «эталонных слов», для которых уже существует разметка по классам. В качестве размеченных слов был выбран список из 6000 прилагательных [7]. Алгоритмом классификации является метод ближайших соседей (K Neighbor Classifier, или, сокращенно, KNN). Метод состоит из двух этапов: на первом эталонные прилагательные (преобразованные в вектора в 300-мерном пространстве) подаются на вход алгоритму. Положение тренировочных примеров запоминается. На втором этапе, для каждого нового прилагательного выбираются k самых близких к этому объекту в 300-мерном пространстве тренировочных объектов. Для прилагательного выбирается тот класс, экземпляров которого оказалось больше в k ближайших объектах. При этом в качестве оценки расстояния могут быть использованы различные метрики (евклидово расстояние, манхэттенское расстояние, косинусная мера). Наглядная иллюстрация принципа работы алгоритма ближайших соседей представлена на рисунке 6. Данный алгоритм является наиболее естественным и понятным для этой задачи, поскольку полученные векторные пространства позволяют рассчитывать на геометрическую близость контекстуальных синонимов (что как раз и нужно). На рис. 3.3 изображен принцип работы KNN.

Далее результаты алгоритма (пары существительное и прилагательное) передаются для последующей визуализации.

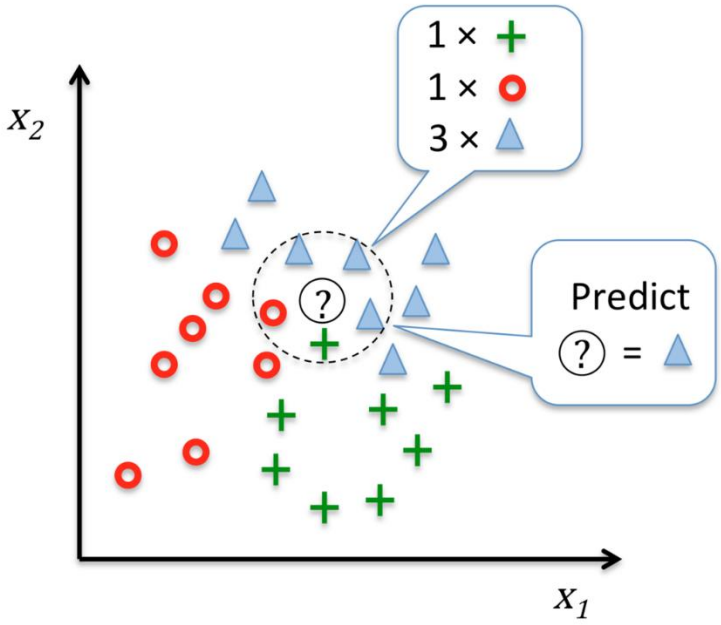

Рис. 3.3. Принцип работы алгоритма KNN.

Для тестирования алгоритма выборка размеченных прилагательных была разделена на 80% для обучения алгоритма и 20% для тестирования. В результате перебора были выбран гиперпараметр (кол-во соседей). График зависимости величины f-score от кол-ва соседей представлен на рис. 3.4.

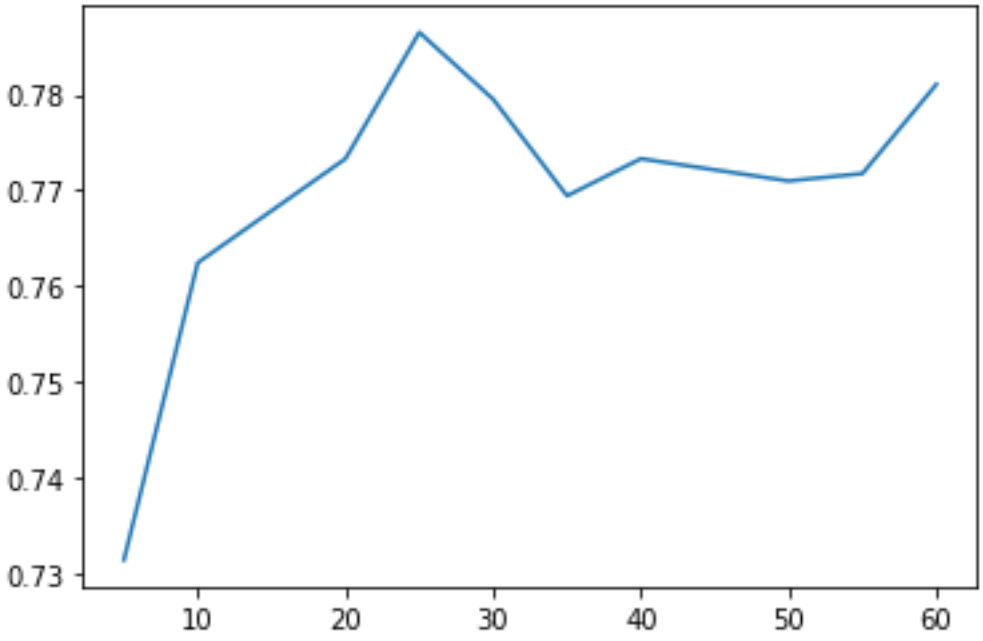

Рис. 3.4. Зависимость величины f-score от кол-ва соседей.

В данной задаче имелась так называемая проблема неравенства классов. Так, количество нейтрально окрашенных прилагательных в словаре

составляет больше половины длины словаря. Для таких случаем метрика точности accuracy не является устойчивой. Поэтому в качестве решающей метрики была выбрана f-мера. Это среднее гармоническое между точностью (precision) и полнотой (recall). Точность здесь - доля правильно определенных объектов класса среди всех объектов, отнесенных к классу. Полнота – доля истинных объектов класса, которую алгоритм смог правильно классифицировать, по отношению ко всем объектам этого класса. Иными словами, точность показывает, насколько точно мы классифицируем объекты, а полнота показывает – какая доля фактических событий этого класса была правильно предсказана. Ниже на рис. 3.5 представлена собирательная статистика по основным метрикам классификации для алгоритма KNN(n_neighbors=25, metric='euclidean', n_jobs=-1). Обучение и тестирование алгоритма проводилось с помощью пакета sklearn на языке программирования python 3.7 в среде Google Collaboratory.

	precision	recall	f1-score
PSTV	0.51	0.83	0.64
NEUT	0.90	0.81	0.85
NGTV	0.68	0.70	0.69
accuracy			0.79
macro avg	0.70	0.78	0.72
weighted avg	0.81	0.79	0.79

Рисунок 3.5. Результаты работа алгоритма.

В качестве примера рассмотрим отзыв к фильму «Способный ученик». Найденные прилагательные и существительные выделены.

« Начав свою карьеру еще в конце 1980-х, лишь к середине следующего десятилетия, в 1995 году, режиссер Брайан Сингер приобретает огромную популярность за успех ленты «Подозрительные лица». Брайан показал себя очень талантливым кинематографистом, который, не имея больших финансовых средств, может нагнетать напряжение и в обыкновенной разговорной сцене, что под силу мне кажется, далеко не всем режиссерам. Та лента, помимо неплохого кассового успеха (все-таки бюджет у фильма вышел очень маленький) получила также и две премии Оскар: лучший сценарий и Лучшая мужская роль второго плана (Кевин Спейси).

После такого успеха, Сингер решил продолжить работу в этом жанре и следующей своей работой он выбирает экранизацию очередной повести одного из самых экранизируемых писателей мира – Стивена Кинга. Сама же лента получилась, на мой взгляд, несколько неровной и чуток отторгающей. Во многом благодаря самому сюжету, который вышел довольно грубоватым и очень уж давящим, особенно учитывая такой непривычный так скажем, финал.

Главным минусом фильма лично для меня стало то, что напряжение здесь нагоняется вокруг вообщем-то обыденных и не особенно пугающих эпизодов. Да, напряжение есть, но сам сюжет почти весь первый час не особенно то и развивается, из-за чего попросту становится скучно. Ты жаждешь узнать, что там дальше будет, а вместо этого режиссер делает акцент на том, как актер сейчас справляется в школе и так далее. Но радует то, что все же дальше сюжет развивается по принципу снежного кома: каждое действие или опрометчивый поступок главного героя в конце-концов приводит его к мягко говоря, не самым предсказуемым последствиям.

Брэд Ренфро – потрясающе отыграл роль главного героя. Ученика, который жаждет чего-то большего, однако из-за постоянного общения о том бесчеловечном насилии, которое в свое время устроил Третий Рейх, он и сам наполняется ненавистью, которую со временем просто не сможет контролировать. Йен МакКеллен воистину зловещ в данной роли бывшего офицера Гитлеровской Германии. Его взгляд холоден, а действия пугающие. Именно такими мне кажется и были те самые убийцы во время Второй Мировой. Кстати, удивило присутствие Дэвида Швиммера в роли зачуханного учителя. Тогда этот актер был уже популярен из-за сериала «Друзья» и было странно видеть его в подобной роли.

Способный ученик – исключительно разговорный триллер, в котором первый час вообще практически ничего не происходит и взамен этого режиссер предлагает медленное и плавное раскрытие персонажей и погружение зрителя в атмосферу. Не скажу, что прошло крайне удачно, все-таки осечки были и нудноватых моментов хватало, но я думаю, фильм заслуживает хотя бы однократного просмотра. Насладитесь вполне себе неплохим триллером. Жаль только, лента не может предложить чего-то большего.

7 из 10».

Из недостатков сразу можно заметить не всегда верное соотнесение существительного и прилагательного, а также некоторые пары описывают действия самого фильма, не говоря в общем о фильме

ЗАКЛЮЧЕНИЕ

В результате проделанной работы удалось разработать алгоритм для анализа тональности текста, атрибутивной тональности текста. При этом точность классификации в обоих задачах составляет более 80%, что является неплохим результатом. В дальнейшем возможно расширение алгоритма для поиска не только существительных, но и глаголов и наречий. Так же возможно использовать более современные алгоритмы (трансформеры) и лингвистические пакеты.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Kaggle | Kinopoisk's movies reviews. – 2020. – URL: https://www.kaggle.com/mikhailklemin/kinopoisks-movies-reviews– (дата обращения: 08.06.2021).
2. Alex Sherstinsky, Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network – ArXiv, 2018.
3. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, Attention Is All You Need – ArXiv, 2018.
4. Github | Natasha. – 2019. – URL: https://github.com/natasha/natasha– (дата обращения: 10.06.2021).
5. Github | Aspect Based Sentiment Analysis. – 2020. – URL: https://github.com/ScalaConsultants/Aspect-Based-Sentiment-Analysis– (дата обращения: 10.06.2021).
6. Aspect-based Sentiment Analysis — Everything You Wanted to Know!. – 2020. – URL: https://intellica-ai.medium.com/aspect-based-sentiment-analysis-everything-you-wanted-to-know-1be41572e238– (дата обращения: 15.06.2021).
7. Github | Тональный словарь русского языка КартаСловСент. – 2021. – URL:https://github.com/dkulagin/kartaslov/tree/master/dataset/kartaslovsent – (дата обращения: 20.06.2021).

ПРИЛОЖЕНИЯ

1. Програмный код решения:

```
2.
3. # -*- coding: utf-8 -*-
4. """aspect_based_sent_analysis.ipynb
5.
6. Automatically generated by Colaboratory.
7.
8. Original file is located at
9.
   https://colab.research.google.com/drive/1GzNlatuzYOLo1w_vm8xvszze_jeZxPwr  - лучше использовать эту ссылку, там графики, картинки и текст.
   строчки, начинающиеся с ! - вводятся в командную строку
10.     """
11.
12.     from google.colab import drive
13.     drive.mount('/content/drive')
14.
15.     !pip install razdel
16.     !pip install navec
17.     !pip install slovnet
18.     !pip install ipymarkup
19.     !pip install pymorphy2
20.     !wget https://storage.yandexcloud.net/natasha-navec/packs/navec_hudlit_v1_12B_500K_300d_100q.tar
21.     !wget https://storage.yandexcloud.net/natasha-slovnet/packs/slovnet_syntax_news_v1.tar
22.     !wget https://storage.yandexcloud.net/natasha-navec/packs/navec_news_v1_1B_250K_300d_100q.tar
23.
24.     !wget https://github.com/dkulagin/kartaslov/raw/master/dataset/kartaslovsent/kartaslovsent.csv
25.
26.     !pip3 install colorama
27.
28.     import torch
29.     import torch.nn as nn
30.     import torch.nn.functional as F
31.     from torch.utils.data import Dataset, DataLoader
32.
33.     from razdel import tokenize, sentenize
34.     from navec import Navec
35.     from slovnet.model.emb import NavecEmbedding
36.     from slovnet import Syntax
37.
38.     import os
```

```
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt

    from tqdm.notebook import tqdm

    from sklearn.metrics import accuracy_score

    import pymorphy2

    from scipy.spatial.distance import cosine
    from sklearn.neighbors import  NearestNeighbors
    from sklearn.neighbors import KNeighborsClassifier as KNN

    from collections import defaultdict
    import re

    from json import dumps, load

    from colorama import Back


    np.random.seed(42)

    data = pd.read_csv(PATH)
    def count_words(row):
        return len(row.split())

    r = data['text'].apply(count_words)
    plt.hist(r, bins=350)
    plt.title('Распределение числа слов в отзывах.')
    plt.xlabel('Число слов')
    plt.xlim((0, 700))  # лимит на 700 словах, но максимум слов порядка 1000
    plt.show()

    path = '/content/navec_hudlit_v1_12B_500K_300d_100q.tar'
    navec = Navec.load(path)

    class TextDataset(Dataset):
        def __init__(self):
            self.targets = {"neg": 0, "neu": 1, "pos": 2}
            self.max_len = 700
            self.data = pd.read_csv(PATH).iloc[:70000]
            #self.data = self.data.filter(lambda x: len(x['text'].split()) > 50)

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

```

```
87.         def __getitem__(self, idx):
88.             sample, target = self.data.iloc[idx]
89.             sample = [_.text for _ in tokenize(sample)]
90.             if len(sample) > self.max_len:
91.               sample = sample[:self.max_len]
92.             sample = sample + ["<PAD>"]*(self.max_len-
   len(sample))
93.             sample = [navec.vocab.get(x, navec.vocab.unk_id) for
   x in sample]
94.             target = self.targets[target]
95.             return torch.LongTensor(sample),
   torch.LongTensor([target])
96.
97.     batch_size = 200
98.     dataset = TextDataset()
99.     train_size = int(0.8 * len(dataset))
100.    test_size = len(dataset) - train_size
101.    train_set, test_set = torch.utils.data.random_split(dataset,
   [train_size, test_size])
102.    train_iterator = DataLoader(train_set,
   batch_size=batch_size, shuffle=True)
103.    test_iterator = DataLoader(test_set, batch_size=batch_size)
104.    device = torch.device("cuda") if torch.cuda.is_available()
   else torch.device("cpu")
105.
106.    class RNN(nn.Module):
107.        def __init__(self, input_dim, embedding_dim, hidden_dim,
   output_dim, num_layers, dropout, bidirectional=False):
108.            super().__init__()
109.            self.num_layers = num_layers
110.            self.hidden_dim = hidden_dim
111.            self.embedding = NavecEmbedding(navec)
112.            self.bi = bidirectional
113.            self.rnn = nn.GRU(embedding_dim, hidden_dim, dropout
   = dropout, num_layers=self.num_layers, bidirectional=self.bi)
114.            self.fc = nn.Linear(hidden_dim*self.num_layers,
   output_dim)
115.            self.dropout = nn.Dropout(dropout)
116.            self.softmax = nn.Softmax(1)
117.
118.        def init_hidden(self, batch_size):
119.            mult = 2 if self.bi else 1
120.            return torch.zeros(mult * self.num_layers,
   batch_size, self.hidden_dim).to(device)
121.
122.        def forward(self, x):
123.            #text,shape = [sent len, batch size]
124.
125.            embedded = self.embedding(x)   # Делаем эмбеддинг
126.            #embedded.shape = [sent len, batch size, emb dim]
```

```
127.             #print(embedded.shape)
128.             output, self.hidden =  self.rnn.forward(embedded,
   self.hidden)   # Прогоняем через Rnn
129.             #hidden.shape = [1, batch size, hid dim]
130.             #print(self.hidden)
131.             x = self.fc(torch.cat((self.hidden[-2,:,:],
   self.hidden[-1,:,:]), dim = 1))
132.             x = self.dropout(x)
133.             return x.squeeze(0)
134.
135.     import torch.optim as optim
136.
137.     INP_DIM = 700
138.     EMB_DIM = 300
139.     HID_DIM = 512
140.     NUM_LAYERS = 2
141.     DROPOUT = 0.75
142.     OUT_DIM = 3
143.     N_EPOCH = 15
144.     BIDIRECTIONAL = True
145.
146.
147.     model = RNN(INP_DIM, EMB_DIM, HID_DIM, OUT_DIM, NUM_LAYERS,
   DROPOUT, BIDIRECTIONAL).to(device)
148.     criterion = nn.CrossEntropyLoss().to(device)
149.     optimizer = optim.Adam(model.parameters(), lr=0.001)
150.
151.     losses_train = []
152.     losses_test = []
153.
154.     for i in range(N_EPOCH):
155.       model.train()
156.       avg_loss = 0
157.       for j, (x, y) in tqdm(enumerate(train_iterator, 1),
   total=len(train_iterator), position=0, leave=True):
158.         y = y.squeeze(1).to(device)
159.         x = x.permute(1, 0).to(device)
160.         optimizer.zero_grad()
161.         model.hidden = model.init_hidden(x.size(1))
162.         output = model.forward(x)
163.         loss_train = criterion(output, y)
164.         loss_train.backward()
165.         avg_loss += loss_train.item()
166.         torch.nn.utils.clip_grad_norm_(model.parameters(), 0.2)
167.         optimizer.step()
168.       losses_train.append(avg_loss / j)
169.       avg_loss = 0
170.       model.eval()
171.       test_acc = []
```

```
172.       for j, (x, y) in tqdm(enumerate(test_iterator, 1),
   total=len(test_iterator), position=0, leave=True):
173.           y = y.squeeze(1).to(device)
174.           x = x.permute(1, 0).to(device)
175.           model.hidden = model.init_hidden(x.size(1))
176.           output = model.forward(x)
177.           test_acc.append(accuracy_score(torch.argmax(output,
   axis=1).cpu(), y.cpu()))
178.           loss_test = criterion(output, y)
179.           avg_loss += loss_test.item()
180.       losses_test.append(avg_loss / j)
181.
182.       print("\nepoch", i)
183.       print("train_loss", losses_train[-1])
184.       print("test_loss", losses_test[-1], ", test accuracy",
   np.array(test_acc).mean())
185.       print()
186.
187.     plt.plot(losses_train, label="train")
188.     plt.plot(losses_test, label ="test")
189.     plt.legend()
190.
191.     results = []
192.     model.eval()
193.     for x, y in train_iterator:
194.         y = y.squeeze(1).to(device)
195.         x = x.permute(1, 0).to(device)
196.         model.hidden = model.init_hidden(x.size(1))
197.         output = model.forward(x)
198.         results.append(accuracy_score(torch.argmax(output, axis
   = 1).cpu(), y.cpu()))
199.
200.     print(np.array(results).mean())
201.
202.     results = []
203.     model.eval()
204.     for x, y in test_iterator:
205.         y = y.squeeze(1).to(device)
206.         x = x.permute(1, 0).to(device)
207.         model.hidden = model.init_hidden(x.size(1))
208.         output = model.forward(x)
209.         results.append(accuracy_score(torch.argmax(output, axis
   = 1).cpu(), y.cpu()))
210.
211.     print(np.array(results).mean())
212.
213.     bi = "" if BIDIRECTIONAL else "non "
214.     #torch.save(model.state_dict(),
   f"{PATH_SAVE}navec_{EMB_DIM}_h_{HID_DIM}_GRU({bi}bi)_drop_{DROPOU
   T}_{N_EPOCH}_epochs.pth")
```

```
215.
216.     bi = "" if BIDIRECTIONAL else "non "
217.     model.load_state_dict(torch.load(f"{PATH_SAVE}navec_{EMB_DIM
   }_h_{HID_DIM}_GRU({bi}bi)_drop_{DROPOUT}_{N_EPOCH}_epochs.pth"))
218.     model.eval()
219.
220.     """## Анализ атрибутивной тональности и визуализация
221.
222.     ## kNN только по эталонным словам (словам из словаря
   тональности)
223.
224.     Возьмем проекцию на 2 измерения с помощью PCA
225.     """
226.
227.     emb = torch.Tensor([navec[vec] for vec in
   navec.vocab.words])
228.     words = navec.vocab.words
229.
230.     n_dim = 2
231.     u, s, vh = torch.pca_lowrank(emb)
232.     s[n_dim::] = 0
233.     emb_2d = torch.matmul(emb, vh[:, :n_dim])
234.
235.     tonal_dict = pd.read_csv("/content/kartaslovsent.csv",
   sep=";")
236.     tonal_dict.head()
237.
238.     plt.bar(tonal_dict.tag.unique(),
   tonal_dict.tag.value_counts())
239.
240.     morph = pymorphy2.MorphAnalyzer()
241.
242.     cnt = 0
243.     adjs = []
244.     for w in tqdm(words):
245.       m = morph.parse(w)
246.       if m[0].tag.POS in ["ADJF", "ADJS"]:
247.         adjs.append(cnt)
248.       cnt+=1
249.
250.     print(len(adjs))
251.     words = np.array(words)
252.     words_adjs = words[adjs]
253.     emb2d_adjs = emb_2d[adjs, :]
254.     emb_adjs = emb[adjs, :]
255.     emb2d_words_adjs = {word: emb for word,emb in
   zip(words_adjs, emb2d_adjs)}
256.     #words_emb2d_adjs = {emb: word for word,emb in
   zip(words_adjs, emb2d_adjs)}
```

```
    emb_words_adjs = {word: emb for word,emb in zip(words_adjs, emb_adjs)}
    #words_emb_adjs = {emb: word for word,emb in zip(words_adjs, emb_adjs)}

    standard_words = [word for word in tonal_dict.term if word in emb_words_adjs]
    print(f"Total standard words: {len(standard_words)}")

    import random
    random.shuffle(standard_words)
    # разделим на train и test
    standard_words_train = standard_words[:int(0.8*len(standard_words))]
    standard_words_test = standard_words[int(0.8*len(standard_words)):]

    classes = {"PSTV": 0, "NEUT": 1, "NGTV": 2}
    dict_pairs_train = dict(tonal_dict[["term", "tag"]][tonal_dict.term.isin(standard_words_train)].itertuples(index=False))
    X_train = np.array([emb_words_adjs[word].numpy() for word in dict_pairs_train])
    y_train = np.array([classes[dict_pairs_train[word]] for word in dict_pairs_train])

    dict_pairs_test = dict(tonal_dict[["term", "tag"]][tonal_dict.term.isin(standard_words_test)].itertuples(index=False))
    X_test = np.array([emb_words_adjs[word].numpy() for word in dict_pairs_test])
    y_test = np.array([classes[dict_pairs_test[word]] for word in dict_pairs_test])

    from sklearn.metrics import f1_score

    scores = []
    _n_neighbors = [5, 10, 20, 25, 30, 35, 40, 50, 55, 60]
    for n_neigh in _n_neighbors:
      print(n_neigh)
      estimator = KNN(n_neighbors=n_neigh, metric='euclidean', n_jobs=-1)
      estimator.fit(X_train, y_train)
      predicted = estimator.predict(X_test)
      scores.append(f1_score(predicted, y_test, average="micro"))

    plt.plot(_n_neighbors, scores)

```

```
    estimator = KNN(n_neighbors=25, metric='euclidean', n_jobs=-1)
    estimator.fit(X_train, y_train)

    from sklearn.externals import joblib
    joblib.dump(estimator, 'KNN.pkl')
    estimator  = joblib.load('KNN.pkl' , mmap_mode ='r')

    # from sklearn.naive_bayes import GaussianNB
    # estimator = GaussianNB()
    # estimator.fit(X_train, y_train)

    from sklearn.metrics import classification_report

    predicted = estimator.predict(X_test)
    print(classification_report(predicted, y_test, target_names=["PSTV", "NEUT", "NGTV"]))

    coords_2d = np.array([emb2d_words_adjs[word].numpy() for word in dict_pairs_test])

    plt.figure(figsize=(15, 15))
    plt.scatter(coords_2d[predicted == 0][:, 0], coords_2d[predicted == 0][:, 1], label="PSTV")
    # plt.scatter(coords_2d[predicted == 0][:, 0], coords_2d[predicted == 0][:, 1], label="NEUT")
    plt.scatter(coords_2d[predicted == 2][:, 0], coords_2d[predicted == 2][:, 1], label="NGTV")
    plt.legend()
    plt.show()

    """## Работа с настоящими отзывами"""

    navec_news = Navec.load('navec_news_v1_1B_250K_300d_100q.tar')
    syntax = Syntax.load('slovnet_syntax_news_v1.tar')
    syntax.navec(navec_news)

    chunk = []
    r = data.iloc[np.random.choice(range(len(data)))]["text"]
    rewiew = [_.text for _ in tokenize(r)]
    for sent in sentenize(r):
      tokens = [_.text for _ in tokenize(sent.text)]
      chunk.append(tokens)

    print(r)
```

```
    for markup in syntax.map(chunk):
      for token in markup.tokens:
          w = token.text
          m = morph.parse(w)
          if m[0].tag.POS in ["ADJF", "ADJS"]:
            n_f = m[0].normal_form
            if n_f in emb_words_adjs:
              e = emb_words_adjs[n_f]
              pred_class  = estimator.predict(e.reshape(1, -1))[0]
              pred_word_token = markup.tokens[int(token.head_id)-1]
              m = morph.parse(pred_word_token.text)
              if not m[0].tag.POS in ["NOUN", "NPRO"] or m[0].normal_form in ['который', 'свой', 'тот', 'этот', 'весь', 'такой', 'такого', 'какой-то', 'самый']:
                  continue
              # while True:
              #   m = morph.parse(pred_word_token.text)
              #   if m[0].tag.POS in ["NOUN", "NPRO"]:
              #     break
              #   if pred_word_token.head_id == '0' or pred_word_token.head_id == pred_id:
              #     break
              #   pred_id = pred_word_token.id
              #   pred_word_token = markup.tokens[int(pred_word_token.head_id)-1]

              template = pred_word_token.text
              if pred_class == 0:
                #print(f"{w}: proba_good-{proba_good}")
                print(f"{template} {Back.GREEN+w+Back.RESET}")
                pass
              elif pred_class == 2:
                #print(f"{w}: proba_bad-{proba_bad}")
                print(f"{template} {Back.RED+w+Back.RESET}")
```