1. Выбор и обоснование комплекса программных средств

В настоящее время существует огромное количество программных продуктов, позволяющих в эффективно и качественно разработать программный комплекс для различных предметных областей. Для правильного и обоснованного выбора во внимание принимались различные критерии для оценки качества программного продукта.

* 1. Выбор языка программирования

Для реализации клиентской части системы выбран язык программирования JavaScript с использованием фреймворка Vue.js.

JavaScript – это интерпретируемый язык программирования, который широко используется для создания динамических веб-сайтов и веб-приложений. Он обеспечивает взаимодействие пользователя с содержимым страницы, обновление данных без перезагрузки страницы и дополнительные функциональные возможности.

Vue.js — JavaScript фреймворк для создания пользовательских интерфейсов. Он создан на стандартах HTML, CSS и JavaScript и предоставляет декларативную и компонентную модель программирования, которая помогает эффективно разрабатывать пользовательские интерфейсы любой сложности.

JavaScript был выбран для реализации фронтенда проекта из-за его широкого использования в веб-разработке и гибкости. Vue.js, был выбран из-за своей простоты, производительности и поддержки различных функциональных возможностей для веб-приложений.

Для реализации серверной части системы был выбран язык программирования Python и фреймворк Flask. Python — мультипарадигмальный высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпретируемый и используется в том числе для написания скриптов.

Python имеет большое количество библиотек и фреймворков для анализа данных и машинного обучения. Для реализации модели, прогнозирующей стоимость автомобиля, использовались NumPy, Pandas, Matplotlib, Seaborn, Scikit-learn и Catboost. Также для создания парсера использовалась библиотека Selenium.

* 1. Выбор среды программирования

В качестве среды программирования была выбрана Visual Studio Code. Visual Studio Code (VS Code) — это редактор исходного кода. Его разработал Microsoft для всех популярных операционных систем: Windows, Linux и macOS. Визуальный редактор кода позволяет:

* работать с IntelliSense — автоматическим дописыванием функций при вводе первых букв;
* выполнять отладку — искать и устранять ошибки в написанном коде;
* удобно писать код — автоматически заполнять нужную информацию, подсвечивать элементы синтаксиса в зависимости от выбранного языка, расставлять нужные отступы;
* контролировать версии кода, в том числе с помощью системы управления версиями Git;
* рефакторить код для улучшения его работы и читабельности.
  1. Выбор операционной системы

В качестве операционной системы (ОС) для клиентской части выбрана Windows 10 – операционная система, разработанная компанией Microsoft в 2015 году. Имеет следующие преимущества:

* обладает удобным интерфейсом для облегчения установки и поддержки любого программного обеспечения;
* справляется с перепадами напряжения в сети и обеспечивает отказоустойчивость;
* практически любое программное обеспечение выпускается помимо других ОС для ОС Windows 10.

1. Описание метода прогнозирования

Существует большое количество методов машинного обучения, решающих задачу регрессии. Были взяты несколько различных моделей, обучены на подготовленных данных и оценены по метрике MAPE (Mean Absolute Percentage Error).

* 1. Метрика качества

MAPE выражает среднее абсолютное отклонение прогнозируемых значений от фактических значений в процентах, что делает эту метрику очень наглядной для интерпретации результатов.

Формула MAPE определяется как:

где:

*N* – количество наблюдений,

*yi* – фактическое значение целевой переменной,

*f(xi)* – прогнозируемое значение, полученное моделью,

*|yi−f(xi)|* – абсолютная ошибка прогноза для (i)-го наблюдения,

*|yi|* – абсолютное значение фактического значения целевой переменной для нормализации ошибки.

В результате оценки моделей на тестовых данных получились следующие значения:

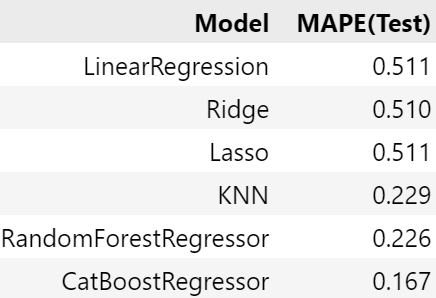


Рисунок – результаты оценки моделей

Наилучший результат показала модель CatBoostRegressor, которая представляет собой реализацию градиентного бустинга над решающими деревьями библиотеки CatBoost. Далее рассмотрим принцип работы градиентного бустинга.

* 1. Градиентный бустинг

Бустинг – это ансамблевый метод машинного обучения, целью которого является объединение нескольких слабых моделей предсказания для создания одной сильной. Слабая модель – это такая, которая выполняет предсказания немного лучше, чем наугад, в то время как сильная модель обладает высокой предсказательной способностью. Цель бустинга – улучшить точность предсказаний.

Бустинг работает путём последовательного добавления моделей в ансамбль. Каждая следующая модель строится таким образом, чтобы исправлять ошибки, сделанные предыдущими моделями. Это достигается путём фокусировки на наиболее проблемных данных, которые были неверно классифицированы или предсказаны ранее.

Одной из основных особенностей бустинга является динамическое взвешивание обучающих данных. После каждого этапа обучения модели в ансамбле, данные, на которых были допущены ошибки, получают больший вес. Это означает, что последующие модели уделяют больше внимания именно этим трудным случаям.

Когда используются решающие деревья, каждое последующее дерево строится с учетом ошибок, сделанных предыдущими деревьями. Новые деревья учатся на ошибках, улучшая общую точность ансамбля.

Несмотря на свою эффективность, бустинг может быть склонен к переобучению, особенно если в ансамбле слишком много моделей или они слишком сложные. Для контроля переобучения используется ранняя остановка (early stopping).

* 1. Принцип построения ансамбля

Ансамбль в градиентном бустинге обычно состоит из последовательности слабых предсказательных моделей. Чаще всего используются решающие деревья из-за их способности моделировать нелинейные зависимости и взаимодействия между признаками. Каждое новое дерево в ансамбле строится так, чтобы уменьшить оставшуюся ошибку предыдущих деревьев.

В градиентном бустинге каждая следующая модель обучается с учетом ошибок, допущенных всеми предыдущими моделями в ансамбле. Это достигается путем фокусировки на самых трудных для предсказания случаях, которые были неправильно классифицированы или предсказаны ранее.

Суть метода заключается в том, что веса для каждого наблюдения в обучающем наборе данных корректируются на каждом шаге. Наблюдения, которые были неправильно предсказаны предыдущей моделью, получают больший вес, тем самым увеличивая вероятность их правильного предсказания последующими моделями.

* 1. Градиентный спуск

Градиентный спуск — это итеративный алгоритм оптимизации, используемый для минимизации функции, чаще всего функции потерь в контексте машинного обучения. Он работает путем нахождения направления, в котором функция потерь уменьшается наиболее быстро, и делает шаги в этом направлении для постепенного уменьшения значения функции потерь.

Градиент функции — это вектор, состоящий из частных производных, который указывает направление наискорейшего роста функции. В контексте оптимизации, мы интересуемся направлением наискорейшего убывания, то есть движемся в противоположном направлении градиента. Частные производные вычисляются для каждого параметра модели.

На каждой итерации алгоритма параметры модели обновляются в направлении, противоположном градиенту функции потерь. Размер шага, который делает алгоритм в этом направлении, определяется скоростью обучения (learning rate). Оптимальная скорость обучения — ключевой параметр, поскольку слишком большой шаг может привести к пропуску минимума, а слишком маленький делает процесс оптимизации медленным.

Формула для обновления параметра θ на каждой итерации выглядит следующим образом:

где *η* - скорость обучения, а *∇θ⋅J(θ)* - градиент функции потерь *J* по параметру *θ*.

* 1. Реализация градиентного бустинга в CatBoost

CatBoost использует решающие деревья глубины 1 или 2 в качестве базовых моделей. Эти неглубокие деревья имеют следующие характеристики:

* каждый узел дерева делает бинарное разбиение на основе значения одной из признаков;
* эти короткие деревья обладают небольшой глубиной, что делает их более устойчивыми к переобучению.

CatBoost включает механизм регуляризации, чтобы предотвратить переобучение модели. Он использует L2-регуляризацию, представляет собой метод добавления штрафа к функции потерь модели с целью предотвратить переобучение. Этот метод основан на добавлении суммы квадратов весов признаков к функции потерь. Формула выглядит следующим образом:

где *λ* (лямбда) — это гиперпараметр, который контролирует силу регуляризации. Он выбирается заранее и может быть настроен в процессе обучения модели, *n* - количество признаков в модели, *wi*​ - вес (коэффициент) i-го признака.

Таким образом, L2-регуляризация добавляет к функции потерь сумму квадратов весов всех признаков, умноженную на гиперпараметр λ. Это вынуждает модель уменьшать веса признаков и предотвращает их чрезмерное увеличение в процессе обучения.

Когда λ равно нулю, L2-регуляризация отсутствует, и модель обучается без ограничений на веса признаков. Когда λ большое, L2-регуляризация становится более сильной, и веса признаков близки к нулю. L2-регуляризация штрафует большие значения весов, поощряя модель использовать только наиболее информативные признаки.

Этот метод помогает улучшить обобщающую способность модели и снизить риск переобучения, особенно в случаях, когда у нас много признаков или они коррелированы между собой.

CatBoost автоматически выполняет отбор признаков путем оценки их важности для модели. Это позволяет модели сосредотачиваться на наиболее информативных признаках и уменьшить шум от менее значимых.

CatBoost применяет градиентный бустинг для обучения ансамбля решающих деревьев. Градиентный бустинг минимизирует функцию потерь с использованием градиентного спуска, постепенно улучшая качество модели.

На каждой итерации градиентного бустинга добавляется новое решающее дерево, которое исправляет ошибки предыдущих деревьев.

1. Разработка и описание интерфейса пользователя

Пользовательский интерфейс – одна из разновидностей интерфейсов, который является совокупностью средств и методов взаимодействия пользователя с вычислительными устройствами (персональным компьютером).

Интерфейс характеризуется удобством, эффективностью, понятностью и дружественностью.

Дружественный интерфейс предоставляет пользователю наиболее удобный способ взаимодействия с программным обеспечением путем обеспечения логичности и простоты в расположении элементов управления.

Разработанная система представляет собой web-приложение, которое может работать на компьютере под управлением любой операционной системы при наличии браузера.

При запуске приложения открывается страница с формой для ввода характеристик автомобиля.

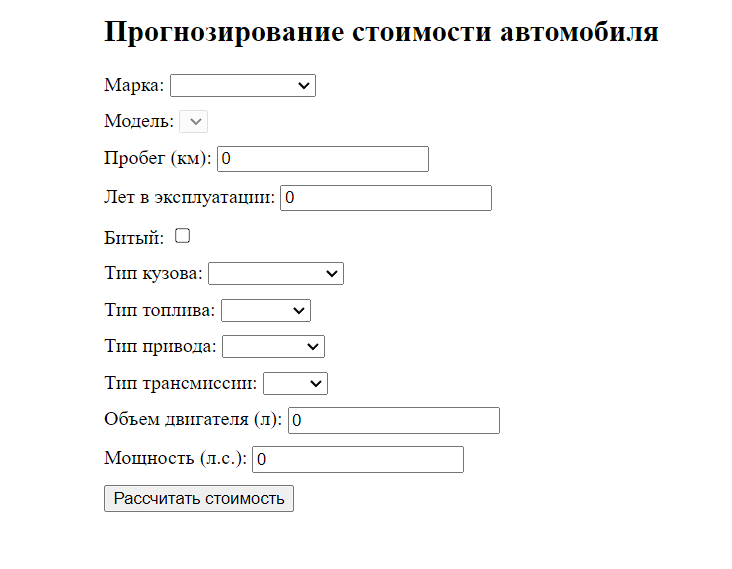


Рисунок – форма ввода характеристик

Здесь пользователь сможет ввести параметры интересующего автомобиля, после чего нажать кнопку «Рассчитать стоимость» и получить прогноз.

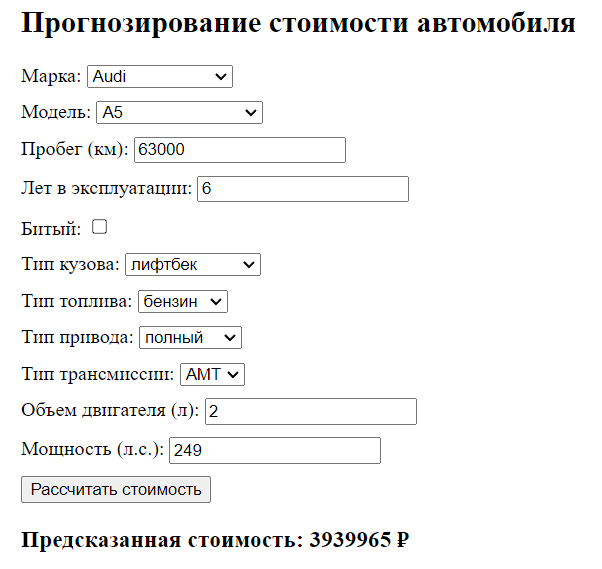


Рисунок – результат прогноза