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.

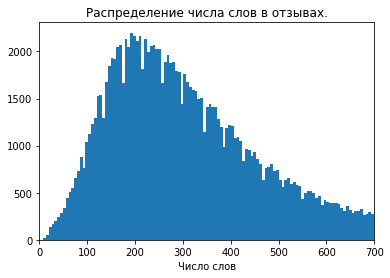


Рис 1.1. Распределение количества слов в отзывах.

1. **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).

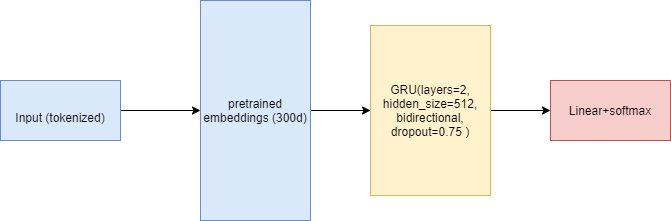


Рис. 2.1. Архитектура сети.

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:

где ( – истинная вероятность принадлежности к классу, -посчитанная вероятность принадлежности к 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.

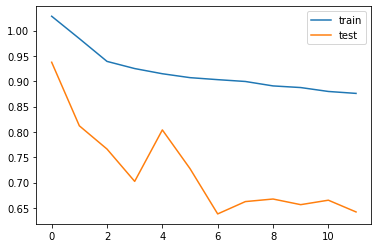


Рис. 2.2. Зависимость величины ошибки от эпохи обучения.

1. **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.

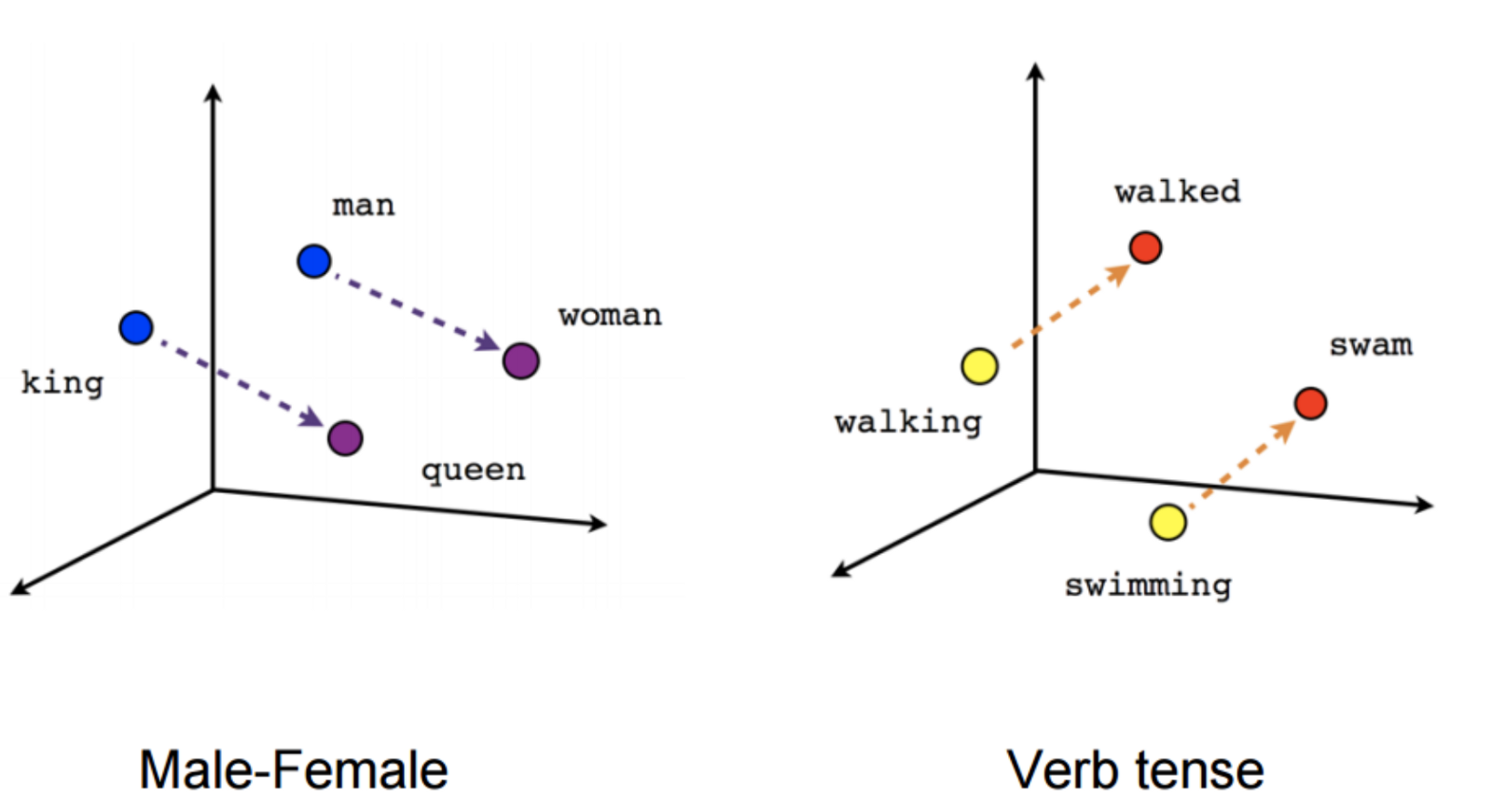


Рис. 3.1. Демонстрация линейных свойств word2vec.

**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.

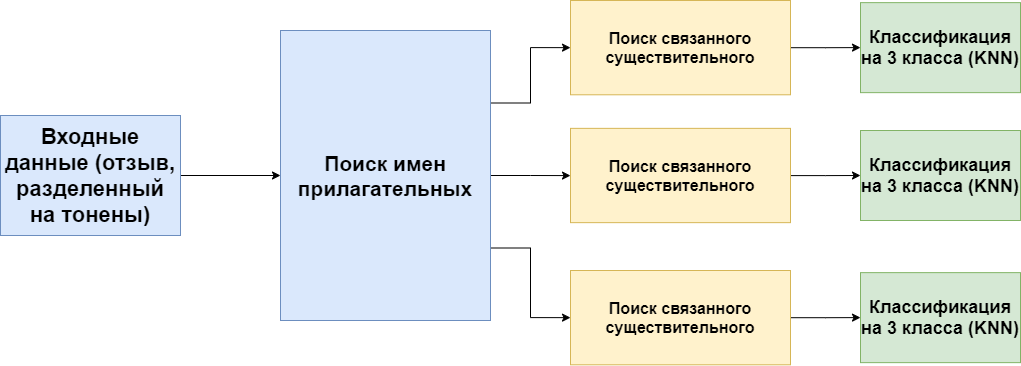


Рис. 3.2. Структура решения.

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.

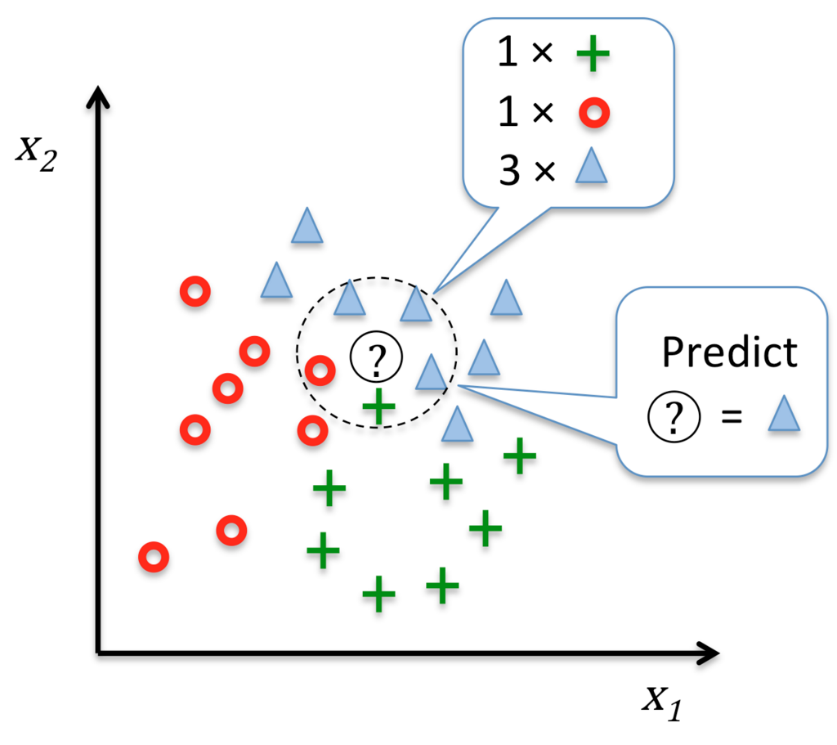


Рис. 3.3. Принцип работы алгоритма KNN.

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.

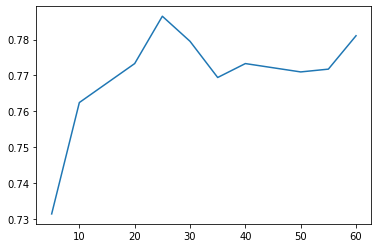


Рис. 3.4. Зависимость величины f-score от кол-ва соседей.

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.

Изображение выглядит как стол

Автоматически созданное описание

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](https://arxiv.org/search/cs?searchtype=author&query=Vaswani%2C+A), [Noam Shazeer](https://arxiv.org/search/cs?searchtype=author&query=Shazeer%2C+N), [Niki Parmar](https://arxiv.org/search/cs?searchtype=author&query=Parmar%2C+N), [Jakob Uszkoreit](https://arxiv.org/search/cs?searchtype=author&query=Uszkoreit%2C+J), [Llion Jones](https://arxiv.org/search/cs?searchtype=author&query=Jones%2C+L), [Aidan N. Gomez](https://arxiv.org/search/cs?searchtype=author&query=Gomez%2C+A+N), [Lukasz Kaiser](https://arxiv.org/search/cs?searchtype=author&query=Kaiser%2C+L), [Illia Polosukhin](https://arxiv.org/search/cs?searchtype=author&query=Polosukhin%2C+I), 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:
3. *# -\*- coding: utf-8 -\*-*
4. """aspect\_based\_sent\_analysis.ipynb
6. **Automatically generated by Colaboratory.**
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. """
12. **from** google.colab **import** drive
13. drive.mount('/content/drive')
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
24. !wget https://github.com/dkulagin/kartaslov/raw/master/dataset/kartaslovsent/kartaslovsent.csv
26. **!pip3 install colorama**
28. **import** torch
29. **import** torch.nn **as** nn
30. **import** torch.nn.functional **as** F
31. **from torch.utils.data import Dataset, DataLoader**
33. **from** razdel **import** tokenize, sentenize
34. **from** navec **import** Navec
35. **from** slovnet.model.emb **import** NavecEmbedding
36. **from slovnet import Syntax**
38. **import** os
39. **import** numpy **as** np
40. **import** pandas **as** pd
41. **import matplotlib.pyplot as plt**
43. **from** tqdm.notebook **import** tqdm
45. **from** sklearn.metrics **import** accuracy\_score
47. **import** pymorphy2
49. **from** scipy.spatial.distance **import** cosine
50. **from** sklearn.neighbors **import** NearestNeighbors
51. **from sklearn.neighbors import KNeighborsClassifier as KNN**
53. **from** collections **import** defaultdict
54. **import** re
56. **from json import dumps, load**
58. **from** colorama **import** Back

61. **np.random.seed(42)**
63. data = pd.read\_csv(PATH)
64. **def** count\_words(row):
65. **return** len(row.split())
67. r = data['text'].apply(count\_words)
68. plt.hist(r, bins=350)
69. plt.title('Распределение числа слов в отзывах.')
70. plt.xlabel('Число слов')
71. **plt.xlim((0, 700)) *# лимит на 700 словах, но максимум слов порядка 1000***
72. plt.show()
74. path = '/content/navec\_hudlit\_v1\_12B\_500K\_300d\_100q.tar'
75. navec = Navec.load(path)
77. **class** TextDataset(Dataset):
78. **def** \_\_init\_\_(self):
79. self.targets = {"neg": 0, "neu": 1, "pos": 2}
80. self.max\_len = 700
81. **self.data = pd.read\_csv(PATH).iloc[:70000]**
82. *#self.data = self.data.filter(lambda x: len(x['text'].split()) > 50)*
84. **def** \_\_len\_\_(self):
85. **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])
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")
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)**
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)
122. **def** forward(self, x):
123. *#text,shape = [sent len, batch size]*
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)
135. **import** torch.optim **as** optim
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

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)
151. **losses\_train = []**
152. losses\_test = []
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)
182. **print**("**\n**epoch", i)
183. **print**("train\_loss", losses\_train[-1])
184. **print**("test\_loss", losses\_test[-1], ", test accuracy", np.array(test\_acc).mean())
185. **print**()
187. plt.plot(losses\_train, label="train")
188. plt.plot(losses\_test, label ="test")
189. plt.legend()
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()))
200. **print**(np.array(results).mean())
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()))
211. **print(np.array(results).mean())**
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\_{DROPOUT}\_{N\_EPOCH}\_epochs.pth")*
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()
220. """## Анализ атрибутивной тональности и визуализация
222. ## kNN только по эталонным словам (словам из словаря тональности)
224. Возьмем проекцию на 2 измерения с помощью PCA
225. """
227. emb = torch.Tensor([navec[vec] **for** vec **in** navec.vocab.words])
228. words = navec.vocab.words
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])
235. tonal\_dict = pd.read\_csv("/content/kartaslovsent.csv", sep=";")
236. **tonal\_dict.head()**
238. plt.bar(tonal\_dict.tag.unique(), tonal\_dict.tag.value\_counts())
240. morph = pymorphy2.MorphAnalyzer()
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
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)}***
257. emb\_words\_adjs = {word: emb **for** word,emb **in** zip(words\_adjs, emb\_adjs)}
258. *#words\_emb\_adjs = {emb: word for word,emb in zip(words\_adjs, emb\_adjs)}*
260. standard\_words = [word **for** word **in** tonal\_dict.term **if** word **in** emb\_words\_adjs]
261. **print(f"Total standard words: {len(standard\_words)}")**
263. **import** random
264. random.shuffle(standard\_words)
265. *# разделим на train и test*
266. **standard\_words\_train = standard\_words[:int(0.8\*len(standard\_words))]**
267. standard\_words\_test = standard\_words[int(0.8\*len(standard\_words)):]
269. classes = {"PSTV": 0, "NEUT": 1, "NGTV": 2}
270. dict\_pairs\_train = dict(tonal\_dict[["term", "tag"]][tonal\_dict.term.isin(standard\_words\_train)].itertuples(index=False))
271. **X\_train = np.array([emb\_words\_adjs[word].numpy() for word in dict\_pairs\_train])**
272. y\_train = np.array([classes[dict\_pairs\_train[word]] **for** word **in** dict\_pairs\_train])
274. dict\_pairs\_test = dict(tonal\_dict[["term", "tag"]][tonal\_dict.term.isin(standard\_words\_test)].itertuples(index=False))
275. X\_test = np.array([emb\_words\_adjs[word].numpy() **for** word **in** dict\_pairs\_test])
276. **y\_test = np.array([classes[dict\_pairs\_test[word]] for word in dict\_pairs\_test])**
278. **from** sklearn.metrics **import** f1\_score
280. scores = []
281. **\_n\_neighbors = [5, 10, 20, 25, 30, 35, 40, 50, 55, 60]**
282. **for** n\_neigh **in** \_n\_neighbors:
283. **print**(n\_neigh)
284. estimator = KNN(n\_neighbors=n\_neigh, metric='euclidean', n\_jobs=-1)
285. estimator.fit(X\_train, y\_train)
286. **predicted = estimator.predict(X\_test)**
287. scores.append(f1\_score(predicted, y\_test, average="micro"))
289. plt.plot(\_n\_neighbors, scores)
291. **estimator = KNN(n\_neighbors=25, metric='euclidean', n\_jobs=-1)**
292. estimator.fit(X\_train, y\_train)
294. **from** sklearn.externals **import** joblib
295. joblib.dump(estimator, 'KNN.pkl')
296. **estimator = joblib.load('KNN.pkl' , mmap\_mode ='r')**
298. *# from sklearn.naive\_bayes import GaussianNB*
299. *# estimator = GaussianNB()*
300. *# estimator.fit(X\_train, y\_train)*
302. **from** sklearn.metrics **import** classification\_report
304. predicted = estimator.predict(X\_test)
305. **print**(classification\_report(predicted, y\_test, target\_names=["PSTV", "NEUT", "NGTV"]))
307. coords\_2d = np.array([emb2d\_words\_adjs[word].numpy() **for** word **in** dict\_pairs\_test])
309. plt.figure(figsize=(15, 15))
310. plt.scatter(coords\_2d[predicted == 0][:, 0], coords\_2d[predicted == 0][:, 1], label="PSTV")
311. ***# plt.scatter(coords\_2d[predicted == 0][:, 0], coords\_2d[predicted == 0][:, 1], label="NEUT")***
312. plt.scatter(coords\_2d[predicted == 2][:, 0], coords\_2d[predicted == 2][:, 1], label="NGTV")
313. plt.legend()
314. plt.show()
316. **"""## Работа с настоящими отзывами"""**
318. navec\_news = Navec.load('navec\_news\_v1\_1B\_250K\_300d\_100q.tar')
319. syntax = Syntax.load('slovnet\_syntax\_news\_v1.tar')
320. syntax.navec(navec\_news)
322. chunk = []
323. r = data.iloc[np.random.choice(range(len(data)))]["text"]
324. rewiew = [\_.text **for** \_ **in** tokenize(r)]
325. **for** sent **in** sentenize(r):
326. **tokens = [\_.text for \_ in tokenize(sent.text)]**
327. chunk.append(tokens)
329. **print**(r)



334. **for** markup **in** syntax.map(chunk):
335. **for** token **in** markup.tokens:
336. **w = token.text**
337. m = morph.parse(w)
338. **if** m[0].tag.POS **in** ["ADJF", "ADJS"]:
339. n\_f = m[0].normal\_form
340. **if** n\_f **in** emb\_words\_adjs:
341. **e = emb\_words\_adjs[n\_f]**
342. pred\_class = estimator.predict(e.reshape(1, -1))[0]
343. pred\_word\_token = markup.tokens[int(token.head\_id)-1]
344. m = morph.parse(pred\_word\_token.text)
345. **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']:
346. **continue**
347. *# while True:*
348. *# m = morph.parse(pred\_word\_token.text)*
349. *# if m[0].tag.POS in ["NOUN", "NPRO"]:*
350. *# break*
351. ***# if pred\_word\_token.head\_id == '0' or pred\_word\_token.head\_id == pred\_id:***
352. *# break*
353. *# pred\_id = pred\_word\_token.id*
354. *# pred\_word\_token = markup.tokens[int(pred\_word\_token.head\_id)-1]*
356. **template = pred\_word\_token.text**
357. **if** pred\_class == 0:
358. *#print(f"{w}: proba\_good-{proba\_good}")*
359. **print**(f"{template} {Back.GREEN+w+Back.RESET}")
360. **pass**
361. **elif pred\_class == 2:**
362. *#print(f"{w}: proba\_bad-{proba\_bad}")*
363. **print**(f"{template} {Back.RED+w+Back.RESET}")