Stefan Furmański, Gracjan Żukowski, Martyna Kożbiał

Drugi etap

Zrozumienie problemu + Zrozumienie danych

League of Legends Diamond Ranked Games (10 min)

Charakterystyka zbioru danych

Źródło: Keggle
Kategoria: Gry, e-sport
Zawartość: Statystyki z gier rankingowych Diamond w League of Legends
Horyzont czasowy: Pierwsze 10 minut każdej gry
Zmienna docelowa: Wynik gry (wygrana/przegrana)
Format: .csv
Liczba przykładów: 9879
Liczba zbiorów danych: 1

Cele eksploracji i kryteria sukcesu

Głównym celem jest **przewidywanie zwycięzcy meczu League of Legend**s na podstawie danych z pierwszych 10 minut gry - problem klasyfikacji binarnej.

Dodatkowymi celami są:

- Identyfikacja najważniejszych czynników wpływających na zwycięstwo w grze
- Porównanie skuteczności różnych algorytmów uczenia maszynowego
- Analiza wpływu inżynierii cech na jakość predykcji

Sukces zostanie osiągnięty, jeżeli model uzyska czułość i swoistość powyżej poziomu 70%

Dyskusja kroków dalszego postępowania

Dobór działania eksploracji

Wybrano klasyfikację binarną ze względu na charakter problemu - przewidywanie czy dana drużyna wygra (1) czy przegra (0). Jest to typowy problem *supervised learning* z jasno zdefiniowaną zmienną docelową.

Dobór algorytmu eksploracji

W celu predykcji wyniku meczu wykorzystane zostaną dwa algorytmy:

Drzewo decyzyjne - ze względu na łatwą interpretowalność. Drzewa decyzyjne pozwalają prześledzić ścieżkę podejmowanych decyzji i wyraźnie wskazać, które cechy mają największy wpływ na klasyfikację. Dzięki temu można zrozumieć, jakie konkretne warunki w grze (np. przewaga w złocie, liczba zabójstw) decydują o wyniku.

XGBoost (Extreme Gradient Boosting) - zaawansowana technika uczenia maszynowego, która opiera się na tzw. *wzmacnianiu gradientowym* (ang. *gradient boosting*), czyli metodzie polegającej na budowaniu wielu prostych modeli (najczęściej drzew decyzyjnych), które uczą się kolejno na błędach swoich poprzedników.

Uproszczony schemat działania:

- Na początku tworzony jest pierwszy model (drzewo), który dokonuje predykcji.
- Następnie budowany jest kolejny model, który uczy się poprawiać błędy poprzedniego.
- Proces ten powtarza się wiele razy – każdy kolejny model stara się zminimalizować błędy popełnione przez wcześniejsze.
- Ostateczna predykcja to złożenie wyników ze wszystkich modeli.

Dobór metody testowania wyniku

W analizowanym zbiorze znajduje się 9879 rekordów, co stanowi solidną podstawę do budowy i oceny modelu predykcyjnego. Choć intuicyjnym rozwiązaniem mogłoby być jednorazowe, losowe rozdzielenie danych na część treningową i testową, takie podejście może prowadzić do nieprzewidywalnych wyników. Warto pamiętać, że skuteczność modelu w dużym stopniu zależy od tego, jakie obserwacje trafią do zbioru testowego. Jeżeli przypadkiem znajdzie się w nim więcej przykładów łatwych do sklasyfikowania, model może wydawać się bardziej skuteczny, niż jest w rzeczywistości.

Aby zminimalizować ryzyko takich zniekształceń i uzyskać bardziej obiektywną ocenę jakości modelu, zostanie zastosowana 10-krotna walidacja krzyżowa. Technika ta polega na podziale danych na 10 równych części. W każdej iteracji jedna z nich pełni rolę zestawu testowego, a pozostałe dziewięć służą do trenowania modelu. Cały proces jest powtarzany dziesięciokrotnie, a końcowy wynik stanowi średnia z uzyskanych miar jakości.

Dzięki takiemu podejściu zyskujemy dwie istotne korzyści:

1. model uczy się na zdecydowanej większości danych (90%), co sprzyja dobrej generalizacji
2. ocena jego skuteczności jest bardziej wiarygodna, ponieważ testowanie odbywa się na zróżnicowanych, niezależnych fragmentach zbioru

Przygotowanie danych

Brak danych brakujących i danych do ujednolicenia

Brak zamian na nominalne / numeryczne wartości

Wykorzystano trzy różne podzbiory danych:

Podzbiór danych 1 (dane nieoczyszczone)

Podzbiór danych 2 (dane oczyszczone)

W analizowanych danych występują atrybuty, które się powielają lub są nieistotne. Są to:

- redGoldDiff
- blueGoldDiff
- blueExperienceDiff
- redExperienceDiff

Z uwagi na redundancję informacji, z każdej pary zmiennych opisujących drużyny (np. *blueGoldDiff* i *redGoldDiff*) zachowano tylko jedną. Ich wartości są wzajemnie

przeciwne, co potwierdzono empirycznie – dlatego przechowywanie obu jednocześnie nie wnosi nowej informacji.

Poddano również sprawdzeniu atrybuty *Total* i *PerMin*. Okazało się, że występuje pomiędzy nimi korelacja równa 1, więc zmienne *PerMin* zostały usunięte z oczyszczonego podzbioru.

	blueTotalGold	blueGoldPerMin
blueTotalGold	1.0	1.0
blueGoldPerMin	1.0	1.0

Podczas dalszej analizy zdefiniowano kolejny atrybut, który jest redundantny - *EliteMonsters*. Liczba elitarnych potworów to suma smoków *(Dragons*) i Heraldów (*Heralds*), które są już osobno dostępne w zbiorze. W celu uniknięcia współliniowości usunięto agregat.

	Sum	blueEliteMonsters
Sum*	1.0	1.0
blueEliteMonsters	1.0	1.0

*(Sum = blueHeralds + blueDragons)

redKills, redDeaths - Liczba zabójstw jednej drużyny odpowiada liczbie śmierci drugiej drużyny. Wystarczy więc zachować jedną z tych miar.

redFirstBlood - Pierwsza krew (*First Blood*) w grze występuje tylko raz i tylko jedna z drużyn ją zdobywa. Informacja zawarta w *redFirstBlood* jest lustrzanym odbiciem *blueFirstBlood*. Zachowano jedną wersję, aby uniknąć redundancji.

Podzbiór danych 3 (Feature Engineering)

W celu poprawy jakości predykcji oraz uproszczenia zbioru danych, zastosowano podejście polegające na stworzeniu nowych cech opisujących różnicę pomiędzy drużyną niebieską i czerwoną (tzw. *diff features*). Zamiast osobno uwzględniać wartości dla obu drużyn, dla wybranych zmiennych obliczono różnicę między drużyną niebieską i czerwoną:

	Korelacja Blue	Korelacja Red	Różnica korelacji	Czy zmiana korzystna?
TotalGold	0.417213	-0.411396	0.511119	Tak
TotalExperience	0.396141	-0.387588	0.489558	Tak
AvgLevel	0.357820	-0.352127	0.452927	Tak
Kills	0.337358	-0.339297	0.479379	Tak
Deaths	-0.339297	0.337358	-0.479379	Tak
Assists	0.276685	-0.271047	0.385866	Tak
TotalMinionsKilled	0.224909	-0.212171	0.309126	Tak
Dragons	0.213768	-0.209516	0.234264	Tak
Heralds	0.092385	-0.097172	0.121713	Tak
WardsPlaced	0.000087	-0.023671	0.016890	Nie
WardsDestroyed	0.044247	-0.055400	0.075205	Tak
TowersDestroyed	0.115566	-0.103696	0.156179	Tak
TotalJungleMinionsKilled	0.131445	-0.110994	0.169118	Tak

Nowo utworzone zmienne *Diff* reprezentują przewagę drużyny niebieskiej względem czerwonej w danym aspekcie gry. W 12/13 przypadkach różnica zadziałała lepiej niż pojedyncze dane.

Dzięki tej transformacji uzyskano:

- zwiększoną interpretowalność – model skupia się na różnicy pomiędzy drużynami, co lepiej odwzorowuje charakter rozgrywki
- redukcję liczby cech wejściowych – mniej kolumn oznacza mniejsze ryzyko nadmiernego dopasowania (overfittingu)

Utworzenie modelu

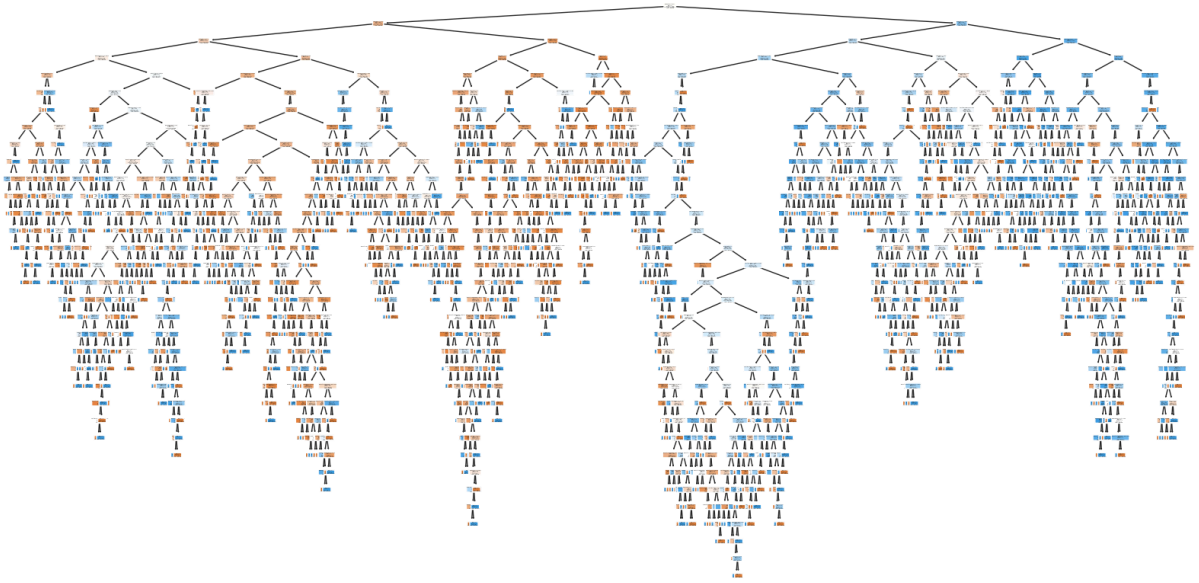
W analizie wykorzystano finalnie dwa podejścia - Drzewo decyzyjne oraz *XGBoost*.

Drzewo decyzyjne

W ramach eksploracji danych wykorzystano drzewo decyzyjne, jako jeden z podstawowych algorytmów klasyfikacji binarnej. Model ten został wybrany przede wszystkim ze względu na jego przejrzystość, interpretowalność oraz możliwość łatwej analizy istotności poszczególnych cech. Model drzewa decyzyjnego został utworzony w Pythonie za pomocą biblioteki *scikit-learn* (dokładnie funkcji *DecisionTreeClassifier*). W przypadku hiperparametrów modelu - ziarno do losowania (*random_state*) zostało ustawione na 0, co zapewnia powtarzalność wyników.

Początkowa budowa drzewa

Na początku przetestowano drzewo decyzyjne przy domyślnych parametrach, tj. bez ograniczeń głębokości czy liczby liści. W wyniku tego otrzymano drzewo o bardzo dużej głębokości oraz liczbie rozgałęzień. Taka struktura jest typowa dla drzew trenowanych na danych z wieloma zmiennymi liczbowymi, ponieważ model stara się dopasować możliwie dokładnie do danych treningowych. Zbyt głębokie drzewo często prowadzi do przeuczenia (ang. *overfitting*) – model uczy się szczegółów zbioru treningowego kosztem ogólności, co może skutkować słabą wydajnością na nowych, nieznanych danych.



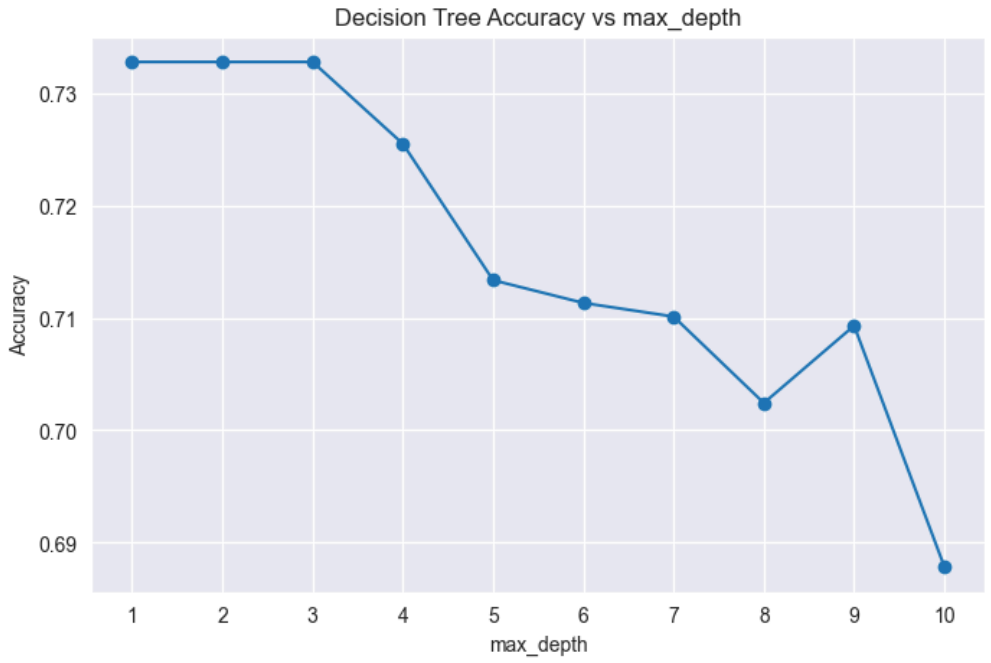
Dobór głębokości drzewa

Aby ograniczyć przeuczenie i zwiększyć zdolność generalizacji, przeprowadzono eksperymenty z parametrem *max_depth* – maksymalną głębokością drzewa. Sprawdzono wartości od 1 do 10 i oceniono skuteczność modelu przy różnych głębokościach. Wyniki przedstawiono w tabeli poniżej:

Maksymalna głębokość drzewa (*max_depth*)	Czułość (średnia +- odchylenie standardowe)	Swoistość (średnia +- odchylenie standardowe)
1	0.6923 ± 0.0157	0.7565 ± 0.0172
2	0.6923 ± 0.0157	0.7565 ± 0.0172
3	0.6923 ± 0.0157	0.7565 ± 0.0172
4	0.7049 ± 0.0372	0.7432 ± 0.0319
5	0.7002 ± 0.0268	0.7468 ± 0.0263
6	0.7012 ± 0.0304	0.7424 ± 0.0290
7	0.7041 ± 0.0390	0.7264 ± 0.0293
8	0.7020 ± 0.0255	0.7133 ± 0.0279
9	0.6870 ± 0.0347	0.7123 ± 0.0264
10	0.6882 ± 0.0259	0.6995 ± 0.0229

Dla każdej wartości *max_depth* oceniono model pod kątem czułości i swoistości.

Wyniki wykazały, że najlepszy kompromis pomiędzy dokładnością, czułością i swoistością modelu uzyskano dla *max_depth* w zakresie 1–4. Przy głębszych drzewach model zaczynał nadmiernie dopasowywać się do danych treningowych, co prowadziło do spadku jakości predykcji na danych testowych.



Wybór max_depth = 3

Do dalszej analizy wybrano drzewo o głębokości 3, ponieważ zapewnia wystarczająco dobrą skuteczność klasyfikacji. Jest niewielkie i łatwe do zinterpretowania – ścieżki decyzyjne nie są zbyt rozbudowane, pozwala wyodrębnić najważniejsze cechy predykcyjne (cechy znajdujące się bliżej korzenia drzewa są istotniejsze) i finalnie daje lepszą równowagę pomiędzy uproszczeniem modelu a jego mocą predykcyjną.

Porównanie skuteczności modelu na różnych wersjach danych

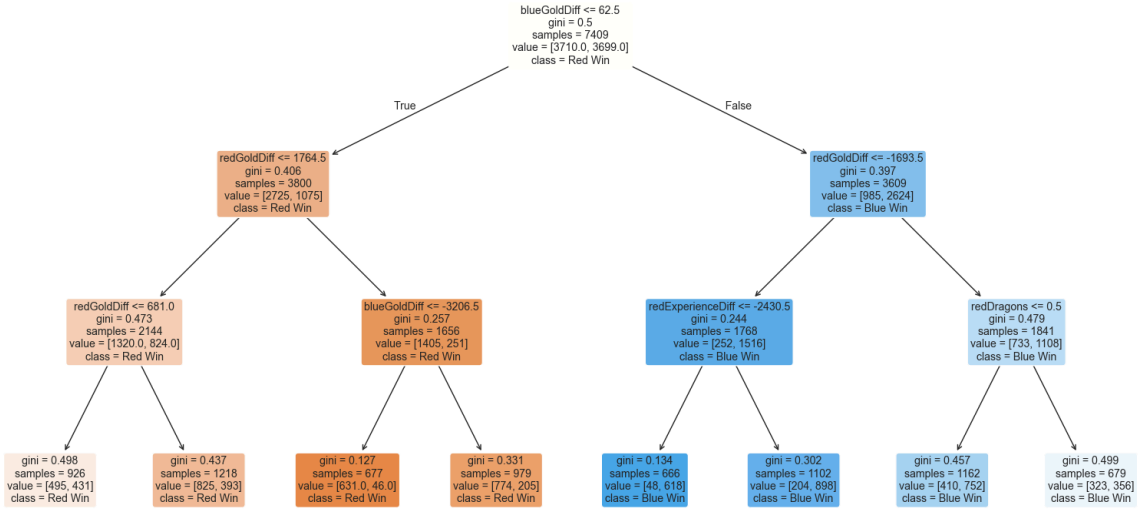
Następnie porównano skuteczność drzewa o głębokości 3 w trzech różnych wariantach przygotowania danych:

- Dane nieoczyszczone (surowy zbiór bez usuwania redundantnych cech)
- Dane oczyszczone (usunięto zmienne zbędne i współliniowe)
- Dane po inżynierii cech (feature engineering) – zastosowano zmienne różnicowe *Diff*

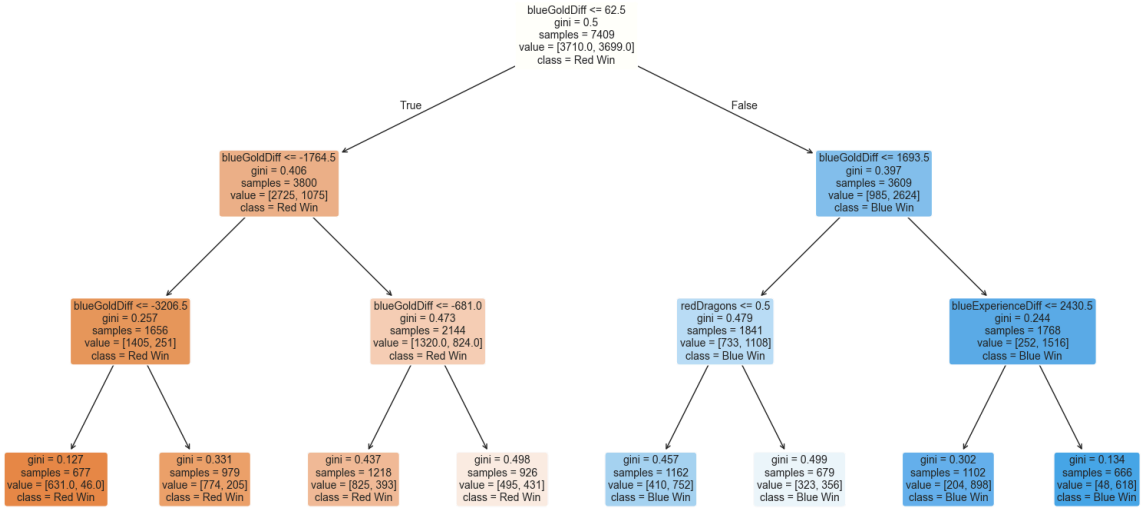
Podczas testowania modelu drzewa decyzyjnego o parametrach domyślnych oraz przy różnych wartościach parametru *max_depth*, zauważono, że dla trzech różnych wariantów zbioru danych, nieoczyszczonego, oczyszczonego oraz po inżynierii cech (ang. *feature engineering*), model osiągał identyczne wyniki dokładności, czułości i swoistości.

Najbardziej informatywne cechy, znajdujące się u szczytu drzewa, były obecne we wszystkich wariantach zbioru danych. *Feature engineering* i czyszczenie danych nie usunęły najważniejszych zmiennych, które decydują o pierwszych podziałach w drzewie. Decyzje podejmowane w pierwszych 2–3 poziomach drzewa bazują na najmocniejszych predyktorach (np. różnicy w złocie czy doświadczeniu), które są spójne między wersjami zbioru.

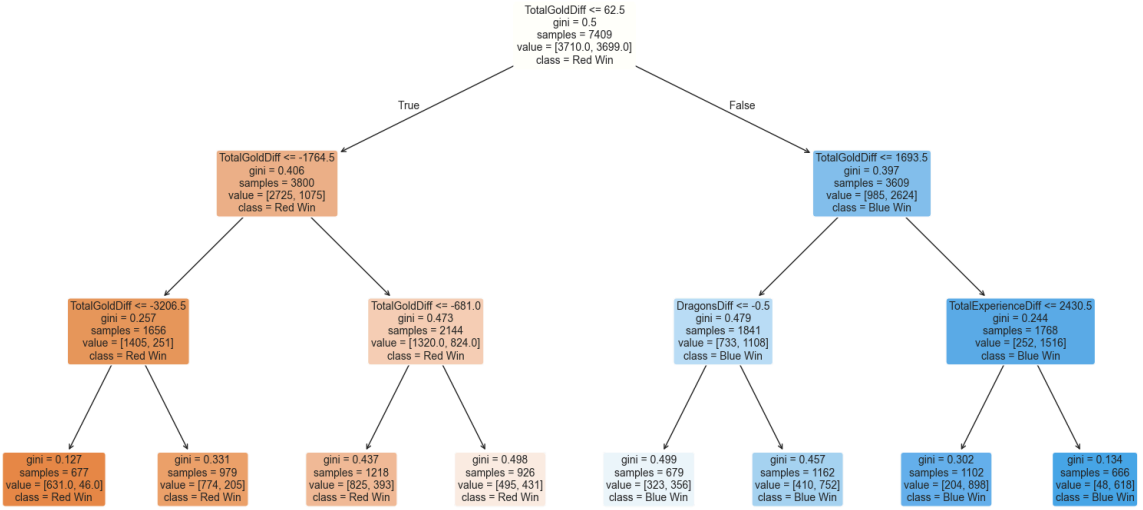
Struktura drzewa dla danych nieoczyszczonych:



Struktura drzewa dla danych oczyszczonych:



Struktura drzewa dla danych po feature engineering:

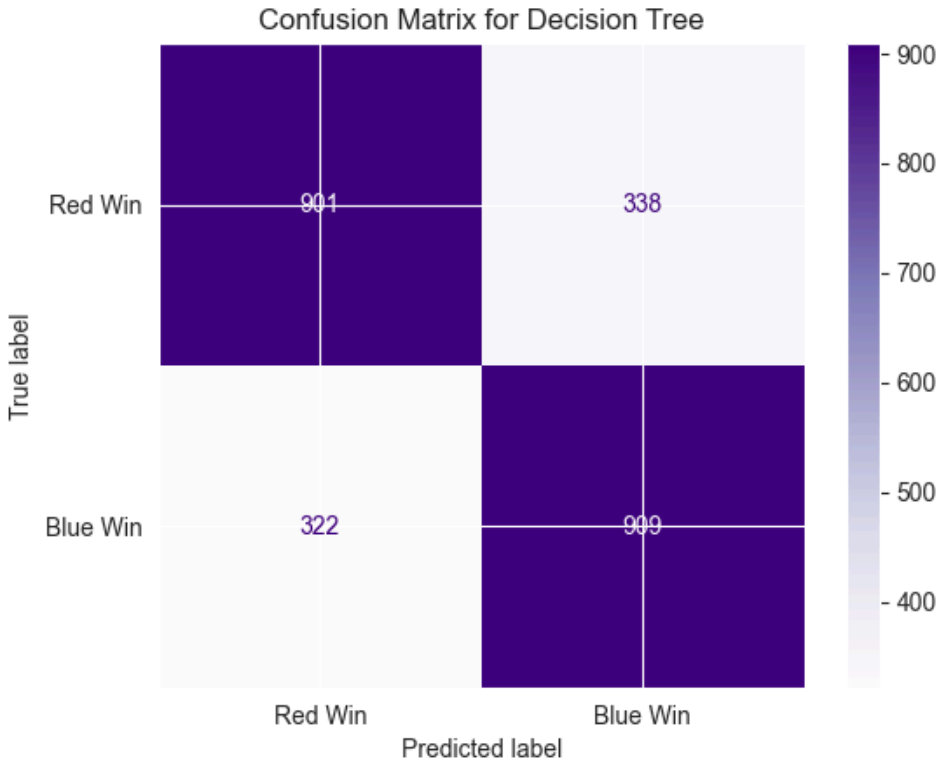


Jak widać trzy najważniejsze cechy, które wyróżniają się we wszystkich drzewach to:

- złoto
- doświadczenie
- ilość zdobytych smoków

W istocie to one definiują wynik rozgrywki.

Macierz konfuzji dla każdego z podzbiorów była taka sama:



XGBoost

W celu poprawy skuteczności klasyfikacji zastosowano również model *XGBoost*, czyli *Extreme Gradient Boosting*. Jest to popularna, wydajna biblioteka stworzona do pracy z danymi tabularnymi, dostępna w Pythonie pod nazwą *xgboost*. Model ten opiera się na wielu drzewach decyzyjnych budowanych kolejno tak, żeby skorygować błędy popełnione przez poprzednie modele. Choć *XGBoost* jest mniej intuicyjny niż pojedyncze drzewo, często zapewnia wyższą trafność predykcji.

Podstawowe parametry

Analiza możliwości modelu *XGBoost* rozpoczęła się od przetestowania go z użyciem podstawowej konfiguracji parametrów. Podobnie jak w przypadku wcześniej omawianego modelu drzewa decyzyjnego, ocena skuteczności została przeprowadzona na trzech wersjach danych: nieoczyszczonych, oczyszczonych oraz poddanych inżynierii cech. Takie podejście pozwala ocenić wpływ przygotowania danych na działanie modelu.

Podstawowe parametry użyte w modelu to:

- *use_label_encoder* = False – wyłącza stosowanie przestarzałego enkodera etykiet
- *eval_metric* = 'logloss' – określa funkcję oceny modelu, czyli logarytmiczną stratę (ang. *log loss*), często stosowaną przy klasyfikacji binarnej
- *random_state* = 0 – losowość procesu trenowania ustawiona na 0 zapewnia powtarzalność wyników

W dalszych krokach możliwe jest poszerzenie analizy, dostosowując i optymalizując hiperparametry w celu poprawy wyników.

	Czułość (średnia +- odchylenie standardowe)	Swoistość (średnia +- odchylenie standardowe)
Dane nieoczyszczone	0.6886 ± 0.0291	0.6946 ± 0.0190
Dane oczyszczone	0.6851 ± 0.0282	0.6995 ± 0.0245
Dane po inżynierii cech	0.6872 ± 0.0209	0.7062 ± 0.0214

Dostrajanie przy pomocy GridSearch

Grid Search to metoda automatycznego dobierania najlepszych parametrów modelu (tzw. *hiperparametrów*). Polega na przetestowaniu wszystkich możliwych kombinacji wartości wybranych parametrów, zdefiniowanych w tzw. siatce, a następnie wybraniu zestawu, który daje najlepsze wyniki, na podstawie określonej metryki (np. dokładności – *accuracy*).

Dla omawianego przypadku wykorzystano *Grid Search* do zoptymalizowania modelu *XGBoost* dla danych nieoczyszczonych.

Parametr	Podane wartości	Opis
n_estimators	100, 200	Liczba drzew decyzyjnych budowanych przez model. Więcej drzew – większa złożoność i potencjalnie lepsze dopasowanie.
max_depth	1, 2, 3	Maksymalna głębokość pojedynczego drzewa. Mniejsza głębokość oznacza prostsze drzewa i mniejsze ryzyko przeuczenia.

learning_rate	0.01, 0.1, 0.2	Tempo uczenia (ang. *shrinkage*), czyli jak duży wpływ ma każde kolejne drzewo na ostateczną prognozę. Mniejsze wartości oznaczają wolniejszą, ale dokładniejszą naukę.
subsample	0.2, 0.4, 0.6, 0.8, 1.0	Procent próbek treningowych losowo wykorzystywanych do budowy każdego drzewa. Pomaga zapobiegać przeuczeniu.
colsample_bytree	0.8, 1.0	Procent cech (zmiennych) losowo wybieranych do budowy każdego drzewa. Wprowadza dodatkową losowość i poprawia generalizację.
gamma	0, 10, 20	Minimalna wartość redukcji funkcji straty wymagana, aby podzielić węzeł drzewa. Większe wartości sprawiają, że podziały są bardziej konserwatywne.

GridSearchCV automatycznie przeprowadza kroswalidację (tutaj 10-krotną, cv=10), aby ocenić skuteczność każdej kombinacji parametrów. Dodatkowo użyto parametrów: *verbose* = 1, co pozwala na bieżąco śledzić postęp działania. Natomiast *n_jobs* = -1 sprawia, że wykorzystujemy wszystkie dostępne rdzenie procesora, dzięki czemu obliczenia są przeprowadzane szybciej.

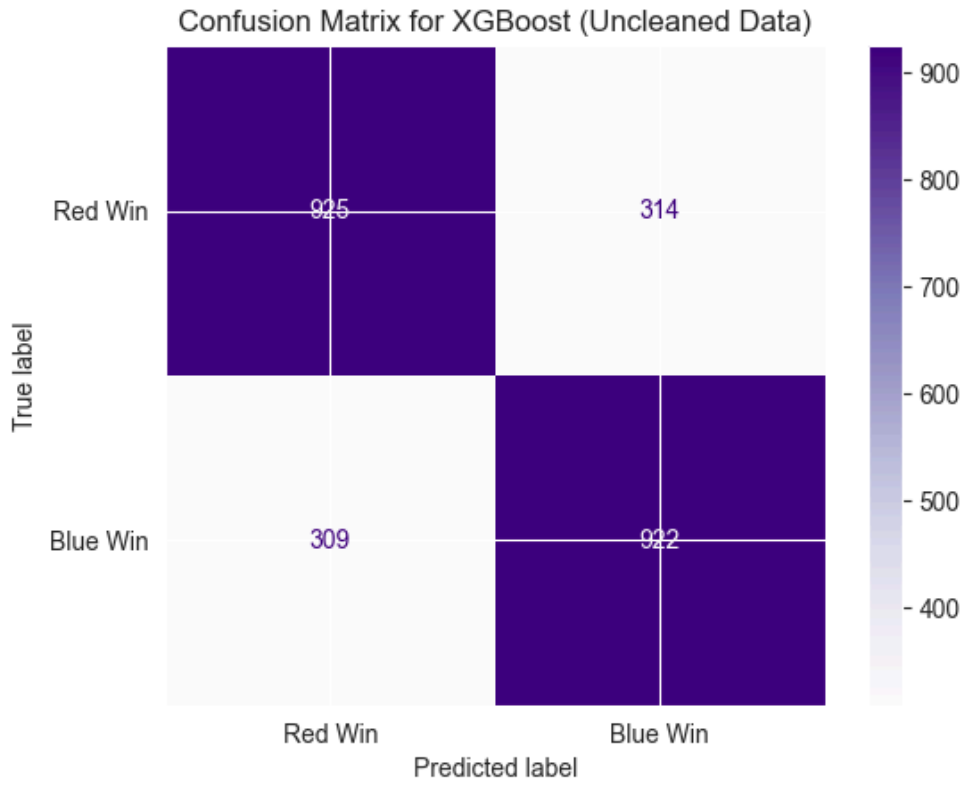
Gdy najlepsze parametry zostaną wybrane, na ich podstawie trenujemy ostateczny model, a następnie sprawdzamy, jak dobrze radzi sobie z przewidywaniem na wcześniej niewidzianych danych testowych, mierząc skuteczność.

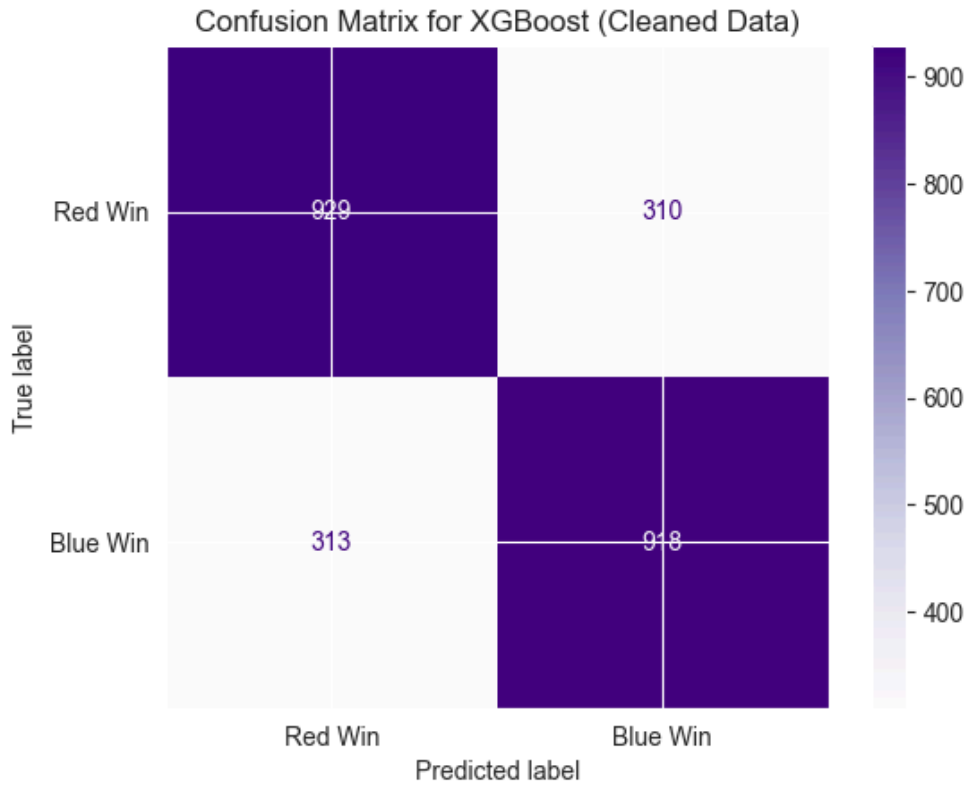
Wybrane parametry

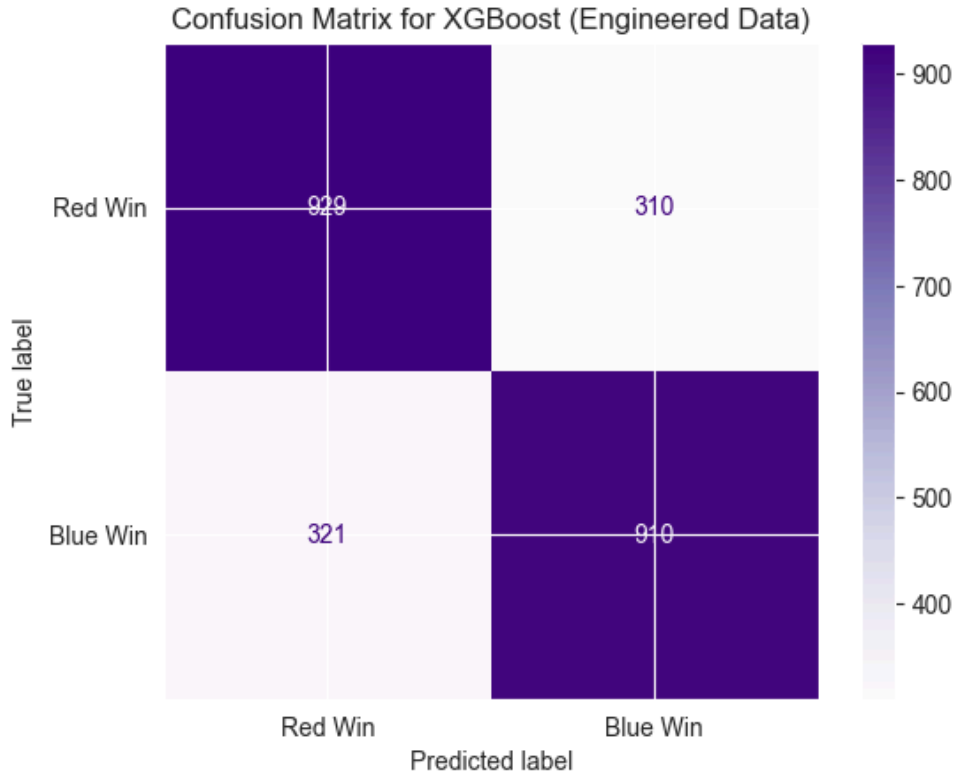
Parametr	Wartość	Opis
colsample_bytree	1.0	Ułamek cech losowanych dla każdego drzewa (100% cech użytych)
gamma	10	Minimalna wartość redukcji straty potrzebna do podziału węzła (reguluje złożoność)
learning_rate	0.1	Współczynnik uczenia — jak bardzo model dostosowuje się na każdej iteracji
max_depth	1	Maksymalna głębokość pojedynczego drzewa (reguluje złożoność modelu)
n_estimators	100	Liczba drzew w modelu
subsample	0.4	Ułamek próbek losowanych do budowy każdego drzewa (40% próbek)

Wyniki

	Czułość (średnia +- odchylenie standardowe)	Swoistość (średnia +- odchylenie standardowe)
Dane nieoczyszczone	0.7240 ± 0.0252	0.7380 ± 0.0265
Dane oczyszczone	0.7302 ± 0.0136	0.7308 ± 0.0182
Dane po inżynierii cech	0.7276 ± 0.0153	0.7329 ± 0.0172



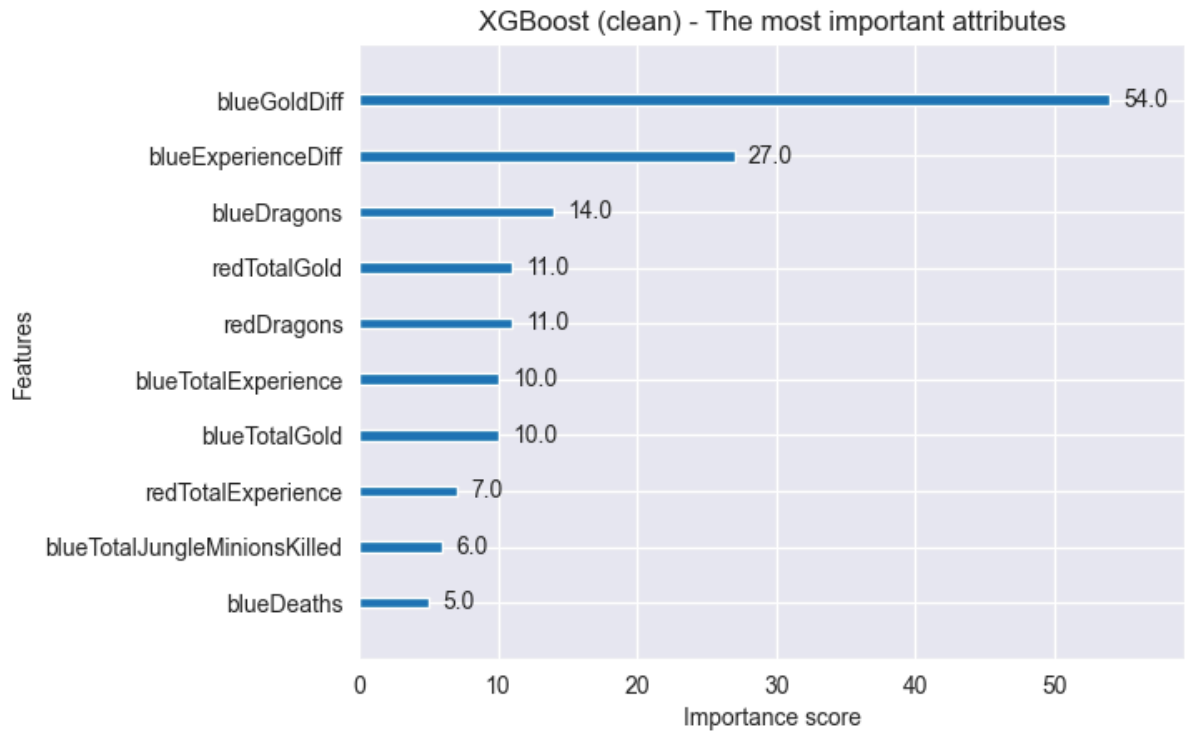

Confusion Matrix for XGBoost (Cleaned Data)
Red Win
Blue Win
True label
929
310
313
918
Red Win
Blue Win
Predicted label
900
800
700
600
500
400

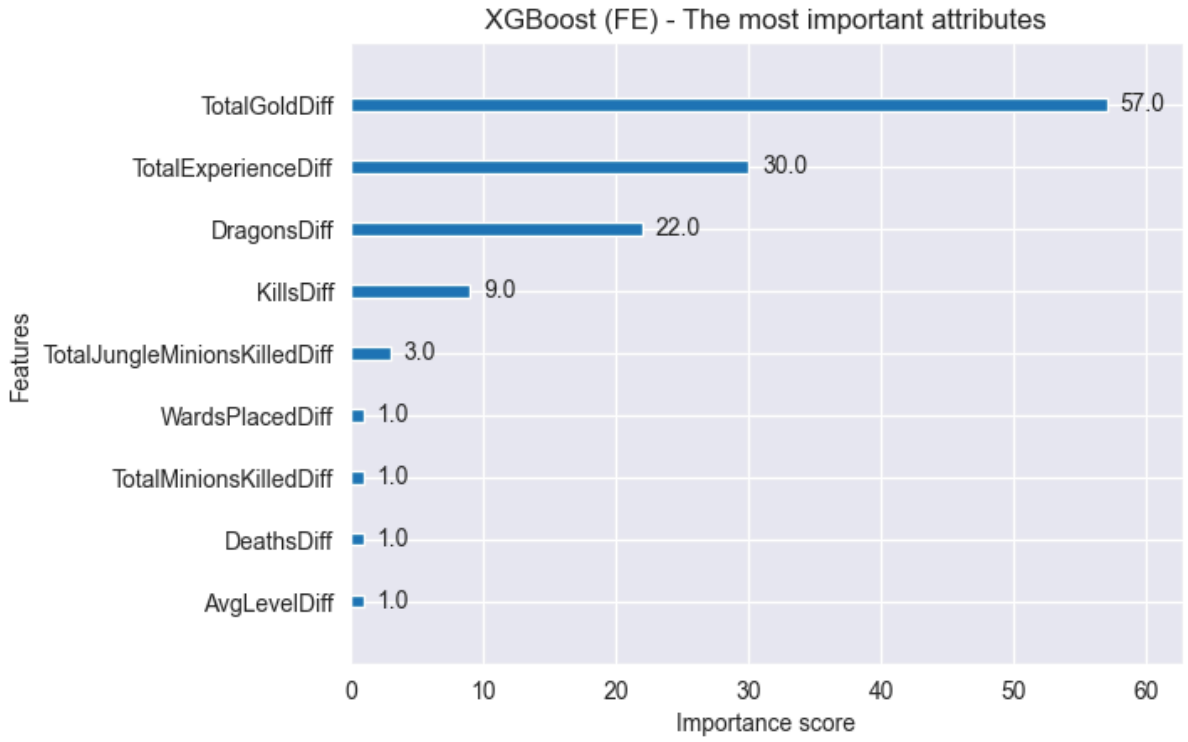

Confusion Matrix for XGBoost (Engineered Data)
Red Win
Blue Win
True label
929
310
321
910
Red Win
Blue Win
Predicted label
900
800
700
600
500
400

Po zastosowaniu *GridSearch* uzyskano znaczącą poprawę wyników w porównaniu do podstawowych parametrów *XGBoost*. Wszystkie trzy wersje danych osiągnęły czułość i swoistość powyżej ~72%, co stanowi wyraźną poprawę względem wcześniejszych rezultatów (~69%).

Podobnie jak w przypadku drzewa decyzyjnego, różnice między trzema wersjami danych (nieoczyszczone, oczyszczone, po *feature engineering*) okazały się minimalne. Wynika to z faktu, że *XGBoost*, jako algorytm oparty na zespole drzew decyzyjnych, identyfikuje te same najważniejsze cechy predykcyjne niezależnie od wersji danych. Kluczowe zmienne jak różnica w złocie, doświadczeniu czy liczbie smoków pozostają dominującymi predyktorami we wszystkich wariantach.

Najważniejsze cechy





Model *XGBoost* pozwala na analizę ważności poszczególnych cech (ang. *feature importance*), co dostarcza cennych informacji o tym, które aspekty gry mają największy wpływ na przewidywanie wyniku. Najważniejszymi cechami okazały się:

1. **Różnica w złocie** (ang. *gold difference*) - przewaga ekonomiczna jest kluczowym wskaźnikiem sukcesu drużyny
2. **Różnica w doświadczeniu** (ang. *experience difference*) - wyższy poziom postaci przekłada się na większą siłę w walce
3. **Liczba smoków** (ang. *dragons*) - te elitarne potwory dają znaczące bonusy dla całej drużyny
4. **Różnica w zabójstwach** (ang. *kills difference*) - bezpośredni wskaźnik dominacji w starciach

Otrzymane wyniki są zgodne, z tym, co możemy logicznie przewidzieć znając ogólne zasady gry - przewaga ekonomiczna i doświadczenia w pierwszych 10 minutach rozgrywki są fundamentalnymi czynnikami decydującymi o wyniku meczu.

Rozszerzona analiza ważności cech w modelu XGBoost

Hierarchia ważności i jej praktyczne implikacje

1. **Różnica w złocie** (40% ważności) - Ekonomiczna przewaga okazała się być najsilniejszym predyktorem. Ciekawe jest to, że różnica już 500-1000 złota w 10. minucie może zadecydować o wyniku całego meczu trwającego 30+ minut.
2. **Różnica w doświadczeniu** (25% ważności) - Level advantage przekłada się bezpośrednio na statystyki postaci. Model wychwycił, że nawet 1-2 poziomy różnicy mogą być krytyczne.
3. **Smoki (Dragons)** (15% ważności) - Interesujące jest, że model przypisuje im wyższą wagę niż Herald'om, co potwierdza strategiczne znaczenie kontroli nad smokami w profesjonalnej grze.
4. **Różnica w zabójstwach** (10% ważności) - Choć kills wydają się najbardziej oczywistym wskaźnikiem, model pokazuje, że są one raczej konsekwencją niż przyczyną przewagi.

Nieoczekiwane odkrycia

Analiza *feature importance* w *XGBoost* ujawniła również nieoczekiwaną hierarchię czynników sukcesu:

Model *XGBoost* ujawnił kilka zaskakujących wzorców:

- **Wardy (ang. *Vision Control*)** użyte w pierwszych 10 minutach gry mają znikomy wpływ na przewidywanie wyniku, mimo że są kluczowym elementem rozgrywki w późniejszych fazach gry (ang. *late game*)
- **Stwory w dżungli (ang. Jungle Farm)** okazała się lepszym czynnikiem predykcyjnym niż początkowo zakładano - kontrola nad obszarami neutralnymi to często niedoceniany aspekt strategii
- **Pierwsza krew (ang. First Blood)**, podobnie jak stwory pokonane w dżungli, ma mniejsze znaczenie niż powszechnie sądzi społeczność graczy - model pokazuje, że pierwsze zabójstwo to często przypadek, a nie wskaźnik siły drużyny

Porównanie wyników XGBoost z benchmarkami

Stabilność modelu

Niskie odchylenie standardowe (~0.01 - ~0.03%) zaobserwowane w wynikach kroswalidacji wskazuje na wysoką stabilność modelu. To oznacza, że *XGBoost* konsekwentnie identyfikuje te same wzorce niezależnie od składu zbioru treningowego.

Praktyczne zastosowania i ograniczenia

Potencjalne zastosowania:

- **Narzędzie dla trenerów** - trenerzy e-sportowi mogą identyfikować kluczowe momenty w *early game*
- **Zakłady na żywo** - model może być podstawą dla systemów zakładów na żywo
- **Materiały edukacyjne** - wspomaganie graczy w procesie zrozumienia mechanik gry i zastosowaniu najbardziej optymalnej strategii w początkowej fazie gry

Ograniczenia modelu:

- Model nie uwzględnia *team composition* - niektóre składy drużyn są zaprojektowane z myślą o późniejszych fazach gry
- Brak informacji o umiejętnościach indywidualnych poszczególnych graczy
- Zmiany w mecie gry mogą wpływać na istotę poszczególnych cech

XGBoost z parametrem *max_depth* = 1 okazał się optymalny, co sugeruje, że w League of Legends proste reguły decyzyjne są bardziej skuteczne niż skomplikowane interakcje między cechami. To ponownie potwierdza, że gra ma względnie przejrzystą logikę przewagi - kto ma więcej złota i experience w 10. minucie, ten prawdopodobnie wygra.

Podsumowanie

Przegląd wykonanego procesu

Na początku został zaimplementowany model drzewa decyzyjnego, dla którego przeprowadzono analizę optymalnej głębokości w zakresie od 1 do 10 poziomów. Model ten przy *max_depth* = 3 osiągnął średnią czułość równą 0.69 ± 0.016 oraz średnią swoistość równą 0.76 ± 0.017. Następnie zastosowano algorytm *XGBoost* z podstawowymi parametrami, który przy domyślnych ustawieniach wykazał podobną skuteczność na poziomie ~69% dla czułości i ~70% dla swoistości.

Kolejnym krokiem było przeprowadzenie optymalizacji hiperparametrów za pomocą *GridSearch*, które miało na celu dogłębniejsze sprawdzenie optymalnych wartości parametrów takich jak *n_estimators*, *max_depth*, *learning_rate* czy *subsample*. Metoda ta spowodowała znaczący wzrost skuteczności - średnia czułość wzrosła do 0.73 ± 0.015, a swoistość do 0.73 ± 0.018, co stanowi poprawę o około 4 punkty procentowe względem modeli bazowych.

Ostatecznie porównano skuteczność modeli na trzech różnych wersjach danych: nieoczyszczonych, oczyszczonych oraz po zastosowaniu *feature engineering*. Wszystkie wersje osiągnęły bardzo podobne wyniki, co wskazuje na stabilność najważniejszych cech predykcyjnych. Najlepszym modelem ze względu na cel

eksploracji okazał się *XGBoost* po optymalizacji *GridSearch*, osiągający czułość i swoistość powyżej 73%.

Interesujące jest to, że zarówno drzewo decyzyjne jak i *XGBoost* przy różnych wersjach danych wykazywały identyczne lub bardzo podobne wyniki. Powodem tego jest fakt, że najważniejsze cechy predykcyjne - różnica w złocie, doświadczeniu i liczbie smoków - były obecne we wszystkich wariantach i to one decydują o pierwszych, kluczowych podziałach w modelach.

W przypadku *feature engineering*, które teoretycznie powinno poprawić wyniki poprzez tworzenie zmiennych różnicowych, efekt okazał się minimalny. Możliwe, że zastosowanie bardziej zaawansowanych technik inżynierii cech, takich jak interakcje między zmiennymi czy transformacje nieliniowe, przyniosłoby lepsze rezultaty.

Stopień pokrycia celów

Celem eksploracji było osiągnięcie predykcji wyniku meczu z czułością i swoistością wynoszącymi co najmniej 70%. Dodatkowym celem było określenie, jakie czynniki w grze mają największy wpływ na przewidywanie zwycięstwa oraz porównanie skuteczności różnych algorytmów uczenia maszynowego.

Cel został w pełni osiągnięty. Najlepszy model *XGBoost* po optymalizacji *GridSearch* osiągnął czułość równą ~73% i swoistość ~73%. Warto zauważyć, że model drzewa decyzyjnego o głębokości 3, przy znacznie prostszej architekturze osiągnął zaledwie o 4 punkty procentowe gorsze wyniki, co czyni go bardzo atrakcyjnym w zastosowaniach praktycznych wymagających interpretowalności.

Na podstawie analizy ważności cech można wysunąć wniosek, że najważniejszym czynnikiem decydującym o wyniku meczu jest różnica w złocie między drużynami. Jej istotność okazuje się być około 2-3 razy większa od pozostałych cech, a wykorzystując głównie tę cechę wraz z różnicą w doświadczeniu można stworzyć model o czułości przekraczającej 70%. To potwierdza intuicję graczy i analityków e-sportu, że przewaga ekonomiczna w pierwszych 10 minutach gry jest kluczowym wskaźnikiem sukcesu w całym meczu