УДК 004.825

Использование продукционной модели для генерации ортодонтического плана лечения

Волков Иван Максимович
Российский Экономический Университет имени Г. В. Плеханова
студент

Токмакова Наталия Романовна
Российский Экономический Университет имени Г. В. Плеханова
студент

Новомирская Милена Анатольевна
Первый Московский государственный медицинский университет имени И. М. Сеченова
студент

Аннотация
В данной статье анализируется возможность использования продукционной модели для генерации предварительного плана лечения в сфере ортодонтии. В результате анализа описывается алгоритм вывода правил на основе определённых входных и соответствующих выходных данных, а также определяются положительные и отрицательные стороны от использования такой модели.
Ключевые слова: продукционная модель, экспертная система, план лечения

Volkov Ivan Maximovich
Russian Plekhanov University of Economics
student

Tokmakova Natalia Romanovna
Russian Plekhanov University of Economics
student

Novomirskaya Milena Anatolyevna
I.M. Sechenov First Moscow State Medical University
student

Abstract
This article analyzes the possibility of using the production model to generate a preliminary treatment plan in the field of orthodontics. As a result of the analysis, an algorithm for deriving rules is described based on certain input and corresponding output data and the positive and negative aspects from using such a model are determined.

Keywords: production system, expert system, treatment plan

Информационные технологии являются неотъемлемой частью экономического, технологического и конкурентного развития организации. К сожалению, медицинская сфера в России не склонна к активному использованию таких технологий в своей повседневной деятельности администрацией и лечащим персоналом. В то же время различные клиники обладают большим объёмом данных об историях лечения своих пациентов, которые можно использовать как основу для продукционной модели представления знаний. Такая модель позволит строить предварительные планы лечения новых пациентов.

Анализ возможности использования продукционной модели

Продукционная модель знаний представляет собой модель, состоящую из правил, которые имеют вид «если …, то …», а также включает в себя механизм, обеспечивающих соблюдение таких правил. [1] Такая модель является частью экспертной системы, которая предназначена для замены специалиста в какой-либо предметной области для разрешения вопросов, требующих специализированных знаний.

В общем случае, такая модель имеет вид: имя_продукции = <сфера_применения, условие_применимости, ядро, постусловие, комментарии>. [2] Совокупность правил, зависящих друг от друга через конъюнкцию или дизъюнкцию, а также постусловия, можно представлять в виде дерева решений (ацикличного графа).

Для реализации такой функциональности продукционной системе необходимо иметь базу данных, хранящую фактические данные, базу знаний, содержащую совокупность правил, и систему управления. [3]

В соответствии с приведенными выше требованиями к продукционной модели, необходимо определить следующее:

1. База данных должна содержать исторические данные о лечении пациентов с индикатором, определяющим успешность лечения. Необходимо проиндексировать такую базу по полю Врач с целью быстрой выборки данных по конкретному врачу. Такая потребность обоснована приведенным ниже пунктом 2b.
2. База знаний должна быть поделена на три слоя, перечисленные в порядке уменьшения приоритетности при генерации плана лечения:
   a. Правила, определенные самим врачом. Совокупность таких правил составляет личный **протокол** врача. Таким образом, врач сможет персонализировать работу системы.
   b. Правила, основанные на данных о лечениях текущего врача.
   c. Правила, основанные на всей совокупности исторических данных, доступных в базе данных клиники.
3. Система управления должна отвечать за реализацию алгоритма вывода правил на основе входных данных.

Необходимо отметить, что такую модель можно использовать не только в ортодонтическом лечении, но и в других областях медицинской сферы, используя различные данные в базах, входные и выходные параметры.

Определение новизны предложенного решения

С целью определения новизны приведенного ниже решения необходимо проанализировать уже существующие системы, функционирующие в различных сферах стоматологии.

В 2015 году была разработана экспертная система для оценки состояния слизистой оболочки полости рта. Созданная экспертная система, способна распознавать неблагоприятный фон для течения раневого процесса в слизистой оболочке и позволяет прогнозировать сроки ее восстановления после нарушения ее целостности в результате повреждения. Данная система может быть также использована на этапах планирования стоматологического лечения, требующего вмешательства на слизистую оболочку на основании вводимых персональных сведений о пациенте. [4] Используемые данные имели порядковые и количественные признаки. Поэтому был использован коэффициент Красскела—Уоллиса, применимый при работе с данными разнородного характера. Все показатели прошли проверку на согласие с нормальным распределением, использовались методы критериальной статистики, был выполнен кластерный анализ и разработана экспертная программа. Таким образом, система была построена на основе обработки входных данных для анализа состояния слизистой оболочки.

Рассмотрим информационную базу для ещё одной экспертной системы, которая была разработана для диагностики заболеваний челюстно-лицевой области. Она включает в себя такие основные компоненты как: база данных и база знаний. База данных представляет собой реляционную базу процесса лечения пациентов. Она содержит данные о клинике, врачах, пациентах, данных о первичном обследовании, посещениях и диагнозах пациентов. База знаний включает в себя модель управления и модель предметной области. Модель управления представляет собой семантическую сеть, в вершинах которой записаны правила принятия решений в виде продукций, а дугами представлены пути конкретизации продукций. [5]

Другая экспертная система была разработана в области терапевтической стоматологии и построена на основе нечёткой логики. [6] Она показала следующий результат: 92% случаев с правильными диагнозами. При этом оставшиеся 8% нельзя считать ошибочными: это нетипичные случаи, которые входят в базу данных, но требуют более детального задания текста запроса за счет включения большего числа специфичных терминов. Эталоном правильности постановки диагноза считался диагноз, поставленный врачом, который предоставил информацию по конкретному случаю. Обучение системы проводилось на основе 11 892 случаев, а точность диагностирования проверялась с помощью контрольной выборки. Основным залогом успеха авторы считаю правильное построение информационной базы - оптимальное

количество вспомогательных полей, обеспечивающих максимальное быстродействие анализа и отсев неподходящих случаев.

Таким образом, новизна заключается в использовании многослойной информационной базе: генерация плана лечения основывается не только на всех существующих исторических данных, но и на данных исключительно текущего врача и на его личных протоколах. Такая необходимость заключается в персонализации работы системы для каждого ортодонта по причине того, что каждый из врачей имеет свои предпочтения по подходам к лечению своих пациентов.

Определение входных и выходных данных для модели

Рассмотрим использование продукционной модели при генерации ортодонтического плана лечения. Для этого необходимо определить входные и выходные данные для модели.

Ортодонтия – это раздел стоматологии, включающий в себя причины возникновения, диагностику, методы профилактики и лечения дефектов развития зубов и челюстно-лицевого скелета. [7] Врачи-ортодонты решают проблемы неправильного прикуса, неровности зубного ряда, скучности зубов. Пациенты обращаются к врачу-ортодонту с различными целями: улучшить эстетический вид зубов или исправить дефект, мешающий нормальной жизнедеятельности. Для правильного лечения также необходимо учитывать общие данные о пациенте, например, его возраст и пол.

При составлении плана лечения врач-ортодонт должен выбрать метод лечения. В современной ортодонтии существует два основных подхода: постановка брекетов или ортодонтических капп (элайнеров). По сравнению с брекетами, ортодонтические каппы прозрачные, съемные, их комфортнее носить, но они стоят дороже. Врач также должен определить продолжительность лечения и предоставить дополнительные комментарии.

Так, к входным данным можно отнести пол и возраст пациента, цель лечения и прикус. В то же время невозможно сгенерировать полный план лечения автоматически, например, нельзя определить, какие зубы нельзя удалять – такие факты относятся к индивидуальным особенностям пациента. В то же время можно сгенерировать часть плана, а именно следующие выходные данные: метод лечения, продолжительность лечения и комментарии. Необходимо отметить, что комментарии определяются только на основе протоколов врача и не должны содержать специфическую информацию о конкретном пациенте. Именно поэтому модель может генерировать предварительный план лечения, который в результате анализируется врачом и может быть отредактирован.

Определение структуры и приоритетности правил

В приведенной продукционной модели правила могут быть простыми и составными. Простое правило имеет вид «если [одно условие], то [один вывод]». Составное правило может использовать несколько условий и выводов, объединенных логическими операциями. Таким образом, общий вид правила следующий: «Если [пол пациента] [возраст пациента] [прикус] [цель

лечения], то [метод лечения] [продолжительность лечения] [комментарии]». Пример правила может быть таким: «Если пациенту 15 лет, и у него открытый прикус, то использовать в лечении брекеты, и продолжительность лечения составит 18 месяцев». Совокупность условий в правиле называются антецендентом, а выводы – консеквентом. [8]

Учитывая то, что в исторических данных могут появляться противоречия, то есть при одних и тех же входных данных выходные данные могут иметь противоположные значения, необходимо применять к правилам приоритеты. [9] Таким образом, сформируются веса правил. Чем выше вес, тем больше вероятность, что правило будет использовано. Приоритеты определяются следующим образом:

1. Если лечение прошло успешно, то приоритет увеличивается на единицу.
2. Если лечение прошло неуспешно, то приоритет уменьшается на единицу.

Правила, полученные на основе историй лечения текущего врача, должны обрабатываться таким же образом, но необходимо предоставить пользователю менять приоритетность тех правил, которые ему более или менее предпочтительны. Если у нескольких правил с совпадающими входными данными и несовпадающими выходными данными одинаковые наивысшие приоритеты, то используется правило, сформированное позже всего.

Правила, содержащиеся в протоколах, не имеют приоритетов по причине того, что они должны быть уникальны. Протоколы позволят врачу сформировать свои предпочтения по лечению различных пациентов со схожими показателями. Именно в протоколах ортодонт может указать общие комментарии.

В результате определения входных и выходных данных, учитывая многослойную структуру базы правил с приоритетами составлен алгоритм, представленный на рисунке 1.

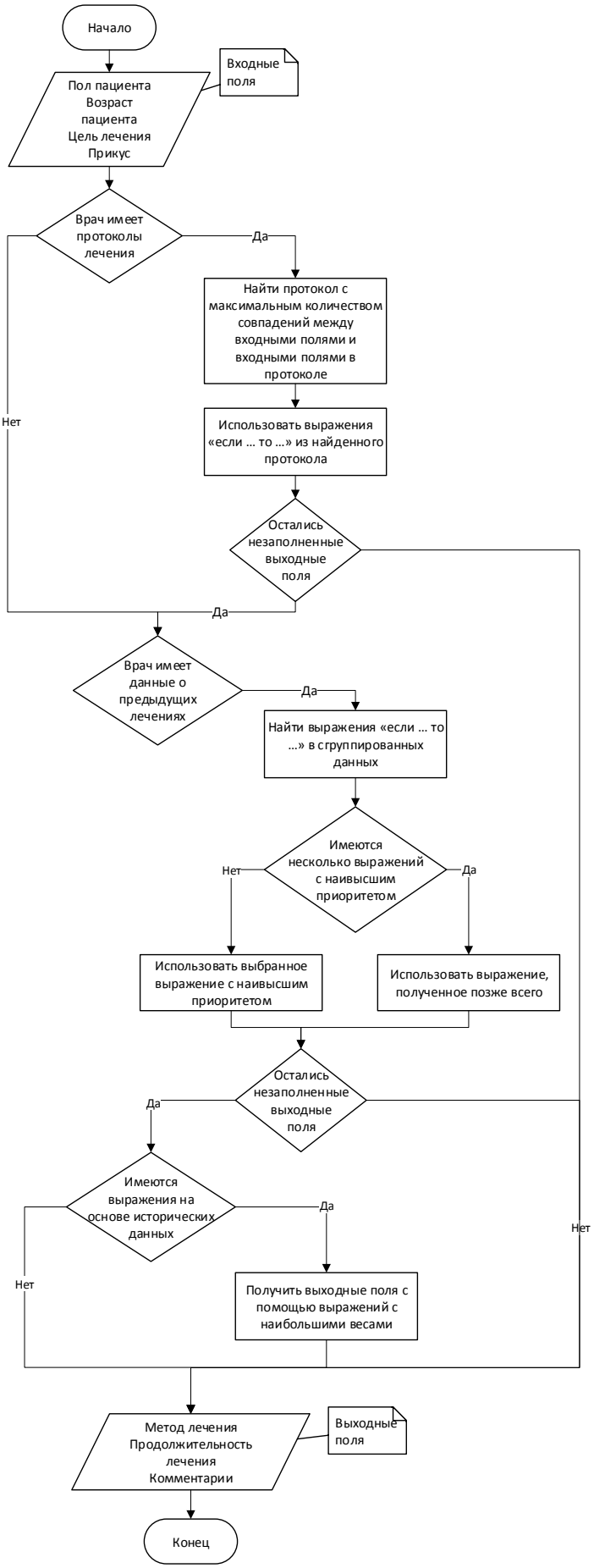

Рис. 1. Алгоритм вывода правил продукционной моделью для генерации предварительного плана лечения.

Реализация алгоритма вывода правил на C++

Для реализации представленного выше алгоритма был выбран язык программирования C++11. Причины использования данного языка следующие: быстрое выполнение программы за счёт прямой работы с областями памяти и наличие большого количества готовых библиотек для решения стандартных задач разработки.

Для обозначения смысловой части правила как в условии, так и в выводе был создан класс Data с открытым конструктором и другими вспомогательными методами. Часть кода класса преставлен ниже. Пример значений скрытых полей, которые может принимать объект данного класса: key = «прикус», value = «открытый», input = true.

```
class Data {
    string key;
    string value;
    bool input;

public:
    Data(string key, string value) :
        key(key), value(value) {
        if (key == "пол пациента"
            || key == "возраст пациента"
            || key == "цель лечения"
            || key == "прикус") {
            input = true;
        }
        else if (key == "метод лечения"
            || key == "продолжительность лечения"
            || key == "комментарий") {
            input = false;
        }
    }

    // ... вспомогательные методы ...
}
```

С использованием класса Data был создан класс Rule, являющийся представлением правила. У правила есть условие и заключение. В случае, если правило получено из базы знаний, то оно имеет приоритет и дату создания. Часто используемый метод данного класса представлен ниже и реализует проверку равенства условий в двух правилах.

```
class Rule {
    vector<Data> condition;
    vector<Data> conclusion;
    int priority;
    Date date_created;

public:
    bool conditions_equal(const Rule& other) {
        if (condition.size() != other.condition.size()) {
            return false;
        }

        int keys_equal = 0;

        // сравнение каждого условия одного правила
        // с каждым условием другого правила
        // и подсчёт количества совпадающих ключей в правилах
        for (Data data : condition) {
            for (Data other_data : other.condition) {
```

```
                if (data.keys_equal(other_data)) {
                    keys_equal++;
                    if (!data.values_equal(other_data)) {
                        return false;
                    }
                }
            }
        }

        // если количество одинаковых совпадающих условий в правиле
        // совпадают с количеством всех условий в правиле
        // то такие правила равны
        return keys_equal == condition.size();
    }

    // ... вспомогательные методы ...
}
```

Класс Protocol содержит в себе несколько правил (объектов класса Rule). Особенность данного класса заключается в нескольких методах.

Первый метод с именем rule_exist проверяет, существует ли правило с заданным условием в протоколе и возвращает индекс в массиве правил. Если такое правило не существует, то метод возвращает -1.

Второй метод с именем add_rule добавляет в протокол новое правило. Учитывая то, что в протоколе не может существовать противоречивых правил, данный метод не добавляет новое правило, если уже существует правило с таким же условием.

Третий метод с именем modify_rule существует для того, чтобы врач мог изменить правило в протоколе. Метод находит правило с таким же условием и удаляет его. После добавляется новое правило.

```
class Protocol {
    vector<Rule> rules;

public:
    int rule_exist(const Rule& other_rule) {
        for (size_t i = 0; i < rules.size(); i++) {
            if (rules[i].conditions_equal(other_rule)) {
                return i;
            }
        }
        return -1;
    }

    bool add_rule(const Rule& other_rule) {
        if (rule_exist(other_rule) == -1) {
            rules.push_back(other_rule);
            return true;
        }
        return false;
    }

    bool modify_rule(const Rule& other_rule) {
        int index = rule_exist(other_rule);
        if (index != -1) {
            rules.erase(rules.begin() + index);
            rules.push_back(other_rule);
            return true;
        }
        return false;
    }
```

```
    // ... вспомогательные методы ...
};
```

Объект класса Doctor имеет ссылку на объект класса Protocol для возможности получения протокола по доктору:

```
class Doctor {
    string id;
    string full_name;
    Protocol protocol;

    // ... вспомогательные методы ...
};
```

Класс Treatment_Plan инкапсулирует в себе векторы входных и выходных данных, а также врача, который запросил генерацию данных. Ниже представлен главный метод, отвечающий за генерацию плана лечения.

```
class Treatment_Plan {
    Doctor doctor;
    vector<Data> input_data;
    vector<Data> output_data;

public:
    Treatment_Plan generate_plan() {
        vector<Data> copy_input_data(input_data.size());
        for (Data data: input_data) {
            copy_input_data.push_back(data);
        }
        Rule rule_without_conclusion(input_data);

        process_doctor_protocol(doctor, copy_input_data,
                                output_data, rule_without_conclusion);
        if (output_data.size() == AMOUNT_CONCLUSIONS) {
            return *this;
        }

        process_doctor_data(doctor, copy_input_data,
                            output_data, rule_without_conclusion);
        if (output_data.size() == AMOUNT_CONCLUSIONS) {
            return *this;
        }

        process_historical_data(copy_input_data, output_data,
                                rule_without_conclusion);
    }

    // ... вспомогательные методы...
};
```

Алгоритм сначала использует протокол врача для генерации, вызывая функцию с именем process_doctor_protocol. Его реализация представлена ниже. С помощью объекта доктор функция получает протокол, и если в нём есть правила, то среди них функция пытается найти правила с подходящими условиями. В случае, если такие правила были найдены, выводы из них добавляются в выходные данные плана лечения. Из копии входных данных плана лечения удаляются использованные условия. Если все выходные поля были заполнены, то функция заранее останавливает свою работу.

```
void process_doctor_protocol(Doctor &doctor,
                             vector<Data> &input_data,
                             vector<Data> &output_data,
                             Rule &rule_without_conclusion) {
    Protocol protocol = doctor.get_protocol();

    if (!protocol.is_empty()) {
        vector<Rule> rules_in_protocol = protocol.get_rules();
        for (Rule rule : rules_in_protocol) {
            if (rule.conditions_equal(rule_without_conclusion)) {
                vector<Data> conclusion = rule.get_conclusion();
                for (Data output : conclusion) {
                    output_data.push_back(output);
                }

                vector<Data> conditions_to_delete = rule.get_condition();
                for (Data condition : conditions_to_delete) {
                    input_data.erase(find(input_data.begin(),
                                          input_data.end(),
                                          condition));
                    rule_without_conclusion = new_rule(input_data);
                }
            }
            if (output_data.size() == AMOUNT_CONCLUSIONS) {
                return;
            }
        }
    }
}
```

После вызова функции генерации вывода из протокола врача, делает проверка, смогли ли правила из протокола заполнить все выходные поля. Если нет, то дальше вызывается функция process_doctor_data, в которой происходит поиск выводов в данных из базе знаний, полученных из прошлых лечений данного врача. Реализация такой функции представлена ниже.

Алгоритм использует оставшиеся неудовлетворенные условия для получения массива правил с такими условиями. Из данных правил отбираются такие, у которых имеется максимальный приоритет. Если таких правил несколько, то берется правило, которое было создано позже всего. Из полученного правила берутся выводы и добавляются в выходные данные в том случае, если в них еще нет вывода с таким же ключём. Если не все поля в выходных данных были заполнены, то удаляются удовлетворенные условия для последующей обработки.

```
void process_doctor_data(Doctor &doctor,
                         vector<Data> &input_data,
                         vector<Data> &output_data,
                         Rule &rule_without_conclusion) {
    vector<Rule> doctor_rules = process_data(get_doctor_data(doctor));
    vector<Rule> rules_conditions_equal;
    for (Rule rule : doctor_rules) {
        if (rule.conditions_equal(rule_without_conclusion)) {
            rules_conditions_equal.push_back(rule);
        }
    }

    vector<Rule> rules_with_max_priority
        = get_rules_with_max_priority(rules_conditions_equal);
    vector<Data> conditions;
    vector<Data> conclusions;
```

```
        if (rules_with_max_priority.size() == 1) {
            conditions = rules_with_max_priority[0].get_condition();
            conclusions = rules_with_max_priority[0].get_conclusion();
        }
        else {
            Rule latest_rule = get_latest_rule(rules_with_max_priority);
            conditions = latest_rule.get_condition();
            conclusions = latest_rule.get_conclusion();
        }

        for (Data conclusion : conclusions) {
            // если в частично сгенерированном плане нет значения для текущего ключа
            if (find(output_data.begin(), output_data.end(), conclusion.get_key()) == output_data.end()) {
                output_data.push_back(conclusion);
            }
        }

        if (output_data.size() != AMOUNT_CONCLUSIONS) {
            for (Data condition : conditions) {
                input_data.erase(find(input_data.begin(), input_data.end(), condition));
                rule_without_conclusion = new_rule(input_data);
            }
        }
    }
```

В данной функции были представлены три функции, которые требуют рассмотрения их реализации. Первая функция – process_data – отвечает за обработку данных, пришедших из базы данных. Ее реализация представлена ниже.

```
    // Исторические данные представлены в следующем виде:
    // Внутренний вектор содержит блок из связных данных
    // Внешний вектор содержит все блоки в исторических данных
    vector<Rule> process_data(vector<vector<Data>> data_from_database) {
        vector<Data> condition;
        vector<Data> conclusion;
        vector<Rule> rules;

        // Построение правил на основе исторических данных
        for (vector<Data> data_block : data_from_database) {
            for (Data data : data_block) {
                if (data.is_input()) {
                    condition.push_back(data);
                }
                else {
                    conclusion.push_back(data);
                }
            }
            Rule rule(condition, conclusion);
            rules.push_back(rule);
        }

        return rules;
    }
```

Вторая функция – get_rules_with_max_priority – находит в списке правил такие, которые имеют наибольшее значение приоритета:

```
    // Получение массива правил с максимальным приоритетом
    // из правил, полученных на вход
    vector<Rule> get_rules_with_max_priority(vector<Rule> rules) {
        vector<Rule> rules_with_max_priority;
```

```
        // найти значение максимального приоритета
        int max_priority = INT32_MIN;
        for (Rule rule : rules) {
            int current_priority = rule.get_priority();
            if (current_priority > max_priority) {
                max_priority = current_priority;
            }
        }

        // отобрать правила с максимальным приоритетом
        for (Rule rule : rules) {
            int current_priority = rule.get_priority();
            if (current_priority == max_priority) {
                rules_with_max_priority.push_back(rule);
            }
        }
        return rules_with_max_priority;
    }
```

Третья функция – get_latest_rule – находит правило, созданное позже всего из списка правил:

```
    // Получение правила, созданного позже всех
    // из правил, полученных на вход
    Rule get_latest_rule(vector<Rule> rules) {
        Rule rule = rules[0];
        for (size_t i = 1; i < rules.size(); i++) {
            if (rule.get_date_created() > rules[i].get_date_created()) {
                rule = rules[i];
            }
        }
        return rule;
    }
```

В случае, если после генерации не все выходные поля были заполнены, то следующая функция – process_historical_data – находит подходящие правила в исторических данных. Если походящих правил несколько, то используются правила с максимальным значением приоритетности.

```
    void process_historical_data(vector<Data> &input_data,
                                 vector<Data> &output_data,
                                 Rule &rule_without_conclusion) {
        vector<Rule> historical_rules = process_data(get_historical_data());
        vector<Rule> rules_conditions_equal;
        for (Rule rule : historical_rules) {
            if (rule.conditions_equal(rule_without_conclusion)) {
                rules_conditions_equal.push_back(rule);
            }
        }

        vector<Rule> rules_with_max_priority = get_rules_with_max_priority(rules_conditions_equal);
        for (Rule rule : rules_with_max_priority) {
            vector<Data> conclusions = rule.get_conclusion();
            for (Data conclusion : conclusions) {
                if (find(output_data.begin(), output_data.end(), conclusion.get_key()) == output_data.end()) {
                    output_data.push_back(conclusion);
                }
            }
        }
    }
```

Экономическая эффективность от внедрения модели в составе экпертной системы

Общая стоимость разработки системы составляет 1000000 рублей, включая в себя стоимость оплаты труда бизнес-аналитиков, разработков и тестировщиков. Распространяться модель будет в рамках продажи годовых лицензий, стоимостью 80000 рублей в год. Так, для достижения срока окупаемости, равному один год, необходимо продать порядка 13 лицензий. Экономия расходов для клиники после покупки лицензии будет составлять около 100000 рублей в год. В таком случае экономический эффект от внедрения будет составлять 20000 рублей в год.

Таким образом, можно сделать вывод, что продукционную модель возможно использовать для генерации предварительного плана лечения. Многослойная структура информационной базы позволит персонализировать работу модели. Это сможет ускорить работу врача, а также позволит осуществлять обмен опытом между врачами через такую единую точку хранения и обработки знаний.

Библиографический список

1. Васильев Д.Н., Чернов В. Г. Интеллектуальные информационные системы. Владимирский Государственный Университет, 2008.
2. Головчинер М.Н. Введение в системы зананий - Томск: 2011.
3. Продукционная модель знаний // Портал искусственного интеллекта URL: http://www.aiportal.ru/articles/knowledge-models/production-model.html (дата обращения: 07.03.2019).
4. Экспертная система для оценки состояния слизистой оболочки полости рта // Издательство "Медиа-Сфера" URL: https://www.mediasphera.ru/issues/rossijskaya-stomatologiya/2015/1/262072-64062015014 (дата обращения: 25.03.2019).
5. Мануков Серго Экспертная система диагностики заболеваний челюстно-лицевой области: дис. д-р. - Грузия, 2008.
6. ИТ в медицинском диагностировании // StatSoft URL: ИТ в медицинском диагностировании (дата обращения: 25.03.2019).
7. Персин Л.С., Алимова М.Я., Колесов М.А. Ортодонтия. Диагностика и лечение зубочелюстно-лицевых аномалий и деформаций. - М.: ГЭОТАР-Медиа, 2016.
8. Джарратано, Д. Экспертные системы: принципы разработки и программирование -4-е изд. - М.: Вильямс, 2007.
9. Production Rules // Computer Science and Engineering URL: http://www.cse.unsw.edu.au/~billw/cs9414/notes/kr/rules/rules.html (дата обращения: 07.03.2019).