Data Exploration Project

Report

Rainbow Six Siege Season 5 Ranked Dataset

vorgelegt am 20. April 2023

Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik

Kurs 1554194

von

Tien Dee Lin

1. Auswahl des Datensatzes von Kaggle

Für dieses Projekt wurde der Datensatz „Rainbow Six Siege – S5 – Ranked Dataset“ ausgewählt. Das Dataset enthält Daten über alle gespielten Rainbow Six Siege Spiele in der Saison vom Februar 2017 – Mai 2017. Rainbow Six Siege ist ein wettbewerbsorientierter, mehrspieler-taktischer Egoshooter, in dem 2 Teams aus jeweils 5 Spielern abwechselnd ein Spielziel in einem Gebäude angreifen oder verteidigen. Die Spieler können ihren Spielcharakter und die Räumlichkeiten ihres Spielziels frei wählen. Aufgrund der hohen Zerstörbarkeit des Gebäudes und weiteren vom Spieldesign gegebenen Variablen ist die korrekte Abstimmung und Auswahl von Spielcharakteren passend zum Ort des Spielzieles der Schlüssel zum Erfolg. Das Ziel von diesem Projekt ist, basierend auf den gegebenen Daten ein ML-Modell zu trainieren, welches bei gegebenen Teamkompositionen und Räumlichkeiten des Spielzieles in der Lage ist, den Ausgang des Spiels (Sieg/Niederlage) vorherzusagen. Da Rainbow Six Siege ein Egoshooter ist, stellt die individuelle Performance der Spieler dennoch eine große Zufallskomponente dar. Dieser Datensatz wurde aufgrund seiner hohen Vollständigkeit und Sauberkeit gewählt. Zudem ist es der einzige passende Datensatz für Rainbow Six Siege (kurz „R6“) auf Kaggle.

2. Datenvorverarbeitung

Der für das ML-Modell wesentlichste Schritt in der Vorverarbeitung war die Selektion der Datensätze. Es wurden für dieses Projekt nur Spiele verwendet, die in dem höchsten Wettbewerbsrang des Spiels stattgefunden haben („Diamond“ - Rang). Mit der Annahme, dass die besten Spieler individuell gut genug spielen können, um ihre Charaktere voll auszunutzen und jedes Team theoretisch in der Lage ist, das gegnerische Team zu schlagen, sollte eine Zufallskomponente des Spiels minimiert werden und die optimale Auswahl der Charaktere der Entscheidungsfaktor für Sieg und Niederlage sein.

3. Feature Engineering

In diesem Projekt wurde das „klassische“ Feature Engineering (z.B. das Berechnen eines neuen Features basierend auf vorhandenen Daten) nicht benötigt. Zum einen da der Datensatz über alle notwendigen Features verfügt (alle 10 ausgewählte Charaktere + Map + Räumlichkeiten des Spielziels), zum anderen potenzielle „neue“ Features wie die durchschnittliche Gewinnrate eines Charakters oder der Bias eines Spielziels für Angreifer/Verteidiger implizit in den Trainingsdaten bereits übergeben werden. Des Weiteren ist es aus den gegebenen Daten nicht möglich, die individuellen Fähigkeiten eines Spielers z.B. durch eine Kill-Death-Ratio zu bewerten, da es nicht möglich ist, einzelne Spieler zu identifizieren. Es wurden jedoch aufwendige Vorverarbeitungsschritte

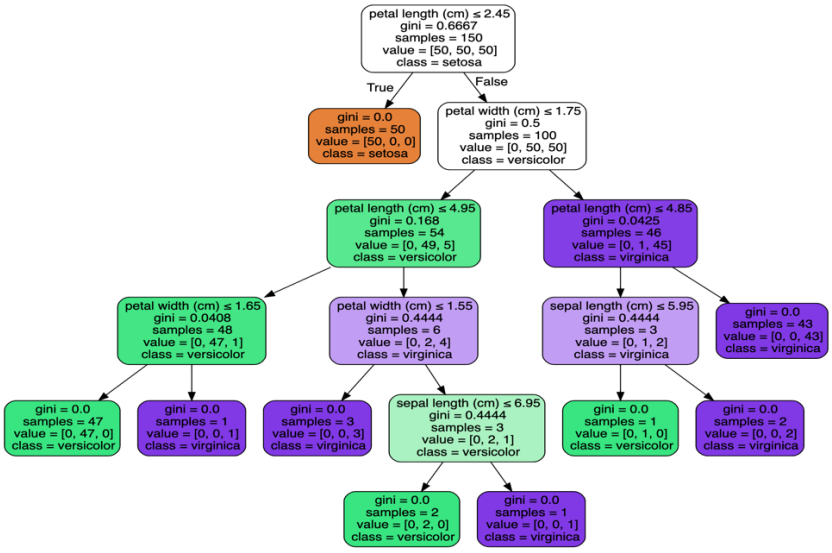
unternommen, um die Daten ins Trainingsformat zu bringen. Diese sind mit Dokumentation in DataExploration.py unter Preprocessing1 und Preprocessing2 zu finden. Zudem wurde One-Hot-Encoding durchgeführt, um die kategorischen Textdaten in binäre Form für das Training des ML-Modells umzuwandeln.

4. Split des Datensatzes

Der Datensatz wurde in 70% Trainingsdaten, 20% Validierungsdaten und 10% Testdaten aufgeteilt. Die Labels sind eine Spalte namens ‚winrole'. 0 bedeutet, dass die Angreifer gewonnen haben. 1 bedeutet, dass die Verteidiger gewonnen haben. Die restlichen Spalten des Dataframes sind die Features: 5 verschiedene Angreifer-Charaktere, 5 verschiedene Verteidiger-Charaktere die Räumlichkeiten des Spielziels und die Map (fiktiver Ort, wo sich das Spiel befindet).

5. Auswahl der ML-Methode

Als ML-Methode wurde der Decision Tree Classifier von Scikitlearn ausgewählt, da dieser Algorithmus sehr gut zu den Trainingsdaten passt und im Kontext von Rainbow Six Siege logisch gut erklärbar ist. Der Decision Tree Classifier ist eine supervisedlearning Methode zur Klassifizierung von Daten und basiert auf der Logik eines Entscheidungsbaumes. Bei einem fertig trainierten Baum wird ausgehend von der Wurzel an jedem Knoten eine auf den Features basierende Entscheidung getroffen, welcher Weg zu dem nächsten Knoten genommen werden sollte. Am Ende des Baumes wird die Klassifizierung vorgenommen. In diesem Fall ist die Klassifizierung binär (Wer hat gewonnen: Angreifer oder Verteidiger?).



Quelle: https://scikit-learn.org/stable/modules/tree.html

Abbildung 1: Beispiel eines visualisierten DecisionTreeClassifier von Scikitlearn

Der DecisionTreeClassifier ist für diesen Datensatz gut geeignet, da alle Features kategorisch und diskret sind. Es ist nicht möglich, einen numerischen Unterschied zwischen den Spielcharakteren und Spielzielen zu berechnen. Aufgrund der streng kategorischen Features gibt es auch keinen „Abstand zwischen Datenpunkten“. Somit können z.B. Regressions- und Clustering-Algorithmen nicht verwendet werden. In R6 ist es bei manchen Spielzielen ohne bestimmte Charaktere fast unmöglich für die Angreifer, innerhalb der Zeit an das Spielziel zu gelangen. Ein Teil des Entscheidungsbaumes könnte z.B. sein: Gibt es den Spielcharakter „Thermite“? ja → Angreifer siegen nein → Verteidiger siegen.

6. Auswahl der Metriken

Für dieses Modell werden die Metriken „accuracy“ und „confusion matrix“ zur Bewertung gewählt. Primär wird die „accuracy“ betrachtet, da es anzeigt, wie viele Datenpunkte das Modell prozentual im Test richtig klassifiziert. Dies ist am wichtigsten, da für z.B. einen Use-Case, in dem man basierend auf diesem Modell ein Glücksspiel betreiben möchte, um mit Wetteinsatz den Ausgang von Spielen vorherzusagen, bei einer accuracy von > 50% mit dem Gesetz der großen Zahlen langfristig Gewinne erzielen könnte. Die confusion matrix zeigt die korrekten und inkorrekten Vorhersagen des Modells für jede Klasse an. Dies ist im Colab-Notebook visualisiert und liefert einen Überblick darüber, ob das Modell besonders gut oder schlecht eine Klasse vorhersagen kann. So erhält man einen Einblick in die Art der Fehler, die das Modell macht.

7. Training des ML-Algorithmus

Zuerst wird ein untrainiertes Modell mit den Parametern „max_depth“ und „criterion“ angelegt und in eine leere Variable gespeichert. Danach wird das Modell auf den Trainingsdaten trainiert, die vorher gesplittet wurden und 70% des gesamten Datensatzes darstellen. Dabei wird übergeben, welche Spalten der Daten die Features und welche Spalten die Labels darstellen.

8. Hyperparameter-Tuning

Es wurden die Hyperparameter „max_depth“ und „criterion“ für das Tuning ausgewählt. „max_depth“ beschreibt die maximale Tiefe (min. 1), die der Entscheidungsbaum haben kann. Der Baum muss folglich innerhalb von n Entscheidungen den Datenpunkt endgültig klassifizieren. Dies dient dazu, die Größe des Baumes zu minimieren, was die allgemeine Rechenleistung einspart, wichtiger ist es jedoch, Overfitting auf die Trainingsdaten und damit eine schlechtere „accuracy“ beim Validieren und Testen zu vermeiden. Der Parameter

„criterion“ kann auf „gini“, „entropy“ oder „log-loss“ gesetzt werden und gibt die Methode an wonach die Qualität der Entscheidungen im Entscheidungsbaum bewertet wird und beeinflusst den Aufbau des Entscheidungsbaumes. Beide Parameter wurden mithilfe einer iterativen Suche nach der optimalen Genauigkeit des Modells getuned. Zwei verschachtelte For-Schleifen iterieren für jedes Criterion die maximale Tiefe von 1-20000 durch und trainieren jeweils ein Modell, das direkt auf Validierungsdaten bewertet wird. Es werden die Parameter von dem Modell mit der besten Validierungsgenauigkeit als optimale Parameter gewählt. Ein optimiertes Suchverfahren wie die binäre Suche ist hier nicht anwendbar, da kein Suchwert vorhanden ist und die Genauigkeit in der Reihe max_depth 1-20000 unsortiert ist. Die maximale Tiefe wird auf 20000 gesetzt, da insgesamt ca. 20000 Datenpunkte zur Verfügung stehen und Overfitting bei steigendem „max_depth“ stärker wird. Die optimalen Parameter sind max_depth = 12 und criterion = entropy. Die optimale Validierungsgenauigkeit liegt bei ca. 53.07%

9. Evaluation auf Testdaten

Im Test wird das optimale Modell mit einer Vorhersagegenauigkeit von ca. 50.38 % bewertet. In der Confusion-Matrix sieht man, dass das Modell leicht besser darin ist den Sieg der Verteidiger vorherzusagen und leicht schlechter darin ist den Sieg der Angreifer vorherzusagen. Jedoch ist zu erkennen, dass die Verteidiger in den Testdaten öfters gewonnen haben als die Angreifer. Insgesamt sind sich die Werte in der Confusion-Matrix jedoch sehr ähnlich. Die vollständige Visualisierung der Matrix befindet sich im Colab / Jupyter Notebook.

10. Reflektion

Die Genauigkeit von 50.38% im Test ist bei 2 Kategorien minimal besser als die Klassifizierung zufällig zu unternehmen und wäre dementsprechend für eine Glücksspielanwendung unbrauchbar. Die schlechte Genauigkeit kann aufgrund eines Selection-Bias sein. Da nur Spiele mit Spielern des höchsten Ranges ausgewählt wurden ist anzunehmen, dass alle diese Spieler bereits nahezu optimale Charakterkonstellationen spielen und der Ausgang rein von der Zufallskomponente des Individuums abhängt. Um dies zu umgehen, könnte man den Trainingsdatensatz wesentlich vergrößern und alle Spiele einbeziehen und den jeweiligen Spielerrang als Feature einfügen. Zusätzlich umfasst der höchste Rang eine große Menge an Spielern, und starke Unterschiede in der individuellen Performance der Spieler ist nicht auszuschließen. Dies kann man umgehen, indem man individuelle Statistiken wie die Anzahl gespielter Stunden, Kill-Death-Ratio oder Win-Loss-Ratio der Spieler einbezieht. Jedoch liegen diese Daten im Datensatz nicht vor.