

**Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Институт вычислительной математики и информационных технологий  
Кафедра системного анализа и информационных технологий

Направление подготовки: 02.03.02 – Фундаментальная информатика  
и информационные технологии

Профиль: Системный анализ и информационные технологии

**КУРСОВАЯ РАБОТА**

**Экспериментальное исследование методов аннотирования текста**

Студент 3 курса

группы 09-832

«\_\_\_» \_\_\_\_\_ 2021 г. \_\_\_\_\_ Кузьмина В.А.

Научный руководитель

к.ф.-м.н., доцент кафедры САИТ

«\_\_\_» \_\_\_\_\_ 2021 г. \_\_\_\_\_ Андрианова А.А.

Казань-2021

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ .....	3
1. Теоретические аспекты задачи автоматического аннотирования .....	5
1.1 Классификация методов аннотирования .....	5
1.2. Метод Луна .....	7
1.3. Метод деревьев решений .....	7
1.4. Метод кластеризации.....	10
1.5. Предобработка данных .....	11
2. Программная реализация методов предобработки и аннотирования.....	16
2.1. Метод Луна .....	16
2.2. Метод деревьев решений .....	16
2.3. Метод кластеризации.....	17
2.4. Предобработка данных .....	18
3. Исходный пакет данных .....	21
4. Используемое программное обеспечение и характеристики ЭВМ.....	22
5. Оценка работы алгоритмов .....	23
6. Результаты экспериментов .....	26
7. Примеры работы алгоритмов и их анализ .....	28
8. Выводы .....	34
ЗАКЛЮЧЕНИЕ .....	35
СПИСОК ЛИТЕРАТУРЫ.....	38
ПРИЛОЖЕНИЕ .....	40

## ВВЕДЕНИЕ

В современном мире количество доступной для чтения информации увеличивается каждый день. Вместе с этим растет актуальность способов сокращения времени на ее изучение. Одним из таких распространенных способов является аннотирование - сжатие текста. Грамотное краткое содержание должно содержать основные идеи текста, исключая второстепенную, излишне подробную информацию. Иногда к объемным текстам прилагается краткое содержание, но только иногда. Ручное аннотирование – процесс трудоемкий, поэтому возникает необходимость в способах автоматического сокращения текстов. Вопрос машинного аннотирования довольно спорный, так как в составлении аннотации большую роль играет индивидуальный человеческий фактор. В одном и том же тексте разные люди могут подчеркивать абсолютно разные основные мысли. Несмотря на это, существуют приложения и сайты, предлагающие технические решения сжатия текста. Для русскоязычных текстов количество подобных утилит сравнительно ограничено. Поэтому данное исследование посвящено аннотированию текстов именно на русском языке.

Целью работы является сравнение нескольких методов машинного обучения в рамках проблемы сокращения текста. Проводимое исследование можно разбить на следующие задачи:

- 1) теоретический обзор существующих методов автоматического аннотирования, изучение их основных принципов, выбор нескольких методов для дальнейшего сравнения;
- 2) рассмотрение вопроса предобработки текста, выделение обязательных этапов предобработки данных для решения поставленной задачи;
- 3) выбор способа оценки работы методов;
- 4) поиск необходимого пакета данных для обучения модели и тестирования методов, библиотек для работы с русскоязычными текстами, реализованных алгоритмов выбранных методов аннотирования;

5) программная реализация предобработки текстов, вспомогательных функций подготовки данных к их использованию в алгоритмах аннотирования;

6) использование существующих методов машинного обучения для получения результатов;

7) программная реализация способа оценивания работы алгоритмов, сравнение полученных результатов.

## **1. Теоретические аспекты задачи автоматического аннотирования**

### **1.1 Классификация методов аннотирования**

Все существующие способы автоматического аннотирования в целом можно поделить на две группы: первая – извлекающие методы, вторая – генерирующие [1]. Извлекающие методы подразумевают выбор из текста наиболее значимых, содержащих основную идею, предложений без их изменения. А генерирующие методы в качестве краткого содержания выдают буквально новый текст, слова и словосочетания в котором отличаются от состава исходного текста, но передают его основную идею. В данном исследовании будут рассмотрены только методы из первой группы, так как они требуют гораздо меньше вычислительной мощности, а значит, являются более доступными в использовании.

В свою очередь, извлекающие методы аннотирования принято делить на три вида методов по их уровню сложности. Первый тип составляют поверхностные способы выбора главных предложений, не использующие инструментов машинного обучения. Алгоритмы такого типа разрабатывались еще с конца прошлого века. Главным представителем поверхностных методов является метод Луна, по идеям которого наиболее значимые предложения текста содержат слова с высокой частотой использования их в данном документе. То есть в аннотацию попадают предложения, средняя частота слов в которых выше, чем в других. По похожему принципу работает метод Эдмундсона, только кроме частоты встречаемости слов он еще учитывает наличие заранее определенных ключевых слов в предложении, структуру документа. Кроме названных методов, к поверхностным алгоритмам относят так же способы аннотирования на основе графов, и другие методы, которые используют какие-либо сравнительные характеристики.

Автоматические способы сокращения текста второго типа являются более сложным и новыми, так как они используют машинное обучение. Сюда

относятся методы классификации и кластеризации. Среди классификации особенно часто для данной задачи выделяют алгоритм построения деревьев решений, метод опорных векторов, нейронные сети. Цель любого классификатора – для каждого предложения по заданным характеристикам определить, входит оно в аннотацию или нет. А цель кластеризации – разбить предложения из текста на некие группы, кластеры, по смыслу, и для краткого содержания выбрать из каждой смысловой группы по одному наиболее «показательному» предложению.

К третьему виду методов аннотирования причисляются самые сложные алгоритмы – это сверточные, рекуррентные нейронные сети и латентно-семантический анализ. Идея латентно-семантического анализа отчасти похожа на суть кластеризации: выявить в тексте основные темы, и в аннотацию выбирать представителя каждой из тем. Такие глубокие методы являются менее изученными и более перспективными по отношению к первым двум типам способов механического создания краткого содержания.

Еще методы аннотирования классифицируют по используемым ими языкам. Монолингвальные методы для случаев, когда исходный текст и созданная аннотация на одном и том же языке, мультилингвальные – если и исходный, и итоговый тексты могут использовать сразу несколько языков, кросслингвальные – если есть необходимость создавать аннотацию на языке, отличающемся от языка исходного текста. Так же следует упомянуть разделение алгоритмов аннотирования на методы для сокращения одного документа, и на методы для сокращения сразу пакетов из нескольких документов.

Для более тщательного изучения и сравнения в данной работе были выделены метод Луна, как представитель поверхностного аннотирования, метод классификации с использованием деревьев решений – пример машинного обучения с учителем, и алгоритм кластеризации – самый распространенный способ обучения без учителя. Все эти методы будут рассмотрены для их использования на отдельных русскоязычных текстах.

## 1.2. Метод Луна

Для того чтобы составить аннотацию по методу Луна [2], необходимо обработать текст следующим образом:

- 1) выделить словарь всех уникальных слов, встречающихся в данном тексте;
- 2) подсчитать частоту использования каждого слова в словаре, минимальная частота каждого слова – единица;
- 3) определить долю слов с наиболее высокой частотой использования, которые будут учитываться для оценки предложений, например, 20% от всех слов в словаре; из этих слов составить оценочный словарь;
- 4) для каждого предложения подсчитать его ранг по формуле:

$$rank = \frac{S_{imp}^2}{S},$$

где  $S$  – количество всех слов из основного словаря в предложении,  $S_{imp}$  – количество слов в предложении из оценочного словаря.

Таким образом, чем больше доля слов из оценочного словаря в предложении, тем больше будет его ранг;

- 5) выбрать долю или количество предложений, которые будут включены в итоговую аннотацию, например, 20% от всех предложений в тексте или статично 4 предложения; из этих предложений с наибольшим рангом состоит результат работы алгоритма – сокращенная версия исходного текста.

## 1.3. Метод деревьев решений

Данный метод является представителем области машинного обучения с учителем. То есть модель тренируется, используя данные с указанными правильными ответами - классами. Классификатор строится следующим образом: изначально все объекты обучающей выборки находятся в корне дерева. С помощью одной из метрик, как правило, энтропии, путем перебора

выбирается характеристика, по которой объекты оптимально разделяются на несколько групп – так создаются новые внутренние узлы дерева [3]. С каждым новым узлом происходит точно такое же разбиение. Цель алгоритма – в какой-то момент получить узел, в котором все объекты будут принадлежать одному классу, такой узел больше не разбивается и считается листовым. При больших объемах данных достичь полностью «чистого» листа сложно, поэтому разбиение узла прекращается при других условиях, например, при достижении определенной глубины дерева. При подаче в обученную модель примера из тестовой выборки, в дереве строится путь от корня до листа, который соответствует характеристикам данного объекта, предсказанный класс будет зависеть от листа, в котором закончится путь.

В рамках задачи аннотирования текста необходимо было придумать, как делить объекты на классы. Самым очевидным кажется решение делить все предложения на два класса, класс 0 – если предложение не входит в аннотацию соответствующего текста, класс 1 – если входит. Но имеющиеся шаблонные аннотации представляют собой не выбранные предложения из исходного текста, а новое сгенерированное краткое содержание. Поэтому нужен был способ как-то сравнить предложения из исходного текста с предложениями из аннотации. Так как предложения после предобработки представляют собой вектора в  $m$ -мерном пространстве, то в качестве степени их схожести или различия использовалось значение угла между ними, посчитанного по формуле:

$$\varphi = \arccos \left( \frac{\sum_{i=1}^m x_i y_i}{\sqrt{\sum_{i=1}^m x_i^2} * \sqrt{\sum_{i=1}^m y_i^2}} \right).$$

Где  $m$  – количество характеристик (размерность мешка слов),  $x_i$  – значение  $i$ -ой характеристики предложения из исходного текста,  $y_i$  – значение  $i$ -ой характеристики предложения из аннотации.



Значение угла может принимать значение в диапазоне  $[0, \frac{\pi}{2}]$ , 0 – если значения характеристик предложений полностью совпадают,  $\frac{\pi}{2}$  – если предложения вообще ничем не похожи. Для удобства восприятия преобразуем значения, сузив диапазон до  $[0, 1]$ , и поменяем полюса таким образом, чтобы функция выдавала ответ 1 при полном совпадении векторов и ответ 0 при полном их различии.

$$\text{Результат сравнения} = 1 - \varphi / \frac{\pi}{2}.$$

Этим способом сравним каждое предложение из исходного текста с каждым предложением из аннотации. В качестве итогового ранга предложению из исходного текста присвоим максимальный из результатов сравнения его с предложениями краткого содержания. Этот ранг и будем называть классом данного объекта.

Обучение классификатора будет проводиться на векторах-предложениях из всех текстов обучающей выборки, правильными классами для которых будут максимальные степени их похожести на предложения из аннотаций.

Таким образом, в качестве предсказания обученная модель должна выдавать число в диапазоне  $[0, 1]$ , которое будет говорить о вероятности того, что данное предложение похоже на потенциальное предложение из аннотации.

Стоит заметить, что редко при использовании данного метода классификации модель ограничивается построением одного дерева решений. Для данных с большим количеством классов и характеристик чаще используют лес деревьев: модель строит несколько деревьев, которые не так оптимальны, так как при разбиении узлов перебирают ограниченное число характеристик. Но зато предсказание в таком случае – среднее число от ответа каждого дерева, такое предсказание может иметь более высокую точность.

#### **1.4. Метод кластеризации**

Метод кластеризации относится к области машинного обучения без учителя. Это значит, что при обучении модель не нуждается в правильных ответах, она учится классифицировать данные просто по их различиям. Наиболее разные объекты раскидываются по разным группам – кластерам, а схожие объекты относятся к одному кластеру. Для того чтобы краткое содержание документа охватывало все темы, раскрытые в исходном тексте, можно попробовать делить на кластеры все предложения из текста. А в итоговую аннотацию выбирать из каждого кластера по одному предложению, причем такому, которое наиболее близко к центру, а значит, более эталонное.

Самое важное и сложное в этом методе – выбор центров кластеров, а именно – их количество. Для сравнения работы разных методов аннотирования, количество кластеров для текста из тестовой выборки будет соответствовать количеству предложений в соответствующем имеющемся кратком изложении.

## 1.5. Предобработка данных

В направлении машинного обучения, связанного с обработкой естественного языка, вопрос предварительной подготовки данных является крайне важным и обязательным, так как машины не могут работать с обычным текстом на естественном языке. Текстовые строки каким-то образом нужно преобразовать в числа или числовые векторы. От качества предобработки напрямую зависит результат работы методов, использующих эти данные. Этапы предобработки и их порядок могут меняться в зависимости от поставленной задачи, но в целом этот процесс имеет общепринятую схему [4].

1) Токенизация по предложениям: текст разбивается на отдельные предложения.

2) Токенизация по словам: каждое предложение разбивается на отдельные символы и слова, составляющие его.

3) Нормализация: все слова приводятся к нижнему регистру, удаляются любые знаки препинания и лишние пробелы, иногда удаляются также все числа. С помощью нормализации можно удастся немного сократить словарь уникальных слов, содержащихся в исходном тексте.

4) Лемматизация или стемминг: это способы приведения всех слов к их начальной форме [5]. Для вычислительной машины слова «играла» и «играл» - разные слова, но с точки зрения смысловой нагрузки эти слова одинаковые, поэтому их необходимо привести к общему виду.

Стемминг – это подход к обработке слова, при котором отсекаются ненужные окончания, приставки и суффиксы. То есть от исходного слова остается только его корень. Например, слово «играла» избавится от окончания «а» и суффикса «ал», в результате обработки получится основа «игр». Такая же основа получится при обработке слова «поиграть» при отсечении от него суффикса и приставки. Стеммеры бывают двух типов – словарные и алгоритмические. Словарные стеммеры для исходного слова ищут наиболее подходящую основу из словаря основ соответствующего

языка, после чего слово заменяется на найденную основу. А алгоритмические стеммеры отсекают от слова такие приставки, окончания и суффиксы, которые есть в словарях всевозможных морфем данного языка. Словарные стеммеры считаются более точными, так как используя алгоритмический подход можно легко недообработать слово или наоборот отсечь важный словообразующий суффикс. Однако большим преимуществам второго подхода является ограниченность морфем, в то время как корней и основ может быть очень много.

Лемматизация – суть данного способа заключается в том, что слово заменяется на его каноническую форму, то есть лексему, из словаря.

- Для существительных начальная форма, лексема – это слово в единственном числе, именительном падеже («ложка» вместо «ложками»).
- Для прилагательных – определение для слова в единственном числе, именительном падеже, мужского рода («полезный» вместо «полезную»).
- Для глаголов – инфинитив, то есть действие несовершенного вида, не привязанное к объекту и времени («делать» вместо «сделают»).

Исходное слово для поиска его леммы необходимо проанализировать. Лемматизация позволяет учитывать часть речи слова, в некоторых случаях даже контекст его использования.

Стемминг и лемматизация имеют достаточно преимуществ и недостатков, поэтому выбирать один способ из этих необходимо с учетом особенностей исследования. Вероятно, будь исходный пакет текстов для аннотирования на английском языке, предпочтение стоило бы отдать стеммингу, так как это гораздо более быстрый процесс, а словообразование в английском языке имеет сравнительно мало исключений из правил, и, даже отсекая морфемы слов, можно сохранить общий смысл предложения. Но стемминг практически не пригоден для русского языка.

Приведем простой пример: «пошел» и «пойдет». Отсекая приставку «по», суффикс «л» и окончание «ет» от данных слов, получим корни «ше» и «йд» соответственно. Для вычислительной машины эти строчки разные, хотя

оба слова произошли от одного глагола – «пойти». Русский язык очень сложный и богатый, поэтому для приведения слов к общей начальной форме требуется более тщательный анализ исходных слов, что возможно при использовании лемматизации.

5) Удаление стоп-слов: исключение тех слов, которые не несут особой смысловой нагрузки. К стоп-словам обычно относятся предлоги, союзы, междометия, частицы, некоторые местоимения. Список стоп-слов зависит от контекста задачи, используемой предобработку текста.

б) Векторизация: преобразование предложения в вектор. Для того чтобы векторизовать текст, необходимо получить мешок слов – это коллекция всех уникальных слов, встречающихся в документе [6]. Мешок слов формально представляет собой большой вектор, где каждому слову соответствует свой порядковый индекс. Тогда результатом векторизации любого предложения из данного текста будет вектор, длина которого совпадает с размерностью мешка слов. И  $i$ -е значение вектора равно 0, если в предложении нет слова с индексом  $i$  из мешка слов, или равно 1, если такое слово в предложении есть. После векторизации исходный текст принимает вид матрицы размерностью  $n * m$ , где  $n$  – количество предложений в тексте,  $m$  – размерность мешка слов для данного текста. Каждый элемент такой матрицы – целое число, равное 0 или 1. Это самый простейший способ векторизации, который не совсем подходит для задачи аннотирования. Хотелось бы, чтобы вектор предложения отображал не только то, какие слова используются в предложении, но и как часто встречается то или иное слово по сравнению с другими.

Еще один вид векторизации – количественный, когда  $i$ -е значение вектора равно 0, если в предложении нет слова с индексом  $i$  из мешка слов, и равно  $k$ , если слово в предложении есть, где  $k$  – целое число от 1 и больше, показывающее, сколько раз встретилось данное слово в предложении. Тоже не самый показательный способ формализации документа.

Но у частотной векторизации тоже есть минус – если слово встречается часто, то его оценка в векторе-предложении будет высокой. Но если слово встречается часто во многих предложениях текста, то оно не несет в себе много информации. Для того чтобы как-то уменьшать оценку слов, которые используются во многих документах, существует метрика TF-IDF (term frequency — inverse document frequency). Значения векторов-предложений по TF-IDF высчитываются с помощью следующей формулы:

$$TFIDF(\text{слово}_i) = TF(\text{слово}_i) * IDF(\text{слово}_i).$$

$$TF(\text{слово}_i) = \frac{\text{Количество слова}_i \text{ в предложении}}{\text{Количество всех слов в предложении}},$$

$$IDF(\text{слово}_i) = \log\left(\frac{\text{Количество предложений в тексте}}{\text{Количество предложений, содержащих слово}_i}\right).$$

Таким образом, если слово встречается во всех предложениях текста, то оценка для него будет равна нулю.

Рассмотрим пример последовательной предобработки следующего текста: «Не все птицы любят зерно. Пингвин, например, любит рыбу!».

### 1) Токенизация по предложениям

[«Не все птицы любят зерно.», «Пингвин, например, любит рыбу!»]

## 2) Токенизация по словам

[[«Не», «все», «птицы», «любят», «зерно», «.»],

[«Пингвин», «», «например», «», «любит», «рыбу», «!»]]

### 3) Нормализация

[[«не», «все», «птицы», «любят», «зерно»],

[«пингвин», «например», «любит», «рыбу»]]

#### 4) Лемматизация

[[«не», «все», «птица», «любить», «зерно»],  
[«пингвин», «например», «любить», «рыба»]]

#### 5) Удаление стоп-слов

[[«птица», «любить», «зерно»],  
[«пингвин», «любить», «рыба»]]

#### 6) Векторизация с помощью TF-IDF

Мешок слов: [«птица», «любить», «зерно», «пингвин», «рыба»].

[[0.333, 0, 0.333],  
[0.333, 0, 0.333]]

## **2. Программная реализация методов предобработки и аннотирования**

### **2.1. Метод Луна**

Программная реализация данного алгоритма состоит из нескольких логически самостоятельных функций: подсчет частоты использования слов и составление оценочного словаря из них, вычисление ранга всех предложений, получение индексов предложений с наибольшим рангом. В качестве результата алгоритм выдает массив индексов предложений, из которых должна состоять аннотация.

Путем перебора нескольких значений, в данном исследовании в качестве доли наиболее часто встречаемых слов, заносимых в оценочный словарь, было выбрано значение 0.1.

Для того чтобы сравнение разных алгоритмов аннотации было равноценным, было принято решение выбирать столько предложений для аннотации, сколько их содержится в примере настоящего краткого содержания, предложенного для соответствующего текста. Поэтому вместо доли предложений с наибольшим рангом, которые будут входить в итоговую аннотацию, указывалось конкретное число требуемых предложений.

### **2.2. Метод деревьев решений**

Наиболее новым инструментом для построения классификатора с использованием дерева решений является библиотека CatBoost, разработанная компанией Yandex. При попытках использования алгоритмов обучения модели из данной библиотеки пришлось столкнуться с ошибкой, связанной с переполнением оперативной памяти устройства. Около 12ГБ оперативной памяти, предоставляемых облачным сервисом Google Collab, тоже не хватило для построения модели по таким размерностям данных. Поэтому для обучения классификатора в данной работе были использованы методы программной библиотеки XGBoost [7].



В качестве обучающих параметров были выбраны или подобраны экспериментальным путем следующие характеристики:

- Коэффициент для уменьшения шага при обновлении весов: 0.2,
- Максимальная глубина дерева: 8,
- Доля объектов выборки, которые случайным образом выбираются каждую итерацию: 0.6,
- Доля столбцов, случайным образом выбирающаяся при построении каждого нового дерева решений: 0.7,
- Задачи обучения: логистическая регрессия для двоичной классификации,
- Метрика для оценки результатов валидации: функция логарифмического правдоподобия,
- Количество итераций обучения: 200.

Таким образом, был построен лес из 200 деревьев решений (каждую итерацию строится новое дерево). Наилучшие показатели соответствуют последней итерации: на валидационной выборке функция логарифмического правдоподобия равна 0.04518. При большем количестве итераций процесс обучения начинал замедляться, а метрики, оценивающие результат работы, варьировались около одного значения.

### **2.3. Метод кластеризации**

Так как алгоритму кластеризации в рамках данного исследования на вход подается небольшое количество объектов для анализа и разделения на кластеры, то высокая скорость работы метода не является веским основанием для поиска самого оптимального технического решения из существующих. Поэтому был использован наиболее популярный алгоритм KMeans из пакета cluster библиотеки scikit-learn [8]. После распределения предложений по кластерам, метод выдает индексы предложений из текста, которые наиболее близки к центроидам.

## 2.4. Предобработка данных

Существует довольно много библиотек, которые предоставляют алгоритмы сразу для всех этапов предобработки текстовых данных. Наиболее быстрой и современной считается библиотека SpaCy, которая стала достойной заменой известной платформе NLTK. Но в силу того, что SpaCy – инструмент новый, официального пакета для обработки русскоязычных пакетов в нем пока нет, а предварительные или любительские версии работают не так хорошо и не так быстро, как требуют большие объемы данных. Поэтому в данном исследовании для предобработки текстов будут использоваться инструменты сразу из нескольких разных NLP-библиотек.

1) Для того чтобы токенизировать тексты по предложениям, наиболее удобным оказался алгоритм NLTK [9].

2) Токенизация по словам автоматически применяется в следующих этапах обработки.

3) Нормализация данных происходит в несколько шагов: для приведения текста к одному регистру используется встроенный метод языка Python для объектов типа string – lower(). Для того чтобы удалить из текста любые символы, кроме слов и чисел, пригодился модуль re [10], который использует регулярные выражения. Регулярное выражение для поставленной задачи выглядит следующим образом: '[^a – zA – Za – яА – Я0 – 9] + '. Из текстов не удаляются вхождения чисел, так как для новостного сектора такой тип информации может иметь большое значение.

4) В вопросе лемматизации данных решающим фактором оказалась скорость работы алгоритмов, потому что предстояло обработать свыше 50 тысяч текстов. При помощи результатов проведенных экспериментов, описанных в статье [11] была выбрана библиотека PyMystem3, которая изначально была создана для анализатора русского языка Yandex Mystem 3.1 [12]. Таким образом, для лемматизации в данной работе используется метод lemmatize() из пакета Mystem.

Так как время работы этого алгоритма можно сократить, уменьшив количество его вызовов, то тексты временно объединялись с помощью разделительного слова «`bbreakk`», по которому они после предобработки разделялись обратно. Но такие пакеты могли содержать максимум 1000 текстов, так как вычислительное устройство имеет ограниченную оперативную память, а ее для данного алгоритма требуется много.

Таким образом, лемматизация тренировочного набора данных, а именно текстов и их аннотаций, суммарно занимала около 50 минут.

5) Пакет русских стоп-слов загружался так же из библиотеки NLTK.

6) Для TF-IDF векторизации данных использовался класс `TfidfVectorizer`, относящийся к популярному пакету инструментов машинного обучения `scikit-learn`.

В рамках данного исследования, необходимо было преобразовать все предложения в вектора по одному шаблону, чтобы они были объектами одного типа и размера, но с разными значениями характеристик. Это значит, что мешок слов был собран со всех текстов обучающей выборки, а затем по этому мешку слов векторизовались все предложения, включая данные из валидационной и тестовой выборок. Если в предложениях встречались слова, которые отсутствуют в мешке слов, то такие слова просто не учитывались.

Мешок слов, полученный после обработки тренировочных данных, имел размерность свыше 2 миллионов слов, что вполне очевидно, но не совсем практично. В мешке слов явно встречались слова, которые, например, использовались всего 1 раз в одном из предложений. Такая характеристика несет в себе очень мало информации, так как с большой вероятностью это слово больше не встретится в обрабатываемых текстах. Были и слова, которые встречались абсолютно в каждом тексте, но они тоже не считаются важными и ключевыми, так как в контексте новостей приравниваются к стоп-словам.

Поэтому было принято решение сократить мешок слов до 50 тысяч максимум, убрав слова со слишком большой частотой встречаемости (более

чем в 85% предложений). Выдвинутые ограничения предусмотрены алгоритмом векторизации и указываются в параметрах при создании объекта класса `TfidfVectorizer`.

Предобработка данных проводилась не только для исходных текстов обучающей, валидационной и тестовой выборок, но и для имеющихся реальных аннотаций.

### 3. Исходный пакет данных

Для обучения модели с помощью инструментов машинного обучения требуется довольно большая выборка данных, для каждого объекта которой есть шаблонный правильный ответ. Так как в рамках данной задачи объект выборки – текст, а правильный ответ для него – аннотация, то самостоятельная генерация большого количества таких качественных данных не представлялась возможной.

Исходный набор данных должен был удовлетворять следующим условиям:

- количество текстов в пакете не меньше 50.000 штук;
- к каждому тексту должна прилагаться его аннотация;
- средний размер текста не должен превышать 50 предложений, а средняя длина кратких изложений должна быть больше, чем 1 предложение;
- все данные должны быть на русском языке.

Кроме того, исходя из особенностей метода с использованием деревьев решений, тексты должны принадлежать одной предметной области.

Учитывая выдвинутые требования, был найден набор из 63435 русскоязычных текстов новостей с их сокращенными содержаниями [13]. Данные уже были поделены на 3 выборки: обучающую, валидационную и тестовую в примерном масштабе 83:8:9 (то есть выборки размером 52400, 5265 и 5770 соответственно).

Обучение и валидация классификатора методом деревьев решений проводились на первых двух выборках данных, а тестирование модели деревьев решений, алгоритма кластеризации и метода Луна на тестовой третьей выборке.

#### **4. Используемое программное обеспечение и характеристики ЭВМ**

Программная часть данного исследования написана на языке Python. В качестве среды разработки были использованы PyCharm от компании JetBrains [14] и Jupyter notebook – один из проектов интерактивной оболочки IPython [15]. Для подключения в проекты необходимых библиотек применялся дистрибутив Anaconda [16].

Все сохраняемые объекты: обработанные данные, обученные модели, промежуточные результаты вычислений – загружались в память устройства в виде файлов .pickle с помощью специального модуля pickle [17]. Это удобный способ сериализации и десериализации объектов программы, при котором они преобразуются в поток байтов. А отдельные части программы хранились либо в виде файлов .py – если они создавались в среде PyCharm, либо в формате .ipynb – если файл был создан в блокноте Jupyter.

Для работы с векторами и массивами использовалась библиотека NumPy [18]. При необходимости отслеживать процесс работы программы в циклах использовалась специальная библиотека tqdm [19].

Ноутбук, использующийся для данного исследования, имеет следующие характеристики:

- процессор – Intel Core i5 7300HQ 2500 МГц;
- доступная ОЗУ – 7.8 ГБ.

## 5. Оценка работы алгоритмов

Алгоритм каждого из трех сравниваемых методов в качестве результата выдавал индексы предложений из текстов тестовой выборки, которые должны войти в итоговое краткое изложение. Количество индексов соответствовало количеству предложений в реальной аннотации каждого текста. Стоит понимать, что при таком условии предсказываемые предложения по версии того или иного метода могли иметь очень маленькую оценку, однако все равно попадали в ответ. Объем предложений на выходе работы алгоритма был строго регламентирован по нескольким причинам:

- стандартизация процесса оценки работы методов (для одного текста разные алгоритмы предложат одинаковое количество предложений);
- отсутствие ситуаций, когда метод не выдвигает ни одного предложения в аннотацию;
- выбор наилучших кандидатов среди «худших».

Для разъяснения третьего пункта приведем пример. Максимальное предсказание, которое сделала модель классификатора на основе деревьев решений, соответствует значению 0.266. Это означает, что вероятность всех предложений текстов тестовой выборки быть включенными в аннотацию алгоритм оценил в диапазоне до 0.266, хотя наибольшая оценка соответствует 1. Так произошло потому, что вектор характеристик состоит из мешка слов, собранного на тренировочной выборке. Однозначно, многие слова из этого мешка не встречались в тестовых текстах, поэтому вектор характеристик по большей части состоял из нулей. Из-за этого, классификатор видел мало слов, которые могли бы повысить вероятность предложения относиться к аннотации, и довольно низко оценивал каждое предложение тестовых данных.

На конкретных примерах предсказанных аннотаций можно будет увидеть, насколько допустим такой способ отбора строго требуемого количества предложений.

Оценка результата работы метода происходила следующим образом:

1) По предсказанным индексам из исходного текста, обработанного до формата векторов TF-IDF, выбирались соответствующие им вектора.

2) Вектора предсказанных предложений аннотации и вектора реальной аннотации имеют одинаковую размерность, поэтому их можно было сравнить точно так же, как оценивалось каждое предложение для обучения классификатора – поиск угла между векторами и преобразование значения в диапазон  $[0, 1]$ , где 0 соответствует наименьшей оценке, 1 – наибольшей. Напомним формулы для вычисления оценки:

$$\varphi = \arccos \left( \frac{\sum_{i=1}^m x_i y_i}{\sqrt{\sum_{i=1}^m x_i^2} * \sqrt{\sum_{i=1}^m y_i^2}} \right),$$

где  $m$  – количество характеристик (размерность мешка слов),  $x_i$  – значение  $i$ -ой характеристики предложения из предсказанной аннотации,  $y_i$  – значение  $i$ -ой характеристики предложения из реальной аннотации;

$$\text{Результат сравнения} = 1 - \varphi / \frac{\pi}{2}.$$

Этим способом сравнивалось каждое предложение из предсказанной аннотации с каждым предложением из действительного краткого содержания.

3) Каждому предложению из реальной аннотации необходимо было сопоставить одно предложение из предсказанного краткого изложения, которое, желательно, как можно больше на него похоже. Причем сделать это так, чтобы все предложения попали в пару, и ни одно предложение не повторилось. Для этого поочередно перебирались предложения из предсказанной аннотации, для каждого в пару выбиралось предложение из действительного краткого содержания с максимальной оценкой их совпадения. Однако выбор происходил не из всех предложений реальной аннотации, а только из тех, которые еще не поставлены в пару.



4) Оценка предсказания краткого содержания для одного текста высчитывалась как среднее от суммы оценок совпадения предложений предсказанной и предложений реальной аннотаций, образующих пару.

5) Общая оценка результата работы алгоритма – среднее от оценок всех текстов тестового набора данных.

## 6. Результаты экспериментов

В таблице 1 представлены общие оценки каждого из трех алгоритмов аннотирования по результатам их тестирования, а также максимальная и минимальная оценки кратких содержаний, предсказанных каждым из методов.

Таблица 1 - Оценки методов аннотирования

	Метод Луна	Классификация	Кластеризация
Общая оценка	0.09100	0.08870	0.09278
Максимальная оценка предсказанной аннотации	0.56569	0.61727	0.77714
Минимальная оценка предсказанной аннотации	0.0	0.0	0.0

По этим данным можно сделать следующие выводы:

1) Метод кластеризации в среднем лучше предсказывает аннотации, схожие с действительными краткими содержаниями. Метод Луна чуть менее точен, а классификация с помощью деревьев решений в среднем справляется с задачей аннотирования хуже всех.

2) Все общие оценки не превышают значения 0.1, хотя по величине максимальных оценок видно, что некоторые предложенные методами аннотации практически совпадали с реальными. Это говорит о большой доле низких оценок предсказанных аннотаций, которые сильно повлияли на средний балл. Кроме того, в результате работы каждого алгоритма была создана как минимум одна аннотация, которая не имела никаких пересечений с существующим кратким содержанием, из-за чего оценка равнялась 0.0.

Однако низкий результат вовсе не обязательно соответствуют действительно некорректному краткому изложению исходного текста. Все аннотации, использующиеся в качестве шаблонов для сравнения, являются абсолютно новыми текстами, словосочетания которых иногда отличаются от фраз в исходном тексте. А аннотации, предсказанные алгоритмами – взятые

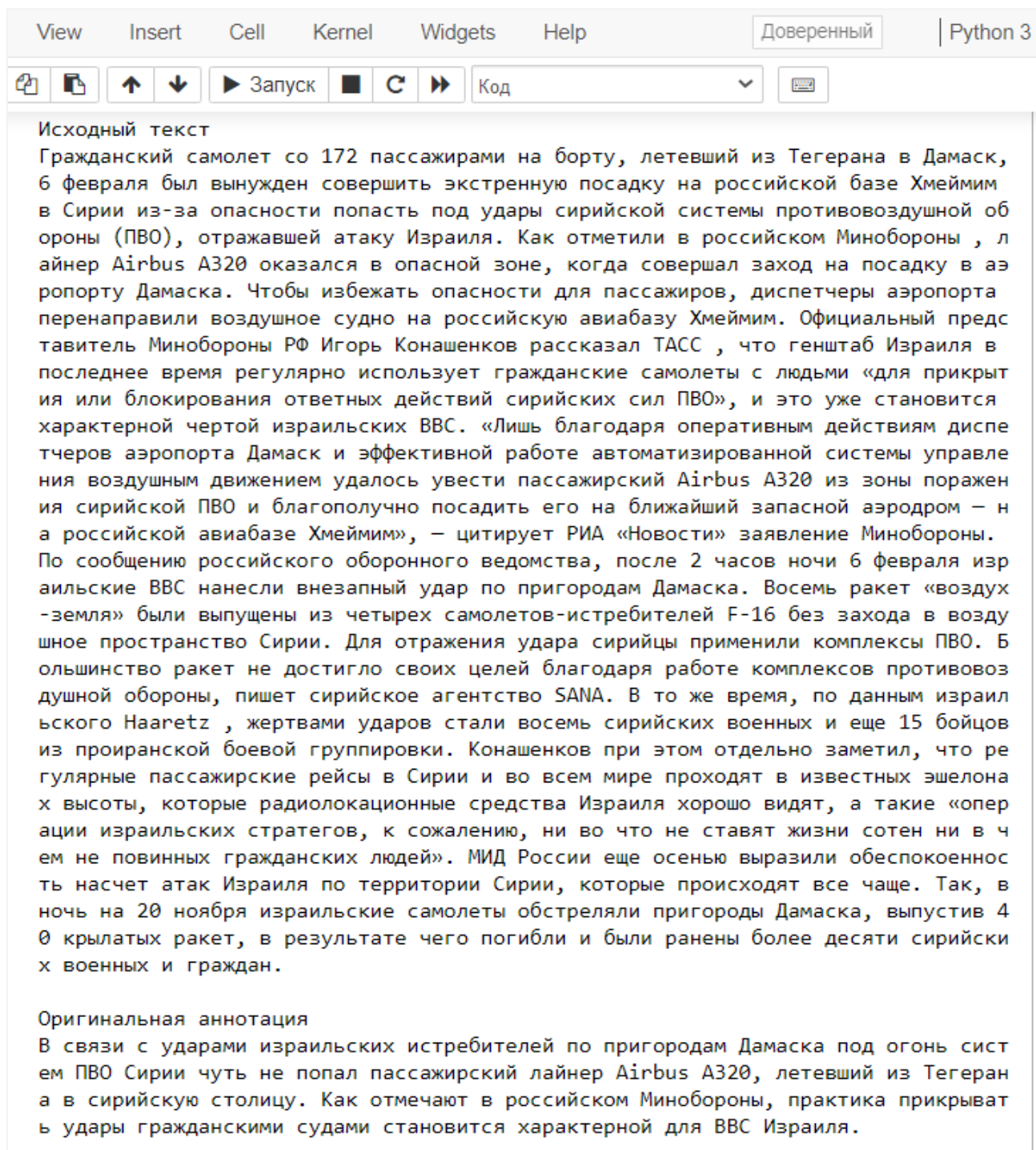
из текста предложения без каких-либо изменений. Поэтому предсказанные методом предложения и предложения из шаблонной аннотации могут передавать одну и ту же информацию, однако оценка будет низкой из-за несовпадений в используемых словах.

3) Максимальная оценка у предсказанных аннотаций с помощью метода Луна меньше, чем у классификации, хотя в среднем метод Луна работает лучше. Такая ситуация может быть обусловлена особенностями алгоритма деревьев решений. Если многие слова из обрабатываемого им текста ранее встречались в тренировочных данных и были включены в базовый мешок слов, то деревья решений могут опираться на большое количество слов, наличие которых влияет на предсказание. Тогда предложенная алгоритмом аннотация будет иметь достаточно высокую оценку. Но если слова из текста имеют очень мало пересечений с содержимым мешка слов, то тогда алгоритм заведомо будет классифицировать предложения такого текста как не подходящие для краткого содержания.

## 7. Примеры работы алгоритмов и их анализ

Для того чтобы в полной мере судить о качестве работы того или иного метода, рассмотрим несколько конкретных примеров.

На рисунке 1 представлен исходный текст и существующая аннотация одной из новостей, являющейся объектом из тестового набора данных.



The screenshot shows a Jupyter Notebook interface. The top bar includes tabs for 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help', along with a 'Доверенный' (Trusted) status indicator and 'Python 3' version information. Below the top bar is a toolbar with icons for file operations, navigation, and execution, including a 'Запуск' (Run) button. The main content area is divided into two sections: 'Исходный текст' (Original text) and 'Оригинальная аннотация' (Original annotation). The 'Исходный текст' section contains a detailed news article about a civilian plane being forced to land in Syria. The 'Оригинальная аннотация' section contains a summary of the event.

**Исходный текст**

Гражданский самолет со 172 пассажирами на борту, летевший из Тегерана в Дамаск, 6 февраля был вынужден совершить экстренную посадку на российской базе Хмеймим в Сирии из-за опасности попасть под удары сирийской системы противовоздушной обороны (ПВО), отражавшей атаку Израиля. Как отметили в российском Минобороны, лайнер Airbus A320 оказался в опасной зоне, когда совершал заход на посадку в аэропорту Дамаска. Чтобы избежать опасности для пассажиров, диспетчеры аэропорта перенаправили воздушное судно на российскую авиабазу Хмеймим. Официальный представитель Минобороны РФ Игорь Конашенков рассказал ТАСС, что генштаб Израиля в последнее время регулярно использует гражданские самолеты с людьми «для прикрытия или блокирования ответных действий сирийских сил ПВО», и это уже становится характерной чертой израильских ВВС. «Лишь благодаря оперативным действиям диспетчеров аэропорта Дамаск и эффективной работе автоматизированной системы управления воздушным движением удалось увести пассажирский Airbus A320 из зоны поражения сирийской ПВО и благополучно посадить его на ближайший запасной аэродром – на российской авиабазе Хмеймим», – цитирует РИА «Новости» заявление Минобороны. По сообщению российского оборонного ведомства, после 2 часов ночи 6 февраля израильские ВВС нанесли внезапный удар по пригородам Дамаска. Восемь ракет «воздух-земля» были выпущены из четырех самолетов-истребителей F-16 без захода в воздушное пространство Сирии. Для отражения удара сирийцы применили комплексы ПВО. Большинство ракет не достигло своих целей благодаря работе комплексов противовоздушной обороны, пишет сирийское агентство SANA. В то же время, по данным израильского Haaretz, жертвами ударов стали восемь сирийских военных и еще 15 бойцов из проиранской боевой группировки. Конашенков при этом отдельно заметил, что регулярные пассажирские рейсы в Сирии и во всем мире проходят в известных эшелонах высоты, которые радиолокационные средства Израиля хорошо видят, а такие «операции израильских стратегов, к сожалению, ни во что не ставят жизни сотен ни в чем не повинных гражданских людей». МИД России еще осенью выразили обеспокоенность насчет атак Израиля по территории Сирии, которые происходят все чаще. Так, в ночь на 20 ноября израильские самолеты обстреляли пригороды Дамаска, выпустив 40 крылатых ракет, в результате чего погибли и были ранены более десяти сирийских военных и граждан.

**Оригинальная аннотация**

В связи с ударами израильских истребителей по пригородам Дамаска под огонь систем ПВО Сирии чуть не попал пассажирский лайнер Airbus A320, летевший из Тегерана в сирийскую столицу. Как отмечают в российском Минобороны, практика прикрывать удары гражданскими судами становится характерной для ВВС Израиля.

Рисунок 1 - Исходный текст новости и его аннотация, пример 1

Метод Луна для итоговой аннотации предсказал предложения с индексами 3 и 4 (количество предложений такое же, как и в оригинальной аннотации). Полученное краткое содержание, состоящее из этих предложений, можно увидеть на рисунке 2.

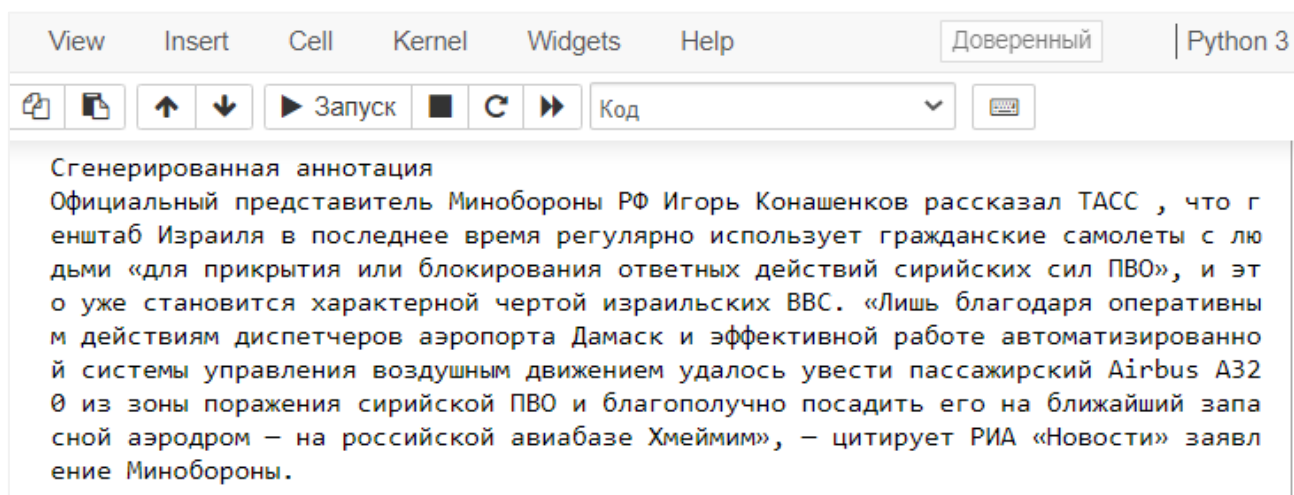


Рисунок 2 - Аннотация, предложенная методом Луна к примеру 1

Оценка данной аннотации равняется примерно 0.2062, это средний результат среди всех оценок кратких содержаний, предложенных методом Луна. Сразу бросается в глаза, что метод выбрал длинные предложения, которые содержат много специализированной лексики и собственных имен. Вероятно, эти слова как раз попали в коллекцию самых часто встречающихся в тексте, поэтому у данных объемных предложений, содержащих много таких слов, оказался самый высокий ранг.

Если говорить о качестве предложенной алгоритмом аннотации, то по смысловой нагрузке она похожа на шаблонное краткое содержание: сказано и про лайнер, чуть не попавший под огонь, и про использование гражданских судов как прикрытие от ударов. Конечно, есть несколько лишних подробностей, которые мешают восприятию, но извлекающее аннотирование не способно избегать таких неудобств. Исходя из полноты тематического содержания сгенерированной аннотации, ее оценка должна быть явно выше, чем 0.2062.

Для этого же текста метод кластеризации предложил краткое содержание из предложений 4 и 12, представленное на рисунке 3.

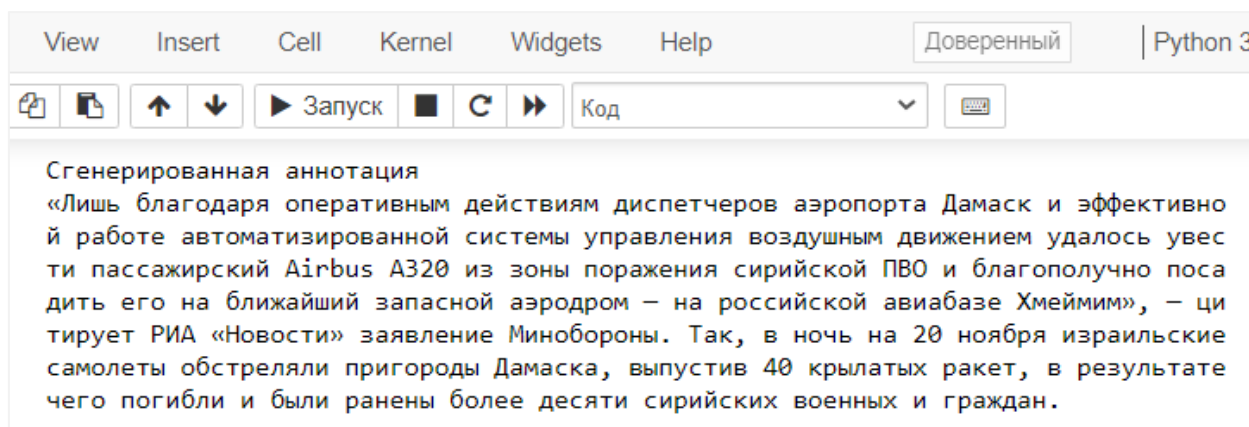


Рисунок 3 - Аннотация, предложенная методом кластеризации к примеру 1

Оценка для этой аннотации равна 0.1027, то есть ниже, чем у результата работы метода Луна. И предсказанные данными методами аннотации отличаются одним из предложений. Если учесть суть алгоритма кластеризации, то очевидно, что предложения 4 и 12 наиболее близки к центрам двух кластеров, которые должны отражать разные темы, затронутые в исходном тексте. Эти предложения действительно передают совершенно разную информацию: с одной стороны, «аэропорт», «воздушное движение», «спасенный пассажирский Airbus», с другой – «крылатые ракеты», «погибшие, раненые военные и граждане». Имеющиеся различия в использованной лексике повлекли распределение этих предложений по разным кластерам. Другой вопрос в том, что тема второго кластера является несколько второстепенной, поэтому в действительной аннотации опущена такая информация. Но в целом, результат работы метода кластеризации на данном примере оказался ожидаемым и достаточно корректным, с точки зрения человеческого восприятия стоило бы выше оценить созданное краткое содержание.

Теперь рассмотрим краткое содержание, которое для исходного текста было предложено методом классификации с помощью деревьев решений. Оно зафиксировано на рисунке 4.

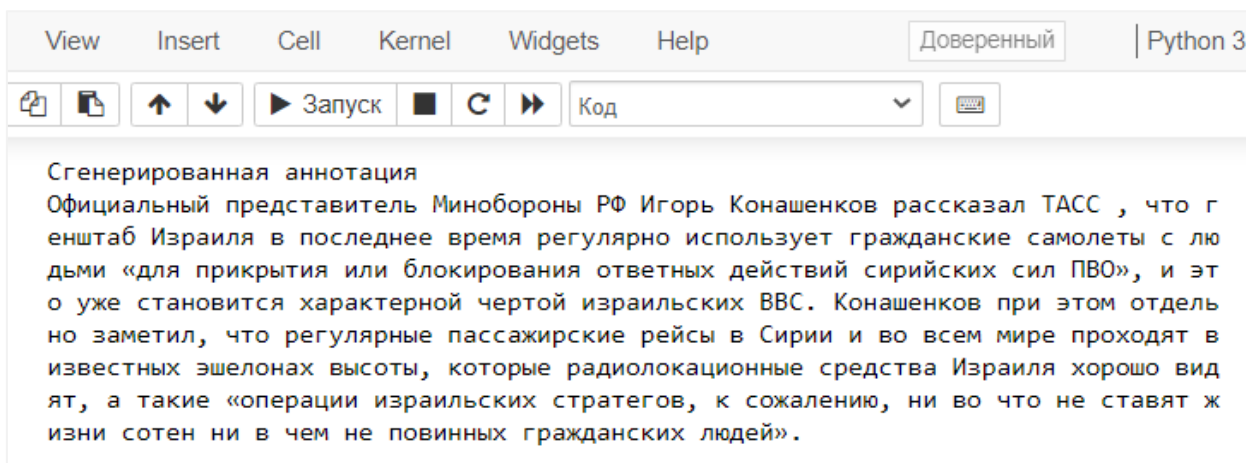


Рисунок 4 - Аннотация, предложенная методов классификации к примеру 1

В аннотации присутствуют предложения с индексами 3 и 10, предложение 3 так же было использовано в результате работы метода Луна. Оценка данного краткого содержания на основании его совпадения с имеющейся аннотацией составила 0.1481 - средний результат между методами Луна и кластеризации. Нужно заметить, что работа алгоритма, использующего деревья решений, наименее предсказуема, так как лес деревьев решений обучен на предложениях из тренировочных данных и совершенно не зависит от исходного тестового текста. Присутствующие в обоих предложениях слова «гражданские», «израильские» наталкивают на мысль о том, что именно эти слова содержатся в базовом мешке слов и каким-то образом повышают рейтинг предложений. Сгенерированная алгоритмом классификации аннотация в целом передает основную мысль исходного текста, но упускает информацию о спасении пассажирского лайнера, ради которой, вероятно, и была написана данная новость.

Также интересно рассмотреть такой пример, когда итоговая аннотация совершенно не похожа на шаблонное краткое содержание, то есть имеет оценку 0.0. Изображение с исходным текстом размещено в приложении [1], а на рисунке 5 представлена действительная аннотация и краткое содержание, предложенное методом кластеризации.

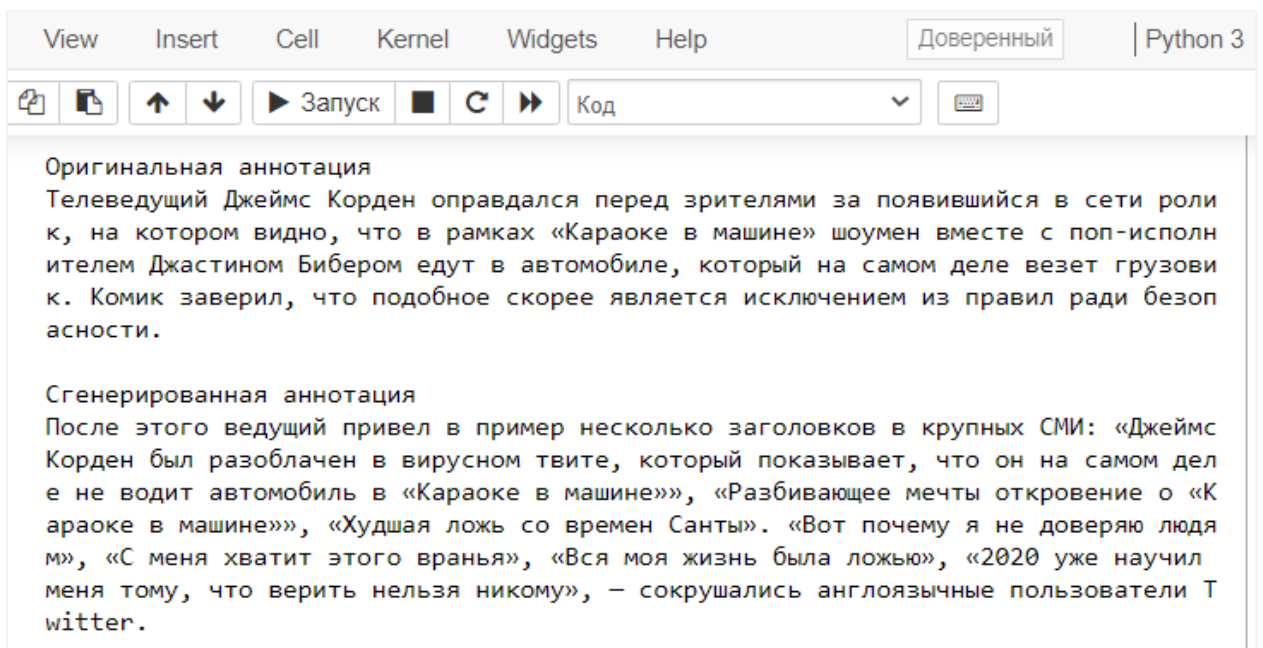


Рисунок 5 – Исходная и сгенерированная аннотации к примеру 2

Легко понять, почему оценка сгенерированной аннотации нулевая – понятия «Караоке» и «Джеймс Корден» маловероятно присутствовали в базовом мешке слов, поэтому вектора-предложения этих кратких изложений существенно различались. В любом случае, метод кластеризации на данном примере показывает очень плохой результат, наивно выбирая предложения с множеством слов из несущественных цитат, благодаря разнообразию которых этим предложениям удалось оказаться наиболее близкими к центрам кластеров.

Но не всегда нулевая оценка – признак плохой аннотации. Так, например, алгоритм классификации сгенерировал для текста, размещенного в приложении [2], краткое содержание, представленное на рисунке 6. Данные аннотации совершенно разные по смысловой нагрузке, чем обусловлена низкая оценка результата работы алгоритма. Однако содержание сгенерированной аннотации передает очень важную информацию, которую некоторые люди, возможно, выделили бы в исходном тексте как главную. Этот пример показывает, что оценки предсказанных методами кратких содержаний могут быть неоправданно занижены.



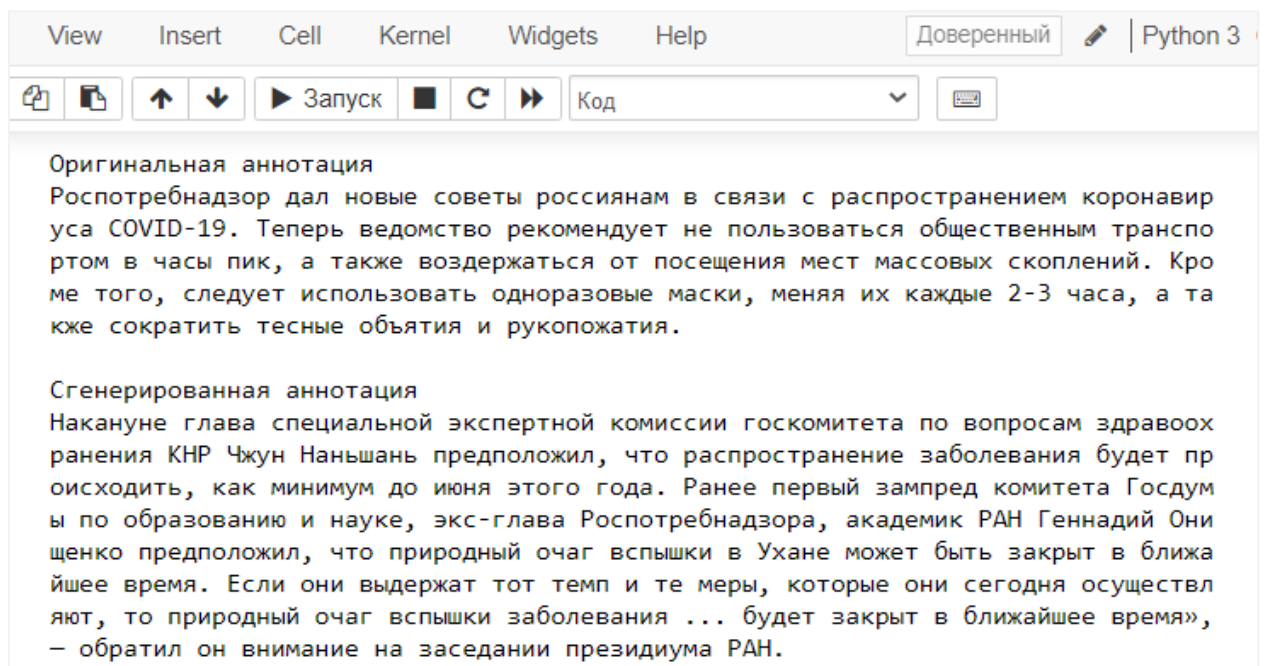


Рисунок 6 - Исходная и сгенерированная аннотации к примеру 3

## 8. Выводы

После анализа конкретных примеров работы алгоритмов и результатов проведенных экспериментов можно сделать несколько выводов.

1) Лучшие результаты показали методы, которые обрабатывали предложения из исходного текста в совокупности. Даже поверхностный алгоритм подсчета частот слов в рамках метода Луна оказался более действенным, чем классификация на основе деревьев решений. Потому что метод Луна работал с текстом целиком, а классификатор обрабатывал отдельно каждое предложение из тестовых данных без какого-либо контекста.

2) Кластеризация как способ автоматического аннотирования получила наибольшие оценки. Может быть, это связано с тем, что данный метод схож с человеческим процессом сокращения текста. И ведь действительно, сначала мы выделяем основные темы, затронутые в исходном тексте – метод осуществляет это с помощью выбора центра кластеров. Затем выбираем или перефразируем такие предложения, чтобы они затрагивали все главные темы и передавали их идею наиболее емко – алгоритм распределяет все предложения по кластерам и выбирает наиболее близкие к центрам. Это наблюдение вновь наводит на мысль о том, что лучший метод аннотирования – человеческий анализ.

3) Выбранный способ оценки результатов работы метода – сравнение векторов предложений из предсказанной аннотации с предложениями из шаблонного краткого содержания, оказался довольно некорректным. Перефразирование исходного текста, используемое в действительных аннотациях, приводило к абсолютному несовпадению векторов-предложений, даже если они были схожими по смыслу.

## ЗАКЛЮЧЕНИЕ

В рамках данной работы для достижения поставленной цели – сравнить несколько методов аннотирования текста, были выполнены следующие задачи:

1) Была изучена общая классификация всех существующих методов автоматического аннотирования, для сравнения выбраны 3 способа – метод Луна, классификация с помощью деревьев решений и кластеризация;

2) После анализа существующих этапов предобработки текста была выделена основная последовательность предобработки данных для решения поставленной задачи – токенизация, нормализация, лемматизация и векторизация TF-IDF;

3) В качестве способа оценки работы алгоритмов был выбран метод сравнения угла между векторами предложений;

4) Для обучения модели классификатора и тестирования методов был найден готовый набор данных русскоязычных новостей с их сокращенными изложениями. Также были выбраны наиболее подходящие библиотеки для работы с текстами и для использования реализованных алгоритмов выбранных методов аннотирования – NLTK, PyMystem3, scikit-learn, XGBoost;

5) На языке Python были написаны функции для предобработки текстов, подготовки данных к их использованию в алгоритмах аннотирования и для оценки работы методов;

6) Была обучена модель классификатора и собраны результаты работы методов на тестовых данных;

7) Итоговые оценки работы каждого метода были проанализированы, для чего потребовалось рассмотрение конкретных примеров аннотирования.

С учетом результатов проведенных экспериментов можно рекомендовать внесение следующих правок и дополнительные исследования:

- Для того чтобы автоматическая система оценки работы алгоритмов аннотирования была более точной, необходимо сравнивать не вектора из слов, используемых в предложениях, а тематики, затрагиваемые в аннотациях. Делать это можно, наверное, с помощью метода кластеризации, который показал хороший результат в данном исследовании, или использовать более сложные алгоритмы латентно-семантического анализа.

- Большое влияние на работу методов оказывает мешок слов, утвержденный после обработки тренировочного набора данных, по которому векторизовались все предложения. Внимание стоит уделить цифрам и числам, которые в данной работе при нормализации данных не удалялись. Так же повлиять на содержимое мешка слов можно с помощью ограничений: например, установить минимальный процент используемости слова, при котором оно включается в базовый набор характеристик.

- Чтобы классификатор с помощью деревьев решений мог делать более уверенные предсказания, он должен обучиться на большем количестве данных. Поэтому для данного метода есть необходимость поиска готовых больших наборов данных с текстами и готовыми аннотациями к ним или самостоятельного сбора подобных данных с интернет-изданий новостных газет.

- Значительно лучших результатов можно попробовать добиться, используя для задачи извлекающего аннотирования глубокие методы, которые на данный момент считаются наиболее перспективными в данной области. Хотя их использование больше оправданно в рамках вопроса генерирующего аннотирования.

За время выполнения курсовой работы были реализованы следующие компетенции:

Шифр компетенции	Расшифровка приобретаемой компетенции	Расшифровка освоения компетенции
УК-6	Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни	Проведение данного исследования шло по плану, составленному в начале работы с указанием сроков выполнения той или иной подзадачи.
ПК-2	Проверка работоспособности и рефакторинг кода программного обеспечения	В течение всей работы отдельные методы программы были протестированы на различных искусственных примерах и, при необходимости, оптимизированы.
ПК-3	Интеграция программных модулей и компонент и верификация выпусков программного продукта	Отдельно написанные методы и встроенные алгоритмы библиотек были объединены в последовательную программу.
ПК-4	Разработка требований и проектирование программного обеспечения	Для программы, сопровождающей данное исследование, были выдвинуты требования, по которым проводилась разработка.
ПК-5	Оценка и выбор варианта архитектуры программного средства	Разные логические компоненты программы выделены в отдельные функции, а их вызов осуществляется в методе main.
ПК-6	Разработка тестовых случаев, проведение тестирования и исследование результатов	Результаты работы методов аннотирования были обработаны и сравнены между собой.

## СПИСОК ЛИТЕРАТУРЫ

1. Обзор и анализ методов автоматического аннотирования текста [Текст] / М.В. Шкурина, О.Ю. Сабинин // Theoretical & Applied Science. – декабрь 2018.
2. The Automatic Creation of Literature Abstracts [Текст] / Н. Р. Luhn // IBM journal. – апрель 1958.
3. Применение метода дерева решений в задачах классификации и прогнозирования [Текст] / А.А. Мифтахова // «Инфокоммуникационные технологии» Том14, №1, с. 64-70. – 2016.
4. Алгоритмы и программы автоматической обработки текста [Текст] / В.А. Яцко // Вестник ИГЛУ. – 2012.
5. Предобработка текста для решения задач NLP [Текст] / Р.К. Акжолов, А.В. Верига // Международный научный журнал «Вестник науки» №3. – март 2020.
6. Основы NLP для текста [Электронный ресурс] – 2019. – URL: <https://habr.com/ru/company/Voximplant/blog/446738/> (дата обращения 10.03.2021).
7. XGBoost [Электронный ресурс]. – URL: <https://xgboost.ai/> (дата обращения 12.04.2021).
8. Scikit-learn [Электронный ресурс]. – URL: <https://scikit-learn.org/stable/> (дата обращения 3.04.2021).
9. NLTK [Электронный ресурс]. – URL: <https://www.nltk.org/> (дата обращения 10.03.2021).
10. Регулярные выражения [Электронный ресурс]. – URL: <https://docs.python.org/3/library/re.html> (дата обращения 27.03.2021).
11. Быстрая лемматизация (PyMorphy2 или PyMystem3) [Электронный ресурс] – 2020. – URL: <https://habr.com/ru/post/503420/> (дата обращения 27.03.2021).
12. PyMystem3 [Электронный ресурс]. – URL: <https://pypi.org/project/pymystem3/> (дата обращения 27.03.2021).

13. Gazeta dataset [Электронный ресурс]. – URL: <https://github.com/IlyaGusev/gazeta> (дата обращения 2.03.2021).
14. JetBrains PyCharm [Электронный ресурс]. – URL: <https://www.jetbrains.com/ru-ru/pycharm> (дата обращения 12.02.2021).
15. Jupyter notebook [Электронный ресурс]. – URL: <https://jupyter.org/> (дата обращения 12.02.2021).
16. Anaconda [Электронный ресурс]. – URL: <https://www.anaconda.com/> (дата обращения 12.02.2021).
17. Pickle [Электронный ресурс]. – URL: <https://docs.python.org/3/library/pickle.html> (дата обращения 3.04.2021).
18. NumPy [Электронный ресурс]. – URL: <https://numpy.org/> (дата обращения 10.03.2021).
19. Tqdm [Электронный ресурс]. – URL: <https://github.com/tqdm/tqdm> (дата обращения 3.04.2021).

## ПРИЛОЖЕНИЕ

### Приложение 1. Исходный текст новости, пример 1.

ViewInsertCellKernelWidgetsHelp

ДоверенныйPython 3

↑

↓

▶

■

↺

↻

▶▶

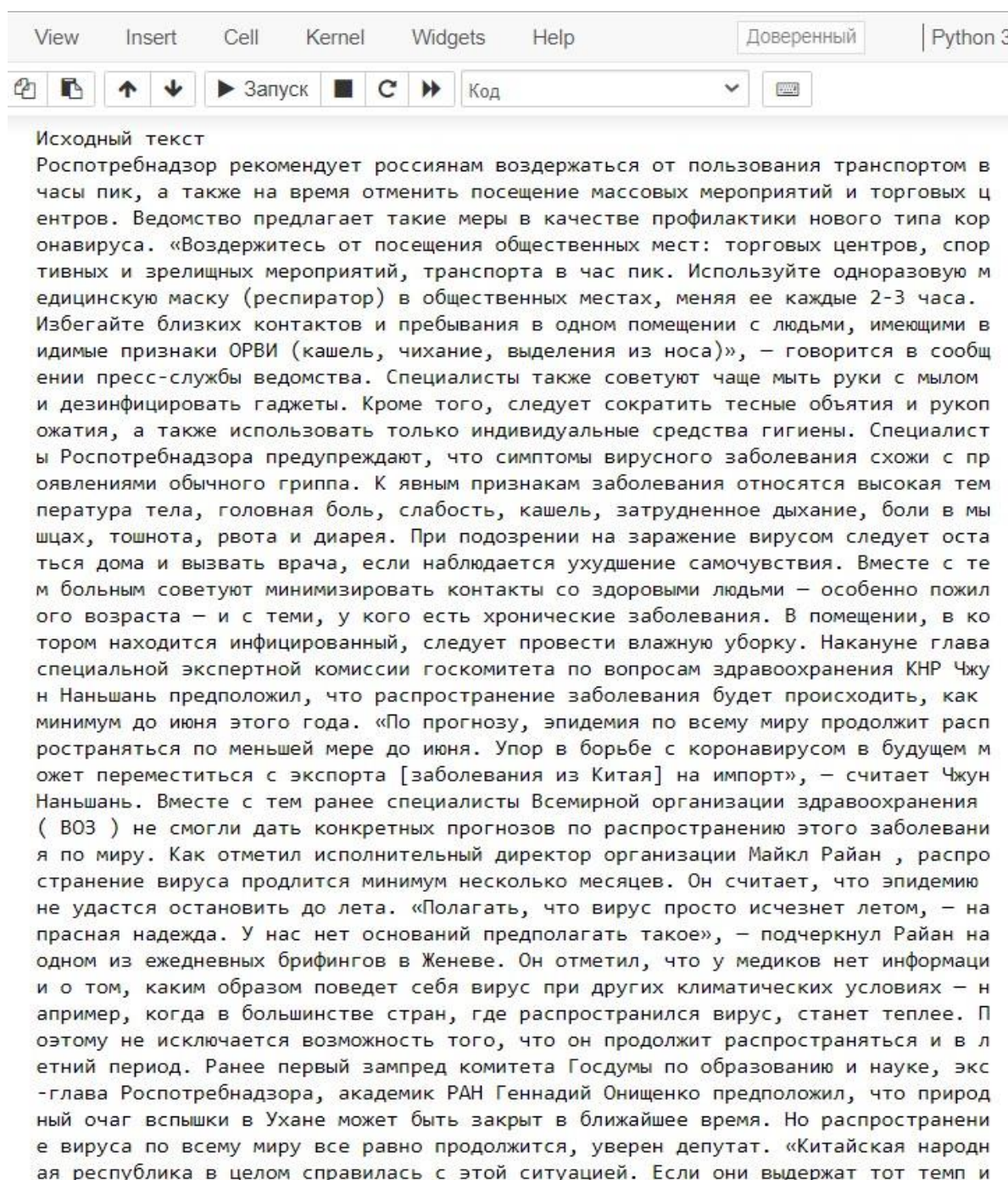
Код

Исходный текст

Ведущий популярного телешоу «Караоке в машине» Джеймс Корден прокомментировал недавний скандал со своим участием, когда пользователи интернета уличили его в том, что в действительности не он водит автомобиль во время съемок своей передачи. Видеообращение Кордена было опубликовано на YouTube-канале «The Late Late Show with James Corden». Телеведущий начал с того, что назвал неправдой все обвинения в свой адрес. Далее он продемонстрировал видео, которое попало несколько дней назад в Twitter. На нем видно, что Корден и поп-звезда Джастин Бибер едут в машине, на которой установлен специальный прицеп, и ее везет грузовик. «Я знаю – это выглядит плохо, – сказал знаменитый англичанин, выдержав драматическую паузу. – Но я просто хочу сказать сейчас, что я всегда сам вожу машину, пока мы не делаем что-то, что может показаться небезопасным. Например, танцы, смена костюмов или если я пьян». Что касается ситуации с Бибером, по словам ведущего, решение воспользоваться страховкой было принято из соображений безопасности, поскольку он боялся «опростоволоситься» в присутствии молодого исполнителя. Далее Корден отметил, что «разоблачительный ролик» собрал более 14 млн просмотров, тогда как сами выпуски «Караоке в машине» не могут похвастаться подобным вниманием зрителей. После этого ведущий привел в пример несколько заголовков в крупных СМИ: «Джеймс Корден был разоблачен в вирусном твите, который показывает, что он на самом деле не водит автомобиль в «Караоке в машине»», «Разбивающее мечты откровение о «Караоке в машине»», «Худшая ложь со времен Санты». Кроме того, шоумен пошутил, что шокирован тем, что «сделал что-то, расстроившее людей больше, чем фильм «Кошки». По словам Кордена, он был поражен, сколь рьяно пользователи соцсетей набросились на него. «Вот почему я не доверяю людям», «С меня хватит этого вранья», «Вся моя жизнь была ложью», «2020 уже научил меня тому, что верить нельзя никому», – сокрушались англоязычные пользователи Twitter. Особое внимание телеведущий уделил пользователю с ником Assgaze («Пристальный взгляд из задницы» – англ.), который назвал его «гребаным лгуном». «Есть люди, с кем я мог быть не до конца честен, но я никогда не солгал бы Assgaze», – подчеркнул комик. Затем Корден поклялся зрителям, что в 95% случаев он действительно «подвергает опасности» жизни величайших мировых поп-звезд. «Но это телешоу, и не все в нем реально. Наша передача никогда не снимается в полночь, мы снимаем его в 5 вечера и притворяемся, что сейчас поздно, – продолжил шоумен. – Мне казалось, что такие простые вещи и так все знают. Я сожалею, что вы так глубоко погрузились в реальность «Караоке в машине», но это телевидение и мы часто делаем вещи только ради развлечения». В конце своей речи Корден предложил зрителям посмотреть новый сезон передачи, где все «действительно сами водят машину, кроме тех моментов, когда они этого не делают».



## Приложение 2. Исходный текст новости, пример 2.



те меры, которые они сегодня осуществляют, то природный очаг вспышки заболевания ... будет закрыт в ближайшее время», – обратил он внимание на заседании президиума РАН. По его мнению, ситуация «не затихнет», потому что за пределами Китая зафиксировано свыше 32 тыс. заражения коронавирусом. «На сегодняшний день самое интенсивное проявление инфекции в Корее, на втором месте Италия, на третьем – Иран, Китай на 4 месте», – обратил внимание Онищенко. В Москве ранее ввели меры, которые позволяют бороться с распространением заболевания. Так, всем прибывшим из потенциально опасных стран следует находиться на карантине в течение 14 дней. Таким образом, в течение этого времени должен быть оформлен больничный на работе. Как ранее отметили в Минтруде, работодатели будут оплачивать такой больничный так же, как и обычный. При обнаружении первых симптомов заболевания следует остаться дома и вызвать врача, чтобы можно было сдать анализы на наличие вируса. Самостоятельной проверки заболевания, как отметили в пресс-службе департамента здравоохранения Москвы, нет. При этом там отметили, что для постановки диагноза пациенту без симптомов ОРВИ проводятся два исследования. Если у пациента есть симптомы ОРВИ, то его обследуют трижды. «Если вы здоровы, то ежедневно о своем самочувствии и температуре тела вы сообщаете в поликлинику по месту жительства в течение всего периода карантина – 14 дней. Если вам нужен больничный лист, необходимо обратиться в вашу поликлинику. Если у вас появились симптомы ОРВИ, незамедлительно вызывайте врача», – говорится в рекомендациях департамента здравоохранения.

### Приложение 3. Модуль, отвечающий за предобработку текста.

```
import json

import pickle

import re

import numpy as np

import nltk


from nltk.corpus import stopwords

from pymystem3 import Mystem

from sklearn.feature_extraction.text import TfidfVectorizer


def save_to(pickle_file, data):

    with open(pickle_file, 'wb') as f:

        pickle.dump(data, f)


# распаковка файла и выбор нужных данных оттуда
def unpack(file):

    with open(file, "r", encoding='utf-8') as f:

        json_list = list(f)

    all_news_text_X = []

    all_news_summ_Y = []

    for i, str in enumerate(json_list):

        one_news = json.loads(str)

        del one_news['url']

        del one_news['title']

        del one_news['date']

        all_news_text_X.append(one_news['text'])

        all_news_summ_Y.append(one_news['summary'])
```

```

    return all_news_text_X, all_news_summ_Y

# разбиение листа внутри на несколько листов по заданному
разделителю

def splitting_list(list_, delimiter):
    splitted = [[]]
    for lemma in list_:
        if lemma == delimiter:
            splitted.append([])
        else:
            splitted[-1].append(lemma)
    return splitted

# предобработка (нормализация, лемматизация)

def preprocess(text_str, punctuation_re):
    mystem = Mystem()
    rus_stopwords = stopwords.words("russian")
    text_lemmas = mystem.lemmatize(text_str.lower())
    text_lemmas = [punctuation_re.sub('', lemma) for lemma in
text_lemmas ] # удаляю все кроме букв и цифр
    text_lemmas = [lemma for lemma in text_lemmas if
lemma.strip() != '' and lemma not in rus_stopwords]
    return text_lemmas

# TF-IDF векторизация с созданием базового мешка слов

def tfidf_bag_words_make_vocab(all_sent_list):
    tf_vectorizer = TfidfVectorizer(max_features=50000,
max_df=0.85) # слово встречается > чем в 85% -> не берем
    all_sent_str = [" ".join(one_sent_list) for one_sent_list in
all_sent_list]

```

```

    bag_of_words = tf_vectorizer.fit_transform(all_sent_str)

    vocabulary = tf_vectorizer.get_feature_names()

    return bag_of_words, vocabulary

# TF-IDF векторизация по базовому мешку слов
def tfidf_bag_words_with_vocab(all_sent_list, train_vocabulary):

    tf_vectorizer = TfidfVectorizer(vocabulary =
train_vocabulary)

    all_sent_str = [" ".join(one_sent_list) for one_sent_list in
all_sent_list]

    bag_of_words = tf_vectorizer.fit_transform(all_sent_str)

    return bag_of_words

# сравнение двух векторов-предложений
def score_compare_sent(vec_X, vec_Y):

    # угол между векторами

    vec_X_unit = (vec_X / np.linalg.norm(vec_X)) if
np.linalg.norm(vec_X) != 0 else np.zeros(vec_X.shape)

    vec_Y_unit = (vec_Y / np.linalg.norm(vec_Y)) if
np.linalg.norm(vec_Y) != 0 else np.zeros(vec_Y.shape)

    angles = np.arccos(np.clip(np.dot(vec_X_unit, vec_Y_unit), -
1.0, 1.0))

    # привожу [0 - хорошо, pi/2 - плохо] к [0 - плохо, 1 -
хорошо]

    score_01 = 1 - (angles / (np.pi/2))

    return score_01

# оценка всех предложений по их схожести с аннотацией
def score_for_all_sent(tfidf_text_X, tfidf_text_Y):

    score_all_sent = []

    for text_index in range(len(tfidf_text_X)):

```

```

sentences_X = tfidf_text_X[text_index].toarray()
sentences_Y = tfidf_text_Y[text_index].toarray()

for vec_x in sentences_X:
    best_min_score = np.min([score(vec_x, vec_y) for
vec_y in sentences_Y])

    score_all_sent.append(best_min_score)

return np.array(score_all_sent)

```

# вспомогательный метод предобработки с формированием пакетов и токенизацией по предложениям

```

def preprocessing_big_text_pack(all_text, punctuation_re):

    all_preproc_sent_text = []
    count_sent_in_text = []
    pack_size = 1000
    start_ind = 0
    while start_ind < len(all_text):
        pack_list_for_preproc = []
        end_ind = start_ind + pack_size
        if end_ind > len(all_text):
            end_ind = len(all_text)

        # соединение и предобработка
        for i in range(start_ind, end_ind):
            sentences = nltk.sent_tokenize(all_text[i])
            count_sent_in_text.append(len(sentences))

            pack_list_for_preproc.append(' bbreakk
'.join(sentences)) # между предложениями'bbreakk'

            pack_str_for_preproc = ' bbreakk
'.join(pack_list_for_preproc) # между текстами'bbreakk'

```

```

        pack_lemmas = preprocess(pack_str_for_preproc,
punctuation_re)

        # разделение на предложения отдельные

        preproc_sentences = splitting_list(pack_lemmas,
'bbreakk')

        all_preproc_sent_text.extend(preproc_sentences)

        start_ind += pack_size

        return np.array(all_preproc_sent_text),
np.array(count_sent_in_text)

# разделение предложений по текстам, а тексты по наборам X и Y
def split_sent_in_text_one_pack(sent, count_X, count_Y):
    all_text_X = []
    all_text_Y = []
    already_taked = 0
    for count in count_sent_in_text_X:
        all_text_X.append(sent[already_taked:already_taked +
count])
        already_taked += count
    for count in count_sent_in_summ_Y:
        all_text_Y.append(sent[already_taked:already_taked +
count])
        already_taked += count
    return all_text_X, all_text_Y

# разделение предложений по текстам
def split_sent_in_text_two_pack(sent_X, sent_Y, count_X,
count_Y):
    all_text_X = []
    all_text_Y = []

```

```

    already_taked = 0

    for count in count_sent_in_text_X:
        all_text_X.append(sent_X[already_taked:already_taked +
count])

        already_taked += count

    already_taked = 0

    for count in count_sent_in_summ_Y:
        all_text_Y.append(sent_Y[already_taked:already_taked +
count])

        already_taked += count

    return all_text_X, all_text_Y

if __name__ == '__main__':
    punctuation_re = re.compile('[^a-zA-Za-яA-Я0-9]+')

    all_text_X_train, all_text_Y_train = unpack(

'C:/Users/Acer/PycharmProjects/ML_NLP_course3/gazeta_jsonl/gazet
a_train.jsonl')

    all_text_X_val, all_text_Y_val = unpack(

'C:/Users/Acer/PycharmProjects/ML_NLP_course3/gazeta_jsonl/gazet
a_val.jsonl')

    all_text_X_test, all_text_Y_test = unpack(

'C:/Users/Acer/PycharmProjects/ML_NLP_course3/gazeta_jsonl/gazet
a_test.jsonl')

# предобработка тестовых данных аналогичная валидационным

    all_preproc_sent_X_train, count_sent_in_text_X_train =
preprocessing_big_text_pack(all_text_X_train, punctuation_re)

    all_preproc_sent_Y_train, count_sent_in_text_Y_train =
preprocessing_big_text_pack(all_text_Y_train, punctuation_re)

```



```

    all_preproc_sent_X_val, count_sent_in_text_X_val =
preprocessing_big_text_pack(all_text_X_val, punctuation_re)

    all_preproc_sent_Y_val, count_sent_in_text_Y_val =
preprocessing_big_text_pack(all_text_Y_val, punctuation_re)


    # надо обработанные тренировочные X Y объединить, чтобы
сделать общий мешок слов

    all_sent_train = np.concatenate((all_preproc_sent_X_train,
all_preproc_sent_Y_train), axis=None)


    tfidf_XY_train, train_vocabulary =
tfidf_bag_words_make_vocab(all_sent_train)

    tfidf_X_val = tfidf_bag_words(all_preproc_sent_X_val,
vocabulary)

    tfidf_Y_val = tfidf_bag_words(all_preproc_sent_Y_val,
vocabulary)


    # получить список предложений тренировочного набора X, не
разделенных на тексты, для обучения модели

    tfidf_no_text_X_train =
tfidf_XY_train[:np.sum(count_sent_in_text_X_train)]


    # разделить предложения братно по текстам, а тексты по
наборам X и Y

    tfidf_text_X_train, tfidf_text_Y_train =
split_sent_in_text_one_pack(
tfidf_XY_train, count_sent_in_text_X_train,
count_sent_in_text_Y_train

    )

    tfidf_text_X_val, tfidf_text_Y_val =
split_sent_in_text_two_pack(
tfidf_X_val, tfidf_Y_val, count_sent_in_text_X_val,
count_sent_in_text_Y_val

    )

```

```
# для каждого предложения получить его оценку

score_Y_train = score_for_all_sent(tfidf_text_X_train,
tfidf_text_Y_train)

score_Y_val = score_for_all_sent(tfidf_text_X_val,
tfidf_text_Y_val)


# сохранение необходимых данных в файлы .pickle
```

#### **Приложение 4. Модуль, отвечающий за работу алгоритмов аннотации.**

```
import numpy as np

import pickle

import xgboost as xgb

from tqdm.auto import tqdm


def load_from(pickle_file):

    with open(pickle_file, 'rb') as f:

        data = pickle.load(f)

    return data


# Метод Луна: поиск наиболее частых слов

def choose_most_freq_words(bag_of_words):

    freq = bag_of_words.sum(axis=0) # подсчет частоты каждого
    слова

    sort_freq_index = np.argsort(freq)

    top_words_index = sort_freq_index[-int(len(freq) / 10):] #
    10% наиболее частых

    return top_words_index


# Метод Луна: подсчет ранга предложений

def count_range_sentences(bag_of_words, top_words_index):

    sent_range_list = []

    for i, sent in enumerate(bag_of_words): # take word-index
    string

        use_words_index = np.nonzero(sent)

        words_count = len(use_words_index)

        top_words_count = len(np.intersect1d(top_words_index,
        use_words_index))
```

```

        sent_range = (top_words_count ** 2) / words_count

        sent_range_list.append(sent_range)

    return sent_range_list

# Метод Луна: выбор предложений для аннотации
def choose_sent_for_summ(sent_range_list, y_sent_count):

    sort_sent_range = np.argsort(sent_range_list)

    sent_for_summ_index = np.sort(sort_sent_range[-
y_sent_count:])

    return sent_for_summ_index

# Метод Луна
def method_Luhn(tfidf_texts_X, count_sent_in_text_Y):

    index_sent_for_summ = []

    for i, tfidf_text in tqdm(enumerate(tfidf_texts_X)):

        tfidf_text = tfidf_text.toarray()

        top_words_index = choose_most_freq_words(tfidf_text)

        sent_range_list = count_range_sentences(tfidf_text,
top_words_index)

        sent_for_summ_index =
choose_sent_for_summ(sent_range_list, count_sent_in_text_Y[i])

        index_sent_for_summ.append(sent_for_summ_index)

    return index_sent_for_summ

# Классификация: обучение модели с валидацией
def tree_model_training(X_train, Y_train, X_val, Y_val):

    xgb_params = {'eta': 0.2,

                  'max_depth': 8,

```

```

        'subsample': 0.6,
        'colsample_bytree': 0.7,
        'objective': 'binary:logistic',
        'eval_metric': 'logloss',
    }

    num_rounds = 200

    d_train = xgb.DMatrix(X_train, Y_train)
    d_val = xgb.DMatrix(X_val, Y_val)
    evallist = [(d_val, 'val')]

    model_xgb = xgb.train(xgb_params, d_train, num_rounds,
evallist, early_stopping_rounds=10)

    return model_xgb

# Классификация: получение предсказания модели и выбор
предложений для аннотации
def tree_predict(model, X_test, count_sent_Y_test):

    d_test = xgb.DMatrix(X_test)

    Y_predict = model.predict(d_test, iteration_range=(0,
my_model.best_iteration))

    index_sent_for_summ = []

    new_text_start = 0

    for text_ind in tqdm(range(len(count_sent_X_test))):

        text_end = new_text_start + count_sent_X_test[text_ind]

        predict_for_one_text = Y_predict[new_text_start :
text_end]

        sort_index = np.argsort(predict_for_one_text)[::-1]

        # выбираем с наибольшим предсказанием

    index_sent_for_summ.append(sort_index[:count_sent_Y_test[text_in
d]])

    new_text_start = text_end

```

```

    return index_sent_for_summ

# Кластеризация: выбор предложений для аннотации
def clustering(tfidf_texts_X, count_sent_in_text_Y):
    index_sent_for_summ = []
    new_text_start = 0
    for i, text in tqdm(enumerate(tfidf_texts_X)):
        cluster_count = count_sent_in_text_Y[i]

        model_km = KMeans(n_clusters = cluster_count)
        cluster_field = model_km.fit(text)
        distance_to_cluster = model_km.transform(text)

        sent_ind_close_centroids =
np.argmin(distance_to_cluster, axis=0)

        index_sent_for_summ.append(sent_ind_close_centroids)
    return index_sent_for_summ

if __name__ == '__main__':
    tfidf_texts_X_test = load_from('file_name.pickle')
    count_sent_in_text_Y_test = load_from('file_name.pickle')
    tfidf_sents_X_train = load_from('file_name.pickle')
    tfidf_sents_Y_train = load_from('file_name.pickle')
    tfidf_sents_X_val = load_from('file_name.pickle')
    tfidf_sents_Y_val = load_from('file_name.pickle')
    tfidf_sents_X_test = load_from('file_name.pickle')

    # Получение индексов предложений для аннотации по методу
Луна

```

```

    Luhn_index_sent_for_summ = method_Luhn(tfidf_texts_X_test,
count_sent_in_text_Y_test)

# Обучение модели классификатора

classif_model = Tree_model_training(
    tfidf_sents_X_train, tfidf_sents_Y_train,
tfidf_sents_X_val, tfidf_sents_Y_val
)

# Получение индексов предложений для аннотации по методу
классификации

Tree_index_sent_for_summ = Tree_predict(
    classif_model, tfidf_sents_X_test,
count_sent_in_text_Y_test
)

# Получение индексов предложений для аннотации по методу
кластеризации

Cluster_index_sent_for_summ = clustering(tfidf_texts_X_test,
count_sent_in_text_Y_test)

# сохранение результатов работы методов

```

## **Приложение 5. Модуль, отвечающий за оценку полученных результатов.**

```
import json

import pickle

import numpy as np

import nltk

from tqdm.auto import tqdm


def load_from(pickle_file):

    # аналогично функции из модуля работы с методами


def unpack(file):

    # аналогично функции из модуля предобработки


def score_compare_sent(vec_X, vec_Y):

    # аналогично функции из модуля предобработки


# подсчет оценок каждого предложения и общей оценки каждого
# текста

def calc_best_scores_texts(index_sent_for_summ,
X_tfidf_texts_test, Y_tfidf_texts_test):

    index_sent_for_summ = np.array(index_sent_for_summ)

    all_best_scores_text = []

    for text_i in tqdm(range(len(X_tfidf_texts_test))):

        sent_X = X_tfidf_texts_test[text_i].toarray()

        sent_for_summ_predict =
sent_X[index_sent_for_summ[text_i]]

        sent_for_summ_real =
Y_tfidf_texts_test[text_i].toarray()
```



```

        best_scores_sent_this_text = []

        can_be_chosed_Y = np.zeros(len(sent_for_summ_real),
dtype=bool) # маска индексов Y , кот еще участвуют

        for vec_x in sent_for_summ_predict:

            all_scores = np.ma.array([score(vec_x, vec_y) for
vec_y in sent_for_summ_real], mask=can_be_chosed_Y)

best_scores_sent_this_text.append(np.max(all_scores))

            can_be_chosed_Y[np.argmax(all_scores)] = True

all_best_scores_text.append(np.sum(best_scores_sent_this_text)/len(best_scores_sent_this_text))

    return all_best_scores_text

if __name__ == '__main__':

    Luhn_index_sent_for_summ = load_from('file_name.pickle')

    Tree_index_sent_for_summ = load_from('file_name.pickle')

    Cluster_index_sent_for_summ = load_from('file_name.pickle')

    X_tfidf_texts_test = load_from('file_name.pickle')

    Y_tfidf_texts_test = load_from('file_name.pickle')

    Luhn_all_best_scores_text = calc_best_scores_texts(

        Luhn_index_sent_for_summ, X_tfidf_texts_test,
Y_tfidf_texts_test)

    # Общая оценка метода

print(np.sum(all_best_scores_text_luhn)/len(all_best_scores_text_luhn))

    # Максимальная оценка созданной методом аннотации

print(np.max(all_best_scores_text_luhn))

    # Tree_index_sent_for_summ и Cluster_index_sent_for_summ
находятся аналогично

```