09 marzec 2022

PUBG Finish Placement Prediction

Specyfikacja i Implementacja Systemu

Szadkowski Michał
Strzechowski Konrad
Szymański Mateusz
Sawicki Piotr

Spis treści

1 Wprowadzenie

Projekt realizowany jest w ramach przedmiotu "Warsztaty z technik uczenia maszynowego". Celem projektu jest na podstawie anonimowych danych z ponad 65 tysięcy rozegranych gier popularnego PUBG znaleźć najlepszą strategię do wygrywania gier.

2 Opis gry

PlayerUnknown's Battlegrounds (lub inaczej PUBG) to wieloosobowa gra komputerowa typu battle royale, w której gracze walczą między sobą a ten gracz lub drużyna, który zostanie ostatni przy życiu wygrywa. Ten typ rozgrywki zainspirowany jest filmem "Battle Royale" z roku 2000 a PUBG jest jednym z największych przedstawicieli gatunku.

Gracze mogą przystąpić do meczu solo, w duecie lub w drużynie czteroosobowej ale w jednym meczu występuje tylko jeden rodzaj drużyn. W meczu bierze udział co najwyżej 100 osób. Wszyscy gracze rozpoczynają rozgrywkę w samolocie lecącym nad mapą, z którego mogą w dowolnym momencie wyskoczyć na spadochronie. Po wylądowaniu gracze mogą przeszukiwać budynki, miasta i inne obiekty w poszukiwaniu pojazdów, broni, pancerza lub innych przedmiotów (np. bandaże), które są rozmieszczone na mapie w sposób przypadkowy. Wyeliminowani gracze pozostawiają swoje wyposażenie w miejscu śmierci a inni gracze mogą je zebrać. Począwszy od startu rozgrywki strefa rozgrywki stopniowo zawęża się w przypadkowo obranym kierunku a gracze przebywający poza nią otrzymują obrażenia. Gracze nie mogą się odradzać ale w przypadku gry w drużynie, postać gracza po śmierci może, w określonym czasie, zostać uratowana przez innego członka zespołu i kontynuować rozgrywkę. Mecz wygrywa gracz lub zespół, którego członek jako jedyny na mapie pozostanie żywy.

3 Analizowane dane

Zestaw danych dostarczonych do analizy można podzielić na 4 kategorie:

- ogólne dane dotyczące rozgrywki (ilość drużyn biorących udział w rozgrywce, czas trwania rozgrywki)
- dane dotyczące indywidualnych osiągnięć gracza w analizowanej rozgrywce (zabójstwa, zadane obrażenia, wskrzeszenia innych graczy)
- dane dotyczące eksploracji mapy przez gracza (przebyty dystans, ilość podniesionych broni)
- ogólne dane na temat historii rozgrywek gracza (pozycja w ogólnym rankingu).

4 Cel projektu

Celem projektu jest stworzenie modelu, który będzie przewidywał końcowe miejsce graczy na podstawie ich ostatecznych statystyk. Model będzie określał zajmowane miejsca w skali od 1 do 0, gdzie jedynka oznacza pierwsze miejsce a zero ostanie miejsce.

5 Założenia technologiczne

Projekt będzie realizowany z wykorzystaniem języka Python w wersji 3.10. Do przetwarzania danych wykorzystany również środowisko Google Colab, które jest środowiskiem Jupyter dostarczanym i obsługiwanym przez Google. Skorzystamy również z licznych bibliotek języka Python. Między innymi do przetworzenia i obróbki danych wykorzystamy NumPy oraz Pandas umożliwiające realizowanie w szybki sposób skomplikowanych operacji. W celu wizualizacji danych oraz otrzymanych wyników użyjemy biblioteki Matplotlib oraz Seaborn. Do implementacji, budowy, trenowania i pracy z modelami wykorzystamy biblioteke Sklearn. Zasób wykorzystywanych bibliotek jest większy jednak nie będziemy zagłębiać się tutaj w jego szczegóły.

6 Podział zadań

Przebieg prac w większości przebiegał grupowo. Zadania były rozdzielane na bieżąco po konsultacjach wewnątrz zespołu. Każdy uczestniczył podczas całego procesu tworzenia i rozwijania projektu. Mimo to zdecydowaliśmy się wytypować osoby odpowiedzialne za dostarczenie danej fazy projektu. Z tego powodu za wczytanie danych, ich wizualizacje oraz początkową obróbkę byli odpowiedzialni Michał Szadkowski i Mateusz Szymański. Do odpowiedzialność za proces budowy i ewaluacji modeli przypisany został do Konrada Strzechowskiego oraz Piotra Sawickiego.

7 Analizowanie i sprzątanie danych

Jak już wcześniej było wspomniane posiadamy zbiór anonimowych danych z ponad 65 tysięcy rozegranych gier popularnego PUBG. Jedna ramka danych składa się między innymi z danych dotyczących rozgrywki, indywidualnych osiągnięć gracza, statystyk eksploracji mapy czy też ogólnych danych na temat historii rozgrywek gracza. Zbiór tych danych ma nam pomóc znaleźć najlepszą strategie do wygrywania gier i stworzyć modele do przewidywania osiąganych wyników przez graczy.

7.1 Analiza ramki danych

Ramka danych jaką będziemy się posługiwać składa się z 28 predykatów, z których 24 ma charakter liczbowy. Kolumny 'id', 'groupId', 'matchId' oraz 'matchType' nie reprezentują liczb. Są w postaci obiektów. Pierwsze trzy identyfikują informacje o graczach z każdej grupy w każdym meczu w którym brali udział. Kolumna 'matchType' wskazuje na jeden z 16 możliwych typów meczy. Rozgrywki mogą być indywidualne,

w dwie osoby lub drużynowe. Mogą posiadać różne ograniczenia i ulepszenia. A także, mogą to być rozgrywki niestandardowe gdzie wszystko wygląda inaczej.

Typy gier:

- squad-fpp, squad, normal-squad-fpp, normal-squad **(gry drużynowe)**
- duo, duo-fpp, normal-duo-fpp, normal-duo **(gry duo)**
- solo-fpp, solo, normal-solo-fpp, normal-solo **(gry solo)**
- crashfpp, crashtpp, flarefpp, flaretpp **(gry niestandardowe)**

Pozostałe 24 predykatów mają typy liczbowe. Mają one typ całkowity lub zmiennoprzecinkowy. Można je w wszelaki sposób grupować i dzielić na kategorię. Zawierają między innymi ilość drużyn biorących udział w rozgrywce, czas trwania rozgrywki, zabójstwa, zadane obrażenia, wskrzeszenia innych graczy, przebyty dystans, ilość podniesionych broni itd. Szczegółowy opis każdej kolumny wraz z jej typem znajduję się w poniższej tabeli.

	Data field	Description	Type
0	Id	Player's Id	object
1	groupId	ID to identify a group within a match	object
2	matchId	ID to identify match	object
3	matchType	String identifing the game mode that the data ...	object
4	assists	Number of enemy players this player damaged th...	int64
5	boosts	Number of boost items used	int64
6	damageDealt	Total damage dealt	float64
7	DBNOs	Number of enemy players knocked	int64
8	headshotKills	Number of enemy players killed with headshots	int64
9	heals	Number of healing items used	int64
10	killPlace	Ranking in match of number of enemy players ki...	int64
11	killPoints	Kills-based external ranking of player	int64
12	killStreaks	Max number of enemy players killed in a short ...	int64
13	kills	Number of enemy players killed	int64
14	longestKill	Longest distance between player and player kil...	float64
15	matchDuration	Duration of match in seconds	int64
16	rankPoints	Elo-like ranking of player	int64
17	revives	Number of times this player revived teammates	int64
18	rideDistance	Total distance traveled in vehicles measured i...	int64
19	roadKills	Number of kills while in a vehicle	int64
20	swimDistance	Total distance traveled by swimming measured i...	float64
21	teamKills	Number of times this player killed a teammate	int64
22	vehicleDestroys	Number of vehicles destroyed	int64
23	walkDistance	Total distance traveled on foot measured in me...	float64
24	weaponsAcquired	Number of weapons picked up	int64
25	winPoints	Win-based external ranking of player	int64
26	numGroups	Number of groups we have data for in the match	int64
27	maxPlace	Worst placement we have data for in the match	int64
28	winPlacePerc	The target of prediction	float64

Rysunek 1: Opis pól danych.

7.2 Czyszczenie danych

Przed przystąpieniem do swobodnego operowania danymi musimy jeszcze pozbyć się danych niepoprawnych, które mogły by znacząco zaburzać nasze obliczenia. Mówiąc o danych niepoprawnych mamy na myśli dane, mające braki, pochodzące z gier niestandardowych lub znacząco odstające. Dane znacząco odstające to takie gdzie statystyki osiągają poziom niemożliwy do osiągnięcia przez zwykłego gracza. Tak wysoki poziom statystyk może oznaczać używanie wspomagaczy podczas rozgrywki, co jest zakazane i nie będzie przez nas rozważane. Przystępujemy do czyszczenie mając dokładnie **4,446,965** wierszy danych. Ta liczba będzie się zmieniała podczas procesu czyszczenia.

7.2.1 Brakujące dane

Czyszczenie danych rozpoczniemy od usunięcia danych, w których brakuje przewidywanej zmiennej "winPlacePerc". W zestawie danych znajduje się tylko jeden taki wiersz i go odrzucamy.

7.2.2 Gry niestandardowe

Jak już wspomnieliśmy w PUBG istnieje wiele trybów gry. Jedną z kategorii takich gier są gry niestandardowe. Jest to niewielki promil wszystkich gier. Gry niestandardowe różnią się jednak znacząco od zwykłych gier. Logika i przebieg rozgrywki jest niepodobny do innych trybów. Z powyżej wymienionych powodów decydujemy się na usunięcie gier niestandardowych (Rozgrywki gdzie matchType to flaretpp, flarefpp, crashtpp lub crashfpp). Okazuję się, że wierszy dotyczących gier niestandardowych było **9,881**. Oznacza to, że po tym kroku zostało nam **4,437,084** wierszy danych.

7.2.3 Gracze nieaktywni (AFKs)

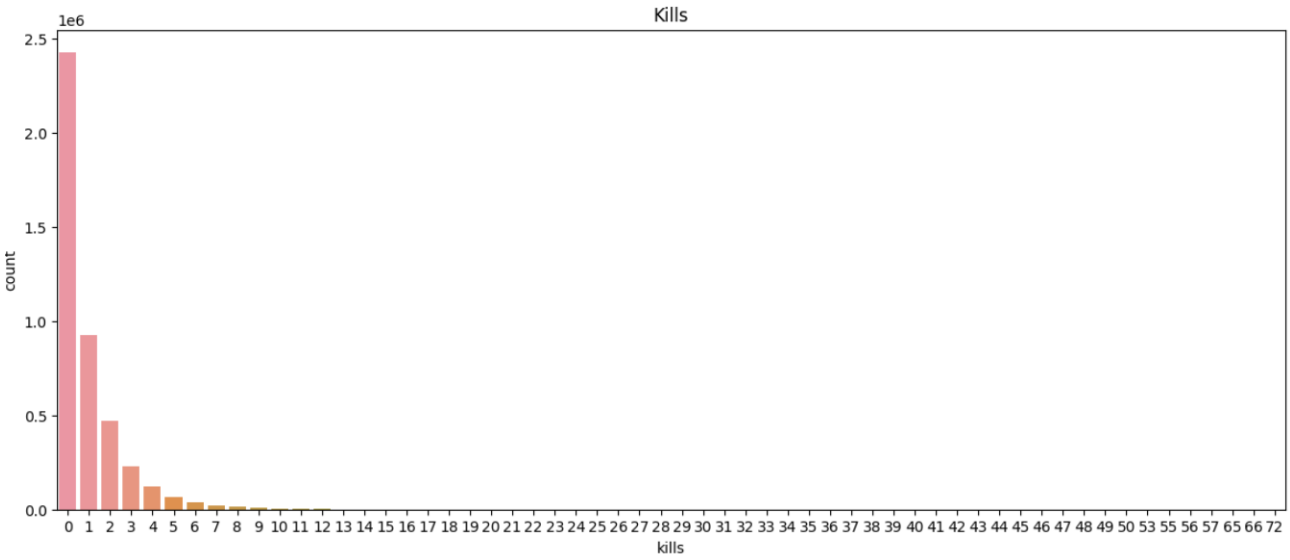
W kolejnym kroku pozbędziemy się danych graczy, którzy nie poruszyli się przez całą rozgrywkę. Bez wątpliwości możemy stwierdzić, że jeśli gracz nie poruszył się przez całą rozgrywkę to nie brał w niej udziału. Chcemy pozbyć się takich danych. W tym celu badamy kolumnę 'walkDistance'. Okazuję się, że nasz zbiór zawierał aż **99,364** wierszy danych, gdzie gracz nie poruszył się przez całą rozgrywkę. Po usunięciu nadal pozostaje nam aż **4,337,720** wierszy danych.

7.2.4 Oszustwo prędkości poruszania

Tak jak w każdego typu grze, znajdą się gracze, którzy w celu poprawienia swojego ego używają wspomagaczy. Uznaliśmy, że nie chcemy danych graczy, podejrzanych o oszustwa w dalszej części projektu. Z tego powodu próbowaliśmy je zidentyfikować i usunąć. Walkę z oszustami zaczęliśmy od próby zidentyfikowania 'SpeedHacka'. Wspomaganie do prędkości poruszania się jest niedozwolone i znacznie ułatwia użytkownikom rozgrywkę. W tym celu usuniemy graczy, którzy przebywali w jednej rozgrywce nieosiągalne dystansy. Identyfikujemy **248** graczy, którzy przepłynęli więcej niż 2km, przejechali więcej niż 30km lub przebiegli więcej niż 10km i je usuwamy.

7.2.5 Wspomaganie do zabijania

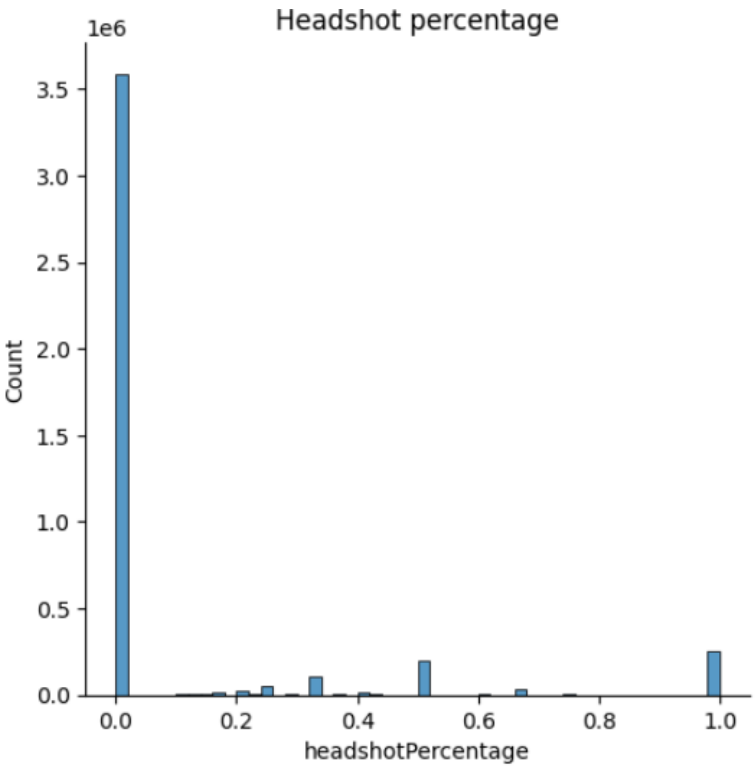
Analizując rozkład całkowitej liczby zabójstw każdego gracza podczas jednej rozgrywki (przedstawiony na rysunku nr. 2) widzimy, że większość graczy nikogo nie zabija podczas całej rozgrywki. Co więcej znikoma część graczy uzyskuję więcej niż 10 zabójstw. Bazując na tym oraz na własnym doświadczeniu z gier Battle Royal uznajemy, że zdobycie więcej niż 40 zabójstw jest niemożliwe w uczciwy sposób. Z tego powodu usuwamy **36** wierszy danych, zawierających znaczne odchylenie w liczbie zabójstw.



Rysunek 2: Liczba zabójstw przypadająca na gracza.

7.2.6 Potencjalny aimbot

Bardzo popularnym oszustwem jest wspomaganie do celowania i strzelania. Dzięki temu tacy graczę zawsze strzelają przeciwnikowi w głowę i osiągają dużą liczbę zabójstw. Szukając graczy używających aimbota, badamy procentowy rozkład zabójstw strzałem w głowę dla każdego gracza podczas jednej rozgrywki. Na rysunku nr. 3 widzimy, że jest pewna grupa graczy mających 100% zabójstw strzałem w głowę. Jednak nie każdy gracz taki gracz musi być oszutem. Mogą to być po prostu dobrzy gracze lub gracze mający szczęście w danej rozgrywce. Z tego powodu usuwamy tylko graczy mających więcej niż 10 zabójstw oraz wszystkie te zabójstwa zakończone strzałem w głowę. Okazuję się, że jest tylko **24** takich wierszy. Usuwamy je.



Rysunek 3: Procent zabójstw strzałem w głowę przypadających na gracza na gracza.

7.3 Podsumowanie czyszczenia danych

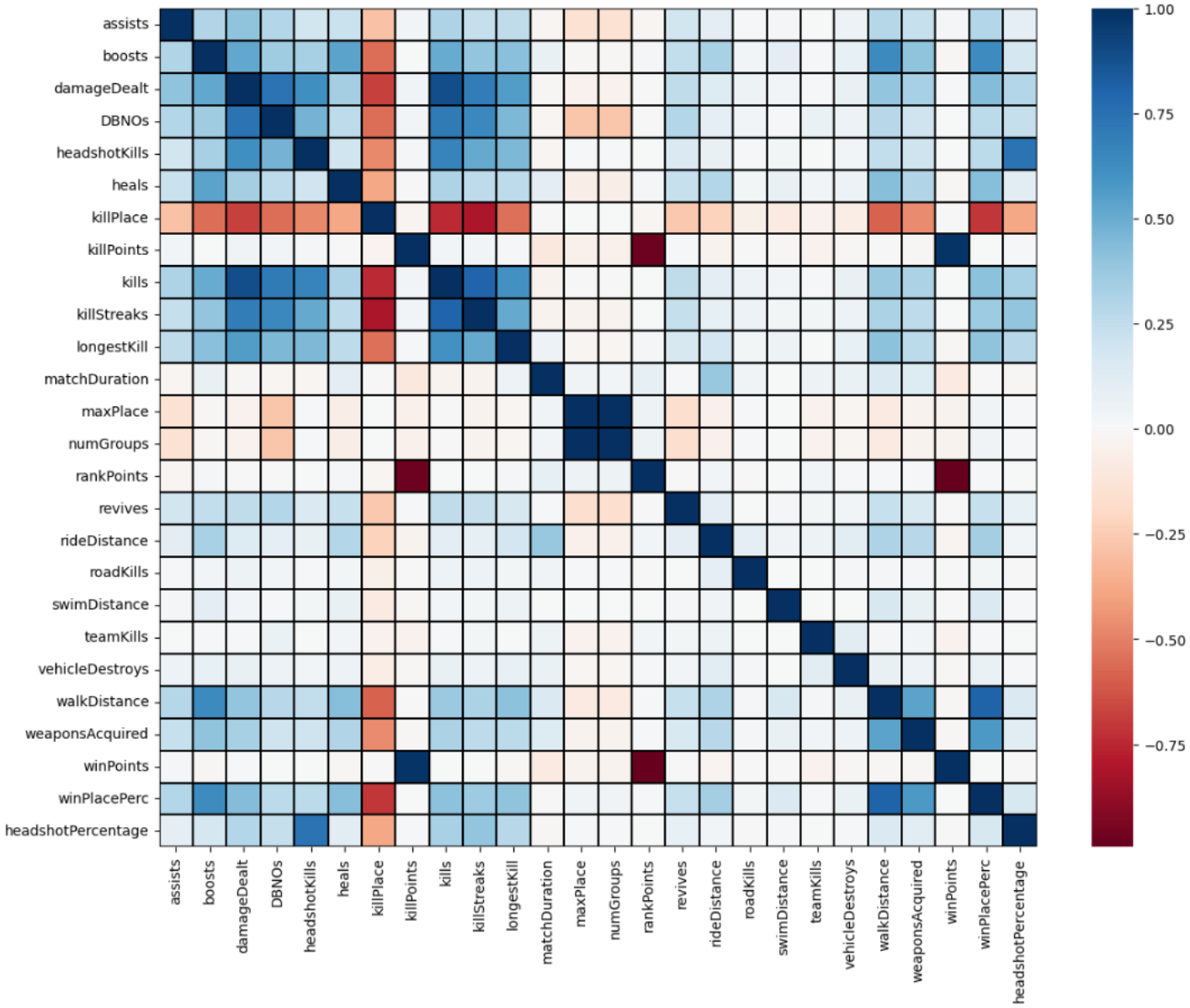
Przez cały proces czyszczenia danych, pozbyliśmy się dokładnie **109,553** wierszy. Dzięki temu, dane są teraz gotowe do dalszej analizy i ewaluowania przez modele.

Etap	Liczba usuniętych danych
Brakujące dane	1
Gry niestandardowe	9,881
Gracze nieaktywni (AFKs)	99,364
Gracze nieaktywni (AFKs)	99,364
Oszustwo prędkości poruszania	248
Wspomaganie do zabijania	36
Potencjalny aimbot	24
Łącznie	109,553

Tablica 1: Tabela prezentuje ilość usuwanych danych podczas konkretnych etapów.

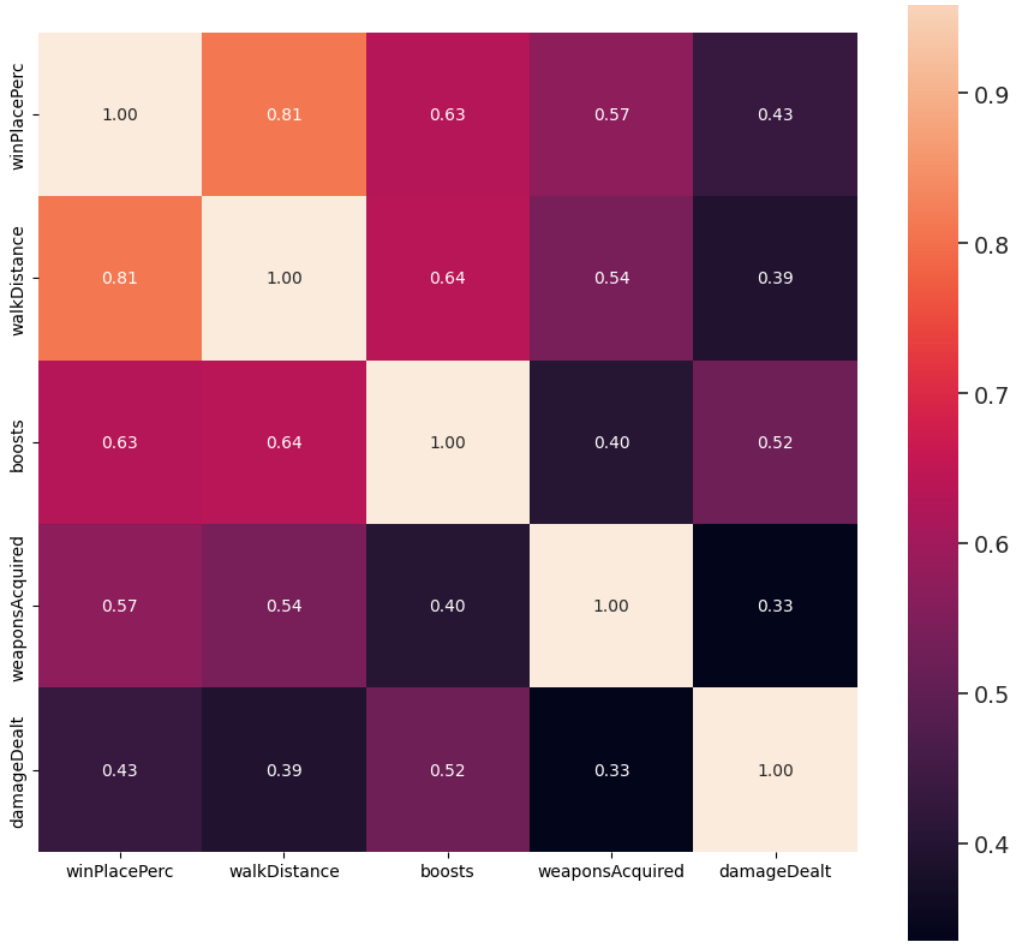
8 Szukanie najlepszej strategii

W tym kroku zajęliśmy się analizą poszczególnych zmiennych, wyznaczyliśmy korelacje zachodzące pomiędzy nimi oraz ich wpływ na końcową pozycję gracza.



Rysunek 4: Współczynniki korelacji

Możemy zauważyć, że pomiędzy niektórymi zmiennymi takimi jak: *roadKills*, *teamKills*, *vehicleDestroys*, nie zachodzi korelacja z żadnymi innymi predykatami. Oznacza to, że nie będą one miały dużego wpływu na końcową pozycję gracza. Widoczne są również zmienne wysoko skorelowane. Spośród nich wybraliśmy 4 najbardziej powiązane z obiektem przewidywań i umieściliśmy je w poniższej macierzy.

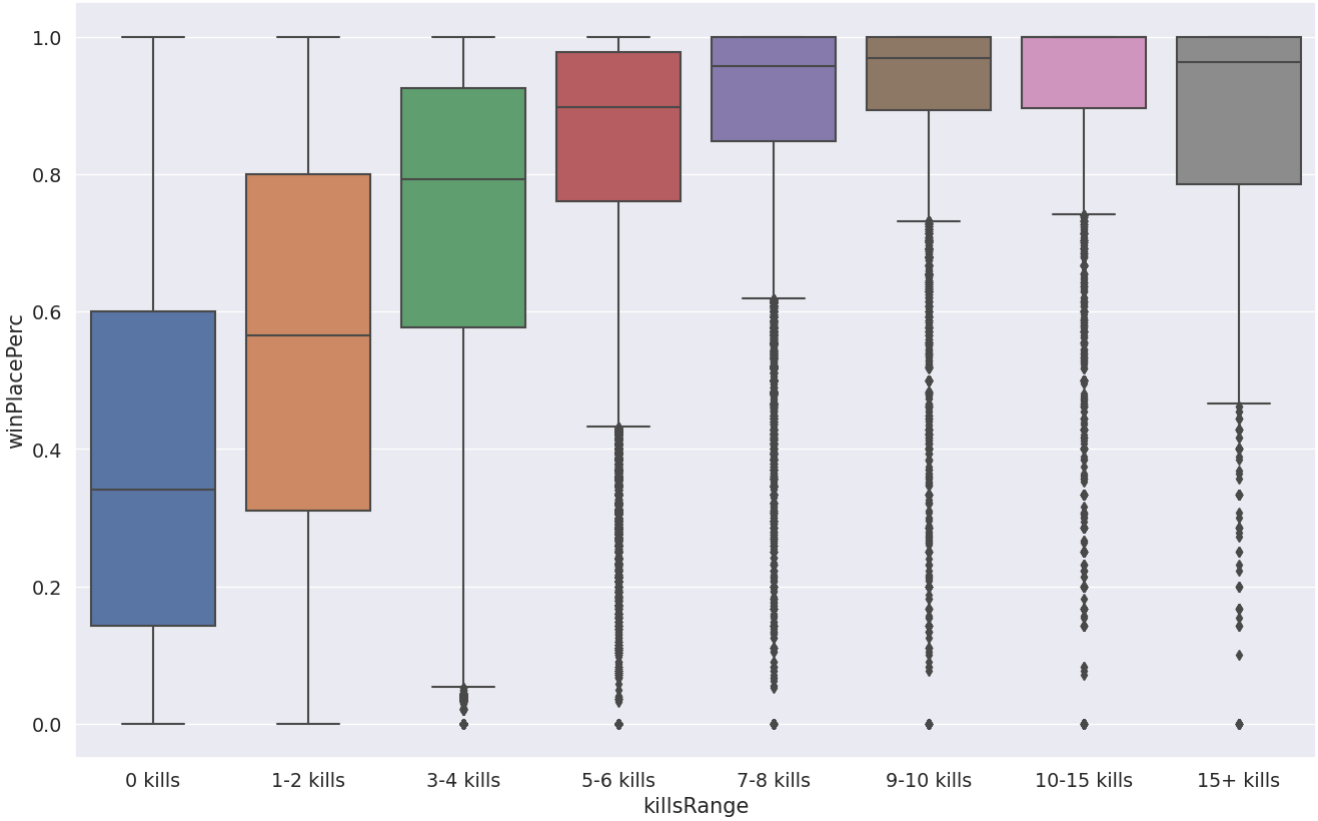


Rysunek 5: Wybrane współczynniki korelacji

Możemy wnioskować, że uzyskane predykaty odegrają kluczową rolę w przewidywaniu końcowej pozycji gracza.

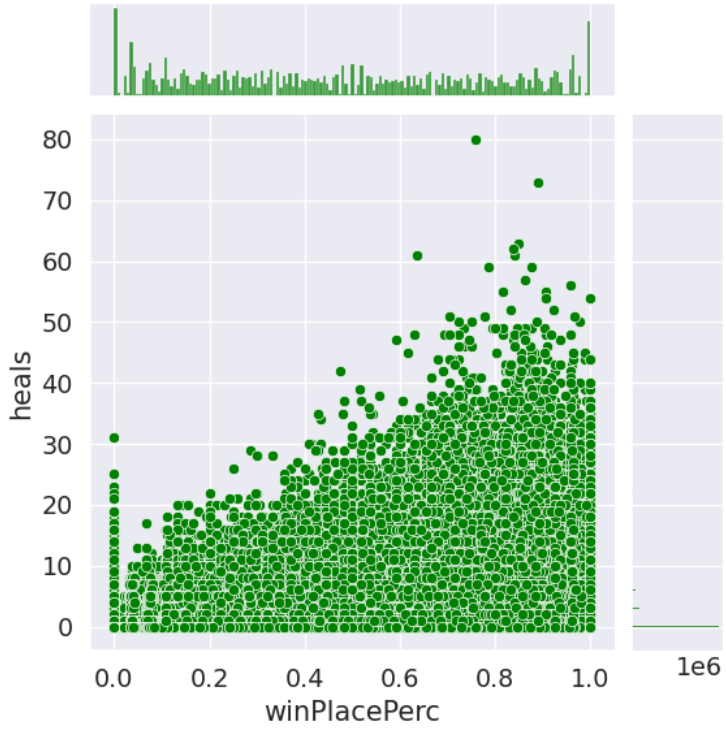
8.1 Wpływ predykatów na końcową pozycję

Następnie skupiliśmy się na dokładnej analizie wybranych predykatów i ich związku z ostateczną pozycją gracza. Jedną z najbardziej oczywistych zmiennych jest oczywiście liczba zabójstw. Podzieliliśmy graczy względem uzyskanych *killi* i przedstawiliśmy na poniższym wykresie.

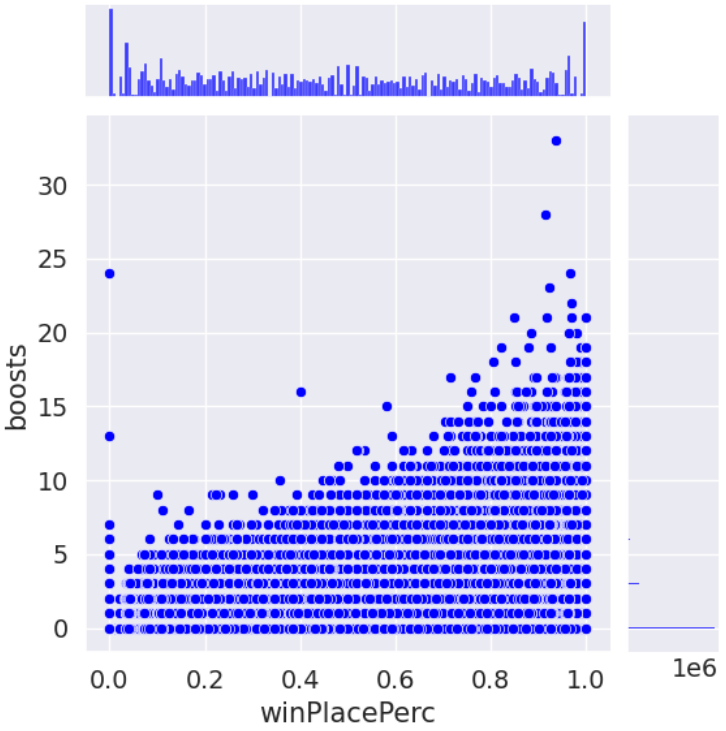


Rysunek 6: Wpływ zabójstw na końcową pozycję

Zaobserwowaliśmy względnie przewidywane rozłożenie się graczy względem ich zdobytych zabójstw. Co ciekawe w przypadku osób, które uzyskały 15 lub więcej *killi*, wielu z nich skończyło grę na niższym miejscu niż osoby, które uzyskały od 7 do 15 *killi*. Możemy podejrzewać, że tacy gracze byli zbyt nieostrożni lub pewni siebie, przez co w późniejszym etapie gry sobie nie radzili.



Rysunek 7: Wpływ leczeń na końcową pozycję



Rysunek 8: Wpływ boostów na końcową pozycję

Kolejnymi rozpatrzonymi predykatami są użyte przedmioty leczące *heals* i tak zwane boosty *boosts* w ciągu trwania rozgrywki. W tym przypadku podobnie do zdobytych zabójstw liczba użytych leczeń oraz boostów rozłożyła się zgodnie z oczekiwaniami. Im lepszą pozycję zajął dany gracz, tym więcej czasu był w grze, a przez to miał średnio więcej okazji do użycia leczenia. Pamiętajmy, także że rozpatrywane przedmioty są jednorazowego użytku, co oznacza, że gracz musiał je wcześniej zdobyć (przebyć większą drogę), aby mieć możliwość ich użycia. W tym przypadku anomalią są gracze, którzy kończyli grę na jednych z najgorszych pozycji, a mimo to używali wielu leczeń. Jest to przede wszystkim widać na wykresie *heals*. Prawdopodobnie są to gracze niedoświadczeni, którzy skupiali się głównie na zbieraniu przedmiotów, ale odnosili przy tym duże obrażenia od innych osób.

9 Nowe predykaty

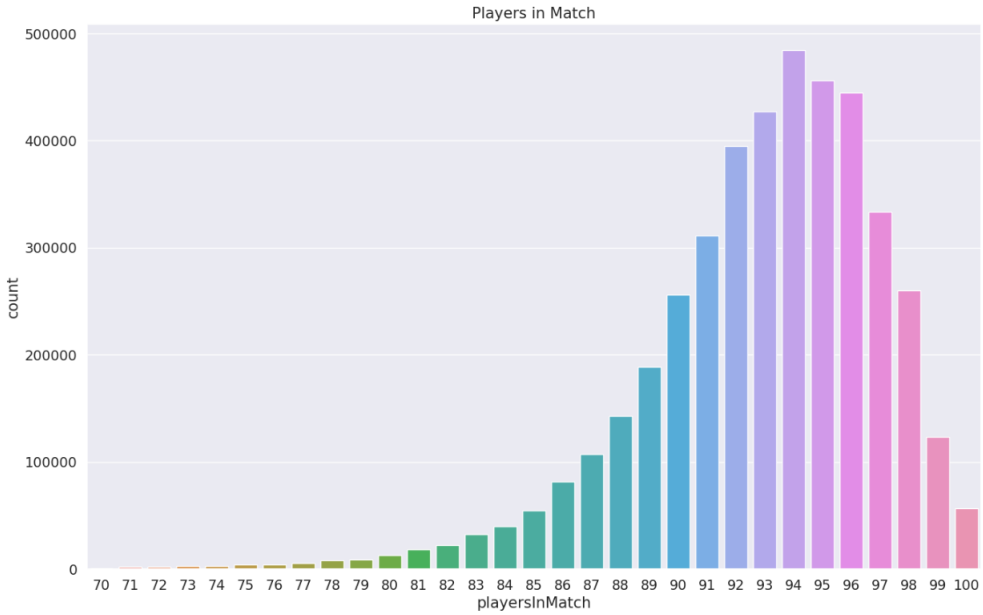
W poniższej sekcji zajmiemy się tworzeniem i omawianiem nowych predykatów. Skupimy się na własnościach będących funkcjami pojedynczych wierszy ale ponadto, ze względu na to, że w meczu zazwyczaj bierze udział trochę mniej niż 100 graczy, będziemy grupować mecze względem *matchId*, zliczać liczbę graczy w meczu i normalizować inne wartości.

9.1 headshotPercentage

Jednym z najlepszych wyznaczników umiejętności gracza w grach takich jak PUBG może być stosunek zabójstw w głowę do wszystkich zabójstw. Im jest on większy tym silniej można przypuszczać, że gracz ma duże doświadczenie w tej oraz podobnych grach. Utworzymy więc nowy predykat obliczany w sposób headshotPercentage = $\frac{\text{headshotKills}}{\text{kills}}$. Wykres przedstawiający utworzoną zmienną znajduje się na Rysunku 3.

9.2 Normalizacja

W jednym meczu maksymalnie bierze udział 100 graczy. W rzeczywistości ta liczba jest zazwyczaj trochę mniejsza, może to wynikać z tego, że gracze rozłączą się z poczekalni przed rozgrywką a nowi gracze nie zastąpią ich. Ze względu na zmniejszoną liczbę uczestników, gracz ma mniejsze szanse na zdobycie np. zabójstwa lub asysty. W tym celu utworzymy zmienną *playersInMatch* grupując nasze dane względem meczu i zliczając uczestników. Uzyskane wartości przedstawione są na Rysunku 9.



Rysunek 9: Liczba graczy w rozgrywce.

Tą wartość wykorzystamy do zdefiniowania zmiennych znormalizowanych:

- killsNorm - liczba zabójstw,
- damageDealtNorm - zadane obrażenia,
- assistsNorm - asysty,
- DBNOsNorm - powalenia.

Wszystkie znormalizowane predykaty powstały przez podzielenie wartości nieznormalizowanej przez $\frac{100-playersInMatch}{100} + 1$.

9.3 totalDistance

Kolejnym wyznacznikiem mocno związanym z ostateczną pozycją zajętą przez gracza może być całkowity przebyty dystans. Wynika to z tego, że wraz z trwaniem rozgrywki gracze będą się przemieszczali na pieszo, płynąc lub wykorzystując pojazdy. Ci, którzy odpadną na samym początku rozgrywki będą mieli mniejsze szanse przebyć tak duży dystans jak ci, którzy odpadną na końcu rozgrywki lub ją wygrają. Z drugiej strony wśród graczy istnieje strategia ukrywania się i przemieszczania się tylko niezbędnych dystansów tak by pozostać w strefie rozgrywki. Załóżymy, że jest ona na tyle rzadko spotykana, że nie będzie miała ona dużego wpływu na dalsze rozważania. Obliczymy ten predykat sumując dystanse przebyte na wszystkie możliwe sposoby: totalDistance = rideDistance + swimDistance + walkDistance.

9.4 items

Analogiczna sytuacja do powyższej ma miejsce w przypadku zebranych podczas rozgrywki przedmiotów leczących oraz tzw. boostów. Czyli im dłużej w rozgrywce gracz pozostał tym miał więcej okazji by je zebrać. Może się to przekładać na ostateczne zajęte miejsce w meczu. Na wykresach 8 oraz 7 przedstawiono graczy, którzy zajęli *winPlacePerc* miejsce oraz użyli *boosts* boostów lub *heals* przedmiotów leczących.

9.5 walkDistancePerKill

Na koniec wprowadzimy zmienną będącą stosunkiem przebytego dystansu do liczby zabójstw. Może ona świadczyć o strategii gracza podczas meczu, tzn. czy angażuje się w konflikty czy stara się ich unikać.

10 Badanie wpływu nowych predykatów

W celu oszacowania wpływu dodanych oraz już istniejących zmiennych stworzymy kilka modeli:

- model liniowy, którego błędy wykorzystamy do porównania dodanych zmiennych,
- las losowy, z którego w bezpośredni sposób pobierzemy ważność predykatów,
- LightGBM, z którym postąpimy tak jak z lasem losowym.

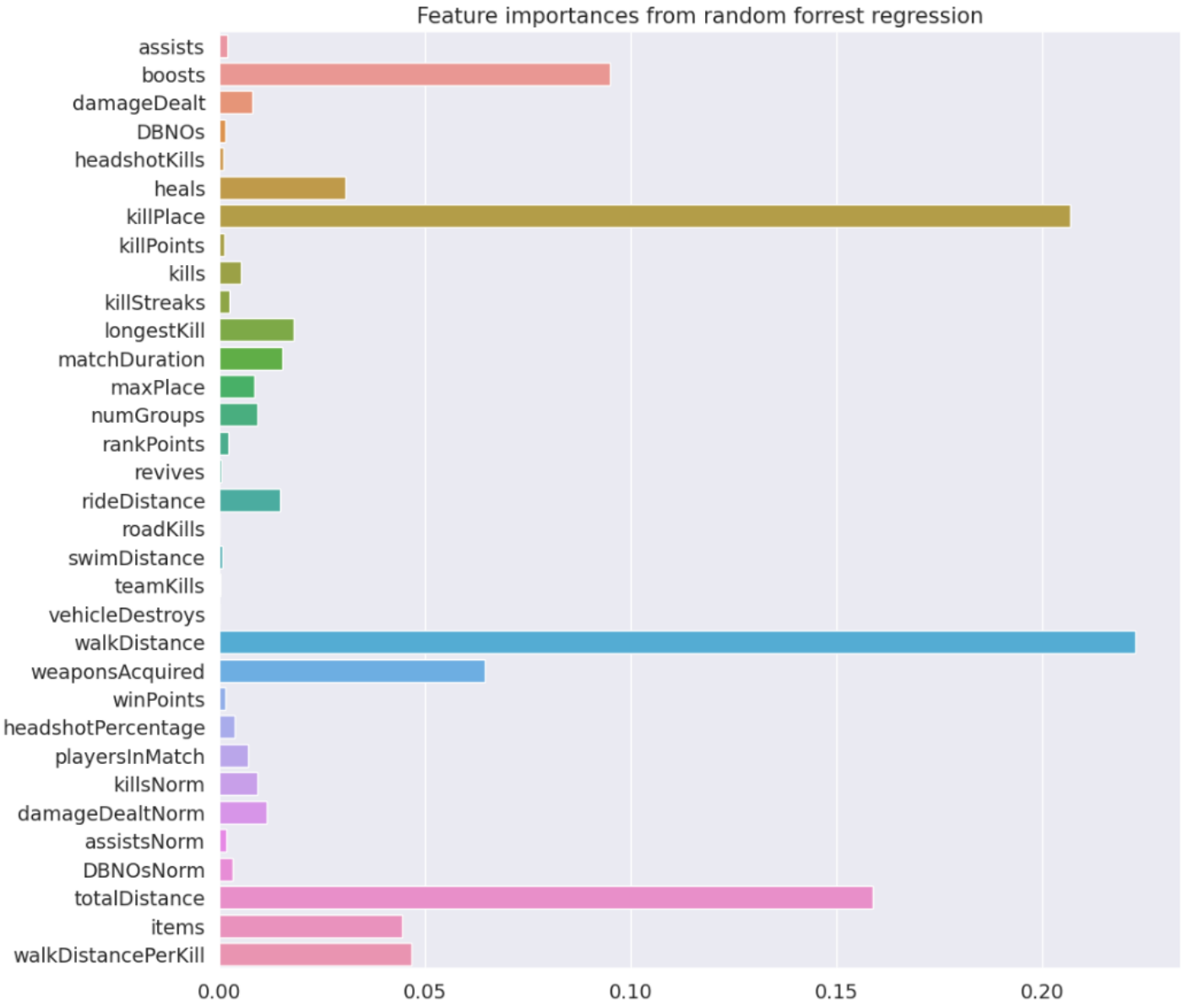
W poniższej tabeli przedstawiono porównanie błędów modeli liniowych tworzonych dla różnych nowych predykatów.

predykaty	błąd
domyślne	0.015742
znormalizowane i nieznormalizowane	0.015623
znormalizowane bez nieznormalizowanych	0.015760
całkowity dystans	0.015742
procent zabójstw w głowę	0.015472
podniesione przedmioty	0.015742
dystans względem zabójstw	0.015717
wszystkie	0.014732

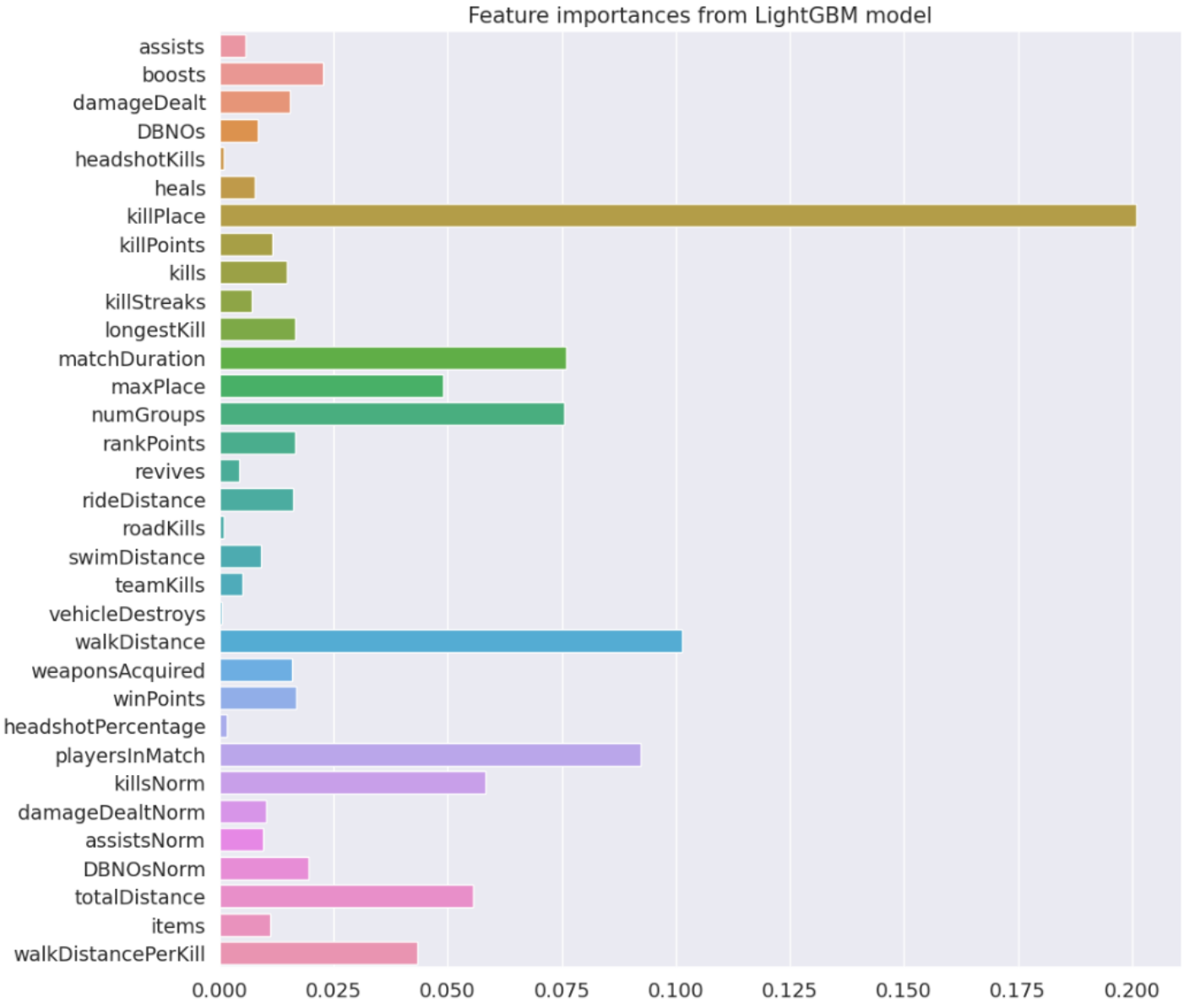
Tablica 2: Modele liniowe

Jak można zaobserwować najlepsze wyniki z pojedynczych predykatów dawał procent zabójstw w głowę. Różnice pomiędzy modelami są jednak bardzo małe. Dopiero łączne wykorzystanie wszystkich utworzonych wartości daje zauważalną poprawę wyniku.

Na rysunkach 10 oraz 11 przedstawiono ważności wszystkich predykatów dla odpowiednio lasu losowego oraz LightGBM. Możemy zaobserwować, że w obu przypadkach, mimo różnic w innych wartościach, wprowadzone parametry znormalizowane mają około dwa razy większą ważność od ich nieznormalizowanych odpowiedników. Dużą wagę ma również w obu przypadkach parametr *totalDistance* wprowadzony wcześniej, jednak jest ona mniejsza niż waga dystansu przebytego na pieszo. Jednym z najważniejszych predykatów okazał się istniejący w zbiorze danych *killPlace* będący miejscem w meczu pod względem liczby zabójstw.



Rysunek 10: Waga parametrów modelu Random Forrest

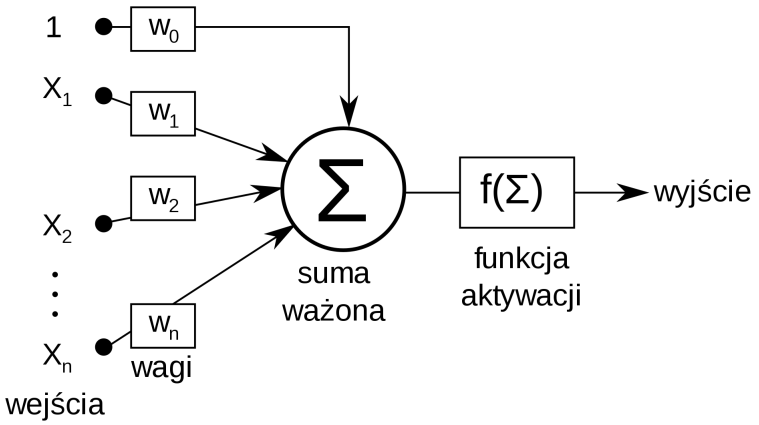


Rysunek 11: Waga parametrów modelu LightGBM

11 Opis modeli oraz proces ich tworzenia

11.1 Multi-Layer Perceptron

Jest to najczęściej używany typ sieci neuronowych. Model sieci tego składa się z wielu warstw: wejściowej, wyjściowej oraz przynajmniej jednej warstwy ukrytej. Warstwy ukryte zbudowane są z modeli neuronów McCullocha-Pittsa. Tego typu model posiada wiele wejść oraz jedno wyjście. Każde z wejść ma przyporządkowaną stałą wagę, natomiast waga wyjścia jest obliczana przy pomocy funkcji aktywacji, która jest taka sama w obrębie warstwy.



Rysunek 12: Schemat neutronu McCullocha-Pittsa

Podczas analizy wykorzystaliśmy model regresyjny, zatem neurony będą liniowe. Oznacza to, że funkcja aktywacyjna wykorzystana w danej strukturze będzie liniowa. Wykorzystaliśmy domyślną funkcję proponowaną przez bibliotekę - relu, która ma postać:

$$f(x) = max(0, x)$$

Stworzony model składa się z 2 warstw ukrytych o 100 i 50 neuronach. Podczas analizy proces trenowania modelu składał się z 10 epok, ponieważ zwiększanie tej wartości nie powodowało poprawy w wynikach modelu.

11.2 Gradient Boosted Regression Tree

Algorytm podbijania gradientu polega na wykorzystaniu wielu słabych modeli (drzew decyzyjnych) w celu stworzenia modelu dokładnego, składającego się ze słabych modeli. Celem modelu jest minimalizacja funkcji straty poprzez iteracyjne zwiększanie ilości drzew. Waga poszczególnych drzew jest obliczana na podstawie minimalizacji błędu głównego modelu, tzw. silnego ucznia.

Podczas analizy ograniczamy głębokość drzewa do 10. Większe drzewa nie dawały lepszych rezultatów, mniejsze również.

11.3 LightGBM

LightGBM to biblioteka open-source, zawierająca implementację algorytmów wykorzystujących podbijanie gradientu (ang. gradient boosting). Osiąga dobre wyniki w kontekście dokładności oraz wykorzystania pamięci.Opiera się na drzewach decyzyjnych co pozwala osiągnąć większą wydajność. Wykorzystuje techniki:

- Gradient-based One Side Sampling - algorytm wykorzystuje instancje z większym gradientem oraz dodaje losowo część tych z mniejszym,
- Exclusive Feature Bundling Technique - algorytm grupujący predykaty i zastępujący grupy predykatów 1. Pozwala zredukować złożoność analizowanych danych.

Podczas budowania modelu wykorzystujemy drzewa z maksymalną ilością liści równą 50, oraz krokiem równym 0.03.

11.4 XGboost

XGBoost to biblioteka open-source, implementująca metody podbijania gradientu. (ang. gradient boosting). Algorytm GBRT z tej biblioteki może pracować równolegle, w przypadku naszego modelu wykorzystywane jest maksimum dostępnych wątków. Biblioteka pozwala w pełni wykorzystać zasoby sprzętowe, dzięki możliwości wykorzystania obliczeń rozproszonych podczas trenowania modelu. Algorytm wykorzystuje metodę Newtona-Raphsona zamiast metoda gradientu prostego. Model wykorzystuje dużo pamięci podczas trenowania głębokich drzew. Podczas analizy głębokość została ustawiona na 5. Ponadto w analizie wykorzystujemy wszystkie dane ze zbioru w każdej iteracji nauki modelu.

12 Porównanie modeli

Parametr	MLP	GBRT	LightGBM	XGBoost
Błąd średniokwadratowy	0.0087	0.0090	0.0062	0.0068

Wśród otrzymanych wyników najwydajniejszymi algorytmami okazały się te z bibliotek XGBoost oraz LightGBM. Prosty model MLP okazał się porównywalny do zdefiniowanego w sklearn.tree algorytmu GBRT.

12.1 Porównanie XGBoost i LightGBM

Porównując implementację 2 najwydajniejszych algorytmów warto wyszczególnić różnice:

- w algorytmie XGBoost drzewa rosną w głąb, podczas gdy drzewa LightGBM rozrastają się w szerokości,
- LightGBM pozwala grupować parametry, zmniejszając złożoność,
- trenowanie modelu LightGBM jest dużo szybsze niż modelu XGBoost,
- XGBoost ma znacząco większe wymagania sprzętowe
- XGBoost pozwala na wykorzystanie obliczeń rozproszonych,
- XGBoost jest bardziej rozwinięty i wspierany przez większą grupę użytkowników