МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО “СЕВЕРО-КАВКАЗСКИЙ ГОРНО-МЕТАЛЛУРГИЧЕСКИЙ ИНСТИТУТ   
(ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)

**Факультет** Информационных технологий и электронной техники

**Кафедра** Информатика и вычислительная техника

**Направление подготовки** Информатика и вычислительная техника

**Профиль** АСУ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

на тему: Разработка системы кластеризации документов на базе методов машинного обучения.

**Студент**  Кастуев Хетаг Асланович

**Руководитель проекта** Будаева А.А.

Проект рассмотрен кафедрой и допущен к защите в ГЭК

**Заведующий кафедрой** проф. Гроппен В. О.

г. Владикавказ 2021 г.

# Реферат

Пояснительная записка 100 с., 13 рис., 12 табл., 8 ист., 2 прил.

Кластеризация текстовых документов; мешок слов.

**Объект разработки:** системы кластеризации документов на базе методов машинного обучения.

**Цель выпускной работы:** создание ПП, способного кластеризовывать документы.

**Использованное прикладное и системное программное обеспечение:**

Операционная система Windows 10 PRO – 64 bit.

Среда разработки IntelliJ IDEA.

**Характеристики вычислительной системы:**

AMD Ryzen 5 3600 6-Core Processor 3.6 (4,2) ГГц   
NVIDIA GeForce RTX 2060 6GB  
ОЗУ 16Гб

**Полученные результаты:**

**<…>**

Оглавление

[Реферат 2](#_Toc73287916)

[Введение 5](#_Toc73287917)

[Глава 1. Аналитический обзор 9](#_Toc73287918)

[1.1. Цели кластеризации 9](#_Toc73287919)

[1.2. Применение 10](#_Toc73287920)

[1.2.1. Биология и биоинформатика 10](#_Toc73287921)

[1.2.2. Медицина 11](#_Toc73287922)

[1.2.3. Маркетинг 11](#_Toc73287923)

[1.2.4. Интернет 11](#_Toc73287924)

[1.2.5. Компьютерные науки 11](#_Toc73287925)

[1.3. Алгоритмы классификации 12](#_Toc73287926)

[1.3.1. Алгоритмы иерархической кластеризации; 12](#_Toc73287927)

[1.3.2. Алгоритмы квадратичной ошибки 12](#_Toc73287928)

[1.3.3. Нечеткие алгоритмы 13](#_Toc73287929)

[1.3.4. Алгоритмы, основанные на теории графов 14](#_Toc73287930)

[1.3.5. Алгоритм выделения связных компонент 15](#_Toc73287931)

[1.3.6. Алгоритм минимального покрывающего дерева 15](#_Toc73287932)

[1.3.7. Послойная кластеризация 16](#_Toc73287933)

[1.4. Сравнение алгоритмов 17](#_Toc73287934)

[1.4.1. Вычислительная сложность алгоритмов 17](#_Toc73287935)

[1.4.2. Сравнительная таблица алгоритмов 17](#_Toc73287936)

[1.5. Мера расстояний 18](#_Toc73287937)

[1.5.1. Евклидово расстояние 19](#_Toc73287938)

[1.5.2. Квадрат евклидова расстояния 19](#_Toc73287939)

[1.5.3. Расстояние городских кварталов 19](#_Toc73287940)

[1.5.4. Расстояние Чебышева 20](#_Toc73287941)

[1.5.5. Степенное расстояние 20](#_Toc73287942)

[1.6. Существующие программные средства 20](#_Toc73287943)

[1.7. Заключение 21](#_Toc73287944)

[Глава 2. Выбор и обоснование алгоритма 22](#_Toc73287945)

[Глава 3. ПРОГРАММНЫЙ КОМПЛЕКС 24](#_Toc73287946)

[3.1. Архитектура программного обеспечения 24](#_Toc73287947)

[3.2. Язык программирования Java 24](#_Toc73287948)

[3.3. Работа программного комплекса 25](#_Toc73287949)

[3.3.1. Кластеризация 25](#_Toc73287950)

[3.3.2. Классификация 28](#_Toc73287951)

[Глава 4. Экспериментальная часть 31](#_Toc73287952)

[4.1. Оценка производительности алгоритма кластеризации 32](#_Toc73287953)

[4.1.1. Зависимость от количества слов в файле 32](#_Toc73287954)

[4.1.2. Зависимость от количества файлов 33](#_Toc73287955)

[4.2. Оценка производительности алгоритма классификации 34](#_Toc73287956)

[4.2.1. Зависимость от количества слов в файле 34](#_Toc73287957)

[4.2.2. Зависимость от количества файлов 35](#_Toc73287958)

[4.2.3. Зависимость от количества классов 36](#_Toc73287959)

[ГЛАВА 5. Заключение 37](#_Toc73287960)

[Список использованных источников 38](#_Toc73287961)

# Введение

Трудно представить себе жизнь современного человека без машинного обучения и искусственного интеллекта. Данные технологии создают новые возможности. Например, системы безопасности в метро используют машинное обучение для распознавания и поиска лиц, которые находятся в розыске. Компьютерное зрение используется в беспилотных автомобилях и летательных аппаратах для коммерческих и военных целей.

Машинный интеллект - это направление технологических исследований. Оно связано с созданием системы, которая способна обучаться самостоятельно.

Искусственный интеллект широко применяется в самых разных областях, как:

1. lT-сфера: разработка приложений;

2. Рекламные компании;

3. Маркетинговые исследования;

4. Медицинская диагностика;

5. Техническая диагностика;

6. Автопилотирование;

7. Биоинформатика и в другие сферы;

8. Обработка текста (в том числи кластеризация).

Все это и многое другое работает при помощи машинного интеллекта.

С одной стороны, такой технический прогресс приносит человеку огромную пользу, взять хотя бы медицинскую диагностику: многие исследования стали быстрыми и доступными, а с другой стороны такой технический прогресс бросает вызов каждому из нас: ведь совсем скоро не нужны будут машинисты в электропоездах, многий обслуживающий персонал заменят компьютеры, такcистов заменят беспилотные автомобили. Поэтому тему машинного обучения я считаю актуальной в настоящее время с научной и практической точки зрения. Нужно не только знать, что существует так называемое машинное обучение, но и понимать, как это работает.

Машинное обучение избавляет программиста от необходимости писать большой код, т.е объяснять компьютеру, как нужно решить какую-нибудь проблему. В процессе машинного обучения компьютер учат самостоятельно находить правильное решение. То есть у нас есть какие-либо известные данные и на основе их мы собираем статистику и уже на новых данных мы учимся что-то понимать (находим новые, ещё не описанные закономерности).

Таким образом: Основная идея машинного обучения состоит в том, чтобы научить компьютер "учиться", т.е. вычленять из каких-либо данных полезные знания.

Надо отметить, что обучение может происходить как до работы программы, так и в процессе выполнения программы. Например, программа для распознавания лиц сначала учится, а затем работает на настоящих данных; программа распознавания спама учится во время исполнения.

На сегодняшний день выделяют два основных типа машинного обучения: дедуктивное обучение (аналитическое) и индуктивное ( статистическое)обучение.

Дедуктивное обучение относится к области экспертных систем. Экспертные системы не могут решать универсальные задачи как нейронные сети.

Экспертные системы - это большой обьем знаний экспертов формализация этих знаний инженерами по знаниям + грамотный код программиста. В итоге мы получаем хорошо составленную базу знаний. Минусом этой системы можно назвать то, что без поддержки разработчиков базы знаний быстро становятся невостребованными. Дедуктивное обучение (поскольку относится к экспертным системам) решает только узконаправленные специализированные задачи. Имеются знания экспертов и перенос их в компьютер в виде базы знаний. Во всех задачах дедуктивного обучения имеются знания, каким-либо образом отобранные и формализованные. Требуется вывести из них правило, применительно к каждому конкретному случаю. Что же касается второго типа, то здесь лежит выявление закономерностей в эмпирической информации. Обучающийся программе мы предлагаем большое количество примеров данных с закономерностями и на основе полученного опыта программа сама учится строить закономерности. Индуктивное обучение подразделяется на:

– Обучение с учителем - также называют "обучение по прецедентам".

– Обучение без учителя - обучение, в котором нет правильных ответов, только данные;

– Обучение с подкрепление, также называют "стимулируемое обучение" - обучение, в котором агент учится посредством собственных проб и ошибок;

– Активное обучение - очень похоже на обучение с учителем. Разница состоит в том, что ответы изначально неизвестны. Идея в том, что алгоритм сам может обучаться на малых выборках, если он сам выбирает какие данные ему нужны. То есть алгоритм составляет запросы, ответы на которые ему помогают обучаться.

– Частичное обучение - в этом случае большая часть ответов неизвестна.

В настоящее время крупные компании вкладывают большие средства в машинное обучение, потому что данная технология по-настоящему окупается. С каждым годом большие данные становятся все более сложнее и человеку становится уже недостаточно своих аналитических способностей. Некоторые задачи перейдут к искусственному интеллекту, который справится с ними лучше и быстрее, чем человек. Прогнозируется, что в ближайшее десятилетие искусственный интеллект займет около 7% рабочих мест в России. Часть процессов в организациях автоматизируют и управлять этими процессами поручат самообучающимся алгоритмам. Огромно место займет ИИ в интернете вещей. Интернет вещей требует обработки большого потока информации в реальном времени. Устройства, подключенные к сети генерируют гигантские массивы данных, которые необходимо будет обрабатывать, анализировать и хранить. Как пример, зубные щетки с функцией блютуз, которые будут отправлять информацию о состоянии зубов стоматологу или дроны, которые станут незаменимы в сельском хозяйстве. Они будут собирать информацию о зрелости урожая, о состоянии почвы, о вредителях и болезнях растений. И во всех технологиях будет использовать машинное обучение. Поэтому машинное обучение готовит для нас перспективное будущее и массу интересных инноваций.

Целью данной дипломной работы является разработка системы кластеризации документов на базе методов машинного обучения.

# Глава 1. Аналитический обзор

Задача кластеризации – частный случай задачи машинного обучения, в частности, обучения без учителя, которая сводится к разбиению имеющегося множества объектов данных на подмножества таким образом, что элементы одного подмножества существенно отличались по некоторому набору свойств от элементов всех других подмножеств. Объект данных обычно рассматривается как точка в многомерном метрическом пространстве, каждому измерению которого соответствует некоторое свойство (атрибут) объекта, а метрика – есть функция от значений данных свойств. От типов измерений этого пространства, которые могут быть как числовыми, так и категориальными, зависит выбор алгоритма кластеризации данных и используемая метрика. Этот выбор продиктован различиями в природе разных типов атрибутов.

* 1. Цели кластеризации
* Понимание данных путём выявления кластерной структуры. Разбиение выборки на группы схожих объектов позволяет упростить дальнейшую обработку данных и принятия решений, применяя к каждому кластеру свой метод анализа (стратегия «разделяй и властвуй»).
* Сжатие данных. Если исходная выборка избыточно большая, то можно сократить её, оставив по одному наиболее типичному представителю от каждого кластера.
* Обнаружение новизны. Выделяются нетипичные объекты, которые не удаётся присоединить ни к одному из кластеров.

В первом случае число кластеров стараются сделать поменьше. Во втором случае важнее обеспечить высокую степень сходства объектов внутри каждого кластера, а кластеров может быть сколько угодно. В третьем случае наибольший интерес представляют отдельные объекты, не вписывающиеся ни в один из кластеров.

Во всех этих случаях может применяться иерархическая кластеризация, когда крупные кластеры дробятся на более мелкие, те в свою очередь дробятся ещё мельче, и т. д. Такие задачи называются задачами таксономии.

Результатом таксономии является древообразная иерархическая структура. При этом каждый объект характеризуется перечислением всех кластеров, которым он принадлежит, обычно от крупного к мелкому.

Классическим примером таксономии на основе сходства является биноминальная номенклатура живых существ, предложенная Карлом Линнеем в середине XVIII века. Аналогичные систематизации строятся во многих областях знания, чтобы упорядочить информацию о большом количестве объектов.

* 1. Применение

### Биология и биоинформатика

* В области экологии кластеризация используется для выделения пространственных и временных сообществ организмов в однородных условиях;
* Кластерный анализ используется для группировки схожих геномных последовательностей в семейство генов, которые являются консервативными структурами для многих организмов и могут выполнять схожие функции;
* Кластеризация помогает автоматически определять генотипы по различным частям хромосом;
* Алгоритмы применяются для выделения небольшого числа групп генетических вариации человеческого генома.

### Медицина

* Используется в позитронно-эмиссионной томографии для автоматического выделения различных типов тканей на трехмерном изображении;
* Применяется для выявления шаблонов устойчивости к антибиотикам; для классификации антибиотиков по типу антибактериальной активности.

### Маркетинг

* Кластеризация широко используется при изучении рынка для обработки данных, полученных из различных опросов. Может применяться для выделения типичных групп покупателей, разделения рынка для создания персонализированных предложений, разработки новых линий продукции.

### Интернет

* Выделение групп людей на основе графа связей в социальных сетях;
* Повышение релевантности ответов на поисковые запросы путем группировки веб-сайтов по смысловым значениям поискового запроса.

### Компьютерные науки

* Кластеризация используется в сегментации изображений для определения границ и распознавания объектов;
* Кластерный анализ применяется для определения образовавшихся популяционных ниш в ходе работы эволюционных алгоритмов для улучшения параметров эволюции;
* Подбор рекомендаций для пользователя на основе предпочтений других пользователей в данном кластере;
* Определение аномалий путем построения кластеров и выявления неклассифицированных объектов.

## Алгоритмы классификации

### Алгоритмы иерархической кластеризации;

Среди алгоритмов иерархической кластеризации выделяются два основных типа: восходящие и нисходящие алгоритмы. Нисходящие алгоритмы работают по принципу «сверху-вниз»: в начале все объекты помещаются в один кластер, который затем разбивается на все более мелкие кластеры. Более распространены восходящие алгоритмы, которые в начале работы помещают каждый объект в отдельный кластер, а затем объединяют кластеры во все более крупные, пока все объекты выборки не будут содержаться в одном кластере. Таким образом строится система вложенных разбиений. Результаты таких алгоритмов обычно представляют в виде дерева – дендрограммы. Классический пример такого дерева – классификация животных и растений.

Для вычисления расстояний между кластерами чаще все пользуются двумя расстояниями: одиночной связью или полной связью (см. обзор мер расстояний между кластерами).

К недостатку иерархических алгоритмов можно отнести систему полных разбиений, которая может являться излишней в контексте решаемой задачи.

### Алгоритмы квадратичной ошибки

Задачу кластеризации можно рассматривать как построение оптимального разбиения объектов на группы. При этом оптимальность может быть определена как требование минимизации среднеквадратической ошибки разбиения:

где *cj* — «центр масс» кластера *j* (точка со средними значениями характеристик для данного кластера).

Алгоритмы квадратичной ошибки относятся к типу плоских алгоритмов. Самым распространенным алгоритмом этой категории является метод k-средних. Этот алгоритм строит заданное число кластеров, расположенных как можно дальше друг от друга. Работа алгоритма делится на несколько этапов:

1. Случайно выбрать *k* точек, являющихся начальными «центрами масс» кластеров.
2. Отнести каждый объект к кластеру с ближайшим «центром масс».
3. Пересчитать «центры масс» кластеров согласно их текущему составу.
4. Если критерий остановки алгоритма не удовлетворен, вернуться к п. 2.

В качестве критерия остановки работы алгоритма обычно выбирают минимальное изменение среднеквадратической ошибки. Так же возможно останавливать работу алгоритма, если на шаге 2 не было объектов, переместившихся из кластера в кластер.

К недостаткам данного алгоритма можно отнести необходимость задавать количество кластеров для разбиения.

### Нечеткие алгоритмы

Наиболее популярным алгоритмом нечеткой кластеризации является алгоритм c-средних (c-means). Он представляет собой модификацию метода k-средних. Шаги работы алгоритма:

1. Выбрать начальное нечеткое разбиение *n* объектов на *k* кластеров путем выбора матрицы принадлежности *U* размера *n x k*.
2. Используя матрицу U, найти значение критерия нечеткой ошибки:  
   https://habrastorage.org/getpro/habr/post_images/21c/c5d/72b/21cc5d72b0c9bee8ad0476798eb4093c.jpg,  
   где *ck* — «центр масс» нечеткого кластера *k*:  
   https://habrastorage.org/getpro/habr/post_images/5fb/811/641/5fb811641b4fa3c84ba2490811ff53ea.jpg.
3. Перегруппировать объекты с целью уменьшения этого значения критерия нечеткой ошибки.
4. Возвращаться в п. 2 до тех пор, пока изменения матрицы *U* не станут незначительными.

Этот алгоритм может не подойти, если заранее неизвестно число кластеров, либо необходимо однозначно отнести каждый объект к одному кластеру.

### Алгоритмы, основанные на теории графов

Суть таких алгоритмов заключается в том, что выборка объектов представляется в виде графа G=(V, E), вершинам которого соответствуют объекты, а ребра имеют вес, равный «расстоянию» между объектами. Достоинством графовых алгоритмов кластеризации являются наглядность, относительная простота реализации и возможность вносения различных усовершенствований, основанные на геометрических соображениях. Основными алгоритмам являются алгоритм выделения связных компонент, алгоритм построения минимального покрывающего (остовного) дерева и алгоритм послойной кластеризации.

### Алгоритм выделения связных компонент

В алгоритме выделения связных компонент задается входной параметр R и в графе удаляются все ребра, для которых «расстояния» больше R. Соединенными остаются только наиболее близкие пары объектов. Смысл алгоритма заключается в том, чтобы подобрать такое значение R, лежащее в диапазон всех «расстояний», при котором граф «развалится» на несколько связных компонент. Полученные компоненты и есть кластеры.

Для подбора параметра R обычно строится гистограмма распределений попарных расстояний. В задачах с хорошо выраженной кластерной структурой данных на гистограмме будет два пика – один соответствует внутрикластерным расстояниям, второй – межкластерным расстояния. Параметр R подбирается из зоны минимума между этими пиками. При этом управлять количеством кластеров при помощи порога расстояния довольно затруднительно.

### Алгоритм минимального покрывающего дерева

Алгоритм минимального покрывающего дерева сначала строит на графе минимальное покрывающее дерево, а затем последовательно удаляет ребра с наибольшим весом. На рисунке изображено минимальное покрывающее дерево, полученное для девяти объектов.



Путём удаления связи, помеченной CD, с длиной равной 6 единицам (ребро с максимальным расстоянием), получаем два кластера: {A, B, C} и {D, E, F, G, H, I}. Второй кластер в дальнейшем может быть разделён ещё на два кластера путём удаления ребра EF, которое имеет длину, равную 4,5 единицам.

### Послойная кластеризация

Алгоритм послойной кластеризации основан на выделении связных компонент графа на некотором уровне расстояний между объектами (вершинами). Уровень расстояния задается порогом расстояния *c*. Например, если расстояние между объектами https://habrastorage.org/getpro/habr/post_images/f67/e6a/707/f67e6a707dcac300ff90658e523b135f.jpg, то https://habrastorage.org/getpro/habr/post_images/e91/fe1/65a/e91fe165a15d6fa1981cbb51f0e7fff3.jpg.

Алгоритм послойной кластеризации формирует последовательность подграфов графа *G*, которые отражают иерархические связи между кластерами:

https://habrastorage.org/getpro/habr/post_images/5ec/cd0/7cc/5eccd07cc4e56b227092cbe66501a67f.jpg,  
  
где *Gt = (V, Et)* — граф на уровне *сt*,  
https://habrastorage.org/getpro/habr/post_images/35f/249/c62/35f249c626b8d0e118efc7f472beee48.jpg,  
*сt* – t-ый порог расстояния,  
m – количество уровней иерархии,  
*G0 = (V, o)*, o – пустое множество ребер графа, получаемое при *t0* = 1,  
*Gm = G*, то есть граф объектов без ограничений на расстояние (длину ребер графа), поскольку *tm* = 1.  
  
Посредством изменения порогов расстояния {*с0, …, сm*},   
где 0 = *с0* < *с1* < …< *сm* = 1, возможно контролировать глубину иерархии получаемых кластеров. Таким образом, алгоритм послойной кластеризации способен создавать как плоское разбиение данных, так и иерархическое.

## Сравнение алгоритмов

### Вычислительная сложность алгоритмов

| **Алгоритм кластеризации** | **Вычислительная сложность** |
| --- | --- |
| Иерархический | O(n2) |
| k-средних c-средних | O(nkl),  где k – число кластеров, l – число итераций |
| Выделение связных компонент | зависит от алгоритма |
| O(n2 log n) | Минимальное покрывающее дерево |
| Послойная кластеризация | O(max(n, m)), где m |

### Сравнительная таблица алгоритмов

| **Алгоритм кластеризации** | **Форма кластеров** | **Входные данные** | **Результаты** |
| --- | --- | --- | --- |
| Иерархический | Произвольная | Число кластеров или порог расстояния для усечения иерархии | Бинарное дерево кластеров |
| k-средних | Гиперсфера | Число кластеров | Центры кластеров |
| c-средних | Гиперсфера | Число кластеров, степень нечеткости | Центры кластеров, матрица принадлежности |
| Выделение связных компонент | Произвольная | Порог расстояния R | Древовидная структура кластеров |
| Минимальное покрывающее дерево | Произвольная | Число кластеров или порог расстояния для удаления ребер | Древовидная структура кластеров |
| Послойная кластеризация | Произвольная | Последовательность порогов расстояния | Древовидная структура кластеров с разными уровнями иерархии |

## Мера расстояний

Итак, как же определять «похожесть» объектов? Для начала нужно составить вектор характеристик для каждого объекта — как правило, это набор числовых значений, например, рост-вес человека. Однако существуют также алгоритмы, работающие с качественными (т.н. категорийными) характеристиками.

После того, как мы определили вектор характеристик, можно провести нормализацию, чтобы все компоненты давали одинаковый вклад при расчете «расстояния». В процессе нормализации все значения приводятся к некоторому диапазону, например, [-1, -1] или [0, 1].

Наконец, для каждой пары объектов измеряется «расстояние» между ними — степень похожести.

### Евклидово расстояние

Наиболее распространенная функция расстояния. Представляет собой геометрическим расстоянием в многомерном пространстве:

### Квадрат евклидова расстояния

Применяется для придания большего веса более отдаленным друг от друга объектам. Это расстояние вычисляется следующим образом:

### Расстояние городских кварталов

Это расстояние является средним разностей по координатам. В большинстве случаев эта мера расстояния приводит к таким же результатам, как и для обычного расстояния Евклида. Однако для этой меры влияние отдельных больших разностей (выбросов) уменьшается (т.к. они не возводятся в квадрат). Формула для расчета манхэттенского расстояния:

### Расстояние Чебышева

Это расстояние может оказаться полезным, когда нужно определить два объекта как «различные», если они различаются по какой-либо одной координате. Расстояние Чебышева вычисляется по формуле:

### Степенное расстояние

Применяется в случае, когда необходимо увеличить или уменьшить вес, относящийся к размерности, для которой соответствующие объекты сильно отличаются. Степенное расстояние вычисляется по следующей формуле:

где *r* и *p* – параметры, определяемые пользователем. Параметр *p* ответственен за постепенное взвешивание разностей по отдельным координатам, параметр *r* ответственен за прогрессивное взвешивание больших расстояний между объектами. Если оба параметра – *r* и *p* — равны двум, то это расстояние совпадает с расстоянием Евклида.

## Существующие программные средства

На данный момент существует не одно программное средство, реализующее различные алгоритмы кластеризации. Примером такого проекта является Apache Mahout. Apache Mahout ― это открытый проект Apache Software Foundation (ASF), основной целью которого является создание масштабируемых алгоритмов машинного обучения, которые предлагаются для бесплатного использования по лицензии Apache. Mahout содержит реализации кластеризации, категоризации, CF и эволюционного программирования. Более того, там, где это разумно, он использует библиотеку Apache Hadoop, что позволяет Mahout эффективно масштабироваться в облаке.

## Заключение

В данной главе были описаны алгоритмы кластеризации и показано их сравнение.

Из рассмотренных в первой главе алгоритмов был выбран алгоритм минимального покрывающего дерева, так как он имеет произвольную форму кластера, а в качестве критерия схожести используется мера TF-IDF, так как этот алгоритм был предложен в задании к выпускной работе.

В третьей главе будет более подробно описан алгоритм кластеризации с помощью минимального покрывающего дерева.

# Глава 2. Выбор и обоснование алгоритма

*Содержательная постановка задачи*

Имеется фиксированное множество документов. Требуется разбить заданное множество документов на группы в соответствии с их семантической схожестью.

Предусмотреть 2 режима работы:

1. автоматическое выявление групп документов
2. выбор одной группы документа (из заранее определенных групп) на основании его содержания.

В качестве критериев схожести документов при кластеризации используется мера TF-IDF.

**Обозначения и формальная постановка задачи**

Обозначения:

*D* – исходное множество документов коллекции,

*di* ∈ *D* – *i*-й документ множества *D*,

*tj* ∈ *di* – *j*-е слово документа *di*,

 – число вхождений *tj*-го слова в документ *di*,

 – общее число слов в документе *di*, 

 – частота *tj*-го слова (term frequency) в документе *di*,

 – обратная частота документа (inverse document frequency) – инверсия частоты, с которой *tj*-ое слово встречается в документах коллекции

– число документов *D*, в которых встречается слово *tj* (когда )

 – булева переменная, равная 1 – если *di*-й документ относится к таксону *pk* и 0 – в противном случае,

 – оценка важности слова в контексте документа (мера TF-IDF)

- евклидово расстояние между *i*-м и *l*-м документами на основании меры TF-IDF



**Цель** – суммарное расстояния между документами, попавшими в один таксон, должно быть минимальным.

# Глава 3. ПРОГРАММНЫЙ КОМПЛЕКС

## 3.1. Архитектура программного обеспечения

Разработанное программное средство реализовано на языке Java, что в свою очередь позволяет запускать данный ПП под управлением любой ОС, на которой заранее установлена соответствующая виртуальная машина.

В качестве среды для разработки программного продукта выбрана интегрированная среда разработки IntelliJ IDEA. Данная среда зарекомендовала себя, как надежный инструмент разработки, и имеет следующие преимущества:

1. Поддержка множества языков программирования;
2. Кроссплатформенность;
3. Наличие бесплатной версии (Community);
4. Удобный инструмент отладки;
5. Поддержка системы контроля версиями;

Из минусов можно выделить разве что высокое потребление ресурсов, но при текущих мощностях современных ЭВМ данный пункт нивелируется.

## 3.2. Язык программирования Java

Язык программирования в значительной степени влияет на качество и надежность программного обеспечения. Для реализации поставленной задачи был выбран язык программирования Java по следующим причинам:

1. Объектно-ориентированный подход к разработке. Это означает, что у разработчика появляется возможность описывать абстрактные конструкции на основе предметной области, а потом реализовывать между ними взаимодействие;
2. Большое количество готовых библиотек, ускоряющих разработку;
3. Удобная среда разработки (IntelliJ IDEA);
4. Безопасный код;
5. Кроссплатформенность.

Java — объектно-ориентированный язык программирования, разрабатываемый компанией Sun Microsystems и официально выпущенный 23 мая 1995 года.

## 3.3. Работа программного комплекса

В программе предусмотрено 2 режима работы:

* 1. Кластеризация
  2. Классификация

Далее будут продемонстрированы оба режима работы.

### 3.3.1. Кластеризация

Изначально, при запуске программы, бывает выбрал режим работы кластеризации, как это показано на рисунке 3.1.

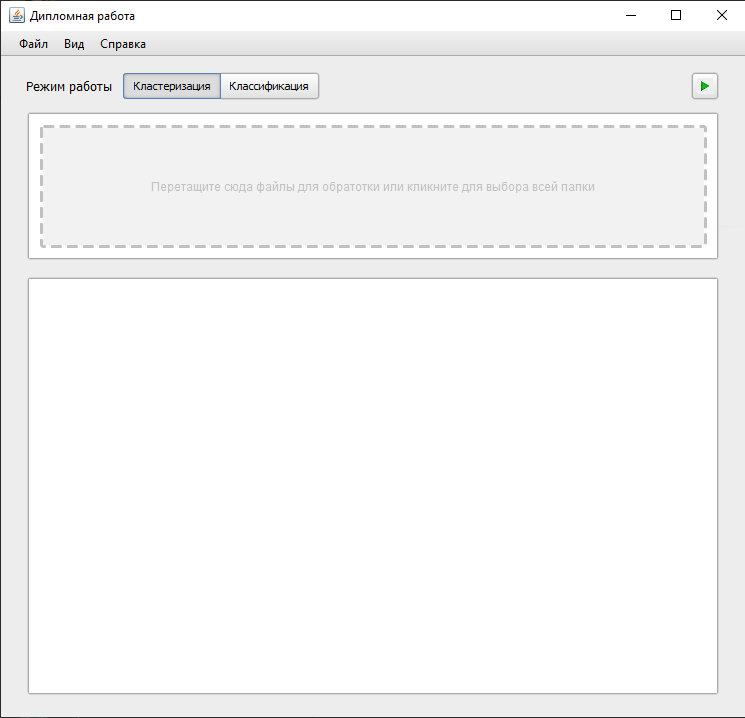
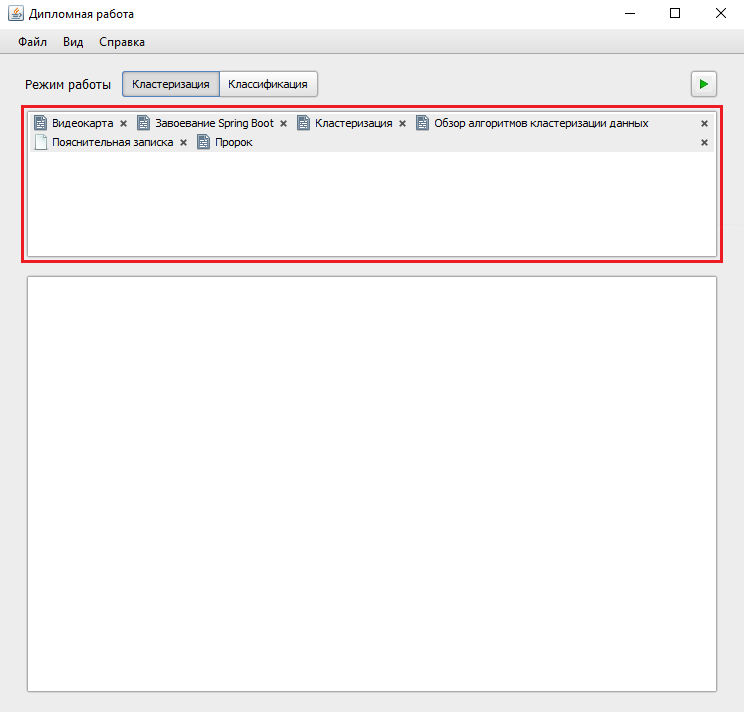


Рисунок 3.1.

Далее необходимо перетащить нужные файлы в соответствующую область или один раз кликнуть по той же области, для того чтобы в диалоговом окне выбрать папку, в которой мы хотим обработать файлы.

  
Рисунок 3.2.

Далее нужно запустить программу.

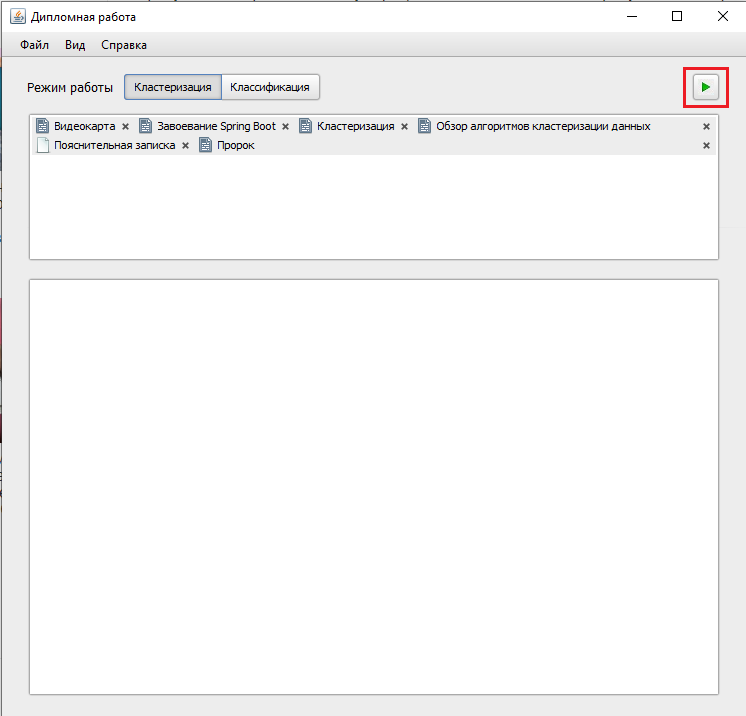


Рисунок 3.3.

И в текстовом поле появится результат работы

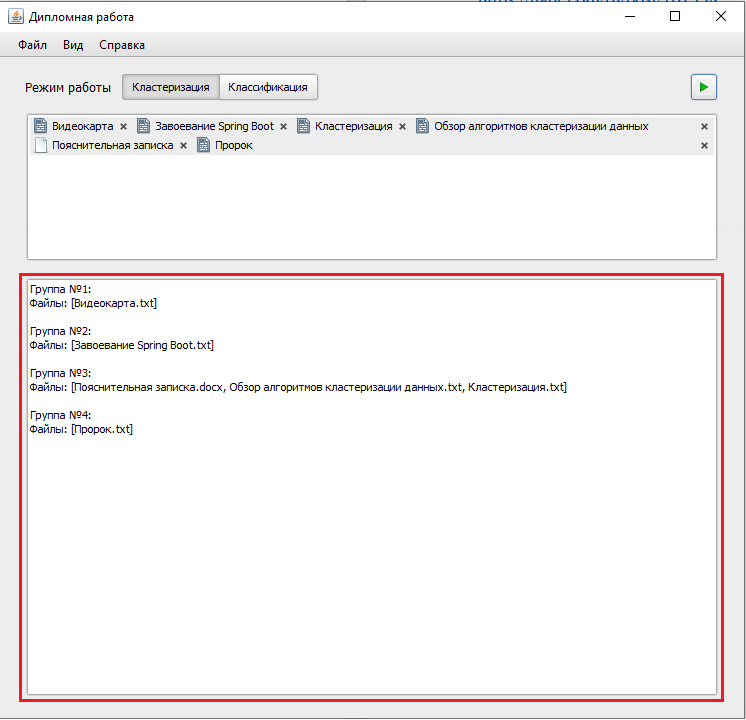
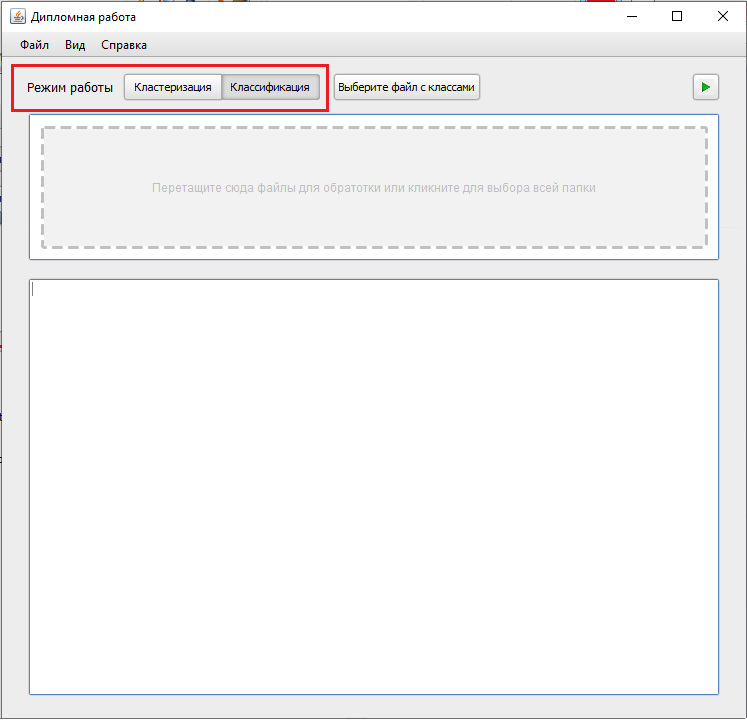


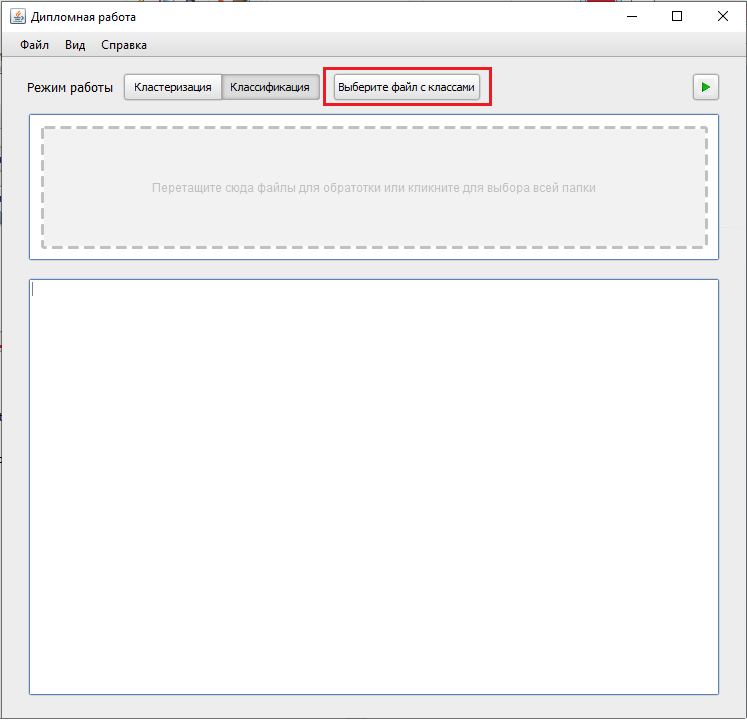
Рисунок 3.4.

### 3.3.2. Классификация

Для того, чтобы программа начала работать в режиме классификации необходимо выбрать соответствующий режим:

  
Рисунок 3.5.

Далее нужно выбрать файл с заранее заготовленными классами

  
Рисунок 3.6.

Далее необходимо перетащить нужные файлы в соответствующую область или один раз кликнуть по той же области, для того чтобы в диалоговом окне выбрать папку, в которой мы хотим обработать файлы.

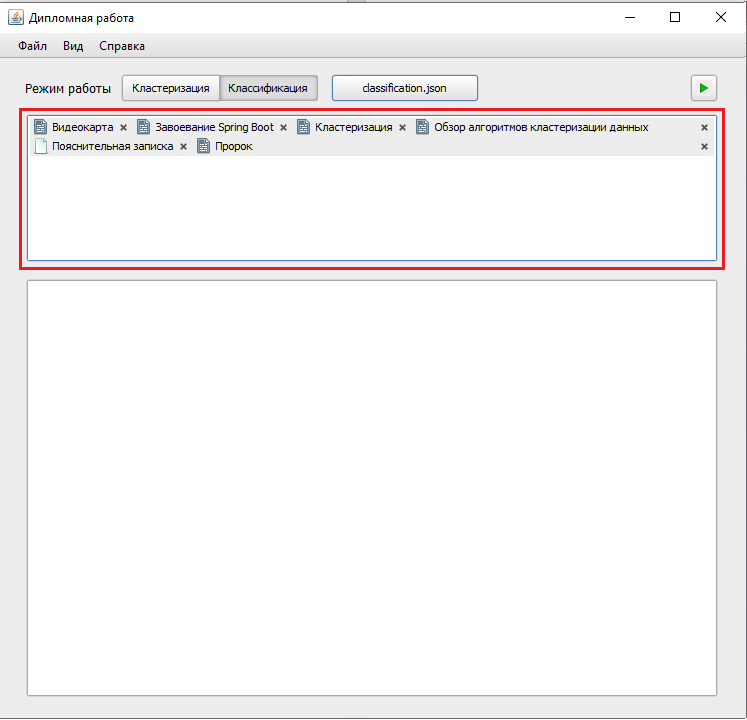
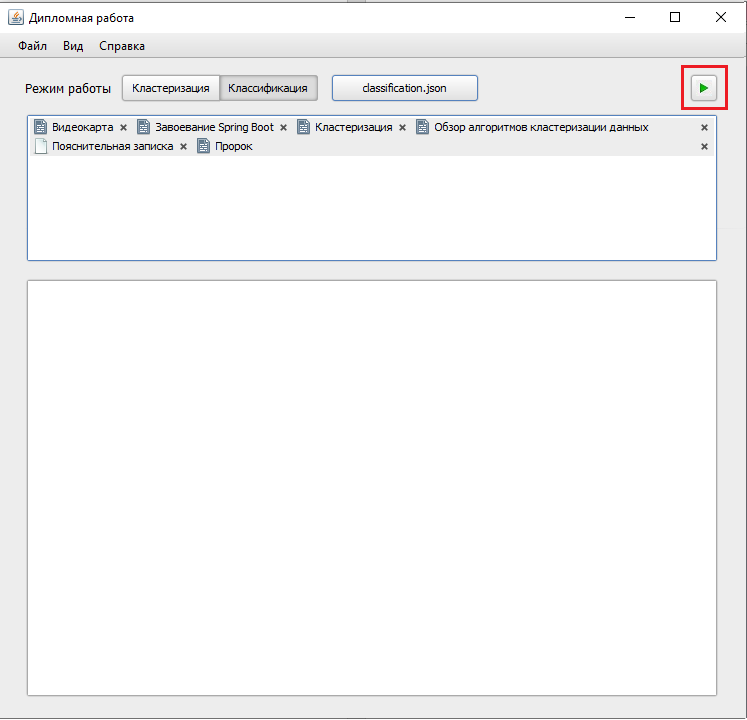


Рисунок 3.7.

И, наконец, запустить программу

  
Рисунок 3.8.

И в текстовое поле выведется результат работы программы

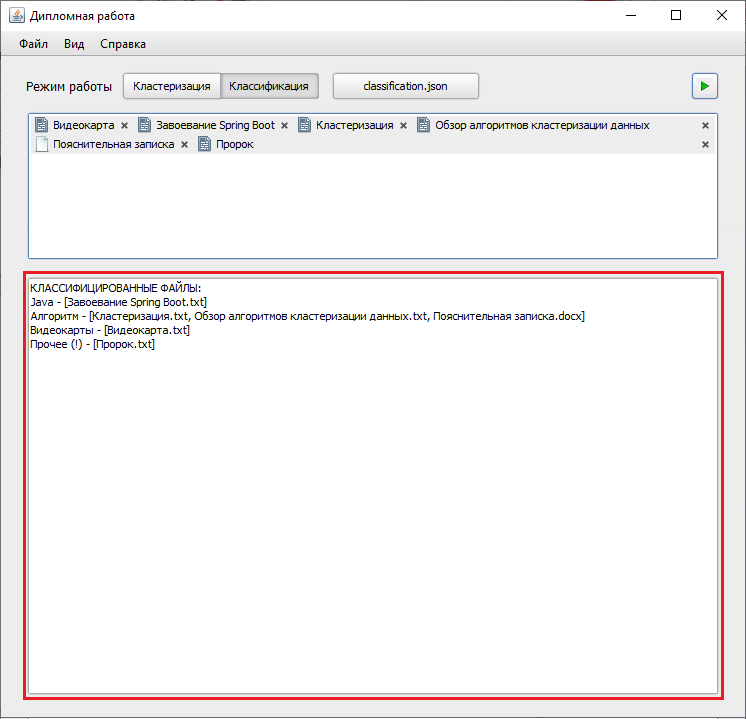
****

Рисунок 3.9.

# Глава 4. Экспериментальная часть

Для проведения анализа была проведена серия экспериментов.

Для проведения экспериментов использовался ПК со следующими характеристиками:

|  |  |
| --- | --- |
| Производитель процессора | AMD |
| Тип процессора | Ryzen 5 3600 |
| Количество ядер процессора | 6 |
| Баз. такт. частота | 3.59 GHz |
| Кэш память | 32 МБ |
| Оперативная память (RAM) | 16 Гб |
| Тип памяти | DDR4 |
| Производитель видеокарты | NVIDIA |
| Графический контроллер | GeForce RTX 2060 |
| Жесткий диск (SSD) | 256 ГБ |
| ОС | Windows 10 PRO (64 bit) |

Таблица 4.1. Характеристики ПК

Главной исследуемой характеристикой работы программного комплекса является время его работы.

## 4.1. Оценка производительности алгоритма кластеризации

### 4.1.1. Зависимость от количества слов в файле

**Цели эксперимента:**

* Построить графические зависимости, отражающие среднее время кластеризации в зависимости от количества слов в файле;
* Определить аналитическую зависимость, отражающие среднее время кластеризации в зависимости от количества слов в файле.

**Ход эксперимента:**

Разместить N файлов, с количеством слов в диапазоне 5000 ≤ N ≤ 35000 c шагом 5000. При каждом фиксированном N алгоритм отрабатывает 100 раз.

|  |  |
| --- | --- |
| Кол-во слов | Время работы (мс) |
| 5000 | 2195,72 |
| 10000 | 2735,15 |
| 15000 | 3108,07 |
| 20000 | 3283,41 |
| 25000 | 3509,29 |
| 30000 | 3813,6 |

Таблица 4.2. Время работы кластеризации при фиксированном кол-ве файлов

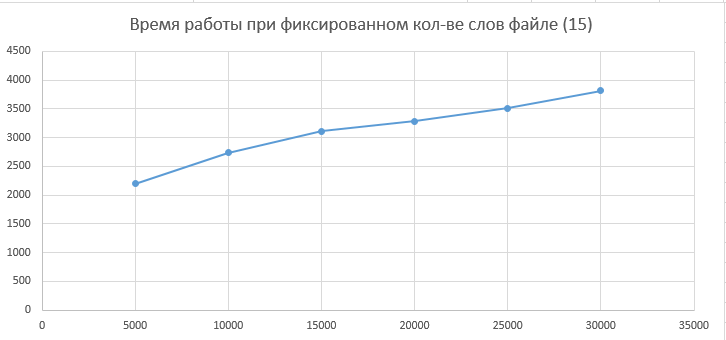


График 4.1. Время работы кластеризации при фиксированном кол-ве файлов

### 4.1.2. Зависимость от количества файлов

**Цели эксперимента:**

* Построить графические зависимости, отражающие среднее время кластеризации в зависимости от количества файлов;
* Определить аналитическую зависимость, отражающие среднее время кластеризации в зависимости от количества файлов.

**Ход эксперимента:**

Разместить N файлов в диапазоне 5 ≤ N ≤ 20 c шагом 5. При каждом фиксированном N алгоритм отрабатывает 100 раз.

|  |  |
| --- | --- |
| Кол-во файлов | Время работы (мс) |
| 5 | 426,88 |
| 10 | 1309,14 |
| 15 | 3031,75 |
| 20 | 5926,5 |

Таблица 4.3. Время работы кластеризации при фиксированном кол-ве файлов



График 4.2. Время работы кластеризации при фиксированном кол-ве файлов

## 4.2. Оценка производительности алгоритма классификации

### 4.2.1. Зависимость от количества слов в файле

**Цели эксперимента:**

* Построить графические зависимости, отражающие среднее время классификации в зависимости от количества слов в файле;
* Определить аналитическую зависимость, отражающие среднее время классификации в зависимости от количества слов в файле.

**Ход эксперимента:**

Разместить N файлов, с количеством слов в диапазоне 5000 ≤ N ≤ 35000 c шагом 5000. При каждом фиксированном N алгоритм отрабатывает 100 раз.

|  |  |
| --- | --- |
| Кол-во слов | Время работы (мс) |
| 5000 | 298,92 |
| 10000 | 576,34 |
| 15000 | 854,73 |
| 20000 | 1166,2 |
| 25000 | 1422,53 |
| 30000 | 1690,57 |

Таблица 4.4. Время классификации в зависимости от кол-ва слов

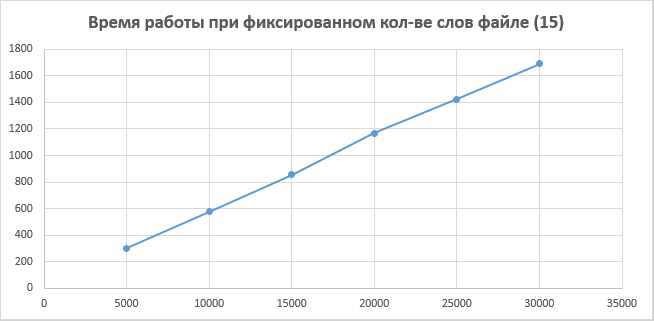


График 4.3. Время классификации в зависимости от кол-ва слов

### 4.2.2. Зависимость от количества файлов

**Цели эксперимента:**

* Построить графические зависимости, отражающие среднее время классификации в зависимости от количества файлов;
* Определить аналитическую зависимость, отражающие среднее время классификации в зависимости от количества файлов.

**Ход эксперимента:**

Разместить N файлов в диапазоне 5 ≤ N ≤ 20 c шагом 5. При каждом фиксированном N алгоритм отрабатывает 100 раз.

|  |  |
| --- | --- |
| Кол-во файлов | Время работы (мс) |
| 5 | 287,19 |
| 10 | 568,7 |
| 15 | 855,5 |
| 20 | 1139,44 |

Таблица 4.5. Время классификации в зависимости от кол-ва файлов

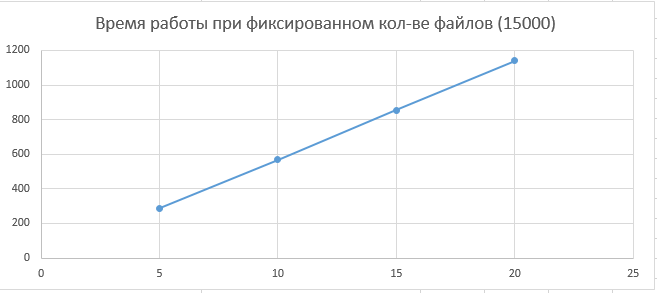


График 4.4. Время классификации в зависимости от кол-ва файлов

### 4.2.3. Зависимость от количества классов

**Цели эксперимента:**

* Построить графические зависимости, отражающие среднее время классификации в зависимости от количества классов;
* Определить аналитическую зависимость, отражающие среднее время классификации в зависимости от количества классов.

**Ход эксперимента:**

Разместить 10 файлов с 15000 слов в каждом и разбить на N классов, в диапазоне 5 ≤ N ≤ 10 c шагом 1. При каждом фиксированном N алгоритм отрабатывает 100 раз.

|  |  |
| --- | --- |
| Кол-во классов | Время работы (мс) |
| 5 | 569,83 |
| 6 | 571,18 |
| 7 | 564,03 |
| 8 | 575,43 |
| 9 | 579,3 |
| 10 | 577,84 |

Таблица 4.6. Время классификации в зависимости от кол-ва классов



График 4.5. Время классификации в зависимости от кол-ва классов

# ГЛАВА 5. Заключение

Данная выпускная работа посвящена программной реализации программного обеспечения, предназначенного для кластеризации и классификации документов.

Поставленную цель считаю достигнутой, а задачи выполненными.

# Список использованных источников

1. Гитис Л. Х*.* Кластерный анализ в задачах классификации, оптимизации и прогнозирования. 104 с.
2. Б. Дюран*.* Кластерный анализ. 128 с.
3. <https://habr.com/ru/post/101338>

# Приложение 1. исходный код программы

Application.java

package diplom;

import com.google.gson.Gson;

import com.google.gson.GsonBuilder;

import diplom.config.Config;

import diplom.gui.Frame;

import java.io.File;

import java.util.Arrays;

import java.util.List;

public class Application {

public static Gson gson;

public static Config config;

public static File currentDir;

public static boolean time;

public static boolean debug;

public static final int REPETITION\_COUNT = 100;

public static final List<String> SUPPORTED\_FILE\_EXTENSIONS = Arrays.asList(".txt", ".docx");

public static boolean experiments;

public static void main(String[] args) throws Exception {

gson = new GsonBuilder().setPrettyPrinting().create();

config = Config.parse(args);

currentDir = new File(System.getProperty("user.dir") + "/sets");

time = false;

debug = false;

experiments = true;

Frame.showForm();

}

public static boolean isSupportableFile(File file) {

for (String s : SUPPORTED\_FILE\_EXTENSIONS) {

if (file.getName().endsWith(s)) {

return true;

}

}

return false;

}

}

Class.java

package diplom.classification;

import com.google.gson.annotations.SerializedName;

import java.util.Map;

public class Class {

public String name;

@SerializedName("terms\_frequency")

public Map<String, Integer> termsFrequency;

@Override

public String toString() {

return "Class{" +

"name='" + name + '\'' +

", termsFrequency=" + termsFrequency +

'}';

}

}

Classification.java

package diplom.classification;

import com.google.gson.reflect.TypeToken;

import diplom.Application;

import diplom.ExperimentsInterface;

import diplom.distance.Distance;

import diplom.nlp.FilteredUnigram;

import diplom.utils.Copy;

import diplom.utils.Reader;

import org.apache.commons.io.FileUtils;

import java.io.File;

import java.io.IOException;

import java.nio.charset.Charset;

import java.util.\*;

import java.util.concurrent.atomic.AtomicReference;

import java.util.function.Function;

import java.util.stream.Collectors;

import static diplom.utils.Copy.SEPARATOR;

public class Classification implements ExperimentsInterface {

private static final String OTHER\_CLASS\_NAME = "Прочее (!)";

private Distance distance;

private Map<String, double[]> dataMap;

private List<String> vocabulary;

private int CLASSES\_END;

private StringBuilder builder;

private double initTime;

private double workTime;

public Classification(List<Class> classes, Distance distance) {

initTime = System.nanoTime();

this.distance = distance;

this.dataMap = new LinkedHashMap<>();

this.CLASSES\_END = classes.size() + 1;

this.builder = new StringBuilder();

vocabulary = new ArrayList<>();

for (Class c : classes) {

String text = String.join(" ", c.termsFrequency.keySet());

vocabulary.addAll(FilteredUnigram.get(text));

}

vocabulary = vocabulary.stream()

.distinct()

.sorted()

.collect(Collectors.toList());

System.out.printf("%10s - %s\n", "VOCABULARY", vocabulary);

if (Application.debug)

builder.append(String.format("%10s - %s\n", "VOCABULARY", vocabulary));

for (Class c : classes) {

double[] dataRow = new double[vocabulary.size()];

Arrays.fill(dataRow, 0.0);

for (Map.Entry<String, Integer> entry : c.termsFrequency.entrySet()) {

List<String> terms = FilteredUnigram.get(entry.getKey());

for (String term : terms) {

int index = vocabulary.indexOf(term);

dataRow[index] = entry.getValue();

}

}

dataMap.put(c.name, dataRow);

}

double[] dataRow = new double[vocabulary.size()];

dataMap.put(OTHER\_CLASS\_NAME, dataRow);

initTime = (System.nanoTime() - initTime) / 1000000;

System.out.printf("ВРЕМЯ ИНИЦИАЛИЗАЦИИ (classify): %.2f мс\n", initTime);

}

public static List<Class> parseClasses(File classesFile) throws IOException {

String json = FileUtils.readFileToString(classesFile, Charset.defaultCharset());

return Application.gson.fromJson(json, new TypeToken<List<Class>>() {

}.getType());

}

public Map<String, List<String>> run(File[] path) throws IOException {

workTime = System.nanoTime();

List<File> rawFiles = Arrays.stream(path).collect(Collectors.toList());

addFilesToMatrix(rawFiles);

normalize();

Map<String, List<String>> result = new TreeMap<>();

List<String> otherFiles = new ArrayList<>();

dataMap.entrySet().stream().skip(CLASSES\_END).forEach(fileRow -> {

AtomicReference<String> className = new AtomicReference<>("");

AtomicReference<Double> dist = new AtomicReference<>((double) 0);

AtomicReference<String> fileName = new AtomicReference<>("");

dataMap.entrySet().stream().limit(CLASSES\_END).forEach(classRow -> {

double d = distance.calc(fileRow.getValue(), classRow.getValue());

if (d > dist.get()) {

className.set(classRow.getKey());

dist.set(d);

fileName.set(fileRow.getKey());

}

});

if (dist.get() == 0) {

otherFiles.add(fileRow.getKey());

} else {

String cName = className.get();

if (!result.containsKey(cName))

result.put(cName, new ArrayList<String>() {{

add(fileName.get());

}});

else

result.get(className.get()).add(fileName.get());

}

});

result.put(OTHER\_CLASS\_NAME, otherFiles);

if (Application.debug)

builder.append("\nКЛАССИФИЦИРОВАННЫЕ ФАЙЛЫ:\n");

else

builder.append("КЛАССИФИЦИРОВАННЫЕ ФАЙЛЫ:\n");

for (Map.Entry<String, List<String>> entry : result.entrySet()) {

System.out.printf("%10s - %s\n", entry.getKey(), entry.getValue());

builder.append(String.format("%s - %s\n", entry.getKey(), entry.getValue()));

}

String dir = "sets" + SEPARATOR + "classification";

Copy.group(dir, path, result);

workTime = (System.nanoTime() - workTime) / 1000000;

System.out.printf("ВРЕМЯ РАБОТЫ (classify): %.2f мс\n", workTime);

if (Application.time)

builder.append("\nВРЕМЯ РАБОТЫ: ").append(String.format("%.2f мс", workTime));

return result;

}

private void addFilesToMatrix(List<File> files) throws IOException {

for (File f : files) {

String text = Reader.readFile(f);

List<String> strings = FilteredUnigram.get(text);

double[] dataRow = new double[vocabulary.size()];

for (Map.Entry<String, Long> term : strings.stream()

.collect(Collectors.groupingBy(Function.identity(), Collectors.counting()))

.entrySet()) {

int index = vocabulary.indexOf(term.getKey());

if (index > 0) // на случай, если в словаре нет такого термина

dataRow[index] = term.getValue();

}

dataMap.put(f.getName(), dataRow);

}

System.out.println("\nЧАСТОТЫ:");

if (Application.debug)

builder.append("\nЧАСТОТЫ:\n");

for (Map.Entry<String, double[]> e : dataMap.entrySet()) {

System.out.printf("%10s - %s\n", e.getKey(), Arrays.toString(e.getValue()));

if (Application.debug)

builder.append(String.format("%10s - %s\n", e.getKey(), Arrays.toString(e.getValue())));

}

}

private void normalize() {

for (Map.Entry<String, double[]> row : dataMap.entrySet()) {

double[] items = row.getValue();

double min = Arrays.stream(items).min().getAsDouble();

double max = Arrays.stream(items).max().getAsDouble();

for (int i = 0; i < items.length; i++) {

items[i] = (items[i] - min) / (max - min);

if (Double.isNaN(items[i])) {

items[i] = 0;

}

}

dataMap.put(row.getKey(), items);

}

System.out.println("\nНОРМАЛИЗОВАННЫЕ ДАННЫЕ:");

if (Application.debug)

builder.append("\nНОРМАЛИЗОВАННЫЕ ДАННЫЕ:\n");

for (Map.Entry<String, double[]> e : dataMap.entrySet()) {

System.out.printf("%10s - %s\n", e.getKey(), Arrays.toString(e.getValue()));

if (Application.debug)

builder.append(String.format("%10s - %s\n", e.getKey(), Arrays.toString(e.getValue())));

}

}

public String getBuilder() {

return builder.toString();

}

public double getInitTime() {

return initTime;

}

public double getWorkTime() {

return workTime;

}

}

Clustering.java

package diplom.clustering;

import diplom.Application;

import diplom.ExperimentsInterface;

import diplom.distance.Distance;

import diplom.utils.Copy;

import java.io.File;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Map;

import java.util.stream.Collectors;

import static diplom.utils.Copy.SEPARATOR;

public class Clustering implements ExperimentsInterface {

private double separateValue;

private Distance distance;

private Vocabulary vocabulary;

private List<Document> documents;

private List<String> docNames;

private double[][] similarityMatrix;

private StringBuilder builder;

private double initTime;

private double workTime;

public Clustering(double separateValue, Distance distance) {

initTime = System.nanoTime();

this.separateValue = separateValue;

this.distance = distance;

this.builder = new StringBuilder();

this.vocabulary = new Vocabulary();

this.docNames = new ArrayList<>();

this.documents = new ArrayList<>();

initTime = (System.nanoTime() - initTime) / 1000000;

System.out.printf("ВРЕМЯ ИНИЦИАЛИЗАЦИИ (cluster): %.2f мс\n", initTime);

}

public Map<String, List<String>> run(File[] files) throws Exception {

workTime = System.nanoTime();

builder.setLength(0);

if (files == null || files.length == 0) {

throw new Exception("docs == null || docs.length == 0");

}

for (File file : files) {

documents.add(new Document(file, vocabulary));

docNames.add(file.getName());

}

this.similarityMatrix = new double[files.length][files.length];

documents.forEach(Document::calcTermsFrequency);

initSimMatrix();

List<Prim.Edge> edges = Prim.solve(similarityMatrix);

if (Application.debug)

builder.append(docNames).append("\n");

System.out.println(docNames + "\n");

for (double[] matrix : similarityMatrix) {

for (int j = 0; j < similarityMatrix.length; j++) {

System.out.printf("%.2f ", matrix[j]);

if (Application.debug)

builder.append(String.format("%.2f ", matrix[j]));

}

if (Application.debug)

builder.append("\n");

System.out.println();

}

if (Application.debug) {

builder.append("\n");

edges.forEach(e -> builder.append(e.toString()).append("\n"));

builder.append("\n");

}

System.out.println();

edges.forEach(System.out::println);

System.out.println();

List<List<Integer>> clusters = new ArrayList<>();

clusters.add(new ArrayList<>(Collections.singletonList(edges.get(0).s)));

for (Prim.Edge edge : edges) {

if (edge.weigh >= separateValue) {

clusters.add(new ArrayList<>(Collections.singletonList(edge.t)));

} else {

int clusterID = relevantCluster(clusters, edge);

clusters.get(clusterID).add(edge.t);

}

}

// List<List<String>> collect = clusters.stream()

// .map(docIDs -> docIDs.stream()

// .map(docID -> docNames.get(docID))

// .collect(Collectors.toList()))

// .collect(Collectors.toList());

Map<String, List<String>> collect = clusters.stream()

.map(docIDs -> docIDs.stream()

.map(docID -> docNames.get(docID))

.collect(Collectors.toList()))

.collect(Collectors.toMap(

this::getDocsKeyWords,

i -> i

));

int i = 1;

for (Map.Entry<String, List<String>> entry : collect.entrySet()) {

// builder.append(String.format("Группа №%d: %s\n", i++, entry.getKey()));

builder.append(String.format("Группа №%d:\n", i++));

builder.append(String.format("Файлы: %s\n\n", entry.getValue()));

// builder.append(String.format("Файлы, принадлежащие группе №%d:\n%s\n\n", i++, entry.getValue()));

}

String dir = "sets" + SEPARATOR + "clustering";

Copy.group(dir, files, collect, true);

workTime = (System.nanoTime() - workTime) / 1000000;

System.out.printf("ВРЕМЯ РАБОТЫ (cluster): %.2f мс\n", workTime);

if (Application.time)

builder.append("ВРЕМЯ РАБОТЫ: ").append(String.format("%.2f мс", workTime));

return collect;

}

public String getBuilder() {

return builder.toString();

}

private int relevantCluster(List<List<Integer>> clusters, Prim.Edge edge) {

for (int i = 0; i < clusters.size(); i++) {

if (clusters.get(i).contains(edge.s))

return i;

}

return -1;

}

private void initSimMatrix() {

for (int i = 0; i < documents.size(); i++) {

for (int j = 0; j < documents.size(); j++) {

similarityMatrix[i][j] = 1 - distance.calc(tfidf(documents.get(i)), tfidf(documents.get(j)));

}

}

}

private double[] tfidf(Document doc) {

double[] tf = tf(doc);

double[] idf = idf(doc);

double[] tfidf = new double[vocabulary.size()];

for (int i = 0; i < vocabulary.size(); i++) {

tfidf[i] = tf[i] \* idf[i];

}

return tfidf;

}

private double[] tf(Document doc) {

double[] tf = new double[vocabulary.size()];

double termsCount = doc.termsCount();

doc.termsFrequencyMap().forEach((k, v) -> tf[k] = v / termsCount);

return tf;

}

private double[] idf(Document doc) {

double[] idf = new double[vocabulary.size()];

doc.termsFrequencyMap().forEach((k, v) -> idf[k] = Math.log(documents.size() / (double) docsWithWord(k)));

return idf;

}

private int docsWithWord(int termID) {

// int count = 0;

// for (Document doc : documents) {

// if (doc.containsWord(termID))

// count++;

// }

// return count;

return (int) documents.stream()

.filter(d -> d.containsWord(termID))

.count();

}

private String getDocsKeyWords(List<String> docsName) {

return docsName.stream()

.limit(3)

.map(name -> documents.get(docNames.indexOf(name))

.getTheHardestWord())

.distinct()

.collect(Collectors.joining(", "));

}

public double getInitTime() {

return initTime;

}

public double getWorkTime() {

return workTime;

}

}

Document.java

package diplom.clustering;

import diplom.nlp.FilteredUnigram;

import diplom.utils.Reader;

import java.io.File;

import java.io.IOException;

import java.util.List;

import java.util.Map;

import java.util.TreeMap;

public class Document {

private Vocabulary vocabulary;

private List<String> terms;

private Map<Integer, Integer> termsFrequency;

private int termWithMaxFrequency;

public Document(File doc, Vocabulary vocabulary) {

this.vocabulary = vocabulary;

this.termsFrequency = new TreeMap<>();

this.termWithMaxFrequency = 0;

try {

String text = Reader.readFile(doc);

this.terms = FilteredUnigram.get(text);

vocabulary.addAll(terms);

} catch (IOException e) {

e.printStackTrace();

}

}

public void calcTermsFrequency() {

for (String term : vocabulary.keySet()) {

termsFrequency.put(vocabulary.get(term), 0);

}

int termID;

for (String term : terms) {

termID = vocabulary.get(term);

Integer termsFreq = termsFrequency.get(termID);

termsFrequency.put(termID, termsFreq + 1);

if (termsFrequency.get(termWithMaxFrequency) < termsFreq) {

termWithMaxFrequency = termID;

}

}

}

public int termsCount() {

return terms.size();

}

public Map<Integer, Integer> termsFrequencyMap() {

return termsFrequency;

}

public boolean containsWord(int termID) {

return termsFrequency.get(termID) > 0;

}

public String getTheHardestWord() {

return terms.get(termsFrequency.get(termWithMaxFrequency));

}

}

Prim.java

package diplom.clustering;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class Prim {

public static List<Edge> solve(double[][] simMatrix) {

List<Edge> edges = new ArrayList<>();

int no\_edge = 0;

boolean[] selected = new boolean[simMatrix.length];

Arrays.fill(selected, false);

selected[0] = true;

int x, y;

double min;

while (no\_edge < simMatrix.length - 1) {

x = y = 0;

min = 1\_000\_000;

for (int i = 0; i < simMatrix.length; i++) {

if (selected[i]) {

for (int j = 0; j < simMatrix.length; j++) {

if (!selected[j] && simMatrix[i][j] != 0) {

if (min > simMatrix[i][j]) {

min = simMatrix[i][j];

x = i;

y = j;

}

}

}

}

}

edges.add(new Edge(x, y, simMatrix[x][y]));

selected[y] = true;

no\_edge++;

}

return edges;

}

public static class Edge {

int s, t;

double weigh;

public Edge(int s, int t, double weigh) {

this.s = s;

this.t = t;

this.weigh = weigh;

}

@Override

public String toString() {

return (s + 1) + " - " + (t + 1) + " : " + String.format("%.2f", weigh);

}

}

}

Vocabulary.java

package diplom.clustering;

import java.util.List;

import java.util.TreeMap;

public class Vocabulary extends TreeMap<String, Integer> {

private int nextIndex = 0;

public void add(String word) {

if (!this.containsKey(word)) {

this.put(word, nextIndex++);

}

}

public void addAll(List<String> words) {

for (String word : words) {

this.add(word);

}

}

}

Config.java

package diplom.config;

import com.google.gson.annotations.SerializedName;

import diplom.Application;

import fi.purkka.jarpa.JarpaArgs;

import fi.purkka.jarpa.JarpaParser;

import org.apache.commons.io.FileUtils;

import java.io.File;

import java.io.IOException;

import java.nio.charset.Charset;

import java.util.Arrays;

import static fi.purkka.jarpa.JarpaArg.string;

public class Config {

public Clustering clustering;

public Classification classification;

public static Config parse(String[] args) throws IOException {

JarpaArgs jargs = JarpaParser.parsing(args).parse();

String configPath = jargs.get(string("-config").optional()).orElse("config.json");

File configFile = new File(configPath);

String json = FileUtils.readFileToString(configFile, Charset.defaultCharset());

Config config = Application.gson.fromJson(json, Config.class);

config.clustering.validate();

config.classification.validate();

return config;

}

public static class Clustering {

@SerializedName("separate\_value")

public double separateValue;

@SerializedName("similarity\_measure")

public String similarityMeasure;

void validate() {

if (separateValue < 0.1 || 1.0 < separateValue)

throw new InvalidConfigValueException("invalid separate value");

if (!Arrays.asList("cos", "euclid").contains(similarityMeasure))

throw new InvalidConfigValueException("invalid similarity measure");

}

}

public static class Classification {

@SerializedName("separate\_value")

public double separateValue;

@SerializedName("similarity\_measure")

public String similarityMeasure;

void validate() {

if (separateValue < 0.1 || 1.0 < separateValue)

throw new InvalidConfigValueException("invalid separate value");

if (!Arrays.asList("cos", "euclid").contains(similarityMeasure))

throw new InvalidConfigValueException("invalid similarity measure");

}

}

public static class InvalidConfigValueException extends RuntimeException {

InvalidConfigValueException(String errorMessage) {

super(errorMessage);

}

}

}

CosineDistance.java

package diplom.distance;

public class CosineDistance implements Distance {

public double calc(double[] vector1, double[] vector2) {

double sum1 = 0, sum2 = 0, sum3 = 0;

for (int i = 0; i < vector1.length; i++) {

sum1 += vector1[i] \* vector2[i];

sum2 += vector1[i] \* vector1[i];

sum3 += vector2[i] \* vector2[i];

}

return sum1 / (Math.sqrt(sum2) \* Math.sqrt(sum3));

}

}

Distance.java

package diplom.distance;

public interface Distance {

static Distance distance(String similarityMeasure) {

switch (similarityMeasure) {

case "cos":

return new CosineDistance();

case "euclid":

return new EuclideanDistance();

}

return new CosineDistance();

}

double calc(double[] vector1, double[] vector2);

}

EuclideanDistance.java

package diplom.distance;

public class EuclideanDistance implements Distance {

public double calc(double[] vector1, double[] vector2) {

double sum = 0;

for (int i = 0; i < vector1.length; i++) {

sum += Math.pow(vector1[i] - vector2[i], 2);

}

return Math.sqrt(sum);

}

}

ExperimentsInterface.java

package diplom;

import java.io.File;

import java.util.List;

import java.util.Map;

public interface ExperimentsInterface {

double getInitTime();

double getWorkTime();

Map<String, List<String>> run(File[] selectedFiles) throws Exception;

String getBuilder();

}

Frame.java

package diplom.gui;

import com.alee.laf.WebLookAndFeel;

import com.alee.laf.window.WebFrame;

import diplom.gui.menu.Menu;

import javax.swing.\*;

import java.awt.event.ComponentAdapter;

import java.awt.event.ComponentEvent;

public class Frame extends WebFrame {

public static final int FONT\_SIZE = 13;

public static final int X\_OFFSET = 16;

public static final int Y\_OFFSET = 39;

private Frame() {

setTitle("Дипломная работа");

setLayout(null);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setSize(760, 700 + Menu.MENU\_Y\_OFFSET);

setLocationRelativeTo(null);

// setExtendedState(MAXIMIZED\_BOTH);

Menu menu = new Menu(this);

add(menu);

Panel panel = new Panel(this);

add(panel);

addComponentListener(new ComponentAdapter() {

public void componentResized(ComponentEvent evt) {

// panel.setSize(getWidth() - X\_OFFSET, getHeight() - Y\_OFFSET);

}

});

setVisible(true);

// new CreateClassesGroupDialog(this);

// new EditClassesGroupDialog(this);

}

public static void showForm() {

SwingUtilities.invokeLater(() -> {

WebLookAndFeel.install();

new Frame();

});

}

}

AboutDialog.java

package diplom.gui.menu;

import com.alee.laf.label.WebLabel;

import com.alee.laf.window.WebDialog;

import com.alee.laf.window.WebFrame;

import diplom.gui.Frame;

import javax.swing.\*;

import java.awt.\*;

public class AboutDialog extends WebDialog {

public AboutDialog(WebFrame frame) {

setTitle("О програаме");

setSize(100, 50);

setLocationRelativeTo(frame);

setDefaultCloseOperation(WindowConstants.DISPOSE\_ON\_CLOSE);

setLayout(new BorderLayout());

WebLabel groupLabel = new WebLabel("Сборка: 1.5.0");

groupLabel.setFontSize(Frame.FONT\_SIZE);

groupLabel.setMargin(25, 50, 25, 50);

groupLabel.setHorizontalAlignment(WebLabel.CENTER);

groupLabel.setVerticalAlignment(WebLabel.CENTER);

add(groupLabel);

pack();

setVisible(true);

}

}

ClassGroup.java

package diplom.gui.menu;

import com.alee.laf.button.WebButton;

import com.alee.laf.label.WebLabel;

import com.alee.laf.panel.WebPanel;

import com.alee.laf.text.WebTextField;

import com.alee.laf.window.WebDialog;

import diplom.classification.Class;

import diplom.gui.Frame;

import diplom.gui.Utils;

import javax.swing.\*;

import java.util.\*;

import java.util.function.Consumer;

public class ClassGroup extends WebPanel {

private final int WIDTH = 395;

private final int startHeight = 35;

private final int ROW\_OFFSET = 35;

private int height = 72;

private List<ClassRow> classRows = new ArrayList<>();

private int index = 0;

int rowIndex = 0;

private WebDialog dialog;

private Consumer<Integer> groupDeleteHandler;

private WebTextField classField;

private Consumer<Integer> rowDeleteHandler = index -> {

if (classRows.size() == 1) return;

int idx = index;

this.index--;

height = startHeight;

classRows.get(idx).hi();

classRows.remove(idx);

for (int i = 0; i < classRows.size(); i++) {

classRows.get(i).setIndex(i);

classRows.get(i).setY(height);

height += ROW\_OFFSET;

}

setSize(WIDTH, height);

setPreferredSize(WIDTH, height);

};

public ClassGroup(WebDialog dialog, int index, Consumer<Integer> groupDeleteHandler) {

this(dialog, index, groupDeleteHandler, null);

}

public ClassGroup(WebDialog dialog, int index, Consumer<Integer> groupDeleteHandler, Class cl) {

this.index = index;

this.dialog = dialog;

this.groupDeleteHandler = groupDeleteHandler;

setLayout(null);

// setBackground(Color.PINK);

setPreferredSize(WIDTH, height);

WebLabel classLabel = new WebLabel("Название класса");

classLabel.setBounds(0, 5, 120, 30);

classLabel.setPadding(0, 5, 0, 0);

classLabel.setFontSize(Frame.FONT\_SIZE);

add(classLabel);

classField = new WebTextField();

if (cl != null) {

classField.setText(cl.name);

}

classField.setBounds(120, 5, 200, 30);

classField.setInputPrompt("Название...");

classField.setPadding(0, 5, 0, 0);

add(classField);

WebButton add = new WebButton(new ImageIcon(Objects.requireNonNull(Utils.loadImage(ClassGroup.class, "icons/new.png"))));

add.setBounds(325, 5, 30, 30);

add.addActionListener(e -> addRow(null, null));

add(add);

WebButton remove = new WebButton(new ImageIcon(Objects.requireNonNull(Utils.loadImage(ClassGroup.class, "icons/remove.png"))));

remove.setBounds(360, 5, 30, 30);

remove.addActionListener(e -> groupDeleteHandler.accept(this.index));

add(remove);

if (cl == null) {

ClassRow row = new ClassRow(rowIndex++, rowDeleteHandler);

row.setLocation(0, startHeight);

add(row);

classRows.add(row);

} else {

List<Map.Entry<String, Integer>> list = new ArrayList<>(cl.termsFrequency.entrySet());

ClassRow row = new ClassRow(rowIndex++, rowDeleteHandler, list.get(0).getKey(), list.get(0).getValue() + "");

row.setLocation(0, startHeight);

add(row);

classRows.add(row);

for (int i = 1; i < list.size(); i++) {

addRow(list.get(i).getKey(), list.get(i).getValue() + "");

}

}

}

private void addRow(String k, String v) {

ClassRow row = new ClassRow(rowIndex++, rowDeleteHandler, k, v);

row.setLocation(0, height);

add(row);

classRows.add(row);

height += ROW\_OFFSET;

setSize(WIDTH, height);

setPreferredSize(WIDTH, height);

}

public void setIndex(int index) {

this.index = index;

}

public void hi() {

setVisible(false);

}

public Class getData() {

Class cl = new Class();

cl.name = classField.getText();

cl.termsFrequency = new LinkedHashMap<>();

for (ClassRow row : classRows) {

cl.termsFrequency.put(row.getWord(), row.getCount());

}

return cl;

}

}

ClassRow.java

package diplom.gui.menu;

import com.alee.laf.button.WebButton;

import com.alee.laf.panel.WebPanel;

import com.alee.laf.text.WebTextField;

import diplom.gui.Panel;

import diplom.gui.Utils;

import javax.swing.\*;

import java.util.Objects;

import java.util.function.Consumer;

public class ClassRow extends WebPanel {

private static final int HEIGHT = 40;

private WebTextField word;

private WebTextField count;

private int index;

public ClassRow(int index, Consumer<Integer> consumer) {

this(index, consumer, null, null);

}

public ClassRow(int index, Consumer<Integer> consumer, String key, String val) {

this.index = index;

setLayout(null);

setSize(CreateClassesGroupDialog.dialogWidth, HEIGHT);

word = new WebTextField();

if (key != null) {

word.setText(key);

}

word.setBounds(0, 5, 165, 30);

word.setInputPrompt("Ключевое слово #" + (index + 1));

word.setPadding(0, 5, 0, 0);

add(word);

count = new WebTextField();

if (val != null) {

count.setText(val);

}

count.setInputPrompt("Кол-во повторений");

count.setBounds(169, 5, 115, 30);

count.setPadding(0, 5, 0, 0);

add(count);

WebButton remove = new WebButton(new ImageIcon(Objects.requireNonNull(Utils.loadImage(Panel.class, "icons/remove.png"))));

remove.setBounds(288, 5, 30, 30);

remove.addActionListener(e -> consumer.accept(this.index));

add(remove);

}

public String getWord() {

return word.getText();

}

public int getCount() {

return Integer.parseInt(count.getText());

}

public int getIndex() {

return index;

}

public void setIndex(int index) {

this.index = index;

word.setInputPrompt("Ключевое слово #" + (index + 1));

}

public void hi() {

setVisible(false);

}

public void setY(int y) {

setLocation(getX(), y);

}

}

CreateClassesGroupDialog.java

package diplom.gui.menu;

import com.alee.laf.button.WebButton;

import com.alee.laf.filechooser.WebFileChooser;

import com.alee.laf.label.WebLabel;

import com.alee.laf.panel.WebPanel;

import com.alee.laf.scroll.WebScrollPane;

import com.alee.laf.text.WebTextField;

import com.alee.laf.window.WebDialog;

import com.alee.laf.window.WebFrame;

import com.alee.managers.style.StyleId;

import diplom.Application;

import diplom.classification.Class;

import diplom.gui.Frame;

import diplom.gui.Utils;

import org.apache.commons.io.FileUtils;

import javax.swing.\*;

import java.io.File;

import java.io.IOException;

import java.nio.charset.Charset;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

import java.util.function.Consumer;

import java.util.stream.Collectors;

public class CreateClassesGroupDialog extends WebDialog {

static final int dialogWidth = 500;

private final int dialogHeight = 500;

private final int startY = 145;

private int currentY = 145;

private WebButton newClassBtn;

private WebPanel classesPanel;

private List<ClassGroup> classGroups = new ArrayList<>();

private Consumer<Integer> groupDeleteHandler = index -> {

if (classGroups.size() == 1) return;

int idx = index;

System.out.println(idx);

ClassGroup classGroup = classGroups.get(idx);

classesPanel.remove(classGroup);

classesPanel.revalidate();

classesPanel.repaint();

for (int i = 0; i < classGroups.size(); i++) {

classGroups.get(i).setIndex(i);

}

this.revalidate();

};

public CreateClassesGroupDialog(WebFrame frame) {

super(frame, true);

setTitle("Создание группы классов");

setSize(dialogWidth, dialogHeight + 20);

setLocationRelativeTo(frame);

setDefaultCloseOperation(WindowConstants.DISPOSE\_ON\_CLOSE);

setLayout(null);

WebLabel groupLabel = new WebLabel("Название группы");

groupLabel.setBounds(15, 5, 120, 30);

groupLabel.setFontSize(Frame.FONT\_SIZE);

// add(groupLabel);

WebTextField groupField = new WebTextField();

groupField.setInputPrompt("Название группы...");

groupField.setBounds(130, 7, 300, 28);

groupField.setPadding(0, 5, 0, 0);

// add(groupField);

WebButton add = new WebButton(new ImageIcon(Objects.requireNonNull(Utils.loadImage(ClassGroup.class, "icons/new.png"))));

add.setBounds(175, 7, 30, 30);

add.addActionListener(e -> {

WebFileChooser chooser = new WebFileChooser();

chooser.setCurrentDirectory(Application.currentDir);

int result = chooser.showSaveDialog(CreateClassesGroupDialog.this);

if (result == WebFileChooser.APPROVE\_OPTION) {

File selectedFile = chooser.getSelectedFile();

String json = Application.gson.toJson(getClasses());

try {

FileUtils.write(selectedFile, json, Charset.defaultCharset());

} catch (IOException ex) {

ex.printStackTrace();

}

}

});

add(add);

newClassBtn = new WebButton("Добавить класс");

newClassBtn.setBounds(15, 7, 150, 30);

add(newClassBtn);

classesPanel = new WebPanel(new VerticalLayout());

// classesPanel.setBackground(Color.PINK);

ClassGroup classGroup = new ClassGroup(this, classGroups.size(), groupDeleteHandler);

classesPanel.add(classGroup);

classGroups.add(classGroup);

WebScrollPane scrollPane = new WebScrollPane(StyleId.scrollpaneTransparent, classesPanel);

scrollPane.setBounds(10, 39, dialogWidth - 5, dialogHeight - 69);

scrollPane.getVerticalScrollBar().setUnitIncrement(8);

scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS);

scrollPane.revalidate();

add(scrollPane);

newClassBtn.addActionListener(e -> {

ClassGroup newGroup = new ClassGroup(this, classGroups.size(), groupDeleteHandler);

classesPanel.add(newGroup);

classGroups.add(newGroup);

classesPanel.revalidate();

classesPanel.repaint();

});

setVisible(true);

}

private List<Class> getClasses() {

return classGroups.stream()

.map(ClassGroup::getData)

.collect(Collectors.toList());

}

}

EditClassesGroupDialog.java

package diplom.gui.menu;

import com.alee.laf.button.WebButton;

import com.alee.laf.filechooser.WebFileChooser;

import com.alee.laf.label.WebLabel;

import com.alee.laf.optionpane.WebOptionPane;

import com.alee.laf.panel.WebPanel;

import com.alee.laf.scroll.WebScrollPane;

import com.alee.laf.text.WebTextField;

import com.alee.laf.window.WebDialog;

import com.alee.laf.window.WebFrame;

import com.alee.managers.style.StyleId;

import com.google.gson.reflect.TypeToken;

import diplom.Application;

import diplom.classification.Class;

import diplom.gui.Frame;

import diplom.gui.Utils;

import org.apache.commons.io.FileUtils;

import javax.swing.\*;

import java.io.File;

import java.io.IOException;

import java.nio.charset.Charset;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

import java.util.function.Consumer;

import java.util.stream.Collectors;

public class EditClassesGroupDialog extends WebDialog {

static final int dialogWidth = 500;

private final int dialogHeight = 500;

private final int startY = 145;

private List<Class> classes;

private int currentY = 145;

private WebButton newClassBtn;

private WebPanel classesPanel;

private List<ClassGroup> classGroups = new ArrayList<>();

private Consumer<Integer> groupDeleteHandler = index -> {

if (classGroups.size() == 1) return;

int idx = index;

ClassGroup classGroup = classGroups.get(idx);

classesPanel.remove(classGroup);

classesPanel.revalidate();

classesPanel.repaint();

classGroups.remove(idx);

for (int i = 0; i < classGroups.size(); i++) {

classGroups.get(i).setIndex(i);

}

this.revalidate();

};

public EditClassesGroupDialog(WebFrame frame) {

WebFileChooser chooser = new WebFileChooser();

chooser.setCurrentDirectory(Application.currentDir);

File selectedFile = null;

int result = chooser.showOpenDialog(this);

if (result == WebFileChooser.APPROVE\_OPTION) {

try {

selectedFile = chooser.getSelectedFile();

String json = FileUtils.readFileToString(selectedFile, Charset.defaultCharset());

classes = Application.gson.fromJson(json, new TypeToken<List<Class>>() {

}.getType());

} catch (IOException e) {

e.printStackTrace();

}

} else {

WebOptionPane.showMessageDialog(

this,

"Вы не выбрали файл для редактирования",

"Ошибка",

WebOptionPane.ERROR\_MESSAGE

);

}

if (selectedFile == null) {

setVisible(false);

}

setTitle("Редактирование группы классов");

setSize(dialogWidth, dialogHeight + 20);

setLocationRelativeTo(frame);

setDefaultCloseOperation(WindowConstants.DISPOSE\_ON\_CLOSE);

setModal(true);

setLayout(null);

WebLabel groupLabel = new WebLabel("Название группы");

groupLabel.setBounds(15, 5, 120, 30);

groupLabel.setFontSize(Frame.FONT\_SIZE);

// add(groupLabel);

WebTextField groupField = new WebTextField();

groupField.setText(selectedFile.getName());

groupField.setInputPrompt("Название группы...");

groupField.setBounds(130, 7, 300, 28);

groupField.setPadding(0, 5, 0, 0);

// add(groupField);

WebButton save = new WebButton(new ImageIcon(Objects.requireNonNull(Utils.loadImage(EditClassesGroupDialog.class, "icons/edit.png"))));

save.setBounds(175, 7, 30, 30);

File finalSelectedFile = selectedFile;

save.addActionListener(e -> {

String json = Application.gson.toJson(getClasses());

try {

FileUtils.write(finalSelectedFile, json, Charset.defaultCharset());

} catch (IOException ex) {

ex.printStackTrace();

}

setVisible(false);

});

add(save);

newClassBtn = new WebButton("Добавить класс");

newClassBtn.setBounds(15, 7, 150, 30);

add(newClassBtn);

classesPanel = new WebPanel(new VerticalLayout());

// classesPanel.setBackground(Color.PINK);

for (Class cl : classes) {

ClassGroup classGroup = new ClassGroup(this, classGroups.size(), groupDeleteHandler, cl);

classesPanel.add(classGroup);

classGroups.add(classGroup);

}

WebScrollPane scrollPane = new WebScrollPane(StyleId.scrollpaneTransparent, classesPanel);

scrollPane.setBounds(10, 39, dialogWidth - 5, dialogHeight - 69);

scrollPane.getVerticalScrollBar().setUnitIncrement(8);

scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS);

scrollPane.revalidate();

add(scrollPane);

newClassBtn.addActionListener(e -> {

ClassGroup newGroup = new ClassGroup(this, classGroups.size(), groupDeleteHandler);

classesPanel.add(newGroup);

classGroups.add(newGroup);

classesPanel.revalidate();

classesPanel.repaint();

});

setVisible(true);

}

private List<Class> getClasses() {

return classGroups.stream()

.map(ClassGroup::getData)

.collect(Collectors.toList());

}

}

Menu.java

package diplom.gui.menu;

import com.alee.demo.content.menu.MenusGroup;

import com.alee.laf.window.WebFrame;

import com.alee.managers.hotkey.Hotkey;

import com.alee.utils.swing.menu.JMenuBarGenerator;

import com.alee.utils.swing.menu.MenuGenerator;

import diplom.Application;

import diplom.gui.Frame;

import javax.swing.\*;

import java.awt.event.ActionListener;

public class Menu extends JMenuBar {

public static final int MENU\_Y\_OFFSET = 25;

private static final String CLASSES\_NEW = "Создать";

private static final String CLASSES\_EDIT = "Редактировать";

private static final String EXIT = "Выход";

private static final String ABOUT = "О программе";

private static final String DEBUG = "Режим отладки";

private static final String TIME = "Время работы";

public Menu(WebFrame frame) {

setBounds(0, 0, frame.getWidth() - Frame.X\_OFFSET, MENU\_Y\_OFFSET);

ActionListener action = e -> {

JMenuItem item = (JMenuItem) e.getSource();

String text = item.getText();

System.out.println("menu command: " + text);

switch (text) {

case CLASSES\_NEW:

new CreateClassesGroupDialog(frame);

break;

case CLASSES\_EDIT:

new EditClassesGroupDialog(frame);

break;

case ABOUT:

new AboutDialog(frame);

break;

case EXIT:

System.exit(0);

break;

case DEBUG:

Application.debug = item.isSelected();

break;

case TIME:

Application.time = item.isSelected();

break;

}

};

JMenuBarGenerator generator = new JMenuBarGenerator(this);

generator.setLanguagePrefix("demo.example.menus.menu");

generator.setIconSettings(MenusGroup.class, "icons/menu/", "png");

MenuGenerator fileMenu = generator.addSubMenu("Файл");

MenuGenerator classes = fileMenu.addSubMenu("Группа классов");

classes.addItem("new", CLASSES\_NEW, Hotkey.CTRL\_C, action);

classes.addItem("edit", CLASSES\_EDIT, Hotkey.CTRL\_E, action);

fileMenu.addItem("exit", EXIT, Hotkey.ALT\_X, action);

MenuGenerator view = generator.addSubMenu("Вид");

view.addCheckItem(TIME, Hotkey.CTRL\_T, false, action);

view.addCheckItem(DEBUG, Hotkey.CTRL\_D, false, action);

MenuGenerator helpMenu = generator.addSubMenu("Справка");

helpMenu.addItem(ABOUT, Hotkey.CTRL\_H, action);

}

}

VerticalLayout.java

package diplom.gui.menu;

import com.alee.laf.button.WebButton;

import java.awt.\*;

class VerticalLayout implements LayoutManager {

private int leftPadding = 0;

private int topPadding = 25;

private int offset = 30;

private Dimension size = new Dimension();

// Метод определения минимального размера для контейнера

public Dimension minimumLayoutSize(Container c) {

return calculateBestSize(c);

}

// Метод определения предпочтительного размера для контейнера

public Dimension preferredLayoutSize(Container c) {

return calculateBestSize(c);

}

// Метод расположения компонентов в контейнере

public void layoutContainer(Container container) {

// Список компонентов

Component[] list = container.getComponents();

int ii = -1;

for (int i = 0; i < list.length; i++) {

if (list[i] instanceof WebButton) {

ii = i;

break;

}

}

int currentY = topPadding;

for (Component component : list) {

// Определение предпочтительного размера компонента

Dimension pref = component.getPreferredSize();

// Размещение компонента на экране

component.setBounds(leftPadding, currentY, pref.width, pref.height);

// Учитываем промежуток

currentY += offset;

// Смещаем вертикальную позицию компонента

currentY += pref.height;

}

}

// Метод вычисления оптимального размера контейнера

private Dimension calculateBestSize(Container c) {

// Вычисление длины контейнера

Component[] list = c.getComponents();

int width, maxWidth = 0;

for (Component component : list) {

width = component.getWidth();

// Поиск компонента с максимальной длиной

if (width > maxWidth)

maxWidth = width;

}

// Размер контейнера в длину с учетом левого отступа

size.width = maxWidth + leftPadding;

// Вычисление высоты контейнера

int height = 0;

for (Component component : list) {

height += offset;

height += component.getHeight();

}

size.height = height;

return size;

}

// Следующие два метода не используются

public void addLayoutComponent(String name, Component comp) {

}

public void removeLayoutComponent(Component comp) {

}

}

Panel.java

package diplom.gui;

import com.alee.extended.filechooser.WebDirectoryChooser;

import com.alee.extended.filechooser.WebFileDrop;

import com.alee.laf.button.WebButton;

import com.alee.laf.button.WebToggleButton;

import com.alee.laf.grouping.GroupPane;

import com.alee.laf.label.WebLabel;

import com.alee.laf.menu.WebMenuItem;

import com.alee.laf.menu.WebPopupMenu;

import com.alee.laf.panel.WebPanel;

import com.alee.laf.scroll.WebScrollPane;

import com.alee.laf.separator.WebSeparator;

import com.alee.laf.text.WebTextArea;

import com.alee.laf.window.WebFrame;

import com.alee.utils.filefilter.FilesFilter;

import diplom.Application;

import diplom.ExperimentsInterface;

import diplom.classification.Class;

import diplom.classification.Classification;

import diplom.clustering.Clustering;

import diplom.distance.Distance;

import diplom.gui.menu.Menu;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ItemEvent;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.io.File;

import java.util.Arrays;

import java.util.List;

import java.util.Map;

import java.util.Objects;

import java.util.stream.Collectors;

public class Panel extends WebPanel {

private int mode = 0;

private WebButton classesChooser;

private WebTextArea textArea;

private File classes;

private File[] selectedFiles;

public Panel(WebFrame frame) {

setLayout(null);

setBounds(0, Menu.MENU\_Y\_OFFSET, frame.getWidth() - Frame.X\_OFFSET, frame.getHeight() - Frame.Y\_OFFSET);

WebLabel modeLabel = new WebLabel("Режим работы");

modeLabel.setBounds(25, 15, 200, 30);

modeLabel.setFontSize(Frame.FONT\_SIZE);

add(modeLabel);

WebToggleButton cluster = new WebToggleButton("Кластеризация", true);

cluster.setPreferredSize(100, 30);

cluster.addItemListener(e -> {

if (e.getStateChange() == ItemEvent.SELECTED) {

mode = 0;

classesChooser.setVisible(false);

classes = null;

}

});

WebToggleButton classify = new WebToggleButton("Классификация", false);

classify.setPreferredSize(100, 30);

classify.addItemListener(e -> {

if (e.getStateChange() == ItemEvent.SELECTED) {

mode = 1;

classesChooser.setText("Выберите файл c классами");

classesChooser.setVisible(true);

}

});

GroupPane togglePane = new GroupPane(cluster, classify);

togglePane.setBounds(120, 15, 200, 30);

add(togglePane);

classesChooser = new WebButton("Выберите файл c классами");

classesChooser.setBounds(330, 15, 150, 30);

classesChooser.setFocusPainted(false);

classesChooser.setCursor(new Cursor(Cursor.HAND\_CURSOR));

classesChooser.addActionListener(e -> {

JFileChooser chooser = new JFileChooser();

chooser.setMultiSelectionEnabled(false);

chooser.setCurrentDirectory(Application.currentDir);

int result = chooser.showOpenDialog(frame);

if (result == JFileChooser.APPROVE\_OPTION) {

classes = chooser.getSelectedFile();

classesChooser.setText(classes.getName());

}

});

classesChooser.setVisible(false);

add(classesChooser);

WebButton run = new WebButton(new ImageIcon(Objects.requireNonNull(Utils.loadImage(Panel.class, "icons/start.jpg"))));

run.setSize(30, 30);

run.setLocation(getWidth() - 25 - 30, 15);

run.setFocusPainted(false);

run.setCursor(new Cursor(Cursor.HAND\_CURSOR));

run.addActionListener(e -> {

try {

if (selectedFiles == null || selectedFiles.length == 0) {

JOptionPane.showMessageDialog(

frame,

"Вы не выбрали файлы для обработки",

"Ошибка",

JOptionPane.ERROR\_MESSAGE

);

return;

}

if (Panel.this.mode == 1 && classes == null) {

JOptionPane.showMessageDialog(

frame,

"Вы не выбрали файл с классами",

"Ошибка",

JOptionPane.ERROR\_MESSAGE

);

return;

}

runAlgorithm();

////////////////////////////////////////////////////////////////////////////////////////////////////////

if (Application.experiments) {

double workTime = 0;

for (int i = 1; i <= Application.REPETITION\_COUNT - 1; i++) {

System.out.println(i);

try {

workTime += runAlgorithm();

} catch (Exception ex) {

ex.printStackTrace();

System.err.println(i);

}

}

workTime /= 100;

System.out.printf("Число повторений: %d; Кол-во файлов: %d; Среднее время: %.2f мс\n",

Application.REPETITION\_COUNT, selectedFiles.length, workTime);

}

////////////////////////////////////////////////////////////////////////////////////////////////////////

} catch (Exception ex) {

ex.printStackTrace();

}

});

add(run);

WebFileDrop fileDrop = new WebFileDrop();

fileDrop.setFont(new Font("Dialog", Font.PLAIN, 12));

fileDrop.setDropText("Перетащите сюда файлы для обратотки или кликните для выбора всей папки");

fileDrop.setMargin(0);

fileDrop.setFileFilter(new FilesFilter() {

@Override

public boolean accept(File file) {

return Application.isSupportableFile(file);

}

});

fileDrop.addFileSelectionListener(files -> {

selectedFiles = files.toArray(new File[0]);

});

fileDrop.addMouseListener(new MouseAdapter() {

@Override

public void mouseClicked(MouseEvent e) {

if (SwingUtilities.isRightMouseButton(e)) {

WebPopupMenu popup = new WebPopupMenu();

WebMenuItem clear = new WebMenuItem("Очистить");

clear.setEnabled(true);

clear.addActionListener(event -> {

fileDrop.removeAllSelectedFiles();

selectedFiles = null;

});

popup.add(clear);

popup.show(fileDrop, e.getPoint().x, e.getPoint().y);

}

if (SwingUtilities.isLeftMouseButton(e)) {

WebDirectoryChooser chooser = new WebDirectoryChooser(frame, "Выберите папку");

chooser.setFilter(new FilesFilter() {

@Override

public boolean accept(File file) {

return file.isDirectory();

}

});

chooser.setSelectedDirectory(Application.currentDir);

final int result = chooser.showDialog();

if (result == JFileChooser.APPROVE\_OPTION) {

File dir = chooser.getSelectedDirectory();

selectedFiles = dir.listFiles();

System.out.println("chooser: " + Arrays.toString(dir.list()));

fileDrop.setSelectedFiles(Arrays.stream(Objects.requireNonNull(selectedFiles)).collect(Collectors.toList()));

}

}

}

});

fileDrop.setCursor(new Cursor(Cursor.HAND\_CURSOR));

WebScrollPane filesDropScroll = new WebScrollPane(fileDrop);

// filesDropScroll.setLayout();

filesDropScroll.setBounds(25, 55, getWidth() - 50, 150);

add(filesDropScroll);

WebSeparator separator2 = new WebSeparator(WebSeparator.HORIZONTAL);

separator2.setBounds(25, 170, getWidth() - 25, 1);

add(separator2);

textArea = new WebTextArea(getStyleId(), 3, 20);

WebScrollPane scrollPane = new WebScrollPane(textArea);

scrollPane.setBounds(25, 220, getWidth() - 50, 420);

add(scrollPane);

}

private double runAlgorithm() throws Exception {

Distance distance = Distance.distance(Application.config.clustering.similarityMeasure);

double initTime, workTime;

Map<String, List<String>> result;

ExperimentsInterface experiments;

File[] clone = selectedFiles.clone();

if (mode == 0) {

experiments = new Clustering(Application.config.clustering.separateValue, distance);

result = experiments.run(clone);

initTime = experiments.getInitTime();

workTime = experiments.getWorkTime();

textArea.setText(experiments.getBuilder());

} else {

List<Class> classes = diplom.classification.Classification.parseClasses(this.classes);

experiments = new Classification(classes, distance);

result = experiments.run(clone);

initTime = experiments.getInitTime();

workTime = experiments.getWorkTime();

textArea.setText(experiments.getBuilder());

}

return workTime;

}

}

Utils.java

package diplom.gui;

import javax.imageio.ImageIO;

import java.awt.\*;

import java.io.IOException;

import java.io.InputStream;

public class Utils {

public static Image loadImage(Class c, String path) {

ClassLoader loader = c.getClassLoader();

InputStream is = loader.getResourceAsStream(path);

try {

if (is != null) {

return ImageIO.read(is);

}

} catch (IOException e) {

return null;

}

return null;

}

}

FilteredUnigram.java

package diplom.nlp;

import diplom.nlp.stemming.StemmerPorterRU;

import diplom.nlp.stopwords.StopWords;

import java.util.\*;

public class FilteredUnigram {

public static Map<String, String> origin = new HashMap<>();

public static List<String> get(String text) {

if (text == null) return Collections.singletonList("");

String[] words = text.toLowerCase()

.replaceAll("[^a-zа-я\\s]", "")

.replaceAll("[\\s]+", " ")

.split("\\s");

for (int i = 0; i < words.length; i++) {

String stem = StemmerPorterRU.stem(words[i]);

origin.put(stem, words[i]);

words[i] = stem;

}

List<String> uniqueValues = new ArrayList<>(Arrays.asList(words));

// при вычислении тематического сходства стоп-слова можно игнорировать

uniqueValues.removeIf(word -> word.equals("") || StopWords.contains(word));

return uniqueValues;

}

}

StemmerPorterRU.java

package diplom.nlp.stemming;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class StemmerPorterRU {

private static final Pattern PERFECTIVEGROUND = Pattern.compile("((ив|ивши|ившись|ыв|ывши|ывшись)|((?<=[ая])(в|вши|вшись)))$");

private static final Pattern REFLEXIVE = Pattern.compile("(с[яь])$");

private static final Pattern ADJECTIVE = Pattern.compile("(ее|ие|ые|ое|ими|ыми|ей|ий|ый|ой|ем|им|ым|ом|его|ого|ему|ому|их|ых|ую|юю|ая|яя|ою|ею)$");

private static final Pattern PARTICIPLE = Pattern.compile("((ивш|ывш|ующ)|((?<=[ая])(ем|нн|вш|ющ|щ)))$");

private static final Pattern VERB = Pattern.compile("((ила|ыла|ена|ейте|уйте|ите|или|ыли|ей|уй|ил|ыл|им|ым|ен|ило|ыло|ено|ят|ует|уют|ит|ыт|ены|ить|ыть|ишь|ую|ю)|((?<=[ая])(ла|на|ете|йте|ли|й|л|ем|н|ло|но|ет|ют|ны|ть|ешь|нно)))$");

private static final Pattern NOUN = Pattern.compile("(а|ев|ов|ие|ье|е|иями|ями|ами|еи|ии|и|ией|ей|ой|ий|й|иям|ям|ием|ем|ам|ом|о|у|ах|иях|ях|ы|ь|ию|ью|ю|ия|ья|я)$");

private static final Pattern RVRE = Pattern.compile("^(.\*?[аеиоуыэюя])(.\*)$");

private static final Pattern DERIVATIONAL = Pattern.compile(".\*[^аеиоуыэюя]+[аеиоуыэюя].\*ость?$");

private static final Pattern DER = Pattern.compile("ость?$");

private static final Pattern SUPERLATIVE = Pattern.compile("(ейше|ейш)$");

private static final Pattern I = Pattern.compile("и$");

private static final Pattern P = Pattern.compile("ь$");

private static final Pattern NN = Pattern.compile("нн$");

public static String stem(String word) {

word = word.toLowerCase();

word = word.replace('ё', 'е');

Matcher m = RVRE.matcher(word);

if (m.matches()) {

String pre = m.group(1);

String rv = m.group(2);

String temp = PERFECTIVEGROUND.matcher(rv).replaceFirst("");

if (temp.equals(rv)) {

rv = REFLEXIVE.matcher(rv).replaceFirst("");

temp = ADJECTIVE.matcher(rv).replaceFirst("");

if (!temp.equals(rv)) {

rv = temp;

rv = PARTICIPLE.matcher(rv).replaceFirst("");

} else {

temp = VERB.matcher(rv).replaceFirst("");

if (temp.equals(rv)) {

rv = NOUN.matcher(rv).replaceFirst("");

} else {

rv = temp;

}

}

} else {

rv = temp;

}

rv = I.matcher(rv).replaceFirst("");

if (DERIVATIONAL.matcher(rv).matches()) {

rv = DER.matcher(rv).replaceFirst("");

}

temp = P.matcher(rv).replaceFirst("");

if (temp.equals(rv)) {

rv = SUPERLATIVE.matcher(rv).replaceFirst("");

rv = NN.matcher(rv).replaceFirst("н");

} else {

rv = temp;

}

word = pre + rv;

}

return word;

}

}

StopWords.java

package diplom.nlp.stopwords;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.util.HashSet;

import java.util.Set;

public class StopWords {

private static Set<String> words;

static {

words = new HashSet<>();

ClassLoader loader = StopWords.class.getClassLoader();

InputStream[] iss = {

loader.getResourceAsStream("stop-words/fergiemcdowall\_stopwords\_ru.txt"),

loader.getResourceAsStream("stop-words/geonetwork-rus.txt"),

loader.getResourceAsStream("stop-words/stop-words-russian.txt")

};

String line;

BufferedReader br;

try {

for (InputStream is : iss) {

br = new BufferedReader(new InputStreamReader(is));

while ((line = br.readLine()) != null) {

words.add(line);

}

}

} catch (IOException | NullPointerException e) {

System.err.println("ошибка чтения стоп-слов");

e.printStackTrace();

}

}

public static boolean contains(String word) {

return words.contains(word);

}

}

Tests.java

package diplom;

import com.google.gson.Gson;

import com.google.gson.GsonBuilder;

import com.google.gson.reflect.TypeToken;

import org.apache.commons.io.FileUtils;

import java.io.File;

import java.io.IOException;

import java.nio.charset.Charset;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

import java.util.stream.Collectors;

public class Tests {

static Gson gson = new GsonBuilder().setPrettyPrinting().create();

public static void main(String[] args) throws Exception {

Files.walk(Paths.get("C:\\Users\\hetikk\\IdeaProjects\\diplom\\src\\main\\java\\diplom"))

.filter(p -> p.getFileName().toString().endsWith("java"))

.forEach(f -> {

System.out.println(f.getFileName());

try {

System.out.println(FileUtils.readFileToString(f.toFile(), Charset.defaultCharset()));

} catch (IOException e) {

e.printStackTrace();

}

System.out.println();

});

// final int[] wc = new int[]{5\_000, 10\_000, 15\_000, 20\_000, 25\_000, 30\_000};

// final int[] fc = new int[]{5, 10, 15, 20};

//

// List<String> list = readDictionary();

//

// // При фикс числе доков меняю число слов

// for (int wci : wc) {

// String dir = "C:\\Users\\hetikk\\IdeaProjects\\diplom\\sets\\clustering\\6\\6.1\\" + wci + "\\";

// for (int i = 0; i < 15; i++) {

// Collections.shuffle(list);

// String s = list.stream().limit(wci).collect(Collectors.joining(" "));

// FileUtils.write(new File(dir + (i + 1) + ".txt"), s, Charset.defaultCharset());

// }

// }

//

// // При фикс числе слов меняю колво доков

// for (int fci : fc) {

// String dir = "C:\\Users\\hetikk\\IdeaProjects\\diplom\\sets\\clustering\\6\\6.2\\" + fci + "\\";

// for (int i = 0; i < fci; i++) {

// Collections.shuffle(list);

// String s = list.stream().limit(15\_000).collect(Collectors.joining(" "));

// FileUtils.write(new File(dir + fci + "." + (i + 1) + ".txt"), s, Charset.defaultCharset());

// }

// }

}

static void createDictionary() throws IOException {

File f = new File("C:\\Users\\hetikk\\IdeaProjects\\diplom\\sets\\clustering\\5\\Новая папка — копия");

File[] files = f.listFiles();

List<String> words = new ArrayList<>();

// Set<String> words = new HashSet<>();

for (File file : files) {

String[] s = FileUtils.readFileToString(file, Charset.defaultCharset())

.toLowerCase()

.replaceAll("[^a-zа-я\\s]", "")

.replaceAll("[\\s]+", " ")

.split("\\s");

words.addAll(Arrays.asList(s));

}

String json = gson.toJson(words);

FileUtils.write(new File("C:\\Users\\hetikk\\IdeaProjects\\diplom\\sets\\dictionary.txt"), json, Charset.defaultCharset());

}

static List<String> readDictionary() throws IOException {

File f = new File("C:\\Users\\hetikk\\IdeaProjects\\diplom\\sets\\dictionary.txt");

String json = FileUtils.readFileToString(f, Charset.defaultCharset());

return gson.fromJson(json, new TypeToken<List<String>>() {

}.getType());

}

}

Copy.java

package diplom.utils;

import diplom.nlp.FilteredUnigram;

import java.io.File;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.nio.file.StandardCopyOption;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.Arrays;

import java.util.List;

import java.util.Map;

import java.util.Optional;

public class Copy {

public static final String SEPARATOR = File.separator;

private static final DateTimeFormatter FORMATTER = DateTimeFormatter.ofPattern("yyyy-MM-dd HH-mm-ss.S");

public static void group(String baseDir, File[] files, Map<String, List<String>> groups) throws IOException {

group(baseDir, files, groups, false);

}

public static void group(String baseDir, File[] files, Map<String, List<String>> groups, boolean replace) throws IOException {

LocalDateTime currentDateTime = LocalDateTime.now();

baseDir += SEPARATOR + "results" + SEPARATOR + currentDateTime.format(FORMATTER);

System.out.printf("base dir: %s\n", baseDir);

Files.createDirectories(Paths.get(baseDir));

int groupIndex = 1;

for (Map.Entry<String, List<String>> entry : groups.entrySet()) {

String dirName;

if (replace) {

StringBuilder nameBuilder = new StringBuilder();

String[] names = entry.getKey().split(", ");

for (int i = 0; i < names.length; i++) {

nameBuilder.append(FilteredUnigram.origin.get(names[i]));

if (i < names.length - 1) {

nameBuilder.append(", ");

}

}

dirName = firstUpperCase(nameBuilder.toString());

} else {

dirName = entry.getKey();

}

String dirPath = String.format("%s%sГруппа №%d (%s)", baseDir, SEPARATOR, groupIndex++, dirName);

Files.createDirectory(Paths.get(dirPath));

for (String fileName : entry.getValue()) {

Optional<File> file = Arrays.stream(files).filter(f -> f.getName().equals(fileName)).findFirst();

if (file.isPresent()) {

copy(dirPath, file.get());

}

}

}

}

private static void copy(String dir, File file) throws IOException {

String copy = dir + SEPARATOR + file.getName();

Files.copy(Paths.get(file.getAbsolutePath()), Paths.get(copy), StandardCopyOption.REPLACE\_EXISTING);

}

private static String firstUpperCase(String word) {

if (word == null || word.isEmpty()) return "";//или return word;

return word.substring(0, 1).toUpperCase() + word.substring(1);

}

}

Reader.java

package diplom.utils;

import org.apache.poi.xwpf.extractor.XWPFWordExtractor;

import org.apache.poi.xwpf.usermodel.XWPFDocument;

import java.io.File;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Paths;

public class Reader {

public static String readFile(File file) throws IOException {

if (file.getName().endsWith(".txt")) {

return String.join(" ", Files.readAllLines(Paths.get(file.getAbsolutePath())));

}

if (file.getName().endsWith(".docx")) {

return readWord(file);

}

throw new RuntimeException("invalid file extension");

}

private static String readWord(File file) throws IOException {

try (XWPFDocument doc = new XWPFDocument(

Files.newInputStream(Paths.get(file.getAbsolutePath())))) {

XWPFWordExtractor xwpfWordExtractor = new XWPFWordExtractor(doc);

return xwpfWordExtractor.getText();

}

}

}