

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ
ИМ. А.Н. ТИХОНОВА

Вдовкин Василий Алексеевич, группа БИВ-144

**МЕТОДЫ СОКРАЩЕНИЯ ТЕКСТА И ИЗВЛЕЧЕНИЯ
КЛЮЧЕВОЙ ИНФОРМАЦИИ ИЗ РУССКОЯЗЫЧНЫХ
НОВОСТНЫХ СТАТЕЙ**

Выпускная квалификационная работа
по направлению 09.03.01 Информатика и вычислительная техника
студентов образовательной программы бакалавриата
«Информатика и вычислительная техника»

Студент  В.А. Вдовкин

Рецензент

Руководитель
старший преподаватель
Ф.В. Строк



Москва 2018 г.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ
им. А.Н. ТИХОНОВА

«УТВЕРЖДАЮ»

Академический руководитель образовательной
программы
«Информатика и вычислительная техника»

Ю.И. Гудков

« ____ » декабря 2017 г.

ЗАДАНИЕ
на выпускную квалификационную работу бакалавра
студенту группы БИВ-144 Вдовкин Василий Алексеевич

1. Тема работы

Методы сокращения текста и извлечения ключевой информации из
русскоязычных новостных статей

2. Требования к работе

Получение моделей, способных извлекать факты из текста и производить
его сокращение с относительно хорошей точностью.

Практическое применение данных моделей на реальных данных в виде
web-сервиса, анализирующего русскоязычные новости.

3. Содержание работы

Обоснование значимости работы, описание предметной области.

Исследование алгоритмов нормализации текста, векторизации,
кластеризации (TF-IDF, K-means и т.д.), методов извлечения информации
и сокращения текста.

Сбор и подготовка датасета русскоязычных новостных статей.

Выбор метрик, определяющих точность моделей.

Использование рассмотренных алгоритмов на собранных данных для
получения моделей извлечения фактов и сокращения текста.

Разработка web-сервиса, демонстрирующего результаты работы
полученных моделей, в режиме реального времени.

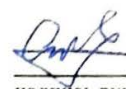
Выводы о проделанной работе.

4. Сроки выполнения этапов работы

Проект ВКР представляется студентом в срок до	«9» февраля 2018 г.
Первый вариант ВКР представляется студентом в срок до	«16» апреля 2018 г.
Итоговый вариант ВКР представляется студентом руководителю до загрузки работы в систему «Антиплагиат» в срок до	«15» мая 2018 г.

Задание выдано

«15» декабря 2017 г.

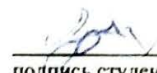


подпись руководителя

Ф.В. Строк

Задание принято к
исполнению

«15» декабря 2017 г.



подпись студента

В.А. Вдовкин

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»




МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ
им. А.Н. ТИХОНОВА

**График сдачи этапов
выпускной квалификационной работы бакалавра**

студента группы БИВ-144 Вдовкин Василий Алексеевич

Тема работы

Методы сокращения текста и извлечения ключевой информации из
русскоязычных новостных статей

Дата представления проекта ВКР	«9» февраля 2018 г.	 подпись руководителя	Ф.В. Строк
Дата представления первого варианта ВКР	«16» апреля 2018 г.	 подпись руководителя	Ф.В. Строк
Дата представления итогового варианта ВКР	«30» апреля 2018 г.	 подпись руководителя	Ф.В. Строк

Аннотация

Данная работа описывает процесс реализации сервиса, способного значительно улучшить и упростить пользовательское взаимодействие с новостным контентом с помощью обработки естественного языка (Natural Language Processing, NLP). Сервис анализирует поток русскоязычных новостных статей в реальном времени, группирует их по конкретным событиям и выделяет ключевую информацию о них. Для построения такого сервиса мы изучаем и используем различные методы и модели, распространенные в NLP для решения следующих задач: нормализация, векторизация, кластеризация и суммаризация текста. Кластеризация используется для выделения из потока данных множества статей, относящихся к одному событию. Для обучения векторизатора TF-IDF собираются и обрабатываются данные с web-сайтов медиа. Для кластеризации векторизованных статей используется K-means. Суммаризация извлекает самые информативные предложения из кластера-события и формирует параграф из 5 следующих по смыслу предложений. Для суммаризации используются две модели: SimBasic и DivRank. Выбранные решения сравниваются и оцениваются.

Abstract

This work describes implementation of the service that can significantly improve user experience in news content consumption by using Natural Language Processing (NLP for short). This service analyses stream of news articles from russian media websites in real-time, groups news by events and extracts the most valuable information about the events. To implement this service we first research and then use models and methods from NLP to find suitable solution for the following problems: normalization, vectorization, clusterization and summarization of text. Clusterization allows us to automatically group news by events. In order for this to work, we collect the data from web-sites of media and train TF-IDF model allows to use K-means algorithm for news clusterization. Summarization depends on SimBasic and DivRank models and extracts the most informative sentences from the event-cluster. We compare and evaluate the proposed solutions.

Содержание

1 Введение	7
2 Данные	8
3 Анализ текста	10
3.1 Нормализация	10
3.2 Векторизация	11
3.3 Кластеризация	13
3.4 Суммаризация	15
3.4.1 SumBasic	15
3.4.2 DivRank	16
4 Реализация сервиса	17
4.1 Архитектура и транспорт данных	17
4.2 Особенности реализации компонентов анализа	18
4.2.1 Система сбора данных	18
4.2.2 Модуль анализа текста	21
5 Оценка решений	22
6 Заключение	23
Список литературы	24

1 Введение

Когда в мире происходит какое-либо событие, различные средства массовой информации пишут статьи с информацией о нём в виде новостей. Пользователю часто бывает сложно ориентироваться в большом потоке данных от разных источников. Автоматическая систематизация и обработка таких данных с целью представить наиболее полную и информативную картину может сэкономить человеку много времени.

Человек очень просто понимает информацию, содержащуюся в тексте на естественном языке, потому что он учится этому с рождения, не замечая для себя, выучивая связи между устройством языка и информацией, которую с помощью него передают. С другой стороны, формализация этих правил очень сложна для людей, поэтому на ней сосредоточено множество разделов лингвистики. Сейчас редакторы новостных медиа почти полностью вручную выполняют все задачи, связанные с текстом: размечают теги, собирают подборки и, чаще всего, «генерируют» новости полностью опираясь статьи-источники.

С помощью математических моделей, описывающих связи между информацией и естественным языком, можно автоматизировать эти процессы. В последнее десятилетие быстрыми темпами развивается обработка естественного языка (Natural Language Processing, NLP), которая с помощью моделей и алгоритмов решает задачи автоматического анализа текста, в частности, объединение новостных статей в группы (кластеры) по релевантности к конкретному событию (кластеризация), извлечение из кластера наиболее информативных данных о событии, например, в виде нескольких предложений. Самые базовые и повседневные интернет-сервисы построены с использованием NLP: поиск, таргетинговая реклама, рекомендательные сервисы и т.п.

Целью выпускного проекта является разработка веб-сервиса, который автоматически группирует русскоязычные новостные статьи по событиям и извлекает из них ключевую информацию в режиме реального времени.

Для достижения данной цели необходимо решить **следующие задачи**:

1. Реализация системы сбора данных: сбор статей (парсинг) с web-сайтов СМИ для получения новостей вместе с их метаданными.
2. Изучение алгоритмов и моделей анализа текста: нормализация, векторизация, кластеризация и суммаризация.

3. Изучение технической реализации модуля анализа текста.
4. Проектирование и разработка инфраструктуры сервиса, интеграция с ранее реализованными модулями сбора и анализа данных.
5. Оценка качества сервиса, анализ предложенных решений и выводы.

2 Данные

Многие NLP модели требуют большого количества предварительно обработанных данных для обучения, в частности TF-IDF векторизатор. В данном случае такими данными является корпус русскоязычных новостей. В исследовательских работах авторы часто используют готовые данные, например, общедоступные размеченные датасеты, но если учитывать цель проекта, то без реализации своей системы, позволяющей получать новости с нескольких источников за определённый период времени, не обойтись.

С помощью собственной системы парсинга собран датасет, состоящий из нескольких сотен тысяч новостных статей с сайтов следующих СМИ: «Новая газета», «Газета.Ru», «Lenta.ru», «ТАСС», «ВЕДОМОСТИ», «Медуза», «РИА Новости» с метаданными (заголовок, текст, дата, тема). При обработке выяснилось, что у многих статей темы указаны некорректно (например, у «РИА Новостей» большая половина контента помечена тегом «происшествие», у «Новой газеты» все новости старше 2014 года — тегом «политика»). После чистки данных в датасете осталось 130 тыс. документов, имеющих 32 различных тега. Распределение тегов и источников показано на рис. 1.

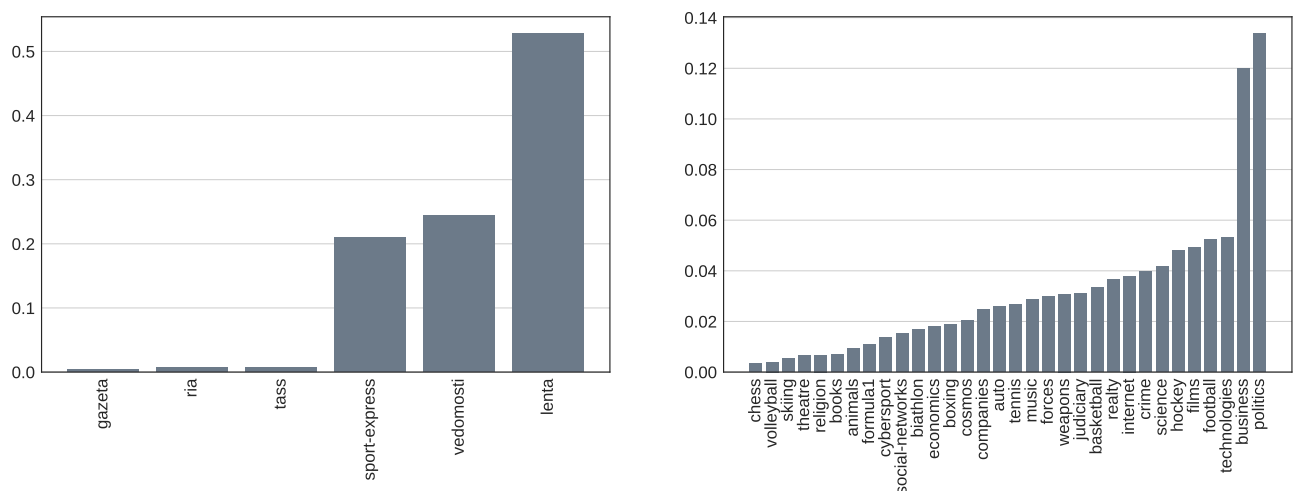


Рис. 1. Распределение тегов и источников в датасете статей для обучения TF-IDF векторизатора.

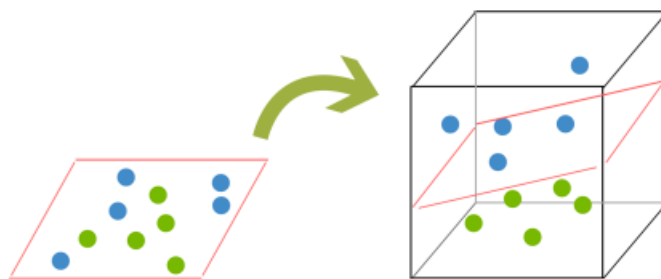


Рис. 2. Явное разделение данных на два класса при переходе в пространство более высокой размерности.

От объективности данного датасета зависит качество всего сервиса, так как на нём обучается векторизатор TF-IDF, на котором основан алгоритм кластеризации, а от него, в свою очередь, зависит суммаризация события. Для проверки валидности данных и обученного векторизатора реализован классификатор SVM (support vector machine).

SVM методы классификации используют операции линейной алгебры при работе с векторизованным текстом, при обучении «пытаясь» разделить многомерное пространство так, чтобы максимальное количество точек (векторов) одного и того же класса находилось в одной части пространства. Это можно достичь с помощью перехода к $n + 1$ -мерному пространству, как условно показано на рис. 2.

SVM классификаторы очень популярный инструмент в NLP задачах, поэтому существует множество реализаций с полезными функциями. Мы использовали его модификацию `SGDClassifier`, которая оптимизирует параметры с помощью градиентного спуска.

Проверка валидности состоит в эмпирической оценке признаков классов — это самые «весомые» слова, больше всего влияющие на принадлежность к конкретному классу. При изучении слов-признаков из таблицы 1 видно: слова-признаки и слова-классы связаны по смыслу и, в некоторых случаях, являются синонимами, что свидетельствует о корректности данных и векторизатора. Метрики качества классификаторы тоже подтверждают правильность датасета: Accuracy = 0,8687, F1 score = 0,8711, матрица ошибок представлена в приложении на рис. 11.

Таблица 1: Слова-признаки для собранного датасета.

animals	жить	вольер	хозяин	животный	зоопарк	питомец	кличка	животное
auto	авторынок	осаго	автомобильный	автопроизводитель	автопром	камаз	автоваз	автомобиль
basketball	рфб	евробаскет	центральной	кубок европа	баскетбол	баскетболист	евролига	нба
biathlon	эстафета	биатлонистка	шпиулин	сбр	хохфильцен	биатлон	ibu	биатлонист
books	библиотека	произведение	писательница	роман	книга	литературный	поэт	писатель
boxing	алоян	лебзяк	mma	поединок	поветкин	бой	бокс	боксер
business	россельхознадзор	fifa	ржд	газпром	туроператор	formula	ритейлер	оао
chess	карякин	шахматист	карякина	магнус	шахматы	карлсен	фид	шахматный
companies	тысяча автомобиль	компания	миллиард кубометр	миллиард	тысяча	процент акция	ретеилер	процент
cosmos	космонавт	светить	прогресс	вселенная	космос	астрофизик	марс	астронавт
crime	грабитель	изымать	группировка	полиция	убивать	летний	преступник	тюрьма
cybersport	gaming	team	valve	киберфутбол	dota	киберспорт	киберспортивный	киберспортсмен
economics	мрот	греция	бюджет	пенсия	ввп	минфин	экономика	инфляция
films	мультфильм	сериал	актриса	кино	картина	режиссер	актер	фильм
football	поле	стадион	нападающий	матч тур	фифа	уефа	футболист	полузащитник
forces	развертывать	выполнение	военнослужащий	военный	шойгу	генштаб	конашенков	минобороны
formula1	манор	цитировать	рено	феррари	макларен	пилот	мерседес	формула
hockey	авангард	ска	шайба	нападающий	хоккей	хоккеист	нхл	кхл
internet	википедия	сервис	ресурс	youtube	сайт	хакер	блогер	интернет
judiciary	стража	статья	арестовывать	колония	следственный комитет	следствие	скр	комитет россия
music	композитор	евровидение	песня	певец	концерт	альбом	певица	музыкант
politics	лидер	кремль	парламентарий	партия	депутат	госдума	глава	мид
realty	объект	строительный	жилищный	строительство	жжк	ипотека	жилье	недвижимость
religion	монастырь	собор	церковный	муфтий	святой	христиан	митрополит	патриарх
science	университет	журнал	математик	физик	научный	археолог	исследователь	ученый
skiing	вяльбе	нортуг	лыжник	fis	легков	йохауг	лахти	устюгов
social-networks	некоторые	юзер	facebook	twitter	пользователь сеть	пользователь	вконтакте	соцсеть
technologies	vimpelcom	apple	оператор	wifi	говорить	мтс	робот	контакт
tennis	кубок федерация	ореп	шарапов	теннис	корт	теннисист	теннисистка	кубок дэвис
theatre	росгосцирк	цирковой	балет	постановка	театральный	мюзикл	театр	спектакль
volleyball	факел	маричев	алекно	суперлига	казанский	белогорье	волейболист	волейбол
weapons	jane	использоваться	defense news	министерство оборона	миллиметр	миллиметровый	defense	тип

3 Анализ текста

3.1 Нормализация

Текст на естественном языке содержит много избыточных элементов, без которых его смысл не изменится. Чаще всего они действуют как «шум», так как встречаются равномерно по всему корпусу языка. Подобными элементами почти всегда выступают союзы, предлоги, части слов, отвечающие за форму. Кроме того, существуют разные способы написания одних и тех же объектов, например, числительные можно написать цифрами. При решении любой задачи в NLP текст нормализуют, то есть приводят в общую, более информативную форму, без «шума». Нормализация включает в себя несколько шагов.

При работе с естественной информацией её дискретизируют. Похожий процесс в NLP называется токенизация, он заключается в делении текста на части — токены, обычно токен является одним словом. К сожалению, просто делить текст по пробелам не совсем корректно, так как существует множество исключений, например, Великие Луки — это один токен, хотя и состоит из двух слов. Если рассматривать это как два токена, то смысл текста будет искажён, что может сказаться на результате и на качестве решения задачи. Для токенизации удобно использовать регулярные выражения.

В данном проекте используется два токенизатора: простое деление на слова при обработке текста для TF-IDF и Punkt токенизатор для деления статей на предложения при суммаризации. Последний использует корпус для обучения без учителя, «выучивая» последовательности, с которых начинаются предложения, что помогает работать с нелитературными данными, например, сообщениями в соц. сетях, где предложения часто начинаются с маленькой буквы.

Следующий шаг нормализации — удаление стоп-слов. Стоп-слова примерно одинаково распределены по всему корпусу языка. В русском языке ими являются многие служебные части (союзы, междометия, предлоги).

Для однозначной идентификации слова его приводят к начальной форме. Данный процесс называется лемматизация, а начальная форма — лемма. Для лемматизации недостаточно использовать только словарь, потому что существует огромное количество неологизмов, подчиняющимся тем же морфологическим правилам при образовании форм. Хорошие лемматизаторы проводят полный морфологический парсинг, при котором слова делятся на морфемы: стемы (самые осмысленные части) и аффиксы (придают дополнительное значение слову). Более простая версия морфологического анализа — стемминг, использующий определённые правила для извлечения основы слова.

Мы используем лемматизатор MyStem, предоставляющий универсальный для многих языков алгоритм морфологического разбора, использующийся в популярном поисковом движке [1].

При обработке данных перед векторизацией последовательно выполняются следующие операции:

1. приведение текста в нижний регистр;
2. удаление символов пунктуации;
3. удаление стоп-слов.
4. лемматизация каждого слова с помощью Python библиотеки MyStem.

Примеры предложений после перечисленных действий представлены в таблице 2.

3.2 Векторизация

TF-IDF — статистическая мера, используемая для оценки важности слова в любом контексте, основанная на его встречаемости в документе. Чем реже слово появляется в документе, тем выше его TF-IDF вес [2].

Таблица 2: Примеры нормализации текста

Оригинал	Нормализация
Однако когда их проверили на восприятие концепций и идей, оказалось, что те, кто писал от руки, понимают пройденный материал лучше однокашников.	однако когда они проверять на восприятие концепция идея оказываться что тот кто писать от рука понимать проходить материал хорошо однокашник
Лингвистическую относительность упоминали в своих сочинениях немецкие философы еще в конце XVIII – начале XIX века, но известность гипотеза получила именно благодаря Уорфу.	лингвистический относительность упоминать свой сочинение немецкий философ еще конец xviii – начало xix век но известность гипотеза получать именно благодаря уорфу

TF-IDF — это произведение двух статистик: TF (term frequency) и IDF (inverse document frequency).

Существует несколько способов подсчёта TF-IDF, в данной работе использовался следующий:

$$\text{tf}(t, d) = \frac{n_t}{\sum_k n_k},$$

где n_t есть число вхождений слова t в документ, а $\sum_k n_k$ — общее число слов в данном документе.

$$\text{idf}(t, D) = \log \frac{|D|}{|\{d_i \in D \mid t \in d_i\}|},$$

где $|D|$ — число документов в корпусе, $|\{d_i \in D \mid t \in d_i\}|$ — число документов из корпуса D , в которых встречается t (когда $n_t \neq 0$).

Таким образом, мера TF-IDF является произведением двух сомножителей:

$$\text{TF-IDF}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D).$$

Признаковым описанием одного объекта $d \in D$ будет вектор

$$(\text{TF-IDF}(t, d, D))_{t \in V},$$

где V — словарь всех слов, встречающихся в корпусе D .

Обучение модели заключается в подсчёте веса каждого уникального слова в каждом документе.

```

1  $compNum \leftarrow 0$ 
2 for  $(u,v) \in E$  do
3    $w(u,v) \leftarrow \cos(u,v)$ 
4 for  $v \in V$  do
5    $c[v] \leftarrow nil$ 
6 for  $v \in V$  do
7   if  $c[v] = nil$  then
8      $c[v] \leftarrow compNum$ 
9      $Q \leftarrow \{v\}$ 
10     $S \leftarrow \emptyset$ 
11    while  $Q \neq \emptyset$  do
12       $u \leftarrow pop(Q)$ 
13       $c[u] \leftarrow compNum$ 
14       $S \leftarrow S \cup \{v\}$ 
15      for  $n \notin S, (u,n) \in E$  do
16        if  $w(u,n) \geq threshold$  and  $c[n] = nil$  then
17           $Q \leftarrow Q \cup \{n\}$ 
18     $compNum \leftarrow compNum + 1$ 

```

Рис. 3. Псевдокод графовой кластеризации.

Одно из полезных свойств векторов TF-IDF: косинусное расстояние между векторами характеризует «похожесть» статей, что можно использовать для кластеризации.

3.3 Кластеризация

Самый базовый алгоритм кластеризации, который в данном случае применим — алгоритм поиска связных компонентов в графе. Его псевдокод показан на рис. 3, где V, E — множество вершин и рёбер графа, $(u,v), (u,n)$ — рёбра, v, u, n — вершины, $w(u,v)$ — вес ребра (u,v) , $c[v]$ — номер компоненты (кластера) вершины v , Q — очереди вершин для посещения, S множество посещённых вершин, $compNum$ — номер текущей компоненты.

Алгоритм работает следующим образом: строится полный граф, значения рёбер выставляются как косинусное расстояние между векторами-вершинами. Далее, от каждой не помеченной номером кластера вершины запускается BFS алгоритм (breadth-first search, поиск в ширину), игнорирующий рёбра меньше определённо-

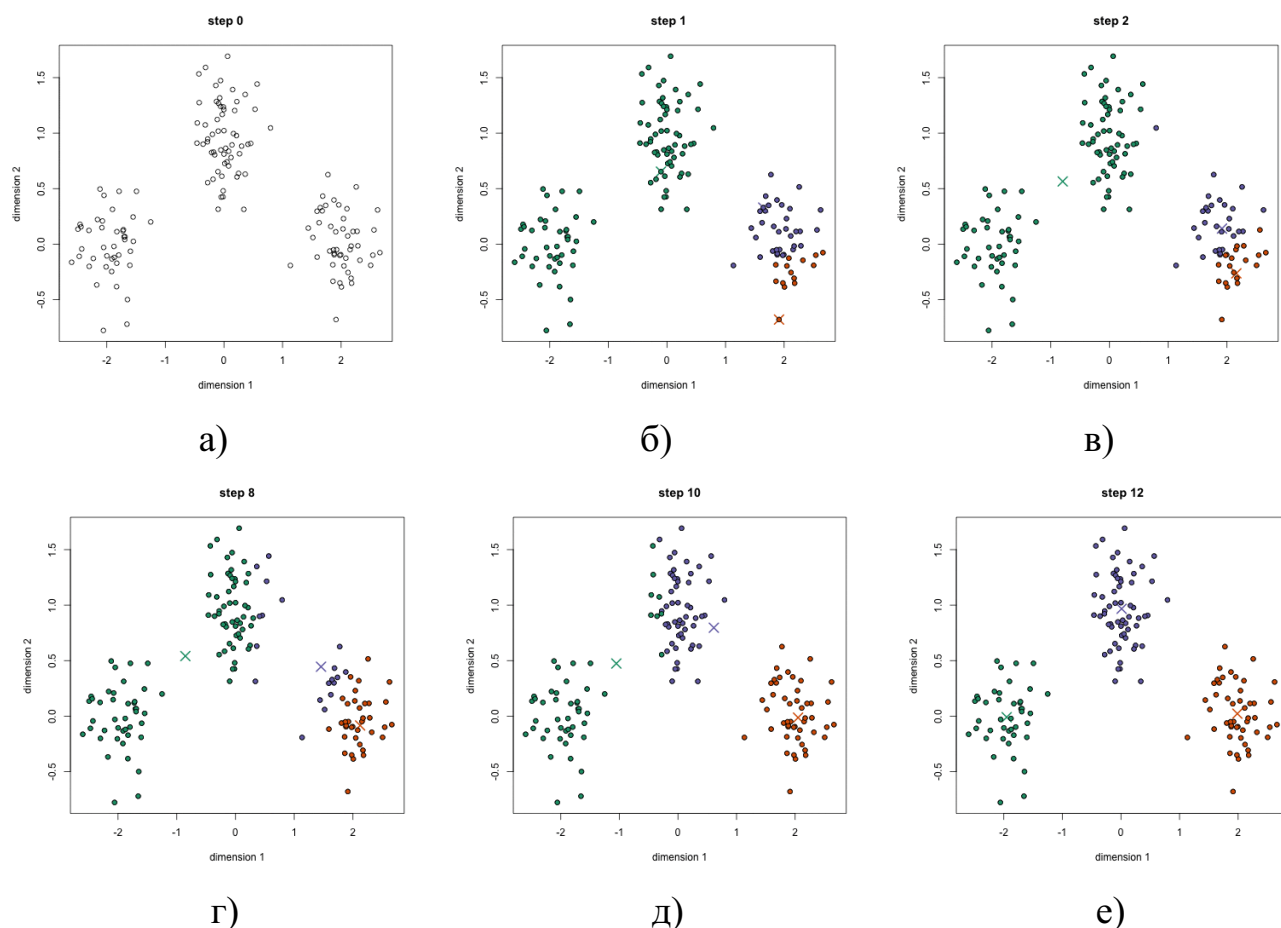


Рис. 4. Визуализация кластеризации K-means.

го значения, и устанавливающий номер своего кластера любой вершине, которой удалось достичь. Этот процесс повторяется до тех пор, пока всем вершинам не назначен номер кластера.

Данный алгоритм очень прост в реализации и показывает хорошие результаты, кроме того, ему не нужно заранее знать количество кластеров, единственный параметр *threshold* — порог, ниже которого рёбра игнорируются.

Другой, более популярный, но не менее простой алгоритм кластеризации K-means рассматривает векторы, как точки в пространстве, выбирает k случайных точек-центроидов и все ближайшие к ним точки-вектора статей, формируя k случайных групп (рис. 4, б). Далее у каждой группы определяется средняя точка, которая становится новым центроидом (рис. 4, в). Процесс повторяется до тех пор, пока точки не перестанут сдвигаться (рис. 4, г–е).

В проекте используется алгоритм K-means, так как при эмпирическом сравнении он показывает результаты чуть лучшие графового подхода и имеет больше параметров, что позволяет его тонко настроить в зависимости от размера данных.

Если данных для кластеризации очень много, как это часто бывает, то алгоритму может не хватить оперативной памяти. Для решения этой проблемы существует вариант K-means, принимающий данные по частям (batches), к сожалению, время работы алгоритма при этом увеличивается в зависимости от размера частей.

3.4 Суммаризация

Алгоритмы суммаризации можно условно поделить на две группы: абстрактная (abstractive) и извлекающая (extractive). Первая генерирует текст по смыслу, вторая выбирает самые информативные предложения из исходного текста и склеивает их. Модели абстрактной суммаризации требуют больших вычислительных мощностей и сложны в реализации, так как основаны на нейронных сетях, при их использовании стоит учитывать особенности языка и использовать эвристики для генерации корректных форм слов или словосочетаний. Извлекающая суммаризация легче в реализации и выглядит правдоподобно, так как состоит из написанных человеком предложений.

В сервисе используются два алгоритма MDS (multi-document summarization): SumBasic и DivRank, подразумевающие суммаризацию сразу нескольких документов на одну тему, что хорошо подходит для кластеров-событий.

3.4.1 SumBasic

SumBasic основан на простом наблюдении: слова, появляющиеся часто в кластере документов, с большей вероятностью окажутся в сокращённых текстах, написанных человеком, чем остальные.

Алгоритм SumBasic состоит из следующих шагов:

1. Для каждого уникального слова w_i посчитать вероятность его присутствия во входных данных $p(w_i) = \frac{n}{N}$, где n — количество вхождений слова w_i в данных, а N — общее количество слов.
2. Взвесить каждое предложение S_j , посчитав среднюю вероятность $p(w_i)$:

$$weight(S_j) = \sum_{w_i \in S_j} \frac{p(w_i)}{|\{w_i | w_i \in S_j\}|}$$

3. Выбрать предложение с лучшим весом, содержащие слово с максимальной вероятностью.

4. Для каждого слова w_i в выбранном предложении обновить их вероятность:

$$p_{new}(w_i) = p_{old}(w_i)^2$$

5. Если уже выбранных предложений недостаточно перейти на шаг 2.

Шаг 3 гарантирует, что предложение с самым вероятным словом будет выбрано. Шаг 4 добавляет алгоритму «чувствительность» к контексту сокращения: обновляя вероятности таким образом мы позволяем изначально невероятным словам оказывать большее влияние на выбор предложений, и, самое главное, этот шаг позволяет эффективно применять алгоритм для нескольких документов, позволяя игнорировать уже сокращённую информацию, реализуя «затухание» вероятности слов [3, 4].

SumBasic выбран из-за его простоты и элегантности для сравнения с более сложным алгоритмом DivRank.

3.4.2 DivRank

DivRank (Diverse Rank) — алгоритм для взвешивания графов, подобный популярному PageRank, но применяющийся для MDS суммаризации. Для его использования необходимо построить граф, схожий с тем, что описан в графовом методе кластеризации в п.3.3. Различия в том, что здесь вершины — это TF-IDF вектора предложений, а не статей. Рёбра удаляются, если косинусное расстояние меньше 0.1. После взвешивания этого графа DivRank'ом выбираются k первых самых весомых предложений.

На рис. 5, в показан «разнообразно» взвешенный граф с помощью DivRank в сравнении с графом, взвешенный используя PageRank (рис. 5, б). Если необходимо выбрать 3 вершины, максимально и ёмко передающие информацию о графе, то алгоритм DivRank выдаст вершины 1,4,5, что даже визуально лучше, чем ответ алгоритма 1,2,3 PageRank [5].

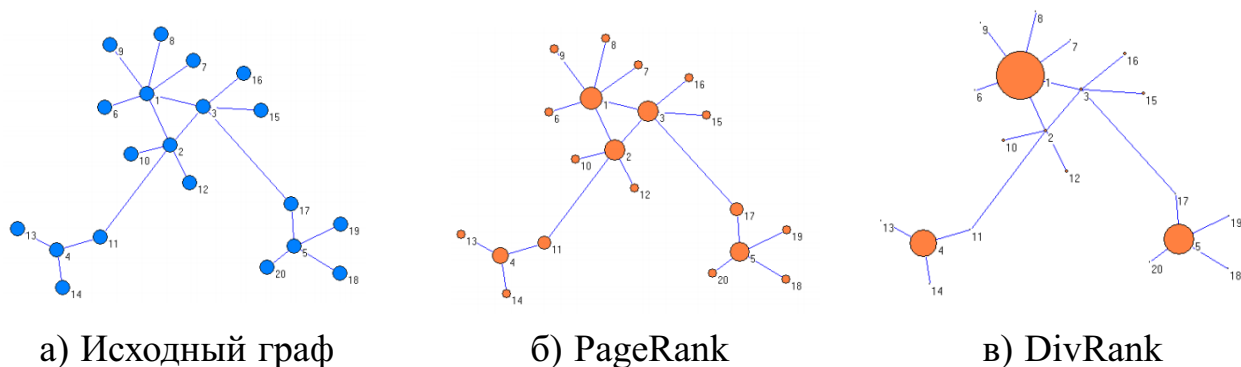


Рис. 5. Взвешивание графа

4 Реализация сервиса

4.1 Архитектура и транспорт данных

Сервис представляет из себя 4 обособленных docker-контейнера взаимодействующие между собой путём обмена json-объектами с помощью базы данных MongoDB.

Контейнеры по порядку развёртывания сервиса:

1. *mongo* — база данных. Запускается мгновенно.
2. *getter* — сервис парсинга. Запускается мгновенно, пишет в коллекцию *raw_news* базу данных новости за последние несколько часов при инициализации. Если при запуске в коллекции остались старые необработанные новости, они удаляются.
3. *analyser* — сервис анализа текста. Запускается через несколько десятков секунд после деплоя, ожидая данные от *getter*. Читает из коллекции *raw_news*, удаляя из неё каждую обработанную новость. Пишет в коллекцию *events*. Если при запуске в коллекции находятся старые данные, они не удаляются, чтобы сервер отдавал их, а не пустоту по время развёртывания.
4. *server* — выступает в роли интерфейса, отдаёт пользователю обработанные данные сервиса.

Развёртывание сервиса происходит с помощью *docker-compose*. Для быстрой установки и развёртывания написан скрипт (рис. 6).

Все контейнеры, кроме базы данных, могут функционировать независимо друг от друга, что очень полезно при разработке. Все приложения внутри реализованы на Python 3.

```

sudo apt install docker.io
sudo docker login -u ... -p ...
sudo pip install docker-compose
cd app/analyzer/nlp/models
URL="https://www.dropbox.com/sh/8s9qfy5rf1o5bbd/AAAVSVQZorVr6JH8LPJQX9tva?dl=1"
wget $URL -O models.zip && unzip models.zip -x /
cd app
sudo docker-compose build
sudo docker-compose up -d
sudo docker-compose logs -f

```

Рис. 6. Скрипт для запуска сервиса с установкой необходимых компонентов.

4.2 Особенности реализации компонентов анализа

4.2.1 Система сбора данных

Система сбора данных представляет собой коллекцию парсеров сайтов российских СМИ, основанные на одном подходе и имеющие одинаковый интерфейс.

Чтобы понять, как реализовать универсальную систему, с возможностью быстрого добавления поддержки нового ресурса, достаточно взглянуть на сайты русскоязычных медиа-ресурсов. Многие сильно отличаются друг от друга внешне, но у всех присутствует следующая логика: существуют страницы со списком новостей в хронологическом порядке, которые либо агрегированы по дням (Lenta.ru, Gazeta.ru, vedomosti.ru), либо используют параметр `offset`, указывающий с какой статьи начинать страницу (novayagazeta.ru, tass.ru, meduza.io). Первый вариант пагинации удобнее, потому что позволяет просто получить новости за любой заданный интервал времени. Во втором случае можно использовать бинарный поиск по страницам, но это не реализовано, так как новости всегда нужны с текущего момента. Большинство СМИ отдают данные в HTML формате и только лишь малая часть использует API в JSON формате.

При инициализации сервиса необходимо получить новости за последние несколько часов, чтобы сформировать актуальные кластера, что занимает долгое время. Парсинг множества статей можно ускорить в несколько раз, обрабатывая их параллельно.

Так как система обособлена, то реализовывать её можно на любом языке, а взаимодействовать с сервисом через внешнее хранилище, но для удобства разработки выбран Python 3, с использованием дополнительных библиотек:

- BeautifulSoup — парсинг HTML.

```

from parsers import Gazeta, Tass, Lenta, Vedomosti, Novaya, Meduza,
    process_init
import datetime
from pymongo import MongoClient
from multiprocessing import Pool, cpu_count
mongo_client = MongoClient('localhost', 27017)

pool = Pool(processes=cpu_count, initializer=process_init)
parsers_ = [
    Gazeta(), Tass(), Meduza(), Lenta(), Vedomosti(), Novaya()
]
until = datetime.datetime.now() - datetime.timedelta(hours=4)
for parser in parsers_:
    parser.parse(pool, until_time=until)
pool.close()
pool.join()
news = list(mongo_client.news.raw_news.find({}))

```

Рис. 7. Пример использования для получения всех новостей за последние 4 часа.

- multiprocessing — распараллеливание задач.
- pymongo — интерфейс над внешним хранилищем MongoDB.

Так как Питон имеет ограничение на потоки из-за GIL, то чтобы обеспечить параллелизм, позволяющий с увеличением количества процессорных ядер ускорять обработку множества статей, используется модуль multiprocessing, что значительно усложняет задачу. Использование корутин или асинхронных задач вместо процессов в данном случае не даст большой прирост производительности, так как большинство процессорного времени тратиться не на чтение сокетов, а на парсинг HTML.

Пример использования представлен на рис. 7.

Главной частью системы является класс BaseParser, инкапсулирующий сетевые запросы, работу по синхронизации процессов и передаче данных между ними. В целом, логика межпроцессного взаимодействия системы достаточно тривиальная и находится в методе parse (рис. 8): в метод передаётся объект pool, принимающий задачи парсинга статей, задачи создаются во время обработки страницы с лентой новостей сайта. Процессы пула пишут результаты в базу данных mongodb, откуда их потом извлекают другие контейнеры при необходимости. Полный код базового класса парсеров представлен в приложении.

Для создания нового парсера необходимо наследоваться от BaseParser, вызвать родительский конструктор с параметрами: название парсера, URL-префикс

```

. . .
url_to_fetch = self._page_url()
while True:
    content = self._get_content(url_to_fetch, type_=self.page_type)
    news_list = self._get_news_list(content)
    for news in news_list:
        try:
            # Url always first, timestamp always second in params
            news_params = self._get_news_params_in_page(news)
            self.curr_date = news_params[1]
            if (self.curr_date <= until_time):
                break
            # Pushing task to pool queue
            pool.map_async(self._process_news, [(news_params)])
        else:
            url_to_fetch = self._next_page_url()
            continue
    break
. . .

```

Рис. 8. Часть метода `BaseParser.parse` с основной логикой.

любой новости ресурса, URL-префикс ленты новостей, дефолтное количество процессов. Определить следующие методы:

- `_get_news_list(self, content)` — возвращает список списков с необработанными параметрами новости, используя контент (HTML или JSON) ленты новостей.
- `_get_news_params_in_page(self, news)` — возвращает tuple с параметрами статьи, используя элемент списка из предыдущего метода. URL и дата (в формате `timestamp`) должны быть первыми в списке параметров (`dict` не используется в данном случае в попытке выиграть время и место на pickle-запаковке объекта).
- `_parse_news(self, news_params)` — возвращает dict с новостью, например,


```
{'title': title, 'url': url, 'text': text, 'topic': topic, 'date': date}.
```
- `_page_url(self)` — возвращает URL текущей страницы.
- `_next_page_url(self)` — «переворачивает» страницу и возвращает `_page_url`.

Пример реализации одного из парсеров представлен в приложении.

```

{
    'kmeans': {
        'proximity_coeff': 0.6, 'n_clusters_coeff': 4,
        'batch_size': 50, 'n_init': 10, 'max_iter': 200
    },
    'append_titles': True,
    'svm_path': 'nlp/models/SVM_classifier.bin',
    'svm_labels_path': 'nlp/models/LabelEncoder.bin',
    'tfidf_path': 'nlp/models/TFIDF_vectorizer.bin',
    'max_news_distance_secs': 12*60*60,
    'drop_duplicates': True,
    'sumbasic': {
        'summary_length': 4
    },
    'divrank': {
        'summary_length': 4
    }
}

```

Рис. 9. Конфигурация моделей и алгоритмов анализа.

4.2.2 Модуль анализа текста

Весь процесс анализа инкапсулирован в классе `Analyzer`. В конструкторе принимает конфигурацию с параметрами моделей и алгоритмов. Конфигурация по умолчанию представлен на рис. 9.

Описание параметров:

- `kmeans.proximity_coeff` — минимальное косинусное расстояние от центра кластера, необходимое для попадания в него.
- `kmeans.n_clusters_coeff` — количество кластеров.
- `kmeans.batch_size` — количество одновременно обрабатываемых статей, при большом значении увеличивает нагрузку на память, при маленьком — на процессор.
- `kmeans.max_iter` — максимальное количество повторений алгоритма, влияет на время исполнения.
- `..._path` — пути к моделям.
- `append_titles` — добавлять заголовок к тексту новости.
- `max_news_distance_secs` — ограничение в секундах по старости новостей.
- `drop_duplicates` — искать и удалять дубликаты в данных.

```

def fit(self, news_list):
    if not news_list:
        return
    # Cleaning and classifying new data
    new_data = self._normalize(news_list)
    self._vectorize(new_data)
    self._classify(new_data)
    # Adding new data to existing data
    self._data = pd.concat([new_data, self._data])
    # Sorting just to be sure
    self._data.sort_values('date', inplace=True, ascending=False)
    if self.config['drop_duplicates']:
        self._data.drop_duplicates(subset='url', inplace=True)
    # Saving most recent news date for next update
    self._last_time = self._data.iloc[0].date
    self._first_time = self._data.iloc[-1].date
    # Dropping all data older then 24 hours
    self._data = self._data[self._data.date
        >= self._last_time - self.config['max_news_distance_secs']]
    self._count = self._data.shape[0]
    # Clusterize all data every time new data is coming
    clusters_no_sum = self._clusterize()
    self._clusters = self._summarize(clusters_no_sum)
    self._form_output()

```

Рис. 10. Метод Analyzer.fit.

- `sumbasic.summary_length`, `divrank.summary_length` — длина суммаризации в предложениях.

Основным методом анализатора является `fit` (рис. 10), принимающий список json-объектов с новостями и полностью их обрабатывающий. Полный код анализатора представлен в приложении.

5 Оценка решений

Несколько примеров выдачи системы представлены на рисунках 12, 13, 14 в приложении.

В первом можно увидеть идеальный вариант: кластеризатор смог корректно найти и объединить новости по данной теме с 4 различных источников, классификатор поставил правильный тег, суммаризаторы сгенерировали корректную информацию. Видны различия между суммаризаторами: у DivRank более «человечная» и связанная суммаризация, а SumBasic сразу начинает с самого информативного сообщения.

Во втором варианте более спорный результат: кластера относятся к одной теме, но бросаются глаза проблемы с токенизацией предложений (висящие точки вначале). Можно не согласиться и с тегом, предоставленным классификатором. SumBasic показал в данном случае себя лучше.

Последний вариант абсолютно некорректен. Такое происходит, когда у сервиса мало данных или все они относятся к разным событиям.

При наблюдении за работой сервиса стало ясно, что конфигурация моделей должна быть динамической и зависеть от количества данных.

Для чистоты эксперимента SumBasic и DivRank запускался на аннотации к данной работе с параметром на сокращения до одного предложения. Оба алгоритма вывели третье предложение аннотации: «Для построения такого сервиса мы изучаем и используем различные методы и модели, распространенные в NLP для решения следующих задач: нормализация, векторизация, кластеризация и суммаризация текста.»

6 Заключение

В данной работе удалось собрать корпус русскоязычных новостных статей, проверить его корректность; в разной степени изучить и применить на практике следующие модели и алгоритмы NLP: нормализацию, TF-IDF векторизацию, SVM классификацию, K-means кластеризацию, SumBasic и DivRank суммаризацию. Создан сервис, демонстрирующий работу данных алгоритмов в режиме реального времени. Качество работы сервиса непостоянно и напрямую зависит от количества данных. Алгоритм суммаризации SumBasic показал очень хорошие результаты для подобной простой модели.

Список литературы

1. *Segalovich Ilya*. A Fast Morphological Algorithm with Unknown Word Guessing Induced by a Dictionary for a Web Search Engine. — 2003. — 01. — Pp. 273–280.
2. *JONES KAREN SPARCK*. A STATISTICAL INTERPRETATION OF TERM SPECIFICITY AND ITS APPLICATION IN RETRIEVAL // *Journal of Documentation*. — 1972. — Vol. 28, no. 1. — Pp. 11–21. <https://doi.org/10.1108/eb026526>.
3. *Nenkova Ani, Vanderwende Lucy*. Tech. Rep.: : Microsoft Research, 2005.
4. Beyond SumBasic: Task-focused summarization with sentence simplification and lexical expansion / Lucy Vanderwende, Chris Brockett, Ani Nenkova et al. // *Information Processing and Management*. — 2007.
5. *Mei Qiaozhu, Guo Jian, Radev Dragomir*. DivRank: the Interplay of Prestige and Diversity in Information Networks. — 2010.

Приложение

	Предсказанный класс																															
	animals	auto	basketball	biathlon	books	boxing	business	chess	companies	cosmos	crime	cybersport	economics	films	football	forces	formula1	hockey	internet	judiciary	music	politics	reality	religion	science	skiing	social-networks	technologies	tennis	theatre	volleyball	weapons
animals	0.89	0	0	0	0	0	0.01	0	0	0	0.02	0	0	0.01	0	0	0	0	0.02	0	0.01	0.02	0	0	0.03	0	0	0	0	0	0	0
auto	0	0.87	0	0	0	0	0.05	0	0.02	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0.03	0	0	0	0	0.01	0	0	0	0
basketball	0	0	0.98	0	0	0	0	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
biathlon	0	0	0	0.99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
books	0	0	0	0	0.87	0	0	0	0	0	0	0	0	0.07	0	0	0	0	0	0.01	0	0.01	0	0	0.01	0	0	0	0	0.01	0	0
boxing	0	0	0	0	0	0.99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
business	0	0.02	0	0	0	0	0.85	0	0	0	0	0	0.01	0	0.01	0	0	0	0	0	0	0.01	0.02	0	0	0	0	0.04	0	0	0	0
chess	0	0	0	0	0	0	0	0.99	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0	0
companies	0	0.04	0	0	0	0	0.12	0	0.72	0	0	0	0	0	0	0	0	0	0.03	0.01	0	0.02	0.01	0	0	0	0	0.02	0	0	0	0.01
cosmos	0	0	0	0	0	0	0	0	0	0.91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.06	0	0	0.02	0	0	0	0
crime	0	0	0	0	0	0	0.01	0	0	0	0.82	0	0	0	0	0	0	0	0	0.08	0	0.07	0	0	0	0	0	0	0	0	0	0
cybersport	0	0	0	0	0	0	0	0	0	0	0	0.98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
economics	0	0.01	0	0	0	0	0.2	0	0	0	0	0	0.68	0	0	0	0	0	0	0	0	0.07	0.02	0	0	0	0	0.01	0	0	0	0
films	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0.96	0	0	0	0	0	0	0.01	0.01	0	0	0	0	0	0	0	0	0	0
football	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
forces	0	0	0	0	0	0	0.01	0	0	0.01	0.01	0	0	0	0	0.76	0	0	0	0.01	0	0.08	0	0	0	0	0	0.01	0	0	0	0.12
formula1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
hockey	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.99	0	0	0	0	0	0	0	0	0	0	0	0	0	0
internet	0.01	0	0	0	0	0	0.01	0	0.01	0	0.02	0	0	0.02	0	0	0	0	0.73	0	0.01	0.03	0	0	0.01	0	0.08	0.05	0	0	0	0
judiciary	0	0	0	0	0	0	0.01	0	0.01	0	0.07	0	0	0	0	0	0	0	0	0.85	0	0.03	0	0	0	0	0	0	0	0	0	0
music	0	0	0	0	0.01	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0.94	0.01	0	0	0	0	0	0	0	0.01	0	0
politics	0	0	0	0	0	0	0.01	0	0	0	0.02	0	0	0	0	0.01	0	0	0	0.01	0	0.92	0	0	0	0	0	0	0	0	0	0
reality	0	0.01	0	0	0	0	0.07	0	0	0	0	0	0.01	0	0	0	0	0	0.01	0	0.01	0.87	0.01	0	0	0	0	0	0	0	0	0
religion	0	0	0	0	0.01	0	0.01	0	0	0	0.01	0	0	0.02	0	0	0	0	0	0.01	0.07	0.01	0.96	0	0	0	0	0	0	0	0	0
science	0	0	0	0	0	0	0	0	0	0.06	0	0	0	0	0	0	0	0	0	0	0	0.01	0	0	0.9	0	0	0.01	0	0	0	0
skiing	0	0	0	0.03	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.96	0	0	0	0	0	0
social-networks	0.02	0	0	0	0	0	0	0	0	0	0.01	0	0	0.03	0	0	0	0	0.15	0	0.03	0.04	0	0	0	0	0.66	0.01	0	0	0	0
technologies	0	0.01	0	0	0	0	0.08	0	0	0.05	0	0	0	0.01	0	0.01	0	0	0.08	0	0	0.01	0.01	0	0.04	0	0.01	0.67	0	0	0	0.02
tennis	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.99	0	0	0
theatre	0.01	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0	0	0	0	0	0.02	0.01	0	0	0	0	0	0	0	0.85	0	0
volleyball	0	0	0.01	0.01	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.97	0
weapons	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.08	0	0	0	0	0	0.02	0	0	0	0	0	0.01	0	0	0	0.87

Рис. 11. Ошибки классификатора SVM.

Теракт в Париже совершил выходец из Чечни

Divrank Summary

Мужчина, напавший с ножом на прохожих в центре Парижа, оказался 20-летним выходцем из Чечни. В результате один человек погиб, несколько получили ранения. Полиция не нашла у нападавшего документов, однако смогла установить личность, взяв отпечатки пальцев. Инцидент произошел вечером 12 мая. Согласно сведениям Agence France-Presse, родители этого человека задержаны полицией и находятся под стражей.

Sumbasic Summary

В результате погиб 29-летний мужчина. Напавший на людей с ножом около Парижской Оперы был 20-летним выходцем из Чечни. Инцидент произошел вечером 12 мая. Кроме того, сообщалось, что группировка «Исламское государство» взяла на себя ответственность за нападение. Об этом в воскресенье сообщила радиостанция Europe 1 со ссылкой на источники в полиции.

Теракт в Париже совершил выходец из Чечни

NOVAYA Sun 10:13 [crime](#)

Следствие установило личность человека, который совершил вечером, 12 мая, теракт около Оперы Гарнье в Париже. Им оказался мужчина 1997 года рождения, выходец из Чечни. Об этом сообщает LeFigaro со ссылкой на директора кабинета префектуры полиции.

Парижский террорист оказался выходцем из Чечни

LENTA Sun 08:59 [crime](#)

Мужчина, напавший с ножом на прохожих в центре Парижа, оказался 20-летним выходцем из Чечни. При нем не было документов, но полиции удалось установить его личность по отпечаткам пальцев, сообщает в воскресенье, 13 мая, французская радиостанция Europe 1 со ссылкой на свои источники.

СМИ: нападение в Париже устроил выходец из Чечни

GAZETA Sun 08:22 [crime](#)

Напавший на людей с ножом около Парижской Оперы был 20-летним выходцем из Чечни. Об этом сообщает радиостанция Europe 1 со ссылкой на источники.

СМИ: напавшим на прохожих в Париже оказался выходец из Чечни

TASS Sun 06:17 [crime](#)

ПАРИЖ, 13 мая. /ТАСС/. Мужчина, совершивший в субботу вечером вооруженное нападение на прохожих в центре Парижа, оказался 20-летним выходцем из Чечни. Об этом в воскресенье сообщила радиостанция Europe 1 со ссылкой на источники в полиции.

Рис. 12. Пример функционирования сервиса 1.

Минкультуры поддержало решение о переводе Малобродского под подписку о невыезде

DivRank Summary

. Ранее официальный представитель Следственного комитета (СК) РФ Светлана Петренко сообщила, что следователем Главного управления по расследованию особо важных дел принято решение изменить меру пресечения обвиняемому Алексею Малобродскому с заключения под стражу на подписку о невыезде. В кардиореанимации его приковали наручниками к кровати. 14 мая уполномоченный по правам человека в России Татьяна Москалькова заявила о необходимости перевода Малобродского из СИЗО под домашний арест, а также выразила удивление позиций Генпрокуратуры, которая не поддерживает перевод Малобродского под домашний арест.

SumBasic Summary

Соответствующее постановление было объявлено обвиняемому, который в настоящее время находится в медицинском учреждении". Следствие изменило меру пресечения экс-гендиректору "Гоголь-центра" Алексею Малобродскому на подписку о невыезде. Об этом ТАСС сообщила официальный представитель Следственного комитета РФ Светлана Петренко. По словам Петренко, такое решение принято "с учетом возраста, состояния здоровья и иных обстоятельств". Кроме того, процесс сбора доказательств по уголовному делу завершен и, находясь на свободе, обвиняемый никак не повлияет на результаты расследования, по мнению следствия.

Минкультуры поддержало решение о переводе Малобродского под подписку о невыезде

TASS Mon 19:45 [business](#)

. Министерство культуры РФ поддерживает решение о переводе экс-гендиректора "Гоголь-центра" Алексея Малобродского из СИЗО под подписку о невыезде. Об этом ТАСС сказала в понедельник пресс-секретарь Владимира Мединского Анастасия Карпова, комментируя изменение меры пресечения Малобродскому.

Малобродского отпустили под подписку о невыезде

TASS Mon 18:51 [business](#)

. Следствие изменило меру пресечения экс-гендиректору "Гоголь-центра" Алексею Малобродскому на подписку о невыезде. Об этом ТАСС сообщила официальный представитель Следственного комитета РФ Светлана Петренко.

Москалькова приветствует перевод Малобродского под подписку о невыезде

TASS Mon 19:40 [business](#)

. Уполномоченный по правам человека в РФ Татьяна Москалькова приветствует изменение меры пресечения на подписку о невыезде арестованному бывшему директору "Гоголь-центра" Алексею Малобродскому.

СК отпустил Малобродского под подписку о невыезде

GAZETA Mon 18:43 [business](#)

Официальный представитель Следственного комитета (СК) России Светлана Петренко заявила об изменении меры пресечения Алексею Малобродскому, которого обвиняют в хищении бюджетных средств при реализации проекта «Платформа», сообщается на сайте СК.

СК решил отпустить Алексея Малобродского под подписку о невыезде: для этого не нужно решения суда

NOVAYA Mon 18:57 [business](#)

Алексей Малобродский. Фото: Светлана Виданова, «Новая газета»

СК освободил Малобродского под подписку о невыезде

MEDUZA Mon 18:33 [business](#)

Следственный комитет изменил меру пресечения бывшему директору «Гоголь-центра» Алексею Малобродскому с содержания под стражей на подписку о невыезде. Соответствующее решение принял следователь по делу «Седьмой студии».

Рис. 13. Пример функционирования сервиса 2.

Няня зарезала детей «по велению дьявола» и получила пожизненное

<div>DivRank Summary</div> <p>В США няню признали виновной в убийстве двух малолетних детей, оставленных ей на попечение. В своем последнем слове женщина заявила, что раскаивается в произошедшем и просит прощения у родителей и бога, а при вынесении приговора сидела неподвижно и с каменным лицом. Британский бренд спортивной одежды Sweaty Betty раскритиковали за «чрезмерную сексуализацию» подростков в новой рекламе укороченных топов и шорт, сообщает The Independent. Как насчет простых, функциональных и в то же время стильных костюмов для тренировок наших дочерей?» — запротестовали другие.</p>	<div>SumBasic Summary</div> <p>После преступления Ортега заявляла, что слышала голос дьявола, велевший ей «убить детей и себя». «Продавать таким способом спортивную одежду девочкам — неприемлемо. «Sweaty Betty — один из моих самых любимых спортивных брендов. Я бы не хотела видеть трех свои дочерей в таком виде.</p>
<div>Няня зарезала детей «по велению дьявола» и получила пожизненное</div> <div>LENTA Tue 18:15 business</div> <p>В США няню признали виновной в убийстве двух малолетних детей, оставленных ей на попечение. Как сообщает Mirror, 55-летняя Йоселин Ортега (Yoselyn Ortega) проведет остаток жизни в тюрьме.</p>	<div>Откровенные наряды детей возмутили родителей</div> <div>LENTA Tue 16:42 football</div> <p>Британский бренд спортивной одежды Sweaty Betty раскритиковали за «чрезмерную сексуализацию» подростков в новой рекламе укороченных топов и шорт, сообщает The Independent.</p>

Рис. 14. Пример функционирования сервиса 3.

```

import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
import pickle
import datetime
from .normalization import TEXT_PIPELINE
from . import sum_basic, divrank
from sklearn.cluster import KMeans, MiniBatchKMeans
import time
import numpy as np
import pytz
import os

TZ = pytz.timezone('Europe/Moscow')

class Analyzer():

    """ Class-wrapper for nlp data-processing """

    def __init__(self, config={
        'kmeans': {
            'proximity_coeff': 0.6, 'n_clusters_coeff': 4,
            'batch_size': 50, 'n_init': 10, 'max_iter': 200
        },
        'append_titles': True,
        'svm_path': 'nlp/models/SVM_classifier.bin',
        'svm_labels_path': 'nlp/models/LabelEncoder.bin',
        'tfidf_path': 'nlp/models/TFIDF_vectorizer.bin',
        'max_news_distance_secs': 12*60*60,
        'drop_duplicates': True,
        'sumbasic': {
            'summary_length': 4
        },
        'divrank': {
            'summary_length': 4
        }
    }):
        self.config = config
        self._data = pd.DataFrame([])
        self._last_time = 0
        self._first_time = 0
        self._clusters = []
        self._output = []
        self._count = 0
        self.SVM = None
        self.labels = None
        self.TFIDF = None
        self._load_models()

    def fit(self, news_list):
        if not news_list:
            return

        # Cleaning and classifying new data
        new_data = self._normalize(news_list)
        self._vectorize(new_data)
        self._classify(new_data)
        # Adding new data to existing data
        self._data = pd.concat([new_data, self._data])
        # Sorting just to be sure

```

```

self._data.sort_values('date', inplace=True, ascending=False)
if self.config['drop_duplicates']:
    self._data.drop_duplicates(subset='url', inplace=True)
# Saving most recent news date for next update
self._last_time = self._data.iloc[0].date
self._first_time = self._data.iloc[-1].date
# Dropping all data older then 24 hours
self._data = self._data[self._data.date >= self._last_time - self.config['max_news_distance_secs']]
self._count = self._data.shape[0]
# Clusterize all data every time new data is coming
clusters_no_sum = self._clusterize()
self._clusters = self._summirize(clusters_no_sum)
self._form_output()

def get_events(self):
    return self._output

def get_last_time(self):
    return self._form_date(self._last_time)

def get_count(self):
    return self._count

def _norimalize(self, news_list):
    # Converting data to pandas table
    data = pd.DataFrame(news_list)
    data = data[['media', 'url', 'title', 'text', 'topic', 'date']]
    data.date = data.date.apply(int)
    data.sort_values('date', inplace=True, ascending=False)
    # Append text_norm, title_norm columns to data w/ normalized text
    data.title = data.title.apply(lambda x: x.strip())
    data['text_norm'] = data.text.apply(TEXT_PIPELINE)
    data['title_norm'] = data.title.apply(TEXT_PIPELINE)
    return data

def _vectorize(self, data):
    if self.config['append_titles']:
        trainX = data['title_norm']
    else:
        trainX = data['title_norm'] + ' ' + data['text_norm']
    trainX = trainX.values
    data['tfidf_vector'] = list(self.TFIDF.transform(trainX).toarray())

def _classify(self, data):
    data['svm_class'] = list(self.SVM.predict(data['tfidf_vector'].tolist()))
    data['svm_class'] = data.apply(
        lambda row:
            self._class_to_str(row['svm_class']),
        axis=1)

def _clusterize(self):
    config = self.config['kmeans']
    tfidf_matrix = self._data['tfidf_vector'].tolist()
    kmeans = MiniBatchKMeans(
        n_clusters=int(self._count // config['n_clusters_coeff']),
        batch_size=int(config['batch_size']),
        n_init=int(config['n_init']),
        max_iter=int(config['max_iter'])
    ).fit(tfidf_matrix)

```



```

clusters_raw = kmeans.predict(tfidf_matrix)
clusters = [[] for _ in range(len(clusters_raw))]
for i, cluster in enumerate(clusters_raw):
    tfidf_news = np.array(tfidf_matrix[i]).reshape(1, -1)
    if cosine_similarity(tfidf_news,
        kmeans.cluster_centers_[cluster].reshape(1, -1))[0][0] >= self.config['kmeans']['proximity_coeff']:
        clusters[cluster].append(i)
return self._sort_clusters(clusters)

def _summirize(self, clusters):
    clusters_summed = []
    for news_ids in clusters:
        text = '\n'.join([self._data.iloc[id]['text'] for id in news_ids])
        try:
            divr = divrank(text, self.config['divrank'])
        except Exception as e:
            divr = ""
            print('divrank error')
        clusters_summed.append({
            'sumbasic': sum_basic(text, self.config['sumbasic']), 'divrank': divr,
            'content': news_ids
        })
    return clusters_summed

def _load_models(self):
    with open(self.config['svm_path'], 'rb') as pickle_file:
        self.SVM = pickle.load(pickle_file)
    with open(self.config['svm_labels_path'], 'rb') as pickle_file:
        self.labels = pickle.load(pickle_file)
    with open(self.config['tfidf_path'], 'rb') as pickle_file:
        self.TFIDF = pickle.load(pickle_file)

def _form_output(self):
    """ Creating a json output for server """
    self._output = []
    for i, cluster in enumerate(self._clusters):
        form_cluster = cluster.copy()
        form_cluster['content'] = []
        self._output.append(form_cluster)
        for id in cluster['content']:
            self._output[i]['content'].append({
                'media': self._data.iloc[id].media,
                'title': self._data.iloc[id].title,
                'url': self._data.iloc[id].url,
                'text': self._cut_text(self._data.iloc[id].text),
                'labels': {
                    'SVM': self._data.iloc[id]['svm_class']
                },
                'date': self._date_to_str(self._data.iloc[id].date),
            })

def _form_date(self, ts):
    return datetime.datetime.utcfromtimestamp(ts).replace(
        tzinfo=pytz.utc).astimezone(TZ)

def _get_avg_time(self, group):
    avg = 0
    for n in group:
        avg += self._data.iloc[n]['date']

```

```

        return avg // len(group)

def _sort_clusters(self, clusters):
    cs_filtered = filter(lambda x: len(x) > 1, clusters)
    # Sort by date of event
    cs_sorted = sorted(cs_filtered, key=lambda x: self._get_avg_time(x), reverse=True)
    # Sort news in cluster by time
    cs_sorted = list(map(lambda x: sorted(x,
                                          key=lambda y: self._data.iloc[y].date, reverse=True), cs_sorted))

    return cs_sorted

def _class_to_str(self, class_num):
    """ Converting number of predicted class to string """
    return self.labels.inverse_transform(class_num)

def _cut_text(self, text):
    """ Returns first paragraph of text """
    ps = []
    for p in text.split('\n'):
        if p != '':
            if len(p) > 500:
                ps.append(p[:500].strip() + '...')
                return "\n".join(ps)
            else:
                ps.append(p.strip())
    if len(ps) == 1:
        return "\n".join(ps)

    return ""

def _date_to_str(self, date):
    return datetime.datetime.utcfromtimestamp(date).replace(
        tzinfo=pytz.utc).astimezone(pytz.timezone('Europe/Moscow')).strftime('%a %H:%M')

```

Рис. 15. Класс Analyzer.

```

import requests
import re
import json
import datetime
from bs4 import BeautifulSoup
import logging
import time
import random
import os
import traceback
import pytz
from pymongo import MongoClient

logger = logging.getLogger(__name__)
if not len(logger.handlers):
    logger.setLevel(logging.INFO)
    console = logging.StreamHandler()
    formatter = logging.Formatter(
        '%(asctime)s - %(message)s', '%Y-%m-%d %H:%M:%S')
    console.setFormatter(formatter)
    console.setLevel(logging.INFO)
    logger.addHandler(console)

```

```

COLLECTION = None
def process_init():
    global COLLECTION
    db_client = MongoClient(
        'mongo',
        27017)
    db = db_client.news
    COLLECTION = db.raw_news

class BaseParser():

    """docstring for BaseParser"""

    def __init__(self, id, root_url, api_url, page_type='html'):
        self.id = id
        self.root_url = root_url # url for news
        self.api_url = api_url # url for pages
        self.page_type = page_type
        self.curr_date = None
        self.TZ = pytz.timezone('Europe/Moscow')

    def parse(self, pool,
              start_time=datetime.datetime.now(), until_time=None,
              news_count=None, topic_filter=None):
        """ Url extraction from pages in parant process """
        t_start = time.time()
        self._check_args(start_time, until_time, news_count,
                        topic_filter)
        # Some parsers do not have start time, so need to check
        start_time = start_time.timestamp()
        if until_time:
            until_time = until_time.timestamp()
        self.curr_date = start_time
        url_counter = 0
        url_to_fetch = self._page_url()
        while True:
            try:
                content = self._get_content(url_to_fetch, type_=self.page_type)
                news_list = self._get_news_list(content)
                if not news_list:
                    raise Exception('No content')
            except Exception as e:
                logger.error(
                    'Error: couldn\'t find content {} {}'.format(url_to_fetch, e))
                break

            logger.info('Look at page {}'.format(url_to_fetch))
            for news in news_list:
                try:
                    # Url always first, date always second in params
                    news_params = self._get_news_params_in_page(news)
                    if not news_params:
                        continue
                    url = news_params[0]
                    self.curr_date = news_params[1]
                    if (self.curr_date <= until_time):
                        break

```

```

        logger.debug('push to queue ' + str(news_params))
        pool.map_async(self._process_news, [(news_params)])
        url_counter += 1
        if url_counter % 10000 == 0:
            logger.warning(
                '{} {} news put to queue'.format(self.id, url_counter))
        except Exception as e:
            logger.error(
                'Error on url {}: {}'.format(url_to_fetch, traceback.format_exc()))
        else:
            url_to_fetch = self._next_page_url()
            continue
        break

logger.info('End of parsing, time: {}'.format(
    time.strftime('%H:%M:%S', time.gmtime(time.time() - t_start))))

def _process_news(self, news_params):
    try:
        logger.debug('pulled ' + str(news_params))
        news_out = self._parse_news(news_params)
        logger.debug('processed ' + str(news_out))
        if not news_out:
            return
        news_out['media'] = self.id
        COLLECTION.insert_one(news_out)
        logger.info('Pushed to db ' + news_out['url'])
    except Exception as err:
        logger.error("Error {} on {}".format(
            traceback.format_exc(), news_params[0]))

def _request(self, url):
    response = requests.get(url)
    response.raise_for_status()
    return response

def _get_content(self, url, type_='html'):
    response = self._request(url)
    if type_ == 'html':
        return BeautifulSoup(response.text.encode('utf-8'), 'lxml')
    elif type_ == 'json':
        return response.json()
    else:
        raise Exception()

def _check_args(self, start_time, until_time,
                news_count, topic_filter):
    pass

```

Рис. 16. Класс BaseParser.

```

import requests
import datetime
import re
from bs4 import BeautifulSoup
from .BaseParser import BaseParser
import time
import pytz

```

```

class Lenta(BaseParser):

    def __init__(self, **kwargs):
        super(Lenta, self).__init__(
            id='LENTA', root_url='https://lenta.ru',
            api_url='https://lenta.ru/news',
            page_type='html', **kwargs)

    def _get_news_list(self, content):
        """ Getting list of news from page content """
        return reversed(list(content.find_all(
            'div', 'item news b-tabloid__topic_news'))))

    def _get_news_params_in_page(self, news):
        news_url = self.root_url + news.find('a')['href']
        news_date = self._str_to_time(
            self._time_to_str(self.curr_date) + ' '
            + news.find('span', 'time').text)
        return news_url, news_date

    def _page_url(self):
        # Example: https://lenta.ru/news/2017/02/01/
        return self.api_url + self._time_to_str(self.curr_date)

    def _next_page_url(self):
        self.curr_date -= int(datetime.timedelta(days=1).total_seconds())
        return self._page_url()

    def _parse_news(self, news_params):
        """ Getting full news params by direct url """
        html = self._get_content(news_params[0])
        date = news_params[1]
        title = html.find('h1', 'b-topic__title').get_text()
        paragraphs = html.find('div', attrs={"itemprop": "articleBody"})
        paragraphs = paragraphs.find_all('p')
        text = '\n'.join([p.get_text() for p in paragraphs])
        try:
            topic = html.find('a', 'b-header-inner__block')
            topic = re.match(
                r'\\/rubrics\\/([A-z0-9]+)\\/', topic['href']).group(1)
        except Exception:
            topic = None
        try:
            tag = html.find('a', 'item dark active')
            tag = re.match(
                r'\\/rubrics\\/([A-z0-9]+\\/((([A-z0-9]+)\\/)?', tag['href']).group(2)
        except Exception:
            tag = None

        news_out = {'title': title, 'url': news_params[0], 'text': text,
                    'topic': topic, 'date': date, 'other': {'tag': tag}}
        return news_out

    def _time_to_str(self, time_):
        return datetime.datetime.utcfromtimestamp(time_).replace(
            tzinfo=pytz.utc).astimezone(self.TZ).strftime('%Y/%m/%d/')

```

```
def _str_to_time(self, time_str):
    return datetime.datetime.strptime(time_str, '%Y/%m/%d/ %H:%M').replace(tzinfo=self.TZ).timestamp()
```

Рис. 17. Пример реализации парсера Lenta.

```
from nltk.corpus import stopwords
from stop_words import get_stop_words
from functools import lru_cache
from pymystem3 import Mystem

en_sw = get_stop_words('en')
ru_sw = get_stop_words('ru')
STOP_WORDS = set(en_sw) | set(ru_sw)
STOP_WORDS = STOP_WORDS | set(
    stopwords.words('russian')) | set(stopwords.words('english'))
STOP_WORDS = STOP_WORDS | set(['лента', 'новость', 'риа', 'тасс',
                                'редакция', 'газета', 'хорра', 'daily',
                                'village', 'интерфакс', 'reuters'])

stemmer = Mystem()

class Pipeline(object):

    def __init__(self, *args):
        self.transformations = args

    def __call__(self, x):
        res = x
        for f in self.transformations:
            res = f(res)
        return res

def get_lower(text):
    return str(text).lower().strip()

def remove_punctuation(text):
    return ''.join([c if c.isalpha() or c in ['-', '"'] else ' ' for c in text])

@lru_cache(maxsize=None)
def get_word_normal_form(word):
    return ''.join(stemmer.lemmatize(word)).strip().replace('ё', 'е').strip('-')

def lemmatize_words(text):
    res = []
    for word in text.split():
        norm_form = get_word_normal_form(word)
        if len(norm_form) > 2 and norm_form not in STOP_WORDS:
            res.append(norm_form)
    return ' '.join(res)

TEXT_PIPELINE = Pipeline(get_lower, remove_punctuation, lemmatize_words)
```

Рис. 18. Функции нормализации normalization.

```

from flask import Flask, render_template, request, \
    redirect, url_for, jsonify
import datetime
import time
from threading import Thread
import os
from pymongo import MongoClient
import logging

logger = logging.getLogger(__name__)
if not len(logger.handlers):
    logger.setLevel(logging.INFO)
    console = logging.StreamHandler()
    formatter = logging.Formatter(
        '%(asctime)s - %(message)s', '%Y-%m-%d %H:%M:%S')
    console.setFormatter(formatter)
    console.setLevel(logging.INFO)
    logger.addHandler(console)

app = Flask(__name__, template_folder='./frontend/templates',
            static_folder='./frontend/static',
            static_url_path='/static')
app.config.from_object(__name__)

app.config.update(dict(
    BOOTSTRAP_WAIT=int(os.environ.get('SERVER_BOOTSTRAP_WAIT', 4*60)),
    UPDATE_RATE=int(os.environ.get('SERVER_UPDATE_RATE', 5*60)),
    OFFSET=int(os.environ.get('OFFSET', 5)),
    PORT=int(os.environ.get('PORT', 5000)),
))

db_client = MongoClient(
    'mongo',
    27017)
db = db_client.news

@app.route('/')
def index():
    return redirect(
        url_for('get_content', topic_count=5))

@app.route('/<int:topic_count>')
def get_content(topic_count=5):
    events = list(db.events.find({}))
    if len(events) < topic_count:
        return redirect(
            url_for('get_content', topic_count=len(events)))
    events_sliced = events[:topic_count]
    count = sum(len(x['content']) for x in events)
    return render_template('main.html', groups_count=len(events),
                           count=count, groups=events_sliced)

@app.route('/api/<int:offset>')
def api_get_content(offset=app.config['OFFSET']):
    events = list(db.events.find({}))
    if offset >= len(events):
        response = jsonify(message="Topic number too large")

```



```

        response.status_code = 404
        return response
    if len(events) < offset + app.config['OFFSET']:
        events = events[offset:len(events)]
    events = events[offset:offset + app.config['OFFSET']]
    return jsonify({'data': render_template('view.html', groups=events)})

@app.template_filter('min')
def reverse_filter(s):
    return min(s)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=app.config['PORT'], threaded=True, use_reloader=False)

```

Рис. 19. Реализация сервера