МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Высшая школа экономики и управления
Кафедра «Информационные технологии в экономике»

РАБОТА ПРОВЕРЕНА
Рецензент,Ведущий разработчик
ООО «Unit6»
_____/ И.С. Никишкин/
«_____» _____20____ г.

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой, д.т.н., с.н.с.
_____/ Б.М. Суховилов /
«_____»_____20____ г.

Разработка интернет-приложения по определению оптимального продукта страхования автомобиля

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.03.2020.301–177.ВКР

Руководитель, доцент, к.т.н.
_____/ В.А. Конов /
«_____»_____20____ г.

Автор
студент группы ЭУ – 220
_____/ М.В. Казанцев /
«_____» _____20____ г.

.

Нормоконтролер, доцент, к.т.н.
_____/ Е.Н. Горных/
«_____»_____20____ г.

Челябинск 2020

АННОТАЦИЯ

Казанцев М.В. «Разработка интернет-приложения по определению оптимального продукта страхования автомобиля». – Челябинск: ЮУрГУ, ЭУ-220, 95с., 16 табл.,13 ил., библиографический список – 20наим..

Выпускная квалификационная работа выполнена с целью разработки и применения алгоритма классификации на основе машинного обучения в приложениях для оформления продуктов страхования автомобилей. Комплекс программного обеспечения, полученный в результате проведения исследования и разработки, должен выполнять все шаги оформления страхового полиса через удаленные каналы продаж.

Актуальность темы исследования. Современный рынок страховых продуктов может предложить достаточно большой пул программ страхования автомобилей, в том числе предоставить простые и недорогие универсальные решения. Формирование таких предложений часто не дает клиенту понимания, какой продукт выбрать именно ему. Также персонализированный подход является одной из тенденций современного рынка товаров и услуг, позволяющей организации выйти в лидеры.Именно поэтому информатизация таких предложений имеет высший приоритет для компании.

Для реализации программного комплекса необходимо собрать и обработать структурированные и неструктурированные массивы данных большого объема.

В исследовании рассматривается ряд методов, библиотек и алгоритмов, которые могут применятся для решения задач классификации. Важной частью исследования являются теоретические основы и практическая реализация готового продукта.

Итогом проделанной работы является интернет-приложение, формирующее готовое предложение по страховке автомобиля для клиента на основе введённых им данных. Разработан комплекс сервисов сопровождения, которые помогают реализовать процесс оформления полиса,страховые программы и алгоритмы расчета, согласованные с аналитиками организации.

Программный комплекс успешно отлажен, протестирован и внедрен в систему учета организации.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Каско – продукт страхования транспортных средств от угона или ущерба.

Требования к продуктам в сравнении с ОСАГО отличаются в значительной мере (ОСАГО - обязательное автострахование). Например, в Каско активно используются франшизы и их различные виды. Для каждого автомобиля стоимость Каско зависит от большого количества факторов, таких как: марка и модель автомобиля, год выпуска, комплектация, силовая установка, мощность, опты и стаж лиц, допущенных к управлению транспортным средством. Также для страховки могут выбираться различные опции, такие как выезд аварийного комиссара, тотальный ущерб, бой стекол. Все эти параметры влияют на итоговую стоимость полиса.

Также в бизнесе автострахования применяются коробочные продукты - готовые персонализированные решения с определенными опциями и фиксированным расчетом. Как правило такие предложения считаются более экономичными. Этот продукт является слабо ориентированным на клиента, так как клиент не может внести изменения или добрать необходимые опции.

Продукт коробочного Каско в страховой организации является предложением подразделения businesstobusiness. Это означает, что данный продукт реализуется через фрейм на сайте партнера организации удаленно.

Для повышения уровня клиентоориентированности принято решение исследовать и внедрить методы машинного обучения для определения программы или набора программ страхования для клиента. При указании вводных данных по авто клиент не выбирает из нескольких программ, а для него формируется готовое предложение. Полис рассчитывается и клиенту нужно просто дозаполнить котировку и оплатить онлайн, если оплата проходит, то клиенту на почту отправляется оригинал страхового полиса.

1 ПОСТАНОВКА ЗАДАЧИ

Разработать программный комплекс, который позволит выявлять потребности клиента и подготавливать персональное предложение по страхованию на основе коробочных продуктов страхования автомобилей каско.

Провести исследование предложений компаний конкурентов. Выявить ключевые особенности продуктов, сравнить с разрабатываемым решением.

Провести анализ алгоритмов и разработать такой, который позволит по первичным вводным данным определить потребность клиента, а также выполнить расчет и выдать клиенту ссылку для оформления на сайте. Алгоритм должен иметь высокую точность и выдавать предложения по сегментам разных ценовых категорий, без учета стоимости самого автомобиля. Не мало важным аспектом является скорость работы алгоритма, примерное время отклика менее 1 секунды. При исследовании и реализации алгоритма необходимо учитывать, что параметры автомобилей и их категории могут изменятся, процесс обновления не должен занимать более 24 часов.

Для создания алгоритма необходимо проанализировать различные реализации, такие как: алгоритм на языке программирования ifelse, машинное обучение, глубинное обучение с использованием нейросетей. Также необходимо рассмотреть разные инструменты для разработки алгоритма и выбрать такой, который можно легко доработать. Итоговый продукт должен являться собственностью организации.

Для интерфейса взаимодействия с пользователем нужно использовать удаленные каналы продаж, вывести процесс оформления в мессенджер. Клиент указывает первичные данные об авто, такие как: марка, модель, год выпуска автомобиля, мощность в лошадиных силах, показания одометра (пробег). Данные проходят обработку, и на основе этих данных для конкретного автомобиля выдается конкретный класс, который связан с ссылкой по идентификатору.

Интерфейс необходимо сделать простым и понятным. В процессе оформления предоставить клиенту возможность ознакомится с информацией о процессе оформления, с программами страхования. Основные тезисы и пункты программ, для удобства, должны отправляться в виде изображений.

Разработать программы страхования каско, с фиксированной стоимостью расчета, возможностью расчета всех марок автомобилей из системы MDM. Первично сделать поля ввода, реализованные в мессенджере, после этого реализовать заполнение данных страхователя.

Программы должны содержать различные наборы франшиз и опций, в каждой программе утвержден свой алгоритм расчета. Для этого разработать отдельные классы внутри информационной системы businesstobusiness, в которых реализуется логика и отрисовка полей для заполнения на стороне партнера.

Необходимо реализовать процесс онлайн оплаты полиса. Подключить программы страхования к оплате через «Сбербанк онлайн».

Разработать фрейм, который будет являться фронт приложением для партнера. На данном фрейме расположить поля ввода, в которых будут указываться данные для полиса.

Разработать интеграционный сервис для общения между фреймом и сервисом расчета котировок. Сервис должен использовать RESTAPI. Для сервиса разработать обертку, для перевода данных в XMLдля синхронизации с сервисом расчета котировок, который используетфреймворкWCF(WindowsCommunicontroltionFoundation).

Разработать чат-бота для взаимодействия с клиентами через удаленные каналы продаж.

2 АНАЛИЗ СУЩЕСТВУЮЩИХ СИСТЕМ

В данном разделе приведены основные плюсы уже имеющихся систем различных страховых организаций. Это необходимо для выявления сильных сторон разрабатываемого сервиса. Определяющим недостатком предлагаемых на рынке сервисов по автострахованию является отсутствие алгоритма, формирующего готовые персонализированные предложения.

Коробочные программы страхования Росгосстрах. СК Росгосстрах - крупнейшая организация по страхованию автомобилей в РФ. В таблице 1 приведены основные плюсы коробочных программ компании Росгосстрах.

Таблица 1 – Сравнение сервисов СК Росгосстрах

Свойство системы	Наличие свойства в сервисе готовых персонализированных предложений
1. Коробочные продукты распространяются на имущество	+
2. Коробочные продукты распространяются на страхование жизни и здоровья	+
3. Коробочные продукты «Помощь на дороге»	+
4. «Каско Профессионал»: на случай ДТП, повреждения или поломки(Одно предложение для всех клиентов с фиксированным товаром)	+

Тинькофф страхование - дочерние предприятие Тинькофф Банк, занимающееся страхованием.Свойства сервиса представлены в таблице 2.

Таблица 2 – Сравнение сервисов Тинькофф страхование

Свойство системы	Наличие свойства в сервисе готовых персонализированных предложений
1. Большой выбор коробочных решений для любого класса автомобиля.	+
2. Скорость расчета, финальная сумма выдается через 2 минуты с учетом всех скидок	+
3. Онлайн продажа на своем сайте.	+

СПАО «Ингосстрах» — одна из крупнейших российских страховых компаний, стабильно входит в Топ 10 страховщиков РФ. Относится к категории системообразующих российских страховых компаний. В таблице 3 можно увидеть основные плюсы продуктов данной страховой организации.

Таблица 3 – Сравнение продуктов Ингосстрах

Свойство системы	Наличие свойства в сервисе готовых персонализированных предложений
1. Большой выбор коробочных решений для любого класса автомобиля.	+
2. Наличие фильтров по видам страхования и отраслям, для сужения пула доступных программ.	-
3. Программы страхования оформляются по заявке или звонку в организацию.	-
4. Механизм проверки активации полиса.	+

Группа «АльфаСтрахование» — крупнейшая российская частная страховая компания* с универсальным портфелем услуг, включающим как комплексные программы защиты интересов бизнеса, так и широкий спектр страховых продуктов для частных лиц. Основные свойства сервисов данной компании описаны в таблице 4.

Таблица 4 – Сравнение продуктов Группа «АльфаСтрахование»

Свойство системы	Наличие свойства в сервисе готовых персонализированных предложений
1. Большой выбор решений для любого класса автомобиля.	+
2. Наличие различных каналов продаж, в том числе через партнеров.	+
3. Страхование авто только для физических лиц.	+
4. Страхование имущества автомобилей	+
5. Онлайн продажа на своем сайте.	+

Страховое акционерное общество ВСК – Страховой Дом ВСК (САО «ВСК») осуществляет страховую деятельность с 11 февраля 1992 года и в настоящее время занимает ведущие позиции на рынке страховых услуг России. Свойства системы описаны в таблице 5.

Таблица 5 – Сравнение продуктов Страховой Дом «ВСК»

Свойство системы	Наличие свойства в сервисе готовых персонализированных предложений
1. Наличие различных каналов продаж, в том числе через партнеров.	+
2. Наличие коробочных продуктов каско страхования.	+
3. Онлайн продажа на своем сайте.	+
4. Большой выбор программ страхования.	+

Сервис формирования готовых персонализированных предложений. Отличается от других систем наличием свойств, которые еще не внедрены в отрасль страхования на должном уровне. Основные свойства сервиса выделены в таблице 6.

Таблица 6 – Основные свойства «Сервиса формирования готовых персонализированных предложений»

Свойство системы	Наличие свойства в сервисе готовых персонализированных предложений
1. Интеграция с сайтами партнеров	+
2. Отсутствие расчета - система выдает уже рассчитанное предложение	+
3. Большой выбор программ страхования	+
4. **Персонализированное предложение**	**+**
5. Удаленное оформление на сайте партнера	+

Проведя анализ систем, представленных на рынке, можно сделать вывод, что исследование и разработка алгоритма для выявления персональных предложений обоснована.

3 АНАЛИЗ АЛГОРИТМОВ РЕАЛИЗАЦИИ

Для анализа подобран датасет, который представляет собой набор структурированных и не структурированных больших данных. Проведенныйанализ данныхпозволил выявить, что основные числовые показатели не дают очевидного понимания к какому классу отнести тот или иной автомобиль. Набор данных необходимо очистить от выбросов и значений, которые создают зашумленность. Набор взят из открытой базы на сайте kaggle.com. Пример данных из набора представлен в таблице 7.

Таблица 7 – «Пример данных из набора

Maker	Model	Mileage	Manufacture Year	EngineDisplacement	Enginepower	Bodytype	Color	Transmission	Doorcount	Seatcount	Fueltype	Datecreated	Datelastseen	Priceeur
ford	galaxy	151000	2011	2000	103			man	5	7	diesel	14.11.2015	27.01.2016	10584.75
skoda	octavia	143476	2012	2000	81			man	5	5	diesel	14.11.2015	27.01.2016	8882.31
bmw		97676	2010	1995	85			man	5	5	diesel	14.11.2015	27.01.2016	12065.06
skoda	fabia	111970	2004	1200	47			man	5	5	gasoline	14.11.2015	27.01.2016	2960.77
skoda	fabia	128886	2004	1200	47			man	5	5	gasoline	14.11.2015	27.01.2016	2738.71
skoda	fabia	140932	2003	1200	40			man	5	5	gasoline	14.11.2015	27.01.2016	1628.42
skoda	fabia	167220	2001	1400	74			man	5	5	gasoline	14.11.2015	27.01.2016	2072.54
bmw		148500	2009	2000	130			auto	5	5	diesel	14.11.2015	27.01.2016	10547.74
skoda	octavia	105389	2003	1900	81			man	5	5	diesel	14.11.2015	27.01.2016	4293.12
		3013	200	190	88			ma	5	5	diesel	14.11.2	27.01.2	1332.3

		81	2	0				n				015	016	5
		202136	2002	1400	55			man	5	5	gasoline	14.11.2015	27.01.2016	740.19
		263840	1998	1900	81			man	5	5	diesel	14.11.2015	27.01.2016	999.26
		105394	2000	1360	55			man	3	5	gasoline	14.11.2015	27.01.2016	1665.43
skoda	favorit	41250	1990	1300	44			man	5	5	gasoline	14.11.2015	27.01.2016	370.1
suzuki	swift	122100	2003	1000	39			man	5	5	gasoline	14.11.2015	27.01.2016	999.26
nissan	x-trail	149465	2005	2500	121			auto	5	5	gasoline	14.11.2015	27.01.2016	4811.25
		115879	2003	1900	88			man	5	5	diesel	14.11.2015	27.01.2016	2220.58
opel	astra	316054	2005	1700	74			man	5	5	diesel	14.11.2015	27.01.2016	2331.61
skoda	superb	269398	2005	1900	96			man	4	5	diesel	14.11.2015	27.01.2016	4663.21
skoda	fabia	87257	2008	1200	44			man	5	5	gasoline	14.11.2015	27.01.2016	4219.1
skoda	fabia	130340	2001	1400	50			man	5	5	gasoline	14.11.2015	27.01.2016	2442.64
ford	focus	227415	2002	1800	85			man	5	5	diesel	14.11.2015	27.01.2016	2146.56

Для выбора алгоритма реализации важным параметром является условие, что при вводе всех данных пользователю присваивается один из трех возможных сегментов автомобиля по цене, а данные вводятся без самой стоимости автомобиля.

Самым простым и точным методом реализации алгоритма является набор условных блоков **(если, то)**. Данный алгоритм не проводит анализ, а лишь сравнивает поля и выдает ответ, который присваивается в переменную по условию. Данный способ, на первый взгляд, является самым простым и не требует больших затрат, как от системы, так и от разработчика.

Рассмотрим пример реализации более подробно. Начнем с простой задачи: в зависимости от года выпуска автомобиля определять его категорию, как новый

автомобиль, среднего возраста или старый. Допустим, что автомобиль можно считать новым, если ему до трех лет, автомобиль имеет средний возраст, если ему от 4 до 7 лет и автомобиль считается старым если его возраст более 8 лет.Для расчета возраста автомобиля написан код, который приведен в листинге 1.

Листинг 1 – «Реализация алгоритма определения возраста автомобиля»

```
Console.WriteLine($"Введите года выпуска автомобиля");
var creationDate = Console.ReadLine();
            var age = DateTime.Now.Year - Convert.ToInt32(creationDate);
            string ageControltegory;
            if (age >=0 & age <= 3)
            {
ageControltegory = "New";
Console.WriteLine(ageControltegory);
            }
            else if (age > 3 & age <= 7)
            {
ageControltegory = "Medium age";
Console.WriteLine(ageControltegory);
            }
            else if (age > 7)
            {
ageControltegory = "Old";
Console.WriteLine(ageControltegory);
}
else
            {
Console.WriteLine($"Введено неверное значение года выпуска автомобиля");
            }
```

Данный алгоритм вычисляет возраст авто и присваивает ему категорию. Далее выводит данные в консоль. При детальном рассмотрении можно выявить существенные недостатки данного алгоритма: доработка, связанная с

расширением категорий, приведет к увеличению базы кода и требует непосредственного вмешательства разработчика. Также стоит обратить внимание, что это вычисление категории по одному параметру, если расширять данный алгоритм, добавляя, например, поле пробег и его градации, количество блоков условий будет увеличено в геометрической прогрессии.

Однозначно нельзя сказать, каковы границы категорий. Это накладывает еще больше ограничений и требует больше трудозатрат.

Одной из главных проблем такой реализации является отсутствие корректного сравнения категориальных признаков. Алгоритм не может корректно сравнить и отличить одну марку от другой так же, как и модели автомобилей, для этих нужд необходимо реализовать большое количество условных блоков, так как в базе MDM находится более ста марок, модельный ряд каждой из которых насчитывает десятки наименований. Также для некоторых моделей авто, таких брендов, как Mercedes-Benz и BMW важно наличие индекса в модельном ряду, для определения объема двигателя.

Алгоритм разрастается до огромных размеров, становится запутанным, нечитаемым и сложным в доработке. Исследование показывает, что это противоречит основным тезисам задачи.

Конечный результат работы может не удовлетворять критериям подбора, а так как разобраться в алгоритме тяжело из-за его витиеватости и объема, то найти и устранить ошибку становится очень сложно.

Подводя итог исследования по реализации алгоритма методом условных блоков, можно сделать вывод, что данный метод не является оптимальным, требует большое количество затрат и времени на доработку.

Для проведения анализа при помощи алгоритмов машинного обучения необходимо выбрать стандарт анализа данных, так как данные имеют большое

количество шумов, пустые значения, значения в числовых столбцах NaN. Пример описания данных представлен в таблице 8.

Таблица 8 – «Описание данных до обработки»

	Пробег	Год выпуска	Мощность
Количество	3190328	3182334	2998035
Среднеезначение	115814	2000	98
Минимум	0	0	1
25% выборки	1880	2004	68
50% выборки	86415	2009	86
75% выборки	158025	2013	110
Максимум	9999999	2020	2237

Из таблицы видно, что пробег имеет значения от 0 до 9999999, год выпуска может принимать значение 0. Пробег указан в милях, а мощность двигателя достигает значения выше двух тысяч лошадиных сил.

Анализ данных проведен по стандарту KDD. KnowledgeDiscoveryinDatabases (KDD) – это процесс поиска полезных знаний в "сырых" данных. KDD включает в себя вопросы: подготовки данных, выбора информативных признаков, очистки данных, применения методов DataMining (DM), постобработки данных и интерпретации полученных результатов.

Процесс KnowledgeDiscoveryinDatabases, состоит из следующих шагов:

Подготовка исходного набора данных. Этот этап заключается в создании набора данных, в том числе из различных источников, выбора обучающей

выборки и т.д. Для этого должны существовать развитые инструменты доступа к различным источникам данных. Желательно иметь поддержку работы с хранилищами данных и наличие семантического слоя, позволяющего использовать для подготовки исходных данных не технические термины, а бизнес понятия.

Предобработка данных. Для того чтобы эффективно применять методы DataMining, следует обратить внимание на вопросы предобработки данных. Данные могут содержать пропуски, шумы, аномальные значения и т.д. Кроме того, данные могут быть избыточны, недостаточны и т.д. В некоторых задачах требуется дополнить данные некоторой априорной информацией. Наивно предполагать, что если подать данные на вход системы в существующем виде, то на выходе получим полезные знания. Данные должны быть качественны и корректны с точки зрения используемого метода DM. Поэтому первый этап KDD заключается в предобработке данных. Более того, иногда размерность исходного пространства может быть очень большой, и тогда желательно применять специальные алгоритмы понижения размерности. Это как отбор значимых признаков, так и отображение данных в пространство меньшей размерности.

Трансформация, нормализация данных. Этот шаг необходим для приведения информации к пригодному для последующего анализа виду. Для чего нужно проделать такие операции, как приведение типов, квантование, приведение к "скользящему окну" и прочее. Кроме того, некоторые методы анализа, которые требуют, чтобы исходные данные использовались вопределенном виде. Нейронные сети, скажем, работают только с числовыми данными, причем они должны быть нормализованы.

DataMining. На этом шаге применяются различные алгоритмы для нахождения знаний. Это нейронные сети, деревья решений, алгоритмы кластеризации, установления ассоциаций и т.д.

Постобработка данных. Интерпретация результатов и применение полученных знаний в бизнес-приложениях.

Данные необходимо обработать. На этапе загрузки из датасета сформируем набор с теми полями, которые являются приоритетными. Для этого выявим основные фичи и их важность. Загрузим все данные, очистим от пропусков, числовые данные оставим нетронутыми, а категориальные перекодируем, чтобы сравнивать их в дальнейшем. Загрузим данные в класс FeatureSelector. Методы этого класса позволяют выполнить отбор коллинеарных признаков (очень зависимых согласно корреляции). Данных признаков не выявлено.

Объем данных большой и не подходит для нашей задачи, а так как некоторые фичи, такие, как количество дверей, объем двигателя и прочие необходимо убрать, чтоб не усложнять продукты, выделяем главные признаки. Проверка гипотезы должна выдать следующие результаты – в важные фичи попали марка, модель, пробег, мощность и год выпуска. Пример реализации отбора и проверки фич представлен в листинге 2.

Листинг 2 – «Выбор и проверка фич(признаков)»

```
train_labels = pd.DataFrame(df['price_eur'])
train = df[['maker',
            'model',
            'mileage',
            'manufacture_year',
            'engine_displacement',
            'engine_power',
            'body_type',
            'stk_year',
            'transmission',
            'seat_count',
```

```
                'door_count',
                'fuel_type']]
        lb = LabelEncoder()
        #train[['maker','model','body_type','transmission','fuel_type']]    =
lb.fit_transform(train[['maker','model','body_type','transmission','fuel_ty
pe']])
        train[['maker']] = lb.fit_transform(train['maker'])
        train[['model']] = lb.fit_transform(train['model'])
        train[['body_type']] = lb.fit_transform(train['body_type'])
        train[['transmission']] = lb.fit_transform(train['transmission'])
        train[['fuel_type']] = lb.fit_transform(train['fuel_type'])
        #Экземпляркласса
        fs = FeatureSelector(data = train, labels = train_labels)
        #отбор коллинеарных признаков (очень зависимых согласно корреляции)
        fs.identify_collinear(correlation_threshold = 0.9)
        # списокпризнаковдляудаления
        #collinear_features = fs.ops['collinear']
        # датафреймколлинеарныхпризнаков
        #fs.record_collinear.head()
        #Смотрим важность признаков для задачи регрессии, метод градиентного
бустинга
        fs.identify_zero_importance(task = 'regression',
        eval_metric = 'auc',
        n_iterations = 10,
        early_stopping = False)
        #Запускаем
        zero_importance_features = fs.ops['zero_importance']
        #Рисуем
        fs.plot_feature_importances(threshold = 0.99, plot_n = 12)
```

На рисунке 1 представлен результат работы алгоритма, выявлены ключевые признаки: пробег, мощность, объем, год выпуска, марка, модель.

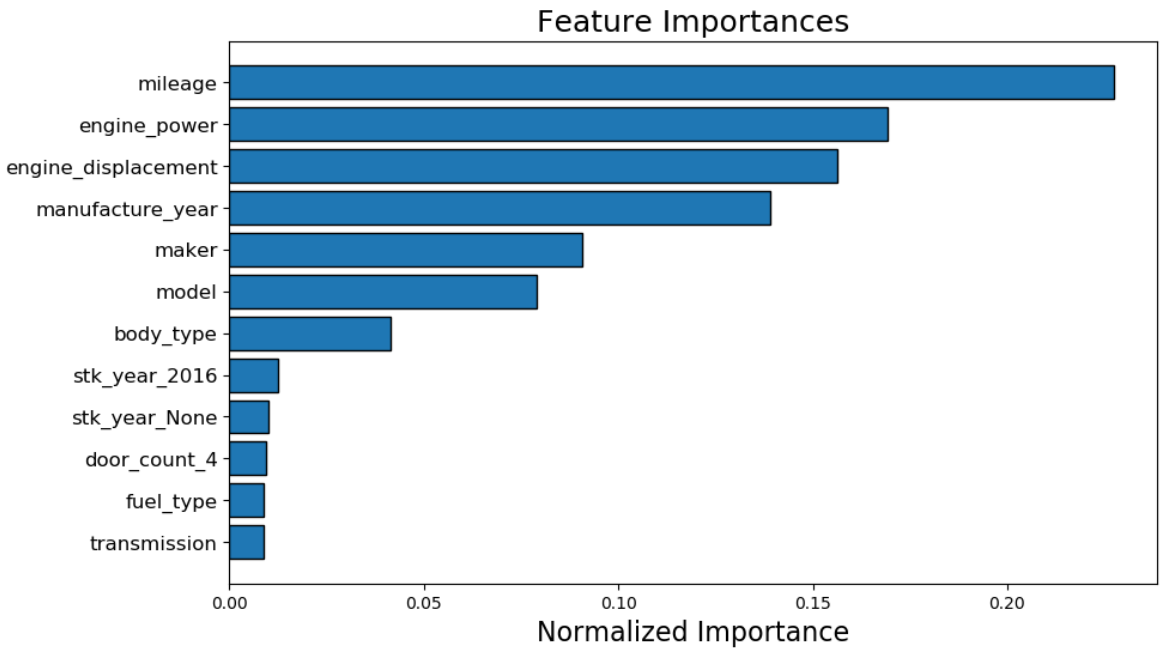

Рисунок 1 – «Важность фич(признаков)»

Принято решение опустить показатель объем двигателя для упрощения ввода параметров.

Послезагрузкиипросмотраописанияданныхметодомбиблиотекинumpy– controlrsDF.describe() необходимоизбавитьсяотзначений, которыесодержатстрокивместочисел, атакжепустыезначения. Для этого воспользуемся методом controlrsDF.dropna(inplace=True). Избавившись от всех NaNи пустых строк, очистим данные от неподходящих для нас значений. При подборе этих значений проведена работа с аналитиками и выявлено, что 97% клиентов, приобретающих коробочные продукты, ездят на бывших в употреблении автомобилях. Даже новый автомобиль, купленный в салоне, показывает несколько километров, так как на заводе его проверяют и перевозят между цехами, грузят на автовоз и т.д. Следовательно пробег автомобилей должен быть больше 0 километров. Дляразрабатываемых программ ограничения по страхованию автомобилей составляют 15 лет, отфильтруем эти данные и оставим в наборе автомобили, год выпуска которых выше 2005. Мощность автомобиля вычислим как интервал между младшей моделью 2005 года и старшей моделью 2020 года, не включая экзотические автомобили. Получим

значения от 70 до 550 лошадиных сил.Последнее ограничение накладывается на пробег, страхованию не подлежат автомобили с показателем одометра более 500000 миль. Так как возраст этих автомобилей, как правило, больше 15 лет. Данные ограничения накладываются в коде, приведенном в листинге 3.

Листинг 3 – «Загрузка и очистка данных»

```
# Загрузили DF
controlrsDF = pd.read_csv("C:/Users/kazan/Desktop/ВКР/Алгоритм/controlrsdataset.csv")
controlrsDF = controlrsDF[['maker',
          'model',
'mileage',
          'manufacture_year',
          'engine_power']]


# чисткаданных
controlrsDF.dropna(inplace=True)
# почиститьпробег
controlrsDF = controlrsDF.loc[controlrsDF['mileage'] > 0]
controlrsDF = controlrsDF.loc[controlrsDF['manufacture_year'] > 2005]
controlrsDF = controlrsDF.loc[(controlrsDF['engine_power'] > 70.) & (controlrsDF['engine_power'] < 550.)]
controlrsDF = controlrsDF.loc[controlrsDF['mileage'] < 500000]
print(controlrsDF.describe())
```

Данныепослеподготовкиипредобработкиимеютописание, представленноевтаблице9.

Таблица 9 – «Описание данных после обработки»

	Пробег	Год выпуска	Мощность
Количество	943675	943675	943675
Среднеезначение	82386	2011	111
Минимум	1	2006	71
25% выборки	19486	2009	81
50% выборки	68986	2012	103
75% выборки	131000	2015	121
Максимум	499000	2020	537

Как видно из описания, данные не имеют выбросов, пропусков и критических значений.

Последним этапом обработки данных является приведение показателей пробега к метрической системе. А также для упрощения работы, фильтрация по марке, в качестве выбранной оставим BMW, так как программы страхования разрабатывались для партнера, который занимается страхования автомобилей этой марки. Код, который трансформирует пробег и оставляет марку BMWпредставлен в листинге 4. В таблице 10 указано описание итоговых данных после всех обработок.

Листинг 4 – «Приведение пробега и фильтрация марки»

```
# Исправили данные (выкинули лишнее)
controlrsDF['mileage']    =    controlrsDF['mileage'].apply(lambdax: x*1.60934)
controlrsDF = controlrsDF.loc[controlrsDF['maker'] == "bmw"]
```

Таблица 10 – «Описание данных после итоговой обработки»

	Пробег	Год выпуска	Мощность
Количество	40846	943675	943675
Среднеезначение	145768	2011	168
Минимум	2	2006	75
25% выборки	48280	2009	130
50% выборки	127137	2012	150
75% выборки	218798	2014	190
Максимум	724162	2020	530

Следующим этапом исследования стал анализ алгоритма машинного обучения с использованием нейросетей. Для решения задачи классификации при помощи нейронных сетей необходимо выбрать такую, которая позволит определять класс автомобиля на основе параметров. Разнообразие сетей и последние исследования показали, что лучше всего такой инструмент справляется с задачами компьютерного зрения. Для вычисления сети используют ресурсы видеокарты, что является одним из факторов удорожания разработки. Для задачи в данном случае необходимо развернуть достаточно мощный комплекс, который позволит обучать нейронную сеть быстро и эффективно. Также существует проблема с переобучением таких сетей. Разработаны целые методологии по качественному обучению. Еще одной из ключевых проблем такого подхода является длительность разработки, необходимо подбирать гипер-параметры сети и обучать ее снова и снова.

Проведя исследование данного метода, можно сделать вывод, что этот метод является более затратным в сравнении с условным алгоритмом.

Нейронные сети хорошо подходят для решения задач кластеризации. Кластеризация – группировка объектов на основе данных, описывающих свойства объектов. Задача кластеризации заключается поиске независимых групп (кластеров) и их характеристик во всем множестве анализируемых данных [1].

Для облегчения работы аналитиков принято решение обработать данные при помощи алгоритмов кластеризации и выявить три кластера, которые бы характеризовали данные по категориям: дешевые авто, авто средней цены и авто премиального класса.

На следующем шаге методологии KDD данные необходимо трансформировать и нормализовать. Для этого изначально нужно разбить данные на вещественные и категориальные. Это делается для отделения сравнений, например, марки и пробега автомобиля, а также для корректного распознавания разных марок и моделей. Пример разделения признаков указан в листинге 5.

Листинг 5 – «Разделение признаков на категориальные и числовые»

```
# Делим признаки на категориальные и числовые
trainDecimal = controlrsDF[['mileage','engine_power','manufacture_year']]
trainControltegoricontroll = controlrsDF[['maker','model']].astype(str)


trainControltegoricontrollTransformed = MultiColumnLabelEncoder(columns = ['maker','model']).fit_transform(trainControltegoricontroll)
```

К числовым признакам относятся – пробег, мощность и год выпуска автомобиля. К категориальным – марка и модель. Далее категориальные данные необходимо трансформировать, то есть перекодировать их в вид, который будет

понятен нейронной сети, для этого используется класс MultiColumnLabelEncoder, который описан в листинге 6.

Листинг6 – «КлассMultiColumnLabelEncoder»

```
    def __init__(self,columns = None):
self.columns = columns # array of column names to encode


    def fit(self,X,y=None):
        return self # not relevant here


    def transform(self,X):
        output = X.copy()
        if self.columns is not None:
            for col in self.columns:
                output[col] = LabelEncoder().fit_transform(output[col])
        else:
            for colname,col in output.iteritems():
                output[colname] = LabelEncoder().fit_transform(col)
        return output


    def fit_transform(self,X,y=None):
        return self.fit(X,y).transform(X)
```

На выходе получается набор массив, который содержит закодированные марки и модели. Марка в данном случае одна, поэтому она имеет код – 0, моделей же может быть от 0 до бесконечности.

После трансформации категориальных данных необходимо нормализовать числовые. Таким образом данные по каждому автомобилю трансформируются в тензоры, с которыми нейронным сетям удобно работать. Тéнзор (от лат. tensus, «напряжённый») — объект линейной алгебры, линейно преобразующий элементы

одного линейного пространства в элементы другого. Создание новых колонок является ключевым отличием кодирования OneHotEncoderот LabelEncoder. То есть вместо одного конкретного параметра получаем три, которые обозначают один показатель, определяющий марку и модель. Далее данные необходимо объединить в одну таблицу. Пример кода нормализации представлен в листинге 7.

Листинг7 – «Нормализацияданных»

```
# Нормализация
scontroller = StandardScontroller()
scontroller.fit(trainDecimal)
trainDecimalTransformed = scontroller.transform(trainDecimal)


ohe = OneHotEncoder(spanelrse = False)
ohe.fit(trainControltegoricontrollTransformed)
trainTransformed                                                    =
ohe.transform(trainControltegoricontrollTransformed)


# Объединениеданных
total = np.hstack((trainTransformed,trainDecimalTransformed))
```

Для реализации и обучения выбран эффективный метод K-средних. Метод k-средних (англ. k-means) — наиболее популярный метод кластеризации. Изобретён в 1950-х годах математиком Гуго Штейнгаузом и почти одновременно Стюартом Ллойдом. Действие алгоритма таково, что он стремится минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров. Данный метод хорошо показал себя в работе, корректно разбив автомобили на 3 категории.

Пример кода алгоритма расположен в листинге 8.

Листинг 8– «Метод KMeans»

```
# Обучаемипредсказываем
```

```
y_pred = KMeans(n_clusters=3, random_state=0).fit_predict(total)
```

В таблице 11 приведен результат работы алгоритма, в столбце ControlrRangотражен класс автомобиля, где 0 класс – средне ценовой сегмент, 1 класс – сегмент дешевых автомобилей, 2 класс – сегмент дорогих премиальных авто.

Проведя анализ этих данных, можно сделать вывод о том, что алгоритм работает корректно.

Таблица 11 – «Результат работы алгоритма кластеризации»

model	ControlrRang	mileage	engine_power	manufacture_year
80	1	362 423	180	2011
407	1	362 102	80	2005
114i	0	115 451	98	2012
	1	294 631	105	2007
	2	18 428	248	2015
116d	0	127 385	86	2013
	1	337 425	89	2010
116i	0	170 141	95	2011
	1	284 531	93	2006
118d	0	82 961	104	2011
	1	305 963	97	2007
118i	0	56 455	99	2009
	1	199 333	102	2007
120d	0	57 676	127	2009
	1	314 573	125	2006
120i	0	109 435	110	2006
	1	210 723	113	2006
125d	0	241 401	160	2012
130i	0	16 093	197	2005
	1	326 306	195	2006
135i	2	16 452	239	2016
216d	0	59 546	85	2015
216i	0	22 990	115	2015
	2	12 129	242	2015
218i	0	19 699	100	2016
220d	0	38 268	139	2015
220i	0	39 912	138	2016
225d	0	8 534	165	2016
225i	0	2	170	2016
316d	0	78 066	85	2013

	1	335 082	85	2012
316i	0	6	85	2003
	1	298 041	81	2001
318d	0	140 595	104	2010
	1	350 409	96	2007
318i	0	101 410	101	2008
	1	287 750	93	2002
320d	0	76 780	127	2010
	1	379 312	114	2004
320i	0	93 442	121	2008
	1	305 600	120	2003
323i	0	334 743	125	2016
	1	277 108	125	1998
325d	0	154 497	156	2012
	1	385 475	146	2008
325i	0	14 724	155	2006
	1	310 219	150	2004
328i	0	174 766	180	2012
	1	350 472	143	1999
330d	0	33 900	172	2008
	1	360 193	149	2003
	2	19 312	205	2006
330i	0	664	190	2005
	1	307 939	177	2003
330xd	0	105 196	136	2013
	1	332 067	124	2004
	2	62 450	236	2014
335d	1	401 798	210	2008
	2	30 396	223	2013
335i	1	292 854	225	2005
	2	126 388	225	2011
420d	0	56 362	138	2015
420i	0	37 830	135	2015
428i	0	102 998	180	2014
430d	0	15 509	190	2016
430xd	0	33 278	148	2015
	1	298 694	173	2008
	2	14 987	242	2016
435d	2	62 637	230	2015
435i	2	47 267	225	2016
520d	0	81 037	135	2014
	1	356 132	128	2009
520i	0	96 731	123	2011
	1	364 505	118	2000
520xd	0	57 070	135	2013
523i	0	145 645	147	2010

Продолжение таблицы 1

	1	357 690	128	2001
525d	0	138 686	150	2010
	1	374 372	132	2005
525i	0	0	160	2005
	1	309 962	152	2005
525xd	0	220 722	158	2012
	1	290 177	145	2008
525xi	0	102 537	171	2014
	1	365 507	156	2006
	2	132 986	251	2013
528i	0	465	172	2012
	1	454 323	142	1997
530d	0	68 155	182	2012
	1	386 886	159	2005
	2	87 888	230	2015
530i	0	153 861	170	2013
	1	328 283	179	2004
530xd	0	100 938	189	2010
	1	373 117	171	2007
535d	0	0	200	2005
	1	380 673	208	2007
	2	143 141	222	2011
535i	1	319 211	180	1999
	2	227 496	233	2011
540i	1	427 630	213	2000
545i	1	296 086	245	2004
550i	1	297 728	270	2006
	2	199 119	275	2008
630ci	1	322 074	220	2006
	2	86 316	258	2013
630i	0	131 966	200	2008
	1	408 772	190	2006
635d	1	338 410	210	2008
	2	146 450	210	2008
640d	2	164 169	230	2013
645ci	1	401 277	245	2004
645i	1	304 948	245	2005
	2	0	245	2006
650ci	2	238 182	270	2006
650i	1	385 437	270	2006
	2	197 387	283	2009
728i	1	441 281	142	1997
728il	0	55 144	192	2015
	1	374 923	190	2006
	2	103 474	267	2014
730d	0	57 946	189	2014

Продолжение таблицы 11

	1	418 016	160	2004
735i	1	547 431	200	2002
740d	0	563	190	2003
	1	441 553	196	2004
	2	88 665	231	2014
740i	1	391 874	215	2003
	2	0	240	2009
745d	1	403 937	242	2007
745i	1	380 503	246	2002
	2	161	245	2002
750i	1	405 554	270	2006
	2	93 490	308	2011
750il	2	260 713	270	2006
760i	2	3 219	448	2017
alpina	0	131 879	146	2010
	1	280 559	175	2004
	2	116 443	305	2007
cooper	1	246 229	85	2005
cooper-s	0	193	120	2004
	1	321 868	120	2004
espanelce	1	288 072	130	2006
i3	0	21 703	121	2014
	2	82 742	385	2013
i8	0	28 655	169	2015
	2	10 429	264	2015
m3	0	124 013	137	2010
	1	255 233	208	2001
	2	101 512	309	2010
m4	0	92 824	141	2009
	1	267 683	102	2005
	2	20 341	316	2015
m5	0	48 809	141	2008
	1	340 992	212	2002
	2	100 165	315	2012
m550d	2	167 418	280	2013
m6	0	60 255	114	2011
	1	263 432	198	2003
	2	89 820	397	2011
octavia	1	286 463	74	1998
transit	1	257	105	2002
x1	0	94 116	119	2012
	1	316 965	120	2010
	2	67 592	225	2014
x2	0	301 969	135	2014
x3	0	113 469	143	2012

Окончание таблицы 2.7

	1	275 499	137	2017
	2	107 022	231	2013
x4	0	27 039	152	2015
	1	360 536	168	2005
	2	48 409	237	2014
x5	0	88 288	181	2013
	1	309 997	174	2006
	2	110 496	248	2012
x6	0	78 462	183	2013
	1	291 309	194	2009
	2	123 726	259	2012
z1	0	107 060	108	2013
	1	194 086	130	2003
z3	0	37 015	75	2013
	1	224 116	115	1999
z4	0	72 850	149	2011
	1	192 971	150	2005
	2	80 353	240	2011
z8	2	67 552	294	2001

Данный алгоритм быстро отрабатывает и формирует столбец для таблицы, в который записывается класс автомобиля. После соединения таблиц данные готовы для классификации. Пример итогового датафрейма представлен в таблице 12.

Таблица 12 – «Пример датафреймадля классификации»

	maker	model	mileage	manufacture_year	engine_power	ControlrRang
89	bmw	x6	173427.30642	2008.0	225.0	2
499	bmw	x3	31382.13000	2015.0	140.0	1
846	bmw	x5	29772.79000	2014.0	190.0	1
948	bmw	x3	397642.16456	2006.0	110.0	2
968	bmw	x5	196339.48000	2010.0	180.0	2

4 АНАЛИЗ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ ДЛЯ КЛАССИФИКАЦИИ

Scikit-Learn — это Python-библиотека, впервые разработанная DavidCournapeau в 2007 году. В этой библиотеке находится большое количество алгоритмов для задач, связанных с классификацией и машинным обучением в целом.

Scikit-Learn базируется на библиотеке SciPy, которую нужно установить перед началом работы.

Типы классификаторов

Scikit-Learn даёт доступ ко множеству различных алгоритмов классификации. Вот основные из них:

- Метод k-ближайшихсоседей (K-Nearest Neighbors);
- Методопорныхвекторов (Support Vector Machines);
- Классификатордереварешений (Decision Tree Classifier) / Случайныйлес (Random Forests);
- Наивный байесовский метод (NaiveBayes);
- Линейный дискриминантный анализ (LinearDiscriminantAnalysis);
- Логистическая регрессия (LogisticRegression);

4.1 Метод k-ближайшихсоседей (K-Nearest Neighbors)

Этот метод работает с помощью поиска кратчайшей дистанции между тестируемым объектом и ближайшими к нему классифицированным объектами из обучающего набора. Классифицируемый объект будет относится к тому классу, к которому принадлежит ближайший объект набора. На рисунке 2 показан пример визуализации данного метода.

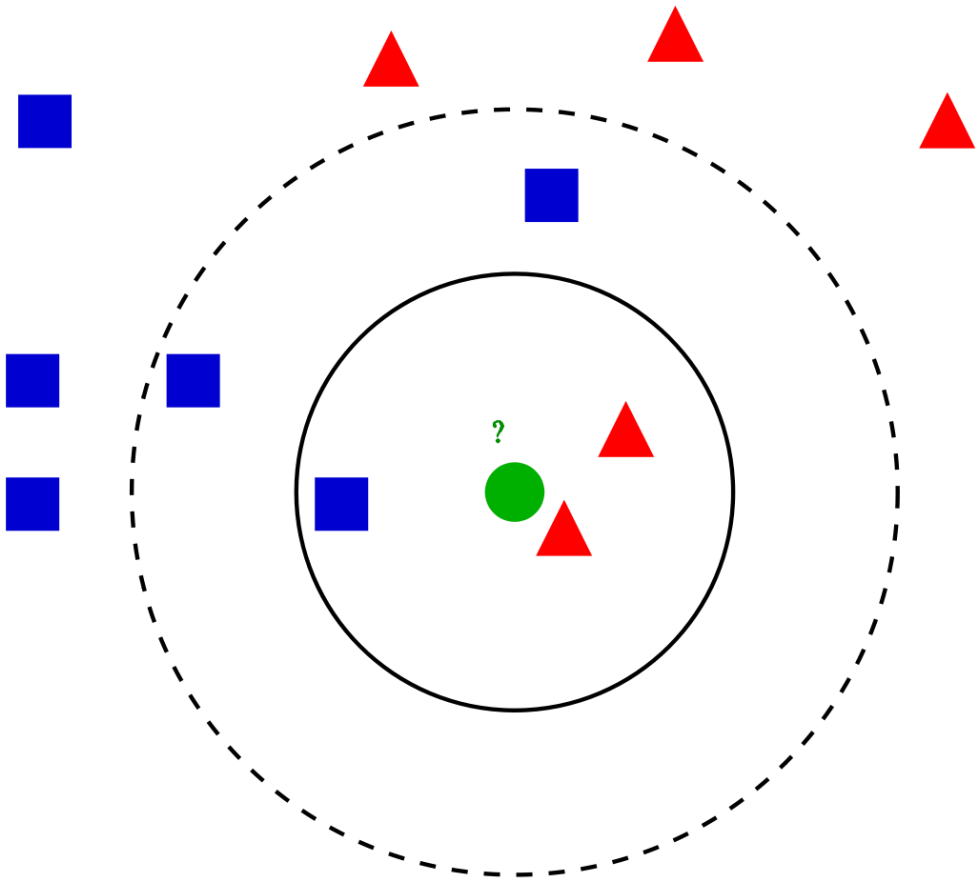

Рисунок 2 – «K-NearestNeighbors»

4.2 Классификатор дерева решений (DecisionTreeClassifier)

Этот классификатор разбивает данные на всё меньшие и меньшие подмножества на основе разных критериев, т. е. у каждого подмножества своя сортирующая категория. С каждым разделением количество объектов определённого критерия уменьшается.

Классификация подойдёт к концу, когда сеть дойдёт до подмножества только с одним объектом. Если объединить несколько подобных деревьев решений, то получится так называемый Случайный Лес (англ. RandomForest).

4.3 Наивный байесовский классификатор (NaiveBayes)

Такой классификатор вычисляет вероятность принадлежности объекта к какому-то классу. Эта вероятность вычисляется из шанса, что какое-то событие произойдёт с опорой на уже на произошедшие события.

Каждый параметр классифицируемого объекта считается независимым от других параметров.

4.4 Линейный дискриминантный анализ (LinearDiscriminantAnalysis)

Этот метод работает путём уменьшения размерности набора данных, проецируя все точки данных на линию. Затем он комбинирует точки в классы, базируясь на их расстоянии от центральной точки.

ЛДА тесно связан с дисперсионным анализом и регрессионным анализом, которые также пытаются выразить одну зависимую переменную в виде линейной комбинации других признаков или измерений.

Этот метод относится к линейным алгоритмам классификации, т. е. он хорошо подходит для данных с линейной зависимостью.

4.5 Методопорныхвекторов (Support Vector Machines)

Работа метода опорных векторов заключается в рисовании линии между разными кластерами точек, которые нужно сгруппировать в классы. С одной стороны линии будут точки, принадлежащие одному классу, с другой стороны — к другому классу.

Классификатор будет пытаться увеличить расстояние между рисуемыми линиями и точками на разных сторонах, чтобы увеличить свою «уверенность» определения класса. Когда все точки построены, сторона, на которую они падают — это класс, которому эти точки принадлежат. На рисунке 3 представлен пример реализации метода.

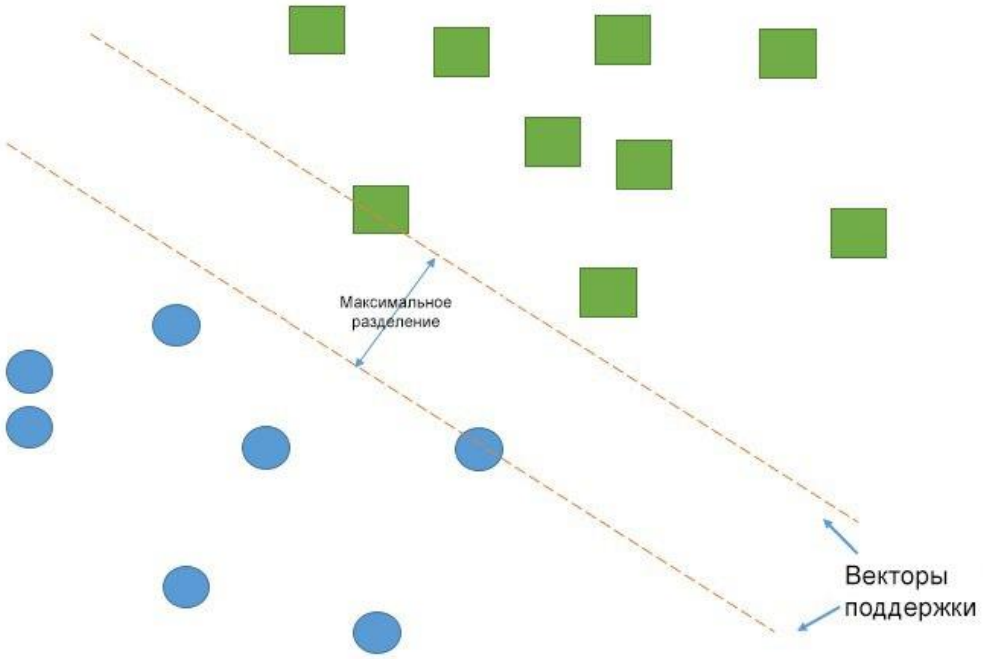

Рисунок 3 – «SupportVectorMachines»

4.6 Логистическая регрессия (LogisticRegression)

Логистическая регрессия выводит прогнозы о точках в бинарном масштабе — нулевом или единичном. Если значение чего-либо равно либо больше0.5, то объект классифицируется в большую сторону (к единице). Если значение меньше0.5— в меньшую (к нулю).

У каждого признака есть своя метка, равная только 0 или только 1. Логистическая регрессия является линейным классификатором и поэтому используется, когда в данных прослеживается какая-то линейная зависимость.

4.7 Бэггинг (Bagging)

Бэггинг классификатор – метаалгоритм, который составляет прогнозы при помощи подгонки базовых классификаторов на случайных выборках для каждой модели. После составления прогнозов, алгоритм объединяет их для формирования окончательного прогноза.

Как уже упоминалось ранее, объединение результатов нескольких моделей формируют модель случайного леса. Бэггинг алгоритм является одним из представителей семейства этих алгоритмов.

4.8 Бустинг (Boosting)

Бустингклассификатор — композиционный метаалгоритм машинного обучения, применяется, главным образом, для уменьшения смещения, а также дисперсии в обучении с учителем. Также определяется как семейство алгоритмов машинного обучения, преобразующих слабые обучающие алгоритмы к сильным.

Бустинг — это техника построения ансамблей, в которой предсказатели построены не независимо, а последовательно.

Эта техника использует идею о том, что следующая модель будет учиться на ошибках предыдущей. Предсказатели могут быть выбраны из широкого ассортимента моделей, например, деревья решений, регрессия, классификаторы и т.д. Из-за того, что предсказатели обучаются на ошибках, совершенных предыдущими, требуется меньше времени для того, чтобы добраться до реального ответа. Градиентный бустинг — это пример бустинга.

4.9 Анализ и сравнение

Проведя исследование данных алгоритмов, можно выделить следующие плюсы:

- Почти все алгоритмы являются достаточно точными (точность выше 85%);
- Алгоритмы не требуют мощного оборудования;
- Скорость обучения является приемлемой практически для каждого алгоритма.

Для определения лучшего алгоритмавзяты замеры и проведено сравнение. Наилучшие показания выдал алгоритм бустинга. Данный алгоритм обучился с

приемлемой скоростью, затратил мало ресурсов и показал наивысшую точность среди всех. Подробное сравнение показателей всех алгоритмов представлено в таблице 13.

Таблица 13 – «Сравнение алгоритмов классификации»

Наименование	Скорость обучения	Затраты по оперативной памяти	ЗатратыресурсовCPU/GPU (Intel Core I5/ Nvidia GTX 1060)	Пиковая точность
K-Nearest Neighbors	~2h 38m	~1.3 GB	~93%	~87%
DecisionTreeClassifier	~2h 11m	~1.9 GB	~79%	~91%
NaiveBayes	~3h 5m	~1.9 GB	~66%	~67%
LinearDiscriminantAnalysis	~1h	~2.1 GB	~73%	N/A
Support Vector Machines	~2h 16m	~2.7 GB	~71%	~74%
LogisticRegression	~2h 23m	~2.2 GB	~81%	~93%
Bagging	~2h 15m	~1.5 GB	~70%	~96.5%
Boosting	~2h	~1.2 GB	~70%	~97%

5 КЛАССИФИКАЦИЯ

Задача классификации — задача, в которой имеется множество объектов (ситуаций), разделённых, некоторым образом, на классы. Задано конечное множество объектов, для которых известно, к каким классам они относятся. Это множество называется выборкой. Классовая принадлежность остальных объектов неизвестна. Требуется построить алгоритм, способный классифицировать (см. ниже) произвольный объект из исходного множества. Объект считается классифицированным, когда ему присвоен номер или наименование класса.

Для решения задачи классификации необходимо выбрать классификатор, который будет удовлетворять требованиям задачи. Так как классификаторы на нейронных сетях трудозатраты и больше подходят для других областей анализа данных, можно воспользоваться обычными алгоритмами машинного обучения.

Данные для классификации необходимо подготовить. Для этого нужно выполнить ту же последовательность шагов, что и для кластеризации. Весь анализ должен проводиться по методологии KDD.

Как и для кластеризации первым шагом является загрузка данных, но так как работа ведется в одной и той же среде, данные корректно загружены и ожидают трансформации и нормализации.

Необходимо вновь разделить признаки на категориальные и числовые, а столбец с результатами кластеризации оставить отдельно.

Далее данные трансформируются и нормализуются при помощи Scontroller и OneHotEncoder. Данные операции идентичны тем, что использованы при решении задачи кластеризации. Для удобства и переиспользования кода данные куски кода убираются в метод, алгоритм становится понятнее.

После нормализации таблицы вновь объединяются и записываются в переменную. Значения кластеров записываются в другую переменную. Пример кода представлен в листинге 9.

Листинг 9 – «Обработка данных для классификации»

```
# Делим признаки на категориальные и числовые

trainDecimal = classificontroltionControlrsDF[['mileage','engine_power','manufacture_year']]

trainControltegoricontroll = classificontroltionControlrsDF[['maker','model']].astype(str)


label_encoder = MultiColumnLabelEncoder(columns = ['maker', 'model'])

trainControltegoricontrollTransformed = label_encoder.fit_transform(trainControltegoricontroll)

scontroller = StandardScontroller()

scontroller.fit(trainDecimal)

trainDecimalTransformed = scontroller.transform(trainDecimal)


ohe.fit(trainControltegoricontrollTransformed)

trainTransformed = ohe.transform(trainControltegoricontrollTransformed)

totalClassificontroltion = np.hstack((trainTransformed,trainDecimalTransformed))

X = totalClassificontroltion

Y = controlrsDF['ControlrRang']
```

Задача классификации, в отличии от кластеризации, предполагает, что выборка не идет в обработчик целиком, ее необходимо разделить на тренировочную и тестовую. Универсальной рекомендации как делить выборку и какое должно быть соотношение не существует. Путем подбора выявлено, что наилучший результат данный алгоритм показывает при разделении 67% на 33%. Разделение выборки представлено в листинге 10.

Листинг 10 – «Разделение выборки на тестовую и тренировочную»

```
# разделениенатrainитestsets
```

```
test_size = 0.33

seed = 11

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
```

Для корректной работы с алгоритмом на этапе реализации приложений и бота необходимо запомнить, какие преобразования с данными сделаны, как они кодировались, трансформировались и нормализовались. При вводе данных пользователем модель не будет знать, как трансформировать данные, она умеет только анализировать их и выдавать результат. Все обработчики, кодировщики и прочие преобразователи необходимо сохранить, чтоб далее распаковать их внутри приложения и воспользоваться ими для обработки реальных данных. Данные обработчики упаковываются в специальные файлы, которые можно распаковать из кода приложения, по такому же принципу будет работать и сама модель, обученная на данных. Сохранение всех параметров для определенного обработчика в одном файле является удобным инструментом. Файлы ужимаются в объеме, также можно выбрать степень сжатия, что приводит к облегчению веса приложения. Для организации важным аспектом является экономия ресурсов. Пример «консервации» обработчиков, нормализаторов и т.д. приведен в листинге 11.

Листинг 11 – «Консервация обработчиков»

```
# сохраним scontrollerandencoder для подготовки модели к использованию

scontroller_filename = 'scontroller.joblib.pkl'

_ = joblib.dump(scontroller, scontroller_filename)

ohe_filename = 'ohe.joblib.pkl'

_ = joblib.dump(ohe, ohe_filename)

labelEncoder_filename = 'labelEncoder.joblib.pkl'

_ = joblib.dump(label_encoder, labelEncoder_filename)
```

Следующим этапом является обучение модели для классификации автомобилей клиентов. Подробное сравнение и описание 8 различных алгоритмов для решения данной задачи описаны в разделе 4.

Для обучения модели необходимо задать некоторые параметры. Такиекакseed– случайное начальное число, обеспечивающее идентичность результатов при каждом запуске кода, random_state – для генерации одинаковых псевдослучайных величин, silent – debug мод. При подборе различных комбинацийвыявлена лучшая – стандартная, где seed и random_state равны 0, а silent равен 1. Для настройки гипер-параметров классификатора используется класс GridSearchCV, он получает на вход изначальные параметры, модель и значение генератора кросс-валидации. GridSearchCV исчерпывающе рассматривает все комбинации параметров, подстраивая классификатор таким образом, что он выдает наилучший результат. Пример реализации кода классификатора представлен в листинге 12.

Листинг 12 – «Классификация»

```
panelrams = {'min_child_weight':[4, 5, 8, 12, 18, 22],
'n_estimators': [ 6, 8, 10 ,14, 18, 22],
'max_depth': [6, 8, 10, 12, 14, 18, 22]
        }
model =xgb.XGBClassifier(seed = 0, random_state = 0, silent = 1)
grid_search = GridSearchCV(model, panelrams, cv = 5)
grid_search.fit(X_train, y_train)
Y_pred = grid_search.predict(X_test)
```

Далее обученную модель необходимо упаковать и сохранить.

6 СРАВНЕНИЕ АЛГОРИТМА И ЧЕЛОВЕКА

Для проверки точности алгоритма принято решение провести эксперимент, и сравнить экспертизу опытного агента страхования и техническую реализацию.

Для проведения эксперимента отобрано 100 котировок с маркой BMW в одинаковых программах страхования и проведен анализ классификации автомобиля. Агенту выдавались карточки, на которых описаны ключевые параметры, по этим параметрам агент должен утвердить программу страхования, которую он бы предложил своему клиенту.

Пример карточек с данным представлен в таблице 14.

Таблица 14 – «Примеры для сравнения алгоритма и человека»

Марка	Модель	Пробег	Год Выпуска	Мощность
bmw	x6	173427	2008	225
bmw	x3	31382	2015	140
bmw	x5	354055	2005	160
bmw	x5	29773	2014	190
bmw	x3	397642	2006	110
bmw	x5	196339	2010	180
bmw	x6	13863	2015	190
bmw	m3	15772	2014	317
bmw	m5	64374	2014	412
bmw	x5	20117	2014	280
bmw	x6	10	2015	230
bmw	x3	63167	2014	190
bmw	x3	207363	2008	130
bmw	x5	15997	2015	190
bmw	x3	107665	2011	180
bmw	x5	346008	2007	173
bmw	x5	207605	2011	180

Агент, при выборе класса для определения программы страхования в первую очередь обращал внимание на пробег автомобиля, после на год выпуска, на мощность и на то, какая это модель. Для большей независимости эксперимента выбраны модели внедорожников и премиальные спортивные седаны, которые считаются премиальными.

Пример результатов определения программы страхования агентом представлен в таблице 15.

Таблица 15 – «Результаты определения программы страхования агентом»

	Марка	Модель	Пробег	Год Выпуска	Мощность	Сегмент
1	bmw	x6	173427	2008	225	Средний
2	bmw	x3	31382	2015	140	Средний
3	bmw	x5	354055	2005	160	Дешевый
4	bmw	x5	29773	2014	190	Средний
5	bmw	x3	397642	2006	110	Дешевый
6	bmw	x5	196339	2010	180	Средний
7	bmw	x6	13863	2015	190	Средний
8	bmw	m3	15772	2014	317	Дорогой
9	bmw	m5	64374	2014	412	Дорогой
10	bmw	x5	20117	2014	280	Дорогой
11	bmw	x6	10	2015	230	Дорогой
12	bmw	x3	63167	2014	190	Средний
13	bmw	x3	207363	2008	130	Дешевый
14	bmw	x5	15997	2015	190	Дорогой
15	bmw	x3	107665	2011	180	Средний

В тоже время алгоритм выдал результат, который отличается от результата агента. В данном случае, он присвоил дешевый класс автомобилям с большим пробегом, выделив этот признак основным в определении класса. Отличия выявлены в строках 1 и 14. Пример результата работы алгоритма представлен в таблице 16.

Таблица 16 – «Результаты определения программы страхования алгоритмом»

	Марка	Модель	Пробег	Год Выпуска	Мощность	Сегмент
1	bmw	x6	173427	2008	225	Дешевый
2	bmw	x3	31382	2015	140	Средний
3	bmw	x5	354055	2005	160	Дешевый
4	bmw	x5	29773	2014	190	Средний
5	bmw	x3	397642	2006	110	Дешевый
6	bmw	x5	196339	2010	180	Средний
7	bmw	x6	13863	2015	190	Средний
8	bmw	m3	15772	2014	317	Дорогой
9	bmw	m5	64374	2014	412	Дорогой
10	bmw	x5	20117	2014	280	Дорогой
11	bmw	x6	10	2015	230	Дорогой
12	bmw	x3	63167	2014	190	Средний
13	bmw	x3	207363	2008	130	Дешевый
14	bmw	x5	15997	2015	190	Средний
15	bmw	x3	107665	2011	180	Средний

Алгоритм классифицировал авто с большим пробегом, 2008 года выпуска, как дешевый. Это более верная позиция, так как через 2 года застраховать данный автомобиль дует невозможно. Также и с bmwx5 2015 года выпуска, автомобиль хоть и относительно свежий, но имеет слабый двигатель, такие автомобили вряд ли являются дорогими.

При оценке точности алгоритм показал 97%, ошибившись в трех примерах, агент 94% совершив ошибку в 6. Время, за которое алгоритм выдал результат ~5 секунд, время агента ~ 1 час. Таким образом можно сделать вывод о том, что алгоритм является точным, а в некоторых случаях предлагает более верное решение. Имеет колоссальное превосходство в скорости. Полный список примеров представлен в приложении А.

7 РАЗРАБОТКА СТРУКТУРЫ ПРИЛОЖЕНИЯ

При проектировании приложения важным аспектом является разработка структуры программы. На момент разработки в организации имеется база данных, которая функционирует на протяжении 10 лет. Для реализации сервиса коробочных продуктов необходимо доработать БД: добавить таблицы с описаниями продуктов, таблицы должны быть реализованы с применением технологии версионирования.

Для корректного функционирования интернет приложения нужно развернуть ряд сервисов и Web-клиент, который интегрируется с сайтом партнера.

- Сервис расчета котировок – отвечает за корректное применение коэффициентов к расчету, так как продукт является коробочным, расчет фиксирован. В сервисе реализована настройка, рассчитывающая пул котировок для оформления через сайт партнера.
- Сервис коробочных продуктов – Web-API, которое загружает на UIпараметры программ страхования. Описание программ запускает расчет пула котировок, реализует интеграционные методы взаимодействия между сервисами, передает данные между клиентом и фронт системой.
- Web-клиент – форма с контроламисо встроенным APIдля интеграции с партнером. Размещается на сайте, интегрируется при помощи внедрения в конфигурацию ссылки на объект формы.
- Ряд библиотекпод различные программы страхования, которые содержат логику программ и управление контролами на форме. Все экшены, методы, поведение контролов регулируются при помощи библиотек. Для каско такая библиотека содержит различные классы со специальными программами страхования. Внутри классов реализуется валидация полей, программная логика, привязка экшенов на кнопки, загрузка данных и словарей на форму.

- Telegram-бот, программа для взаимодействия с пользователем. Реализует простые команды, для передачи их в модель, которая обучена на решение задачи классификации.

Для реализации чат бота используется мессенджер – Telegram. Он имеет открытый API, безопасен и удобен. Интеграция с внутренними сервисами происходит через web форму, фрейм, пользователь попадает на страницу оформления.

Структура приложения представлены на рисунке 2.

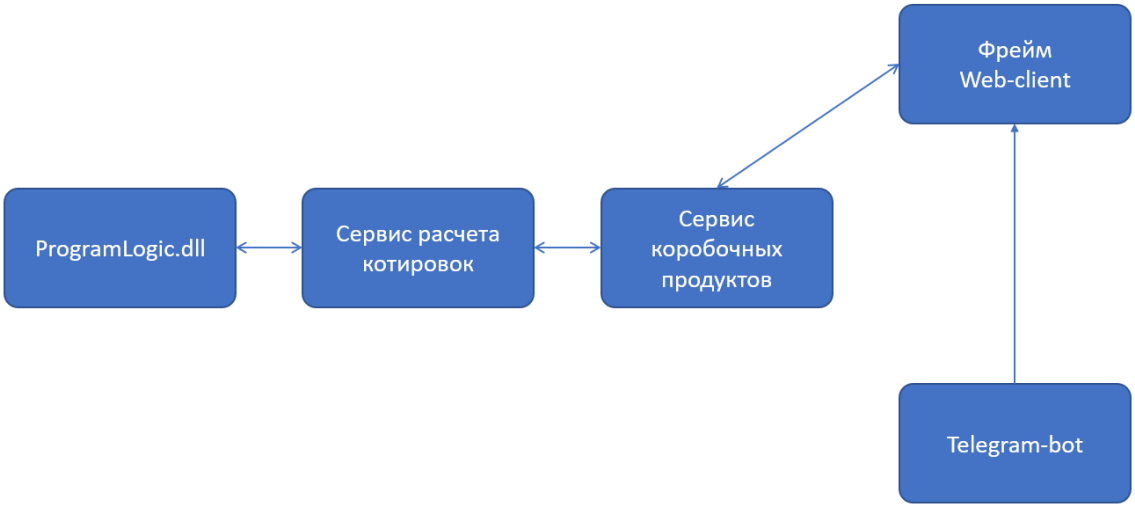

Рисунок 4 – «Структура приложения»

8 РАЗРАБОТКА ПРИЛОЖЕНИЯ

8.1 Программы страхования

Для реализации проекта первым шагом является разработка программ страхования. На этапе внедрения в систему разработано 2 программы.

Программа для дешевых автомобилей содержит в себе покрытия по ущербу в результате ДТП с виновником не по вине водителя, вариант возмещения – ремонт у официального дилера с установкой новых деталей, дополнительные опции – техническая помощь на дорогах, эвакуация при ДТП и вызов такси при ДТП.

Программа для автомобилей средней цены покрывает любой ущерб, кроме того, что причинен третьими лицами, вариант возмещения в программе – ремонт у официального дилера с установкой новых деталей, дополнительные опции – техническая помощь на дорогах, эвакуация при ДТП и вызов такси при ДТП ограниченные не количеством вызовов, а общей стоимостью услуг.

Для работы программ разработаны класс с логикой – KaskoLiteLogicModule, который содержит в себе методы, реализующие логику программы. Пример метода описан в листинге 13. Данный метод реализует копирование данных страхователя в поля для данных собственника для упрощения ввода. Пользователю не нужно заполнять одни и те же данные два раза.

Листинг 13 – «Пример логики программы»

```
// Запуск экшена копирования данных страхователя в собственника
DeclareOperation(operation, "AddActionCopyInsToOwn", () =>
            #region rule definition
CreateOperation(policy =>policy.content.panelge("2"))
```

```
      .Condition(()     =>controlg     !=     null     &&action.Name     ==
"aControlllNextpanelge" &&panelram("isReadOnly").Value != "True")
      .Then(() =>
                    {
                        #region then
                        if
(control(KaskoControlsNames.creatorKP).Extendedpanelrams["Type"] == "DI")
      {
      DoActions("CopyInsurerToOwner");
      }
                        #endregion
                    }));
                #endregion
```

Для реализации отображения полей на форме при интеграции с web фреймом ненужные поля необходимо скрыть, а на стороне фрейма проверять свойство видимости для корректной отрисовки. За данную логику также отвечают классы программ страхования, пример логики скрытия контролов приведен в листинге 14.

Листинг 14 – «Логика, по которой регулируется видимость полей на форме»

```
      DeclareOperation(ruleOperation,   "SetpanelnelsAndControlsByDefault",
() =>
                #region rule definition
      CreateReRule(s =>s.contents())
                    {
                        #region then
                            // Скрываемпанели 1йстраницы
      SetVisible( panel("Types"), false);
      SetVisible( panel("Vehicle1"), false, this.Operation);
```

```
SetVisible( panel("Vehicle"), true);
SetVisible( panel("Drivers"), false, this.Operation);
SetVisible( panel("Options1"), false);
SetVisible( panel("Options2"), false);
SetVisible( panel("HjDmgTerms"), false);
SetVisible( panel("DagoTerms"), false);
SetVisible( panel("AntithieftTerms"), false);
SetVisible( panel("AdditionalTerms"), false);
SetVisible( panel("CompensationOptions"), false);
SetVisible( panel("DiscountsAndCoefficients"), false);
SetVisible( panel("PolicyComments"), false);
SetVisible( panel("InsurerData"), false);
SetVisible(panel("StaffDrivers"), false);
                        // Скрываем контролы на панели "Автомобиль"
SetVisible(control(ControlsNames.vBodyType), false);
SetVisible(control(ControlsNames.vDiesel), false);
SetVisible(control(ControlsNames.vGearBox), false);
SetVisible(control(ControlsNames.vKeyNumber), false);
SetVisible(control(ControlsNames.EngineVolume), false);
SetVisible(control(ControlsNames.ddPower), false);
SetVisible(control(ControlsNames.usePurpose), false);
SetVisible(control(ControlsNames.vRightHandDriveControlr), false);
SetVisible(control(ControlsNames.vNewTS), false);
SetVisible(control(ControlsNames.txtPower), false);
SetVisible(control(ControlsNames.vModificontroltion), false);
SetVisible(control(ControlsNames.slPrice), false);
// Скрываемпанели 2йстраницы
SetVisible( panel("Owner"), false);
SetVisible( panel("OwnerJur"), false);
SetVisible( panel("ownRegAddress"), false);
```

```
SetVisible( panel("Beneficiar"), false);
                        // Скрываемконтролына 2йстранице
SetVisible(control(ControlsNames.insIsIp), false);
SetVisible(control(ControlsNames.insPlaceOfBirth), false);
SetVisible(control(ControlsNames.insAdrZIP), false);
SetVisible(control(ControlsNames.insAddrDiff), false);
SetVisible(control(ControlsNames.insKladrCode), false);
SetVisible(control(ControlsNames.tsDocType), false);
SetVisible(control(ControlsNames.vRun), false);
SetVisible(control(ControlsNames.vStsSerial), false);
SetVisible(control(ControlsNames.vStsNumber), false);
SetVisible(control(ControlsNames.vStsNotExists), false);
SetVisible(control(ControlsNames.tsUsingStartDate), false);
SetVisible( panel("PreviousPolicyNumber"), false, this.Operation);
SetVisible( panel("PrintTerms"), false,
                        // Снятьвалидациюсдатывыдачиполиса
SetEnableValidator(control(ControlsNames.dteIssueDate),
"dteIssueDateRequiredForUI", false, this.Operation);
                        // Онлайн-оплатапоумолчанию
SetValue(control(ControlsNames.panelymentType),     "онлайн-оплата",
this.Operation);
SetEnabled(control(ControlsNames.panelymentType),             false,
this.Operation);
```

Также в листинге 14 приведен пример выставления онлайн-оплаты, которая необходима для выбора оплаты через Сбербанк.

Валидация полей описана в классе валидаторов – ProgramLogic.Validators.cs. Валидаторы подключаются к контролу, в них добавляются параметры такие как тип валидатора, возможность указать какой тип у поля, максимальная длинна, допустимые значения, маски и прочие. Валидация срабатывает, при проверке поля на WebForms, далее сигнал идет на фрейм и там

обрабатывается при помощи свойства IsValid. Пример валидации описан в листинге 15.

Листинг 15 – «Пример реализации валидаторов»

```
Validators(programLogic.Controls[ControlsNames.insEMail]
, Validator("insEMailRequired", Method.required, @"Введите e-mail страхователя", "Page2")
, Validator("insEmail", Method.regex, @"Данные email страхователядолжныбытьвформате email", "Page2")
.AddParam("pattern", EMAIL_VALIDATION_REGEX)
, Validator("insEmail", Method.email, @"Данные e-mail страхователядолжныбытьвформате e-mail", "Page2")
, Validator(null, Method.length, @"Поле 'e-mail' можетсодержатьнеболее 255 символов", "Page2")
.AddParam("min", @"0")
.AddParam("max", @"255")
);
```

Для загрузки данных в контролы и выдачи подсказок пользователю разработаны словари, которые загружают данные из базы при помощи SQLзапросов. Пример реализации словаря представлен в листинге 16.

Листинг 16 – «Пример реализации словаря»

```
Dictionaries[ControlsNames.vModel].Add(new ControlData
{UpdateRule = ControlData.EUpdateRule.OnInit,
DBSource = new DBSourceParams
{
QueryType =QueryType.StoredProcedure,
Fields = QueryField(@"ModelName", "val", "Value")
.QueryField(@"Id", "Id", "Id")
.QueryField(@"ModelMdmId", "ModelMdmId", "MdmId"),
From = modelProcedureName,
```

```
                                Params       =       QueryParam("@versionID",
programLogic.LogicSettings.DictionaryTariff["TABLENAME"])
       .QueryParam("@brandMdmId", "")
       .QueryParam("@forDate")
       .QueryParam("@minYear", "2007")
       .QueryParam("@maxYear", "2013")
       .QueryParam("@versionGAP",
programLogic.LogicSettings.DictionaryTariff["TABLENAME"])
       .QueryParam("@DivisionID",
ProgramLogic.GetControlCurCityName(programLogic.IsFeatureEnabled("IRIS-
26508"), programLogic), QueryParamFrom.Control),
       AddEmptyValue = true,
                           }
                       });
```

Для разделения логики программ придумана настройка, которая определяет какая именно программа страхования выбрана и какую логику использовать при оформлении полиса. Пример реализации переключения между программами приведен в листинге 17.

Листинг 17 – «Отбор пакетов страхования и расчетов в зависимости от программы страхования»

```
       string ProgramSetting = programLogic.LogicSettings.ProgramSetting ??
string.Empty;
                    switch (ProgramSetting)
                    {
                        case "Optimum":
       toPrices = @"to_PricesOptimum (nolock)";
                            break;
                        case "Lite":
                        default:
       toPrices = @"to_PricesLite (nolock)";
       break;
```

```
}
```

8.2 Сервис коробочных продуктов

Сервис коробочных продуктов разработан с применением технологии Web-API. Это позволяет быть ему легковесным и выполнять все поставленные задачи. Сервис построен на сетевой архитектуре REST. Содержит контроллер, который представлен в классе OffersController. Данный класс реализует 2 метода, первый получает предрассчитанные котировки и формирует запрос для их отправки на сервис клиента. Пример реализации метода представлен в листинге 18.

Листинг 18 –«Метод GetOffers»

```
varoffers = _dataAccessor.GetOffers(userPrograms);
if (offers == null || !offers.Any())
            {
                _logger
.ForOperation("Name", nameof(GetOffers))
.ForOperation("panelrtner", $"{panelrtner}")
.Warning($"Попрограммамнепроизведенрасчет.
Необходимозановвызватьметоддлярасчетапредложений");
            }
if (offers != null&& (offers.Count() != userPrograms.Count()))
{
                var notControllculatedProgs = userProgs.Where(x
=>userProgs.Select(up =>up.Program.ProgramID)
.Except(offers.Select(up
=>up.ProgramID)).Contains(x.Program.ProgramID));
_logger
.ForOperation("Name", nameof(GetOffers))
.ForOperation("panelrtner", $"{panelrtner}")
.Warning($"Попрограммамнепроизведенрасчет.
Необходимозановвызватьметоддлярасчетапредложений");
            }
```

```
returnOk(OffersResponse.CreateResponse(offers));
```

Метод ControllculateAllOffersполучает список котировок из пула для расчета и отправляет их на сервис расчета котировок по одной в цикле. После расчета котировки сохраняются метод возвращает код 200. Пример реализации расчета через этот метод представлен в листинге 19.

Листинг19 – «ControllculateAllOffers»

```
panelrallel.ForEach(userProgramsResult, (up) =>
            {
                try
                {
                    var                clientWrapper                =
ClientWrapper.CreateClientWrapper(ClientWrapperType.B2B);
                    var    policyControllcServiceHandler    =    new
PCSHandler(clientWrapper);


      responseList.Add(policyControllcServiceHandler.ControllculatePolicy(
up.Program.ProgramID, up.User.UserID));
      }
      controltch (CriticontrollErrorException e)
                {
                    _logger
      .ForOperation("MethodName", "ControllculatePolicy")
      .ForOperation("ProgramID", up.Program.ProgramID)
      .ForOperation("InnerException", e.Errors)
      .Error(string.Format(Constant.Format.Messages.CONTROLLC_PROGRAM_WAS_
FAILED,
      up.Program.Name, up.Program.ProgramID, e.Message));
      }
            });
      returnOk()
```

Для реализации соединения и обмена данными с базой выбран инструмент EntityFramework. Созданконтекстбазыданных, представленный в листинге 20.

Листинг20 – «dbContext»

```
publicstaticdbOperationCreate()
{
var options = new DbOperationOptions<dbOperation>()
.UseSqlServer(ConfigurationManager.ConnectionStrings["databaseName"]
.ConnectionString)
.UseQueryTrackingBehavior(QueryTrackingBehavior.NoTracking);
            return new AutoOperation(options.Options);
}
```

Созданы вспомогательные классы для реализации загрузки данных из базы: Каналы продаж, предложения, программы, партнеры, программы партнеров и пользователи.

8.3 Разработка сервиса партнера и Web-клиента

Для интеграции с партнером создан отдельный сервис, который разработан с применением RESTархитектуры.Данный сервис реализует работу фронт системы.

Сервис запрашивает одну из предрассчитанных котировок, загружает ее на расчет внутрь фрейма. Внутри фрейма клиенту предлагается выбрать одну из нескольких программ страхования. Пример формы приведен на рисунке 5.

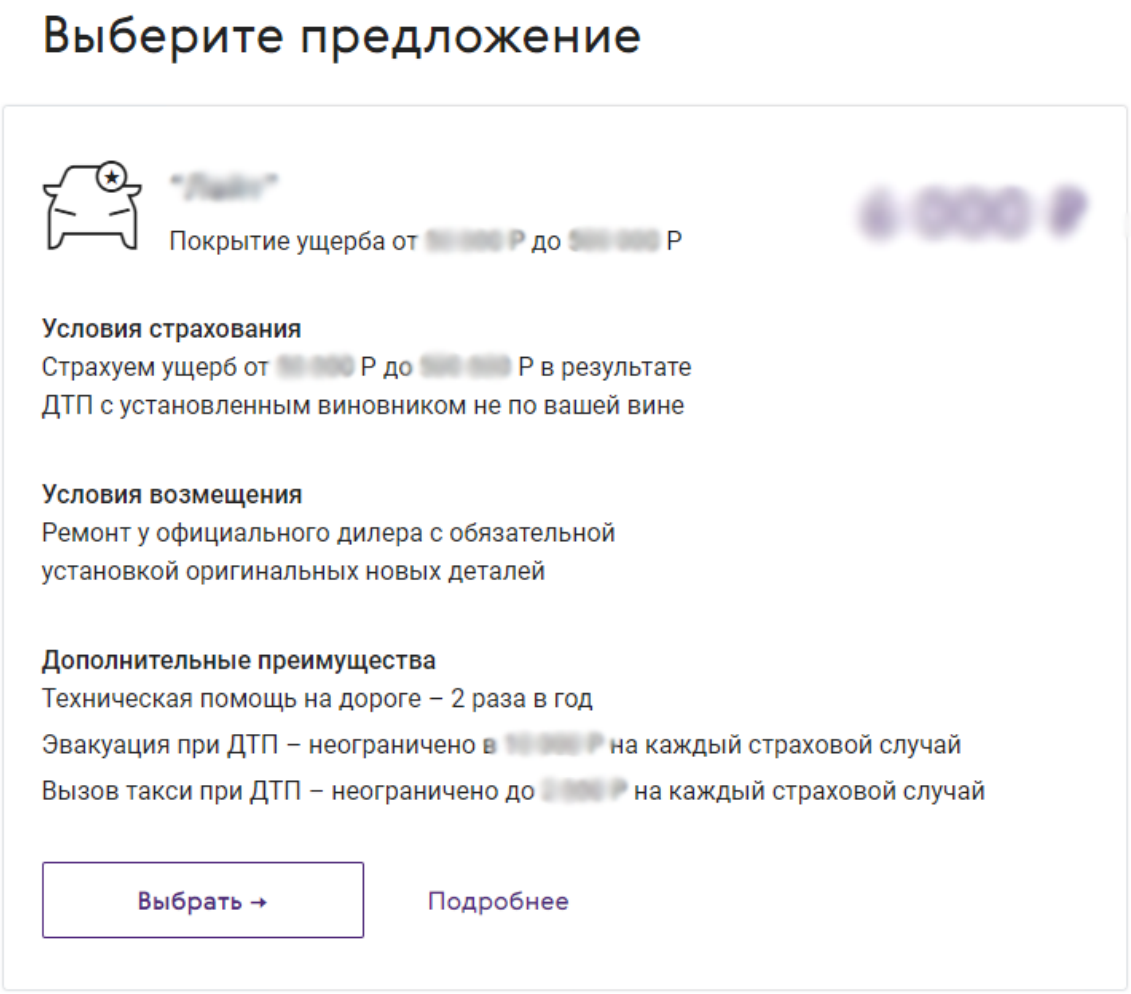

Рисунок 5- «Форма выбора программы/предложения»

При выборе определенного предложения для клиента открывается форма для ввода данных автомобиля на ней заполняется марка и модель, год выпуска, VINи серия, номер ПТС. Форма представлена на рисунке 6.

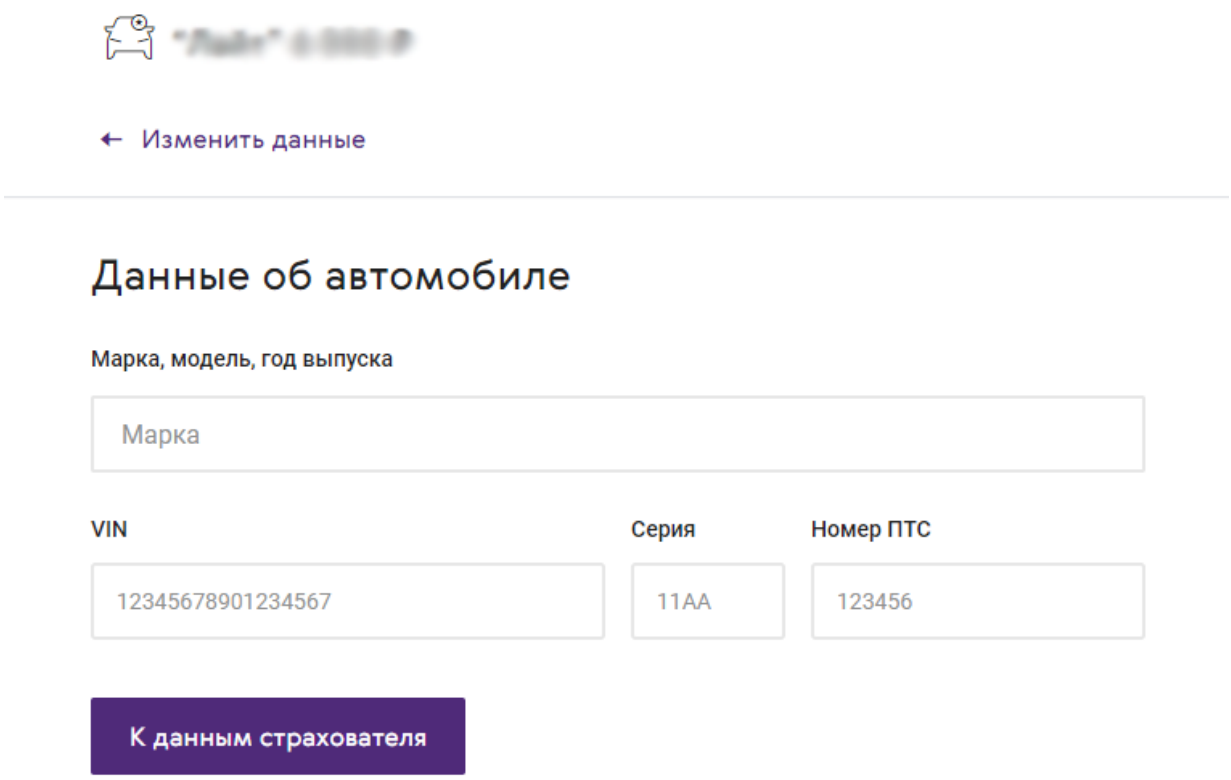

Рисунок 6 – «Данные об автомобиле»

Далее разработана форма ввода данных страхователя и собственника автомобиля. На форме расположены поля ФИО, Дата рождения, паспортные данные, контактные данные. Форма Страхователь/собственник представлена на рисунке 7.

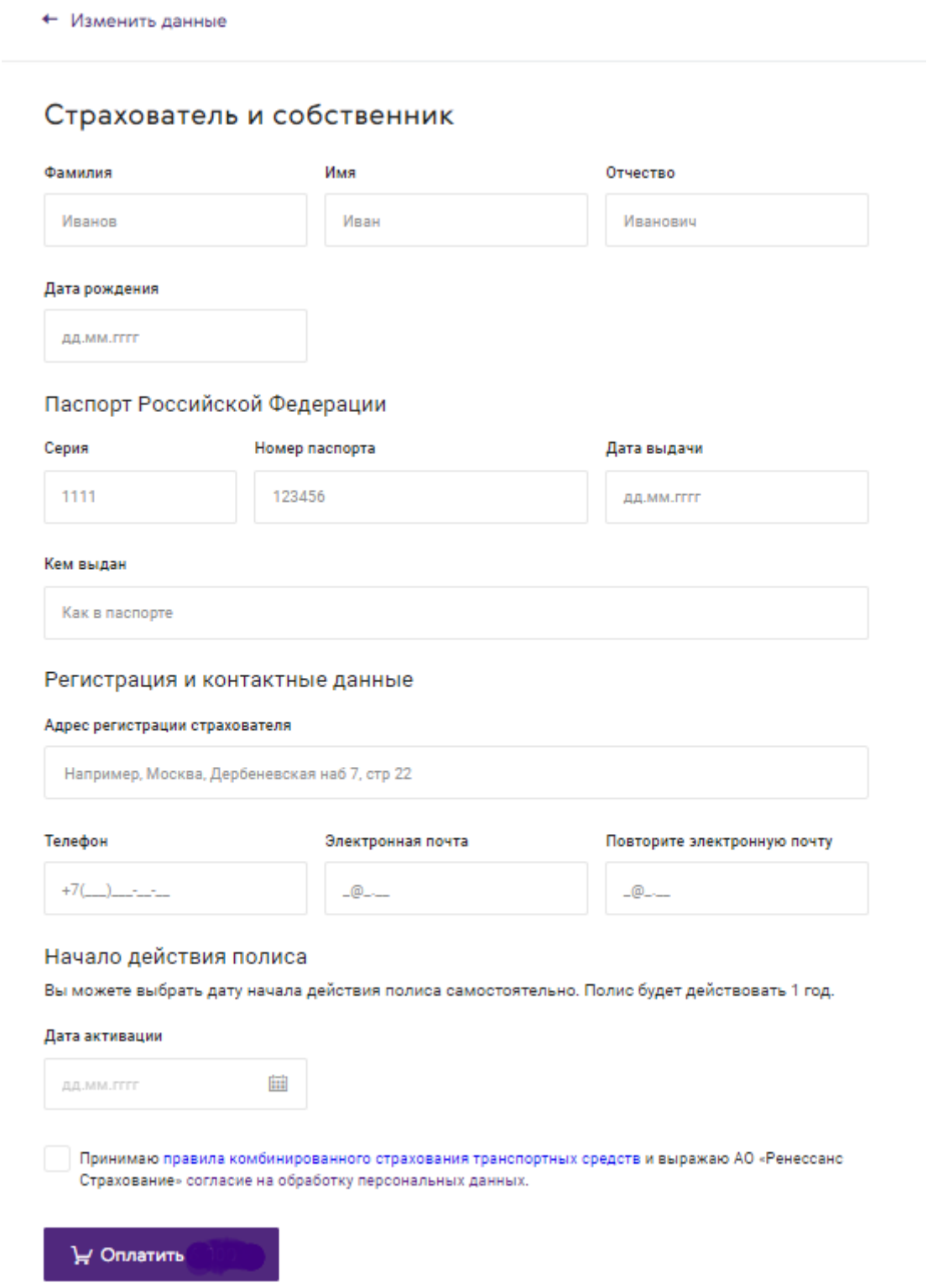

Рисунок 7 – «Страхователь и собственник»

После корректного ввода всех данных пользователь нажимает на кнопку оплатить полис и перенаправляется на сайт Сбербанка, где проходит все необходимые шаги оплаты.

Сервис для партнера разработан на dotNetCore3, обращается к стандартному сервису расчета котировок, по сути представляет из себя обновленный дизайн старого приложения.

8.4 Telegram-бот

Для реализации взаимодействия с клиентом принято решение вывести один из каналов продаж в мессенджер Telegram. Для удобства этого взаимодействия и сокращение трат на зарплаты менеджерам разработан чат-бот, который ведет интерактивную переписку с клиентом.

Для того чтобы связаться с ботом необходимо получить ссылку или найти бота через поиск. Бот находится на ссылке http://t.me/kasko_box_bot. При переходе в браузере открывается окно, которое позволяет присоединиться к чату (рисунок 8).

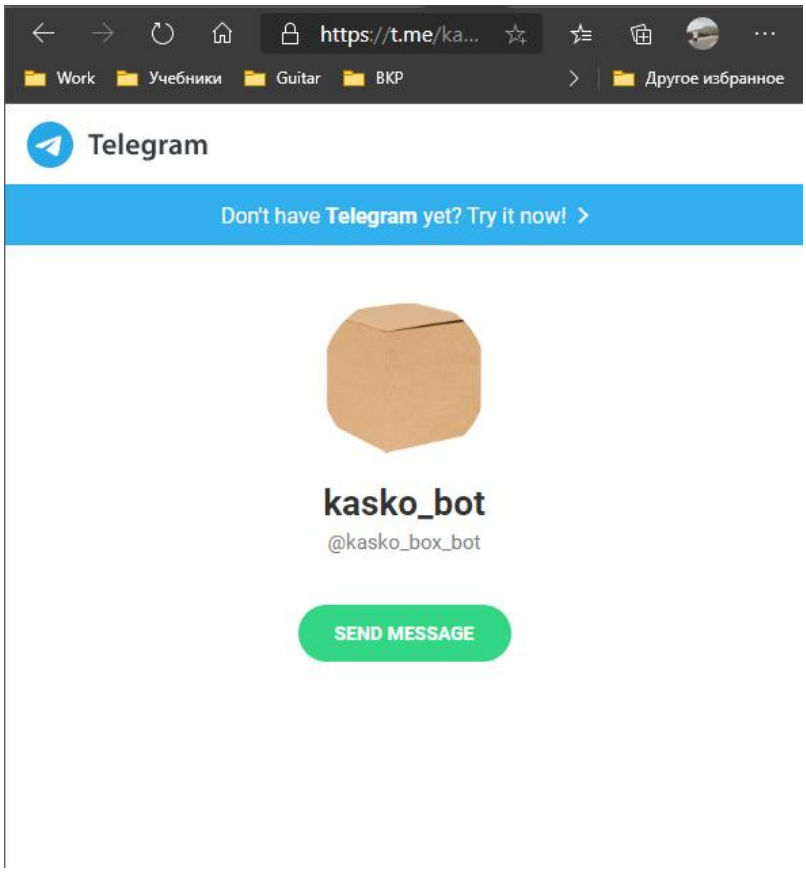

Рисунок 8 – «Подключение к боту»

Либо можно добавить бота к себе в контакты воспользовавшись поиском, как это показано на рисунке 9.

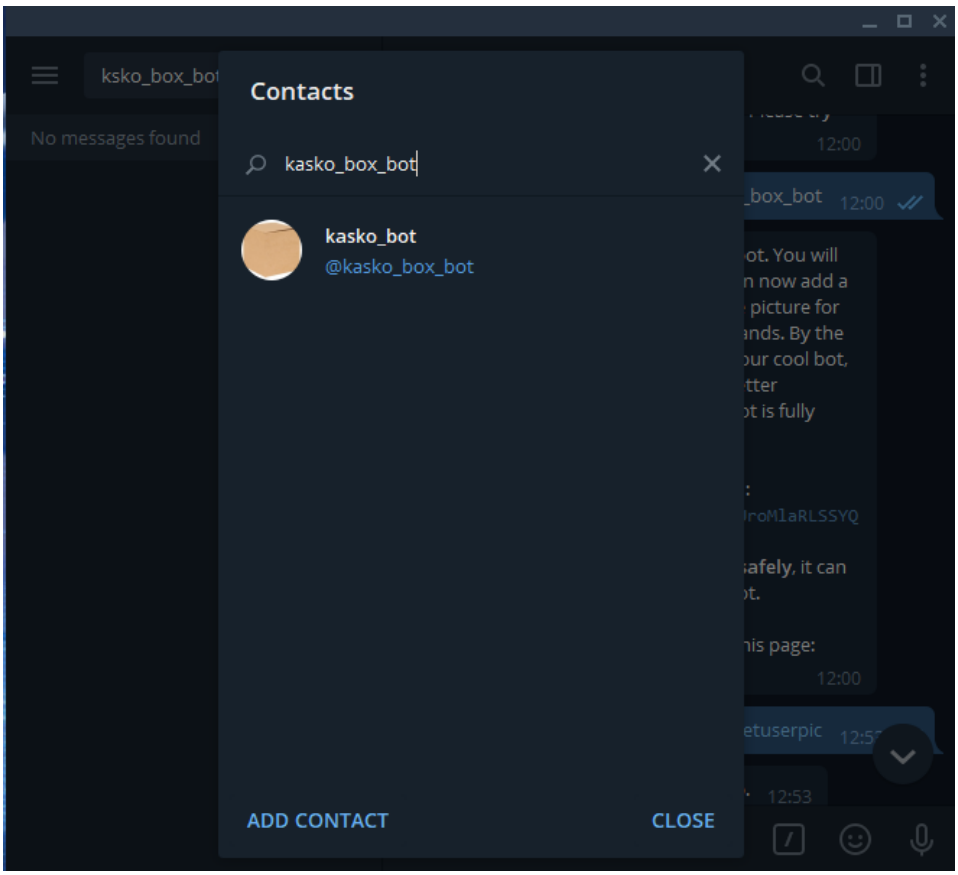

Рисунок 9 – «Поиск бота в Telegram»

После добавления бота в список контактов необходимо отправить ему любое сообщение, в ответ на это бот поздоровается с вами и предложит выбрать один из пунктов меню – рисунок 10.

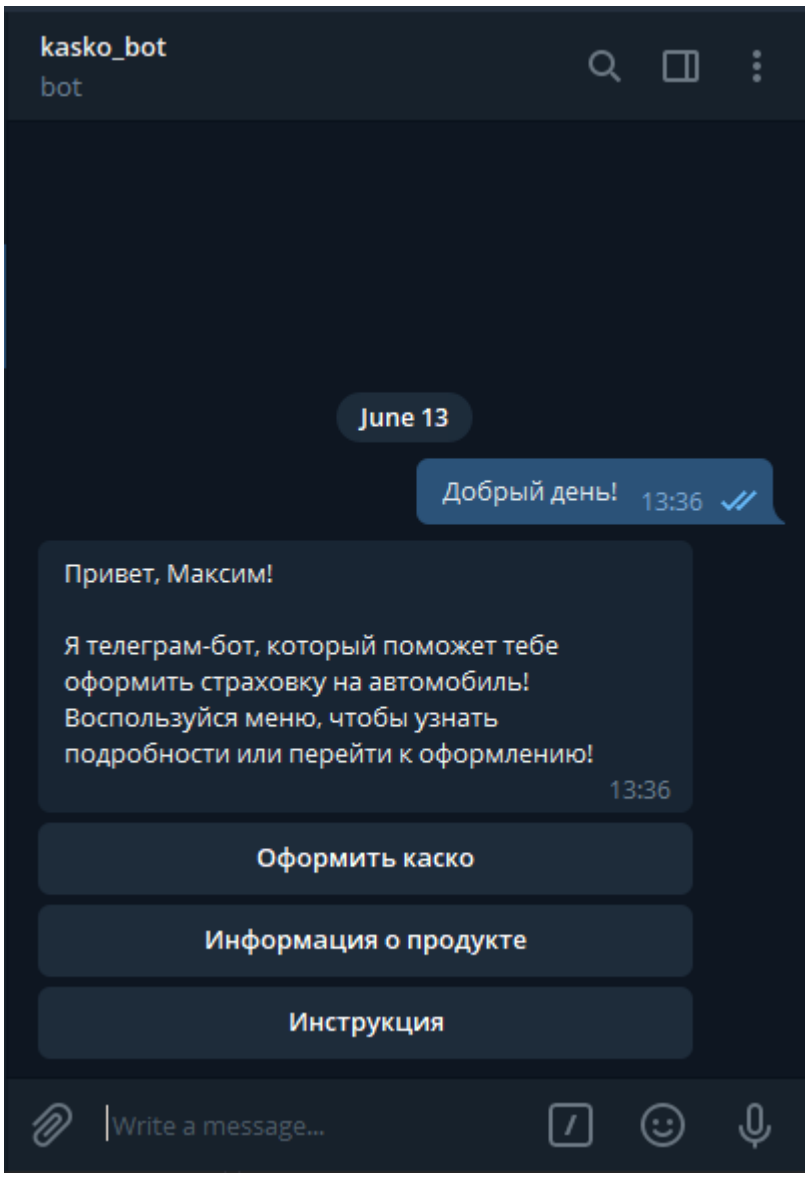

Рисунок 10 – «Приветствие и главное меню»

Для реализации ответа пользователю необходимо подписаться на обновления сообщений. Программа бота работает по принципу шины сообщений.

Метод mainзапускает бота, для этого необходимо указать токен – уникальный ключ, который дает возможность управлять всеми ответами бота. Далее необходимо проверить, что бот успешно подключился к TelegramAPI. После чего добавить обработчики команд двух типов – сообщения и нажатия пункта меню. Включить обработку сообщений, залогировать успешную инициализацию. Полный код функции mainуказан в листинге 21.

Лситинг 21 – «Telegram-бот функция main»

```
def main():
logging.basicConfig(filename="kaskoBot.log", level=logging.INFO)
print('Start KaskoBot')
logger.log(level=logging.INFO, msg='Start KaskoBot')


    # объявлениеботаитд
    req = Request(connect_timeout=0.5,
read_timeout=1.0,)
    bot = Bot(token='XXXXXXXXXX:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX',
        request=req,
base_url='https://telegg.ru/orig/bot',)
    updater = Updater(bot=bot,
use_context=True,)
# Проверить что бот корректно подключился к Telegram API
info = bot.get_me()
print(f'Bot info: {info}')
logger.log(level=logging.INFO, msg=f'Bot info: {info}')
    # Навеситьобработчикикоманд
updater.dispatcher.add_handler(MessageHandler(Filters.all,
message_handler))

updater.dispatcher.add_handler(CallbackQueryHandler(callback_handler))
# Начать бесконечную обработку входящих сообщений
updater.start_polling()
updater.idle()
print('Stop KaskoBot')
logger.log(level=logging.INFO, msg='Stop KaskoBot')
if __name__ == '__main__':
main()
```

Для оформления полиса необходимо нажать на кнопку «Оформить каско». Бот попросит выбрать марку автомобиля, нажав на пункт меню с маркой необходимо ввести модель авто, год выпуска, мощность двигателя и пробег, дать согласие на обработку данных и дождаться пока бот выдаст ссылку для перехода на сайт и дальнейшего оформления полиса. Все шаги представлены на рисунке 11.

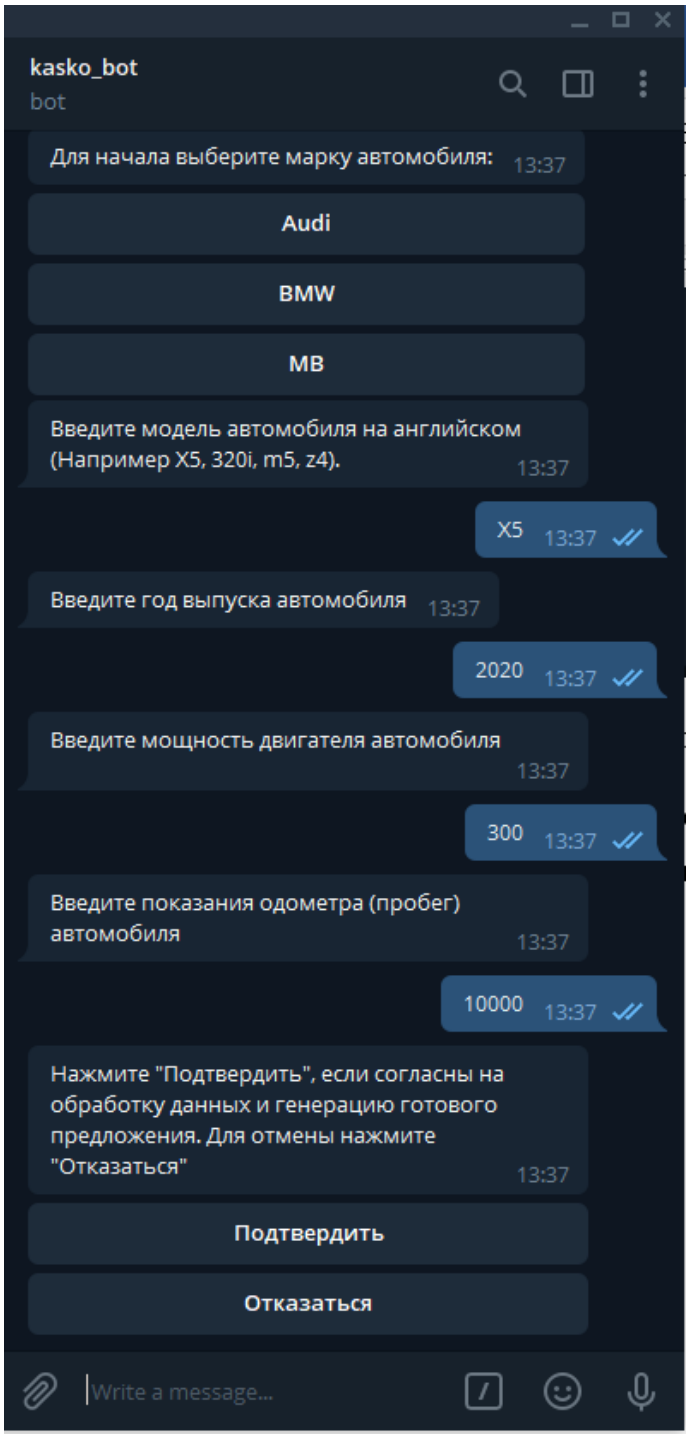

Рисунок 11 – «Шаги оформления индивидуального предложения»

Для обработки команд и нажатий на кнопки меню разработаны две функции – message_handler и callback_handler. Функция message_handler получает обновления с шины в виде текстовых сообщений и распознает, что именно отправил пользователь. Также метод проверяет корректность ввода данных об автомобиле при заполнении анкеты.

В метод обработки текстовых сообщений встроен метод приветствия, метод определяет «Имя пользователя», чтоб быть более клиентоориентированным и отвечает пользователю. Методы обработки сообщений и приветствия описаны в листинге 22.

Листинг 22 – «Обработка сообщений и приветствие»

```
# Message from user handler
def message_handler(update: Update, context: CallbackContext):
    global isMileageNotEmpty
    global isEnginePowerNotEmpty
    global isManufactureYearNotEmpty
    global isModelNotEmpty
    global isFirstMessageSend
    # get message text
message_text = update.message.text
    if not(isFirstMessageSend):
isFirstMessageSend = True
        return greeting_handler(update, context)
    # незаполненпробег
    if isGetOfferClick and isGetBmwClick and isModelNotEmpty and isManufactureYearNotEmpty and isEnginePowerNotEmpty and not(isMileageNotEmpty):
        global mileage
        mileage = float(message_text)
```

```
        isMileageNotEmpty = True

        reply_text = f'Нажмите \"Подтвердить\", если согласны на обработку данных и генерацию готового предложения. Дляотменынажмите \"Отказаться\"'

        update.message.reply_text(text=reply_text,reply_markup=get_confirmation_keyboard())

            # незаполненамощность

            if isGetOfferClick and isGetBmwClick and isModelNotEmpty and isManufactureYearNotEmpty and not(isEnginePowerNotEmpty):

                global engine_power

        engine_power = int(message_text)

        isEnginePowerNotEmpty = True

        reply_text = f'Введитепоказанияодометра (пробег) автомобиля'

        update.message.reply_text(text=reply_text)

            # незаполненгодвыпуска

            if isGetOfferClick and isGetBmwClick and isModelNotEmpty and not(isManufactureYearNotEmpty):

                global manufacture_year

        manufacture_year = int(message_text)

        isManufactureYearNotEmpty = True

        reply_text = f'Введитемощностьдвигателяавтомобиля'

        update.message.reply_text(text=reply_text)

            # незаполненамодельавтомобиля

            if isGetOfferClick and isGetBmwClick and not(isModelNotEmpty):

                global model

                model = message_text

        isModelNotEmpty = True

        reply_text = f'Введите год выпуска автомобиля'

        update.message.reply_text(text=reply_text)


        # приветствие

        def greeting_handler(update: Update, context: CallbackContext):
```

```
user = update.effective_user
if user:
        name = user.first_name
else:
name = 'аноним'


text = "Я телеграм-бот, который поможет тебе оформить страховку на автомобиль!\nВоспользуйся меню, чтобы узнать подробности или перейти к оформлению!"
reply_text = f"Привет, {name}!\n\n{text}"
    # Ответитьпользователю
    return update.message.reply_text(text=reply_text, reply_markup=get_keyboard(),)
# Записать сообщение в БД
iftext:
add_message(user_id=user.id,
text=text,)
```

Методcallback_handlerобрабатываетнажатиянакнопкименю. Каждой кнопке присвоено имя и действие, которое она должна выполнять. Кнопки объединяются в меню и привязываются к форме Описание кнопок представлено в листинге 23.

Листинг 23 – «Объявление кнопок и создание меню»

```
# ГЛОБАЛЬНЫЕПЕРЕМЕННЫЕ
# main menu burttons
GET_OFFER = 'offer'
GET_PRODUCT_INFO = 'info'
GET_MANUAL = 'manual'
# maker menu buttons
GET_AUDI_MODELS = 'audi'
GET_BMW_MODELS = 'bmw'
```

```
    GET_MB_MODELS = 'mb'

    # confirmation menu bittons

    CONFIRM = 'CONFIRM'

    CANCEL = 'CANCEL'

    # main menu

    def get_keyboard():

        return
InlineKeyboardMarkup(inline_keyboard=[[InlineKeyboardButton(text='Оформить
каско', callback_data=GET_OFFER),],

                  [InlineKeyboardButton(text='Информация    о    продукте',
callback_data=GET_PRODUCT_INFO),],

                  [InlineKeyboardButton(text='Инструкция',
callback_data=GET_MANUAL),],

                  ],)

    # maker menu

    def get_maker_keyboard():

        return
InlineKeyboardMarkup(inline_keyboard=[[InlineKeyboardButton(text='Audi',
callback_data=GET_AUDI_MODELS),],

                  [InlineKeyboardButton(text='BMW',
callback_data=GET_BMW_MODELS),],

                  [InlineKeyboardButton(text='MB',
callback_data=GET_MB_MODELS),],

                  ],)

    # confirmation menu

    def get_confirmation_keyboard():

        return
InlineKeyboardMarkup(inline_keyboard=[[InlineKeyboardButton(text='Подтверди
ть', callback_data=CONFIRM),],

                  [InlineKeyboardButton(text='Отказаться',
callback_data=CANCEL),],

    ],)
```

Далее при нажатии на кнопку программа распознает ее и выдает в ответ определенный текст, картинку или действие.

Для отправки сообщений пользователю используется метод update.effective_message.reply_text, который принимает отформатированный текст сообщения. За отправку фотографий и картинок отвечает метод reply_photo, который принимает ссылку на изображение, скачивает его и отправляет пользователю.

На рисунке 12 представлен процесс предоставления согласия на обработку данных и выдача ссылки на оформление.

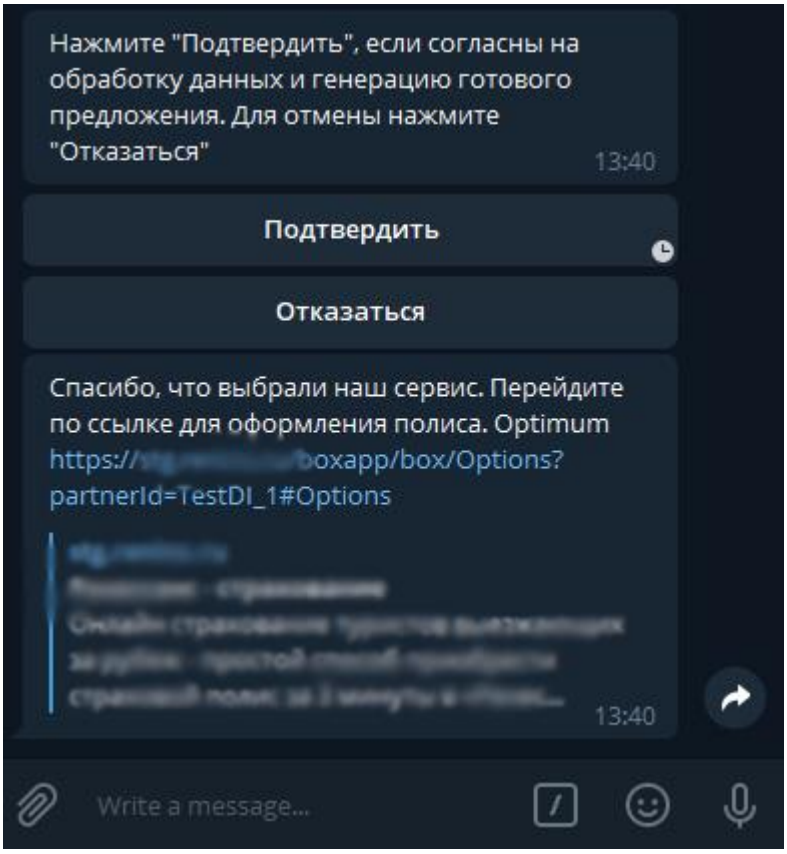

Рисунок 12 – «Итог генерации предложения»

При нажатии на кнопку «Подтвердить» программа начинает формировать ссылку. Алгоритм классификации, помещенный в модель, необходимо распаковать при старте программы, также необходимо распаковать и все кодировщики. После распаковки, модель начинает функционировать очень быстро, поэтому данный шаг необходимо выполнять при загрузке программы, а так как Python поддерживает структурное, обобщенное, объектно-

ориентированное, функциональное и аспектно-ориентированное программирование этот шаг автоматически выполняется при инициализации программы.

Для определения класса автомобиля разработан метод predict. В данном методе полученные от пользователя данные попадают в распакованные обработчики, после чего формируется датасет, который загружается в модель. Данный датасет содержит запись всего об одном автомобиле. На выходе в зависимости от предсказанного класса программа возвращает ссылку для перехода на сайт. Полный код метода predictпредставлен в листинге 24

Листинг 24 – «Метод predict»

```
    def predict(mileage: float, engine_power: int, manufacture_year: int, maker: str, model: str):
        # creating test data frame
    testDF = pd.DataFrame({"mileage":[mileage],"engine_power":[engine_power],"manufacture_year": [manufacture_year],"maker":[maker],"model": [model]})
        # devide on two data frame (categorical and decimal)
    testDecimal = testDF[['mileage','engine_power','manufacture_year']]
    testCategorical = testDF[['maker','model']].astype(str)


        # transform it!
    testCategoricalTransformed = LabelEncoder_loaded.transform(testCategorical)


        # train decimal on scaler
    testDecimalTransformed = scaler_loaded.transform(testDecimal)
        # train cat on OHE
    ohe_loaded.transform(testCategoricalTransformed)
    oheTestCategoricalTransformed = ohe_loaded.transform(testCategoricalTransformed)
```

```
            # stack cat and decimal data
        totalClassification                                                    =
np.hstack((oheTestCategoricalTransformed,testDecimalTransformed))
            # get total normalized dataframe
            X = totalClassification


            # prediction
            Y = result_loaded.predict(X)
        # choose programs and link
        if Y == 1:
                link='lite
https://xxx.xxxxxx.xx/boxapp/box/Options?partnerId=TestDI_1#Options'
        elif Y == 2:
                link='Lux
https://xxx.xxxxxx.xx/boxapp/box/Options?partnerId=TestDI_1#Options'
        elif Y == 0:
                link=f'Optimum
https://xxx.xxxxxx.xx/boxapp/box/Options?partnerId=TestDI_1#Options'
        returnlink
```

В главном меню программы можно запросить информацию о программах страхования нажав на кнопку «Информация о продукте». Бот выдаст две картинки с условиями страхования. Данный шаг представлен на рисунке 13.

Рисунок 13 – «Информация о продукте»

Последним пунктом главного меню является «Инструкция», который отправляет пользователю указания, как пользоваться чат-ботом.

Полный листинг чат-бота представлен в приложении Б.

ЗАКЛЮЧЕНИЕ

В процессе выполнения выпускной квалификационной работы поставлены цели и задачи, реализован программный комплекс, который успешно протестирован и внедрен в информационную систему организации.

Проведен анализ различных методов решения задачи определения оптимального продукта страхования автомобилей каско. Изучен материал о технологиях анализа данных.

Проведены анализ и сравнение алгоритмов классификации, выявлен наиболее эффективный. Исследование показало, что алгоритм удовлетворяет потребности организации и может заменить человека в реализации продуктов через удаленные каналы продаж.

Разработана структура приложения для оформления коробочных продуктов каско и описана работа отдельных его частей, сервисов.

Таким образом, цель работы – разработка интернет-приложения для расчета страховых продуктов каско – достигнута. В полной мере решены поставленные задачи, а именно:

Проведен анализ программ компаний конкурентов;

Проанализированы алгоритмы реализации, среди которых отобраны наиболее эффективные;

Разработан алгоритм, который позволяет классифицировать автомобиль по ценовому сегменту без учета стоимости;

Проведено сравнение с реальным агентом страхования;

Разработан новый интерфейс и сервис для интеграции с партнером;

Разработаны программы страхования;

Разработан Чат-бот для реализации страховых продуктов по средствам мессенджера Telegram.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Поллак, Г.А. Современные технологии анализа информации: учебное пособие / Г.А. Поллак – Челябинск: Издательский центр ЮУрГУ, 2013. – 115 с.
2. Вагнер Б. C# эффективное программирование. – «Лори», 2018. – 257 с.
3. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2017. – 368 с.: ил. (Серия «Библиотека программиста»)
4. Гурвиц, Джудит. «Просто о больших данных» – Москва.: Эксмо, 2015. – 400 с. – (Библиотека Сбербанка. Т.58)
5. Макмахан Брайан, РаоДелип. «Знакомство с PyTorch: глубокое обучение при обработки естественного языка. – СПб.: Питер, 2020. – 256 с.: ил. (Серия «Бестселлеры O'Reilly»)
6. Мартин Р. «Чистый код: создание, анализ и рефакторинг». – СПб.: Питер, 2018. – 464 с.: ил.
7. Мартин Р., Мартин М. «Принципы, паттерны и методики гибкой разработки на языке C#». – Пер. с англ. – СПб.: Символ-плюс, 2018. – 768с., ил.
8. Jeffrey Richter. «CLR via C#». – Microsoft Press, 2012. – 813p.
9. Lam Thuy Vo. «Mining Social Media». – No Starch Press, Inc, 2019. – 208p.
10. Марк Лутц. «Изучаем Python». – СПб.: Символ-плюс, 2011. – 1280., ил.
11. https://www.python.org/– The official home of the Python Programming Language.
12. https://metanit.com/ – Приоритетные направления – язык C# и семейство технологий .NET
13. https://msdn.microsoft.com/ru-ru – Комплексный справочник по API для работы с инструментами, службами и технологиями Microsoft.
14. https://docs.microsoft.com/ru-ru/–Хранилище документации Майкрософт для пользователей, разработчиков и ИТ-специалистов. Ознакомьтесь

с нашими краткими руководствами, учебными материалами, справочниками по API и примерами кода.

15. https://scikit-learn.org/stable/ – Scikit-learn: machine learning in Python.
16. https://xgboost.readthedocs.io/en/latest/–XGBoost Documentation.
17. https://ru.wikipedia.org/wiki– Википедия свободная энциклопедия
18. https://jupyter.org/–Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.
19. https://core.telegram.org/bots/api–Telegram APIs
20. https://python-telegram-bot.readthedocs.io/en/stable/–Python Telegram Bot’s documentation

ПРИЛОЖЕНИЕ А

Список автомобилей для сравнения агента и алгоритма.

Марка	Модель	Пробег	Год Выпуска	Мощность
bmw	x6	173427	2008	225
bmw	x3	31382	2015	140
bmw	x5	354055	2005	160
bmw	x5	29773	2014	190
bmw	x3	397642	2006	110
bmw	x5	196339	2010	180
bmw	x6	13863	2015	190
bmw	m3	15772	2014	317
bmw	m5	64374	2014	412
bmw	x5	20117	2014	280
bmw	x6	10	2015	230
bmw	x3	63167	2014	190
bmw	x3	207363	2008	130
bmw	x5	15997	2015	190
bmw	x3	107665	2011	180
bmw	x5	346008	2007	173
bmw	x5	207605	2011	180
bmw	x3	16	2015	140
bmw	x5	250719	2011	225
bmw	x5	275197	2006	200
bmw	m5	128747	2011	412
bmw	x3	273749	2004	170
bmw	x3	273749	2004	170
bmw	x6	321868	2011	225
bmw	x5	473146	2004	160
bmw	x3	350836	2005	150
bmw	x5	389460	2008	173
bmw	x5	490849	2000	210
bmw	x6	275385	2010	210
bmw	x6	26	2015	230
bmw	x3	13	2015	190
bmw	x3	13	2015	190
bmw	x5	197949	2010	225
bmw	x5	248160	2008	210
bmw	x5	7173	2015	190
bmw	x6	99779	2013	180
bmw	x5	313821	2008	210

bmw	x6	197949	2009	210
bmw	x3	336352	2004	150
bmw	m5	32026	2014	412
bmw	x6	205996	2011	225
bmw	x6	41070	2015	230
bmw	x5	279750	2003	135
bmw	x5	196586	2010	225
bmw	x5	310603	2005	170
bmw	x3	329915	2006	160
bmw	x5	352682	2005	160
bmw	x3	376586	2005	150
bmw	x3	240114	2006	110
bmw	x3	297728	2005	110
bmw	x3	127138	2006	160
bmw	x5	320259	2002	135
bmw	x6	214042	2011	225
bmw	x5	16	2015	230
bmw	x6	205996	2010	408
bmw	x5	16	2015	230
bmw	x3	318649	2004	150
bmw	x5	321868	2004	160
bmw	x3	1609	2015	140
bmw	x5	304165	2009	173
bmw	x5	38624	2015	160
bmw	x6	188293	2008	300
bmw	x3	265541	2006	110
bmw	x3	277529	2005	160
bmw	m3	91732	2012	309
bmw	x3	39204	2009	130
bmw	x3	8546	2015	140
bmw	x5	405019	2003	135
bmw	m3	190493	2005	252
bmw	m4	40555	2014	317
bmw	x5	442569	2011	225
bmw	x5	262322	2008	210
bmw	x5	427021	2005	160
bmw	x6	16	2015	280
bmw	x5	318649	2005	160
bmw	x5	80	2015	230
bmw	x6	344399	2009	173
bmw	x5	334743	2008	210
bmw	x5	176257	2011	180
bmw	x5	305775	2008	210

bmw	x6	191511	2008	300
bmw	x3	302556	2005	150
bmw	x3	386242	2007	110
bmw	x5	531082	2001	135
bmw	x5	305614	2003	170
bmw	x3	308189	2009	130
bmw	x3	299701	2007	110
bmw	x3	382379	2008	130
bmw	x5	318649	2005	160
bmw	x5	273588	2006	160
bmw	x5	378195	2008	173
bmw	x5	354055	2002	135
bmw	x5	262322	2008	210
bmw	x6	180246	2011	180
bmw	x3	210824	2008	130
bmw	x6	337678	2009	210
bmw	x5	449006	2007	173
bmw	x3	265541	2006	110
bmw	x5	334743	2002	135
bmw	x5	309835	2004	160

ПРИЛОЖЕНИЕ Б

Листинг чат-бота

```
import logging
import pymongo
from pymongo import MongoClient
from telegram import Bot
from telegram import Update
from telegram import PhotoSize
from telegram import InlineKeyboardButton
from telegram import InlineKeyboardMarkup
from telegram.ext import CallbackContext
from telegram.ext import CallbackQueryHandler
from telegram.ext import Updater
from telegram.ext import MessageHandler
from telegram.ext import Filters
from telegram.utils.request import Request
import pandas as pd
import numpy as np
from joblib import load
from sklearn.preprocessing import StandardScaler,Normalizer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
class MultiColumnLabelEncoder:
    def __init__(self,columns = None):
self.columns = columns # array of column names to encode
    def fit(self,X,y=None):
        return self # not relevant here
    def transform(self,X):
        output = X.copy()
```

```
            if self.columns is not None:
                for col in self.columns:
                    output[col] = LabelEncoder().fit_transform(output[col])
            else:
                for colname,col in output.iteritems():
                    output[colname] = LabelEncoder().fit_transform(col)
            return output
        def fit_transform(self,X,y=None):
            return self.fit(X,y).transform(X)
    # системалоггирования
    logger = logging.getLogger(__name__)
    client = MongoClient('localhost', 27017)
    # ГЛОБАЛЬНЫЕПЕРЕМЕННЫЕ
    # main menu burttons
    GET_OFFER = 'offer'
    GET_PRODUCT_INFO = 'info'
    GET_MANUAL = 'manual'
    # maker menu buttons
    GET_AUDI_MODELS = 'audi'
    GET_BMW_MODELS = 'bmw'
    GET_MB_MODELS = 'mb'
    # confirmation menu bittons
    CONFIRM = 'CONFIRM'
    CANCEL = 'CANCEL'
    # ключи
    isGetOfferClick = False
    isGetBmwClick = False
    isModelNotEmpty = False
    isMileageNotEmpty = False
```

```
        isManufactureYearNotEmpty = False
        isEnginePowerNotEmpty = False
        isFirstMessageSend = False
        # файлыдляработымодели
        result_file_name = 'result.joblib.pkl'
        scaler_file_name = 'scaler.joblib.pkl'
        ohe_file_name = 'ohe.joblib.pkl'
        LabelEncoder_file_name = 'labelEncoder.joblib.pkl'
        scaler_loaded = load(open(scaler_file_name, 'rb'))
        ohe_loaded = load(open(ohe_file_name, 'rb'))
        LabelEncoder_loaded = load(open(LabelEncoder_file_name, "rb"))
        result_loaded = load(open(result_file_name, "rb"))
        # example params
        mileage = 121999.5
        engine_power = 100
        manufacture_year = 2003
        maker = 'bmw'
        model = '330xd'
        def predict(mileage: float, engine_power: int, manufacture_year:
int, maker: str, model: str):
            # creating test data frame
        testDF = pd.DataFrame({"mileage":[mileage],"engine_power":
[engine_power],"manufacture_year": [manufacture_year],"maker":
[maker],"model": [model]})
            # devide on two data frame (categorical and decimal)
        testDecimal = testDF[['mileage','engine_power','manufacture_year']]
        testCategorical = testDF[['maker','model']].astype(str)
            # transform it!
```

```
      testCategoricalTransformed                                        =
LabelEncoder_loaded.transform(testCategorical)


          # train decimal on scaler

      testDecimalTransformed = scaler_loaded.transform(testDecimal)

          # train cat on OHE

      ohe_loaded.transform(testCategoricalTransformed)

      oheTestCategoricalTransformed                                     =
ohe_loaded.transform(testCategoricalTransformed)

          # stack cat and decimal data

      totalClassification                                               =
np.hstack((oheTestCategoricalTransformed,testDecimalTransformed))

          # get total normalized dataframe

          X = totalClassification


          # prediction

          Y = result_loaded.predict(X)

          # choose programs and link

          if Y == 1:

              link                          =                          'lite
https://xxx.xxxxxx.xx/boxapp/box/Options?partnerId=TestDI_1#Options'

      elif Y == 2:

              link                          =                           'Lux
https://xxx.xxxxxx.xx/boxapp/box/Options?partnerId=TestDI_1#Options'

      elif Y == 0:

              link                        =                       f'Optimum
https://xxx.xxxxxx.xx/boxapp/box/Options?partnerId=TestDI_1#Options'

          return link

      # main menu

      def get_keyboard():

          return
InlineKeyboardMarkup(inline_keyboard=[[InlineKeyboardButton(text='Оформитьк
аско', callback_data=GET_OFFER),],
```

```
                    [InlineKeyboardButton(text='Информацияопродукте',
callback_data=GET_PRODUCT_INFO),],

                    [InlineKeyboardButton(text='Инструкция',
callback_data=GET_MANUAL),],

                    ],)

      # maker menu

      def get_maker_keyboard():

          return
InlineKeyboardMarkup(inline_keyboard=[[InlineKeyboardButton(text='Audi',
callback_data=GET_AUDI_MODELS),],

                    [InlineKeyboardButton(text='BMW',
callback_data=GET_BMW_MODELS),],

                    [InlineKeyboardButton(text='MB',
callback_data=GET_MB_MODELS),],

                    ],)

      # confirmation menu

      def get_confirmation_keyboard():

          return
InlineKeyboardMarkup(inline_keyboard=[[InlineKeyboardButton(text='Подтверди
ть', callback_data=CONFIRM),],

                    [InlineKeyboardButton(text='Отказаться',
callback_data=CANCEL),],

                    ],)

      # Message from user handler

      def message_handler(update: Update, context: CallbackContext):

          global isMileageNotEmpty

          global isEnginePowerNotEmpty

          global isManufactureYearNotEmpty

          global isModelNotEmpty

          global isFirstMessageSend

          # get message text

      message_text = update.message.text

          if not(isFirstMessageSend):
```

```
        isFirstMessageSend = True

                return greeting_handler(update, context)

            # незаполненпробег

            if isGetOfferClick and isGetBmwClick and isModelNotEmpty and
isManufactureYearNotEmpty        and        isEnginePowerNotEmpty        and
not(isMileageNotEmpty):

                global mileage

                mileage = float(message_text)

        isMileageNotEmpty = True

        reply_text = f'Нажмите \"Подтвердить\", если согласны на обработку
данных и генерацию готового предложения. Дляотменынажмите \"Отказаться\"'

        update.message.reply_text(text=reply_text,reply_markup=get_confirmat
ion_keyboard())

            # незаполненамощность

            if isGetOfferClick and isGetBmwClick and isModelNotEmpty and
isManufactureYearNotEmpty and not(isEnginePowerNotEmpty):

                global engine_power

        engine_power = int(message_text)

        isEnginePowerNotEmpty = True

        reply_text = f'Введитепоказанияодометра (пробег) автомобиля'

        update.message.reply_text(text=reply_text)

            # незаполненгодвыпуска

            if isGetOfferClick and isGetBmwClick and isModelNotEmpty and
not(isManufactureYearNotEmpty):

                global manufacture_year

        manufacture_year = int(message_text)

        isManufactureYearNotEmpty = True

        reply_text = f'Введитемощностьдвигателяавтомобиля'

        update.message.reply_text(text=reply_text)

            # незаполненамодельавтомобиля

            if isGetOfferClick and isGetBmwClick and not(isModelNotEmpty):

                global model
```

```
            model = message_text
        isModelNotEmpty = True
        reply_text = f'Введите год выпуска автомобиля'
        update.message.reply_text(text=reply_text)


        # приветствие
        def greeting_handler(update: Update, context: CallbackContext):
            user = update.effective_user
            if user:
                name = user.first_name
            else:
        name = 'аноним'


        text = "Я телеграм-бот, который поможет тебе оформить страховку на автомобиль!\nВоспользуйся меню, чтобы узнать подробности или перейти к оформлению!"
        reply_text = f"Привет, {name}!\n\n{text}"


            # Ответитьпользователю
            return                  update.message.reply_text(text=reply_text, reply_markup=get_keyboard(),)


        # Записать сообщение в БД
        iftext:
        add_message(user_id=user.id,
                    text=text,)



        # menu's handler
        def callback_handler(update: Update, context: CallbackContext):
            user = update.effective_user
```

```
    callback_data = update.callback_query.data


        global isGetOfferClick

        global isGetBmwClick


    photoLight=''

    photoOptimum=''

        keyboard= None

        text =''

        if callback_data == GET_OFFER:

    text = f'Для начала выберите марку автомобиля:'

    keyboard=get_maker_keyboard()

    isGetOfferClick = True

    elifcallback_data == GET_PRODUCT_INFO:

    text = f'Условия продукта страхования. Для вызова главного меню напишите любое сообщение.'

    photoLight = 'https://github.com/JRSY23/PY/blob/master/%D0%9B%D0%B0%D0%B9%D1%82.png?raw=true'

    photoOptimum = 'https://github.com/JRSY23/PY/blob/master/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D1%83%D0%BC.png?raw=true'

    restart_process()

    elifcallback_data == GET_MANUAL:

    text = f'Инструкция \"Как пользоваться нашим ботом\" (Сделать ПДФ). Для вызова главного меню напишите любое сообщение.'

    restart_process()

    elifcallback_data == GET_BMW_MODELS:

    text = f'Введите модель автомобиля на английском (Например X5, 320i, m5, z4).'

    global maker

            maker = 'bmw' # Марка - BMW

    isGetBmwClick = True
```

```
        elifcallback_data == CONFIRM:
                link = predict(mileage, engine_power, manufacture_year,
maker, model)
        text = f'Спасибо, что выбрали наш сервис. Перейдите по ссылке для
оформления полиса. {link}'
        restart_process()
        elifcallback_data == CANCEL:
                text = f'Оченьжаль:('
        restart_process()
            else:
                text = 'Произошлаошибка'
        restart_process()


        update.effective_message.reply_text(text=text,reply_markup=keyboard)
            if (photoLight!='' and photoOptimum !=''):
        update.effective_message.reply_photo(photo=photoLight,)
        update.effective_message.reply_photo(photo=photoOptimum,)


        # restart process and clean data
        def restart_process():
            global isFirstMessageSend
            global isGetOfferClick
            global isGetBmwClick
            global isModelNotEmpty
            global isMileageNotEmpty
            global isManufactureYearNotEmpty
            global isEnginePowerNotEmpty
        isFirstMessageSend = False
        isGetOfferClick = False
        isGetBmwClick = False
```

```
        isModelNotEmpty = False
        isMileageNotEmpty = False
        isManufactureYearNotEmpty = False
        isEnginePowerNotEmpty = False



        def main():
        logging.basicConfig(filename="kaskoBot.log", level=logging.INFO)
        print('Start KaskoBot')
        logger.log(level=logging.INFO, msg='Start KaskoBot')

            # объявлениеботаитд
            req = Request(connect_timeout=0.5,
        read_timeout=1.0,)
            bot                                                                    =
Bot(token='1201845137:AAHh2QcXRHAKcJ1HK7LVUroMlaRLSSYQj9s',
                request=req,
        base_url='https://telegg.ru/orig/bot',)
            updater = Updater(bot=bot,
        use_context=True,)

        # Проверить что бот корректно подключился к Telegram API
        info = bot.get_me()
        print(f'Bot info: {info}')
        logger.log(level=logging.INFO, msg=f'Bot info: {info}')

            # Навеситьобработчикикоманд
        updater.dispatcher.add_handler(MessageHandler(Filters.all,
message_handler))

updater.dispatcher.add_handler(CallbackQueryHandler(callback_handler))
```

```
# Начать бесконечную обработку входящих сообщений
updater.start_polling()
updater.idle()
print('Stop KaskoBot')
logger.log(level=logging.INFO, msg='Stop KaskoBot')


if __name__ == '__main__':
main()
```