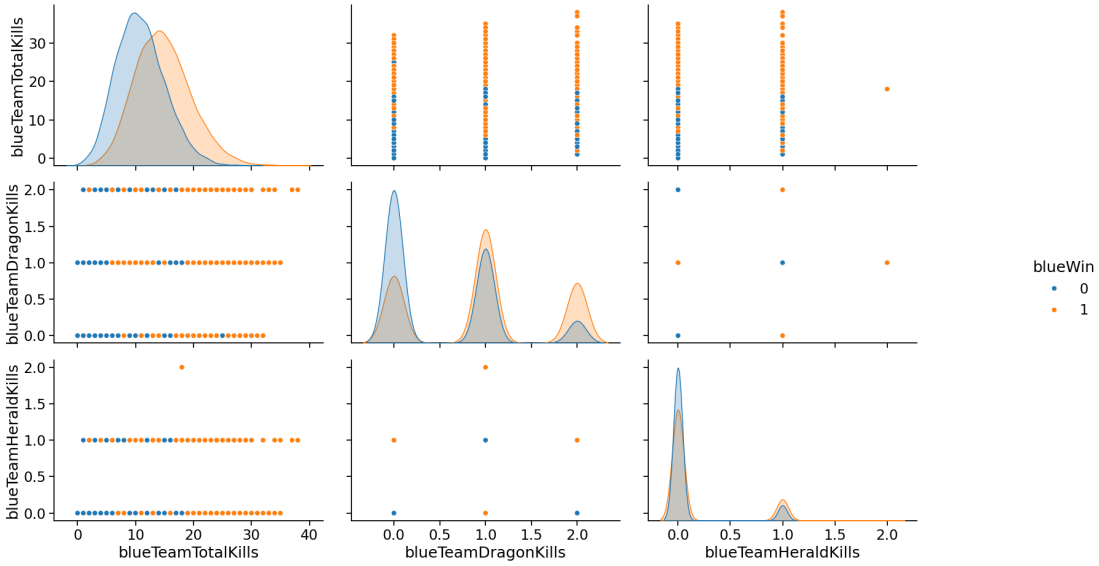
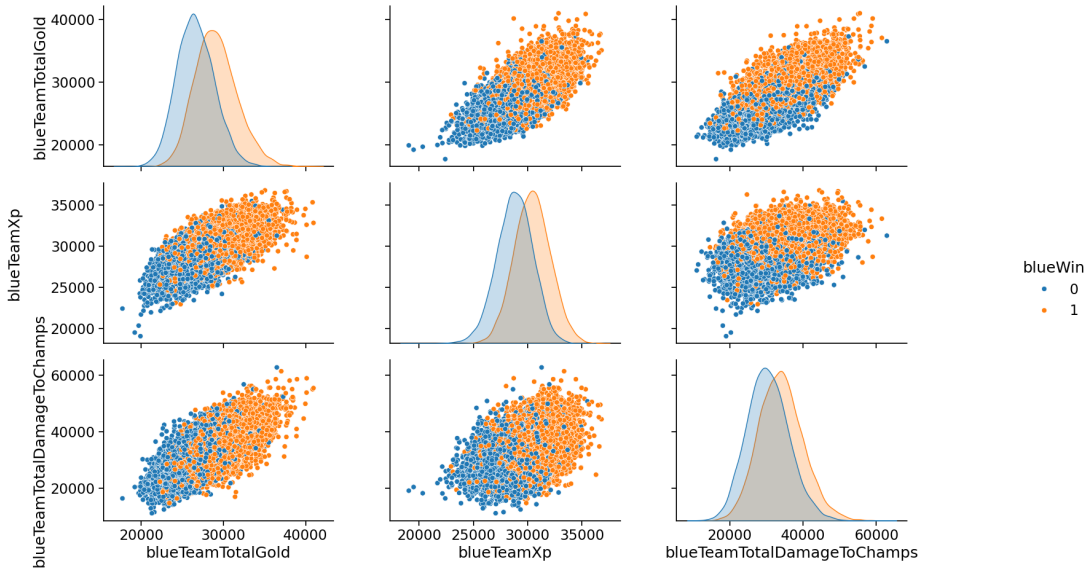
Paweł Zawistowski 16.05.2024

Inteligencja Obliczeniowa
Uczenie maszynowe - League Of Legends

1 Wstęp

Zbiór danych użyty do klasyfikacji zawiera różne statystyki z rozgrywek rankingowych w League of Legends na serwerze EUW, obejmujących przedziały rangowe od Szmaragdu do Diamentu. Dane zostały zebrane z różnych meczów do piętnastej minuty gry, co pozwala na analizę kluczowych aspektów rozgrywki w tym okresie czasowym. Wśród zebranych informacji znajdują się liczba zabójstw, kontrola obszarów widoczności *(control wardów)*, zdobyte złoto oraz inne istotne wskaźniki, które mogą wpływać na wynik meczu. Analiza tych danych może dostarczyć cennych wglądów w strategie, taktyki oraz czynniki determinujące sukces w grze League of Legends. Baza danych została pobrana z serwisu **Kaggle**.





2 Preprocessing

Nie było tego zawiele, aczkolwiek pojawił się błąd, który trzeba było usunąć. W mojej bazie danych zdarzały się rozgrywki, w której dana drużyna utraciła ponad 15 tzw. *plate'ów*, gdzie jest to niemożliwe, ponieważ maksymalna ilość jaką można zniszczyć to 15. Podzieliłem dane na cechy i etykiety, a następnie kolejny raz podzieliłem zbiory, ale już na zbiory treningowe oraz testowe. Na koniec zastosowałem StandardScaler do znormalizowania cech treningowych i testowych.

```
62  data = pd.read_csv("match_data_v5.csv")
63  remove1 = data[data['blueTeamTurretPlatesDestroyed'] > 15].index
64  data.drop(remove1, inplace=True)
65  remove2 = data[data['redTeamTurretPlatesDestroyed'] > 15].index
66  data.drop(remove2, inplace=True)
67  print(data.shape)
68  X = data.drop(columns=["matchId", "blueWin"])
69  y = data["blueWin"]
70  X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)
71  scaler = StandardScaler()
72  X_train_normalized = scaler.fit_transform(X_train)
73  X_test_normalized = scaler.transform(X_test)
```

3 Klasyfikacja

W celu skutecznego przewidywania wyników meczów League of Legends, zastosowałem różne klasyfikatory oraz przeprowadziłem analizę ich wyników.

3.1 Random Forest Classifier

Klasyfikator Random Forest jest algorytmem opartym na zasadzie drzew decyzyjnych. Po przeprowadzeniu treningu na danych z użyciem domyślnych parametrów oraz wykorzystaniu techniki oceny wydajności klasyfikatora, uzyskałem następujące wyniki:

Dokładność klasyfikacji: **75.32%**
Pole pod krzywą ROC: **0.84**
Wynik czułości: **75.53%**

Macierz błędu:

$$\begin{bmatrix} 1852 & 617 \\ 635 & 1741 \end{bmatrix}$$

3.2 Support Vector Machine (SVM)

Klasyfikator SVM jest metodą klasyfikacji opartą na konstrukcji hiperpłaszczyzn separujących dane wejściowe na podstawie etykiet klas. Po przeprowadzeniu treningu z domyślnymi parametrami, uzyskałem następujące wyniki:

Dokładność klasyfikacji: **75.17%**
Pole pod krzywą ROC: **0.74**
Wynik czułości: **74.51%**

Macierz błędu:

$$\begin{bmatrix} 1850 & 619 \\ 631 & 1745 \end{bmatrix}$$

3.3 KNeighborsClassifier

Algorytm KNeighborsClassifier to metoda klasyfikacji oparta na znalezieniu najbliższych sąsiadów dla każdego punktu danych. Wypróbowałem różne wartości parametru liczby sąsiadów (3, 5, 11), z czego 11 dawała najlepsze wyniki:

Dokładność klasyfikacji: **73.38%**
Pole pod krzywą ROC: **0.72**
Wynik czułości: **72.51%**

Macierz błędu:

$$\begin{bmatrix} 1766 & 613 \\ 648 & 1710 \end{bmatrix}$$

3.4 MLPClassifier

Klasyfikator MLPClassifier bazuje na sieciach neuronowych typu Multi-Layer Perceptron (MLP). Próbowałem różne warianty, z różnymi funkcjami aktywacji oraz ukrytymi warstwawmi, ale najbardziej optymalne wyniki wychodziły mi gdy zastosowałem warstwy ukryte o rozmiarach (6,3) oraz funkcje aktywacji **relu**.

Dokładność klasyfikacji: **73.78%**
Pole pod krzywą ROC: **0.74**
Wynik czułości: **79.42%**

Macierz błędu:

$$\begin{bmatrix} 1688 & 781 \\ 489 & 1887 \end{bmatrix}$$

3.5 Gaussian Naive Bayes

Algorytm Gaussian Naive Bayes jest modelem probabilistycznym, który zakłada, że cechy są wzajemnie niezależne. Otrzymane wyniki po treningu to:

Dokładność klasyfikacji: **74.15%**
Pole pod krzywą ROC: **0.74**
Wynik czułości: **73.10%**

Macierz błędu:

$$\begin{bmatrix} 1856 & 613 \\ 639 & 1737 \end{bmatrix}$$

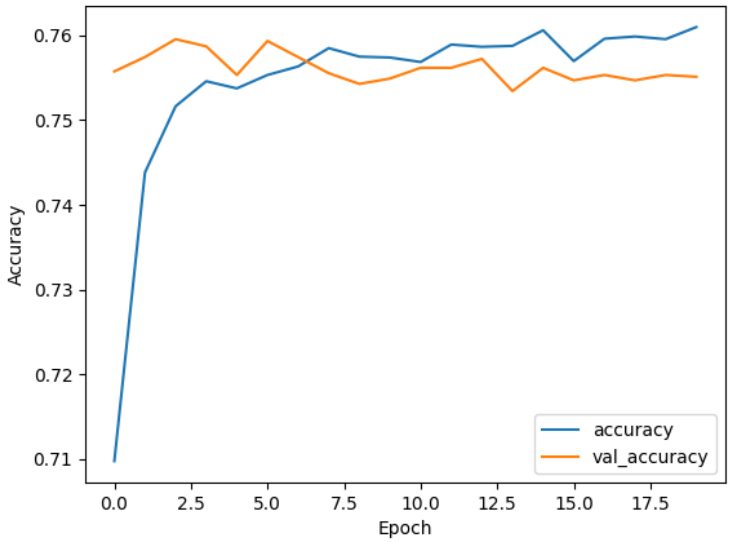
3.6 Własny Model z wykorzystaniem Tensorflow

W ramach eksperymentu stworzyłem również własny model klasyfikacyjny. Pierwsza warstwa zawierała 64 neurony z funkcją ReLu. Następnie zastosowałem warstwę Dropout w celu regularyzacji modelu i zapobieganiu nadmiernemu dopasowaniu. Kolejna warstwa miała 32 neurony z funkcją aktywacji ReLU oraz regularyzacją L2 o współczynniku 0.001. Na końcu model zawierał warstwę wyjściową z jednym neuronem i funkcją aktywacji sigmoidalną. Model został skompilowany przy użyciu optymalizatora Adam oraz funkcji straty binaryCrossentropy, a jako metrykę oceny wykorzystano dokładność (accuracy). Podczas treningu modelu przeprowadzono 50 epok z rozmiarem batcha równym 32. Dodatkowo, zaimplementowałem mechanizm wczesnego zatrzymywania (Early Stopping). Tutaj również probowałem różne warianty, czy chociażby zmiana warstw, ilość neuronów czy zmiana epok, ale ten przypadek dał mi najlepsze wyniki:

Dokładność klasyfikacji: **75.74%**
Pole pod krzywą ROC: **0.85**
Wynik czułości: **74.15%**

Macierz błędu:

$$\begin{bmatrix} 1811 & 568 \\ 593 & 1765 \end{bmatrix}$$



Rysunek 1: Wykres ukazujący zmianę dokładności modelu w czasie trenowania

```
model = tf.keras.Sequential([
        tf.keras.layers.Dense( units: 64, activation='relu', input_shape=(X_train.shape[1],)),
        tf.keras.layers.Dropout(0.5),  # Dodanie warstwy Dropout
        tf.keras.layers.Dense( units: 32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)),
        tf.keras.layers.Dropout(0.5),  # Dodanie warstwy Dropout
        tf.keras.layers.Dense( units: 1, activation='sigmoid')
    ])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train_normalized, y_train, epochs=50, batch_size=32,
                    validation_data=(X_test_normalized, y_test),
                    callbacks=[tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)])
```

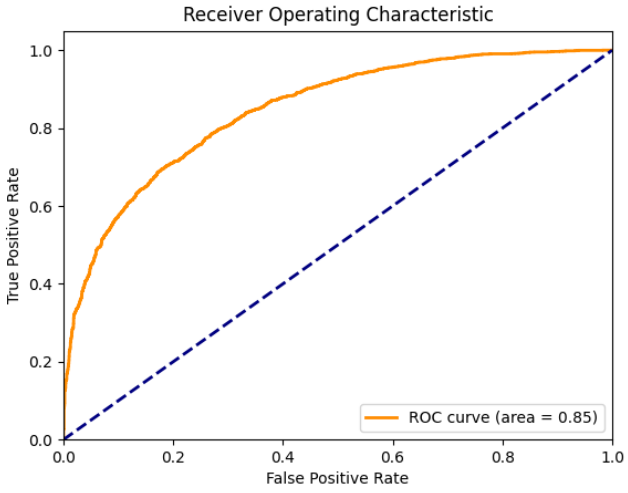
3.7 Regresja Logistyczna - Najlepsze wyniki

Regresja logistyczna jest modelem liniowym, który jest szeroko stosowany do zadań klasyfikacji binarnej. Po dopasowaniu najlepszego modelu z wykorzystaniem solver='liblinear', penalty='l2', maxIter=400 i C=0.1, uzyskałem najlepsze wyniki:

Dokładność klasyfikacji: **75.85%**
Wynik czułości: **75.02%**
Wynik swoistości: **76.67%**
Pole pod krzywą ROC: **0.85**

Macierz błędu:

$$\begin{bmatrix} 1811 & 568 \\ 593 & 1765 \end{bmatrix}$$



4 Reguły Asocjacyjne

Za pomocą analizy reguł asocjacyjnych szukałem ciekawych powiązań lub ich braku między różnymi cechami występującymi w rozgrywkach. Niektóre wyniki były dla mnie oczywiste (dzieki wieloletniemu doświadczeniu), a niektóre dały mi trochę do myślenia.

```
309  Antecedents: frozenset({'blueTeamTotalDamageToChamps', 'redTeamTotalGold', 'blueTeamTotalKills'})
310  Consequents: frozenset({'blueTeamTotalGold', 'redTeamTotalKills', 'redTeamTotalDamageToChamps'})
311  Confidence: 0.6441541892322395
312  Lift: 3.9474235373779027
313  Conviction: 2.351626814866641
314  Zhang's metric: 0.8949273845752262
```

Reguła asocjacyjna dotycząca łączności między liczbą zabójstw, zdobytym złotem oraz zadawanym obrażeniem czempionom wykazała wysoki poziom zaufania (64.4%) oraz znaczący współczynnik wzrostu (3.95). Oznacza to, że istnieje silna korelacja między tymi czynnikami, co sugeruje, że gracze, którzy osiągają wysoki poziom w jednym z tych wskaźników, mają również tendencję do osiągania wysokich wyników w pozostałych. Wartość współczynnika Zhang'a wynosząca 0.89 sugeruje, że reguła ta jest stosunkowo silna i wiarygodna. Bazując na swoich doświadczeniach uważam, że jest to wiarygodne, ponieważ (lekko spłaszczając to) zabijając innych czempionów zdobywamy więcej złota od przeciwnej drużyny, a drużyna przeciwna traci, tak samo jeśli jako drużyna niebieska zadajemy obrażenia przeciwnej drużynie, to najprawdopodobniej muszą też padać jakieś zabójstwa na naszą korzyść.

```
330  Antecedents: frozenset({'redTeamControlWardsPlaced', 'blueTeamControlWardsPlaced'})
331  Consequents: frozenset({'redTeamDragonKills', 'redTeamTowersDestroyed'})
332  Confidence: 0.43046650453239005
333  Lift: 1.00042675430881
334  Conviction: 1.0003224131228063
335  Zhang's metric: 0.0005603252214893278
```

Analizując ten wynik reguły asocjacyjnej, możemy wyciągnąć kilka wniosków. Wartość lift wynosi 1.0004, co oznacza, że wystąpienie antecedents i consequents nie ma żadnej zależności między sobą. Wartość conviction jest bliska 1. Możemy więc wnioskować, że brak kontroli wardów nie jest mocnym wskaźnikiem dla drużyny czerwonej, że nie zniszczy ona wież lub nie zabije smoka. Niska wartość Zhang's metric (0.0006) wskazuje na brak istotnego związku między antecedents a consequents. Z początku te wyniki lekko mnie zaskoczyły, ponieważ wizja w grze jest jedną z najważniejszych aspektów gry, ale po dłuższej analizie nabiera to sensu. Gracze na tym poziomie rzadko korzystają z *Control Ward'ów*, a już na pewno nie używają ich po to by zniszczyć wizję wroga.

5 Podsumowanie

Uważam, że osiągnięcie dokładności na poziomie ponad 75% stanowi bardzo dobry wynik, biorąc pod uwagę ograniczenia tej bazy danych. Nie dysponujemy informacjami na temat składu drużyn, kto jakie czempiony wybrał, posiadanej ilości złota przez poszczególnych czempionów czy nawet takimi danymi jak opuszczenie meczu przez któregoś z graczy, a jest tego więcej. Biorąc tę bazę danych objawiałem się tego, że posiadanie 65-70% skuteczności będzie bardzo ciężkie, ale przewidywania poszły lepiej niż się spodziewałem, więc sam osobiście jestem zadowolony

6 Źródła informacji

Baza Danych: **Link**
Chat-GPT: **Link**
Rozwiązanie użytkowników:*Murasame Lin*, *haobohan*, *JASONWCF*, *Yifei_ cx*: **Link**