САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Абубакиров Азат Радисович

Выпускная квалификационная работа бакалавра

Автоматизация сверки и устранения дубликатов в персональных данных

Направление 010300

Фундаментальная информатика и информационные технологии

Научный руководитель,

старший преподаватель

Севрюков С. Ю.

Санкт-Петербург

2017

СОДЕРЖАНИЕ

Введение в проблематику

Во время работы с информационной системой почти всегда возникает ситуация, в которой пользователю нужно ввести какие-либо данные вручную. Опечатки, изначально некорректные данные, невнимательное заполнение полей в форме — все это приводит к тому, что в данных возникают так называемые дубликаты — две или более записи, которые описывают одну сущность. Наиболее наглядно проблема наличия дубликатов выражена в работе с персональными данными.

Специфика процесса накопления персональных данных подробно описана в статье [1]. В течение жизни у человека появляются и исчезают такие идентификаторов личности, как номер бирки новорождённого, номер свидетельства о рождении, номера паспортов (российских и заграничных), номера страховых полисов и т.д. В различных учреждениях идентификация пациента происходит на основе своих собственных политик, следовательно при агрегации данных из различных источников возникает задача сопоставления данных, которые описывают одну и ту же сущность. Помимо того что данные представлены в различных форматах, не стоит забывать о человеческом факторе: записи могут содержать ошибки и опечатки. Кроме вышеописанных сложностей также существует проблема подмена идентичности. Ее причиной может стать как пациент (воспользовался чужим удостоверением личности), так и врач (взял биоматериал одного человека, а в документах указал другого). В качестве решения этой проблемы автор статьи предлагает предоставление доступа пациентам к информации о том, какие медицинские услуги им были оказаны. Данная статья дает весьма полное представление о том, какие проблемы возникают при идентификации личности, в чем их причины и что стоит обязательно учесть при создании таких информационных

систем, как медицинские.

Помимо сложностей, возникающих из–за особенностей приобретения идентификаторов, авторы статьи [2] обозначают еще одну немаловажную проблему, с которой сталкиваются специалисты по работе с данными — приватность персональных данных. Из-за юридических ограничений далеко не всегда есть возможность идентифицировать человека по его уникальному номеру (аналогом в РФ могут служить серия и номер паспорта). В таких ситуациях идентификация происходит на основе комбинации таких признаков, как имя, фамилия, пол, дата рождения и т.д. Однако, если даже у одного исследовательского центра есть доступ к уникальному идентификатору пациента, то они могут быть использованы только в рамках собственных исследований. Иными словами, при агрегации данных с другими источниками уникальные идентификаторы не могут быть переданы другим организациям, что существенно снижает полезность таких идентификаторов.

Постановка задачи

Целью данной работы является повышение качества данных, предоставленных информационно-аналитическим отделом по медицинской помощи населению Медицинского Информационного Аналитического Центра [3] (далее МИАЦ). Для достижения поставленной цели были **поставлены следующие задачи**:

1. ознакомиться с литературой, посвященной рассматриваемой проблематике, и существующими решениями;
2. проанализировать и выявить типы ошибок в данных;
3. разработать методы для устранения выявленных ошибок;
4. разработать алгоритм для нахождения дубликатов;
5. применить алгоритм на различных данных и проанализировать результаты;
6. определить подходы к устранению дубликатов;
7. оформить использованный программный код в виде библиотеки.

Разработка библиотеки необходима для удобного использования написанного программа кода при внедрении реализации разработанного алгоритма в информационную систему МИАЦ.

Глава 1. Обзор литературы и существующих решений

1.1. Литература

Как известно, перед тем как начать какую–либо работу с данными, необходимо увеличить качество этих данных. Авторы статей [4], [5], в своих исследованиях выделяют несколько типичных задач, которые нужно решить входе предобработки данных. Ошибки в записях, которые можно классифицировать как опечатки, могут быть выявлены, например, с помощью подсчета расстояния Левенштейна, фонетических алгоритмов и т.д. . Однако данный подход не сможет обнаружить то, что значения полей стоят не на своем месте. Например, дата рождения и дата выдача паспорта могут быть перепутаны местами. Для этого в систему нужно заложить возможность такого типа ошибок, также реализовать методы их устранения. К сожалению, универсальных методов в данном вопросе не существует, и для каждой задачи необходимо искать свое собственное решение, которое можно получить путем адаптации и комбинации классических фундаментальных подходов.

В статье [6] приведён обзор существующих методы и подходов для решения задачи поиска дубликатов.

В первой части статьи рассматриваются методы, направленные на сравнение двух записей и на определение того, ссылаются ли они на одну сущность или нет. Здесь указываются следующие подходы:

- Точное совпадение записей или их отдельных частей.
- Вычисление различных расстояний между записями:
    - Расстояние Левенштейна.
    - Расстояние с учетом расположения букв на клавиатуре. Данный метод эффективен, если в данных присутствует

большое количество опечаток.

- Косинусное расстояние. В этом случае сложность заключается в том, каким образом векторизовать записи.

- Конвертация записей с помощью фонетических алгоритмов.
- Использование сторонних баз знаний из предметной области. Например, база данных соответствий полных и кратких русских имен.

Вторая часть описывает способы применения методов из первой части к базе, в которой более двух записей.

- Полный перебор (каждая запись сравнивается со всеми другими). Ввиду своей сложности $\boldsymbol{O}(\boldsymbol{n}^2)$, где $\boldsymbol{n}$ — количество записей, полный перебор имеет смысл применять только на относительно небольших данных.
- Предварительная сортировка данных.
- Построение иерархии записей. Если в записи есть информация об адресе пользователя, то все записи можно разбить на вложенные друг в друга группы относительно городов, районов, улиц.

Также в статье рассматривается вопрос о поддержании созданной структуры в базе данных. Если на основе дубликатов не была создана одна итоговая запись, то при добавлении новой записи будет более эффективно сравнивать ее не со всеми записями, а лишь с одним представителем из каждой группы (кластера) дубликатов (например, с последним добавленным в базу).

Если принять во внимание некоторые особенности датасета, то можно существенно улучшить алгоритм. В статье [7] исследователи из Microsoft провели ряд экспериментов для того, чтобы выяснить влияние ограничения на количество дубликатов на качество алгоритма. В качестве

датасетов рассматривались базы данных IMDB и iTunes. Перед авторами стояла следующая задача: найти пары записей из обеих баз данных, которые относятся к одному и тому же фильму. Внутри каждой из баз все записи были уникальны. Следовательно, из этого выводится естественное ограничение: каждой записи из одной базы данных взаимно однозначно соответствует одна или ноль записей из другой базы. Авторы статьи реализовали два вида алгоритмов: первая группа учитывала ограничение, вторая не учитывала. Итоги экспериментов были следующие: алгоритмы из первой группы были более независимы от функции сравнения двух записей. Таким образом, если данные обладают свойством “один к одному”, то нет необходимости требовать от функции сравнения максимальной точности.

В статье [8] рассматривается задача поиска точных и приближенных копий записей о пациентах. Метод решения основан на сравнении строк и кластеризации записей. Первый этап заключается в стандартизации и индексировании записей. В каждой записи выделяются ключевые поля: имя, фамилия, девичья фамилия (если есть), пол, дата рождения. Над полями проводится ряд преобразований: перевод всех букв в верхний регистр, удаление знаков препинания и т.д. Затем создаются прямые и обратные индексы для каждой записи. Для того чтобы уменьшить количество сравнений применяется алгоритм кластеризации. Внутри каждого кластера вычисляется мера схожести объектов друг на друга на основе редакционного расстояния. Для определения того, являются ли две записи копиями друг друга, вводятся пороговые значения, на основе которых принимается решение. Применив данный подход к базе данных из 300,589 записей, авторы статьи получили следующие результаты. Использование индексов и предварительной кластеризации уменьшило

количество сравнений с $45 \cdot 10^9$ (попарное сравнение всех записей) до $24.8 \cdot 10^6$. Всего было выявлено 38,083 пары записей, сходство которых превышало пороговое значение. Это множество пар описывало 28,948 уникальных пациентов. Авторы статьи рассматривают только одну базу данных, следовательно все записи имели одинаковый формат. Для нахождения дубликатов сравниваются записи на основе расстояния между строками.

Упомянутые выше работы показывают, что задача поиска и устранения дубликатов актуальна в современном мире. Результаты этих исследований являются примерами эффективного применения различных методов обработки данных. Следовательно, научный подход к решению подобных задач имеет право на существование.

1.2. Существующие решения

На рынке программного обеспечения и облачных сервисов уже присутствует ряд решений, которые решают задачу поиска дубликатов в данных.

Библиотека dedupe [9] решает задачу поиска дубликатов в данных с ошибками с использованием методов машинного обучения. Перед запуском алгоритма поиска необходимо предоставить обучающую выборку. Ее можно сформировать с помощью так называемого активного обучения: программа предоставляет пары записей, которые потенциально могут быть дубликаты, а пользователь должен выбрать являются они таковыми или нет. После обучения алгоритм можно применять на основном датасете. Чем больше и репрезентативнее была обучающая выборка, тем выше будет качество поиска. В этом и заключается главный недостаток этого решения, так как далеко не всегда есть возможность

предоставить обучающую выборку. На основе этой библиотеке создан веб-сервис Dedupe.io [10], который позволяет находить дубликаты в данных, объединять несколько датасетов в один, добавлять новые записи в уже обработанные данные.

Аналогом предыдущего решения может выступить сервис Dadata [11], который учитывает российские особенности. Например, сервис предоставляет возможность привести адреса в соответствии с такими общепринятыми стандартами, как КЛАДР и ФИАС. Помимо поиска дубликатов сервис также позволяет проверять серию и номер паспорта через ФМС. Dadata использует для своей работы большое количество справочников: ФИАС, Индексы Почты, Банки и т.д. В контексте решения задачи для МИАЦ этот сервис не может быть использован ввиду конфиденциальности обрабатываемых данных: данные являются персональными и могут храниться и обрабатываться только в информационных системах МИАЦ.

На рынке присутствуют также решения, которые могли бы быть полезными не для поиска дубликатов, а для предварительной обработки данных. Например, сервис «Мастер адресов» [12]. Как можно понять из названия, данный сервис позволяет распознать адреса и привести их к единому виду. Если адресов немного, то их можно обработать бесплатно с использованием веб–интерфейса на сайте сервиса. Однако обработка больших объемов данных уже реализуется на коммерческой основе.

При выборе сервиса для решения задачи поиска дубликатов необходимо учитывать национальную специфику данных. Из этого следует, что в первую очередь стоит обратить внимание на отечественные решения, так как они, как правило, используют различные справочные базы данных, которые существенно повышают качество полученных

результатов.

Вышеперечисленные сервисы были создано для решения задач, которые могут возникнуть в практически любой сфере деятельности. Однако существуют компании, которые занимаются разработкой сервисов для более специализированных областей. Например, компания «Нетрика» [13], которая расположена в Санкт-Петербурге, занимается созданием и внедрением информационных систем в государственном секторе, в частности, в сфере здравоохранения. Среди множества их проектов наиболее интересным в контексте данной работы является решение под названием «N3. Индекс пациентов» [14]. Эта система предназначена для идентификации пациентов лечебных учреждений. Для того чтобы найти нужную карточку пациента, системе необходимо решить задачу поиска дубликатов среди карточек, хранящихся в различных база данных. Во время общения с разработчиком из «Нетрика» удалось выяснить лишь общие сведения об используемых подходах ввиду их секретности. В частности, стало известно, что решение основано на расчете редакционных расстояний между записями.

Проведенный обзор существующих решений позволяет сделать вывод о том, что рассмотренные решения обладают теми или иными недостатками, которые не позволяют их использовать для решения задачи поиска дубликатов для МИАЦ.

Глава 2. Данные

В данной работе рассматриваются две группы данных:

1. данные, предоставленные МИАЦ;
2. тестовая выборка из данных, необходимая для подбора параметров и оценивания точности алгоритма.

2.1. Данные, предоставленные МИАЦ

Источником реальных данных в данной работе является МИАЦ. Данные представлены в виде таблицы, состоящей из 4 столбцов: фамилия, имя, отчество, дата рождения. Краткое описание данных представлено на таблице ниже.

	Фамилия	**Имя**	**Отчество**	**Дата рождения**
Количество непустых значений	34402	34391	32005	34392
Количество уникальных значений	16324	1324	2034	16503
Количество пустых значений	4	15	2401	14

Табл. 1. Количество уникальных и пропущенных значений в базе данных.

2.2. Тестовые данные

Для того, чтобы проверить качество работы алгоритма, необходимо сформировать тестовую выборку. Так как в рассматриваемом датасете имеется всего 4 признака, то могут возникнуть такие ситуации, в которых крайне затруднительно оценить корректность работы алгоритма. Например, если опечатка допущена в полях «Фамилия», «Имя» и/или

«Отчество», то она не помешает эксперту определить истинное значение в этих полях. Однако если в двух записях значения вышеперечисленных признаков совпадают, а значения поля «Дата рождения» отличаются, то нельзя точно сказать, относятся ли эти две записи к одной сущности или имело место ошибка при вводе данных. Чтобы избежать неоднозначности, необходимо определить допустимые различия значений поля «Дата рождения»: если две записи являются дубликатами, то значения поля «Дата рождения», приведенные к единому формату, могут отличаться не более чем на 1 символ. Например, записи «Петров Иван Владимирович 11/12/1950» и «Петров Иван Владимирович 11/12/1960» будут считаться дубликатами, так как различия можно классифицировать как опечатка.

Из вышеописанного датасета было выбрано 1000 записей, среди которых вручную были найдены дубликаты. Все записи были разбиты на кластеры (подмножества записей, которые попарно являются дубликатами).

Всего записей	1000
Уникальных записей	990
Кластеры по 1 записи	981
Кластеры по 2 записи	8
Кластеры по 3 записи	1

Табл. 2. Дубликаты в тестовых данных.

Тестовые данные нужны не только для того, чтобы оценить точность работы алгоритма, но и для подбора оптимальных значений параметров.

Глава 3. Предварительная обработка данных

Перед тем как начать работу с данными, необходимо повысить их качество. Для этого нужно исправить различные ошибки и определить общий формат для каждого поля.

3.1. Виды ошибок

Все ошибки можно разделить на две большие группы: ошибки на уровне поля и ошибки на уровне записи.

Ниже перечислены типы ошибок на уровне поля, которые были обнаружены в данных:

1. Недопустимые символы: цифры, знаки препинания, пробелы и т.д.
2. Написание имён и фамилий транслитерацией.
3. Замена символов одного алфавита символами другого алфавита, имеющие одинаковое написание. Например, буквы «о», «е», «а», «у» в русском и английском алфавитах.
4. Нестандартное использование некоторых символов. Например, «0» вместо «о», «1» вместо «!», «ь1» вместо «ы» и так далее.
5. Разные форматы отчеств. Некоторые отчества формируются по следующей формуле: имя отца + «Оглы» или «Кызы», в зависимости от пола.
6. Разные варианты обозначения пропущенного значения: пустая строка, ключевое слово «нет», специальные знаки типа «-» и т.п.
7. Различные форматы написания даты рождения.

В данных встречались следующие ошибки на уровне записи:

1. Несоответствие поля и значения. Например, в поле «Фамилия» через пробел записаны и фамилия, и имя, и отчество, а поля «Имя» и «Отчество» остаются пустыми.

2. Двойные и девичьи фамилии. Основная проблема заключается в том, что в данных встречаются различные форматы написания множественных фамилий. Например, «Медведева-Соколова», «Медведева (Соколова)», «Медведева (дев. Соколова)», «Медведева (была Соколова)», «Медведева Соколова» и т.д.
3. Двойные имена и отчества. Этот пункт аналогичен предыдущему.

3.2. Способы устранения ошибок

Наиболее эффективным способом устранения ошибок является привлечение эксперта и исправление ошибок вручную. Однако это возможно лишь на маленьких объемах данных, к тому же многие ошибки могут быть исправлены в автоматическом режиме.

Способы исправления ошибок можно условно разделить на три группы: регулярные выражения, эвристические правила, полуавтоматическое исправление.

С помощью регулярных выражений можно устранить такие ошибки, как наличие недопустимых символов или неверный формат даты.

Эвристические правила требуют уже более активного участия эксперта, однако само применение правил может быть автоматическим. Например, вопрос о различных форматах отчества может быть разрешен с помощью эвристического правила, по которому образуются отчества в Российской Федерации. Упрощённый вариант правила можно записать так: имя отца + «ович/евич» или «овна/евна» в зависимости от пола и последней буквы в имени отца. После применения этого правила к отчествам, которые содержат в себе такие ключевые слова, как «Оглы» или «Кызы», можно приблизить все отчества к общему формату. Восстановить достоверно корректное написание такого типа отчеств без

привлечения экспертов в соответствующих областях крайне затруднительно.

Полуавтоматическое исправление ошибок подразумевает активное взаимодействие эксперта и компьютера: компьютер находит записи, в которых потенциально могут быть ошибки, а эксперта принимает решение по их исправлению. Таким образом, например, могут быть обработаны записи, поля которых содержат пробелы. Как правило, если в поле есть пробел, то скорее всего в него записано несколько значений, однако решение о том, к каким полям должны относиться эти значения, должен принимать эксперт. Данный подход к исправлению ошибок является самым долгим, и применять его стоит только после обработки датасета первыми двумя группами методов. Как показывают данные из МИАЦ, после применения регулярных выражений и эвристических правил количество ошибок, которые требуют активного участия эксперта, остается очень мало.

3.3. Результаты предобработки

После исправления ошибок и приведения значений к общему виду была собрана следующая статистика.

	Фамилия	**Имя**	**Отчество**
Недопустимые символы	35	8	21
Двойные значения	38	6	11
Некорректные заполнения пустых значений	0	3	10

Табл. 3. Результаты предобработки данных.

Также было выявлено 62 отчеств, которые содержали слова « Оглы» или «Кызы».

Глава 4. Алгоритм поиска дубликатов

Идея метода заключается в том, чтобы найти для каждой записи множество других записей, максимально похожих на неё. На основании опыта компании «Нетрика» и сервиса «Dedupe» в качестве меры схожести выбрано редакционное расстояние Левенштейна. В данном подходе рассматриваются четыре поля: имя, фамилия, отчество и дата рождения.

4.1. Индексирование

В первой реализации алгоритма каждая запись сравнивалась с каждой. Однако такой вариант перебора подразумевает большое количество сравнений совершенно непохожих друг на друга записей, так как большинство записей не являются дубликатами. Например, запись о *Сухомлине Кирилле Валерьевиче 1950 года рождения* никак не может быть дубликатом записи о *Петрове Сергее Степановиче 1984 года рождения*, так как нет ни одного поля, которое бы имело общее значение.

Чтобы снизить количество рассматриваемых пар необходимо определить критерий, по которому бы отбирались потенциальные дубликаты, то есть те записи, которые имеют между собой нечто общее. Например, общим может быть значение так называемого ключевого поля. В данной работе реализовано два критерия: на основе подстроки и на основе общих символов.

Критерий на основе подстроки: две записи являются потенциальными дубликатами тогда и только тогда, когда значение ключевого поля одной записи является подстрокой значения ключевого поля другой записи.

Критерий на основе общих символов: две записи являются потенциальными дубликатами тогда и только тогда, когда доля общих

символов значений ключевого поля двух записей превышают определённое пороговое значение.

Оба эти критерия обладают своими недостатками. Критерий на основе подстроки бесполезен, если допущены опечатки в начале или в середине значений ключевых полей. Недостаток критерия на основе общих символов заключается в выборе порогового значения.

Критерий нужен для того, чтобы создать следующую структуру данных, называемую индексом. Индекс — это ассоциативный массив, ключами которого являются значения ключевого поля, а значениями — списки записей, которые удовлетворяют критерию.

4.2. Вычисление матрицы расстояний

Для каждой пары записей рассчитывается расстояние Левенштейна, и, таким образом, для каждого поля записи создаётся матрица расстояний размером $n \times n$. Так как операция вычисления расстояния Левенштейна коммутативна, то матрица схожести будет симметричной.

$$x_{ij} = distance(i, j) = distance(j, i),$$

где i, j — это номера записей в датасете, а *distance* — функция расстояния Левенштейна.

Функция вычисления расстояния Левенштейна возвращает минимальное количество операций удаления, вставки символа и замены символа одного на другой, необходимых для преобразования одной строки в другую. По этому значению трудно определить, какие пары записей более похожи друг на друга. Для преобразования абсолютной величины в относительную необходимо провести нормализацию по формуле:

$$x_{ij} = \frac{length_{max} - x_{ij}}{length_{max}}, \text{где } length_{max} = max(length(i), length(j))$$

После нормализации значения матрицы находятся на промежутке [0, 1]: чем ближе к 0, тем менее похожи записи друг на друга.

4.3. Поиск дубликатов

Для поиска дубликатов необходимо определить вектор пороговых значений ***threshold***, на основе которого будет приниматься решение о добавления записи в кластер. Это наиболее сложное место данного подхода, так как значения подбираются эмпирическим путем.

Установить, что две записи ссылаются на одну и ту же сущность можно несколькими способами:

1. Все значения расстояний между соответствующими полями больше значений ***threshold***.
2. Хотя бы одно из значений расстояний между соответствующим полями больше значений ***threshold***.

Целесообразнее выбрать первый способ, так как во втором случае совпадение фамилии, имени или отчества будет достаточно для рассмотрения добавления записей в один кластер.

Поиск дубликатов реализуется на основе матрицы расстояний. В данной работе рассматривается два алгоритма: рекурсивный и линейный.

Первоначальный вариант алгоритма был рекурсивным. Для записи ***i***, которая еще не принадлежит ни одному из кластеров, находится такая запись ***j***, которая потенциально могла бы быть в одном кластере с ***i***. Если таких записей больше одной, то берётся такая запись, что норма вектор расстояний между этой записью и записью ***i*** максимальна. Записи ***i*** и ***j*** помещаются в кластер ***C***. Затем вышеописанный поиск повторяется для записи ***j*** и так далее. Если на очередном этапе для записи ***i*** не находится ни одной похожей записи, то кластер ***C*** считается полностью

сформированным. Далее поиск дубликатов применяется по отношению к следующей записи, которая еще не состоит ни в одном из кластеров. Алгоритм работает до тех пор, пока все записи не будут помещены в кластеры.

Недостаток этого алгоритма заключается в том, что он не учитывает отсутствие свойства транзитивности отношения «похожести» между записями. Если запись ***a*** является дубликатом записи ***b***, а запись ***b*** является дубликатом записи ***c***, то из этого не следует, что ***a*** является дубликатом ***c***. В частности это хорошо видно в случае с пропущенными значениями полей.

Линейный алгоритм отличается от рекурсивного тем, что в кластер помещается не наиболее похожая запись, а все записи, которые удовлетворяют ***threshold***. Следующий кластер формируется из записи, которая еще не состоит ни в одном из кластеров, и всех записей, которые похожи на неё и удовлетворяют ***threshold***.

4.4. Мера качества

Результатом работы алгоритма является разбиение множества записей на непересекающиеся подмножества. Мерой качества результата можно выбрать количество допущенных ошибок. Под ошибкой понимается неправильное помещение записи в кластер. Например, рассмотрим два разбиения: предполагаемое — [{1, 2, 3}, {4, 5}, {6}] и истинное — [{1, 2}, {3, 4}, {5, 6}]. Всего 2 ошибки: «3» и «5» находятся не в своих кластерах.

Количество ошибок также можно интерпретировать как минимальное количество перемещений, которые надо совершить, чтобы одно разбиение преобразовалось в другое.

4.5. Полученные результаты

Для подтверждения работоспособности алгоритма первые эксперименты были проведены на тестовых данных, которые были описаны ранее. Алгоритм применялся со следующими параметрами:

1. Вектор пороговых значений ***threshold***. Компоненты вектора принимают значения от 0 до 1. Количество компонент равно количеству полей в записи, то есть в данном случае 4.
2. Вариант алгоритма поиска дубликатов: линейный или рекурсивный.

Так как значения поля «Дата рождения» имеют строго фиксированную длину, и для этого поля ранее был введен строгий критерий равенства значений, то пороговое значение для этого поля можно определить заранее. Если принять во внимание нормализацию редакционного расстояния Левенштейна, то пороговое значение для поля «Дата рождения» признака должно быть не более чем 0.9.

Для признаков «Фамилия», «Имя», «Дата рождения» пороговые значения лежат в диапазоне от 0.7 до 0.9 с шагом 0.2.

Итоги экспериментов представлены на таблице 4. Точность вычислялась по следующей формуле:

$$accucary = \frac{total - errors}{total},$$

$total$ — общее количество записей,

$errors$ — количество допущенных ошибок.

По результатам, представленным в виде таблицы 4, видно, что линейный алгоритм справляется со своей задачей лучше рекурсивного, а пороговые значения должны быть принимать значения в диапазоне от 0.76 до 0.84.

Пороговые значения				**Рекурсивный поиск**		**Линейный поиск**	
Фамилия	Имя	Отчество	Дата рождения	Количество ошибок	Точность	Количество ошибок	Точность
0.7	0.7	0.7	0.90	9	0.991	4	0.996
0.72	0.72	0.72	0.90	9	0.991	4	0.996
0.74	0.74	0.74	0.90	9	0.991	4	0.996
0.76	0.76	0.76	0.90	7	0.993	2	0.998
0.78	0.78	0.78	0.90	7	0.993	2	0.998
0.80	0.80	0.80	0.90	7	0.993	2	0.998
0.82	0.82	0.82	0.90	7	0.993	2	0.998
0.84	0.84	0.84	0.90	7	0.993	2	0.998
0.86	0.86	0.86	0.90	8	0.992	3	0.997
0.88	0.88	0.88	0.90	8	0.992	3	0.997
0.90	0.90	0.90	0.90	8	0.992	3	0.997

Табл. 4. Результаты применения алгоритма на тестовых данных.

После экспериментов над тестовой выборкой алгоритм был применен на основных данных. Были использованы оптимальные пороговые значения, выявленные на тестовом этапе.

Всего записей	34406
Уникальных записей	33987
Кластеры по 1 записи	33575
Кластеры по 2 записи	405
Кластеры по 3 записи	7

Табл. 5. Результаты применения алгоритма на основных данных.

Ручная проверка результатов показала, что все записи, отмеченные как дубликаты, являются таковыми. Однако нельзя утверждать, что алгоритм выявил все дубликаты, которые существуют в данных.

4.6. Устранение дубликатов

Следующим шагом после нахождения дубликатов является их последующая обработка. Существует несколько подходов к решению этой задачи:

1. Оставить одну запись, а остальные удалить. При таком подходе возникает вопрос: какую запись оставить? Например, в качестве такой записи может быть выбрана та, которая максимально похожа на все остальные.
2. Оставить все записи, а одну отметить как «мастер-запись». Под «мастер-записью» имеется в виду запись, которая будет «эталонной», которая будет «представлять» данный кластер. Однако возникает вопрос о выборе такой записи. Претендентами могут быть, например, самая полная запись, последняя (по времени добавления в базу) запись.

Разработчики решения «N3. Индекс пациентов» следуют принципу неизменности поступивших данных, все записи являются равноценными, поэтому все они остаются в системе. По словам создателей системы у них не возникала необходимость создавать «мастер-запись» на основе дубликатов.

Не существует универсального решения задачи по обработке найденных дубликатов: все зависит от того, какие цели преследуют разработчики. Однако всегда стоит помнить о ценности данных и обращаться с ними максимально аккуратно. Например, МИАЦ в качестве решения этой задачи выбрал подход, при котором все записи остаются в базе данных и каждой из них присваивается номер кластера, которому она принадлежит.

Глава 5. Структура библиотеки

5.1. Принципы построения

Решение задачи поиска дубликатов очень сильно зависит от данных, в которых ведется поиск. К сожалению, нельзя создать единый универсальный алгоритм, который смог бы обрабатывать любые данные. Для каждой задачи требуется индивидуальный подход. Однако структура решения остается общей вне зависимости от данных. Решение с помощью вышеописанного алгоритма подразумевает следующие этапы: чтение данных, предобработка, индексирование, вычисление матрицы, нормализация значений, поиск дубликатов.

Реализация каждого этапа основывается на 3 принципах:

1. Каждый этап должен быть максимально независим от других.
2. Каждый этап должен содержать в себе как можно больше параметров: данное свойство облегчит использование кода для решения других экземпляров задачи.
3. Все промежуточные результаты должны сохраняться.

Для удобного использования алгоритма был создан прототип библиотеки [15], в которой каждый этап алгоритма реализован в виде отдельного модуля. Разработка велась с использованием языка программирования **Python** [16], а также пакетов **pandas** [17] и **numpy** [18].

5.2. Описание модулей

Реализация алгоритма с помощью рассматриваемой библиотеки подразумевает последовательное взаимодействие соответствующих модулей (рис. 1).

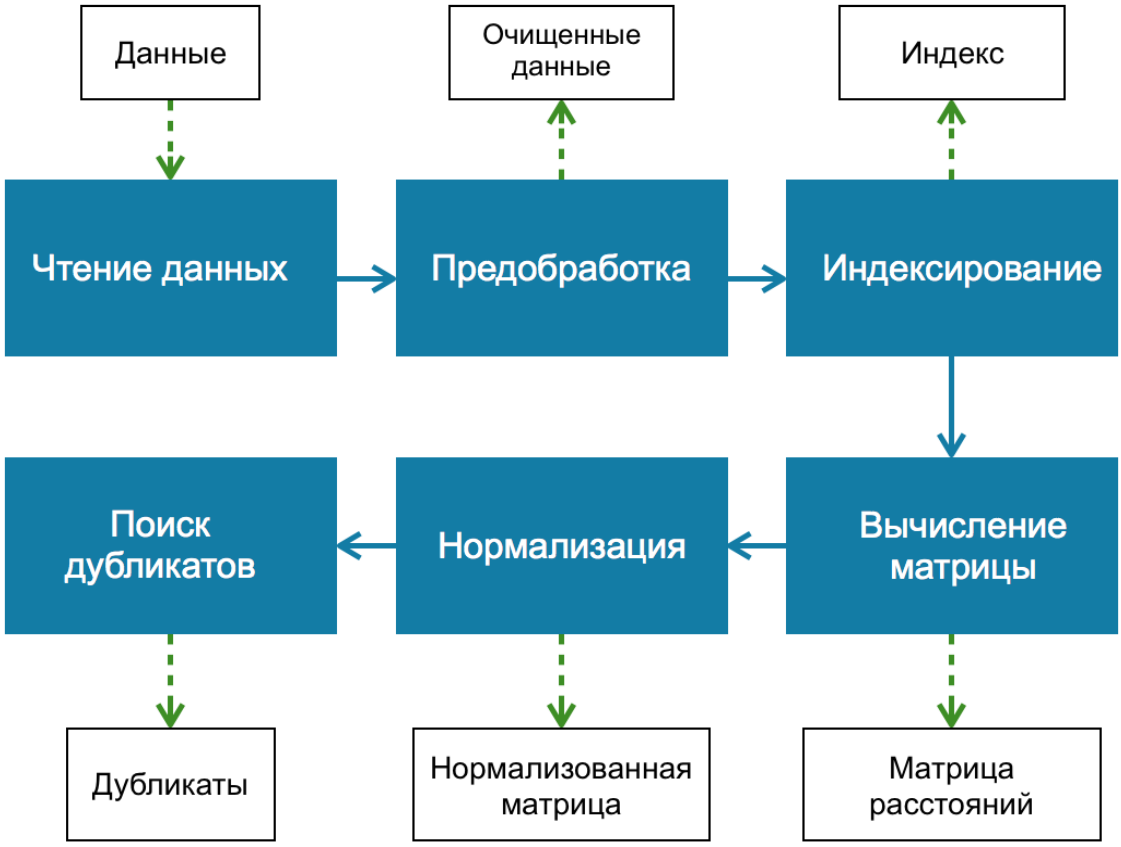

Рис.1. Схема взаимодействия модулей. Сплошные стрелки — передача данных между модулями. Пунктирные стрелки — чтение или запись данных.

Модуль **«Чтение данных»** отвечает за получение данных из источника. Источники данных могут быть различными, и поэтому необходимо было реализовать поддержку различных форматов. На данный момент реализована работа с такими форматами, как файлы csv, excel, реляционные базы данных (MySQL, Oracle). Основными структурами, в которых хранятся прочитанные данные, находясь в памяти компьютера, являются классы **DataFrame** [19] библиотеки **pandas** и класс

Series [20] библиотеки **numpy**.

Модуль **«Предобработка»** содержит в себе инструментарий для устранения ошибок в данных и приведения значений к единому формату. Как было описано в главе 2.2, методы условно делятся на 3 типа: регулярные выражения, эвристические правила и полуавтоматическое исправление. Пользователю предоставлен ряд стандартных функций, которые могут быть применены для «очистки» данных: удалить недопустимые символы, перевести все буквы в нижний регистр, привести дату к общему формату и тому подобные. Так как требования пользователей могут быть крайне разнообразными, то реализована возможность использовать собственные методы.

Модуль **«Индексирование»** ответственен за создание индекса, который описан в главе 3.1. На данный момент реализовано два способа индексирования: на основе подстроки и на основе общих символов. Если пользователя не устраивают предложенные варианты, то он может реализовать свой критерий, на основе которого будет создан индекс.

Модуль **«Вычисление матрицы»** является одним из главных и отвечает за вычисление матрицы расстояний между записями. На данный момент реализована только одна функция, которая рассчитывает редакционное расстояние — это расстояние Левенштейна. Пользователю также предоставлена возможность использовать функцию, созданную им самим. На этапе вычисления расстояний можно сразу нормализовать значения, как было описано в главе 3.2.

Модуль **«Нормализация»** нужен для того, чтобы пользователь мог нормализовать значения матрицы с помощью альтернативной функции нормализации. Для этого нет необходимости снова рассчитывать матрицу расстояний, поэтому эта функциональность была вынесена из модуля

«Вычисление матрицы».

Модуль **«Поиск дубликатов»**, как это понятно из названия, ответственен за нахождение дубликатов, и на данный момент реализовано два способа это сделать: линейный и рекурсивный подходы, подробное описание которых было представлено в главе 3.3.

Модуль **«Логирование»** крайне полезен в использовании при проведении экспериментов или настройке параметров. Информация о текущем состоянии процесса обработки (время начала и окончания обработки, количество обработанных записей на текущий момент, информация о текущем этапе работы) в легко читаемом виде может быть выведена как на экран , так и может быть сохранена в файлы. Функции из этого модуля используются внутри всех остальных модулей.

Модульность прототипа библиотеки, сохранение промежуточных результатов и логирование обеспечивают удобное применение разработанных инструментов и возможность качественного анализа полученных результатов.

Заключение

В рамках решения задачи поиска и устранения дубликатов в персональных данных был проведен анализ данных, предоставленных МИАЦ, а также существующих решений на рынке программного обеспечения. На основе результатов анализа было принято решение реализовать собственный подход, так как недостатки рассмотренных решений, описание которых представлено в главе 1.2, не позволяли эффективно решить задачу для МИАЦ.

В ходе работы был разработан прототип библиотеки, включающий в себя как и инструменты для предобработки данных, так и методы поиска дубликатов. Алгоритм был протестирован на тестовой выборке, показал приемлемые результаты, тем самым доказав, что идеи, на которых он основан, являются перспективными. Дальнейшую разработку прототипа библиотеки планируется направить на устранение недостатков текущей версии алгоритма, среди которых главным является необходимость подбора пороговых значений вручную.

Список источников и литературы

1. И. В. Емелин. Идентификация пациентов // Врач и информационные технологии, 2010. № 2. С.24–29.
2. Bakker, B., Groot, M.D., Grootheest, G.V., & Laan, J.V. Record Linkage in Health Data: a Simulation Study // The Hague: Statistics Netherlands, 2014 P. 60.
3. Медицинский информационно–аналитический центр. http://spbmiac.ru/o-miats/istoriya-miats/
4. Pullen D., Wang P., Talburt J., Wu N. Mitigating Data Quality Impairment on Entity Resolution Errors in Student Enrollment Data // Information and Knowledge Engineering Conference, 2013. P. 96–100.
5. Pullen D., Wang P., Talburt J., Wu N. Applying Phonetic Hash Functions to Improve Record Linking in Student Enrollment Data // Information and Knowledge Engineering Conference, 2015. P. 187–191.
6. Brizan D. G., Tansel A. Uz. Survey of Entity Resolution and Record Linkage Methodologies // Communications of the IIMA, 2006. Vol. 3, No 3. P. 41 – 50.
7. Gemmell J., Rubinstein B. I. P., Chandra A. K. Improving Entity Resolution with Global Constraints // Microsoft Research Technical Report MSR-TR-2011-100, 2011. P. 10.
8. Sauleau E., Paumier J. P., Buemi A. Medical record linkage in health information systems by approximate string matching and clustering // BMC Medical Informatics and Decision Making, 2005. Vol. 5. Art. No 32.
9. Dedupe. Python library for accurate and scalable fuzzy matching, record deduplication and entity resolution. https://github.com/dedupeio/dedupe
10. Сервис «Dedupe.io». https://dedupe.io
11. Сервис «Dadata». https://dadata.ru/clean/

12. Сервис «Мастер адресов». http://www.addressmaster.ru/index.html
13. Нетрика. Компания по разработке ИТ–решений для государственных ведомств. http://netrika.ru
14. N3. Индекс пациентов. ИТ–решение компании «Нетрика». http://netrika.ru/solution/health#solutions
15. Разработанная библиотека. https://github.com/KirovVerst/record_linkage
16. Python. https://www.python.org
17. Pandas. Library providing high–performance, easy–to–use data structures and data analysis tools. http://pandas.pydata.org/index.html
18. Numpy. Package for scientific computing with Python. http://www.numpy.org
19. Pandas DataFrame class. http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html
20. Numpy Series class. https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html