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.

Аннотация

Данная работа описывает процесс реализации сервиса, способного значительно улучшить и упростить пользовательское взаимодействие с новостным контентом, используя обработку 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 событии. 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, относящемся к одному событию. Мы собираем и обрабатываем большое количество данных с web-сайтов медиа и обучаем векторизатор TF-IDF, позволяющий использовать K-means для кластеризации. 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 web-sites 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 4

2 Данные 5

3 Анализ текста 7

3.1 Нормализация 7

3.2 Векторизация 8

3.3 Кластеризация 10

3.4 Суммаризация 12

3.4.1 SumBasic 12

3.4.2 DivRank 13

4 Реализация сервиса 13

4.1 Архитектура и транспорт данных 13

4.2 Особенности реализации компонентов анализа 13

4.2.1 Система сбора данных 13

4.2.2 Модуль нормализации 17

4.2.3 TF-IDF и SVM 17

4.2.4 K-means 17

4.2.5 SumBasic 17

4.2.6 DivRank 17

4.3 Инфраструктура и развёртка 17

5 Оценка решений 17

6 Заключение 17

Список литературы 17

1 Введение

Когда в мире происходит какое-либо событие, различные средства массовой информации пишут статьи с информацией об этом событии в виде новостей. Пользователю часто бывает сложно ориентироваться в большом потоке данных от разных источников. Автоматическая систематизация и обработка таких данных с целью предоставить 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 заметно 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.

Создание математических моделей, описывающих связи между информацией и естественным языком, позволяет автоматизировать эти процессы. Последние десятилетия быстрыми темпами развивается обработка естественного языка (Natural Language Processing, NLP), которая с помощью моделей и алгоритмов решает задачи автоматического анализа текста, в частности, при использовании нескольких источников объединять новостные статьи в группы (кластеры) по релевантности к конкретному событию (кластеризация), извлекать из кластера наиболее информативные данные 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: извлечения статей (парсинг) из web-сайтов СМИ для получения новостей вместе с их метаданными.
2. Изучение алгоритмов и моделей анализа текста: нормализация, векторизация и кластеризация, суммаризация.

3. Изучение технической реализация 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 используемый нами TF-IDF для векторизации текста. В данном случае такими данными является корпус русскоязычных новостей. В исследовательских работах авторы часто используют готовые данные —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 редакторами некорректно (например, у «РИА Новостей» большая половина контента помечена тегом «проишествие», у «Новой газеты» все новости старее 2014 года — тегом «политика»). После чистки данных в датасете осталось 130 тыс. статей, имеющих 32 различных тега. Распределение тегов и источников показано на рис. 1.

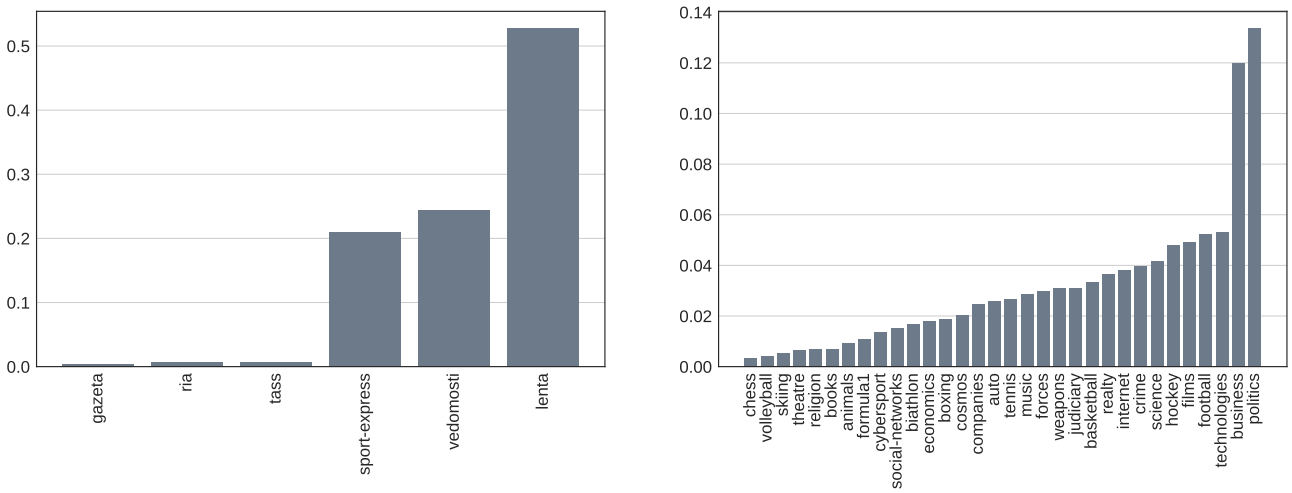


Рис. 1. Распределение тегов и источников в датасете статей для обучения TF-IDF векторизатора.

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

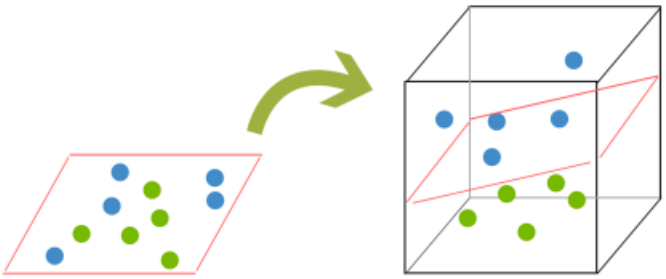


Рис. 2. Явное разделение данных на два класса при переходе в пространство более высокой размерности.

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 поддтверждают правильность датасета: Accuracy = 0,8687, F1 score = 0,8711, матрица ошибок представлена в приложении на рис. 8.

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	центровой	кубок европа	баскетбол	баскетболист	евролига	нба
biathlon	эстафета	биатлонистка	шипулин	сбр	хохфильцен	биатлон	ibu	биатлонист
books	библиотека	произведение	писательница	роман	книга	литературный	поэт	писатель
boxing	алоян	лебзяк	мма	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	ретейлер	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	жкх	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	open	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 встречается 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 дискретизуют. 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 обучение 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 лемматайзеры проводят полный морфологический парсинг, при котором слова делятся на морфемы: стемы (самые осмысленные части) и афиксы (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.

При обработке данных перед векторизаций 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 данных действий представлен 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.

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:

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

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

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

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:

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

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

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

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

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

Одно из полезных свойств векторов TF-IDF: косинусное расстояние между векторами характеризует «похожесть» статьей, 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 рёбер и вершин 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.

```
1  compNum ← 0
2  for (u,v) ∈ E do
3      w(u,v) ← cos (u,v)
4  for v ∈ V do
5      c[v] ← nil
6  for v ∈ V do
7      if c[v] = nil then
8          c[v] ← compNum
9          Q ← {v}
10         S ← ∅
11         while Q ≠ ∅ do
12             u ← pop(Q)
13             c[u] ← compNum
14             S ← S ∪ {v}
15             for n ∉ S, (u,n) ∈ E do
16                 if w(u,n) ⩾ threshold and c[n] = nil then
17                     Q ← Q ∪ {n}
18         compNum ← compNum + 1
```

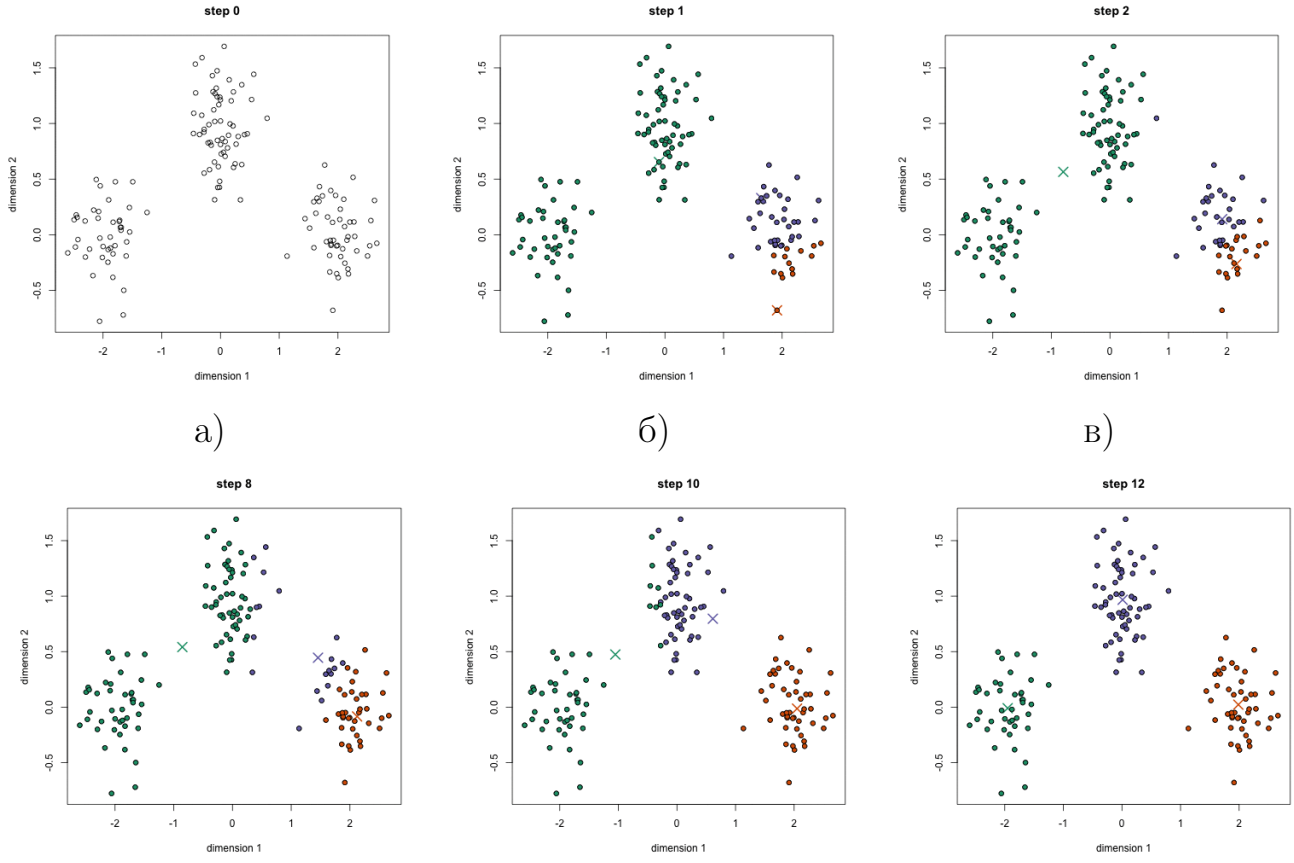
Рис. 3. Псевдокод графовой кластеризации

Алгоритм работает следующим образом: строится полный граф, значения ребёр выставляется 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 опреде-

лённого значения, и устанавливающий любой вершине, которой удалось достичь, номер своего кластера. 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).



а) б) в)

г) д) е)

Рис. 4. Визуализация кластеризации K-means.

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.

3.4.2 DivRank

4 Реализация сервиса

4.1 Архитектура и транспорт данных

4.2 Особенности реализации компонентов анализа

4.2.1 Система сбора данных

ПЕРЕПИСАТЬ, ВСЁ УЖЕ НЕ ТАК Система сбора данных представляет собой коллекцию парсеров сайтов российских СМИ, основанные на одном подходе и имеющие одинаковый интерфейс. Пример использования представлен на рис. 5.

В общей архитектуре сервиса система является обособленным фоновым процессом, собирающим раз в некоторое время новые статьи, поэтому нет необходимости постоянно поддерживать строгие ограничения к скорости парсинга. Но эту же систему можно использовать для относительно быстрого сбора собственного датасета. Кроме того, при инициализации сервиса понадобится получить новости за последние несколько часов, чтобы сформировать актуальные кластера. Парсинг множества статей можно ускорить в несколько раз, обрабатывая их параллельно.

```
from parsers import Gazeta, Tass, Lenta, Vedomosti, Novaya
import datetime

parsers = [
Gazeta(procs=4), # Number of processes used
Tass(),
Lenta(),
Vedomosti(),
Novaya(procs=4)
]
until_time = datetime.datetime.now() - datetime.timedelta(hours=4)
for parser in parsers:
print(parser.id)
for n in parser.get_news(until_time=until_time):
print(n['title'])
```

Рис. 5. Пример использования для получения всех новостей за последние 4 часа.

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.

Так как система обособлена, то писать её можно на любом языке, а взаимодействовать с сервисом через внешнее хранилище, но для удобства выбран `Python 3`, с использованием дополнительных библиотек: `BeautifulSoup` для парсинга HTML и `requests` для выполнения HTTP запросов. 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`.

Главной частью системы является класс `BaseParser`, инкапсулирующий сетевые запросы, работу по синхронизации процессов и передаче данных между

```
def get_news(self, start_time=None, until_time=None,
news_count=None, topic_filter=None):
. . .

Q_urls = Queue(0) # News urls and for deeper parsing
Q_out = Queue(0) # Results of parsing
sync_flag = Value('i', 1) # Flag to stop processes

workers = []
# Getting news by url in proceses
for _ in range(procs):
workers.append(Process(target=self._process_news,
args=(Q_urls, Q_out, sync_flag, topic_filter)))
workers[-1].start()
# Parsing pages with urls ("Лента новостей") and putting them to Q_urls
self.parse_pages(Q_urls, sync_flag, start_time,
until_time, news_count, topic_filter)
# Clearing output queue while processes still working
# Probably significantly slowing down other workers, need to fix it
out = []
self._listen_queue(workers, Q_out, out)
. . .
```

Рис. 6. Часть кода класса `BaseParser`.

ними. В целом, логика межпроцессного взаимодействия системы достаточно тривиальная и находится в методе `get_news` (рис. 6): процесс-предок запускает процессы-потомки и начинает заполнять очередь задач ссылками на статьи, в этот момент дочерние процессы уже разбирают ссылки из очереди и обрабатывают их. В качестве очереди задач выступает класс `multiprocessing.Queue`, который комбинирует межпроцессное взаимодействие через pipe и разделяемые блокировки. Благодаря модулю pickle очередь может передавать сложные объекты и часто применяется в подобных случаях.

После того, как основной процесс закончил парсинг страниц и отправил все задачи в очередь, он меняет значение переменной `sync_flag`, которая находится в общем для процессов сегменте памяти (Shared memory), если значение меняется, процессы-потомки понимают, что после опустошения очереди можно больше её не «слушать».

Результаты выполненных задач не принято передавать родительскому процессу, но в данном случае это было сделано для удобства интерфейса с помощью второй очереди. После смены значения общего флага предок начинает «слушать» очередь и ждать результатов. Такой подход крайне нежелателен, так как резуль-

тат в разы больше параметров самой задачи (из-за текста статьи), и при большом количестве обработанных новостей все процессы-потомки останутся висеть в простое, пока предок будет извлекать из очереди результаты. На небольших объёмах это незаметно, но запускать такое на 64 ядерном процессоре в 64 процесса с целью выкачать новости за несколько лет не стоит.

Описанная проблема будет решена заменой Pipe-очереди на базу данных в оперативной памяти (in-memory database), например, на Redis. Можно решить её и с использованием Shared memory, но это гораздо сложнее, так как объекты Питона придётся сериализовывать для хранения и десериализовывать для использования вручную.

Чтобы создать новый парсер необходимо наследоваться от `BaseParser`, вызвать родительский конструктор с параметрами: название парсера, URL-префикс 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 `datetime` объекте) 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`.

```
from nltk.corpus import stopwords
from pymystem3 import Mystem; mystem = Mystem()
import string
from stop_words import get_stop_words

STOP_WORDS = (set(stopwords.words('russian')) | set(stopwords.words('english')) |
set(get_stop_words('ru')) | set(get_stop_words('en')))

def get_word_normal_form(word):
return ''.join(mystem.lemmatize(word)).strip().replace('ё', 'е').strip('-')

def lemmatize_words(text):
text = text.lower()
text = ''.join([i for i in text if ( i not in string.punctuation )])
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)
```

Рис. 7. Простая функция нормализации.

4.2.2 Модуль нормализации

4.2.3 TF-IDF и SVM

4.2.4 K-means

4.2.5 SumBasic

4.2.6 DivRank

4.3 Инфраструктура и развёртка

5 Оценка решений

6 Заключение

Список литературы

Приложение

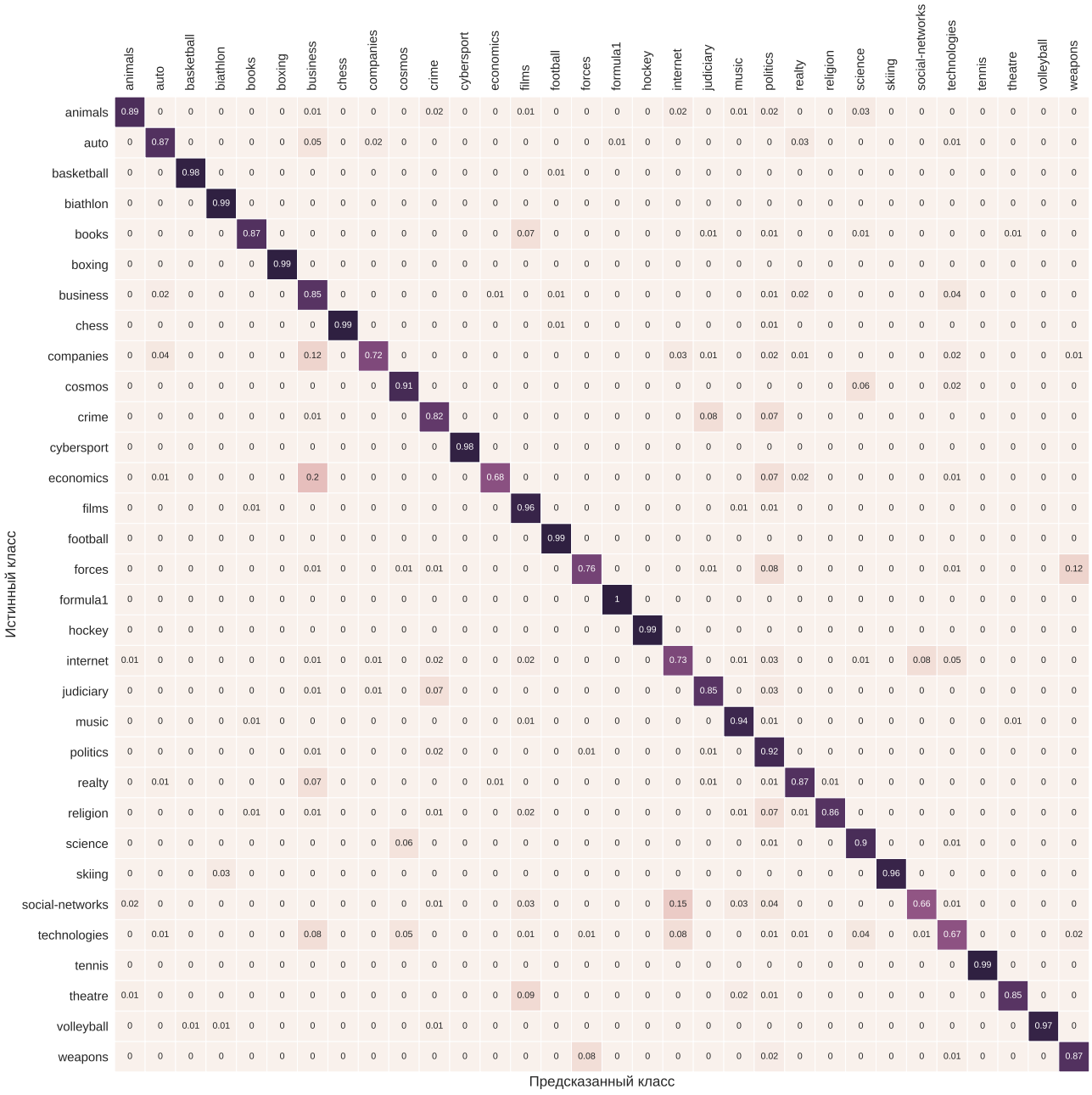


Рис. 8. Ошибки классификатора SVM