Оглавление

Введение

Основной задачей, с которой и началось развитие обработки естественного языка, является машинный перевод. В 1933 году советский ученый Петр Троянский представил Академии Наук СССР машину для подбора и печатания слов. В 1966 году американский комитет ALPAC, подводя итоги десятилетних исследований американских ученых в данной сфере, опубликовал отчет, в котором машинный перевод назывался дорогим, неточным и бесперспективным. Два десятилетия спустя происходит революция в обработке естественного языка, и на смену сложным наборам рукописных лингвистических правил приходят алгоритмы машинного обучения. В 2010 году к проблеме машинного перевода подключают нейронные сети, и теперь, в наше время, высокое качество и доступность онлайн-переводчиков не кажется чем-то необычным.

Однако машинный перевод – не единственная задача, требующая обработки естественного языка. Сюда же относятся классификация текстов, анализ тональности, извлечение именованных сущностей, создание вопросно-ответных систем и т.д. Однако качество решения данных задач напрямую зависит от того, насколько хорошо был обработан исходный текст и были ли извлечены из него все полезные признаки.

Цель работы: на примере конкретного датасета продемонстрировать способы предварительного анализа текстовых данных, а также с помощью простой нейронной сети сравнить различные методы извлечения признаков.

Объектом изучения являются задачи обработки текстов на естественных языках.

Предмет анализа: методы анализа и извлечения признаков.

1. Теоретическая справка

1.1. Обработка машинного языка

Обработка машинного языка (более популярным является английский вариант термина – Natural language processing, или NLP) – это область лингвистики, информатики и искусственного интеллекта, которая связана с взаимодействием между компьютерами и человеческим (естественным) языком.

Сейчас NLP представляет собой огромное количество задач разного уровня. Приведём некоторые примеры, разбив задачи по уровню детализации:

- Уровень сигнала – это преобразование речи, рукописных записей или печатного отсканированного текста;
- Уровень слова – морфологический анализ, канонизация, исправление ошибок;
- Уровень словосочетаний – определение части речи, распознавание именных сущностей;
- Уровень предложений – поиск ответа на вопрос;
- Уровень абзацев – извлечение отношений, анализ эмоциональной окраски, определение языка;
- Уровень документа – автоматическая аннотация, перевод, определение тематики, генерация документа.

В задачах NLP используются такие понятия, как токен, документ и корпус. Токен – это основная рассматриваемая единица, обычно буква (например, в задаче определения национальности по фамилии) или слово. Документ – это совокупность токенов: для токена-слова документов будет слово, для токена-слова – предложение и т.д. Следовательно, корпус – это совокупность документов.

Решение задач NLP, как и любой другой задачи машинного бучения, требует предварительного анализа данных и извлечения из них признаков, которые и будет изучать модель.

1.2. Предварительный анализ текстовых данных

Первый шаг в анализе текстовых данных, как и при анализе числовых – это поиск пропущенных значений и дубликатов. Основная проблема со строками заключается в том, что отсутствующее значение не всегда будет обозначено общепринятыми на языках программирования аббревиатурой NaN. Например, если исходные данные были собраны с сайта или блога, то они могут содержать удаленные комментарии, обозначенные каким-то специальным словом. Эти данные не несут смысловой нагрузки для модели, и поэтому подлежат удалению. Дубликаты же не всегда удаляются: основное опасение в наличии заключается в том, что если один и тот же пример попадет и в обучающую выборку, и в тестовую, то оценка результатов будет не совсем справедливой.

Поскольку при дальнейшей работе с текстовыми данными потребуется создание словаря, то особое значение имеет приведение к нижнему регистру (в противном случае «король» и «Король» будут считаться разными словами), очистка от знаков препинания и стоп-слов. Если рассматриваемый токен – буква латинского алфавита, то размер словаря составит 23 токена. Однако если за основу берутся слова и прочие символы, то потребуется немало памяти только лишь для его сохранения. По этой причине избавляются от знаков препинания и смайликов, которые зачастую встречаются в текстах, собранных из Интернета. Стоп-словами называют слова, которые встречаются в большинстве текстов и не несут информативной нагрузки. Это предлоги, артикли, глаголы-связки и т.д.

При работе с очень большими корпусами даже после такой очистки словарь может казаться больше, чем хотелось бы. В таком случае переходят к стеммингу: поиску основы (или корня) слова. Однако обрезание приставок, суффиксов и окончаний иногда приводит к ухудшению модели, так как мы лишаемся части информации. В данном случае все зависит от конкретной решаемой задачи.

Другой подход к этой проблеме: лемматизация, то есть проведение слова к его словарной форме. Для этого сначала требуется на основе контекста слова определить его часть речи, чтобы избежать путаницы.

Описанные выше шаги являются базовыми в предварительном анализе текстовых данных, однако какие из них стоит применить, к какие можно опустить, решает сам исследователь. Различные задачи обработки естественного языка и даже различные датасеты требуют совершено разного подхода, и иногда для несложных примеров (или если используются модели глубокого обучения) достаточно только очистить данные от знаков препинания и стоп-слов.

После того, как текстовые данные были проверены и очищены, производится извлечение признаков. Большинство моделей машинного обучения принимают на вход только числовые последовательности фиксированного размера, в связи с чем возникает вопрос кодирования текста.

1.3. Извлечение признаков

1.3.1. Bag of Words

Самым простым и очевидным способом представления текстовых данных в числовом виде является так называемый мешок слов (Bag of words, BoW). Метод очень похож на one-hot кодирование: корпус представляется как матрица, где количество строк – это документы, а длина каждой строки – число уникальных токенов. Создается словарь вида «токен: индекс», где каждому токену присвоено уникальное целое число. Если рассматриваемый токен встречается в документе, то в этой строчке ставится 1 на позиции, соответствующей индексу токена.

Данный метод хорош тем, что он легко реализуем и просто в понимании, но подходит он только для небольших корпусов и для очень простых задач. Очевидно, что мешок слов сильно зависит от количества уникальных токенов, и если, например, мы рассматриваем 1 000 документов, в которых 100 000 уникальных токенов, то в результате получится матрица 1 000 на 100 000,

которая будет занимать много места в памяти и с которой будет трудно оперировать.

Еще один недостаток метода: он работает только как счетчик и не учитывает «важность» токена и его контекст. В связи с этом был разработан другой метод - TF-IDF.

1.3.2. TF-IDF

Этот метод является комбинацией двух, которые и заложены в аббревиатуре.

TF — это term frequency, частота слова. Отражает отношение числа вхождений токена в документ к числу всех токенов в документе. Иначе говоря, TF оценивает важность токена в пределах одного документа.

IDF — inverse document frequency, обратная частота документа. Обратное значение частоты, с которой токен встречается в корпусе. Чем чаще токен встречается в корпусе, тем меньше будет его вес.

IF-IDF – это ничто иное как произведение этих двух показателей.

Данный метод решает проблему «важности» токена, позволяя отделять мало встречаемые токены, но сохраняет проблему размера словаря. Еще одно достоинство такого представления текста: возможность сравнивать документы с помощью метрик. Обычно метод используют в задачах определения тональности текста или в задаче информационного поиска.

1.3.3. N-gramms

В методах, приведённых выше, рассматриваются отдельные токены, однако иногда куда большей полезностью будут обладать пары или тройки токенов. N-грамма – это последовательность из n токенов, идущих в документе подряд. При n=2 обычно говорят, что рассматриваются биграммы, при n=3 – триграммы. После разделения текста на n-граммы используется уже описанные BoW или TF-IDF.

Такой подход позволяет определить устойчивые фразы, тренды. Обычно используется в задачах подсказки следующего слова, определения авторства (так как помогает выявить авторский стиль), плагиата, поиска ошибок.

К сожалению, этот метод также имеет недостаток: размерность словаря, который теперь будет больше количества уникальных токенов из-за того, что один и тот же токен рассматривается с соседями с двух сторон.

1.3.4. Word2Vec

Этот метод является наиболее продвинутым для извлечения признаков из текста. Использует нейронную сеть для изучения корпуса и обнаружения словесных ассоциаций. Преимущество данного метода в том, что при обучении модель подбирает вектора таким образом, что похожие по смыслу слова будут иметь похожие численные представления.

Существуют два варианта этого метода: на основе CBoW и на основе skip-gram. В первом случае модель учится предсказывать слово в зависимости от его контекста, во втором модель действует ровно наоборот: предсказывается близлежащее слово на основе одного.

В данном методе важную роль играют такие гиперпараметры, как ширина окна, размер вектора и минимальная частотность токена в корпусе. Последние два нужны для настраивания размера словаря и позволяют не учитывать редко встречаемые слова. Ширина окна позволяет подстраиваться под конкретные данные, так как иногда достаточно информации о двух ближайших соседях, а иногда для полноты контекста требуется большее расстояние.

Метод широко используется для автоматической разметки текстов и машинного перевода.

Тем не менее, этот метод также имеет недостаток: для обучения модели нужно иметь огромный корпус слов, в противном случае модель просто не сможет дать качественный результат. В том случае, если данных мало, то можно воспользоваться любой из уже предобученных моделей, лежащих в открытом

доступе. Например, обученные на датасете новостей Гугл, в котором содержится больше 100 миллиардов слов, или на Википедии.

Библиотека gensin предлагает простую и удобную реализацию метода word2vec (с возможностью выбирать между CBoW и Skip-gram), а также позволяет использовать уже предобученные модели.

2. Предварительный анализ данных

2.1. Выбор модели, метрик и анализ исходного датасета

Поскольку задач обработки естественного языка довольно много, и каждая из них требует своего подхода, остановимся на одной и подробно рассмотрим все возникающие проблемы на конкретном датасете.

В качестве задачи NLP была выбрана задача классификации текстов.

Датасет: комментарии с сайта reddit.com, относящиеся к биологии, физике или химии. Данные были собраны пользователем VIVEK и опубликованы на сайте Kaggle.com.

Для сравнения результатов воспользуемся простой нейронной моделью, состоящей из одного слоя. В каждом случае зададим параметры размер батча 32, количество эпох 10. Оптимизатор – Adam; для вычисления ошибки возьмем sparse categorical crossentropy, которая рассчитывает кросс-энтропийную потерю между предсказанной меткой и истинной и подходит для мультиклассовых задач. Реализация нейронной сети выполнена с помощью библиотек Keras и TensorFlow.

Строго говоря, каждый из описанных в предыдущем разделе метод извлечения признаков будет работать лучше или хуже для разных архитектур моделей. Так, например, может понадобиться добавление еще одного слоя, нормализация, релуляризация или свертка. Будем исходить из предположения, что нам подходит только очень простая модель классификации, и посмотрим, какие результаты дадут различные методы. Кроме этого, будем сравнивать результаты для очищенного и исходного датасета.

Сравнение будем проводить по четырем метрикам классификации:

- Accuracy – качество модели;
- Precision – способность модели отличать классы;
- Recall – как полно модель определяет класс;
- F1 – среднее гармоническое между precision и recall.

В итоговую таблицу будем заносить взвешенные средние этих метрик - рассчитанные с учетом количества истинных меток в каждом классе. Также будем замерять время обучения модели.

Перейдем непосредственно к анализу датасета.

Таблица 1. *Первые пять строчек датасета*

	Id	Comment	Topic
0	0x840	A few things. You might have negative- frequen...	Biology
1	0xbf0	Is it so hard to believe that there exist part...	Physics
2	0x1dfc	There are bees	Biology
3	0xc7e	I'm a medication technician. And that's alot o...	Biology
4	0xbba	Cesium is such a pretty metal.	Chemistry

Датасет содержит 8695 объектов и всего три столбца: ID, комментарий и тему (топик). Последний столбец является нашей целевой переменной. Выведем информацию о пропусках и уникальных значениях.

Таблица 2. *Описательная таблица*

	Тип	Количество Nan	Количество уникальных	Уникальные значения
Id	object	0	8695	[0x840, 0xbf0, 0x1dfc, 0xc7e, 0xbba, 0xb39, 0x...
Comment	object	0	7950	[A few things. You might have negative- freque...
Topic	object	0	3	[Biology, Physics, Chemistry]

На первый взгляд, в датасете нет пропусков, однако надо учитывать, что используемый метод isnull() ищет объекты типа NaN для числовых массивов, None или NaN для данных типа object и NaT для данных типа дата-время. Однако владелец датасета мог использовать какие-то другие обозначения для

указания на пропущенные значения. Например, поставить нули или отметку «удалено» для комментариев, которые позже были убраны комментатором или модератором сайта. В описании к датасету нет информации на этот счет, поэтому эта проблема требует отдельного рассмотрения.

Кроме того, можно сразу заметить, что столбец ID состоит только из уникальных значений. Эти данные не понадобятся для обучения моделей, а за идентификационный номер объекта можно рассматривать индекс строчки в таблице. Удалим этот столбец.

Вывод уникальных значений по столбцу Topic позволяет определить, что целевая переменная действительно может принимать только три значения. В общем случае возможны ситуации, когда название категории в некоторых случаях написана с опечаткой или с прописной буквы. Если это вовремя не заметить и не исправить, то ‘Biology’ и ‘biology’ будут считаться за две разные категории, что приведет к ошибкам.

Прежде чем переходить к выявлению и устранению возможных проблем, посмотрим, является ли наша выборка сбалансированной.

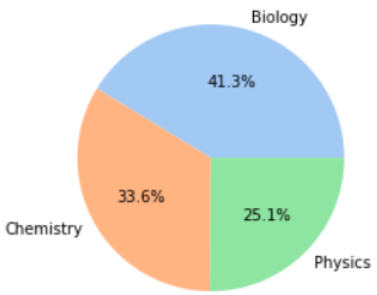

Рис 1. *Доля наблюдений в классах*

При идеальном раскладе в случае трех категорий каждая из них должна занимать примерно 33,3% из датасета. У нас наблюдается небольшая разбалансировка. Будем учитывать этот факт в дальнейшей работе: во-первых, учтем это при разбиении выборки на обучающую и тестовую, во-вторых, будем

помнить, что accuracy не сможет достоверно отобразить качество на нашем датасете, если один класс будет доминировать над другими.

Теперь посмотрим, есть ли в датасете строчки-дубликаты.

Таблица 3. *Дубликаты*

	Comment	Topic
84	[removed]	Physics
86	[removed]	Physics
108	[removed]	Physics
115	[removed]	Physics
176	[removed]	Physics
...	...	...
8669	Thank you!	Chemistry
8681	Nice	Chemistry
8685	Thank you for your kindness <3 Found this [inj...	Biology
8690	I make similar observations over the last week...	Biology
8693	What about the ethical delimmas, groundbreaki...	Biology

723 rows × 2 columns

Наблюдаем 723 дубликата (первое вхождение строки с тем же наполнением не учитывается). Поиск дубликатов сразу же решил проблему с удаленными комментариями: теперь мы знаем, что в датасете они отмечены как [removed]. Кроме них в выборку попали не несущие смысловой нагрузки комментарии-ответы, например благодарность за ответ. Отсюда появляется новая проблема: необходимо как-то оценить комментарии на пригодность для использования в задачи классификации текстов по предметным категориям. Скорее всего здесь поможет очищение от стоп-слов и различные методы визуализации данных. В довершении всего обнаруживаются полностью совпадающие комментарии.

Проведем следующие операции:

- Удалим все дубликаты;
- Удалим комментарии, состоящие только из строки [removed];

Количество строк сократилось до 7970-ти. Теперь 38,3% датасета занимают комментарии по теме «биология», 36% - «химия», оставшиеся 25,6% - «физика».

Хорошим средством для визуализации часто встречающихся слов является инструмент «облако слов»: чем чаще слово встречается во всем тексте, тем больше оно на картинке.

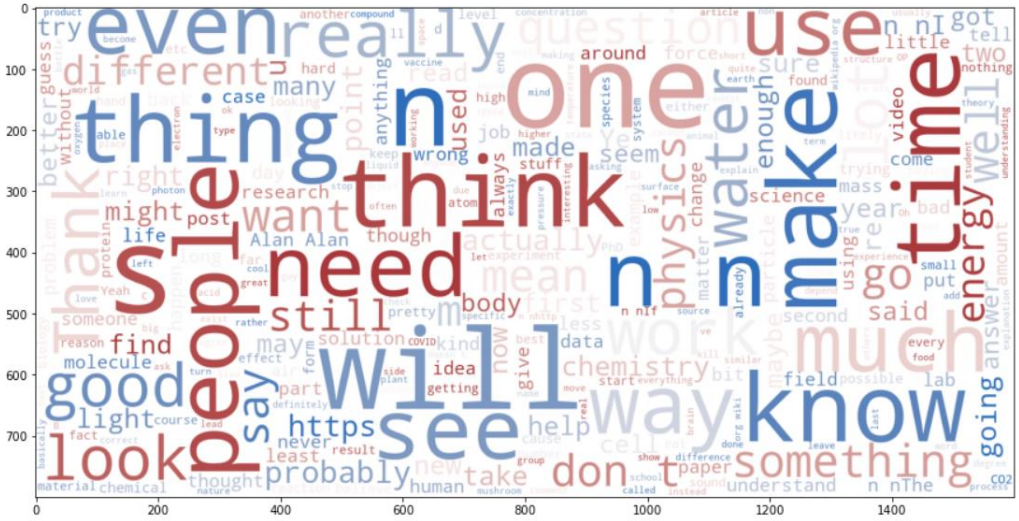

Рис 2. *Облако слов*

Так как датасет еще не очищен, то мы можем видеть преобладание стоп-слов (например, «don`t»). Помимо них заметны слова, которые могут стать ориентирами для классификации по темам: «вода», «клетка», «человек», «энергия», «физика» и др. В нижнем правом углу заметна формула «CO2». Возьмем на заметку, что в датасете могут быть и более сложные формулы – следовательно, и математические символы.

Приведем небольшую статистику по комментариям.

Таблица 4. *Статистика по комментариям*

Средняя длина комментария	35.10
Длина самого короткого комментария	1.00
Длина самого длинного комментария	1481.00

2.2. Преобразование и очистка датасета

Следующий этап при работе с текстами – приведение к нижнему регистру и очистка от знаков пунктуации и странных символов. В этом могут помочь регулярные выражения и строковая константа string.punctuation, содержащая

популярные знаки, но даже они не гарантируют 100% очистку. Так как у нас не очень большой датасет, то эту проблему можно решить вручную, выведя список уникальных символов и внеся в список для удаления те, что могут быть бесполезны для задачи и только усложнят извлечение признаов. Выведем список символов в тексте, очищенном от популярных знаков.

```
Исходный текст:

A few things. You might have negative- frequency dependent selection going on where the least common phenotype, reflected by ge
notype, is going to have an advantage in the environment. For instance, if a prey animal such as a vole were to have a light an
d a dark phenotype, a predator might recognize the more common phenotype as food.  So if the light voles are more common, foxes
may be keeping a closer eye out for light phenotypic voles, recognising them as good prey. This would reduce the light causing
alleles due to increased predation and the dark genotypes would increase their proportion of the population until this scenario
is reversed. This cycle continues perpetually. \n\nHowever, this is unlikely to be strictly yearly as it usually takes more tim
e than a year for an entire populations allele frequencies to change enough to make a large enough difference to alter fitness.
\n\nMore likely on a *year to year* basis, the population is experiencing fluctuating selection where alternating conditions in
the environment favor one genotype over another. Perhaps a plant species is living in an area that is flooded every other year
and the two phenotypes in the population are plants that do much better in the dryer year and one that does better in the wet y
ear. If there is no flooding, the dry-type genotype will have more fitness leading to more offspring and therefore more dry all
eles in the population, however, in flooded years the wet-liking phenotype will do better and propagate the wet genes.


Очищенный текст:

a few things  you might have negative frequency dependent selection going on where the least common phenotype  reflected by gen
otype  is going to have an advantage in the environment  for instance  if a prey animal such as a vole were to have a light and
a dark phenotype  a predator might recognize the more common phenotype as food  so if the light voles are more common  foxes m
ay be keeping a closer eye out for light phenotypic voles  recognising them as good prey  this would reduce the light causing a
lleles due to increased predation and the dark genotypes would increase their proportion of the population until this scenario
is reversed  this cycle continues perpetually  nnhowever  this is unlikely to be strictly yearly as it usually takes more time
than a year for an entire populations allele frequencies to change enough to make a large enough difference to alter fitness  n
nmore likely on a year to year basis  the population is experiencing fluctuating selection where alternating conditions in the
environment favor one genotype over another  perhaps a plant species is living in an area that is flooded every other year and
the two phenotypes in the population are plants that do much better in the dryer year and one that does better in the wet year
if there is no flooding  the drytype genotype will have more fitness leading to more offspring and therefore more dry alleles i
n the population  however  in flooded years the wetliking phenotype will do better and propagate the wet genes
```

```python
all_symbols = []
for index, row in data.iterrows():
    for symb in list(row.text_wo_punkt):
        if symb not in all_symbols:
            all_symbols.append(symb)
print(all_symbols)
```

```
['a', ' ', 'f', 'e', 'w', 't', 'h', 'i', 'n', 'g', 's', 'y', 'o', 'u', 'm', 'v', 'r', 'q', 'c', 'd', 'p', 'l', 'b', 'k', 'z',
'x', 'j', 'ɜ', '0', '2', '4', '8', '1', '5', '7', '6', '9', 'ø', 'µ', 'μ', 'ñ', 'é', 'ç', 'ä', 'õ', 'ö', 'ó', 'á', '₂', '₃',
'å', 'ν', 'ã', 'í', '²', 'ê', 'ψ', 'ч', 'е', 'р', 'н', 'о', 'б', 'ы', 'л', 'ь', '³', 'ú', '⁷', 'ℝ', '⅞', 'γ', 'ф', 'φ', 'è',
'ü', 'æ', 'ر' ,'ب' ,'ے' ,'ہ' ,'ن' ,'ا']
```

Рис 3. *Пример очищенного текста и оставшиеся уникальные символы*

Помимо латинских букв и арабских цифр можно наблюдать буквы из других алфавитов (русские буквы, выведенные в порядке встречаемости в текстах, и вовсе складываются в слово «чернобыль»), а также математические символы.

В задаче классификации текстов по темам токеном является слово, и если мы оставим эти символы (и неанглийские слова), то скорее всего они никак не повлияют на работу модели. Чернобыльская катастрофа явно приводилась как пример, и скорее всего в датасете больше нет упоминаний о Чернобыле. При

преобразовании текстов в вектора странные символы и буквы будут отброшены из-за малой частоты встречаемости. Не будем их удалять.

Переходим к этапу очистки от стоп-слов. Загрузим список таких слов из библиотеки NLTK. Так как мы уже удалили знаки препинания, а стоп-слова содержат апострофы, добавим в перечень стоп-слов слова, где апострофы заменены на пробелы и слова, где апостроф заменили на пустую строчку. Таким образом, вместо ‘don`t’ мы получим ‘don t’ и ‘dont’.

```
Исходный текст:

A few things. You might have negative- frequency dependent selection going on where the least common phenotype, reflected by ge
notype, is going to have an advantage in the environment. For instance, if a prey animal such as a vole were to have a light an
d a dark phenotype, a predator might recognize the more common phenotype as food.  So if the light voles are more common, foxes
may be keeping a closer eye out for light phenotypic voles, recognising them as good prey. This would reduce the light causing
alleles due to increased predation and the dark genotypes would increase their proportion of the population until this scenario
is reversed. This cycle continues perpetually. \n\nHowever, this is unlikely to be strictly yearly as it usually takes more tim
e than a year for an entire populations allele frequencies to change enough to make a large enough difference to alter fitness.
\n\nMore likely on a *year to year* basis, the population is experiencing fluctuating selection where alternating conditions in
the environment favor one genotype over another. Perhaps a plant species is living in an area that is flooded every other year
and the two phenotypes in the population are plants that do much better in the dryer year and one that does better in the wet y
ear. If there is no flooding, the dry-type genotype will have more fitness leading to more offspring and therefore more dry all
eles in the population, however, in flooded years the wet-liking phenotype will do better and propagate the wet genes.


Очищенный текст:

things might negative frequency dependent selection going least common phenotype reflected genotype going advantage environment
instance prey animal vole light dark phenotype predator might recognize common phenotype food light voles common foxes may keep
ing closer eye light phenotypic voles recognising good prey would reduce light causing alleles due increased predation dark gen
otypes would increase proportion population scenario reversed cycle continues perpetually nnhowever unlikely strictly yearly us
ually takes time year entire populations allele frequencies change enough make large enough difference alter fitness nnmore lik
ely year year basis population experiencing fluctuating selection alternating conditions environment favor one genotype another
perhaps plant species living area flooded every year two phenotypes population plants much better dryer year one better wet yea
r flooding drytype genotype fitness leading offspring therefore dry alleles population however flooded years wetliking phenotyp
e better propagate wet genes
```

Рис 4. *Пример очищенного текста*

Текст стал заметно меньше. Проверим, не получилось ли так, что после удаления стоп-слов комментарии стали пустыми строками.

Таблица 5. *Статистика по комментариям*

Средняя длина комментария	15.98
Длина самого короткого комментария	0.00
Длина самого длинного комментария	572.00

К сожалению, такие комментарии действительно есть. Посмотрим на один из них.

```
Topic                          Biology
Comment                        Then no
words                       [Then, no]
text_wo_punkt                  then no
words_wo_punkt_wo_stopwords         []
text_wo_punkt_wo_stopwords
Name: 123, dtype: object
```

Рис 5. *Комментарий, состоящий только из стоп-слов*

Очевидно, что такие комментарии будут лишними при построении модели. Удалим строчки с комментариями, которые состоят только из стоп-слов.

Теперь посмотрим на часто встречающиеся биграммы слов.

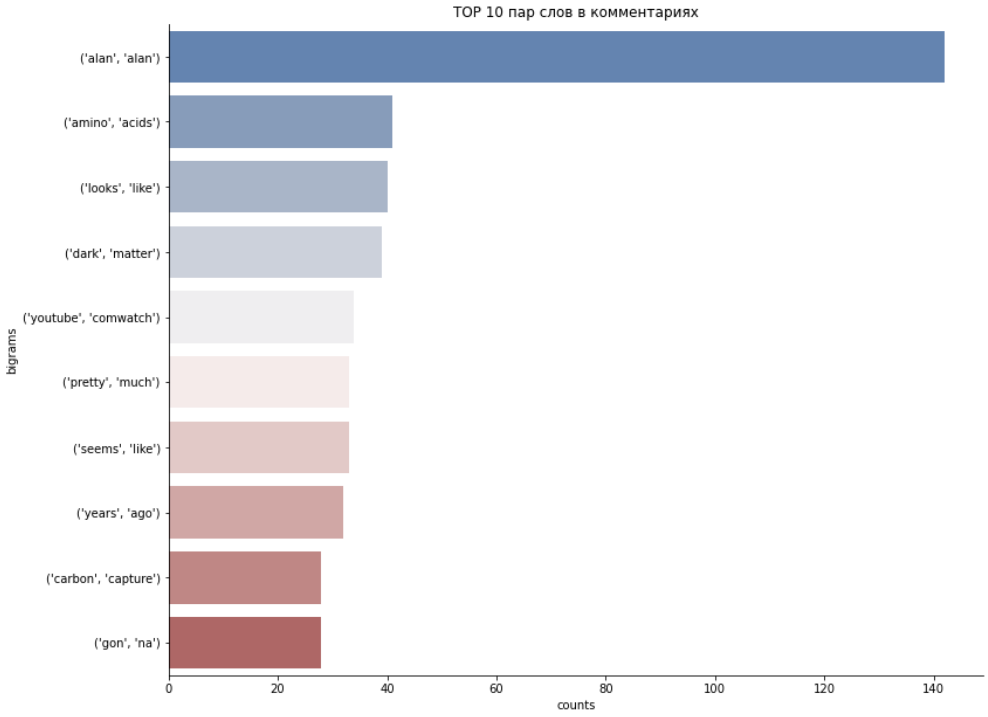

Рис 6. *ТОП-10 биграмм*

Большинство биграмм имеет смысл. Темная материя указывает на физику, амино кислоты – на химию или биологию. Но неизвестное выражение «alan alan», встречающееся более 140 раз, кажется лишенным смысла. Выведем комментарий, где встречается это словосочетание, чтобы понять, имеем ли мы дело с научным термином или аномалией.

```
Индекс комментария: 4170
Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! A
lan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Al
an! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Ala
n! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Ala
n! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Ala
n! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Ala
n! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! Alan! \n\nEtc
```

Рис. 7. *Подозрительный комментарий*

Таким образом, биграммы помогли найти еще один комментарий, не подходящий для решения задачи классификации текстов по категориям.

В заключение предварительного анализа датасета снова посмотрим на облака слов.

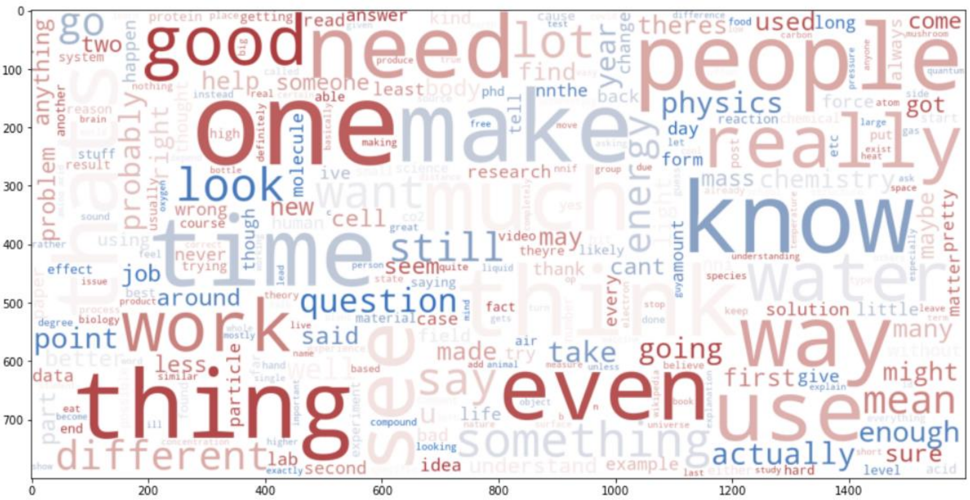

Рис.8. *Облако слов по всем комментариям*

А также по биологии, физике и химии соответственно:

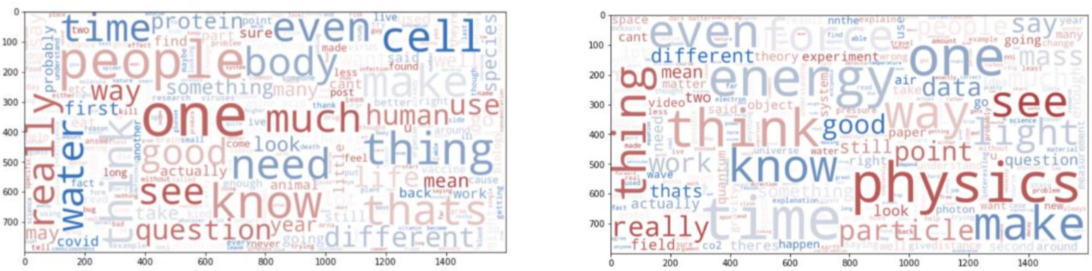

Рис.9. Слева: *облако слов по теме «Биология»;* справа: по теме *«Физика»*

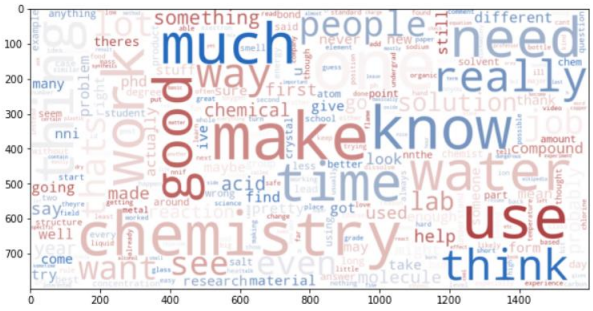

Рис.10. *Облако слов по теме «Химия»*

Довольно интересно, что в последних двух случаях название темы входит в число ключевых слов, в то время как «биология» не встречается в своем облаке.

Благодаря визуализации по темам хорошо видно, что в каждой теме есть свои ключевые слова. Можно предположить, что мешок слов и Tf-IDF дадут относительно неплохие результаты и при тщательной настройки модели, однако все равно рассмотрим все методы. Кроме этого, возможно, что неплохие результаты можно будет получить и без проведения стемминга/лемматизации. Чтобы это проверить, создадим еще один набор данных на основе очищенного датасета, где дополнительно проведем стемминг.

Полученные датасеты (с неочищенными данными, очищенными и преобразованными) будем делить на обучающую и тестовую выборку. Для этого воспользуемся методом train_test_split из библиотеки scikit-learn. При разбиении обаятельно нужно указать параметр stratify(=целевая переменная), который позволит поделить датасет, сохранив отношения одного класса к другому. Для теста оставим 20% данных. Параметр shuffle=True означает, что данные будут перемешаны перед деланием, чтобы разбиение вышло случайным (следовательно, каждый раз при запуске этого кода для одних и тех же данных в обучающую и тестовую выборки будут попадать разные объекты).

3. Построение признаков

3.1. Мешок слов

Мешок слов реализован в бибилотеке scikit-learn, поэтому воспользуемся уже готовым вариантом. Метод CountVectorizer имеет параметр min_df, который позволяет не учитывать слова, которые встречаются в корпусе меньше заданного количества раз. Рассмотрим случаи, когда этот параметр равен 0 (то есть учитываем все слова), и когда он равен 2.

Таблица 6. *Результаты по CountVectorizer*

	Описание	Train_accuracy	Test_accuracy	Train_precision	Test_precision	Train_recall	Test_recall	Train_f1	Test_f1	Time
0	Исходный, CountVectorizer(min_df=0)	0.487205	0.445083	0.481287	0.437921	0.488068	0.445083	0.482890	0.439562	15.327865
1	Исходный, CountVectorizer(min_df=2)	0.439764	0.424382	0.464609	0.448648	0.440627	0.424382	0.441296	0.427077	15.351695
2	Очищенный, CountVectorizer(min_df=0)	0.536774	0.490530	0.571218	0.520542	0.529672	0.490530	0.533099	0.490480	17.335176
3	Очищенный, CountVectorizer(min_df=2)	0.548769	0.515152	0.563095	0.516576	0.552872	0.515152	0.547099	0.506876	9.883869
4	Очищенный (stemming), CountVectorizer(min_df=0)	0.520044	0.498106	0.543342	0.505441	0.526673	0.498106	0.529223	0.498942	12.312298
5	Очищенный (stemming), CountVectorizer(min_df=2)	0.536932	0.497475	0.548172	0.509840	0.536301	0.497475	0.538593	0.499201	6.778001

Таблица позволяет сделать следующие выводы:

- Если брать все слова (min_df=0), то модель работает дольше, чем при min_df=2;
- Очищенный датасет и датасет с использованием стемминга дают результаты выше, чем для исходного;
- При использовании мешка слов неплохих результатов можно добиться даже без приведения слов к основе;
- Простая модель лучше отделяет классы при использовании всех слов, однако общее качество и полнота лучше при min_df=2.

3.2. TF-IDF

Данный метод также можно найти в библиотеке scikit-learn. Здесь воспользуемся параметром norm, который отвечает за норму строки: при l2 сумма квадратов элементов вектора равна1, при l1 – сумма абсолютных значений элементов вектора равна 1.

Таблица 7. *Результаты по TfidfVectorizer*

	Описание	Train_accuracy	Test_accuracy	Train_precision	Test_precision	Train_recall	Test_recall	Train_f1	Test_f1	Time
6	Исходный, TfidfVectorizer(norm=l2)	0.375791	0.377803	0.403764	0.406229	0.376078	0.377803	0.376613	0.379048	11.776042
7	Исходный, TfidfVectorizer(norm=l1)	0.392036	0.368603	0.392421	0.367117	0.392036	0.368603	0.377193	0.354099	11.368586
8	Очищенный, TfidfVectorizer(norm=l2)	0.434501	0.406566	0.462002	0.416124	0.448706	0.406566	0.448937	0.406936	11.195340
9	Очищенный, TfidfVectorizer(norm=l1)	0.899148	0.621212	0.906881	0.638427	0.905303	0.621212	0.904630	0.612146	12.633354
10	Очищенный (stemming), TfidfVectorizer(norm=l2)	0.391256	0.366162	0.521961	0.455053	0.392045	0.366162	0.371882	0.343083	10.514707
11	Очищенный (stemming), TfidfVectorizer(norm=l1)	0.379104	0.385732	0.385359	0.361811	0.386679	0.385732	0.331174	0.332244	11.993465

Выводы по TfidfVectorizer:

- L2-регуляризация работает лучше, чем l1 для неочищенного датасета и для датасета с использование стемминга;
- Очищенный датасет дает прирост по качеству модели, при чем l1 работает намного лучше, чем l2;
- Датасет с использованием стемминга дает наихудшие результаты;
- Модель работает быстро на каждом датасете, но немного дольше, чем при использовании мешка слов.

3.3. Bigrams

Как уже упоминалось в предыдущем разделе, биграммы приводят к расширению словаря. В связи с этим мы не будем использовать этот метод на неочищенном датасете, так как это приводит к переполнению памяти компьютера.

Для построения биграмм используется уже знакомый метод CountVectorizer, но с указанием дополнительного параметра ngram_range=(2,2) (это задает нижние и верхние границы диапазона).

Таблица 8. *Результаты для биграмм*

	Описание	Train_accuracy	Test_accuracy	Train_precision	Test_precision	Train_recall	Test_recall	Train_f1	Test_f1	Tim
12	Очищенный, Bigrams	0.479956	0.442551	0.536236	0.473766	0.480587	0.442551	0.437035	0.393045	7.98993
13	Очищенный (stemming), Bigrams	0.519255	0.467172	0.575876	0.501819	0.519413	0.467172	0.488739	0.428615	9.91848

Датасет с использованием стемминга дает результаты лучше, чем без него, однако разница не слишком большая: метрики для очищенных данных тоже можно считать неплохими. Стоит заметить, что из-за стемминга модель работала немного дольше.

3.4. Word2Vec

Этот метод также можно не реализовывать самому, а использовать уже готовый вариант из библиотеки gensim.

Сначала даются параметры модели: размер вектора, ширина окна, минимальная частотность для отсекания слова. Возьмем стандартные значения 100, 5 и 2 соответственно. Затем строится словарь на основе предложений из корпуса (метод .build_vocab()), после чего выполняется обучение модели (.train()) при указанных количествах эпох (возьмем 5).

Библиотечный метод word2vec также поддерживает выбор между двумя вариантами метода: BoW и Skip-gram. Настройка выполняется через указание параметра sg:

- Sg = 0 => используется BoW;
- Sg = 1 => используется skip-gram

В связи с необходимостью дополнительно обучать модель word2vec, снова ограничимся только очищенными датасетами, чтобы избежать израсходования памяти.

Таблица 9. *Результаты word2vec*

	Описание	Train_accuracy	Test_accuracy	Train_precision	Test_precision	Train_recall	Test_recall	Train_f1	Test_f1	Time
14	Очищенный, word2vec (skip-gram)	0.365530	0.361111	0.373081	0.360048	0.365530	0.361111	0.251786	0.251908	72.082139
15	Очищенный, word2vec (bow)	0.359691	0.359848	0.769687	0.769642	0.359691	0.359848	0.190304	0.190449	71.965287
16	Очищенный (stemming), word2vec (skip-gram)	0.306660	0.273359	0.364680	0.321948	0.306660	0.273359	0.283529	0.251450	59.544469
17	Очищенный (stemming), word2vec (bow)	0.256155	0.256313	0.809548	0.809383	0.256155	0.256313	0.104683	0.104586	58.401974

Результаты получились неоднозначные. Как можно увидеть, precision на датасете со стеммингом при использовании метода BoW дал очень высокий результат. Это значит, что word2vec смог так закодировать слова, что даже простая нейронная сеть научилась отличать один класс то другого на 80%. К сожалению, другие метрики дали более негативные результаты.

Также можно заметить, что стемминг дает преимущество по времени обучения модели.

Важное замечание: так как word2vec – это нейронная сеть, то она требует тщательной настройки параметров. Попробуем теперь уменьшить ширину окна с 5 до 2-х.

Таблица 10. *Результаты word2vec при window=2*

	Описание	Train_accuracy	Test_accuracy	Train_precision	Test_precision	Train_recall	Test_recall	Train_f1	Test_f1	Time
19	Очищенный, word2vec (skip-gram, window=2)	0.359691	0.360480	0.769687	0.769724	0.359691	0.360480	0.190304	0.191797	73.914813
20	Очищенный, word2vec (bow, window=2)	0.384312	0.383838	0.403716	0.763494	0.384312	0.383838	0.213409	0.212932	78.085997
21	Очищенный (stemming), word2vec (skip-gram, window=2)	0.384470	0.383838	0.763407	0.763494	0.384470	0.383838	0.213725	0.212932	60.322063
22	Очищенный (stemming), word2vec (bow, window=2)	0.359691	0.359848	0.769687	0.769642	0.359691	0.359848	0.190304	0.190449	58.275167

Можно заметить, что результаты в целом стали немного выше. Precision на тестовой выборке для всех случаев колеблется около 0.76.

Такое улучшение доказывает, что при использовании данного метода следует уделить особое внимание настраиванию гиперпараметров. Тогда, при правильно подобранных значениях для конкретного датасета, даже простая нейронная сеть сможет дать отличные результаты.

3.5. GloVe

Последним протестируем уже обученную модель векторизации слов из библиотеки gensim – GloVe. Поскольку в датасете много терминов, возьмем версию, обученную на Википедии.

Таблица 11. *Результаты по GloVe*

	Описание	Train_accuracy	Test_accuracy	Train_precision	Test_precision	Train_recall	Test_recall	Train_f1	Test_f1	Time
18	Очищенный, GloVe	0.39	0.41	0.38	0.41	0.39	0.41	0.30	0.32	60.17

Результаты использования преобученной модели оказались на такими высокими, как хотелось бы, и, что главное, они уступают модели word2vec, которую мы обучили самостоятельно. На самом деле, все довольно очевидно: word2vec, который обучался на нашем датасете, сам подстраивался под конкретные слова и контексты в рамках корпуса. При использовании уже обученной модели нам приходилось пропускать слова, которые отсутствуют в

словаре (появление таких слов в основном обусловлено опечатками, которые свойственны комментариям).

Еще одна причина может заключаться в чрезмерной простоте классификатора.

4. Сравнение результатов и выводы

Приведем итоговую таблицу с со всеми методами и настройками. Чем темнее цвет ячейки, тем выше значение метрики. Благодаря такой раскраске легко заметить, что самые высокие результаты были получены на очищенном датасете при векторизации методом TF-IDF с нормализацией l1. Хотя precision на тестовом наборе остается на уровне 0.6, в целом результат довольно хороший при учете, что использовался самая простая модель нейронной сети.

Использование word2vec привело к осту времени обучения классификатора, и именно этот метод позволил лучше отделять один класс от другого, хотя метрика recall едва ли дает приличные результаты – здесь, наоборот, лидируют более простые методы векторизации.

Таблица 12. *Итоговая таблица*

	Описание	Train_accuracy	Test_accuracy	Train_precision	Test_precision	Train_recall	Test_recall	Train_f1	Test_f1	Time
0	Исходный, CountVectorizer(min_df=0)	0.487205	0.445083	0.481287	0.437921	0.488068	0.445083	0.482890	0.439562	15.327865
1	Исходный, CountVectorizer(min_df=2)	0.439764	0.424382	0.464609	0.448648	0.440627	0.424382	0.441296	0.427077	15.351695
2	Очищенный, CountVectorizer(min_df=0)	0.536774	0.490530	0.571218	0.520542	0.529672	0.490530	0.533099	0.490480	17.335176
3	Очищенный, CountVectorizer(min_df=2)	0.548769	0.515152	0.563095	0.516576	0.552872	0.515152	0.547099	0.506876	9.883869
4	Очищенный (stemming), CountVectorizer(min_df=0)	0.520044	0.498106	0.543342	0.505441	0.526673	0.498106	0.529223	0.498942	12.312298
5	Очищенный (stemming), CountVectorizer(min_df=2)	0.536932	0.497475	0.548172	0.509840	0.536301	0.497475	0.538593	0.499201	6.778001
6	Исходный, TfidfVectorizer(norm=l2)	0.375791	0.377803	0.403764	0.406229	0.376078	0.377803	0.376613	0.379048	11.776042
7	Исходный, TfidfVectorizer(norm=l1)	0.392036	0.368603	0.392421	0.367117	0.392036	0.368603	0.377193	0.354099	11.368586
8	Очищенный, TfidfVectorizer(norm=l2)	0.434501	0.406566	0.462002	0.416124	0.448706	0.406566	0.448937	0.406936	11.195340
9	Очищенный, TfidfVectorizer(norm=l1)	0.899148	0.621212	0.906881	0.638427	0.905303	0.621212	0.904630	0.612146	12.633354
10	Очищенный (stemming), TfidfVectorizer(norm=l2)	0.391256	0.366162	0.521961	0.455053	0.392045	0.366162	0.371882	0.343083	10.514707
11	Очищенный (stemming), TfidfVectorizer(norm=l1)	0.379104	0.385732	0.385359	0.361811	0.386679	0.385732	0.331174	0.332244	11.993465
12	Очищенный, Bigrams	0.479956	0.442551	0.536236	0.473766	0.480587	0.442551	0.437035	0.393045	7.989933
13	Очищенный (stemming), Bigrams	0.519255	0.467172	0.575876	0.501819	0.519413	0.467172	0.488739	0.428615	9.918487
14	Очищенный, word2vec (skip-gram)	0.365530	0.361111	0.373081	0.360048	0.365530	0.361111	0.251786	0.251908	72.082139
15	Очищенный, word2vec (bow)	0.359691	0.359848	0.769687	0.769642	0.359691	0.359848	0.190304	0.190449	71.965287
16	Очищенный (stemming), word2vec (skip-gram)	0.306660	0.273359	0.364680	0.321948	0.306660	0.273359	0.283529	0.251450	59.544469
17	Очищенный (stemming), word2vec (bow)	0.256155	0.256313	0.809548	0.809383	0.256155	0.256313	0.104683	0.104586	58.401974
18	Очищенный, GloVe	0.392677	0.406566	0.384738	0.408051	0.392677	0.406566	0.304295	0.316371	60.166724
19	Очищенный, word2vec (skip-gram, window=2)	0.359691	0.360480	0.769687	0.769724	0.359691	0.360480	0.190304	0.191797	73.914813
20	Очищенный, word2vec (bow, window=2)	0.384312	0.383838	0.403716	0.763494	0.384312	0.383838	0.213409	0.212932	78.085997
21	Очищенный (stemming), word2vec (skip-gram, window=2)	0.384470	0.383838	0.763407	0.763494	0.384470	0.383838	0.213725	0.212932	60.322063
22	Очищенный (stemming), word2vec (bow, window=2)	0.359691	0.359848	0.769687	0.769642	0.359691	0.359848	0.190304	0.190449	58.275167

Подведем итоги.

Для сравнения методов извлечения признаков была использована очень простая модель классификации, и, строго говоря, для любого этого метода можно добиться лучших результатов путем усложнения архитектуры модели или изменения параметров. Тем не менее, благодаря этому исследованию уже можно сделать некоторые выводы.

Очистка текста от стоп-слов и знаков препинания в первую очередь дает выигрыш по времени обучения и объему затраченной памяти. Приведение к основе слова требуется не всегда, и при решении простой задачи вполне можно пропустить этот шаг.

Очевидно, что TF-IDF дает отличные результаты – во многом благодаря специфике данных. Как уже неоднократно подмечалось при анализе и

визуализации текстов, каждый топик характеризовался своим набором ключевых слов, поэтому векторизация с учетом важности токенов позволяет классификатору более точно отделять классы.

Биграммы – тоже довольно полезный инструмент, хотя они и ведут к потере по времени и объему памяти. Научные термины часто представляются в виде словосочетаний, поэтому при дальнейшей работе с этими данными стоит остановиться именно на варианте разбиения текста на биграммы и использовании мешка слов. При правильно подобранной архитектуре нейронной сети такой подход может дать результаты лучше, чем BoW или TF-IDF для отдельных токенов.

Word2Vec считается более продвинутым способом извлечения признаков, но когда речь идет о простых данных, то эффективнее использовать другие методы. Такая модель лучше подходит для других задач NLP, где большую роль играет контекст слова.

ЗАКЛЮЧЕНИЕ

Анализ и извлечение признаков в задачах NLP – это сложный, но интересный процесс. Помимо стандартных способов обнаружения аномалий в случае с текстовыми данными хорошим инструментов является визуализация: отображение наиболее часто встречающихся токенов позволит вовремя заметить подозрительные объекты.

Построение облака слов может помочь выбрать стратегию извлечения признаков: в задаче классификации при не очень большом количестве классов уже на этом этапе можно предположить, будет ли давать хорошие результаты обычный мешок слов или TF-IDF.

Построение биграмм/триграмм позволит определить, стоит ли рассматривать токены по одному, или полезнее будет объединить их в n-граммы.

В большинстве случаев при обработке текстов достаточно удаления знаков препинания и стоп-слов, однако иногда для повышения качества модели требуются дополнительные шаги, например, стемминг или лемматизация. В работе было наглядно показано, что этот этап требуется не для всех данных.

В некоторых случаях следует попробовать более сложный метод извлечения признаков - Word2Vec. Хотя он требует дополнительных настроек, он может помочь в том случае, если в тексте есть неисправленные опечатки, так как обучается на . За счет обучения на данных с упором на контекст word2vec будет более гибко подходить к данной проблеме.

В рамках курсовой работы было рассмотрено несколько методов извлечения признаков из текстовых данных: мешок слов, TF-IDF, word2vec с использованием CBoW, word2vec с использованием skip-gram, предобученная модель. Наилучшие результаты показали TF-IDF с нормой l1 и методы word2vec, поэтому при дальнейшем изучении этого датасета стоит попробовать усложнить архитектуру нейронной сети и сравнить результаты для TF-IDF(norm = “l1”) и word2vec, предварительно дообучив модель векторизации слов.

Предположительно, повысить recall для второго метода возможно при увеличении количества эпох обучения векторизатора и определения оптимальной ширины окна.

СПИСОК ИСТОЧНИКОВ

1. Гольдберг Й. Нейросетевые методы в обработке естественного языка / пер. с анг. А. А. Слинкина. – М.: ДМК Пресс, 2019. – 282 с.: ил
2. Макмахан Б., Рао Д.. Знакомство с PyTorch: глубокое обучение при обработке естественного языка. — СПб.: Питер, 2020. — 256 с.: ил. — (Серия «Бестселлеры O’Reilly»).
3. Цитульский А.М, Иванников А.В. - обработка естественных языков // StudNet. 2020. №6. URL: https://cyberleninka.ru/article/n/nlp-obrabotka-estestvennyh-yazykov (дата обращения: 10.11.2022).
4. Eshban, Suleman – Feature Engineering in NLP [Электронный ресурс] URL: https://eshban9492.medium.com/feature-engineering-in-nlp-7d89bf47f7ae
5. Энциклопедия Википедия [Электронный ресурс] URL: wikipedia.org
6. Документация библиотеки scikit-learn [Электронный ресурс] URL: https://scikit-learn.org/stable/index.html
7. Документация библиотеки gensim [Электронный ресурс] URL: https://radimrehurek.com/gensim/index.html

ПРИЛОЖЕНИЯ

Приложение 1

Тип процессора: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx

Объем кэш-памяти второго уровня: 512 Кб

Тактовая частота: 2.1 ГГц

В приложении 1 приведены некоторые строчки кода, относящихся к визуализации и анализу датасета.

```
v, c = np.unique(data_full['Topic'], return_counts=True)
plt.pie(c, labels=v,
        colors=sns.color_palette('pastel'),
        autopct='%.1f%%');
print(f'Количество строк: {data_full.shape[0]}\nКоличество столбцов: {data_full.shape[1]}')
data_dict = {'Тип':data_full.dtypes,
             'Количество Nan':data_full.isnull().sum(),
             'Количество уникальных':[data_full[i].unique().shape[0] for i in data_full.columns],
             'Уникальные значения':[data_full[i].unique() for i in data_full.columns]}
df_descriptive_statistics = pd.DataFrame.from_dict(data_dict,
                                                   orient='columns')
data_full[data_full.duplicated()]
index_list = data_full[data_full['Comment']=='[removed]'].index
data_full = data_full.drop(index_list, axis=0).reset_index(drop=True)
cloud = WordCloud(width=1600, height=800, max_font_size=200, max_words=300, colormap='vlag',background_color="white", collocations=True).generate(all_comments)
plt.figure(figsize=(15,10))
plt.imshow(cloud);
def comments_statistics(col):
    df = pd.DataFrame(None)
    comments_length = []
    short_comments = []
    for index, row in data.iterrows():
```

```
        words = word_tokenize(row[col])
        if len(words) < 1:
            short_comments.append(index)
        comments_length.append(len(words))
    df['Средняя длина комментария'] = [round(np.mean(comments_length),
2)]
    df['Длина самого короткого комментария'] = [min(comments_length)]
    df['Длина самого длинного комментария'] = [max(comments_length)]
    return (df.T, short_comments)
```

Приложение 2

В приложении 2 приведены некоторые строчки кода, относящихся к извлечению признаков и построению классификатора.

```
description = 'Исходный, CountVectorizer(min_df=0)'
vect = CountVectorizer(min_df=0)
X = vect.fit_transform(X_dirty).toarray()
X_train, X_test, y_train, y_test = train_test_split(X, y_dirty,
                                                    test_size=0.2,
                                                    shuffle=True,
                                                    stratify=y_dirty)
model = Sequential()
model.add(Flatten())
model.add(Dense(3))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
start = time.time()
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test,
y_test), batch_size=batch_size)
end = time.time()
train_acc = history.history['accuracy'][-1]
test_acc = history.history['val_accuracy'][-1]
pred_train = np.argmax(model.predict(X_train), axis=1)
pred_test = np.argmax(model.predict(X_test), axis=1)
train_precision = precision_score(y_train, pred_train,
average='weighted')
test_precision = precision_score(y_test, pred_test, average='weighted')
train_recall = recall_score(y_train, pred_train, average='weighted')
```

```
test_recall = recall_score(y_test, pred_test, average='weighted')
train_f = f1_score(y_train, pred_train, average='weighted')
test_f = f1_score(y_test, pred_test, average='weighted')
df = df.append({'Описание': description,
               'Train_accuracy':train_acc, 'Test_accuracy':test_acc,
                'Train_precision':train_precision,
'Test_precision':test_precision,
                'Train_recall':train_recall, 'Test_recall':test_recall,
                'Train_f1':train_f, 'Test_f1':test_f,
                'Time':end-start}, ignore_index=True)
```