САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Прикладная математика и информатика
Исследование операций и принятие решений в задачах оптимизации, управления и экономики

Енгалыч Глеб Артурович

Методы решения задачи регрессии

Выпускная квалификационная работа

Научный руководитель:
д. ф.-м. н., профессор Малоземов В. Н.

Рецензент:
Главный эксперт аналитик «Газпромнефть – Альтернативное топливо», к. ф.-м. н., доцент Аббакумов В. Л.

Санкт-Петербург

2019

SAINT-PETERSBURG
STATE UNIVERSITY

Applied Mathematics and Computer Science
Operation Research and Decision Making in Optimisation, Control and Economics Problems

Yengalych Gleb Arturovich

Comparison of various methods for solving regression problems

Graduation Project

Scientific supervisor:
Professor Malozemov V. N.

Reviewer:
Chief expert analyst «Gazpromneft – Alternativnoyee Toplivo»,
PhD Abbakumov V. L.

Saint-Petersburg

2019

Содержание

Введение

Машинное обучение — одна из самых бурно развивающихся отраслей прикладной математики в последнее десятилетие. Все крупные компании имеют свои собственные отделы по анализу данных, что позволяет оптимизировать производство. В связи с этим появился огромный спрос на квалифицированных специалистов в области машинного обучения. Одним из способов обмена опытом и получения новых навыков в этой отрасли являются соревнования по машинному обучению.

Данная исследование основано на работе с базой данных о сыгранных партиях в популярную многопользовательскую видеоигру PlayerUnknown's Battlegrounds. На сайте `kaggle.com` с октября 2018 г. по январь 2019 г. проводилось соревнование по машинному обучению по предсказанию места, занятого игроком в рамках одной партии в PlayerUnknown's Battlegrounds.

Эта база данных обладает своими особенностями, поэтому она интересна для анализа и разнообразных экспериментов.

Работа состоит из 7 глав и введения. В 1 главе определяются цели работы, и приводится постановка задачи. Глава 2 содержит описание необходимых преобразований данных. В 3 главе кратко описываются используемые алгоритмы машинного обучения. Глава 4 содержит описание процесса обучения моделей. В 5 главе сравниваются результаты полученных моделей. Глава 6 содержит краткий анализ одной из построенных моделей. В 7 главе описаны итоги работы.

1 Постановка задачи

Имеется база данных, в которой содержится информация о сыгранных партиях в PlayerUnknown's Battlegrounds. Каждая запись — информация о результативности одного игрока в одной конкретной партии, представленная 28 характеристиками. Данные разделены на две части: обучающее и тестовое множество. В обучающем множестве также есть ещё одна характеристика — мера близости игрока к первому месту. Это вещественнозначная переменная, значения которой находятся в отрезке $[0;\ 1]$. Чем больше значение этой переменной, тем более высокое место игрок занял в конкретном матче. В частности, она равна единице для игрока, занявшего первое место, и равна нулю для игрока, занявшего последнее место.

Требуется разработать модель оценивания меры близости игрока к первому месту для игроков из тестового множества. Для оценивания модели используется функция потерь mean absolute error.

Скачать базу данных и посмотреть страницу соревнования можно по ссылке [7].

1.1 Цели работы

- Провести предварительный анализ и предобработку данных.
- Примененить различные алгоритмы машинного обучения к полученной после предобработки базе данных.
- Сравнить результаты работы использованных алгоритмов друг с другом.
- Провести анализ влияния переменных на работу лучшей модели.
- Сравнить полученные результаты с результатами других участников соревнования.

1.2 Правила игры PlayerUnknown's Battlegrounds

PlayerUnknown's Battlegrounds (сокр. PUBG) — многопользовательская онлайн-игра в жанре королевской битвы.

В одной партии может участвовать до ста игроков. Игровая карта представляет собой остров, на котором расположены здания и другие виды различные укрытий. Игра идёт до последнего выжившего игрока. Партия начинается с того, что игроки десантируются с самолёта на игровую карту и начинают искать снаряжение, например: оружие, медикаменты, бронежилеты, автомобили.

Через некоторое время на игровой карте выделяется зона безопасности — круг, в который должны проследовать все игроки, те, кто находится вне круга, получают урон с течением времени. Через некоторое время внутри предыдущего круга выбирается новой круг, который станет новой зоной безопасности, процесс повторяется, пока зона безопасности не станет достаточно маленькой. Таким образом, вероятность столкновений игроков друг с другом растёт с течением времени.

Рис. 1: Пример игровой карты с выделенными на ней зонами безопасности

Первое место в партии присуждается последнему оставшемуся в живых игроку, второе место — игроку, умершему последним, третье — игроку, умершему предпоследним, и так далее.

Существует несколько режимов игры. Самые важные, которые стоит отметить, это игра в одиночку, в паре или в скваде (команде из четырёх человек). Игра в одиночку уже описана выше.

Игра в команде имеет несколько серьёзных отличий. Если игрок получил смертельный урон, то он не умирает, а падает на землю. В этом состоянии он передвигается лишь ползком, не может выполнять никаких действий и получает повышенный урон. Если в течение определённого времени никто из сокомандников не приведёт такого игрока в сознание, то он погибает. Также игроки в одной команде получают доступ к голосовому чату, в котором они могут общаться и обсуждать план действий на игру. Место команды в окончательном рейтинге определяется моментом смерти последнего игрока в команде. То есть, теоретически, команда, в которой выжил всего лишь один игрок, может занять первое место в партии.

Рис. 2: Интерфейс игрока

1.3 Описание базы данных

База данных состоит из обучающего и тестового множества. В обучающем множестве содержится информация о примерно 4.45 млн. игроков, в тестовом множестве содержится информация о примерно 1.93 млн. игроков.

Для каждого игрока X приведены следующие характеристики:

- DBNOs — количество вырубленных игроков (всегда равна нулю для одиночной игры).
- assists — количество содействий, то есть количество игроков, которым игрок X нанёс урон, но убитых сокомандниками X (всегда равна нулю для одиночной игры).
- boosts — количество энергетиков, принятых игроком X.
- damageDealt — количество урона, которое нанёс X другим игрокам.
- headshotKills — количество убийств, совершённых выстрелом в голову.
- heals — количество аптечек или бинтов, использованных игроком.
- Id — номер игрока в базе данных.
- killPlace — порядковая переменная, показывающая ранг игрока по количеству убийств внутри текущего матча.
- killPoints — глобальный рейтинг игрока, учёт которого ведётся по только количеству убийств.
- killStreaks — максимальное количество противников, убитых игроком X в течение короткого промежутка времени.
- kills — количество убитых противников.
- longestKill — наибольшая дистанция между игроком X и убитым им противником.
- matchDuration — длительность матча в секундах.

- matchId — уникальный идентификатор матча.
- match-type — строка, показывающая тип матча.
- rankPoints — глобальный рейтинг игрока. Создатели базы данных утверждают, что использование этой переменной может быть ненадёжным, так как это не официальный рейтинг.
- revives — количество сокомандников, которых привёл в сознание X.
- rideDistance — суммарное расстояние в метрах, которое игрок преодолел на автомобиле или другом средстве передвижения.
- roadKills — количество убийств, совершённых игроком X, пока он передвигался на автомобиле или другом средстве передвижения.
- swimDistance — суммарное расстояние в метрах, которое проплыл игрок.
- teamKills — количество сокомандников, убитых игроком X.
- vehicleDestroys — количество автомобилей или других средств передвижения, уничтоженных игроком.
- walkDistance — суммарное расстояние в метрах, пройденное игроком пешком.
- weaponsAcquired — количество единиц оружия, поднятых игроком.
- winPoints — глобальный рейтинг игрока, основанный только на его победах в партиях.
- groupId — уникальный идентификатор команды.
- numGroups — количество команд, участвующее в текущем матче.
- maxPlace — ранг наименьшего места, занятого кем-либо в этом матче (не всегда совпадает с numGroups из-за некоторых особенностей расстановки мест)

Переменная winPlacePerc представлена только в обучающем множестве. Это переменная, которую требуется предсказать для игроков в тестовом множестве. winPlacePerc измеряется в промежутке [0; 1], чем она больше, тем более высокое место занял игрок. Значение, равное единице, соответствует первому месту, значение, равное нулю, соответствует последнему месту. Рассчитывается на основе переменной maxPlace, а не numGroups.

Стоит чуть больше внимания уделить описанию переменных matchId и groupId:

- Утверждается, что если в обучающее или тестовое множество попал игрок, то тогда и все игроки, игравшие в этом же матче попали в это же множество. То есть, используя matchId, можно получить исчепывающую информацию о характеристках всех игроков внутри одного матча. Также это гарантирует то, что модель во время обучения будет получать информацию только из обучающего множества.

- Характеристика groupId позволяет получить исчерпывающую информацию о характеристиках игроков, играющих в команде, внутри одного матча. Утверждается, что никакие две команды в никаких двух матчах не могут иметь одинаковый groupId, что позволяет очень просто обрабатывать игроков в одной команде, делая групповые запросы в базу данных по значению groupId.

Важно отметить, что все игроки обезличены. То есть, у нас нет возможности получить статистику обо всех матчах, в которых сыграл игрок.

2 Предобработка данных

2.1 Exploratory data analysis

Перед тем как работать с данными важно тщательно изучить их.

Обучающее множество состоит из 4446966 записей, каждая снабжена 29 переменными.

Из описания таргетной переменной логично предположить, что её распределение близко к равномерному, проверим это, построив гистограмму:

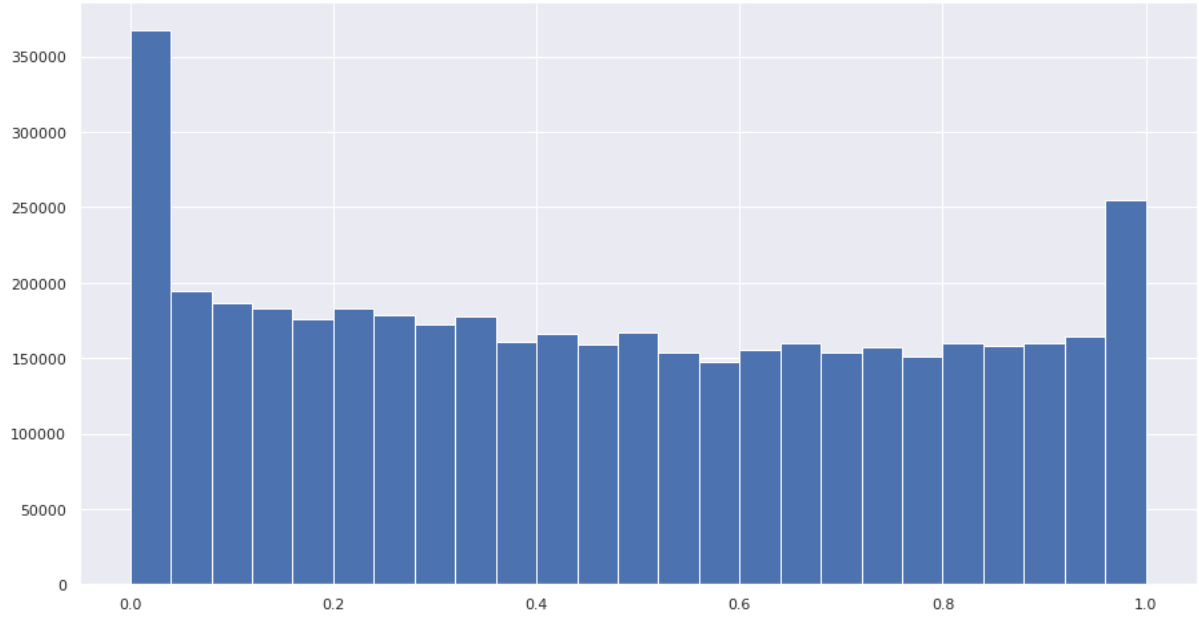

Рис. 3: Гистограмма распределения переменной winPlacePerc

Значения 0 и 1 встречаются гораздо чаще других, что объясняется тем, что они есть в каждом матче, но это никак пагубно не повлияет на работу моделей. Распределение можно всё равно считать близким к равномерному.

Имеет смысл посмотреть на распределение других переменных и на распределение таргетной переменной, сгруппированной по значениям одного из предикторов.

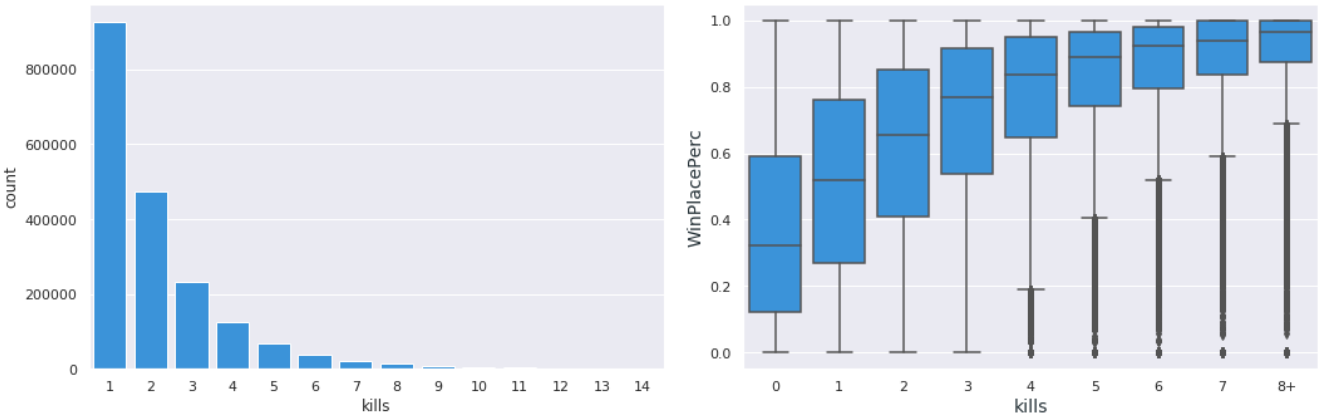

Рис. 4: Гистограмма распределения переменной kills и boxplot'ы таргетной переменной, сгруппированной по kills

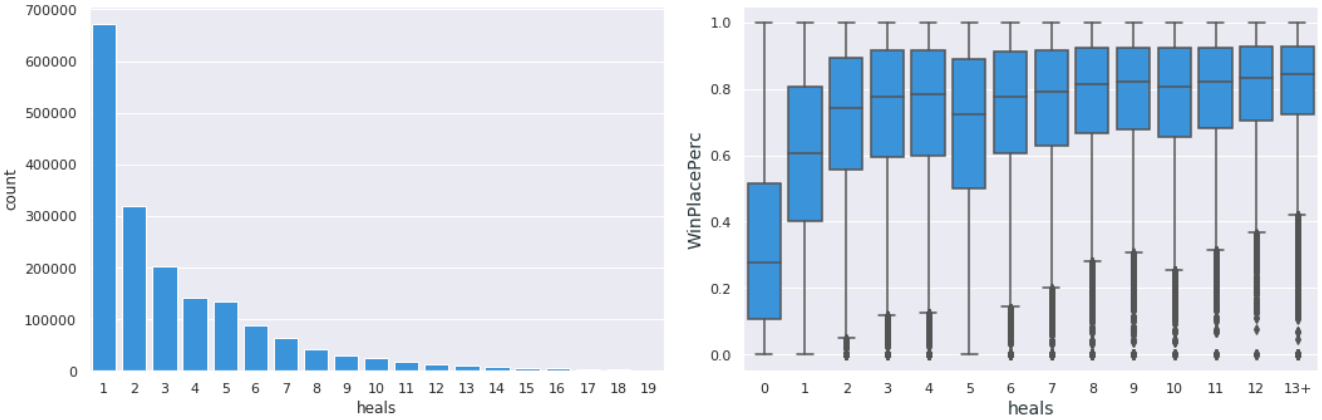

Рис. 5: Гистограмма распределения переменной heals и boxplot'ы таргетной переменной, сгруппированной по heals

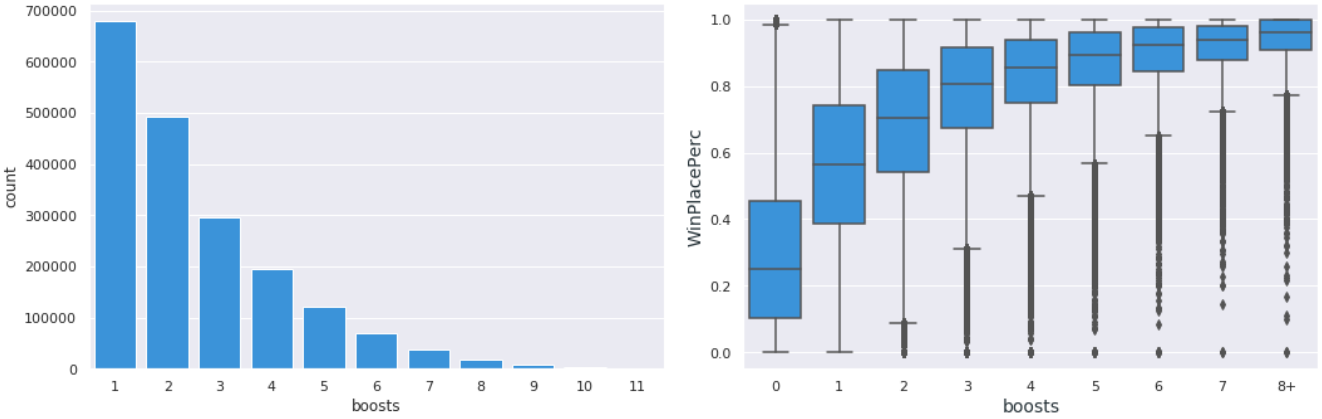

Рис. 6: Гистограмма распределения переменной boosts и boxplot'ы таргетной переменной, сгруппированной по boosts

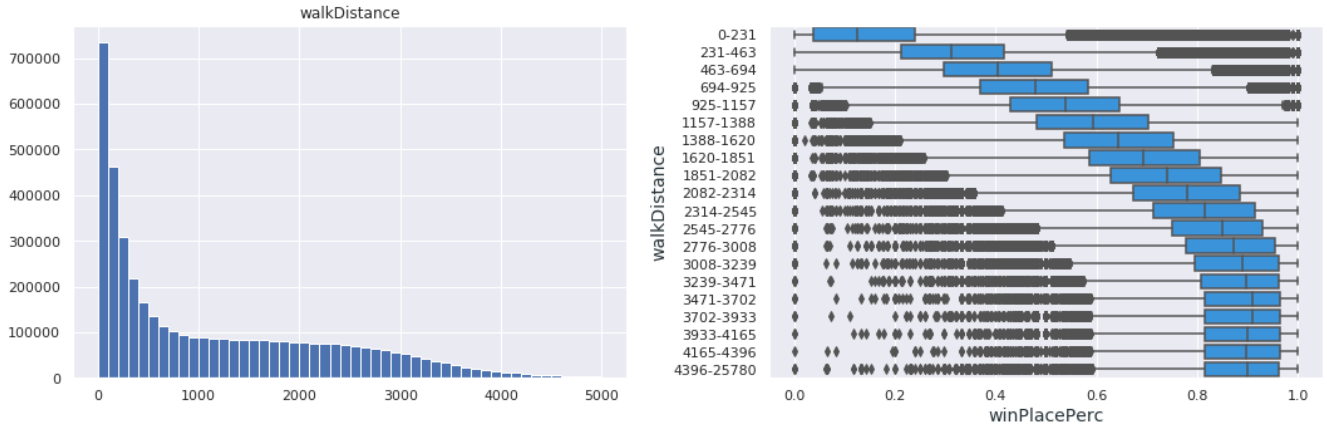

Рис. 7: Гистограмма распределения переменной walkDistance и boxplot'ы таргетной переменной, сгруппированной по walkDistance

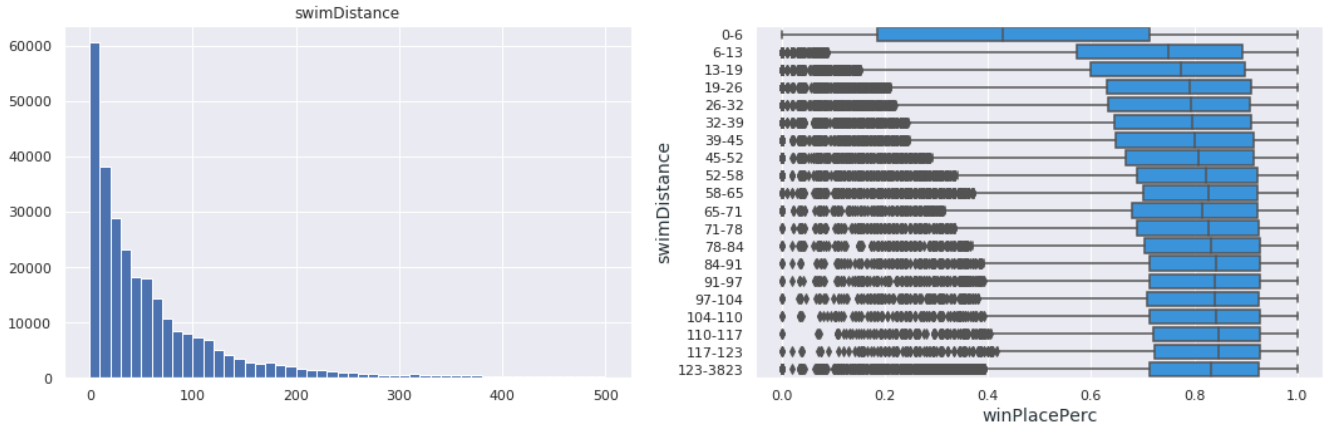

Рис. 8: Гистограмма распределения переменной swimDistance и boxplot'ы таргетной переменной, сгруппированной по swimDistance

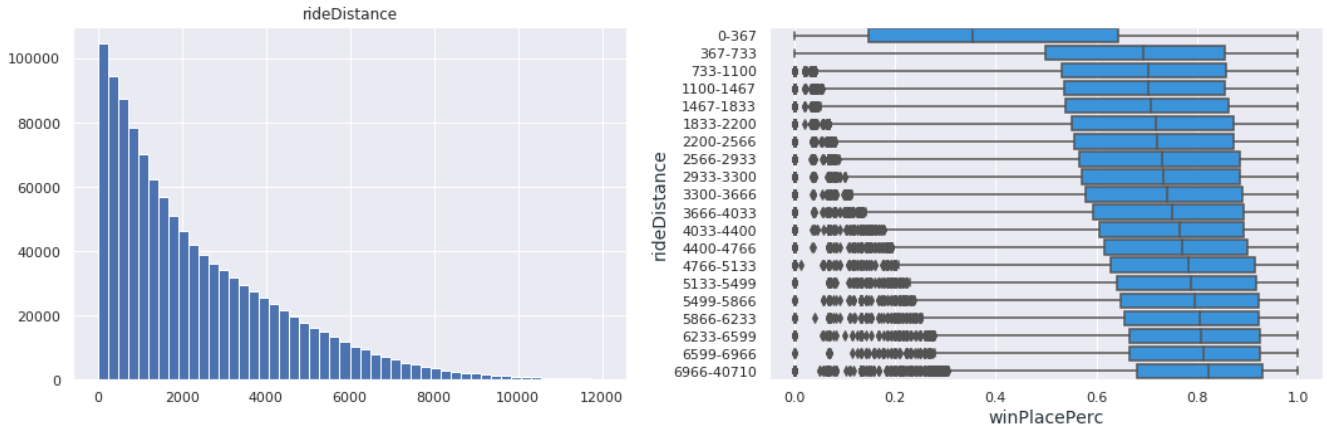

Рис. 9: Гистограмма распределения переменной rideDistance и boxplot'ы таргетной переменной, сгруппированной по rideDistance

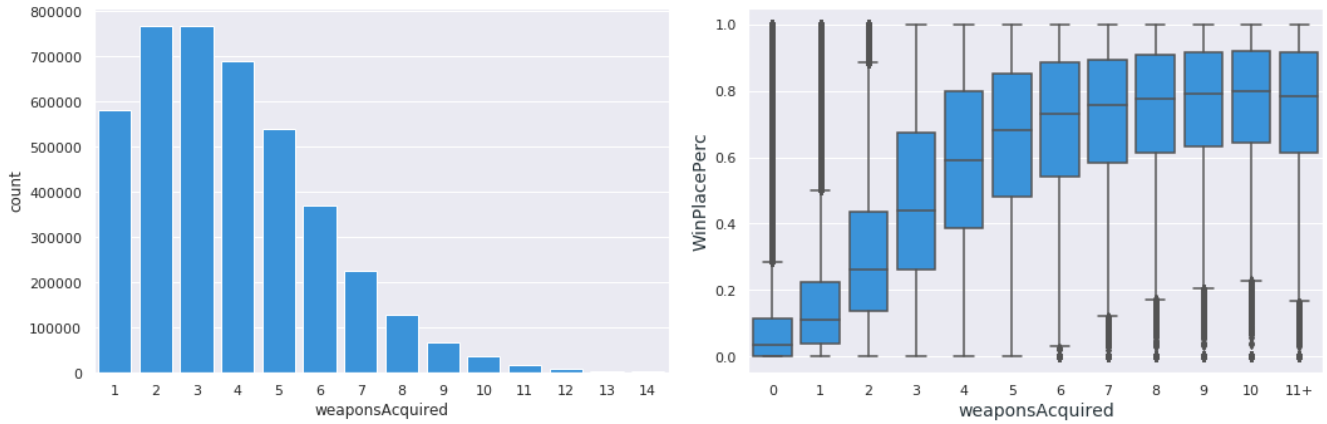

Рис. 10: Гистограмма распределения переменной weaponsAcquired и boxplot'ы таргетной переменной, сгруппированной по weaponsAcquired

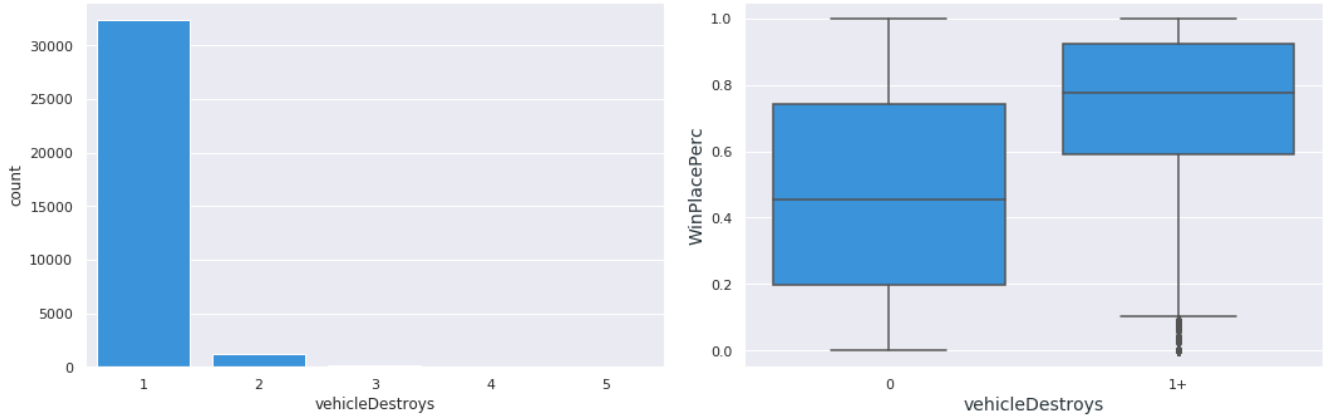

Рис. 11: Гистограмма распределения переменной vehiclesDestroyed и boxplot'ы таргетной переменной, сгруппированной по vehiclesDestroyed

Вышеприведённые графики говорят о том, что опытные игроки совершают больше убийств, чаще используют аптечки, больше передвигаются и активно меняют оружие, что помогает им занимать более высокие места.

На графике ниже представлена информация о линейной корреляции между предикторами и таргетом, а также между парами предиктор-предиктор. Указанные ниже переменные сильнее всего коррелируют с winPlacePerc.

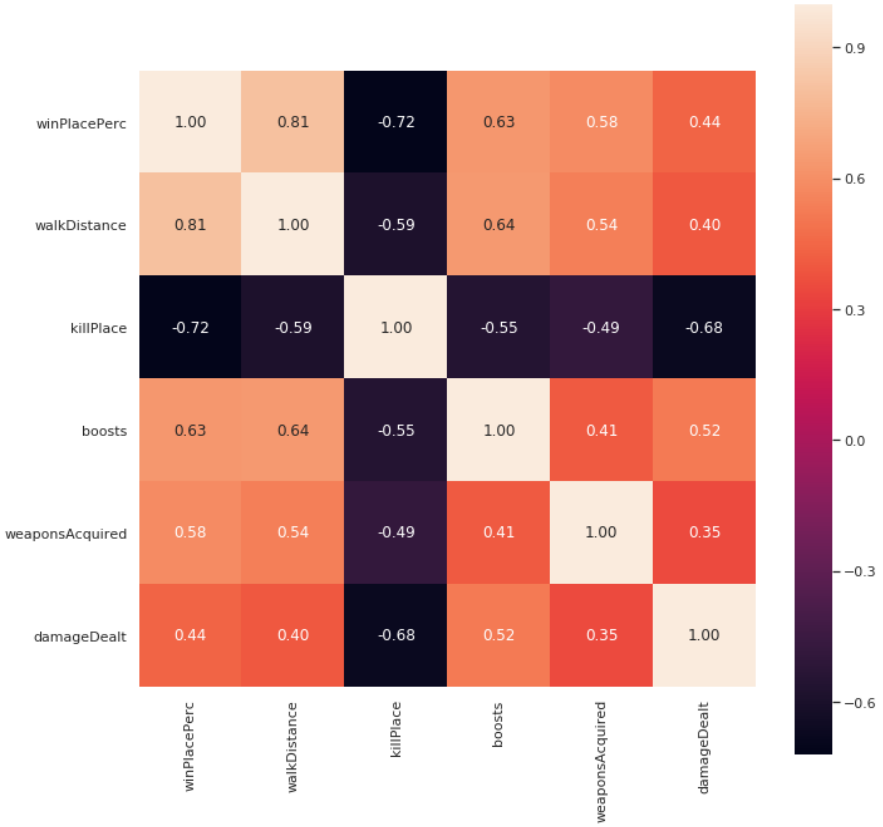

Рис. 12: Матрица значений коэффициента линейной корреляции между переменными

Отмеченные в этом разделе переменные в первую очередь стоит использовать для создания новых переменных.

2.2 Feature engineering

В ходе работы на основе исходных 28 переменных было введено приблизительно 30 новых переменных, из соображений ограничений по памяти и пользы для предсказания были оставлены следующие:

- $headshotrate = \frac{headshotKills}{kills}$
- $killStreakrate = \frac{killStreaks}{kills}$
- $healthitems = heals + boosts$
- $totalDistance = rideDistance + walkDistance + swimDistance$
- $killPlace\_over\_maxPlace = \frac{killPlace}{maxPlace}$

- $distance\_over\_weapons = \frac{totalDistance}{weaponsAcquired}$
- $walkDistance\_over\_heals = \frac{walkDistance}{heals}$
- $walkDistance\_over\_kills = \frac{walkDistance}{kills}$
- $killsPerWalkDistance = \frac{kills}{walkDistance}$
- $skill = headshotKills + roadKills$

Если значения переменной не существует (знаменатель равен нулю), то оно принимается равным нулю.

В итоге в датасете теперь 34 количественные переменные. На основе каждой из них построим переменные следующего вида:

- Сгруппируем наблюдения по значениям matchId и groupId и возьмём минимум из значения переменной по всей группе. Это значение минимума и есть значение новой переменной. Таким образом, мы рассматриваем каждую команду, как единого игрока. Аналогично, создадим новые переменные по максимуму и среднему арифметическому.
- Отранжируем вышеописанные значения минимума внутри матча по всем командам. Значение ранга и будет значением новой переменной. Аналогично, создадим новые переменные по максимуму и среднему арифметическому.
- Для каждого наблюдения вычисляется среднее арифметическое и максимальное значение переменной по всем игрокам внутри матча.

Также к каждому наблюдению была добавлена переменная, показывающая количество игроков внутри матча.

На выходе получается датасет, состоящий из 281 переменной.

Для нейронных сетей затем значения всех переменных были отмасштабированы в отрезок $[-1; 1]$.

2.3 Сжатие памяти

По умолчанию pandas.DataFrame может использовать избыточные типы данных для хранения данных. Поэтому после предобработки данных была применена техника сжатия памяти: для каждой переменной оценивался максимум модуля значения наблюдения, и на его основе выбирался оптимальный тип данных для хранения каждой переменной. Эта техника позволила сократить объём затраченной на хранение датасета памяти на 46%.

3 Алгоритмы машинного обучения

Машинное обучение — раздел науки об алгоритмах и статистического моделирования, изучающий системы, эффективно решающие задачи, основываясь на закономерностях и паттернах в исходных данных.

Машинное обучение с учителем используется для построения модели оценивания значения неизвестной функции на основе набора пар: вектора предикторов x и отклика y.

В этой главе приводится формулировка исходной задачи в терминах машинного обучения и краткое описание алгоритмов, которые используются для моделирования.

3.1 Задача регрессии

Имеется матрица исходных данных X, состоящая из n строк-наблюдений x_i, каждое из которых представляет собой вектор размера m, где m — количество предикторов.

$$X = (x_i)_{i=1}^{n},\ x_i \in \mathbb{R}^m,$$

и вектор результатов

$$y = (y_i)_{i=1}^{n},\ y_i \in \mathbb{R}.$$

Каждому наблюдению x_i соответствует значение отклика y_i. Задачу предсказания вещественнозначной переменной по размеченным данным, то есть по набору пар предиктор-отлик, называют задачей регрессии.

Если взять в качестве матрицы X взять базу данных, полученную после предобработки данных, а в качестве отклика y — вектор со значением переменной winPlacePerc, то исходная задача попадает под определение задачи регрессии.

Для оценивания качества модели используется функция потерь mean absolute error:

$$MAE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|,$$

где $y \in \mathbb{R}^n$ — реальное значение переменной, $\hat{y} \in \mathbb{R}^n$ — предсказание модели.

3.2 Нейронные сети

В данной работе используются нейронные сети прямого распространения, обученные с помощью алгоритма обратного распространения ошибки.

Метод обратного распространения ошибки использовался ещё в 70-х годах прошлого века, но именно в 1986г. была опубликована работа [1], после которой метод стал активно использоваться в нейронных сетях.

Нейронная сеть представляет собой ориентированный граф, в котором вершины это нейроны, а рёбра — связи между нейронами. Граф состоит из последовательности слоёв, на каждом из которых находится некоторое количество нейронов. Нейроны на k-ом слое имеют связи со всеми нейронами на $(k-1)$-ом слое, каждая из которых имеет свой веществennозначный вес.

Архитектура нейронной сети, используемой в работе, состоит из следующих слоёв:

Описание слоя
Входной слой, состоящий из m нейронов
Скрытый слой 1
Dropout-слой 1
Скрытый слой 2
Dropout-слой 2
Скрытый слой 3
Dropout-слой 3
Скрытый слой 4
Выходной слой с одним нейроном

Скрытые слои это стандартные слои с нейронами, описанные выше. Dropout-слои это специальные слои, которые обнуляют долю выходов нейронов на предыдущем слое. Dropout-слои позволяют бороться с эффектом переобучения. Долю обнуляемых нейронов будем называть dropout-rate. Количе-

ство нейронов на каждом скрытом слое и dropout-rate для каждого dropout-слоя — параметры архитектры нейронной сети.

Во входной слой подаются векторы-строки из матрицы X. Значения нейронов скрытого и выходного слоя равны:

$$s = f\left(\sum_{i=1}^{k} w_i x_i\right),$$

где k — число нейронов на предыдущем слое, w_i — вес связи между рассматриваемым нейроном и i-ым нейроном на предыдущем слое, x_i — значение i-ого нейрона на предыдущем слое.

$f : \mathbb{R} \to \mathbb{R}$ — функция активации.

В данной архитектуре для всех слоёв, кроме выходного, используется активационная функция relu: $f(x) = x_+ = max(0;\ x)$.

Так как решается задача регрессии, то на выходном слое обязательно используется линейная активационная функция: $f(x) = x$.

Обучение нейронной сети происходит по алгоритму обратного распространения ошибки, который кратко описан ниже:

1. Случайным образом инициализируются значения начальных весов связей.
2. Подача матрицы X на входной слой.
3. Вычисление вектора откликов $\hat{y}$ нейронной сети.
4. Вычисление разницы между вектором $\hat{y}$ и реальным значением вектора y.
5. Корректировка весов связей, приводящая к минимизации функции потерь. Для корректировки используется алгоритм $adam$.
6. Шаги с 2–6 повторяются некоторое заранее определённое число раз.

3.3 Random forest

Случайный лес — алгоритм машинного обучения, представляющий собой ансамбль независимых решающих деревьев.

Решающие деревья, как алгоритм машинного обучения, впервые были предложены в книге [2] в 1984г.

Случайный лес подробно описан в работе [3]. Далее приводится краткое описание идеи этого алгоритма.

Решающее дерево это двоичное дерево, в каждом узле которого записано правило, делящее множество наблюдений на две части. Правило представляет собой неравенство вида $x < a$, где x — значение переменной, по которой идёт разделение, a — вещественное число. Процесс предсказания можно описать следующим образом:

1. Корень дерева объявляется текущей вершиной.

2. К наблюдению применяется правило, записанное в текущей вершине дерева. Если правило в этой вершине выполняется для наблюдения, то текущей вершиной объявляется правый потомок, иначе — левый потомок.

3. Пункт (2) повторяется до тех пор, пока текущая вершина дерева не окажется листом.

4. Результат предсказания есть, например, среднее арифметическое всех значений таргетной переменной объектов обучающей выборки, попавших в этот лист.

Работа алгоритма случайного леса заключается в построении заранее определённого количества независимых решающих деревьев. Процесс предсказания в задаче регрессии представляет собой некий вариант усреднения результатов каждого отдельного дерева решений, например, среднее арифметическое.

3.4 Gradient boosting machine

Метод градиентного бустинга предложен впервые в 1999г. в работе [4].

Основная идея этого метода состоит в следующем: к уже имеющемуся ансамблю деревьев добавляется новое дерево, наилучшим образом улучшающее качество предсказания всего ансамбля. То есть, ансамбль из деревьев строится последовательно, используя жадную стратегию.

Существует много разных вариаций градиентного бустинга. В этой работе используется алгоритм LightGBM, разработанный корпорацией Microsoft, подробное описание которого можно найти в работе [5].

3.5 Обзор используемых технологий

В качестве языка программирования для написания данной работы используется Python 3.7. В частности активно используются следующие Python-фреймворки:

- pandas [8] — библиотека для работы с датафреймами.
- numpy [11] — библиотека для вычислений и статистической обработки данных.
- sklearn [9] — фреймворк, в котором реализован набор алгоритмов машинного обучения, в частности алгоритм случайного леса, а также алгоритмы для предобработки данных.
- keras [10]— фреймворк, реализующий алгоритмы нейронных сетей.
- lightgbm [15] — фреймворк, реализующий алгоритм градиентного бустинга.

Также в силу отстуствия достаточно большого объёма вычислительных мощностей на персональном компьютере все вычисления проводились в облачной системе Kaggle Kernels ([14]), специально созданной командой `kaggle.com` для экспериментов в области машинного обучения. Kaggle Kernels предоставляет вычислительную систему, на которой можно запускать проекты, работа которых занимает до 9 часов, для работы предоставляются 21 гигабайт пространства на виртуальном диске, 4 CPU-ядра и 17 гигабайт оперативной памяти.

Исходные коды программ размещены по адресу `https://yadi.sk/d/6RpivY94POp5WA`.

4 Оптимизация гиперпараметров

При решении задачи машинного обучения для повышения качества предсказания важно не только провести работу с исходными данными, но и провести настройку гиперпараметров модели.

Гиперпараметры модели это параметры модели, определяемые программистом до начала процесса обучения модели.

Ниже приведены конкретные примеры гиперпараметров для различных классов решающих функций:

- Нейронные сети: learning rate, архитектура сети, в частности: количество нейронов на каждом слое, доля нейронов, отбрасываемых dropout-слоем.
- Random forest: количество деревьев, максимальная глубина дерева, количество используемых каждым деревом факторов.
- Gradient boosting: количество итераций алгоритма, learning rate, а также гиперпараметры решающих деревьев.

4.1 Описание алгоритма GridSearch

Один из самых простых алгоритмов нахождения оптимального набора гиперпараметров — GridSearch. Этот алгоритм основан на идее кросс-валидации. Исходное обучающее множество делится на три части: обучающую выборку, выборку валидации и тестовую выборку.

Для каждого гиперпараметра, подлежащего определению, задаётся набор рассматриваемых значений. Затем в цикле перебираются все возможные комбинации гиперпараметров, для каждой комбинации на обучающей выборке обучается модель, задаваемая текущей комбинацией, после этого на выборке валидации вычисляется функция потерь.

Модель с набором гиперпараметров, доставляющим минимум функции потерь на выборке валидации, объявляется оптимальной. После этого вычисляется функция потерь для оптимальной модели на тестовой выборке.

Если значение функции потерь на выборке валидации сильно меньше значения функции потерь на тестовой выборке, то это означает, что модель «переобучена», то есть, она подстроилась под случайно обнаруженные закономерности в выборке валидации, которых нет в генеральной совокупности.

Если же значения функции потерь на выборке валидации и на тестовой выборке приблизительно равны, то такую модель можно использовать.

4.2 Сэмплирование

Исходное обучающее множество слишком большое, чтобы его целиком использовать для поиска оптимальных гиперпараметров, обучение занимает слишком много времени. Поэтому были применены методы сэмплирования.

Исходная задача имеет особенность: наблюдения связаны друг с другом, так как игроки играют в одном и том же матче. Поэтому выделение подвыборки из исходного множества должно быть произведено не просто по строкам матрицы X, но по отдельным матчам.

Всего в исходном обучающем множестве представлена информация о примерно 45000 сыгранных матчей.

Выберем три раза без возвращения по 5000 случайных матчей и сформируем три новых выборки: обучающую, валидационную и тестовую, на которых будет проводиться поиск оптимальных гиперпараметров.

Такой способ сэмплирования позволяет избежать утечки данных из валидационного множества в обучающее.

Значение 5000 подбиралось из соображений балансирования между временем обучения моделей и репрезентативностью выборок.

4.3 Поиск гиперпараметров для нейронной сети

Рассмотрим архитектуру нейронной сети, описанную в 3.2, и проведём для неё поиск оптимальных гиперпараметров по описанному выше алгоритму. Эксперименты показали, что значения dropout-rate выше 0.01 отрицательно влияют на модель, поэтому поиск вёлся только среди значений, достаточно близких к нулю. В таблице ниже представлены результаты работы:

Слой	Количество нейронов	Dropout-rate	Выбранное значение
скрытый слой 1	[100; 150; 200]	—	200
dropout-слой 1	—	[0; 0.001; 0.01]	0.001
скрытый слой 2	[100; 150; 200]	—	200
dropout-слой 2	—	[0; 0.001; 0.01]	0.01
скрытый-слой 3	[100; 150; 200]	—	200
dropout-слой 3	—	[0; 0.001; 0.01]	0.001
скрытый слой 4	[100; 150; 200]	—	200

При сравнении сети с нулевыми dropout-rate и с найденными выше оптимальными dropout-rate было замечено, что ненулевые значения параметра дают выигрыш функции потерь лишь в четвёртом знаке после десятичной точки, поэтому согласно методу Бритвы Оккама было решено остановиться на нулевых значениях dropout-rate, то есть, фактически, отказаться от dropout-слоёв.

4.4 Поиск гиперпараметров для случайного леса

Ниже приведена таблица со списком параметров, среди которых осуществлялся перебор, и значения параметров, которые были выбраны:

Параметр	Набор параметров	Выбранное значение
max_depth	[5; 9; 12; $None$]	$None$
min_samples_split	[2; 0.05; 0.1; 0.2]	2
min_samples_leaf	[1; 0.01; 0.05]	1
max_features	[$'sqrt'$; 0.5; 0.75]	0.5

Подробное описание этих параметров модели можно найти в документации [12].

4.5 Поиск гиперпараметров для градиентного бустинга

Ниже приведена таблица со списком параметров, среди которых осуществлялся перебор, и значения параметров, которые были выбраны:

Параметр	Набор параметров	Выбранное значение
num_leaves	[10; 20; 31; 40]	31
learning_rate	[0; 0.01; 0.05; 0.1]	0.05
bagging_fraction	[0.7; 0.8; 0.9; 1]	0.9
colsample_bytree	[0.7; 0.8; 0.9; 1]	0.7

Подробное описание этих параметров модели можно найти в документации [13].

5 Результаты работы

5.1 Калибровка модели

Значение таргетной переменной winPlacePerc рассчитывается на основе переменной maxPlace, а не numGroups, как было отмечено в 1.3. Значения winPlacePerc внутри одного матча следует понимать как maxPlace равноотстоящих точек в отрезке $[0;\ 1]$, где первая точка это 0, а последняя — 1.

Задача регрессии это задача предсказания непрерывной переменной, но в нашей задаче, по сути, требуется предсказать дискретную переменную.

Поэтому для повышения качества предсказания использовалась калибровка вектора предсказаний $\hat{y}$, которую можно описать следующим образом:

- $\hat{y}_i^* = 0$; если $\text{maxPlace}[i] = 0$.
- $\hat{y}_i^* = 1$; если $\text{maxPlace}[i] = 1$.
- $\hat{y}_i^* = round(\hat{y}_i / gap) \times gap$, где $gap = 1/(maxPlace[i] - 1)$, а $round(x)$ — функция округления x до ближайшего целого; если $\text{maxPlace}[i] > 1$.

Вектор $\hat{y}^*$ есть результат калибровки.

5.2 Результаты работы моделей

Обучим полученные в 4 модели на всём обучающем множестве.

В качестве критерия остановки обучения для нейронных сетей из соображений ограничения по времени работы программы в системе Kaggle Kernel было выбрано разное количество эпох обучения: 30, 300, 700.

Был выбран следующий критерий остановки обучения алгоритма градиентного бустинга: сделано 20000 итераций обучения или остановить обучение, если качество модели не меняется в течение 200 итераций.

В качестве критерия остановки обучения для случайного леса было выбрано: построение 50 и 100 деревьев.

После обучения по объектам тестовой выборки предсказывается неизвестный вектор $\hat{y}$.

Затем к результатам каждой модели был применён описанный выше процесс калибровки предсказания.

Проверка качества модели проводилась на сайте `kaggle.com` в интерфейсе дорешивания соревнования (то есть, отображается результат посылки, но не меняется позиция в турнирной таблице).

В таблице ниже приведены результаты проверки качества моделей:

Модель	Значение функции потерь MAE
Нейронная сеть (30 эпох обучения)	0.03938
Нейронная сеть (300 эпох обучения)	0.02923
Нейронная сеть (700 эпох обучения)	0.02860
Случайный лес (50 деревьев)	0.02384
Случайный лес (100 деревьев)	0.02369
Градиентный бустинг	0.02055

Лучшим по значению метрики качества MAE оказался алгоритм градиентного бустинга. На момент окончания соревнования такой результат бы занял 251 место среди 1534 участников. Этот результат попадает в топ-16% участников соревнования.

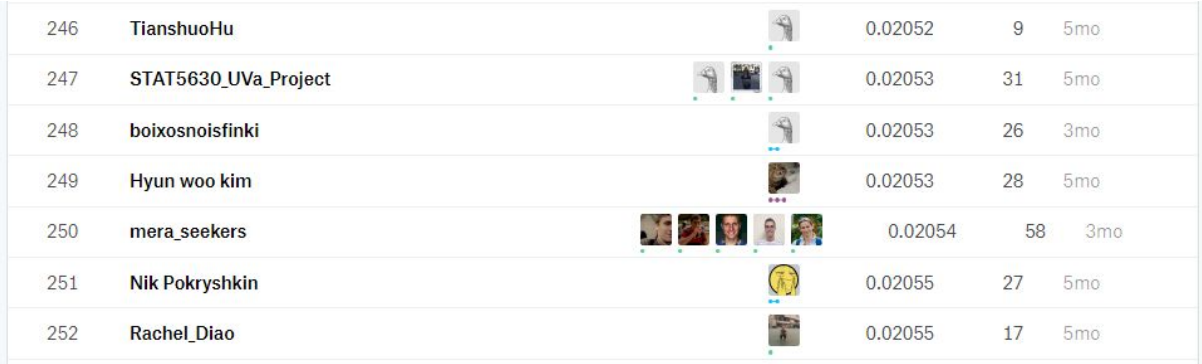

Рис. 13: Фрагмент турнирной таблицы соревнования

6 Анализ модели градиентного бустинга

В библиотеке LightGBM [15] реализованы не только алгоритмы обучения и предсказания, но также и инструменты для анализа полученных моделей.

Оценить силу влияния каждой конкретной переменной на предсказание модели можно, например, следующими величинами:

1. Суммарное количество расщеплений по переменной среди всех деревьев. Этот метод называется split.

2. Суммарное количество величины gain, используемой решающими деревьями для оценки качества расщепления по переменной среди всех деревьев. Этот метод называется gain.

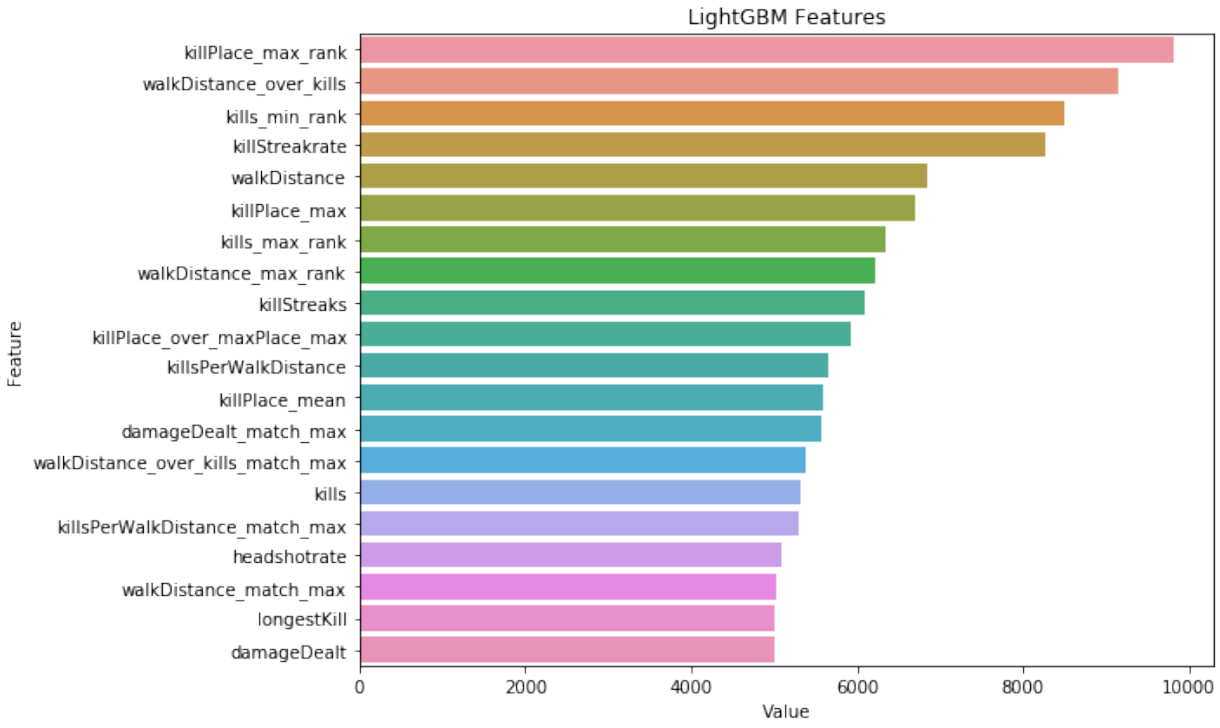

Рис. 14: 20 самых значимых переменных по методу split

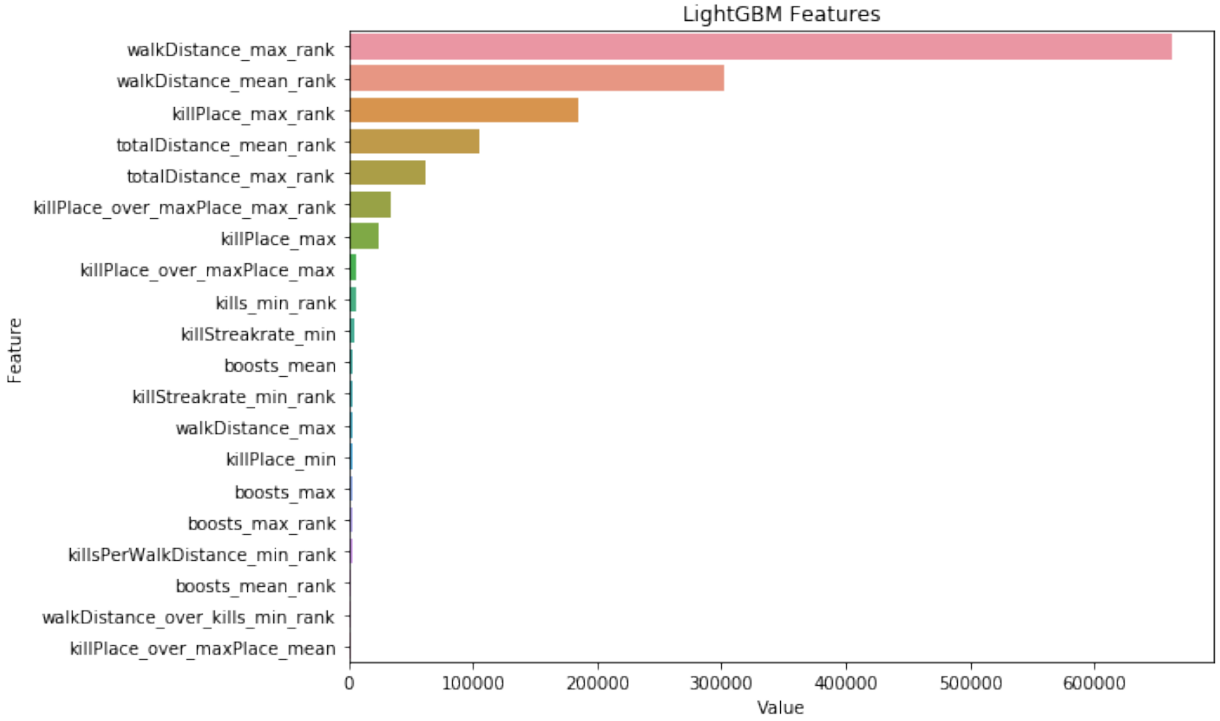

Рис. 15: 20 самых значимых переменных по методу gain

По этим диаграммам можно судить о важности введённых новых переменных. В первом случае 15 из 20 переменных были добавлены в ходе работы с базой данных. Во втором же случае все 20 переменных были добавлены в ходе работы с базой данных.

7 Заключение

В ходе работы получены следующие результаты:

- применены различные алгоритмы машинного обучения для решения исходной задачи,
- подробно описан процесс предобработки данных и поиска лучшей модели,
- приведено сравнение полученных результатов между собой, работа лучшей модели попадает в топ-16% среди участников соревнования,
- приведён краткий анализ лучшей из полученных моделей.

Список литературы

[1] D. E. Rumelhart, G. E. Hinton, R. J. Williams. Learning representations by back-propagation errors. Nature v. 323, p.533-536, 1986.

[2] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone. Classification and regression trees. Wadsworth, Inc., 1984.

[3] L. Breiman. Random forests. Machine Learning, 45:5 – 32, 10 2001.

[4] J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. The Annals of Statistics v. 29, p.1189-1232, 1999.

[5] Ke G. et al. Lightgbm: A highly efficient gradient boosting decision tree //Advances in Neural Information Processing Systems. – 2017. – С. 3146-3154.

[6] С. А. Арефьев. Модель оценивания пар игроков в теннис: дипломная работа. СПбГУ, Санкт-Петербург, 2017.

[7] PUBG Finish Placement Prediction (Kernels Only), URL: `https://www.kaggle.com/c/pubg-finish-placement-prediction`

[8] Python Data Analysis Library, URL: `https://pandas.pydata.org/`

[9] scikit-learn, URL: `https://scikit-learn.org/stable/`

[10] Keras: The Python Deep Learning library, URL: `https://keras.io/`

[11] NumPy, URL: `https://www.numpy.org/`

[12] sklearn.ensemble.RandomForestRegressor, URL: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html`

[13] LightGBM parameters, URL: `https://lightgbm.readthedocs.io/en/latest/Parameters.html`

[14] How to use Kaggle, URL: `https://www.kaggle.com/docs/kernels`

[15] LightGBM documentation, URL: https://lightgbm.readthedocs.io/en/latest/