Содержание

Перечень сокращений, условных обозначений, терминов

ЖКХ – жилищно-коммунальное хозяйство

1 Техническое задание на создание системы

В данном разделе приводится техническое задание на разработку системы статистического, морфологического, кластерного и онтологического анализа данных для ЖКХ.

1.1 Назначение и цели создания системы

Данная система предназначена для общего круга лиц и предусматривает использования ее любым пользователем, которому требуется провести комплексный анализ текстов.

Система создается для того, чтобы:

а) автоматизировать расчет статистических характеристик анализируемых текстов;

б) автоматизировать определение списка терминов текста и степень их значимости.

1.2 Характеристика объекта автоматизации

Объектом автоматизации системы является автоматическое выделение терминов из текста по предметной области.

1.2.1 Общее описание

Данная система предназначается для автоматизации проведения анализа текстовых данных. Система должна включать в себя статистический, морфологический, кластерный анализы для определения набора терминов обрабатываемого текста. Статистический анализ теста должен проводиться как для взятых слов, так и для словосочетаний. Морфологический анализ должен быть организован с применением специального программного обеспечения “Mystem”. Кластерный анализ должен быть реализован на основе метода К-средних.

1.2.2 Структура и принципы функционирования

Структура разрабатываемой системы должна быть последовательной и должна состоять из следующих ключевых этапов:

а) выбор обрабатываемых файлов и последующая их загрузка в память;

б) проведение морфологического анализа загруженных текстов;

в) расчет статистические характеристики для отдельных слов и словосочетаний;

г) поиск терминов.

1.3 Требования к функциям, выполняемым системой

В разделе приводиться описание требований к основным функциям, выполняемым системой статистического, морфологического, кластерного и онтологического анализа данных для ЖКХ.

1.3.1 Функция загрузки обрабатываемых данных

Загрузка данных для обработки в разрабатываемом программном обеспечении должна производиться из текстовых файлов. Данные этих файлов должны быть загружены полностью в память программы. В программном обеспечении должна быть реализована возможность как загрузки одного файла, так и параллельное чтение данных из файлов в нескольких потоках.

1.3.2 Функция морфологического анализа текста

Функция должна выделять слова из загруженных на обработку текстов. Исключать знаки препинания, пробелы и другие разделители. Функция должна приводить все слова в каноническую форму для повышения точности выполнения последующих функций программы. Для реализации данной функции является обязательным использование специального программного обеспечения “Mystem”. Приложение должно быть встроено в реализуемое программное обеспечение, а обмен данными построен на основе файлового взаимодействия.

1.3.3 Функция статистического анализа слов и словосочетаний

Назначение функции заключается в произведении расчетов статистических характеристик слов и словосочетаний. Этими характеристиками являются:

а) частота встречаемости слова/словосочетания в тексте;

б) статистическая значимость встречаемости слов словосочетания в тексте по методу Mutual Information;

в) степень взаимосвязи двух слов по методу T-Score;

г) степень значимости словосочетания в тексте по методу Log-Likelihood;

д) степень значимости словосочетания в тексте по методу TF×IDF.

1.3.4 Функция кластерного анализа текста и определения набора терминов

Функция осуществляет поиск слов/словосочетаний-терминов в тексте на основе данных, полученных в ходе статистического анализа.

2 Информационное обеспечение системы

В данном разделе рассматриваются структуры источников данных для системы статистического, морфологического, кластерного и онтологического анализа данных для ЖКХ. Отдельно приводится описание онтологии предметной области.

2.1 Организация входных данных

Основными входными данными системы являются текстовые файлы формата .pdf, .txt. Файлы могут иметь произвольную структуру.

2.2 Организация обработки данных

Обработка данных выполняется по порядку в соответствии с описанием функций программы в разделе техническое задание. Каждая функция выполняется только при успешном выполнении всех предыдущих. Перед вызовом каждой функции возможна предварительная настройка их параметров.

2.3 Организация выдачи информации

Выдача промежуточных и конечного результатов работы системы осуществляется с помощью специальных элементов пользовательского интерфейса. Результаты, полученные в ходе вычислений, могут быть экспортированы в файл формата .json для последующего использования, например для визуализации данных.

2.4 Онтология предметной области

Онтология дает возможность представить данные в виде упорядоченной иерархической структуры. В рамках разработанной системы для анализа слабоструктурированных данных, текстов, была создана онтология, которая отражает особенности предметной области.

На рисунке 1 представлена онтология, дающая общее представление об основных объектах сферы ЖКХ.

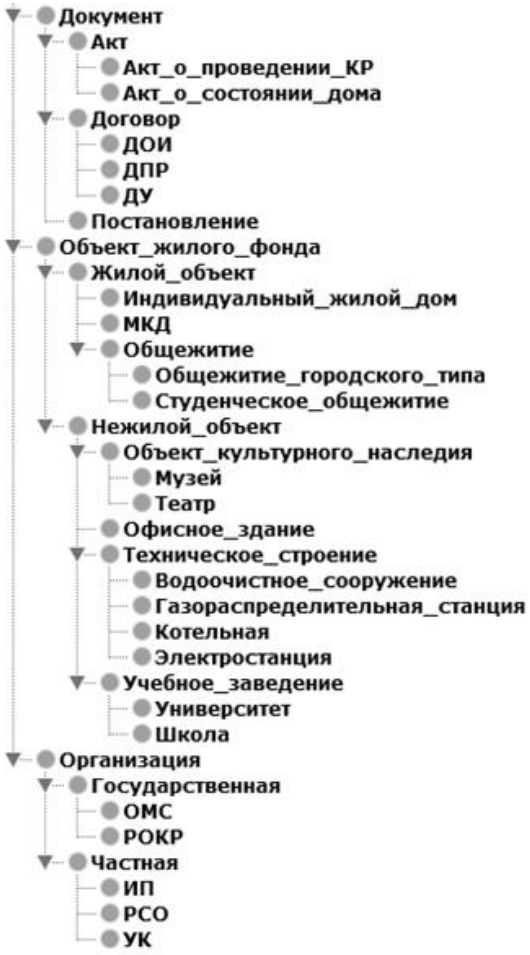

Рисунок 1. Структура онтологии

Онтология подразделяется на 3 группы классов:

а) объекты жилого фонда, многоквартирные дома, муниципальные учреждения и т.д.;

б) организации, производящие контроль и управление над объектами жилого фонда;

в) документы, которые описывают основные положения бизнес-процессов управления и контроля.

В таблице 2 приводится подробное описание представленной онтологии.

Таблица 2. Описание онтологии предметной области

Название	Описание	Data property	Примечание
Документ	Класс «Документ» объединяет в себе все возможные документы касающихся состояния	Дата начала действия: dateTime, Дата окончания действия: dateTime,	Содержит в себе подклассы: Акт и Договор.

Название	Описание	Data property	Примечание
	дома и управления им.	Дата подписания: dateTime, Номер: string	
Акт	Класс «Акт» предназначен для сбора информации об актах на состояние дома и проведения КР.	Положения: string, Состояние дома: string	Является подклассом класса «Документ». Содержит в себе подклассы: Акт о проведении КР и Акт о состоянии дома.
Акт о проведении Капитального ремонта (КР)	Класс «Акт о проведении КР» предназначен для сбора информации о проведении в домах КР.	Дата проведения КР: dateTime	Является подклассом класса «Акт».
Акт о состоянии дома	Класс «Акт о состоянии дома» предназначен для сбора информации о состоянии дома.	-	Является подклассом класса «Акт».
Договор	Класс «Договор» предназначен для сбора информации о договорах на управление, поставку ресурсов и общего имущества.	Обязательства: string	Является подклассом класса «Документ». Содержит в себе подклассы: Договор общего имущества, Договор поставки ресурсов и Договор управления.
Договор общего имущества (ДОИ)	Класс «Договор общего имущества» предназначен для сбора информации о договорах общего имущества.	Организация: string	Является подклассом класса «Договор».
Договор поставки ресурсов (ДПР)	Класс «Договор поставки ресурсов» предназначен для сбора информации о поставке ресурсов.	РСО: string	Является подклассом класса «Договор».
Договор управления (ДУ)	Класс «Договор управления» предназначен для сбора информации о договорах управления.	Управляющая организация: string	Является подклассом класса «Договор».
Постановление	Класс «Постановление» предназначен для сбора информации о постановлениях.	Закон: string	Является подклассом класса «Документ».
Объект жилого фонда	Класс «Объект жилого фонда» предназначен для сбора информации о	Аварийность: boolean, Адрес: string, Год постройки:	Содержит в себе подклассы: Жилой объект и Нежилой

Название	Описание	Data property	Примечание
	жилых и нежилых объектах.	dateTime, Описание: string, Площадь: decimal, Этажность: int	объект
Жилой объект	Класс «Жилой объект» предназначен для сбора информации о индивидуальных жилых домах, многоквартирных домах и общежитие.	Жилая площадь: decimal, Тип управления: string, Число жителей: int	Является подклассом класса «Объект жилого фонда». Содержит в себе подклассы: Индивидуальный жилой дом, Многоквартир-ный дом и Общежитие
Индивидуаль-ный жилой дом	Класс «Индивидуаль-ный жилой дом» предназначен для сбора информации о индивидуальных жилых домах.	Наличие канализации: boolean, Площадь участка: decimal	Является подклассом класса «Жилой объект».
Многоквартир-ный дом (МКД)	Класс «Многоквартирный дом» предназначен для сбора информации о многоквартирных домах	Количество квартир: int, Количество лифтов: int, Количество подъездов: int, Наличие технического этажа: boolean	Является подклассом класса «Жилой объект».
Общежитие	Класс «Общежитие» предназначен для сбора информации о общежитиях городского типа и студенческих общежитиях	Количество комнат: int, Наличие вахты: boolean	Является подклассом класса «Жилой объект». Содержит в себе подклассы: Общежитие городского типа и Студенческое общежитие
Общежитие городского типа	Класс «Общежитие городского типа» предназначен для сбора информации об общежитиях городского типа.	-	Является подклассом класса «Общежитие».
Студенческое общежитие	Класс «Студенческое общежитие» предназначен для сбора информации о студенческих общежитиях.	-	Является подклассом класса «Общежитие».
Нежилой объект	Класс «Нежилой объект» предназначен для сбора информации об объектах	-	Является подклассом класса «Объект жилого фонда».

Название	Описание	Data property	Примечание
	культурного наследия, офисных зданиях, технических строениях и учебных заведениях.		Содержит в себе подклассы: Объект культурного наследия, Офисное здание, Техническое строение и Учебное заведение
Объект культурного наследия	Класс «Объект культурного наследия» предназначен для сбора информации о музеях и театрах.	Максимальное число посетителей: int	Является подклассом класса «Нежилой объект». Содержит в себе подклассы: Музей и Театр
Музей	Класс «Музей» предназначен для сбора информации о музеях.	Количество выставочных помещений: int	Является подклассом класса «Объект культурного наследия».
Театр	Класс «Театр» предназначен для сбора информации о театрах.	Количество концертных залов: int	Является подклассом класса «Объект культурного наследия».
Офисное здание	Класс «Офисное здание» предназначен для сбора информации об офисных зданиях.	Количество организаций: int	Является подклассом класса «Объект жилого фонда».
Техническое строение	Класс «Техническое строение» предназначен для сбора информации о водоочистных сооружениях, газораспределительных станциях, котельных и электростанциях.	Вид ресурса: string, Вид услуги: string, Количество потребителей: int, Производственная мощность: decimal	Является подклассом класса «Нежилой объект». Содержит в себе подклассы: Водоочистное сооружение, Газораспредели-тельная станция, Котельная и Электростанция
Водоочистное сооружение	Класс «Водоочистное сооружение» предназначен для сбора информации о водоочистных сооружениях.	Тип фильтрации: string	Является подклассом класса «Техническое строение».
Газораспреде-лительная станция	Класс «Газораспреде-лительная станция» предназначен для сбора информации о газораспределитель-ных станциях.	Диаметр труб: decimal, Объем распределяемого газа: decimal	Является подклассом класса «Техническое строение».

Название	Описание	Data property	Примечание
Котельная	Класс «Котельная» предназначен для сбора информации о котельных.	Количество котлов: int, Марка котла: string	Является подклассом класса «Техническое строение».
Электро-станция	Класс «Электро-станция» предназначен для сбора информации о электростанциях.	Вид подстанции: string, Напряжение: decimal	Является подклассом класса «Техническое строение».
Учебное заведение	Класс «Учебное заведение» предназначен для сбора информации о университетах и школах.	-	Является подклассом класса «Нежилой объект». Содержит в себе подклассы: Университет и Школа
Университет	Класс «Университет» предназначен для сбора информации о университетах.	-	Является подклассом класса «Учебное заведение».
Школа	Класс «Школа» предназначен для сбора информации о школах.	-	Является подклассом класса «Учебное заведение».
Организация	Класс «Организация» предназначен для сбора информации о государственных и частных организациях.	ИНН: string, Количество инженеров: int, Количество работников: int, Количество рабочих: int, Контактные данные: string, КПП: string, Название: string, ОГРН: string	Содержит в себе подклассы: Государственная и Частная.
Государствен-ная	Класс «Государственная» предназначен для сбора информации о органах местного самоуправления и региональных операторах капитального ремонта.	Официальный представитель: string	Является подклассом класса «Организация». Содержит в себе подклассы: Органы местного самоуправления и Региональный оператор капитального ремонта.
Органы местного самоуправ-ления (ОМС)	Класс «Органы местного самоуправ-ления» предназначен для сбора информации о органах местного	-	Является подклассом класса «Государственная».

Название	Описание	Data property	Примечание
	самоуправления.		
Региональный оператор капитального ремонта (РОКР)	Класс «Региональный оператор капитального ремонта» предназначен для сбора информации о региональных операторах капитального ремонта.	-	Является подклассом класса «Государственная».
Частная	Класс «Частная» предназначен для сбора информации об индивидуальных предпринимателях, ресурсоснабжающих организациях и управляющих компаниях.	Директор: string, Одобренный подрядчик: boolean	Является подклассом класса «Организация». Содержит в себе подклассы: Индивидуальный предпри-ниматель, Ресурсо-снабжающая организация и Управляющая компания.
Индивидуальный предпри-ниматель (ИП)	Класс «Индивидуаль-ный предприниматель» предназначен для сбора информации об индивидуальных предпринимателях.	-	Является подклассом класса «Частная».
Ресурсо-снабжающая организация (РСО)	Класс «Ресурсо-снабжающая организация» предназначен для сбора информации о ресурсоснабжающих организациях.	Тип обслуживания: string	Является подклассом класса «Частная».
Управляющая компания (УК)	Класс «Управляющая компания» предназначен для сбора информации об управляющих компаниях.	-	Является подклассом класса «Частная».

Классы онтологии взаимосвязаны с помощью объектных слов, перечень которых представлен на рисунке 3.

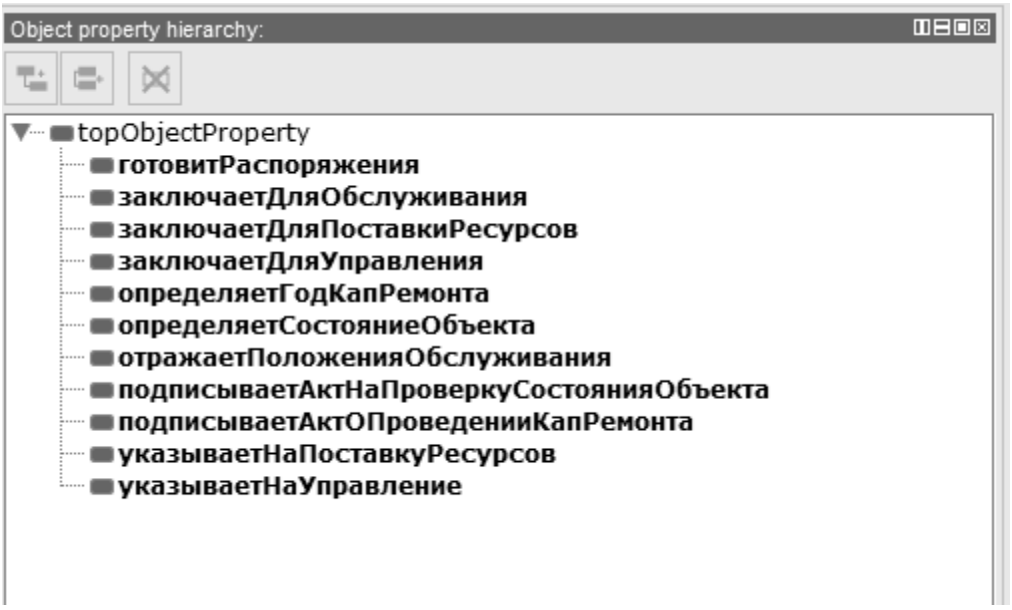

Рисунок 3. Объектные свойства онтологии

В таблице 4 представлено подробное описание объектных свойств онтологии.

Таблица 4. Описание объекных свойств онтологии

Наименование	Описание
Готовит распоряжения	Свойство относится к государственной организации, а именно органам местного самоуправления. Указывает на подготовку ОМС распоряжения
Заключает для обслуживания	Свойство относится к частной организации, а именно индивидуальным предпринимателям. Указывает на подготовку договора общего имущества, заключенного для обслуживания объекта
Заключает для поставки ресурсов	Свойство относится к частной организации, а именно ресурсоснабжающим организациям. Указывает на заключение договора поставки энергетических ресурсов до объекта
Заключает для управления	Свойство относится к частной организации, а именно управляющим компаниям. Указывает на заключение договора управления объектом жилого фонда
Определяет год капитального ремонта	Свойство относится к документам, а именно актам о проведении капитального ремонта. Указывает на отношение договора к объекту жилого фонда
Определяет состояние объекта	Свойство относится к документам, а именно актам о состоянии дома. Указывает на отношение договора к объекту жилого фонда
Отражает положения обслуживания	Свойство относится к документам, а именно договорам общего имущества. Указывает на отношение договора к жилому объекту
Подписывает акт	Свойство относится к государственной организации, а именно

Наименование	Описание
на поверку состояния объекта	региональному оператору капитального ремонта. Указывает на подписание акта поверки состояния объекта РОКР
Подписывает акт о проведении капитального ремонта	Свойство относится к государственной организации, а именно региональному оператору капитального ремонта. Указывает на подписание акта о проведении капитального ремонта РОКР
Указывает на поставку ресурсов	Свойство относится к договору на поставку ресурсов, а именно поставку ресурсов ресурсоснабжающими организациями. Указывает на заключение договора поставки энергетических ресурсов до объектов жилого фонда
Указывает на управление	Свойство относится к договору управления, а именно управления объектами жилого фонда. Указывает на заключение договора управления

3 Алгоритмическое обеспечение системы

В разделе рассматриваются алгоритмы, разработанные в процессе создания системы статистического, морфологического, кластерного и онтологического анализа данных для ЖКХ. Реализация алгоритмов на ЯВУ C# представлена в приложении А.

3.1 Алгоритм кластеризации

FCM (Fuzzy Classifier Means) - это алгоритм кластеризации, с помощью которого производится автоматическая классификация множества объектов, которые задаются векторами признаков в пространстве признаков. В случае рассматриваемой системы объектами является множество слов и словосочетаний. Их векторы признаков – наборы характеристик, получаемые в результате статистического анализа.

На схеме 5 представлена схема подпрограммы, которая реализует алгоритм кластеризации.

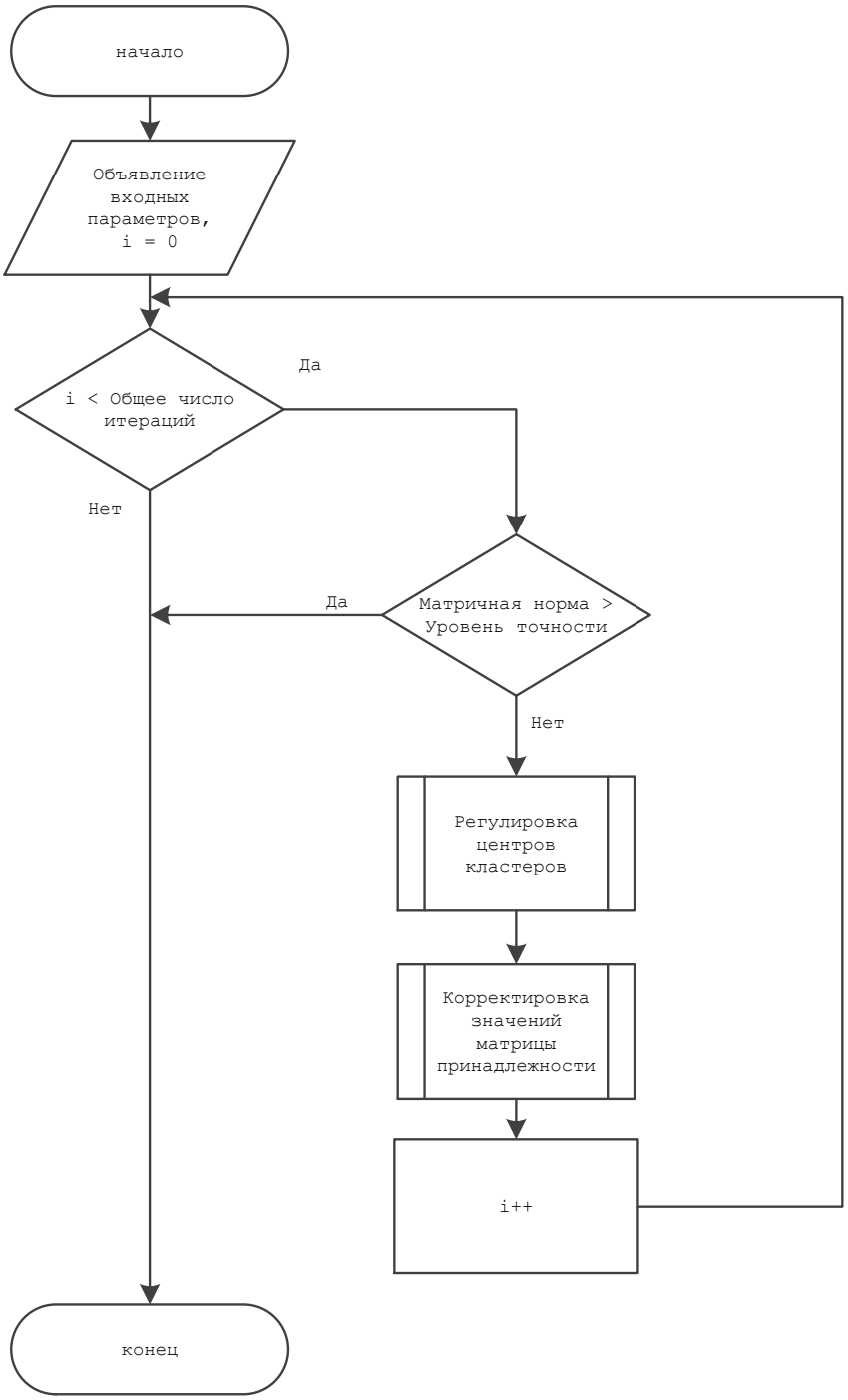

Схема 5. Блок-схема подпрограммы FCM-алгоритма

Входными данными для алгоритма являются:

а) количество итераций, определяющих точность выполнения алгоритма;

б) центры кластеров;

в) матрица, описывающая степень принадлежности каждого из объектов к центрам кластеров;

г) параметр нечеткости кластеризации. Рекомендуемое значение параметра выбирается в пределах ~1,5...2. При реализации алгоритма величина параметра была установлена в значение 1,6;

д) параметр сходимости алгоритма (уровень точности). Значение параметра установлено в 0.001.

На первом шаге алгоритма производится заполнение матрицы принадлежности объектов установленным центрам кластеров. Значения матрицы устанавливаются случайным образом, но с условием, что сумма значений вектора принадлежности отдельного объекта не превышает 1.0.

На втором шаге производится запуск основного цикла алгоритма кластеризации данных, регулируются значения центров кластеров по формуле:

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^{1.6} x_i}{\sum_{i=1}^{N} u_{ij}^{1.6}}$$

, где c_j – центр j-го кластера, N – количество объектов, x_i – значение i-Го объекта, u_{ij} – степень принадлежности объекта i кластеру j.

На третьем этапе происходит корректировка значений матрицы принадлежности по формуле:

$$u_{ij} = \frac{1}{\sum_{l=1}^{c} \left(\frac{\|x_i - c_j\|}{\|x_i - c_l\|} \right)^{3.33}}$$

, где c – количество кластеров, c_j – вектор центра j-го кластера, c_l – вектор центра l-го кластера.

По завершении каждой итерации цикла алгоритма производятся проверки:

а) сравнения числа уже выполненных итераций с заранее предустановленным количеством;

б) сравнения параметра сходимости алгоритма и матричной нормы, которая должна быть больше для продолжения работы алгоритма.

При выполнении хотя бы одного из условий алгоритм считается

завершенным.

Результом работы алгоритма является итоговая матрица принадлежности объектов к каждому из кластеров.

4 Программное обеспечение системы

В данном разделе пояснительной записки рассматривается структура разработанного программного обеспечения на уровне модулей и отдельных компонентов.

4.1 Структура программного обеспечения

Приложение для проведения анализа текстов состоит из четырех компонентов:

а) Core, ядро, содержащее в себе реализацию всех необходимых для проведения расчетов алгоритмов, параллельной обработки текстов, API для внешнего использования, среды выполнения программы морфологического анализа текстов mystem;

б) Helper, модуль, реализующий вспомогательные методы для работы с файловой системой;

в) FileHandler, реализующий функционал загрузки данных из файлов разных типов;

г) DesktopApplication, приложение для проведения анализа текстов для ОС Windows.

4.1.1 Модуль Core

Компонент Core, являющийся ядром разработанного программного обеспечения, включает в себя семь классов:

а) API.cs, модуль, реализующий внешний API, который используется при разработке пользовательского приложения;

б) ClasterAnalysis.cs, модуль, реализующий FCM-алгоритм кластеризации данных;

в) CommonClasses.cs, модуль, включающий описания всех используемых структур данных;

г) Configuration.cs, модуль, описывающий основные константы приложения

(например, обозначения частей речи);

д) MorphologicalAnalysis.cs, модуль, реализующий методы для морфологического анализа заданного текста;

е) Multiprocessor.cs, модуль, обеспечивающий возможность параллельной обработки нескольких текстов;

ж) MystemProvider.cs, модуль, реализующий среду для выполнения программы морфологического анализа текстов mystem

з) StatisticsAnalysis.cs, модуль, включающий реализацию алгоритмов для проведения статистического анализа

В таблице 6 представлено описание методов класса API.cs.

Таблица 6. Спецификация методов класса API.cs

Наименование	**Назначение**
1. public List<string> LoadFile(string path)	Преобразование файла в список строк его данных
2. public List<FileData> LoadFilesMulticore(List<string> paths)	Многопоточное преобразование файлов в списки строк их данных
3. public List<Lemm> HandleByMystem(List<string> fileLines)	Обработка mystem строк данных файла и получение списка лемм слов, содержащихся в тексте из файла
4. public MystemData HandleByMystem(string filePath)	Обработка файла программой mystem и получение списка лемм слов, содержащихся в тексте из файла
5. public List<MystemData> HandleByMystemMulticore(List<FileData > dataList)	Многопоточная обработка данных файлов и получение списков лемм слов
6. public List<MystemData> HandleByMystemMulticore(List<string> filesPaths)	Многопоточная обработка нескольких файлов программой mystem
7. public StatsAnalysisResult<string> ProvideWordsStatsAnalysis(List<Lemm> list)	Проведение статистического анализа на основе результатов, полученных в после обработки файла программой mystem. Результат работы функции - статистические характеристики слов в тексте.
8. public StatsAnalysisResult<WordDigram> ProvideDigramsStatsAnalysis(MystemDat	Проведение статистического анализа на основе результатов, полученных в после обработки файла программой mystem. Результат работы функции -

a data)	статистические характеристики двусловий в тексте.
9. public List<StatsAnalysisResult<string>> ProvideWordsStatsAnalysisMulticore(List <MystemData> list)	Проведение многопоточного статистического анализа для отдельных слов
10. public List<StatsAnalysisResult<WordDigram>> ProvideDigramsStatsAnalysisMulticore(List<MystemData> list)	Проведение многопоточного статистического анализа для двусловий
11. public MystemData ProvideMorphAnalysis(MystemData data, string[] excludedTypes)	Проведение морфологического анализа для одного текста
12. public List<MystemData> ProvideMorphAnalysisMulticore(List<MystemData> list, string[] excludedTypes)	Проведение многопоточного морфологического анализа для данных нескольких текстов
13. public double[,] GetDefaultClustersCenters<T>(Dictionary <T, double[]> statisticDictionary)	Поиск центров для проведения кластерного анализа на основе данных, полученных в результате выполнения кластерного анализа
14. public ClasterAnalysisResult<T> ProvideClusterAnalysis<T>(ClasterSettings<T> settings, string name)	Проведение кластерного анализа слов/двусловий (зависит от типа входных данных) для данных одного текста
15. public List<ClasterAnalysisResult<string>> ProvideWordClusterAnalysisMulticore(List<ClasterAnalysisData<string>> data)	Проведение многопоточного кластерного анализа слов для нескольких текстов
16. public List<ClasterAnalysisResult<WordDigram>> ProvideWordDigramClusterAnalysisMulticore(List<ClasterAnalysisData<WordDigram>> data)	Проведение многопоточного кластерного анализа двусловий из нескольких текстов
17. public Dictionary<T, double[]> GetDataReady<T>(StatsAnalysisResult<T> analysisResult)	Преобразование данных, полученных в результате статистического анализа в формат для проведения кластерного анализа

В таблице 7 рассматривается спецификация методов класса ClasterAnalysis.cs

Таблица **7.** Спецификация методов класса ClasterAnalysis.cs

Наименование	**Назначение**
1. private List<T> FillDataMatrix(Dictionary<T, double[]> wordStats)	Заполнение матрицы данных из словаря
2. private void InitializeMatrix()	Инициализация матрицы принадлежности
3. private double CalculateEuclidLength(int i_data, int j_cl)	Получение Евклидова расстояния между векторами
4. private double CalculateNewMemberDegreeItem(int i, int j)	Расчет нового значения матрицы принадлежности
5. private void CalculateClusterCenters()	Функция перерасчета центров кластеров, согласно данным матрицы принадлежности на i-ой итерации выполнения кластерного анализа
6. private double RecalculateMemberDegree()	Перерасчет матрицы принадлежности
7. public Dictionary<T, double[]> Clasterize()	Реализация алгоритма кластеризации данных

В таблице 8 рассматривается спецификация структур данных, определенных в классе CommonClasses.cs.

Таблица **8.** Спецификация методов класса CommonClasses.cs

Наименование	**Назначение**
1. public class FileData	Класс, описывающий данные файла
2. public class MystemData	Класс, применяющийся для сохранения данных, полученных в результате обработки файла программой mystem
3. public class Lemm	Класс, описывающий лемму слова (начальную форму)
4. public class Analysis	Класс, описывающий морфологические признаки слова, тип части речи
5. public class WordDigram	Класс, описывающий двусловие
6. public class StatsAnalysisResult<T>	Класс, предназначенный для описания результата статистического анализа слов/двусловий из заданного текста
7. public class ClasterSettings<T>	Класс, описывающий входные параметры кластерного анализа

8. public class ClasterAnalysisData<T>	Класс, описывающий результат выполнения кластерного анализа

В таблице 9 рассматривается спецификация основных методов, класса Multiprocessor.cs.

Таблица 9. Спецификация методов класса Multiprocessor.cs

Наименование	Назначение
1. public void MultiprocessorMorphAnalysis(List<MystemData> list, string[] excludedTypes)	Метод, обеспечивающий проведение морфологического анализа на основе данных нескольких текстов в параллельном режиме
2. public void MultiprocessorWordDigramClasterAnalysis(List<ClasterAnalysisData<WordDigram>> list)	Метод, обеспечивающий проведение кластерного анализа двусловий на основе данных нескольких текстов в параллельном режиме
3. public void MultiprocessorWordClasterAnalysis(List<ClasterAnalysisData<string>> list)	Метод, обеспечивающий проведение кластерного анализа слов на основе данных нескольких текстов в параллельном режиме
4. public void MultiprocessorWordStatsAnalysis(List<MystemData> list)	Метод, обеспечивающий проведение статистического анализа слов для данных нескольких текстов в параллельном режиме
5. public void MultiprocessorDigramsStatsAnalysis(List<MystemData> list)	Метод, обеспечивающий проведение статистического анализа двусловий на основе данных нескольких текстов в параллельном режиме
6. public void MultiprocessorMystemHandler(List<FileData> list)	Метод, обеспечивающий проведение параллельной обработки текстов программой mystem.exe
7. public void MultiprocessorFileRead(List<string> paths)	Метод, обеспечивающий чтение данных из нескольких текстовых файлов (поддержка расширений .pdf, .txt) в параллельном режиме

В таблице 10 представлены описания методов класса MystemProvider.cs.

Таблица 10. Спецификация методов класса MystemProvider.cs

Наименование	Назначение
1. public List<Lemm> LaunchMystem(List<string> lines, string flags = "-cgin --format json")	Метод, обеспечивающий запуск приложения mystem.exe с предустановленным режимом записи результата в файл с расширением .json.
2. private List<Lemm>	Преобразование данных json-файла, полученного в

GetMystemResult(StreamReader srdr)	результате выполнения приложения mystem.exe, в список отдельных лемм

В таблице 11 представлены описания методов класса StatiscticsAnalysis.cs.

Таблица 11. Спецификация методов класса StatisticsAnalysis.cs

Наименование	**Назначение**
1. public static Dictionary<string, double> GetFrequencyDictionary(List<string> words)	Получение частотного словаря слов текста
2. public static Dictionary<string, double> GetTF(Dictionary<string, double> words, int count)	Нахождение абсолютной частоты встречаемости слова в тексте
3. public static Dictionary<string, double> GetTF_IDF(int ccSize, Dictionary<string, double> tf_dictionary, int docNumber = 1)	Реализация алгоритма TFxIDF, предназначенного для поиска наиболее значимых слов
4. public static Dictionary<WordDigram, double> GetDigramFrequenceDictionary(List<string> wordList)	Получение частотного словаря двусловий
5. public static Dictionary<WordDigram, double> CalculateMutualInformation(Dictionary<WordDigram, double> frequencyDiagram, Dictionary<string, double> frequencyDictionary, int wordsCount)	Реализация алгоритма поиска наиболее значимых слвовосочетаний согласно методу Mutual Information
6. public static Dictionary<WordDigram, double> CalculateTScore(Dictionary<WordDigram, double> frequencyDiagram, Dictionary<string, double> frequencyDictionary, int wordsCount)	Реализация алгоритма определения степени взаимосвязи двух слов по методу TScore
7. public static Dictionary<WordDigram, double> CalculateLogLikelihood(Dictionary<WordDigram, double> frequencyDiagram)	Реализация алгоритма метода Log-Likelihood для выявления наиболее статистически значимых двусловий
8. public static int GetNgramFrequence(List<string> wordList, List<string> ngramList)	Получение частотного словаря n-грамм для заданного текста
9. public static Dictionary<T, double[]> MergeDictionaries<T>(this List<Dictionary<T, double>> dictionaries)	Преобразование данных, полученных в результате проведенных статистических методов, в единый словарь данных для кластерного анализа
9. public static int GetWordsCount(this Dictionary<string, int> words)	Получение общего числа слов в тексте
10. public static List<string> GetWords(this List<Lemm> list)	Получение списка лемм слов текста

4.1.2 Модуль Helper.cs

Модуль Helper.cs содержит реализацию вспомогательных методов, которые используются для взаимодействия с файловой системой. В таблице 12 представлена спецификация методов модуля.

Таблица 12. Описание методов модуля Helper.cs

Наименование	**Назначение**
1. public static List<string> CheckFiles(List<string> paths)	Метод, обеспечивающий проверку на возможность дальнейшей обработки входного файла с заданным расширением
2. public static void WriteFile(List<string> lines, string path)	Метод, обеспечивающий запись строк в файл
3. public static bool DeleteFile(string path)	Метод, удаляющий файл по указанному пути
4. public static FileHandler GetFileHandler(string path)	Метод, запускающий алгоритм загрузки в зависимости от расширения файла. Метод реализован с использованием паттерна проектирования “фабрика”

4.1.3 Модуль FileHandler

Модуль FileHandler реализует алгоритмы обработки файлов. Данная подпрограмма состоит из трех классов:

и) FileHandler.cs, абстрактный класс, который описывает общие свойства и методы, необходимые для реализации обработчиков считывания данных из файлов с разными расширениями;

к) TxtHandler.cs, реализация чтения данных из файла, имеющего расширение .txt;

л) PdfHandler.cs, реализация чтения данных из файла, имеющего расширение .pdf.

В таблице 13 представлена спецификация методов класса FileHandler.cs

Таблица 13. Спецификация методов абстрактного класса FileHandler.cs

Наименование	Назначение
public abstract void ReadFile(out List<string> lines)	Абстрактный метод, в реализации которого необходимо описывать алгоритм чтения данных из файла с определенным расширением

В таблице 14 представлена спецификация методов класса PdfHandler.cs, реализующего чтение данных из файла с расширением .pdf.

Таблица 14. Спецификация методов класса PdfHandler.cs

Наименование	Назначение
public override void ReadFile(out List<string> lines)	Метод, реализующий алгоритм загрузки данных из файла с расширением .pdf

В таблице 15 представлена спецификация методов класса TxtHandler.cs, реализующего чтение данных из текстового файла.

Таблица 15. Спецификация методов класса TxtHandler.cs

Наименование	Назначение
public override void ReadFile(out List<string> lines)	Метод, реализующий алгоритм загрузки данных из текстового файла

4.2 Интерфейс пользователя с системой

Пользовательский интерфейс приложения состоит из шести форм, каждая из которых выполняет определенные функции.

При запуске приложения пользователю отображается форма, представленная на рисунке 16.

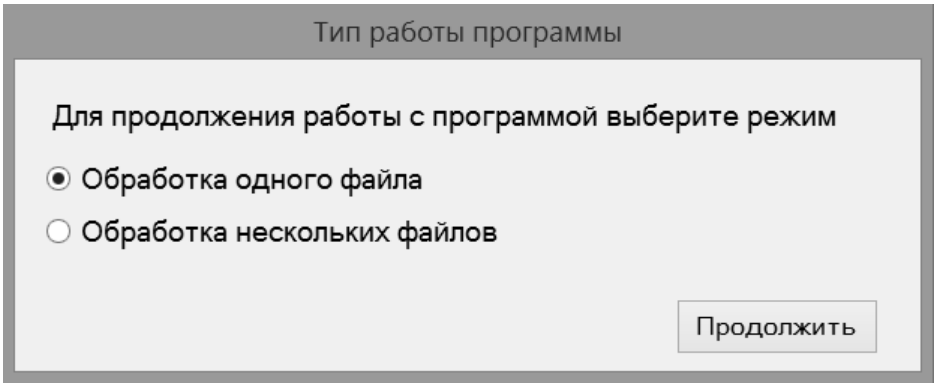

Рисунок 16. Выбор режима работы приложения

Пользователь может выбрать как вариант проведения анализа одного текста, так и нескольких. При этом необходимые вычисления будут проводиться в параллельном режиме

После определения режима работы приложения, пользователю необходимо выбрать файл (или группу файлов), который будет обработан программой. Выбор файла осуществляется с помощью диалогового окна, показанного на рисунке 17.

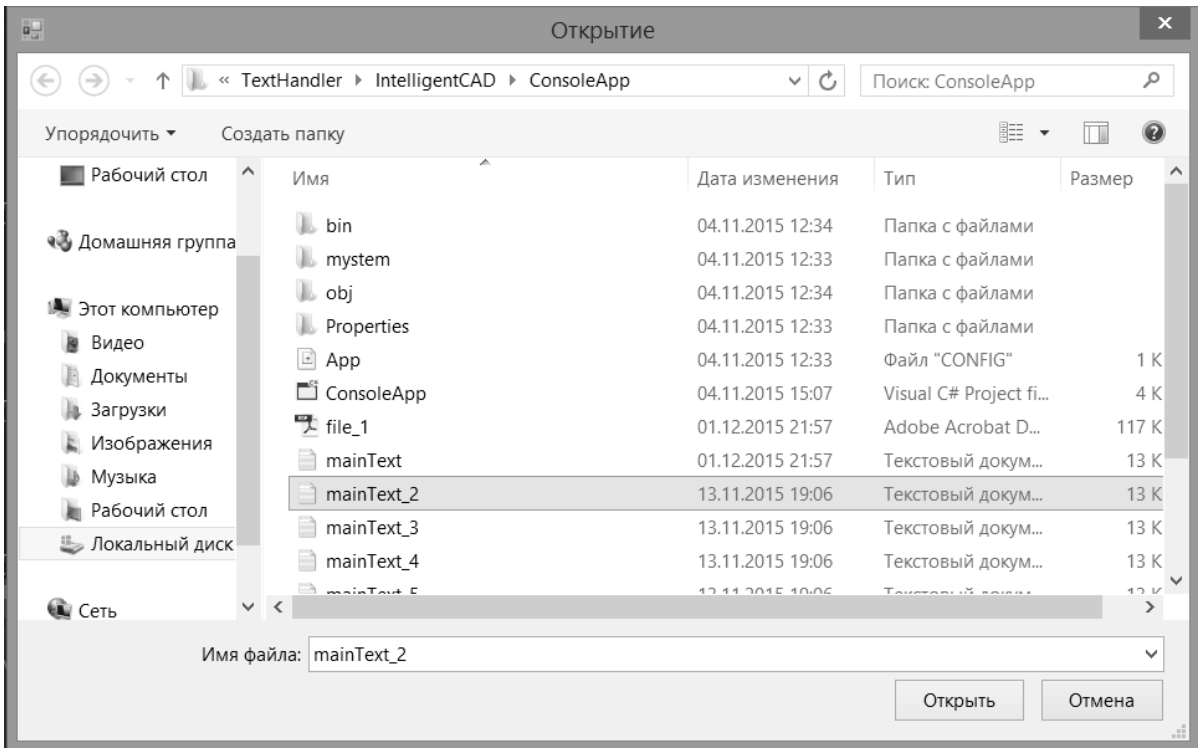

Рисунок 17. Процесс выбора файла для проведения анализа

Все выбранные файлы отображаются в списке в левой части окна приложения, как показано на рисунке 18.

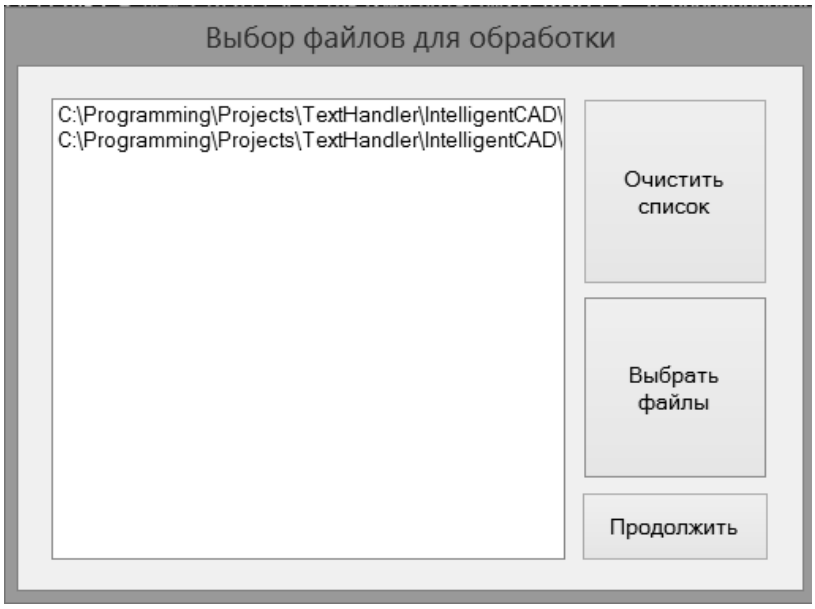

Рисунок 18. Отображение группы выбранных файлов

Также у пользователя есть возможность перезагрузить файлы, предварительно очистив список. Для этого нужно нажать на кнопку “Очистить список”, а затем выбрать новый файл (группу файлов) в диалоговом окне.

На следующем этапе производится обработка указанных файлов утилитой mystem.exe. При этом список лемм слов файла отображается в списке, как показано на рисунке 19.

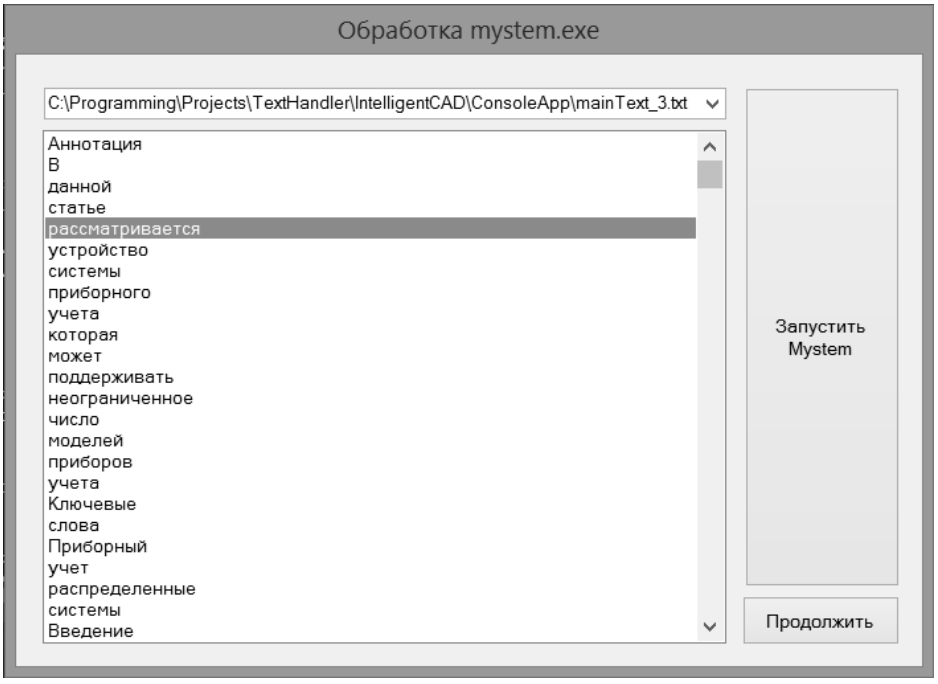

Рисунок 19. Отображение результата обработки файла приложением mystem.exe

Если обработка проводится для нескольких файлов, то отобразить результат для каждого из них можно, выбрав соответствующий файл в выпадающем меню в верхней части окна формы.

На следующем этапе пользователю предлагается удалить из дальнейшей обработки наименее значимые по своему смыслу части речи: союзы, предлоги и частицы. Интерфейс этого окна приложения представлен на рисунке 20.

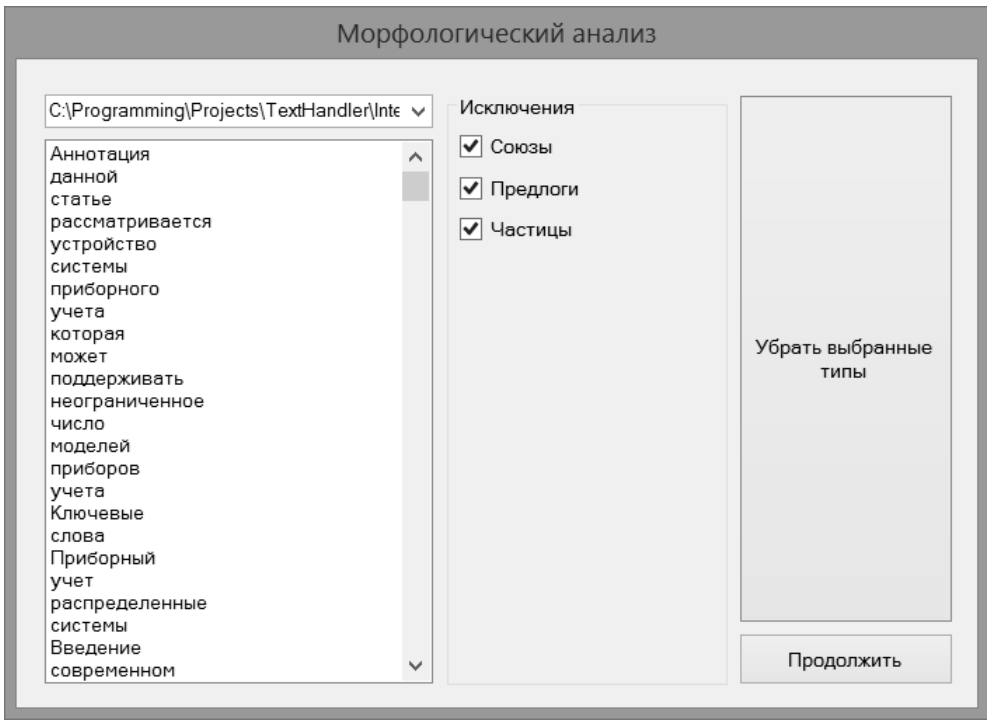

Рисунок 20. Удаление не значимых частей речи из списка лемм текста

Функционал следующей формы приложения позволяет провести статистический анализ двусловий текста (группы текстов). Анализ состоит из четырех характеристик:

а) частота встречаемости двусловий в тексте;

б) loglikelihood, определение наиболее стастически значимых двусловий;

в) mutual information, определение наиболее значимых двусловий;

г) TScore, степень взаимосвязи двух слов;

Отображение результатов статистического анализа представлено на рисунке 21.

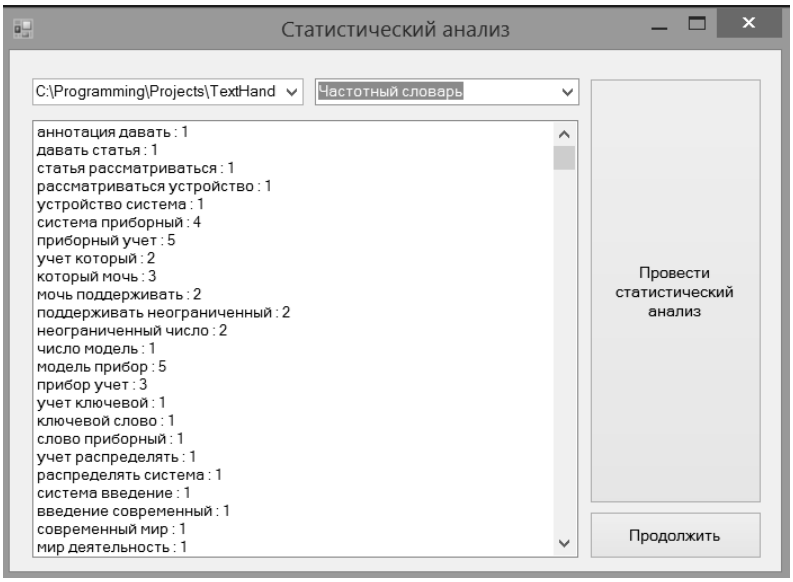

Рисунок 21. Окно приложения с результатами статистического анализа

Пользователю предоставляется возможность просмотра значений каждой из указанных характеристик, полученных в результате вычислений. Этот выбор можно произвести с помощью выпадающего меню в верхней части данного окна приложения.

На следующем этапе производится кластерный анализ словосочетаний текста. Результат выполнения кластерного анализа представлен на рисунке 22.

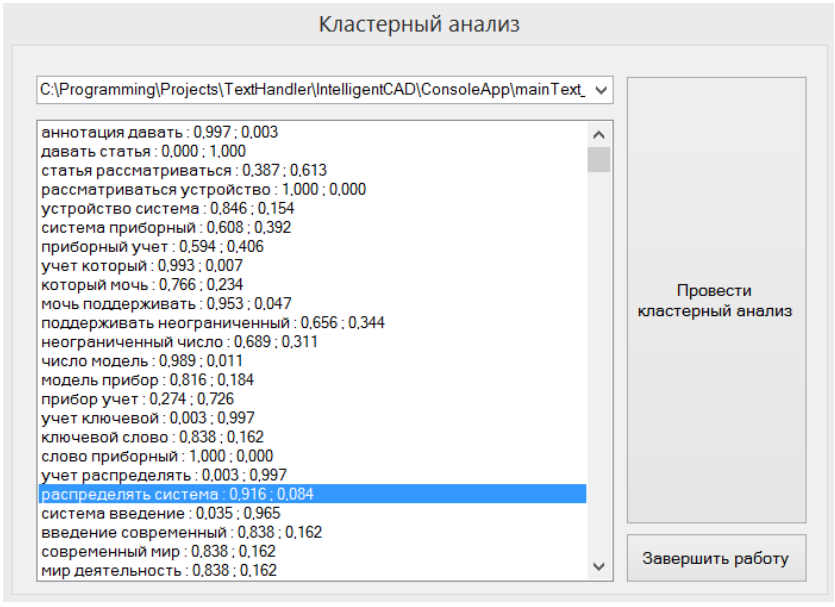

Рисунок 22. Форма отображения результатов кластерного анализа

5 Результаты проведенных исследований

В данном разделе пояснительной записки представлены результаты, полученные в ходе проведенных исследований, представлено обоснование выбора параметров центров для кластерного анализа.

5.1 Область исследований

Первая задача, решаемая разработанной системой, заключается в сборе статистических данных о слабо структурированном наборе информации, тексте. В качестве предмета исследования была выбрана статья, описывающая систему приборного учета для сбора и хранения показаний со счетчиков, установленных в многоквартирных домах и других объектах сферы жилищно-коммунального хозяйства.

Система позволяет выполнять процедуру кластерного анализа на основании полученной статистической информации. Цель проведения такого анализа – выделение слов и словосочетаний, являющихся терминами из предметной области текста.

Для получения точных результатов перед проведением кластерного производится морфологический анализ. Он позволяет привести слова, составляющие текст, в каноническую форму. Также такой тип анализа позволяет определить части речи слов и удалить слова, имеющие наименьшую значимость в тексте (например, частицы, предлоги и союзы).

5.2 Выбор центров кластеров

Целью проведения кластерного анализа является определение принадлежности слова/словосочетания к категориям термин/нетермин. Примером слова-термина является “прибор”, словосочетания “приборный учет” в рамках текста, на котором производились исследования.

Входными данными для кластерного анализа является массив значений статистических характеристик всех слов или словосочетаний. Кластеры

определяются как набор с минимальными значениями, обозначающий “нижнюю границу”, и с максимальными – “верхнюю границу”.

Такой выбор центров позволяет четко обозначить области принадлежности словосочетаний/слов к центрам и повысить точность определения индекса значимости рассматриваемой единицы языка в тексте.

5.3 Результат проведения кластерного анализа для словосочетаний

После проведения процедуры кластерного анализа словосочетаний в тексте был получен результат, представленный на рисунке 23.

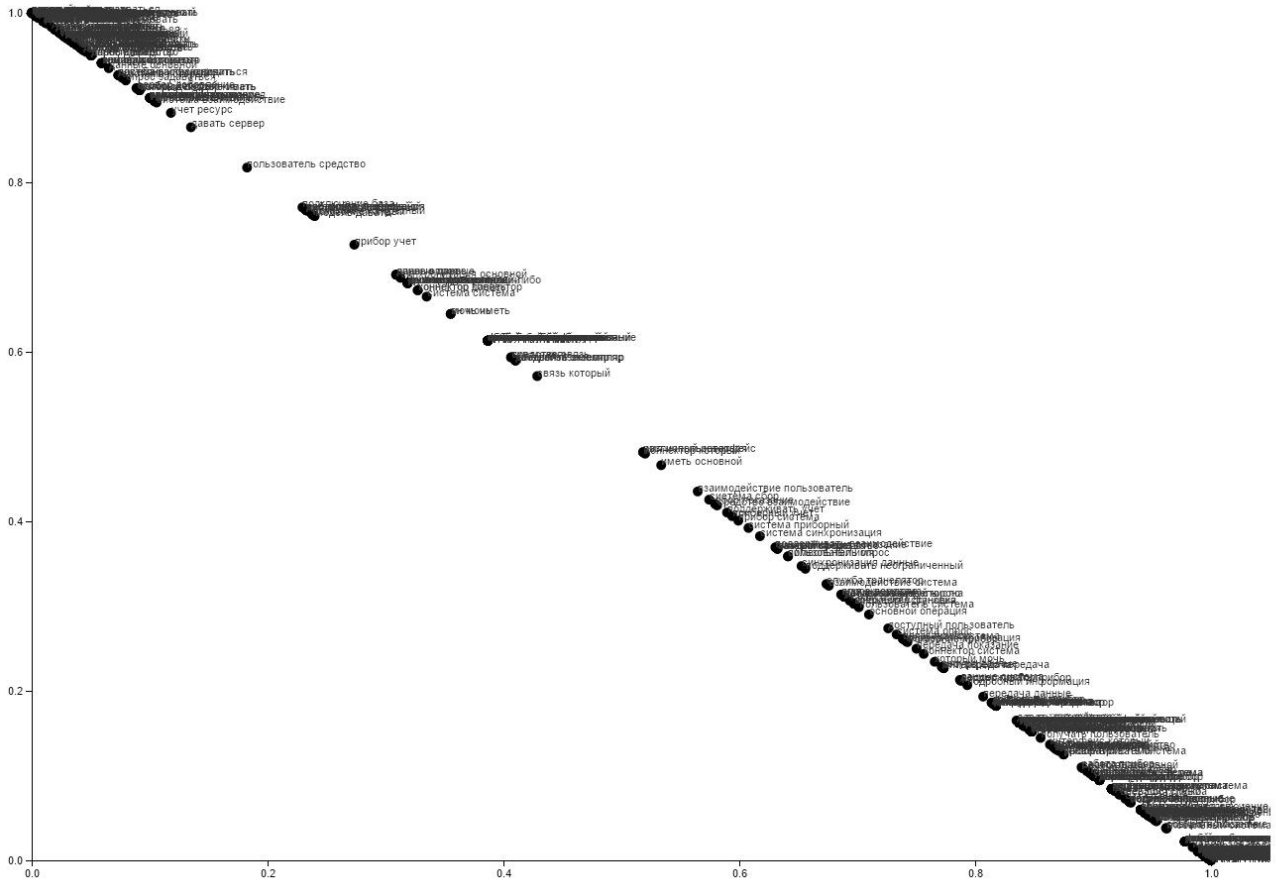

Рисунок 23. Визуализация результата проведенного кластерного анализа для словосочетаний

При подробном рассмотрении результатов можно увидеть, что словосочетания, имеющие наименьшую значимость, в своем большинстве входят в “нижний” или “верхний” кластеры. Значения принадлежности терминов к

центрам обоих кластеров стремятся к равенству, что позволяет достаточно легко выделить такие словосочетания из текста.

5.4 Результат проведения кластерного анализа для слов

Кластерный анализ позволяет точно разделить слова на категории термин/нетермин. На рисунке 24 представлен результат выполнения кластерного анализа для слов текста статьи о приборном учете.

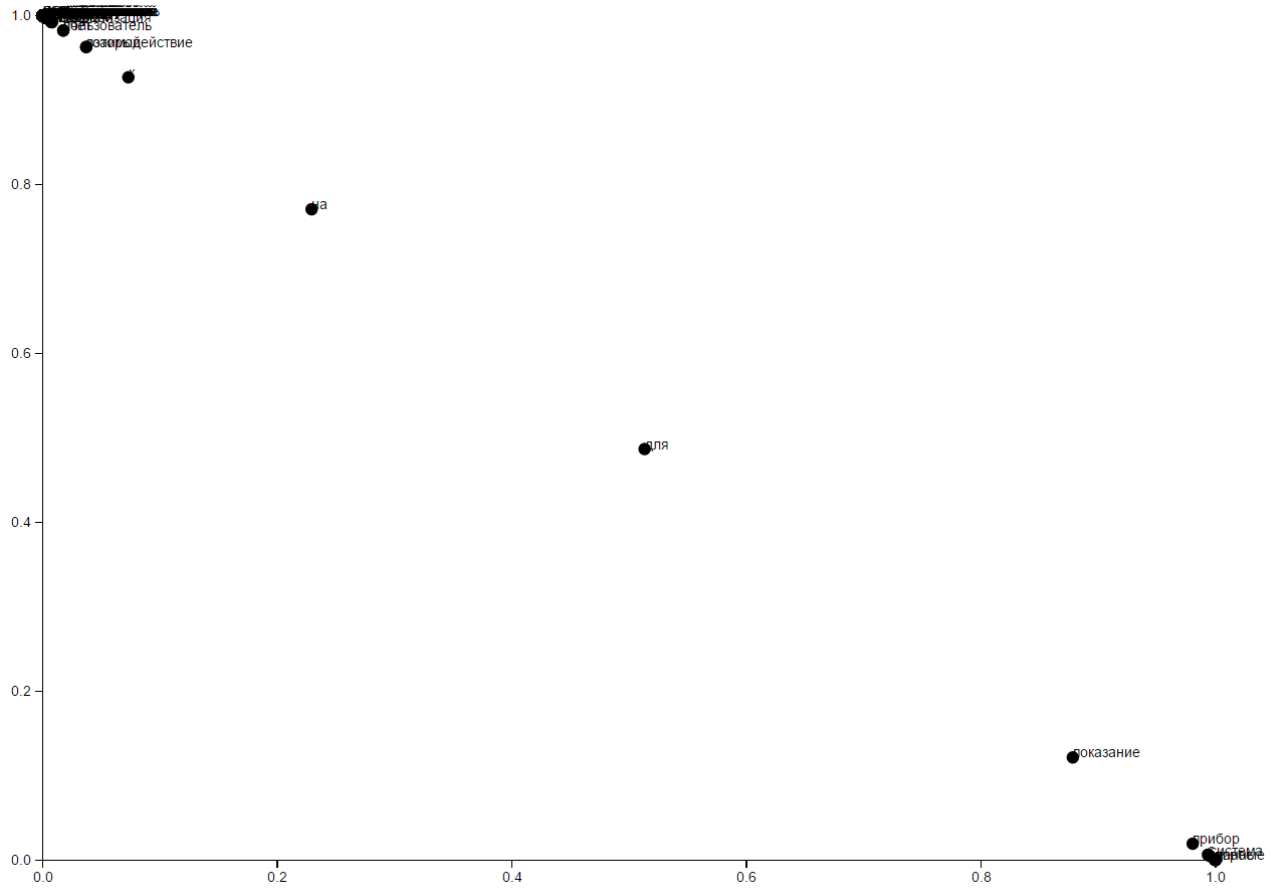

Рисунок 24. Визуализация результатов проведенного кластерного анализа для слов

Среди терминов можно видеть такие слова, как «система», «прибор», «с», «и», «данные», «показание», «опрос». Союзы и предлоги не являются терминами, поэтому они могут быть ужалены из представленного списка путем вызова процедуры морфологического анализа.

Список литературы

1. Гамма, Э. Приемы объектно-ориентированного программирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб. : Питер, 2001. – 344 с.
2. ГОСТ 19.701-90. Единая система конструкторской документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М. : Стандартинформ, 2010. – 8 с.
3. ГОСТ 2.105-95. Единая система конструкторской документации. Общие требования к текстовым документам. – М. : Стандартинформ, 1996. – 9 с.
4. Троелсен, Э. Язык программирования C# 2010 и платформа .NET 4 / Э. Троелсен. – 5-е изд. - М. : Вильямс, 2010. – 1392 с.
5. Фленов, М. Е. Библия C# / М.Е.Фленов. – 2-е изд. – СПб. : БХВ-Петербург, 2011. – 560 с.
6. Mukherjee, S. Thinking in LINQ / S. Mukherjee. – NY : Appress, 2015. – 259 pp.
7. Skeet, J. C# in depth. Third edition / J.Skeet. – Shelter Island, NY : Manning, 2014. – 614 pp.
8. Lee A., Powers E. Ontology-Aided Web Search Assistant / Lee A., Powers E. - [Электронный ресурс] - Режим доступа: [http://protege.stanford.edu/conference/2004/abstracts/Lee.pdf]
9. Dr. Singh M. Ontology Development and Query Retrieval using Protégé Tool / Singh M. - [Электронный ресурс] - Режим доступа: [http://www.mecs-press.org/ijisa/ijisa-v5-n9/IJISA-V5-N9-8.pdf]
10. Knublauch H., Musen M. A., Noy N. F. Creating Semantic Web (OWL) Ontologies with Protege / Knublauch H., Musen M. A., Noy N. F. - [Электронный ресурс] - Режим доступа: [http://iswc2003.semanticweb.org/pdf/Protege-OWL-Tutorial-ISWC03.pdf]

Приложение А

(обязательное)

Текст программы

Модуль API.cs

```
namespace Core
{
    /// <summary>
    /// API для подключения в приложение
    /// </summary>
    public class API
    {
        #region Работа с файлами (1-уровень)
        /// <summary>
        /// Загрузка одного файла
        /// </summary>
        /// <param name="path">Путь до файла</param>
        /// <returns></returns>
        public List<string> LoadFile(string path)
        {
            FileHandler fh = FileHelper.GetFileHandler(path);
            if (fh == null)
                return null;

            List<string> lines;
            fh.ReadFile(out lines);

            return lines;
        }
        /// <summary>
        /// Загрузка нескольких файлов (мультипроцессорная)
        /// </summary>
        /// <param name="paths">Пути до файлов</param>
        public List<FileData> LoadFilesMulticore(List<string> paths)
        {
            Multiprocessor mps = new Multiprocessor();
            mps.MultiprocessorFileRead(paths);
            return mps.FileCache;
        }
        #endregion

        #region Mystem-api (2-уровень)
        /// <summary>
        /// Обработка строк файла программой mystem
        /// </summary>
        /// <param name="fileLines"></param>
        /// <returns></returns>
        public List<Lemm> HandleByMystem(List<string> fileLines)
        {
            MystemProvider mst = new MystemProvider(Guid.NewGuid().ToString());
            return mst.LaunchMystem(fileLines);
        }

        /// <summary>
        /// Обработка конкретного файла с помощью Mystem
        /// </summary>
        /// <param name="filePath">Путь до файла</param>
        /// <returns></returns>
        public MystemData HandleByMystem(string filePath)
        {
```

```
            var fileLines = LoadFile(filePath);
            MystemProvider mst = new MystemProvider(Guid.NewGuid().ToString());
            return new MystemData(filePath, mst.LaunchMystem(fileLines));
        }

        /// <summary>
        /// Мультипроцуессорная обработка файлов программой mystem
        /// </summary>
        /// <param name="dataList"></param>
        /// <returns></returns>
        public List<MystemData> HandleByMystemMulticore(List<FileData> dataList)
        {
            Multiprocessor mps = new Multiprocessor();
            mps.MultiprocessorMystemHandler(dataList);
            return mps.MystemCache;
        }

        /// <summary>
        /// Обработка нескольких файлов с помощью Mystem
        /// </summary>
        /// <param name="filesPaths"></param>
        /// <returns></returns>
        public List<MystemData> HandleByMystemMulticore(List<string> filesPaths)
        {
            Multiprocessor mps = new Multiprocessor();
            mps.MultiprocessorFileRead(filesPaths);
            mps.MultiprocessorMystemHandler(mps.FileCache);
            return mps.MystemCache;
        }

        #endregion

        #region Статистический анализ (3-уровень)
        /// <summary>
        /// Статистический анализ файла для отдельных слов
        /// </summary>
        /// <param name="list"></param>
        /// <returns></returns>
        public StatsAnalysisResult<string> ProvideWordsStatsAnalysis(List<Lemm> list)
        {
            List<string> words = list.GetWords();
            StatsAnalysisResult<string> analysisResult = new StatsAnalysisResult<string>();

            analysisResult.Frequency_Dictionary =
StatisticsAnalysis.GetFrequencyDictionary(words);
            analysisResult.TF_Dictionary =
StatisticsAnalysis.GetTF(analysisResult.Frequency_Dictionary, words.Count);
            analysisResult.TF_IDF_Dictionary =
StatisticsAnalysis.GetTF_IDF(Configuration.CCSize, analysisResult.TF_Dictionary);

            return analysisResult;
        }
        /// <summary>
        /// Статистический анализ для биграмм
        /// </summary>
        /// <param name="list"></param>
        /// <returns></returns>
        public StatsAnalysisResult<WordDigram> ProvideDigramsStatsAnalysis(MystemData data)
        {
            List<string> words = data.List.GetWords();
            StatsAnalysisResult<WordDigram> analysisResult = new
StatsAnalysisResult<WordDigram>();
            var wordsFrequency = StatisticsAnalysis.GetFrequencyDictionary(words);

            analysisResult.Name = data.Name;
            analysisResult.Frequency_Dictionary =
```

```
StatisticsAnalysis.GetDigramFrequenceDictionary(words);
            analysisResult.MutualInformation_Dictionary =
StatisticsAnalysis.CalculateMutualInformation(analysisResult.Frequency_Dictionary,
wordsFrequency, words.Count);
            analysisResult.TScore_Dictionary =
StatisticsAnalysis.CalculateTScore(analysisResult.Frequency_Dictionary, wordsFrequency,
words.Count);
            analysisResult.LogLikelihood_Dictionary =
StatisticsAnalysis.CalculateLogLikelihood(analysisResult.Frequency_Dictionary);

            return analysisResult;
        }
        /// <summary>
        /// Мультипроцессорный статистический анализ для слов
        /// </summary>
        /// <param name="list"></param>
        /// <returns></returns>
        public List<StatsAnalysisResult<string>>
ProvideWordsStatsAnalysisMulticore(List<MystemData> list)
        {
            Multiprocessor mps = new Multiprocessor();
            mps.MultiprocessorWordStatsAnalysis(list);
            return mps.WordsStatsAnalysisCache;
        }
        /// <summary>
        /// Мультипроцессорный статистический анализ для биграмм
        /// </summary>
        /// <param name="list"></param>
        /// <returns></returns>
        public List<StatsAnalysisResult<WordDigram>>
ProvideDigramsStatsAnalysisMulticore(List<MystemData> list)
        {
            Multiprocessor mps = new Multiprocessor();
            mps.MultiprocessorDigramsStatsAnalysis(list);
            return mps.DigramsStatsAnalysisCache;
        }
        #endregion

        #region Морфологический анализ (4-уровень)
        /// <summary>
        /// Морфологический анализ (Исключение)
        /// </summary>
        /// <param name="list"></param>
        /// <param name="excludedTypes"></param>
        /// <returns></returns>
        public MystemData ProvideMorphAnalysis(MystemData data, string[] excludedTypes)
        {
            return MorphologicalAnalysis.ExcludeWordsByType(data, excludedTypes);
        }

        /// <summary>
        /// Мультипроцессорный анализ для Лемм
        /// </summary>
        /// <param name="list"></param>
        /// <param name="excludedTypes"></param>
        /// <returns></returns>
        public List<MystemData> ProvideMorphAnalysisMulticore(List<MystemData> list,
string[] excludedTypes)
        {
            Multiprocessor mps = new Multiprocessor();
            mps.MultiprocessorMorphAnalysis(list, excludedTypes);
            return mps.MultiMorphAnalysisCache;
        }
        #endregion

        #region Кластерный анализ (5-уровень)
```

```
        public double[,] GetDefaultClustersCenters<T>(Dictionary<T, double[]>
statisticDictionary)
        {
            var statisticsElements = statisticDictionary.Values;
            var length = statisticsElements.Max(el => el.Length);
            if (length != statisticsElements.Min(el => el.Length))
                throw new Exception("Differ statistics array lengths!");
            if (length < 1)
                throw new Exception("Statistics is empty!");
            var orderStaticstics = statisticsElements.OrderBy(el => el[0]);
            for (var i = 1; i < length; i++)
            {
                var i1 = i;
                orderStaticstics = orderStaticstics.ThenBy(el => el[i1]);
            }
            var first = orderStaticstics.First();
            var last = orderStaticstics.Last();
            var result = new double[2, length];
            for(var i=0;i<length;i++)
            {
                result[0, i] = first[i];
                result[1, i] = last[i];
            }
            return result;
        }

        /// <summary>
        /// Производит кластерный анализ для одного текста
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="settings"></param>
        /// <returns></returns>
        public ClasterAnalysisResult<T> ProvideClusterAnalysis<T>(ClasterSettings<T>
settings, string name)
        {
            ClasterAnalysis<T> ca = new ClasterAnalysis<T>(settings);
            var result = ca.Clasterize();
            return new ClasterAnalysisResult<T>(name, result);
        }

        /// <summary>
        /// Многопроцессорная обработка для кластерного анализа слов
        /// </summary>
        /// <param name="data"></param>
        /// <returns></returns>
        public List<ClasterAnalysisResult<string>>
ProvideWordClusterAnalysisMulticore(List<ClasterAnalysisData<string>> data)
        {
            Multiprocessor mps = new Multiprocessor();
            mps.MultiprocessorWordClasterAnalysis(data);
            return mps.MultiWordsClusterAnalysisCache;
        }

        /// <summary>
        /// Много процессорная обработка для кластерного анализа биграмм
        /// </summary>
        /// <param name="data"></param>
        /// <returns></returns>
        public List<ClasterAnalysisResult<WordDigram>>
ProvideWordDigramClusterAnalysisMulticore(List<ClasterAnalysisData<WordDigram>> data)
        {
            Multiprocessor mps = new Multiprocessor();
            mps.MultiprocessorWordDigramClasterAnalysis(data);
            return mps.MultiWordDigramsClusterAnalysisCache;
        }
```

```
        /// <summary>
        /// Методы сбора данных для кластерного анализа
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="analysisResult"></param>
        /// <returns></returns>
        public Dictionary<T, double[]> GetDataReady<T>(StatsAnalysisResult<T>
analysisResult)
        {
            List<Dictionary<T, double>> data = new List<Dictionary<T, double>>();

            if (analysisResult.Frequency_Dictionary != null)
                data.Add(analysisResult.Frequency_Dictionary);

            if (analysisResult.LogLikelihood_Dictionary != null)
                data.Add(analysisResult.LogLikelihood_Dictionary);

            if (analysisResult.MutualInformation_Dictionary != null)
                data.Add(analysisResult.MutualInformation_Dictionary);

            if (analysisResult.TF_Dictionary != null)
                data.Add(analysisResult.TF_Dictionary);

            if (analysisResult.TF_IDF_Dictionary != null)
                data.Add(analysisResult.TF_IDF_Dictionary);

            if (analysisResult.TScore_Dictionary != null)
                data.Add(analysisResult.TScore_Dictionary);

            return data.MergeDictionaries();
        }
        #endregion
    }
}
```

Модуль ClasterAnalysis.cs

```
namespace Core
{
    /// <summary>
    /// Класс, реализующий FCM-алгоритм кластеризации
    /// </summary>
    /// <typeparam name="T">Тип ключа для словаря данных</typeparam>
    public class ClasterAnalysis<T>
    {
        private int clustersCount;
        private int dataVectorsCount;
        private int dimensionsCount;
        private int iterationCount;
        private double[,] memberDegree;
        private double epsilon;
        private double fuzziness;
        private double[,] data;
        private double[,] clusterCenters;
        private List<T> wordsDigrams;

        #region Конструкторы
        public ClasterAnalysis(ClasterSettings<T> settings)
        {
            //основные параметры
            epsilon = settings.Epsilon;
            fuzziness = settings.Fuzziness;
            clusterCenters = settings.ClusterCenters;
            iterationCount = settings.IterationCount;

            //данные
```

```
            memberDegree = new double[settings.Data.Count, clusterCenters.GetLength(0)];
            data = new double[settings.Data.Count, settings.Data.First().Value.Length];
            wordsDigrams = FillDataMatrix(settings.Data);
            InitializeMatrix();

            clustersCount = clusterCenters.GetLength(0);
            dataVectorsCount = data.GetLength(0);
            dimensionsCount = data.GetLength(1);
        }
        #endregion

        /// <summary>
        /// Сохранение данных
        /// </summary>
        /// <param name="wordStats"></param>
        private List<T> FillDataMatrix(Dictionary<T, double[]> wordStats)
        {
            wordsDigrams = wordStats.Keys.ToList();
            int n = 0;
            foreach (var item in wordStats)
            {
                double[] stats = item.Value;
                for (int j = 0; j < stats.Length; j++)
                {
                    data[n, j] = stats[j];
                }
                n++;
            }
            return wordsDigrams;
        }

        /// <summary>
        /// Инициализация матрицы принадлежности, с условием, что сумма всех значений не
превышает 1.0
        /// </summary>
        private void InitializeMatrix()
        {
            double s;
            int r = 100, rval;
            Random rnd = new Random();
            for (int i = 0; i < memberDegree.GetLength(0); i++)
            {
                s = 0.0f;
                r = 100;
                for (int j = 1; j < memberDegree.GetLength(1); j++)
                {
                    rval = rnd.Next(1, r);
                    r -= rval;
                    memberDegree[i, j] = rval / 100.0f;
                    s += memberDegree[i, j];
                }
                memberDegree[i, 0] = 1.0 - s;
            }
        }

        /// <summary>
        /// Получение Евклидова расстояния
        /// </summary>
        /// <returns></returns>
        private double CalculateEuclidLength(int i_data, int j_cl)
        {
            double sum = 0f;
            for (int d = 0; d < dimensionsCount; d++)
            {
                sum += Math.Pow(data[i_data, d] - clusterCenters[j_cl, d], 2);
            }
```

```
            return Math.Sqrt(sum);
        }

        /// <summary>
        /// Расчет нового значения матрицы принадлежности
        /// </summary>
        /// <param name="i"></param>
        /// <param name="j"></param>
        /// <returns></returns>
        private double CalculateNewMemberDegreeItem(int i, int j)
        {
            double power = 2 / (fuzziness - 1), sum = 0.0f;
            for (int k = 0; k < clustersCount; k++)
            {
                sum += Math.Pow(CalculateEuclidLength(i, j) / CalculateEuclidLength(i, k),
power);
            }
            return 1.0f / sum;
        }

        /// <summary>
        /// Перерасчет центров кластеров
        /// SUM(k = 1, K)(M_jk)^q * x_k
        /// ___________________________
        /// SUM(k = 1, K)(M_jk)^q
        /// </summary>
        private void CalculateClusterCenters()
        {
            double numerator, denominator;
            double[,] M_jk = new double[dataVectorsCount, clustersCount];

            //расчет M_jk
            for (int i = 0; i < dataVectorsCount; i++)
            {
                for (int j = 0; j < clustersCount; j++)
                {
                    M_jk[i, j] = Math.Pow(memberDegree[i, j], fuzziness);
                }
            }

            //перерасчет центров кластеров
            for (int j = 0; j < clustersCount; j++)
            {
                for (int k = 0; k < dimensionsCount; k++)
                {
                    numerator = 0f;
                    denominator = 0f;
                    for (int i = 0; i < dataVectorsCount; i++)
                    {
                        numerator += M_jk[i, j] * data[i, k];
                        denominator += M_jk[i, j];
                    }
                    clusterCenters[j, k] = numerator / denominator;
                }
            }
        }

        /// <summary>
        /// Перерасчет матрицы принадлежности
        ///                          1
        /// ____________________________________________________
        ///                   |x_i - c_j|
        /// SUM(k = 1, C) (___________________) ^ (2/(q - 1))
        ///                   |x_i - x_k|
        /// </summary>
        /// <returns></returns>
```

```
        private double RecalculateMemberDegree()
        {
            double maxDiff = 0.0f, diff = 0.0f, mdValue;

            for (int j = 0; j < clustersCount; j++)
            {
                for (int i = 0; i < dataVectorsCount; i++)
                {
                    mdValue = CalculateNewMemberDegreeItem(i, j);
                    diff = Math.Abs(mdValue - memberDegree[i, j]);
                    //diff = mdValue - memberDegree[i, j];
                    if (diff > maxDiff)
                        maxDiff = diff;
                    memberDegree[i, j] = mdValue;
                }
            }
            return maxDiff;
        }

        public Dictionary<T, double[]> Clasterize()
        {
            double maxDiff = 0.0f;
            int i = 0;
            do
            {
                CalculateClusterCenters();
                maxDiff = RecalculateMemberDegree();
                i++;
                Console.WriteLine(maxDiff);
            }
            while (maxDiff > epsilon && i < iterationCount);

            //формирование выходного словаря
            var dict = new Dictionary<T, double[]>();
            int n = 0;
            foreach (var key in wordsDigrams)
            {
                double[] analysis = new double[clustersCount];
                for (i = 0; i < clustersCount; i++)
                {
                    analysis[i] = memberDegree[n, i];
                }
                dict.Add(key, analysis);
                n++;
            }
            return dict;
        }
    }
}
```

Модуль CommonClasses.cs

```
namespace Core
{
    #region Дополнительные классы для мультипроцессорной обработки
    /// <summary>
    /// Данные файла
    /// </summary>
    public class FileData
    {
        public List<string> List { get; private set; }
        public string Name { get; private set; }

        public FileData(string name, List<string> list)
        {
            Name = name;
            List = list;
```

```
        }
    }

    /// <summary>
    /// Данные Mystem
    /// </summary>
    public class MystemData
    {
        public List<Lemm> List { get; set; }
        public string Name { get; private set; }
        public MystemData(string name, List<Lemm> list)
        {
            Name = name;
            List = list;
        }
    }
    #endregion

    #region Лемма
    [DataContract]
    public class Lemm
    {
        [DataMember(Name = "text")]
        public string text { get; set; } //слово в тексте
        [DataMember(Name = "analysis")]
        public Analysis[] analysis { get; set; } //анализ

        public override string ToString()
        {
            return text;
        }
    }
    [DataContract]
    public class Analysis
    {
        [DataMember(Name = "lex")]
        public string lex { get; set; }
        [DataMember(Name = "gr")]
        public string gr { get; set; }

        public string wordType
        {
            get
            {
                return GetWordType(gr);
            }
        }

        /// <summary>
        /// Получение части речи
        /// </summary>
        /// <param name="gr"></param>
        /// <returns></returns>
        private string GetWordType(string gr)
        {
            return gr.Split(new[] { ',', '=' },
System.StringSplitOptions.RemoveEmptyEntries)[0];
        }
    }
    #endregion

    #region Биграмма
    /// <summary>
    /// Класс, описывающий биграмму
    /// </summary>
    public class WordDigram
```

```
{
    private const char separator = ' ';
    public string FirstWord { get; private set; }
    public string SecondWord { get; private set; }

    public WordDigram(string firstWord, string secondWord)
    {
        FirstWord = firstWord;
        SecondWord = secondWord;
    }

    public override string ToString()
    {
        return string.Concat(FirstWord, separator, SecondWord);
    }

    public override bool Equals(object obj)
    {
        var dObj = obj as WordDigram;
        if (dObj == null)
            return base.Equals(obj);
        return dObj.FirstWord == this.FirstWord && dObj.SecondWord == SecondWord;
    }

    public override int GetHashCode()
    {
        return FirstWord.GetHashCode() ^ SecondWord.GetHashCode();
    }
}
#endregion

#region Анализы
/// <summary>
/// Хранение данных для статистического анализа
/// </summary>
/// <typeparam name="T">Биграмма, слово (WordDigram, string)</typeparam>
public class StatsAnalysisResult<T>
{
    public string Name { get; set; }
    public Dictionary<T, double> Frequency_Dictionary { get; set; }
    public Dictionary<T, double> TF_Dictionary { get; set; }
    public Dictionary<T, double> TF_IDF_Dictionary { get; set; }
    public Dictionary<T, double> MutualInformation_Dictionary { get; set; }
    public Dictionary<T, double> TScore_Dictionary { get; set; }
    public Dictionary<T, double> LogLikelihood_Dictionary { get; set; }

    public StatsAnalysisResult(string name = "")
    {
        Name = name;
    }
}
#endregion

#region Кластерный анализ
/// <summary>
/// Класс, содержащий настройки проводимого кластерного анализа (для UI)
/// </summary>
public class ClasterSettings<T>
{
    public double[,] ClusterCenters { get; private set; }
    public double Epsilon { get; private set; }
    public double Fuzziness { get; private set; }
    public int IterationCount { get; private set; }
    public Dictionary<T, double[]> Data { get; private set; }

    public ClasterSettings(double[,] cts, double e, double f, int ic, Dictionary<T,
```

```
double[]> data)
        {
            ClusterCenters = cts;
            Epsilon = e;
            Fuzziness = f;
            IterationCount = ic;
            Data = data;
        }
    }

    /// <summary>
    /// Класс, описывающий набор параметров (Settings) для проведения кластерного анализа с
указанием имени файла (Name)
    /// </summary>
    /// <typeparam name="T"></typeparam>
    public class ClasterAnalysisData<T>
    {
        public string Name { get; private set; }
        public ClasterSettings<T> Settings { get; private set; }

        public ClasterAnalysisData(string name, ClasterSettings<T> settings)
        {
            Name = name;
            Settings = settings;
        }
    }

    /// <summary>
    /// Класс, описывающий результат выполнения кластерного анализа
    /// </summary>
    /// <typeparam name="T"></typeparam>
    public class ClasterAnalysisResult<T>
    {
        public string Name { get; private set; }
        public Dictionary<T, double[]> Result { get; private set; }

        public ClasterAnalysisResult(string name, Dictionary<T, double[]> result)
        {
            Name = name;
            Result = result;
        }
    }
    #endregion
}
```

Модуль Configuration.cs

```
namespace Core
{
    public static class Configuration
    {
        public static int CCSize = 100000;
        public static class WordType
        {
            //Типы частей речи
            /// <summary>
            /// Прилагательное
            /// </summary>
            public const string adjective = "A";
            /// <summary>
            /// Наречие
            /// </summary>
            public const string adverb = "ADV";
            /// <summary>
            /// Наречие-местоимение
            /// </summary>
            public const string adverb_pronoun = "ADVPRO";
```

```
            /// <summary>
            /// Прилагательное-числительное
            /// </summary>
            public const string adjective_numeral = "ANUM";
            /// <summary>
            /// Прилагательное-местоимение
            /// </summary>
            public const string adjective_pronoun = "APRO";
            /// <summary>
            /// Сложное слово
            /// </summary>
            public const string composite = "COM";
            /// <summary>
            /// Союз
            /// </summary>
            public const string conjunction = "CONJ";
            /// <summary>
            /// Междометие
            /// </summary>
            public const string interjection = "INTJ";
            /// <summary>
            /// ЧИслительное
            /// </summary>
            public const string numeral = "NUM";
            /// <summary>
            /// Частица
            /// </summary>
            public const string particle = "PART";
            /// <summary>
            /// Предлог
            /// </summary>
            public const string preposition = "PR";
            /// <summary>
            /// Существительное
            /// </summary>
            public const string noun = "S";
            /// <summary>
            /// Существительное-местоимение
            /// </summary>
            public const string noun_pronoun = "SPRO";
            /// <summary>
            /// Глагол
            /// </summary>
            public static string verb = "V";
        }
    }
}
```

Модуль MorphologicalAnalysis.cs

```
namespace Core
{
    public static class MorphologicalAnalysis
    {
        /// <summary>
        /// Получение лемм по типу речи
        /// </summary>
        /// <param name="lemms"></param>
        /// <param name="type"></param>
        /// <returns></returns>
        public static List<Lemm> GetWordsByType(List<Lemm> lemms, string type)
        {
            return lemms.Where(i => i.analysis.Length > 0 && i.analysis[0].wordType ==
type).ToList();
        }

        /// <summary>
```

```
        /// Исключение лемм по типу речи
        /// </summary>
        /// <param name="lemms"></param>
        /// <param name="type"></param>
        /// <returns></returns>
        public static List<Lemm> ExcludeWordsByType(List<Lemm> lemms, params string[]
types)
        {
            return lemms.Where(i => i.analysis.Length > 0 && !i.analysis.Any(j =>
types.Contains(j.wordType))).ToList();
        }

        /// <summary>
        /// Исключение лемм по типу речи
        /// </summary>
        /// <param name="lemms"></param>
        /// <param name="type"></param>
        /// <returns></returns>
        public static MystemData ExcludeWordsByType(MystemData mstData, params string[]
types)
        {
            mstData.List = mstData.List.Where(i => i.analysis.Length > 0 &&
!i.analysis.Any(j => types.Contains(j.wordType))).ToList();
            return mstData;
        }
    }
}
```

Модуль Multiprocessor.cs

```
namespace Core
{
    public class Multiprocessor
    {
        private Thread[] threads; //потоки
        private List<FileData> multiFileCache; //кэш с данными
        private List<MystemData> multiMystemCache; //кэш с данными mystem
        private List<StatsAnalysisResult<string>> multiWordsStatsAnalysisCache; //кэш для
хранения результата по статистическому анализу для слов
        private List<StatsAnalysisResult<WordDigram>> multiDigramsStatsAnalysisCache; //кэш
для хранения результата по статистическому анализу для биграмм
        private List<ClasterAnalysisResult<string>> multiWordsClusterAnalysisCache; //кэш
для хранения результата кластерного анализа слов
        private List<ClasterAnalysisResult<WordDigram>> multiDigramsClusterAnalysisCache;
//кэш для хранения результата кластерного анализа биграмм
        private List<MystemData> multiMorphAnalysisCache; //кэш для хранения результата
морфологического анализа лемм

        public List<FileData> FileCache
        {
            get { return multiFileCache; }
            private set { multiFileCache = value; }
        }
        public List<MystemData> MystemCache
        {
            get { return multiMystemCache; }
            private set { multiMystemCache = value; }
        }
        public List<StatsAnalysisResult<string>> WordsStatsAnalysisCache
        {
            get { return multiWordsStatsAnalysisCache; }
            private set { multiWordsStatsAnalysisCache = value; }
        }
        public List<StatsAnalysisResult<WordDigram>> DigramsStatsAnalysisCache
        {
            get { return multiDigramsStatsAnalysisCache; }
            private set { multiDigramsStatsAnalysisCache = value; }
```

```
        }

        public List<ClasterAnalysisResult<string>> MultiWordsClusterAnalysisCache
        {
            get { return multiWordsClusterAnalysisCache; }
            private set { multiWordsClusterAnalysisCache = value; }
        }

        public List<ClasterAnalysisResult<WordDigram>> MultiWordDigramsClusterAnalysisCache
        {
            get { return multiDigramsClusterAnalysisCache; }
            private set { multiDigramsClusterAnalysisCache = value; }
        }

        public List<MystemData> MultiMorphAnalysisCache
        {
            get { return multiMorphAnalysisCache; }
            private set { multiMorphAnalysisCache = value; }
        }

        public Multiprocessor()
        {
            multiFileCache = new List<FileData>();
            multiMystemCache = new List<MystemData>();
            multiWordsStatsAnalysisCache = new List<StatsAnalysisResult<string>>();
            multiDigramsStatsAnalysisCache = new List<StatsAnalysisResult<WordDigram>>();
            multiWordsClusterAnalysisCache = new List<ClasterAnalysisResult<string>>();
            multiDigramsClusterAnalysisCache = new
List<ClasterAnalysisResult<WordDigram>>();
            multiMorphAnalysisCache = new List<MystemData>();
        }

        private void _cleanCache<T>(List<T> cache)
        {
            cache.Clear();
        }

        #region functions for parallel computing
        private void _readFile(string path)
        {
            FileHandler fh = FileHelper.GetFileHandler(path);
            if (fh == null)
                return;

            List<string> lines;
            fh.ReadFile(out lines);
            multiFileCache.Add(new FileData(path, lines));
        }

        private void _runMystem(FileData data, string index)
        {
            MystemProvider mp = new MystemProvider(index);
            List<Lemm> list = mp.LaunchMystem(data.List);
            multiMystemCache.Add(new MystemData(data.Name, list));
        }

        private void _provideWordsStatsAnalysis(MystemData data)
        {
            List<string> words = data.List.GetWords();
            StatsAnalysisResult<string> analysisResult = new StatsAnalysisResult<string>();

            analysisResult.Frequency_Dictionary =
StatisticsAnalysis.GetFrequencyDictionary(words);
            analysisResult.TF_Dictionary =
StatisticsAnalysis.GetTF(analysisResult.Frequency_Dictionary, words.Count);
            analysisResult.TF_IDF_Dictionary =
```

```
StatisticsAnalysis.GetTF_IDF(Configuration.CCSize, analysisResult.TF_Dictionary);

            multiWordsStatsAnalysisCache.Add(analysisResult);
        }
        private void _provideDigramsStatsAnalysis(MystemData data)
        {
            List<string> words = data.List.GetWords();
            StatsAnalysisResult<WordDigram> analysisResult = new
StatsAnalysisResult<WordDigram>();
            var wordsFrequency = StatisticsAnalysis.GetFrequencyDictionary(words);

            analysisResult.Name = data.Name;
            analysisResult.Frequency_Dictionary =
StatisticsAnalysis.GetDigramFrequenceDictionary(words);
            analysisResult.MutualInformation_Dictionary =
StatisticsAnalysis.CalculateMutualInformation(analysisResult.Frequency_Dictionary,
wordsFrequency, words.Count);
            analysisResult.TScore_Dictionary =
StatisticsAnalysis.CalculateTScore(analysisResult.Frequency_Dictionary, wordsFrequency,
words.Count);
            analysisResult.LogLikelihood_Dictionary =
StatisticsAnalysis.CalculateLogLikelihood(analysisResult.Frequency_Dictionary);

            multiDigramsStatsAnalysisCache.Add(analysisResult);
        }

        private void _provideWordClusterAnalysis(ClasterAnalysisData<string> data)
        {
            ClasterAnalysis<string> ca = new ClasterAnalysis<string>(data.Settings);
            multiWordsClusterAnalysisCache.Add(new ClasterAnalysisResult<string>(data.Name,
ca.Clasterize()));
        }

        private void _provideDigramClusterAnalysis(ClasterAnalysisData<WordDigram> data)
        {
            ClasterAnalysis<WordDigram> ca = new
ClasterAnalysis<WordDigram>(data.Settings);
            multiDigramsClusterAnalysisCache.Add(new
ClasterAnalysisResult<WordDigram>(data.Name, ca.Clasterize()));
        }

        private void _provideMorphAnalysis(MystemData data, string[] excludedTypes)
        {
            var result = MorphologicalAnalysis.ExcludeWordsByType(data, excludedTypes);
            multiMorphAnalysisCache.Add(result);
        }
        #endregion

        public void MultiprocessorMorphAnalysis(List<MystemData> list, string[]
excludedTypes)
        {
            _cleanCache(multiMorphAnalysisCache);
            if (list != null && list.Count > 0)
            {
                threads = new Thread[list.Count];
                for (int i = 0; i < threads.Length; i++)
                {
                    MystemData data = list[i];
                    threads[i] = new Thread(() => _provideMorphAnalysis(data,
excludedTypes));
                    threads[i].Start();
                }
                foreach (Thread th in threads)
                {
                    th.Join();
                }
```

```
                threads = null;
            }
        }

        public void
MultiprocessorWordDigramClasterAnalysis(List<ClasterAnalysisData<WordDigram>> list)
        {
            _cleanCache(multiWordsClusterAnalysisCache);
            if (list != null && list.Count > 0)
            {
                threads = new Thread[list.Count];
                for (int i = 0; i < threads.Length; i++)
                {
                    ClasterAnalysisData<WordDigram> data = list[i];
                    threads[i] = new Thread(() => _provideDigramClusterAnalysis(data));
                    threads[i].Start();
                }
                foreach (Thread th in threads)
                {
                    th.Join();
                }
                threads = null;
            }
        }

        public void MultiprocessorWordClasterAnalysis(List<ClasterAnalysisData<string>>
list)
        {
            _cleanCache(multiWordsClusterAnalysisCache);
            if (list != null && list.Count > 0)
            {
                threads = new Thread[list.Count];
                for (int i = 0; i < threads.Length; i++)
                {
                    ClasterAnalysisData<string> data = list[i];
                    threads[i] = new Thread(() => _provideWordClusterAnalysis(data));
                    threads[i].Start();
                }
                foreach (Thread th in threads)
                {
                    th.Join();
                }
                threads = null;
            }
        }

        public void MultiprocessorWordStatsAnalysis(List<MystemData> list)
        {
            _cleanCache(multiWordsStatsAnalysisCache);
            if (list != null && list.Count > 0)
            {
                threads = new Thread[list.Count];
                for (int i = 0; i < threads.Length; i++)
                {
                    MystemData data = list[i];
                    threads[i] = new Thread(() => _provideWordsStatsAnalysis(data));
                    threads[i].Start();
                }
                foreach (Thread th in threads)
                {
                    th.Join();
                }
                threads = null;
            }
        }
```

```
        public void MultiprocessorDigramsStatsAnalysis(List<MystemData> list)
        {
            _cleanCache(multiWordsStatsAnalysisCache);
            if (list != null && list.Count > 0)
            {
                threads = new Thread[list.Count];
                for (int i = 0; i < threads.Length; i++)
                {
                    MystemData data = list[i];
                    threads[i] = new Thread(() => _provideDigramsStatsAnalysis(data));
                    threads[i].Start();
                }
                foreach (Thread th in threads)
                {
                    th.Join();
                }
                threads = null;
            }
        }

        public void MultiprocessorMystemHandler(List<FileData> list)
        {
            _cleanCache(multiMystemCache);
            if (list != null && list.Count > 0)
            {
                threads = new Thread[list.Count];
                for (int i = 0; i < threads.Length; i++)
                {
                    FileData data = list[i];
                    threads[i] = new Thread(() => _runMystem(data,
Guid.NewGuid().ToString()));
                    threads[i].Start();
                }
                foreach (Thread th in threads)
                {
                    th.Join();
                }
                threads = null;
            }
        }

        public void MultiprocessorFileRead(List<string> paths)
        {
            _cleanCache(multiFileCache);
            paths = FileHelper.CheckFiles(paths);
            if (paths != null && paths.Count > 0)
            {
                threads = new Thread[paths.Count];
                for (int i = 0; i < threads.Length; i++)
                {
                    string path = paths[i];
                    threads[i] = new Thread(new ThreadStart(() => _readFile(path)));
                    threads[i].Start();
                }
                foreach (Thread th in threads)
                {
                    th.Join();
                }
                threads = null;
            }
        }
    }
}
```

Модуль MystemProvider.cs

```
namespace Core
```

```
{
    /// <summary>
    /// Класс, обеспечивающий запуск mystem.exe
    /// </summary>
    public class MystemProvider
    {
        private string mystemPath;
        private string index;
        private string inputFileName;
        private string outputFileName;

        public MystemProvider(string index, string mystemPath = @"mystem\mystem.exe")
        {
            this.mystemPath = mystemPath;
            this.index = index;
            inputFileName = "tmp_input_" + index + ".txt";
            outputFileName = "tmp_output_" + index + ".json";
        }

        /// <summary>
        /// Парсинг json-файла результата выполнения mystem
        /// </summary>
        /// <param name="srdr"></param>
        /// <returns></returns>
        private List<Lemm> GetMystemResult(StreamReader srdr)
        {
            List<Lemm> lemms = new List<Lemm>();
            DataContractJsonSerializer ser = new DataContractJsonSerializer(typeof(Lemm));

            string line = srdr.ReadLine();
            while (!srdr.EndOfStream)
            {
                Lemm obj = (Lemm)ser.ReadObject(new
MemoryStream(Encoding.UTF8.GetBytes(line)));
                if (obj.analysis != null)
                    lemms.Add(obj);
                line = srdr.ReadLine();
            }
            srdr.Close();

            return lemms;
        }

        /// <summary>
        /// Запуск mystem.exe
        /// </summary>
        public List<Lemm> LaunchMystem(List<string> lines, string flags = "-cgin --format
json")
        {
            Console.WriteLine(inputFileName);
            FileHelper.WriteFile(lines, inputFileName);

            Process process = new Process()
            {
                StartInfo = new ProcessStartInfo()
                {
                    Arguments = String.Format("{0} {1} {2}", flags, inputFileName,
outputFileName),
                    FileName = mystemPath,
                    UseShellExecute = false,
                    CreateNoWindow = true
                }
            };
            process.Start();
            process.WaitForExit();
```

```
            List<Lemm> lemms = GetMystemResult(new StreamReader(outputFileName));

            FileHelper.DeleteFile(inputFileName);
            FileHelper.DeleteFile(outputFileName);

            return lemms;
        }
    }
}
```

Модуль StatisticsAnalysis.cs

```
namespace CoreLib
{
    /// <summary>
    /// Статистический анализ текстов
    /// </summary>
    public static class StatisticsAnalysis
    {
        #region Однословия
        /// <summary>
        /// Получение частотного словаря (слова)
        /// </summary>
        /// <param name="words"></param>
        /// <returns></returns>
        public static Dictionary<string, double> GetFrequencyDictionary(List<string> words)
        {
            return
                (
                    from i in words
                    group i by i into grp
                    select new { word = grp.Key, count = (double) grp.Count() }
                )
                .ToDictionary(i => i.word, i => i.count);
        }

        /// <summary>
        /// Метод TF (Абсолютная частота встречаемости слова в тексте)
        /// </summary>
        /// <param name="words"></param>
        /// <param name="count"></param>
        /// <returns></returns>
        public static Dictionary<string, double> GetTF(Dictionary<string, double> words,
int count)
        {
            return words.ToDictionary(i => i.Key, i => ((double)i.Value / count));
        }

        /// <summary>
        /// Метод TFxIDF (наиболее статистически значимые однословия)
        /// </summary>
        /// <param name="ccSize"></param>
        /// <param name="tf_dictionary"></param>
        /// <param name="docNumber"></param>
        /// <returns></returns>
        public static Dictionary<string, double> GetTF_IDF(int ccSize, Dictionary<string,
double> tf_dictionary, int docNumber = 1)
        {
            return tf_dictionary.ToDictionary(i => i.Key, i => i.Value *
Math.Log((double)(ccSize - docNumber) / docNumber));
        }
        #endregion

        #region Биграммы (Digrams)
        /// <summary>
        /// Получение частотного словаря биграмм
        /// </summary>
```

```
        /// <param name="wordList"></param>
        /// <returns></returns>
        public static Dictionary<WordDigram, double>
GetDigramFrequenceDictionary(List<string> wordList)
        {
            var result = new Dictionary<WordDigram, double>();
            List<string> digram;
            for (var i = 0; i < wordList.Count - 1; i++)
            {
                digram = wordList.Skip(i).Take(2).ToList();
                var digramKey = new WordDigram(digram[0], digram[1]);
                if (!result.ContainsKey(digramKey))
                    result.Add(digramKey, GetNgramFrequence(wordList, digram));
            }
            return result;
        }

        /// <summary>
        /// Метод Mutual Information (поиск наиболее статистически значимых двусловий)
        /// </summary>
        /// <param name="frequencyDiagram"></param>
        /// <param name="frequencyDictionary"></param>
        /// <param name="wordsCount"></param>
        /// <returns></returns>
        public static Dictionary<WordDigram, double>
CalculateMutualInformation(Dictionary<WordDigram, double> frequencyDiagram,
            Dictionary<string, double> frequencyDictionary, int wordsCount)
        {
            var result = new Dictionary<WordDigram, double>();
            foreach (var pair in frequencyDiagram)
            {
                var key = pair.Key;
                //частота первого слова
                var fx = frequencyDictionary[key.FirstWord];
                //частота второго слова
                var fy = frequencyDictionary[key.SecondWord];
                result.Add(key, Math.Log(
                    (pair.Value * wordsCount) / fx * fy
                    , 2));
            }
            return result;
        }

        /// <summary>
        /// Метод TScore ()
        /// </summary>
        /// <param name="frequencyDiagram"></param>
        /// <param name="frequencyDictionary"></param>
        /// <param name="wordsCount"></param>
        /// <returns></returns>
        public static Dictionary<WordDigram, double> CalculateTScore(Dictionary<WordDigram,
double> frequencyDiagram,
            Dictionary<string, double> frequencyDictionary, int wordsCount)
        {
            var result = new Dictionary<WordDigram, double>();
            foreach (var pair in frequencyDiagram)
            {
                var key = pair.Key;
                //частота первого слова
                var fx = frequencyDictionary[key.FirstWord];
                //частота второго слова
                var fy = frequencyDictionary[key.SecondWord];
                result.Add(key,
                    ((pair.Value - (fx * fy / (double)wordsCount)) / (pair.Value *
pair.Value))
                    );
```

```
            }
            return result;
        }

        /// <summary>
        /// Метод Log-Likelihood (выявление наиболее статистически значимых двусловий)
        /// </summary>
        /// <param name="frequencyDiagram"></param>
        /// <returns></returns>
        public static Dictionary<WordDigram, double>
CalculateLogLikelihood(Dictionary<WordDigram, double> frequencyDiagram)
        {
            var result = new Dictionary<WordDigram, double>();
            foreach (var pair in frequencyDiagram)
            {
                var a = pair.Value;
                var b = frequencyDiagram
                    .Where(el => el.Key.FirstWord == pair.Key.FirstWord &&
el.Key.SecondWord != pair.Key.SecondWord)
                    .Sum(el => el.Value);
                var c = frequencyDiagram
                    .Where(el => el.Key.FirstWord != pair.Key.FirstWord &&
el.Key.SecondWord == pair.Key.SecondWord)
                    .Sum(el => el.Value);
                var d = frequencyDiagram
                    .Where(el => el.Key.FirstWord != pair.Key.FirstWord &&
el.Key.SecondWord != pair.Key.SecondWord)
                    .Sum(el => el.Value);

                var value = a * Math.Log(a + 1)
                    + b * Math.Log(b + 1)
                    + c * Math.Log(c + 1)
                    + d * Math.Log(d + 1)
                    - (a + b) * Math.Log(a + b + 1)
                    - (a + c) * Math.Log(a + c + 1)
                    - (b + d) * Math.Log(b + d + 1)
                    - (c + d) * Math.Log(c + d + 1)
                    + (a + b + c + d) * Math.Log(a + b + c + d + 1);
                result.Add(pair.Key, value);
            }
            return result;
        }
        #endregion

        #region N-граммы
        /// <summary>
        /// Частота N-граммы в предложении
        /// </summary>
        /// <param name="wordList"></param>
        /// <param name="ngramList"></param>
        /// <returns></returns>
        public static int GetNgramFrequence(List<string> wordList, List<string> ngramList)
        {
            if (wordList == null || ngramList == null)
                return 0;
            if (wordList.Count < ngramList.Count || ngramList.Count <= 1)
                return 0;

            var result = 0;
            //Возможна также реализация через IndexOf.
            var startNgramWord = ngramList[0];
            for (var i = 0; i < wordList.Count - ngramList.Count + 1; i++)
            {
                //если не первое слово энГраммы не совпадает с текущим в списке слов,
катимся дальше
                if (wordList[i] != startNgramWord)
```

```
                    continue;

                var isEqual = true;
                //начинаем цикл со 2 слова, т.к. первое проверили на предыдущем шаге
                for (var j = 1; j < ngramList.Count; j++)
                {
                    if (wordList[i + j] != ngramList[j])
                    {
                        isEqual = false;
                        continue;
                    }
                }
                if (isEqual) result++;
            }
            return result;
        }
        #endregion
    }

    /// <summary>
    /// Extension-методы для статистического анализа
    /// </summary>
    public static class StatisticsAnalysisExtensions
    {
        /// <summary>
        /// Возвращает словарь со всеми проведенными исследованиями
        /// </summary>
        /// <param name="dictionaries"></param>
        /// <returns></returns>
        public static Dictionary<T, double[]> MergeDictionaries<T>(this List<Dictionary<T,
double>> dictionaries)
        {
            Dictionary<T, double[]> mergedDictionary = new Dictionary<T, double[]>();

            foreach (var key in dictionaries[0].Keys)
            {
                List<double> metrics = new List<double>();
                foreach (var dic in dictionaries)
                {
                    double val;
                    if (dic.TryGetValue(key, out val))
                        metrics.Add(dic[key]);
                    else
                        metrics.Add(0.0f);
                }
                mergedDictionary.Add(key, metrics.ToArray());
            }
            return mergedDictionary;
        }

        /// <summary>
        /// Общее количество слов
        /// </summary>
        /// <param name="words"></param>
        /// <returns></returns>
        public static int GetWordsCount(this Dictionary<string, int> words)
        {
            return words.Sum(i => i.Value);
        }

        /// <summary>
        /// Получить список слов на основе List<Lemm>
        /// </summary>
        /// <param name="list">Данные mystem</param>
        /// <returns></returns>
        public static List<string> GetWords(this List<Lemm> list)
```

```
        {
            return list.Select(el =>
            {
                if (el.analysis.Length == 0)
                    return el.text;
                return el.analysis[0].lex;
            }).ToList();
        }
    }
}
```