Rollenbasierte Klassifizierung im Bezug auf Matchmaking in League of Legends

Florian Eckerstorfer

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2015

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 29. Juni 2015

Florian Eckerstorfer

Inhaltsverzeichnis

Kurzfassung

In League of Legends, einem Multiplayer Online Battle Arena Game, spielen täglich mehrere Millionen Menschen in 5vs5 oder 3vs3 Spielen gegeneinander. Dazu bedarf es eines guten Matchmakings um geeignete Gruppen zu bilden. Im Rahmen dieser Gruppenfindung kann es zu Frustration und Streit kommen. Diese Arbeit bietet für dieses Problem einen Lösungsansatz. Durch eine Einteilung der Spieler in ihre bevorzugten Rollen könnte das Matchmaking verbessert werden. Mit Hilfe von Entscheidungsbäumen werden 1500 Datensätze mit Spieldaten vergangener Spiele klassifiziert und dem Spieler eine Rolle zugewiesen.

Dabei wird zwischen fünf unterschiedlichen Rollen, die eingenommen werden können, differenziert: Top-Laner, Jungler, Mid-Laner, ADCarry und Supporter. Mithilfe 18 ausgewählter Attribute können diese unterschieden werden. Precision, Recall und f1-score haben im Durschnitt einen Wert zwischen 75–90% für die einzelnen Klassen. Eine Auswertung aller Confusion Matrizen zeigt, dass im Schnitt etwa 80% aller Klassifikationen richtig und 20% falsch liegen.

Diese Zahlen sind ein vielversprechender erster Versuch, der etwa durch eine Vergrößerung des Datensatzes noch verbessert werden könnte. Auch eine maschinelle Bewertung der Attribute wäre eine sinnvolle Erweiterung.

Abstract

In League of Legends, a Multiplayer Online Battle Arena Game, several million people play in 5vs5 and 3vs3 Matches on a daily basis. To build a fitting group for those games a decent matchmaking system is needed. Throughout the process of group finding frustration, disagreement and arguments between the players can take place.This thesis tries to find a solution for this problem. Through classifying the preferred roles of the players the matchmaking could be improved. With the classification of 1500 datasets of past game data, player roles should be found.

There are five different roles: Top-Laner, Jungler, Mid-Laner, ADCarry and Support. These can be separated by 18 chosen attributes. Precision, recall and f1-score show on average numbers between 75 to 90% for the individual roles. The combination of all retrieved confusion matrices shows that on average around 80% of all classifications are correct and 20% are wrong.

These numbers are a promising first try, but can be enhanced by increasing the dataset. Also an evaluation of the used attributes by machine learning algorithms could be a useful extension.

Kapitel 1

Einleitung

Diese Arbeit beschäftigt sich mit der Idee, das Matchmaking von League of Legends (LoL) zu erweitern, bzw. zu verbessern. LoL ist ein Multiplayer Online Battle Arena Game, welches täglich von mehreren Millionen Spielern gespielt wird. Dabei werden Gruppen von fünf oder drei Personen benötigt, die gegen eine weitere Gruppe in derselben Größe spielen. Dazu bedarf es eines guten Matchmakings. Leider kommt es im Zuge dieser Gruppenfindung häufig zu Frustration und Streit. LoL ist ein strategisches Spiel und für ein gutes Handeln der Gruppe ist eine Einteilung in gewisse Spielrollen nötig. Viele Spieler präferieren eine dieser Rollen, weshalb es zu Konflikten kommen kann, wer welche Rolle spielen darf bzw. muss.

Mit Hilfe von Entscheidungsbäumen und den Spieldaten vergangener Spiele versucht diese Arbeit diese Problematik zu verhindern. Durch eine Einteilung der Spieler, basierend auf ihrer Vergangenheit, könnte das Matchmaking versuchen, homogene Gruppen zu bilden.

Durch die Klassifizierung konnten vernünftige und brauchbare Ergebnisse erzielt werden. Eine durchschnittliche Erkennung der Rollen zwischen 75–90% gibt Grund zur Annahme, dass eine Verbesserung des Matchmakings möglich wäre. Mehr Datensätze und eine Ausweitung der Lern-Algorithmen könnten noch bessere Ergebnisse liefern. Die Attribute für die Klassifikation sind manuell ausgewählt, was einen Schluss zulässt, das hier durch passende Methoden eventuell bessere Attribute gefunden werden könnten.

Die Spieler werden in fünf Rollen eingeteilt, wobei 18 unterschiedliche Attribute zur Verfügung stehen. In LoL gibt es Ranglisten, welche die Spieler nach ihrer Stärke einteilen. Aus jeder dieser Ligen wurden 25 Spieler entnommen und klassifiziert. Neben einem Ergebnis einer Liga wurden auch Kombinationen dieser getestet. Durch Precision, Recall und f1-score wurden die Klassifikation bewertet und eine Confusion Matrix gab Auskunft über die aufgetretenen Verwechslungen.

Die Ergebnisse zeigen, dass durchaus ein Verbesserungsbedarf herrscht, jedoch die Idee und deren Ausführung ohne weiteres möglich wäre. Der

Schritt zur Verbesserung des Matchmaking ist ein großer, könnte aber zu einem besseren Spielerlebnis und mehr Freude beitragen. In Anbetracht der Größe des Herstellers von League of Legends erscheint es als leichtes Unterfangen, einen Versuch in diese Richtung zu starten.

1.1 Struktur

Diese Arbeit gliedert sich in 6 Kapitel. Im folgenden Kapitel wird auf bestehende Arbeiten rund um das Thema Klassifikation im Zusammenhang mit Games eingegangen. Auch Ideen bezüglich Matchmakings und andere Spiele desselben Genres werden gezeigt.

In Kapitel 3 folgen die theoretischen Grundlagen dieser Arbeit. So wird zuerst das Spiel League of Legends erklärt um ein Verständnis für die Spielerrollen, sowie das Spiel an sich zu entwickeln. Der zweite Teil des Kapitels widmet sich den Grundlagen der Entscheidungsbäume. Hier findet auch der Algorithmus, der im weiteren Verlauf verwendet wird, Platz zur näheren Erklärung.

Im Rahmen dieser Arbeit wurde ein Projekt realisiert, welches in Kapitel 4 dargestellt wird. An dieser Stelle werden auch die Details zur Implementierung beschrieben.

Kapitel 5 beschäftigt sich mit den Ergebnissen und der Auswertung der Daten. Außerdem werden sowohl die ausgewählten Daten, als auch wichtige Kennzahlen in Verbindung mit Klassifikation erklärt. Anhand von Diagrammen werden die Ergebnisse anschaulich präsentiert und Schlüsse daraus gezogen.

Ein Fazit schließt die Arbeit in Kapitel 6 ab. Darin wird auf Probleme und Verbesserungen eingegangen, ein Rückschluss gezogen und versucht einen kleinen Ausblick zu geben.

1.2 Gendering

Um einen besseren Lesefluss und eine angenehme Lesbarkeit zu bewirken, wird in dieser Arbeit auf eine geschlechtsgerechte Schreibweise verzichtet. Es wird daher darauf hingewiesen, dass mit dem verwendetem Maskulinum gleichzeitig die weibliche und die männliche Person angesprochen werden.

Kapitel 2

Bestehende Arbeiten

Diese Arbeit beschäftigt sich mit der Einteilung von Spielertypen und deren Rollen innerhalb des Spiels League of Legends. Aufgrund der immer größeren Anzahl an Spielern und der wachsenden Gaming Industrie [17] beschäftigen sich verschiedenste wissenschaftliche Arbeiten mit ähnlichen Bereichen.

Wie bereits erwähnt, entstand die Motivation für diese Arbeit aufgrund des oft problematischen Matchmakings in League of Legends. Sie soll einen Weg aufzeigen, wie dieses verbessert werden könnte bzw. wie es erweitert werden könnte um zu weniger Konflikten zwischen den Spielern zu führen.

Es gibt unterschiedliche Arten, wie man sich mit diesen Themen beschäftigen kann und hier sollen ein paar dieser aufgezeigt werden. Es folgen Arbeiten, die sich mit Matchmaking auseinandersetzen. Welche, die sich mit der Kategorisierung von Spielrelevanten Daten (entweder Spielern, oder auch den spielbaren Charakteren) beschäftigen und auch Arbeiten welche zeigen, was mit den verfügbaren Daten von League of Legends erstellt werden kann.

2.1 Spieler

In dieser Arbeit spielt das Verhalten des Spielers eine große Rolle. Was wählt der Spieler aus? Wie setzt er es um? Fällt die Leistung in ein gewisses Schema? Ein großes Problem in League of Legends ist schlechtes Benehmen der User. Durch die Anonymität des Internets geschützt, kommt es häufig zu Beschimpfungen und ausfallendem Verhalten. Oft reichen dafür Kleinigkeiten aus. Viele Spieler reduzieren ihren Frust dadurch. So kommt es oft schon vor dem Spiel zu Streitereien aufgrund der Rollen, die die Spieler einnehmen sollten und welche sie einnehmen wollen. In [2] beschäftigen sich Blackburn und Kwak mit dem System von Riot Games, das mit verhaltensauffälligen Spielern arbeitet, dem Tribunal. Dieses System ist mittlerweile überholt. Wie in [22] und [21] nachzulesen ist, versucht Riot Spieler zu ermutigen und weniger zu bestrafen, weshalb das Tribunal überarbeitet werden soll. Bei dem Tribunal handelte es sich um ein System, bei dem jeder Spieler mit gemelde-

ten Fällen arbeiten und diese bewerten kann. Nach jedem absolvierten Spiel kann ein Teilnehmer andere melden, weil diese sich schlecht benommen haben, oder beispielsweise das Spiel frühzeitig verlassen haben usw. Blackburn und Kwak schlagen einen Algorithmus vor, der anstatt der Spieler, die Fälle behandeln soll und somit eine Zeitersparnis und eine Arbeitserleichterung mit sich bringt.

Auch in [9] beschäftigen sich Kou und Nardi mit dem System des Tribunals und welche Regeln und Vorschriften beachtet werden sollten und analysieren die unterschiedlichen Themen rund um das Verhalten der Spieler und das Tribunal. Interessant dabei ist, dass sie feststellen, dass die Vorschriften und Regeln von LoL nicht nur von Riot vorgegeben werden, sondern eine große Interaktion zwischen Community und Riot herrscht. Dabei wird stets versucht Normen zu finden, wie sich ein Spieler zu verhalten hat.

Wie im Interview mit Jeffrey Lin [21] hervorgeht handelt es sich nur um einen kleinen Bruchteil an Spielern, die scheinbar unverbesserlich sind, jedoch reicht dieser geringe Anteil aus, um weitere Spieler "anzustecken".

In [4] werden die Spieler eines anderen MOBAs (3.1), Heroes of Newerth (HoN), analysiert. Dabei geht es um ihre Performance im Spiel verglichen mit ihrem Rating, also ihrer Reihung. Das Matchmaking in HoN hat ein Rating-System zu Grunde liegen, anhand dessen Spieler zusammengewürfelt werden, ähnlich wie auch bei League of Legends. Neven et al. betrachten sehr viele unterschiedliche Features, die den "true skill" wie sie es nennen, zeigen sollen. Es geht ihnen primär darum, ob das derzeitige Matchmaking auch tatsächlich Gamer auf gleichem Niveau zusammenführt. Sie kommen dabei zu dem Schluss, dass dies durchaus der Fall ist, bis auf wenige Ausnahmen, die aber schwer auszubessern sind.

2.2 Matchmaking

Auch Myslak und Deja beschäftigen sich in [11] mit Matchmaking, um Streitereien vorzubeugen. Sie schlagen dabei vor, die Spieler zu betrachten und ihre vergangenen Spiele zu analysieren und anhand der meist gewählten Rolle vorzugehen. Dieser Versuch ist ähnlich zu dem, was auch in dieser Arbeit vorgeschlagen wird. Die Vorteile sehen die beiden darin, dass jeder Spieler somit seiner Präferenz nachgehen kann. Vorgeschlagen wird, dass anhand der Stärke in den jeweiligen Rollen ein Team gebildet wird. Sie stützen sich dabei auf die Prämisse, dass ein Spieler eine Rolle, die er besser spielt, auch bevorzugt. Myslak und Deja versuchen anhand der Analyse von ungefähr 2000 Datensätzen ihre Behauptung zu untermauern, indem sie Teams vergleichen und betrachten, wie viele der fünf Spieler in ihrer bevorzugten Rolle spielen. Eindrucksvoll ist dabei, dass ein Team mit 65 prozentiger Wahrscheinlichkeit gewinnt, wenn alle fünf Rollen bevorzugt werden, gegenüber einem Team, in dem kein Spieler seine bevorzugte Rolle erhielt.

Jimenez-Rodriguez et al. beschreiben in ihrer Arbeit [7] zwei Varianten um passende Teams zu finden. Erstere beschäftigt sich mit dem klassischen Ansatz, Spielern eine Reihung je nach Können zu geben. In etwa wie das Elo Rating System, oder das TrueSkill System. Die Spieler werden dabei in Kategorien eingeteilt und anschließend wird geprüft, ob genügend Spieler zur Verfügung stehen um eine passende Gruppe zu erstellen. Dies ist die Art von Matchmaking, die großteils in den Spielen verwendet wird.

Jimenez-Rodriguez et al. beschreiben daher noch eine weitere Variante, in der sie auf die Rollenverteilung eingehen. Dabei schreiben sie auch, dass dies eine Verbesserung des Matchmakings darstellen soll. Sie berichten über Ideen, die sich nicht nur um Balancing drehen. Für sie ziehen auch in Betracht, ungleiche Teams für Trainingszwecke zu erstellen. Sie beziehen sich dabei nicht auf ein konkretes Spiel, sondern versuchen ihren Ansatz möglichst allgemein zu halten. Sie weisen dabei auf das Problem der Kombinatorik hin und dass in diesem Bereich viele Berechnungen nötig sind, da eine hohe Anzahl an Spielern mit Rollen zu einer hohen Anzahl an Berechnungen führt. Welcher Spieler entspricht welcher Rolle und wie kann dieser Spieler mit anderen Spielern zusammengeführt werden?

2.3 MOBAs

Die wachsende Community und Spieleranzahl von League of Legends scheint sich auch in wissenschaftlichen Arbeiten zu zeigen. Doch auch weitere Spieltitel, die auf dem selben Spielprinzip (siehe Abbildung 2.1) des MOBAs beruhen, sind am boomen. So wie etwa Dota2, der Nachfolger des Urvaters des Genres.

So ist es nicht verwunderlich, dass sich Gao et al. mit den Spielfiguren von Dota2 beschäftigen in [6]. Ihr Ziel ist es, die spielbaren Charaktere (Champions) in gewisse Rollen einzuteilen. Dabei betrachten sie Spieler und deren Rolle. Sie beschreiben, dass sie 275 verschiedene Features verwenden um eine Klassifikation zu erstellen. Jeder Champion soll in eine von drei Kategorien – *Carry*, *Solo Lane* und *Support* – eingeteilt werden. Mittels *Überwachter Lerntechniken* wie Logistische Regression und Random Forest erzielen sie eine Erfolgsrate von ungefähr 80%. Dabei stellen sie eine Streuung zwischen professionellen und normalen Spielern fest. Bei Profi-Spielern ist die Erfolgsrate sogar 89%. Weiters sind sie der Meinung, dass eine Steigerung der Rate möglich wäre, wenn der Hersteller von Dota2 mehr Daten zur Verfügung stellen würde.

Feo, Ma und Sirolly verwenden die von Riot Games offiziell zur Verfügung gestellte Schnittstelle, um ein Programm zur Vorhersage des Gewinners zu erstellen [5]. Sie vergleichen dabei die Spieler untereinander und untersuchen deren Spielstärke anhand von beispielsweise insgesamt gespielten Spielen, insgesamt gewonnenen Spielen oder auch insgesamt ausgeteiltem

(a)

(b)

Abbildung 2.1: Karte von League of Legends (a), Karte von Dota2 (b).

Schaden. Anhand von unterschiedlicher Maschineller Lernmethoden erreichen sie dabei eine Quote von zwischen 58,15 und 62,38% richtig voraus gesagter Spiele.

Auch Kim, Ghorasy und van der Vecht beschäftigten sich mit der Vorhersage des Gewinners. In [8] ist nachzulesen, dass ein anderer Weg eingeschlagen wird, um zu einem Ergebnis zu kommen. Kim et al. verwenden eine nicht genauer beschriebene Kategorisierung der spielbaren Charaktere in neun Klassen. Dabei beziehen sie sich auf einen Experten des Spiels. Anschließend werden die Spiele betrachtet und die Spieler je nach Wahl ihres Charakters in die Klassen eingeteilt. Mit 15 000 Trainingssätzen wurde anschließend gearbeitet und wiederum mittels Maschineller Lernmethoden versucht, das siegreiche Team zu ermitteln. Das bedeutet, dass Kim et al. anhand der Teamkomposition alleine versuchen, einen Sieger zu finden. Sie verwenden dafür auch nur Datensätze von Spielern mit durchwegs hoher Wertung, weil sie der Meinung sind, dass nur bei diesen Spielern genügend Teaminteraktion herrscht. Während bei schlechteren Spielern eine Teamkomposition weniger ins Gewicht fällt.

Kapitel 3

Grundlagen

In diesem Kapitel sollen die Grundlagen für das Verständnis der Arbeit gelegt werden. Dazu gliedert sich das Kapitel in zwei Teile: *League of Legends* und *Entscheidungsbäume*. Im ersten Teil dieses Kapitels wird dem Leser das zu Grunde liegende Computerspiel League of Legends näher gebracht und die im weiteren Verlauf verwendeten Begriffe und Ausdrücke erklärt. Außerdem werden zur Ermittlung der Ergebnisse Entscheidungsbäume herangezogen, weshalb sich der zweite Teil des Kapitels mit diesen beschäftigt.

3.1 League of Legends

Diese Arbeit beschäftigt sich mit sogenannten MOBAs, oder *Multiplayer Online Battle Arena Games*. Das Spiel League of Legends ist dieser Gattung von Spielen zuzuordnen. Grundsätzlich handelt es sich dabei um Spiele in denen sich zumeist zwei Teams mit jeweils drei bis fünf Spielern duellieren. Ziel ist dabei, die Basis des jeweils anderen Teams zu zerstören.

Jeder Spieler kontrolliert einen Helden oder auch Champion (diese Spielfigur hat unterschiedliche Namen in den Spielen). Diese Charaktere werden im Laufe des Spiels immer stärker, da sie Erfahrung sammeln und Gegenstände kaufen können.

League of Legends ist das einzige Computerspiel des Herstellers Riot Games und erstmals im Jahr 2009 erschienen. Es handelt sich dabei um ein Free-to-play-Spiel. Das heißt, es finanziert sich durch sogenannte Micro Transactions: Spieler können sich mit Hilfe von Geld gewisse Dinge im Spiel kaufen. Wenn sich die Spieler während eines Matches duellieren, kann jedoch weder mit Riot- noch mit Einflusspunkten ein Vorteil erworben werden. Unter Riot-Punkten versteht man eine in-Game Währung, die mit Echtgeld zu erwerben ist. Einflusspunkte sind hingegen eine in-Game Währung, die man durch Spielen erlangt. Das ist ein essentieller Punkt von League of Legends: kein Spieler kann sich einen Vorteil erkaufen. Beliebte Käufe sind sogenannte Skins, die das Erscheinungsbild der unterschiedlichen Charaktere

Abbildung 3.1: Zwei unterschiedliche Erscheinungen des Champions *Renekton*.

verändern (siehe Abbildung 3.1).

Täglich spielen in etwa 27 Millionen Menschen (Stand Jänner 2014 vgl. [16]), womit sich auch erklärt, wie mit kleinen Beträgen ein Umsatz gemacht werden kann: über die schiere Masse an Spielern.

3.1.1 Spielprinzip

League of Legends (LoL) bietet zur Zeit zwei verschiedene Spielmodi auf vier verschiedenen Spielkarten an. Für diese Arbeit relevant ist der Klassische Modus und die Kluft der Beschwörer. Dabei handelt es sich um eine fünf gegen fünf Spielkarte, in der die gegnerische Basis zerstört werden muss, um das Spiel zu gewinnen. Bevor ein jedes Spiel beginnt, wählen die 10 Spieler entweder gleichzeitig oder einer Reihenfolge nach, ihre *Champions* aus. Dabei handelt es sich um die spielbaren Charaktere, die unterschiedliche Stärken und Aufgaben haben. In LoL wird der Spieler zumeist als Beschwörer bezeichnet und die spielbare Figur als Champion.

In der Kluft der Beschwörer(siehe Abbildung 3.3) befinden sich die Basen der Teams in den gegenüber liegenden Ecken (links unten und rechts oben). Zwischen diesen befinden sich drei Wege, sogenannte *Lanes*. Jede Basis besteht aus zehn Gebäuden: dem Brunnen, dem Nexus, drei Inhibitoren, sowie fünf Türmen. Zusätzlich befinden sich auf jeder der Lanes noch weitere zwei

Abbildung 3.2: v. li. n. re. Nexus, Inhibitor, Turm (Ingame Models).

Türme.

Der **Brunnen** ist der Spawn-Punkt eines Spielers. Sowohl zu Beginn, als auch nach dem Ableben eines Champions, wird dieser hier wiederbelebt. Außerdem können auf dem Brunnen Gegenstände gekauft werden, die den Champion verbessern.

Der **Nexus** ist das zentrale Gebäude in dem Spiel, diesen gilt es zu zerstören bzw. zu verteidigen. Ist der Nexus eines Teams zerstört, so hat dieses verloren. Gleichzeitig werden von dem Nexus auch computergesteuerte Kreaturen erschaffen (Minions) welche sich auf die einzelnen Lanes aufteilen und dort gegen die Minions des Gegners sowie deren Türme kämpfen.

Die **Inhibitoren** sind Gebäude, deren Wirkung sich erst entfacht, wenn sie zerstört werden. Wie sich aus dem Namen schon erkennen lässt, schwächen Inhibitoren die Minions des Gegners auf der jeweiligen Lane, sodass wenn einer zerstört wird, anschließend die Minions des Gegners besser werden. Das bedeutet, sie verursachen mehr Schaden und sind widerstandsfähiger. Nach fünf Minuten ersteht ein Inhibitor wieder von selber auf.

Die **Türme** dienen rein der Verteidigung: Diese attackieren die gegnerischen Einheiten und auch die gegnerischen Champions. Wobei die Türme in der Basis etwas mehr Schaden austeilen, als die außerhalb. Zu beachten gilt, dass ein Gebäude immer nur dann zerstört werden kann, wenn das Gebäude davor schon zerstört worden ist, bzw. es das äußerste Gebäude ist. Das heißt, dass beispielsweise ein Inhibitor erst angegriffen werden kann, wenn die drei Türme, die sich davor befinden, schon ausgeschaltet wurden. Die letzten beiden Türme können zerstört werden, sobald ein Inhibitor zerstört ist und der Nexus danach. Es müssen also nicht alle Inhibitoren vernichtet werden, um das Spiel zu gewinnen.

Zwischen den Lanes, die aufgrund ihrer Position auf der Karte allgemein hin als *Top-, Middle-* und *Bottom-Lane* bezeichnet werden, befindet sich ein neutraler Bereich: Der *Dschungel*. Hier gibt es mehrere Lager, wo sich

Abbildung 3.3: Karte der Kluft der Beschwörer.

Monster (neutrale Minions) aufhalten. Diese attackieren den Spieler nur, wenn sie auch attackiert werden. Im Gegensatz dazu greifen Minions immer an, sobald man sich in ihrer Nähe befindet. Außerdem gibt es in der Mitte einen großen Fluss, an dem sich zwei wichtige Punkte befinden: Im unteren Teil der Karte wartet der Drache und im oberen Teil der Karte befindet sich Baron Nashor.

Baron Nashor ist ein mächtiges, neutrales Monster, welches erst nach 20 Minuten Spielzeit erscheint. Schafft es ein Team Baron Nashor zu besiegen, so erhält es eine sehr starke Aufwertung. Für drei Minuten erhöht sich die Angriffsstärke der Champions und alle Minions in der Nähe eines Champions werden auch aufgewertet. Diese Aufwertungen nennt man in LoL *Buffs*. Baron Nashor ersteht nach sieben Minuten wieder auf.

Der **Drache** ist ebenfalls ein mächtiges, neutrales Monster, welches nach zwei Minuten und 30 Sekunden erscheint und nach dem erfolgreichen Töten einen Buff (Stärkungszauber) gewährt. Dieser wird mit jeder erneuten Tötung durch dasselbe Team stärker. Der Drache ersteht nach 6 Minuten wieder auf.

Durch Töten der genannten Gebäude sowie der erwähnten Monster, er-

hält ein Spieler Erfahrung und Gold. Ebenso wie durch das Töten der gegnerischen Minions und Champions. Stirbt ein Champion, so erscheint dieser nach einer gewissen Zeit wieder auf dem Brunnen. Diese Zeit variiert und nimmt mit der Dauer des Spiels und dem Level des Champions (mehr dazu in Abschnitt 3.1.4) zu.

3.1.2 Champions

In LoL gibt es die unterschiedlichsten Champions zu spielen. Mittlerweile handelt es sich um 124 (Stand: Juni 2015) verschiedene Champions, die einem Spieler zur Verfügung stehen können. Jeder Spieler kann pro Woche zehn Champions spielen und testen. Diese zehn Champions stehen gratis zur Verfügung und wechseln alle sieben Tage. Andere als diese zehn Champions müssen gekauft werden. In Abbildung 3.4 ist ein Beispiel eines Champions zu sehen. Riot Games veröffentlicht in unregelmäßigen Abständen neue Champions.

Gekauft werden kann auf zwei verschiedene Arten: Riot Punkte und Einflusspunkte. Riot Punkte kann jeder Spieler mit echtem Geld erwerben und damit in dem Spiel verschiedene Dinge freischalten, wie etwa Champions oder unterschiedliches Aussehen. Einflusspunkte erhält jeder Spieler nach jedem Spiel. Für einen Sieg mehr, für eine Niederlage weniger. Diese kann man sparen und damit Champions und weitere Dinge in dem Spiel kaufen.

Jeder der 124 Champions verfügt über fünf verschiedene Fähigkeiten. Diese teilen sich in eine passive und vier aktive Fähigkeiten auf. Zusätzlich gibt es noch zwei Beschwörerfähigkeiten. Der Unterschied liegt, wie der Name vermuten lässt, in der Art, wie diese ausgelöst werden. Während aktive Fähigkeiten von dem Spieler selber verwendet und ausgewählt werden müssen, wird die passive Fähigkeit automatisch ausgelöst und muss auch nicht speziell verwendet werden. Mit jedem Levelaufstieg während des Spiels kann sich der Spieler entscheiden, welche der Fähigkeiten er erlernen, bzw. verbessern möchte. Ein Spiel beginnt mit Level 1, somit hat jeder Spieler zu Beginn eine Fähigkeit zur Verfügung.

Neben dem Effekt der Fähigkeiten unterscheiden sich diese auch noch darin, wie sie verbessert werden können. Grundsätzlich gibt es bei LoL zwei Möglichkeiten, die Auswirkungen von Fähigkeiten zu verbessern. Auf der einen Seite durch Erhöhung des Angriffsschadens, auf der anderen Seite durch Erhöhung des Fähigkeitsschadens. Dies sind die zwei großen Kategorien. Dabei handelt es sich um Attribute die ein jeder Champion besitzt. Weitere dieser Attribute, wären beispielsweise Leben, Mana, Rüstung, Angriffstempo, Regeneration usw. Jede dieser Eigenschaften bestimmt darüber, wie sich ein Champion verhält und worin seine Stärken und Schwächen liegen. Ebenso wichtig dafür ist die Abklingzeit, welche die Zeit bezeichnet, in der einem Champion seine Fähigkeiten nach deren Verwendung nicht zur Verfügung stehen.

Abbildung 3.4: *Lissandra* und ihre Fähigkeiten.

Nimmt man alle diese Eigenschaften und Besonderheiten zusammen, so findet man unterschiedlichste Arten von Champions und auch unterschiedlichste Vor- und Nachteile.

3.1.3 Gegenstände

Wie nun bereits mehrmals erwähnt, besteht für einen Spieler die Chance seinen Champion zu verbessern, in dem Gegenstände gekauft werden. Mit Hilfe des verdienten Goldes kann sich ein Spieler frei entscheiden, welche Gegenstände er seinem Champion kaufen möchte. Anhand der Stärken und Schwächen und auch der Rolle (siehe Abschnitt 3.1.4), sollte ein Spieler entscheiden, was am besten zu seinem Charakter passt. Zur Unterstützung werden unzählige Gegenstände vorgeschlagen. Durch genaues Betrachen der Fähigkeiten und deren Skalierung, ist einem Spieler auch meist klar, was gekauft werden sollte.

Als kurzes Beispiel betrachten wir einen Champion, bei dem drei von vier Fähigkeiten mit Angriffsschaden skalieren und nur eine von vier mit Fähigkeitsstärke. Dies bedeutet mit sehr wenigen Ausnahmen, dass der Spieler gut darin tut, sich Gegenstände, die den Angriffsschaden erhöhen zu kaufen.

Gegenstände können auch kombiniert werden, um mächtigere Boni zu erhalten. Generell kann jeder Spieler sieben Gegenstände besitzen, wobei sechs davon frei wählbar sind und bei dem siebten die Auswahl zwischen drei unterschiedlichen Gegenständen besteht.

Allgemein haben die Spieler auf der Karte keine Sicht über Orte, an denen sich nicht ein Mitspieler, ein eigenes Gebäude, ein Minion oder sie selbst befinden. Deshalb gibt es Gegenstände (Wards), welche platziert werden können, um in einem Bereich für Sicht zu sorgen.

3.1.4 Rollen

League of Legends hat eine sehr große E-Sports-Community und eigene E-Sports Ligen mit gesponserten Teams und bezahlten Spielern. In Südkorea

beispielsweise wird ein Team von der Firma Samsung gesponsert. Im Jahr 2014 betrachteten 11.2 Millionen Menschen das Finale der League of Legends Weltmeisterschaft [18]. Aus dieser Profiszene entwickelt sich ein großer Teil des Spiels. Die Spieler entwickeln ständig neue Strategien und neue Arten, wie das Spiel am effizientesten und besten gespielt werden kann. Im Laufe der Zeit haben sich so gewisse Rollen etabliert, die ein Spieler einnehmen kann [19], vergleichbar etwa wie im Fußball, wo es Spieler, die verteidigen, einen Tormann und Stürmer usw. gibt. Oder auch bei fast allen anderen Sportarten, wie etwa Basketball und American Football und vielen weiteren.

Riot Games teilt alle veröffentlichten Champions in gewisse Rollen, in denen sie am besten gespielt werden sollten. Durch einen Wandel in der Turnierszene können aber unterschiedliche Facetten des Champions zum Vorschein gebracht werden, wodurch dieser in anderen Rollen glänzt. Auch die ständige Weiterentwicklung des Spiels kann dazu beitragen, dass sich die Rolle des Champions verändert.

Champions werden sechs verschiedenen Kategorien zugeteilt. Nicht immer entspricht ein Champion nur einer Rolle, in vielen Fällen kann der Spieler durch das Kaufen der Gegenstände zwischen den Kategorien entscheiden. Die sechs Kategorien sind: Assassine, Kämpfer, Magier, Unterstützung, Tank und Schütze [27]. Zusätzlich zur Kategorie spielt auch die Position des Spielers auf der Karte zu Beginn eines Matches eine Rolle. Die Kategorie und die Position bestimmen über die Rolle eines Spielers.

- Der *Assassine* bezeichnet einen Champion, dessen Ziel es ist, Gegner sehr schnell zu töten und gut im Duellieren ist. Das heißt, der Assassine ist im 1vs1 oft anderen Champions überlegen.
- Der *Kämpfer* (Bruiser) ist ein Champion, welcher verhältnismäßig viel aushält, aber gleichzeitig immer noch Schaden an schwachen Gegnern ausrichtet. Sein Ziel ist es, sich auf den Schützen zu stürzen und diesen auszuschalten. Hier hängt auch viel von den gekauften Gegenständen ab. Meist kann ein Kämpfer auch als Tank gespielt werden.
- Der *Tank* ist ein Champion, der sehr viel Schaden aushält und dessen Ziel es ist, die *Carries* (die Spieler, die den meisten Schaden verursachen) zu beschützen. Bei der Wahl der Gegenstände wird auf Rüstung, Magieresistenz und Leben geachtet. Auch kann es sein, dass manche Fähigkeiten mit eben diesen Attributen skalieren.
- Als *Unterstützung* (Supporter) werden jene Champions bezeichnet, die auch ohne viele Gegenstände immer noch einen entscheidenden Input für das Match haben. Das heißt, diese Champions verfügen über sogenannte Crowd-Control Fähigkeiten oder Buffs. Ziel des Supporters ist es, zu Beginn des Matches den Schützen zu beschützen und im späteren Verlauf allen anderen Spielern auszuhelfen und für Sicht auf der Karte zu sorgen.
- Der *Schütze* (Marksman) ist ein Champion, der vorwiegend physischen

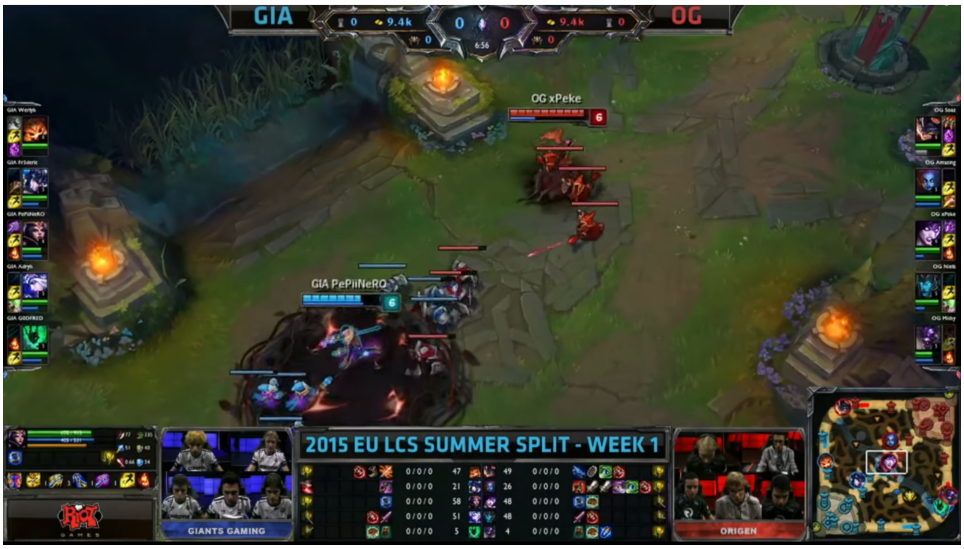


Abbildung 3.5: Zwei Mid-Laner beim Duell während eines Spieles der Professionellen E-Sports-Liga.

Schaden anrichtet. Das Ziel des Schützen ist es viel Gold zu erhalten und viele Gegenstände zu kaufen, die seinen Basisschaden erhöhen. Ein Schütze hat zumeist wenig Verteidigung und kann schnell getötet werden, weshalb er Hilfe braucht. Der Schütze ist wichtig, um in Team-Fights durchgehend Schaden zu verursachen.

- Der *Magier* (Mage) ist, ähnlich wie der Schütze, hauptverantwortlich für den Schadensoutput, jedoch verlässt sich der Magier hauptsächlich auf seine Fähigkeiten und skaliert mit Fähigkeitenstärke. Der Magier kann in längeren Kämpfen nicht durchgehend für Schaden sorgen, da seine Fähigkeiten eine Abklingzeit haben.

In dieser Arbeit werden die Rollen eines Spielers wie folgt festgelegt: Top-Laner, Mid-Laner, Supporter, ADCarry und Jungler.

Der **Top-Laner** beginnt ein Spiel zumeist alleine auf einer Lane, dies muss nicht zwingend, wie der Name vermuten lässt, die obere Lane sein. Eine der beiden langen Lanes, sprich Unten oder Oben wird ausgewählt. Charakteristisch für diese Rolle sind *Tanks*, *Kämpfer* und *Magier*. Weiters erhält der Top-Laner oft keine Unterstützung, weshalb es gut ist, wenn dieser über Fähigkeiten zur Flucht, oder zur Selbstheilung verfügt.

Der **Mid-Laner** ist gemeinsam mit dem ADCarry hauptverantwortlich für den Schaden. Er startet das Spiel in der mittleren Lane, da diese etwas kürzer ist und zugleich der Mid-Laner variabler in das Spiel eingreifen kann und schneller an den meisten Orten im Spiel ist. Der Mid-Laner spielt zumeist einen *Assassinen* oder *Magier*.

Der **Supporter** ist, wie auch schon in der Championkategorie beschrieben, hauptverantwortlich für die Sicht auf der Karte und unterstützt seine Team-Mitglieder so gut wie möglich. Ein Supporter spielt zumeist *Tanks* oder eben *Unterstützer*.

Der **ADCarry** (Attack Damage Carry) ist die Rolle des *Schützen*. Dieser ist in der Anfangsphase eher schwach, wird aber mit laufender Dauer des Spiels immer wichtiger. Der Supporter versucht deshalb, den ADCarry so gut es geht in der Anfangsphase zu unterstützen, da der ADCarry sehr gegenstands-abhängig ist und erst mit zunehmender Anzahl an Gegenständen an Stärke zunimmt.

Der **Jungler** bezeichnet den Spieler, der sich nicht auf einer der Lanes aufhält, sondern das Spiel im neutralen Dschungel bestreitet. Ziel des Junglers ist es, die neutralen Monster zu töten und gleichzeitig Präsenz in den Lanes zu zeigen. Dies nennt man *ganken*. Das bedeutet, dass der Jungler versucht, durch sein Erscheinen eine Überzahl zu bilden und überraschend aufzutauchen, um die Gegner zu töten. Für den Jungler werden *Tanks* und *Kämpfer* bevorzugt.

Selbstverständlich ist eine eindeutige Klassifizierung nicht immer möglich und nicht jeder Spieler lässt sich in eine Rolle drängen.

3.1.5 Strategie

In League of Legends entwicklen sich laufend neue Arten, wie das Spiel am effizientesten gespielt werden kann. Das nennt man Meta-Game. Die Strategie dahinter bleibt bei den diversen Meta-Games (Metas) zumeist sehr ähnlich, da sich ja das Ziel des Spiels nicht ändert.

In diesem Abschnitt wird kurz die gängigste Strategie beschrieben und auch versucht zu erklären, was sich dabei ständig ändert. Geprägt durch den professionellen E-Sports, der League of Legends umgibt, entwickelte sich im Laufe der Zeit eine Variante des Spiels, die von den meisten Spielern gespielt wird. Dabei wird versucht, eine gute Rolleneinteilung zu finden. Das bedeutet, dass in einem Großteil der Spiele ein Top-Laner, ein Mid-Laner, ein Jungler, ein ADCarry und ein Supporter zu finden sind. ADCarry und Supporter teilen sich die untere Lane. Der ADCarry darf dabei alle Minions töten, um möglichst viel Geld zu erhalten, während der Supporter versucht, den ADCarry zu unterstützen. Durch diese Rolleneinteilung ist auch das Aufgabengebiet der einzelnen Spielerklar abgegrenzt und die Spieler halten sich daran.

Betrachtet man die Meta, so ändert sich wenig an den Rollen oder den Aufgaben, jedoch verändern sich die Champions, die Gegenstände und die Beschwörerfähigkeiten, die die jeweiligen Spieler verwenden. Eine gängige Variante, bei der sich etwas mehr ändert, ist der sogenannten Lane-Swap, bei dem der Top-Laner mit dem ADCarry und dem Supporter die Lanes tauscht um so einen Vorteil zu erhalten: Um beispielsweise den gegnerischen

Turm schneller zu zerstören, oder einem ungeliebten Matchup aus dem Weg zu gehen. Jeder Champion hat Stärken und Schwächen gegenüber anderen, dies wird versucht auszumerzen.

Durch das Wählen der Champions entwicklen sich auch unterschiedliche Kompositionen der Teams. So setzt ein Team etwa auf eine Belagerungskomposition, die darauf spezialisiert ist, Türme zu zerstören, während ein anderes Team auf eine *Split-Push* Komposition baut. Das bedeutet, ein Spieler (der Top- oder Mid-Laner) befindet sich, zumeist alleine, in einer Lane und versucht, die Türme zu zerstören, während sich das restliche Team gemeinsam aufhält. Dabei wird auf die 1vs1 Stärke gewisser Champions gesetzt, denn sollten sich mehrere Spieler in den Kampf mit dem designierten Split-Pusher begeben, so haben die übrigen vier ein Überzahl-Spiel.

Solche und weitere Kompositionen existieren etliche. Die Profi-Spieler versuchen ständig, die Beste zu wählen oder Neue zu kreieren, wodurch sich das System ständig weiterentwickelt.

3.1.6 Rangliste

In LoL steht es jedem Spieler frei, Ranglisten-Spiele zu absolvieren. Dabei erhält man nach zehn Spielen eine Einteilung in Ligen und Divisionen. Zur Zeit gibt es sieben Ligen: Bronze, Silber, Gold, Platinum, Diamant, Meister und Herausforderer (in aufsteigender Reihenfolge). Die ersten fünf haben jeweils fünf Divisionen (1–5). Je höher die Liga und je niedriger die Division, desto mehr Ranglisten-Punkte hat der jeweilige Spieler gesammelt. Pro Spiel erhält/verliert ein Spieler eine gewisse Punkteanzahl je nach Sieg bzw. Niederlage. Erreicht man 100 Punkte in einer Division, so kann man durch den Gewinn einer Best-of-3 Serie in die nächste Division aufsteigen. Erreicht man in Division1 100 Punkte, so kann man durch einen Best-of-5 Sieg in die nächst höhere Liga aufsteigen.

In der Meister und Herausforderer Liga gibt es keine Divisionen. Nach Diamant1 erreicht man die Meister-Liga. Die besten 200 Spieler einer Region werden im 24h Rhythmus in die Herausforderer-Liga aufgestuft, bzw. die die nicht mehr unter den besten 200 sind, abgestuft.

Kurz gesagt heißt das, dass ein Spieler viele Spiele gewinnen muss, um aufzusteigen, weshalb man sagen kann: Je höher ein Spieler eingestuft ist, desto besser ist er auch.

3.1.7 Matchmaking

Matchmaking dient im Allgemeinen dazu, dass sich ein Spieler für ein Spiel anmeldet und anschließend wird er in eine Gruppe mit vier weiteren Spielern gebracht. Da League of Legends nur online gespielt werden kann und weiters ein Ranglisten-Spiel nur gegen zufällige ermittelte Gegner möglich ist, bleibt einem Spieler kein anderes Verfahren über. Daher stellt das Matchmaking

einen sehr wichtigen Teil des Spiels dar.

Das derzeitige Matchmaking funktioniert in drei Schritten:

1. Die eigene Spielstärke wird ermittelt. Für diesen Zweck verwendet Riot ein Rating, dass sich an das Elo-Rating der Schachwelt (siehe [24]) anlehnt und modifiziert. Je höher das Rating, desto besser der Spieler.
2. Geeignete Gegner werden gesucht: Das System sucht nach Spielern mit einem ähnlichen MMR-Wert.
3. Sind neun weitere Spieler gefunden, so werden alle Spieler zusammen in ein Spiel gebracht.

Dieses System birgt Stärken und Schwächen in sich. Für die meisten Spieler ist die Wartezeit eher gering, was sehr gut ist, jedoch wird nicht auf das Verlangen des Spielers, eine gewisse Rolle zu spielen, eingegangen.

3.2 Entscheidungsbäume

Entscheidungsbäume sind Bäume, die im Machine Learning und der Klassifikation eingesetzt werden können. Ein Entscheidungsbaum (siehe Abbildung 3.6) verfügt über folgende Eigenschaften:

- Er hat einen *Wurzelknoten*. Dies ist der Ausgangsknoten.
- Jeder interne Knoten repräsentiert ein *Attribut*.
- Jede Kante repräsentiert einen Test auf das Attribut des Elternknotens.
- Jedes Blatt repräsentiert eine der möglichen Klassen.

Jeder interne Knoten im Baum hat exakt eine eingehende Kante. Verfügt ein Knoten über keine ausgehenden Kanten, so wird er als Blatt bezeichnet. Alle anderen Knoten verfügen über zwei oder mehrere ausgehende Knoten. Bei einem binärem Baum hat jeder Knoten nur zwei ausgehende Kanten. In jedem Knoten wird über die Kanten auf das jeweilige Attribut getestet und daher bestimmt, wie der Baum weiter zu durchlaufen ist. Wird ein Objekt kategorisiert, so muss es den Baum von oben nach unten (top-down) durchwandern. Je nach dem Ergebnis entlang der Pfade landet das Objekt in einem Blatt-Knoten und gilt als klassifiziert.

Einer der größten Vorteile eines Entscheidungsbaums lässt sich damit sehr einfach erkennen. Der Baum ist leicht verständlich und zu interpretieren. Jede Route entlang des Baumes kann in einfachen Regeln formuliert werden.

Betrachten wir eine *Objektsammlung* M, die durch m Objekte gebildet wird. Jedes dieser Objekte verfügt über n Attribute, die in k_i Ausprägungen auftreten und in k Klassen fallen. Um einen Entscheidungsbaum bilden zu können, muss für M gelten, dass es mindestens zwei verschiedene Klassifikationen C enthält. Außerdem darf M keine Objekte enthalten, die bei gleichen Attributen einer unterschiedlichen Klasse angehören.

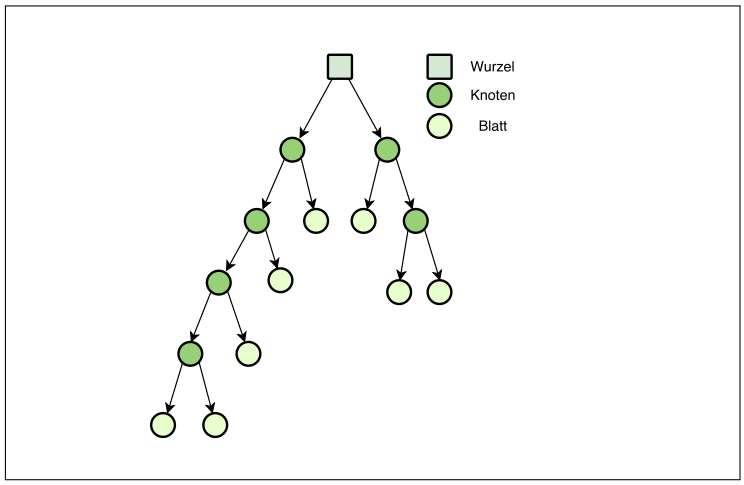


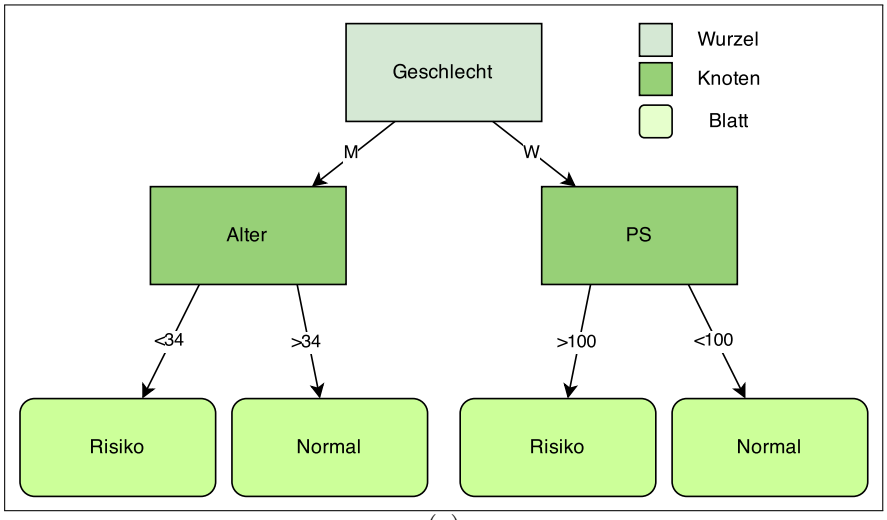
Abbildung 3.6: Beispielhafte Darstellung eines binären Baumes.

Geschlecht	Alter	PS	Klasse
M	22	150	Risiko
W	26	50	Normal
W	19	145	Risiko
M	35	120	Normal
W	45	90	Normal
M	24	120	Risiko
W	23	45	Normal

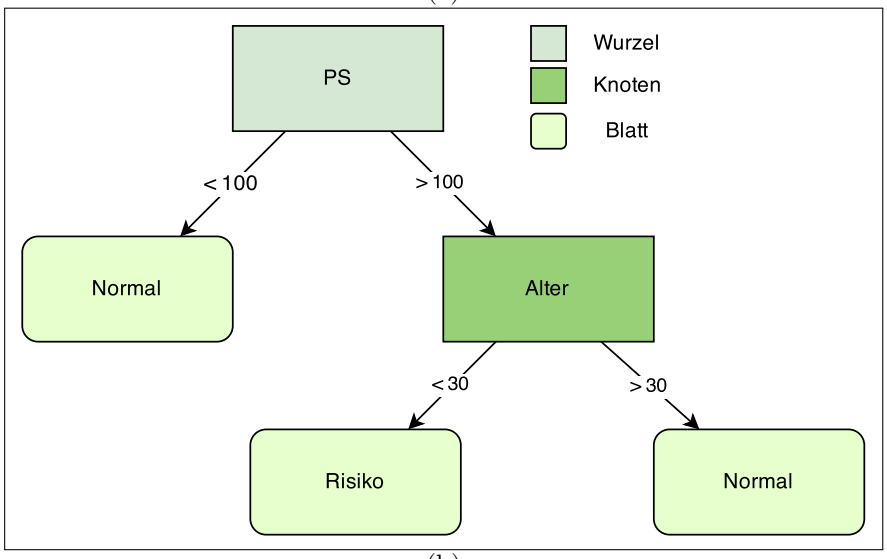
Tabelle 3.1: Entscheidungsbaum Beispiel: Versicherung (fiktive Werte).

Weiters werden die auftretenden Attribute mit $A_1, ..., A_n$ bezeichnet. Diese verfügen über unterschiedliche Ausprägungen $a_{i,1}, ..., a_{i,n}$. Als Beispiel dazu könnte man einen Baum betrachten, der der Entscheidung über Versicherungsklassen einer Autoversicherung dient. Dann würde etwa A_1 das Geschlecht sein und $a_{i,1}$ etwa männlich. Die Klassifikation wird durch C gegeben, das wiederum über unterschiedliche Werte $c_i, ..., c_n$ verfügt. Um bei dem oben genannten Beispiel zu bleiben, wäre das dann z.B. die Versicherungsstufe der Person.

Betrachten wir zum besseren Verständnis folgende Tabelle 3.1: Aus diesen Beispieldaten lässt sich ein Problem einer Versicherung ablesen. Verschiedene Personen benötigen für ihre Autos eine Versicherung. Vereinfacht dargestellt haben wir hier drei Attribute (Geschlecht, Alter der Person und PS des Autos) mit jeweils unterschiedlichen Ausprägungen und zwei Klassen c_1 = Risiko und c_2 = Normal. Diese beiden sollen eine entsprechende Versicherung für die Personen darstellen. Es lässt sich beispielsweise ablesen,



(a)



(b)

Abbildung 3.7: Zwei mögliche Bäume gebildet aus den fiktiven Daten in Tabelle 3.1.

dass ein Mann mit 22 Jahren und einem Auto mit 150 PS in die Versicherungsstufe Risiko eingeteilt wird. Aus dieser Trainingsmenge lassen sich unterschiedliche Bäume bilden, wie in Abbildung 3.7 gezeigt wird. Die Fragen, die sich nun stellen sind: Welcher der Bäume ist der Beste? Welche Bäume lassen sich noch bilden? Klassifiziert der Baum garantiert immer richtig? In der Literatur lassen sich unterschiedliche Ansätze finden, um einem Entscheidungsbaum eine gewisse Güte zu geben. In [12] lassen sich folgende finden:

- **Einfachheit des Baumes, bzw. Anzahl der Blätter:** Je nach Anzahl der Blätter lassen sich aus einem Baum mehr oder weniger Regeln bilden, die der Entscheidungsfindung dienen. Dies schlägt sich auch auf die Größe des Baumes nieder. Je weniger Blätter, desto kleiner und umgekehrt.
- **Die Höhe/Tiefe des Baumes** spielt eine Rolle. Wie bereits erwähnt, lässt sich ein Baum in einfache Regeln umwandeln und beim Durchwandern werden mathematische *UND* ($\wedge$)Verknüpfungen gebildet. Somit ist die maximale Regellänge definiert über die Höhe/Tiefe des Baumes.
- **Die Genauigkeit des Baumes**: Der Baum soll so gut wie möglich zu einem Ergebnis und einer erfolgreichen Klassifikation kommen.
- **Die externe Pfadlänge** ist die Summe der Länge von allen Pfaden des Baumes, von der Wurzel bis zu den Blättern [28].
- **Die gewichtete externe Pfadlänge** gleicht der externen Pfadlänge, mit einem Unterschied in der Berechnung: die Länge der Pfade wird mit der Anzahl der klassifizierten Objekte, die der Pfad repräsentiert, multipliziert. Das bedeutet, für die gewichtete externe Pfadlänge muss man zusätzlich zum Baum selber auch wissen, welche Anzahl an Objekten wie bestimmt wurde und diese in die Berechnung mit einfließen lassen.

Somit bleibt die Frage, wie ein Entscheidungsbaum am besten gebildet wird.

3.2.1 Lernen von Entscheidungsbäumen

Wie werden Entscheidungsbäume erstellt? Dazu gibt es verschiedene Varianten und Algorithmen. Zuerst wird das Verfahren ganz allgemein betrachtet, und anschließend speziell der Algorithmus *CART* (Classification and Regression Trees), da dieser im späteren Verlauf der Arbeit Verwendung findet. Die meisten Verfahren haben sehr ähnliche Grundzüge, weshalb diese in der Gesamtheit betrachtet werden können. Prinzipiell könnte man aus einer beliebigen Datenmenge eine große Anzahl an Bäumen erstellen und aus diesen dann den besten auswählen. Dies wäre aber aufgrund der hohen Komplexität nicht umsetzbar. Es gilt also die bestmöglichen Attribute zu finden, um einen möglichst genauen und gut passenden Baum zu erstellen. Hier unterscheiden sich die Verfahren untereinander. Während bei CART in der Regel nach dem Informationsgehalt und hierbei mit Hilfe des Gini-Index (siehe Abschnitt 3.2.3) getrennt wird, verwenden andere Algorithmen etwaige abweichende Auswahlverfahren. Der ID3-Algorithmus beispielsweise verwendet den sogenannten Information Gain (siehe Abschnitt 3.2.2).

Durch das jeweilige Auswahlverfahren (Split-Verfahren) werden die Attribute bewertet und das zu dem jeweiligen Zeitpunkt beste Attribut wird verwendet, um die Menge an Objekten zu teilen. Das bedeutet, das es sich

bei den meisten Algorithmen um sogenannte *greedy (gierige)* Algorithmen handelt, da nur der jeweilige Zeitpunkt betrachtet wird, und nicht die Folgen der Auswahl. Nach dem Finden eines passenden Attributs wird die Menge entsprechend der Werte aufgeteilt und das Verfahren wird weiterhin auf die gebildeten Teilmengen rekursiv angewandt.

Angenommen eine Menge S wird herangezogen. Jetzt soll S anhand des gewählten Verfahrens an einem Attribut geteilt werden. Als Attribute stehen $A_1, A_2, ..., A_i$ zur Verfügung. An jedem Knoten findet eine Teilung der Menge S in Teilmengen σ für die gilt $\sigma \subseteq S$. Am Wurzelknoten gilt $\sigma = S$. Bei der Wahl der Attribute soll nun dasjenige herangezogen werden, welches die Menge in möglichst gleichklassige Untermengen teilt.

Wichtig für diesen Schritt sind dabei die Stopp-Kriterien. Wann soll der Algorithmus mit dem unterteilen der Mengen aufhören? Wann ist ein weiteres Unterteilen nicht mehr nötig, bzw. bringt wenig ein? Ein Split soll also so lange stattfinden, bis entweder ein Stopp-Kriterium erfüllt wird, oder aber alle Objekte klassifiziert wurden. Gestoppt wird, wenn

- alle Objekte klassifiziert wurden,
- eine festgelegte maximale Tiefe des Baumes erreicht ist,
- ein Split keinen größeren Gewinn einbringt als ein vorher festgelegter Schwellwert oder
- wenn die Anzahl der Objekte kleiner ist als die Anzahl der möglichen Kindknoten.

Nach dem Auswählen des Attributs und dem Stoppen zu einem gegebenen Kriterium ist der Baum vollständig gebildet und klassifiziert entsprechend der gegebenen Objekte. Nun folgt ein Schritt, bei dem versucht wird, den Baum wieder auf eine gute Größe zu stutzen (to *prune*), sollte dieser möglicherweise zu groß geraten sein.

Ein Problem beim Erstellen der Bäume ist das *Overfitting*. Durch Störungen in den Daten kann es dazu kommen, dass ein Baum, der weniger an die Trainingsdaten angepasst ist, letzten Endes besser funktioniert als der Baum, der bis zur Perfektion getrieben wurde. Wird die gesamte Menge an verfügbaren Daten herangezogen und es zeigt sich, dass der Baum, der mit den Trainingsdaten eventuell noch ein paar Probleme hatte, mit den gesamten Daten besser funktioniert, so wird dies als *Overfitting* bezeichnet. Angenommen aus der Gesamtheit der Daten O wird ein Trainingsset $O_{tr} \subset O$ gebildet. Bei O_{tr} hat der Baum t_1 eine sehr geringe Fehlerrate. Nun jedoch existiert ein Baum t_2, welcher auf O_{tr} eine größere Fehlerrate, aber auf O eine geringere Rate aufweist, so wird von *Overfitting* gesprochen [10]. Aufgrund des *gierigen* Ansatzes der meisten Algorithmen, ist es oft schwer, ein vorzeitiges Stoppen zu erzwingen. Da es durchaus vorkommt, dass einige Splits relativ wenig zur Verringerung der Fehlerrate beitragen, aber nach diesen wieder ein guter Split auftaucht, ist es üblich, einen Baum sehr groß wachsen zu lassen, um ihn anschließend wieder zurechtzuschneiden [1].

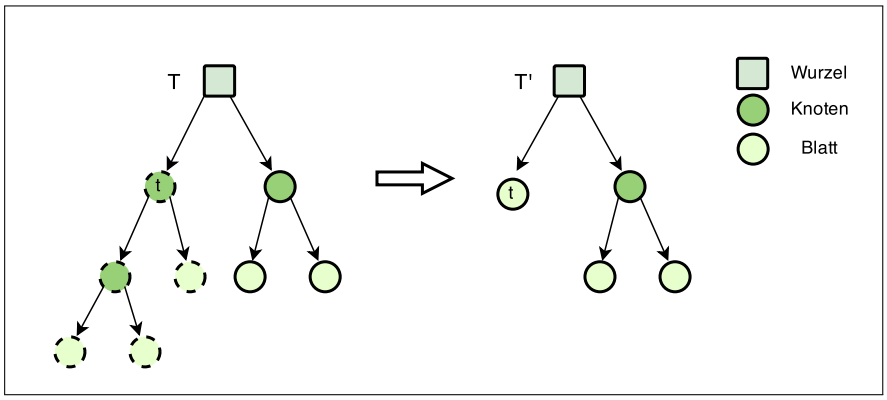


Abbildung 3.8: Schematische Darstellung des *Pruning*.

Das Zurechtstutzen des Baumes nach dem Erstellen wird allgemein auch als *Postpruning* bezeichnet. Von *Prepruning* wird gesprochen, wenn Stopp-Kriterien existieren, die das Wachsen eines Baumes verhindern können. Für das Postpruning werden unterschiedliche Kriterien herangezogen, die versuchen, die Fehlersumme auf den gesamten Datensatz zu verringern. Bei CART wird beispielsweise das Kosten-Komplexitäts-Kriterium (Cost-Complexity-Pruning) herangezogen. Nimmt man einen Baum T und einen inneren Knoten n, welcher nicht der Wurzelknoten und auch kein Blattknoten ist, so spricht man von pruning, wenn alle nachfolgenden Knoten von n in T gelöscht werden. Daraus resultiert ein neuer Baum $T\prime$ und n ist nun ein Blattknoten (siehe dazu Abbildung 3.8). Allgemein gesprochen wird versucht, anhand von statistischen Verfahren, "unwichtige" Äste des Baumes zu entfernen.

3.2.2 ID3

Der ID3 – Iterative Dichotomiser 3 – Algorithmus ist so etwas wie der Urvater der Entscheidungsbaum-Algorithmen. In [13] wurde darin die Erstellung eines Entscheidungsbaumes beschrieben. Dieser Algorithmus ist mittlerweile etwas überholt und wird zumeist durch neuere ersetzt, wie etwa den C4.5, der ebenfalls von Quinlan [14] entwickelt wurde, oder CART. ID3 arbeitet anhand des Informationsgewinns (Information Gains) um ein geeignetes Attribut auszuwählen. Diese Attribute können bei ID3 nur diskrete Wertbereiche annehmen. Die Idee hinter diesem Verfahren ist:

1. Ein Attribut, welches die Unsicherheit am meisten verringert muss gefunden werden.
2. Es folgt ein rekursiver Aufruf aller Werte des vorher gewählten Attributs.
3. Dies wird so lange wiederholt, bis nur mehr Objekte derselben Klasse

in einem Blatt zu finden sind.

Da nur diskrete Werte gehandhabt werden können, bedarf es des öfteren binärer Entscheidungen. Diese werden über Vergleichstests, wie *größer* $>$, *kleiner* $<$ usw. gebildet. In ID3 ist noch kein Pruning eingebaut, dies wird erst bei anderen Algorithmen verwendet.

Information Gain

Da der Informationsgewinn schon erwähnt wurde, wird hier nun näher auf diesen eingegangen. Um den Information Gain überhaupt bestimmen zu können, muss man vorher ein Maß betrachten, das dazu dient, den Informationsgehalt festzulegen, die *Entropie*. Die Entropie E einer Menge S definiert sich wie folgt:

$$E(S) = \sum_{k=1}^{K} p_k \log_2(p_k). \tag{3.1}$$

p_k ist in diesem Fall die Auftrittswahrscheinlichkeit der Klasse C_k in der Menge S. Der Informationsgehalt is umso größer, je seltener C_k auftritt. Der Information Gain G berechnete sich nun aus der Veränderung der Entropie. Es ist die Reduktion der Entropie, die durch die Aufteilung der Menge S anhand des Attributs A entsteht:

$$G(S, A) = E(S) - \sum_{k \in \text{Werte}(A)} \frac{|S_k|}{|S|} E(S_k). \tag{3.2}$$

Werte(A) ist die Menge von allen möglichen Werten für das Attribut A und S_v ist die Teilmenge von S, für welche gilt, dass A den Wert v hat. Die Berechnung der Entropie, die in dieser Formel verwendet wird, ist in der Formel 3.1 zu finden. Das bedeutet, es wird zuerst die Entropie im Datensatz berechnet und anschließend wird die Entropie der Teilmenge von S, wenn das Attribut A gewählt wurde, abgezogen. Der Bruch $\frac{|S_i|}{|S|}$ dient dabei der Gewichtung. ([10])

3.2.3 CART

Der CART (Classification and Regression Tree) Algorithmus wurde 1984 von Breiman et al. [3] erstmals publiziert. Dieses Verfahren arbeitet nur mit Binärbäumen. Das bedeutet, an jedem Knoten N befinden sich nur zwei Kanten: N_l, die linke und N_r die rechte Kante. Gleichzeitig heißt das, dass für jedes Attribut ein gewisser Schwellwert gefunden werden muss, um die Menge gut splitten zu können. Ein CART Split erfolgt also nach einem einfachen Prinzip. Vom Attribut A wird ein Wert i gewählt. Für ein Objekt m gilt also: *wenn Bedingung*(m) *dann links, sonst rechts. Bedingung*(m) ist in diesem Fall der Vergleich mit einem Schwellwert [15]. Dies kann auch zu einer schlechteren Lesbarkeit des Baumes führen, da es vorkommen kann,

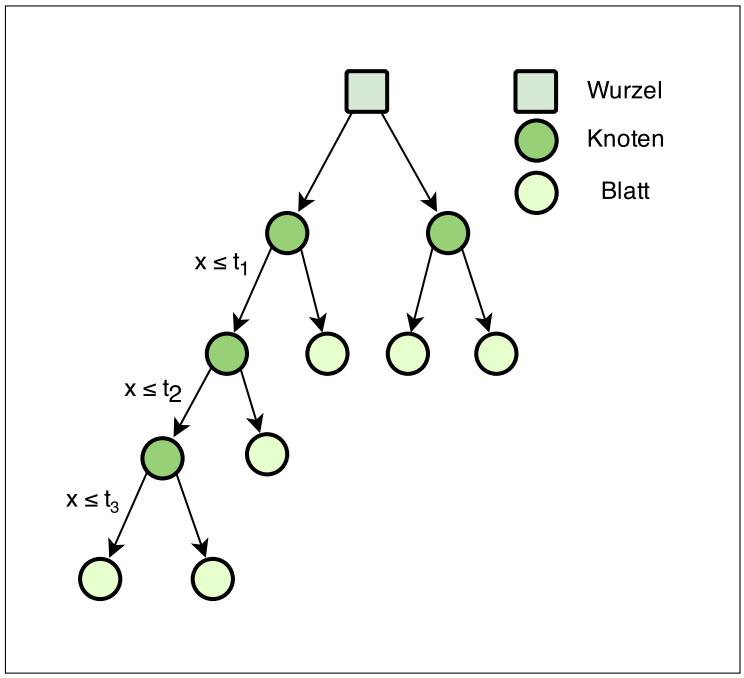


Abbildung 3.9: Mehrere Splits auf das selbe Attribut.

dass ein Baum mehrmals daselbe Attribut mit unterschiedlichen Schwellwerten prüfen muss, um zu einem passenden Ergebnis zu kommen. In Abbildung 3.9 lässt sich ein Beispiel dafür erkennen. Dabei wird das Attribut x mehrmals hintereinander auf unterschiedliche Grenzwerte t_1, t_2 und t_3 geprüft. Bei CART treten auch Regression Trees auf, wie der Name impliziert. Diese unterscheiden sich etwas zu den Classification Trees, da sie auch kontinuierliche Werte behandeln können. Einfach ausgedrückt, findet man mit Classification Trees heraus, zu welcher Gruppe ein Objekt gehört, während mit Regression Trees eine Vorhersage oder eine Einschätzung getroffen wird. Als Beispiele dafür würde man einen Preis für ein Haus, beruhend auf einem Trainingsset, mit einem Regression Tree einschätzen, oder einen Tumor mit Hilfe von Trainingsdaten und einem Classification Tree in gutartig oder bösartig einteilen können.

Weiters wird der Baum zur vollen Größe entwickelt und kein Prepruning vollzogen. Breiman et al. sind der Meinung, dass es keine Regel geben wird, die sicher stellt, dass wichtige Daten übersehen werden. Mit Hilfe des Kosten-Komplexität-Kriteriums (siehe Abschnitt 3.2.3) wird der Baum anschließend zurückgeschnitten. Außerdem entwickelt CART nicht nur einen Baum, sondern eine Anzahl an Bäumen, welche alle in Frage kommen. Mit Test-Daten wird dann der passendste Baum anschließend gefunden.

Gini Splitting

Das Splitting wird anhand des Gini-Kriteriums vollzogen. Dieses ermöglicht nur binäre Splits. Es wird bevorzugt, weil es sich schneller berechnen lässt. Die Berechnung der Unreinheit einer Menge S lautet

$$\text{Gini}(S) = 1 - \sum_{k=1}^{K} p_k^2, \tag{3.3}$$

wobei p_k, wie auch bei der Berechnung der Entropie, die Wahrscheinlichkeit darstellt, dass die Klasse C_k in der Menge S auftritt. Nach dem Splitten der Menge S in zwei Teilmengen S_1 und S_2 mit jeweils K_1 und K_2 Elementen erhält man folgende Formel zur Berechnung des Gini-Index:

$$\text{Gini}_{\text{split}}(S) = \frac{K_1}{K}\text{Gini}(S_1) + \frac{K_2}{K}\text{Gini}(S_2). \tag{3.4}$$

Das Attribut, welches über den geringsten $\text{Gini}_{\text{split}}(S)$ verfügt, wird verwendet.

Kosten-Komplexitäts-Kriterium

Da, wie bereits erwähnt, bei CART kein Prepruning durchgeführt wird, ist ein Postpruning vonnöten um den maximal gewachsenen Baum wieder zurecht zu stutzen. Dieses Postpruning trägt zwar zur Rechenintensität bei, jedoch kann man so sicher sein, dass alle wichtigen und guten Splits zu tragen kommen. Laut Bishop [1] ist beim Kosten-Komplexitäts-Pruning noch wichtig zu erwähnen, dass jeder Entscheidungsbaum auch als zwei-dimensionaler Raum betrachtet werden kann, der entsprechend der Splits aufgeteilt wird. Einem Blattknoten τ kann dann eine Region R_τ in diesem Raum zugeteilt werden. Siehe dazu Abbildung 3.10, in der ein Baum und der entsprechende zwei-dimensionale Raum dazu dargestellt sind. k_1, k_2, k_3 und k_4 stellen dabei Bedingungen dar, anhand derer die Splits durchgeführt werden. Die Regionen A, B, C und D können dabei zum Beispiel einer Klasse zugewiesen werden.

Nimmt man einen groß gewachsenen Baum T_0, so definieren wir jeden Unterbaum der in T_0 gebildet werden kann als, $T \subseteq T_0$. Wir nehmen alle Blattknoten als $\tau = 1, ..., |T|$, und jeder Blattknoten τ stellt eine Region R_τ mit N_τ Elementen dar. $|T|$ liefert uns die Nummer an Blattknoten des Baumes T. Das Pruning-Kriterium $C(T)$ wird dann folgendermaßen definiert:

$$C(T) = \sum_{\tau=1}^{|T|} Q_\tau(T) + \lambda|T|, \tag{3.5}$$

Für Q_τ wird bei der Regression die mittlere quadratische Abweichung verwendet, die lautet

$$Q_\tau(T) = \sum_{x_n \in R_\tau} \{t_n - y_\tau\}^2, \tag{3.6}$$

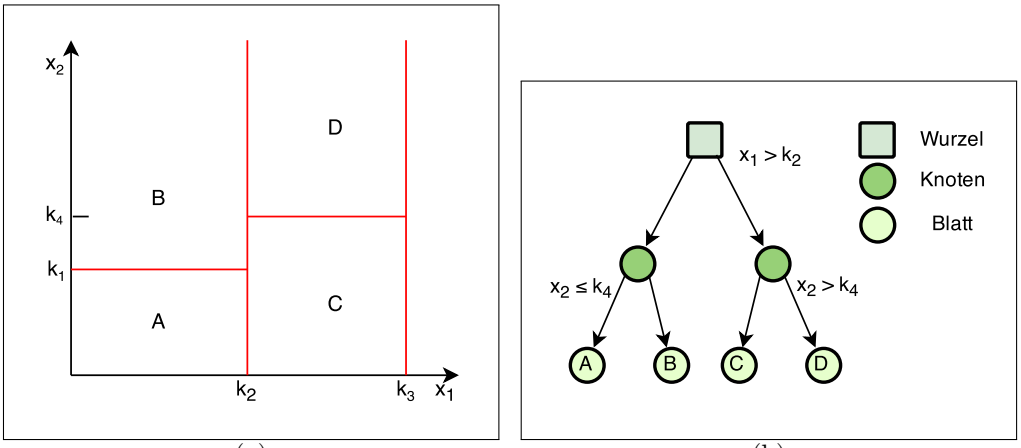


Abbildung 3.10: Einteilung des zwei-dimensionales Raumes in vier Regionen (a). Der zugehörige Entscheidungsbaum (b).

wobei y_τ die optimale Einschätzung für eine Region R_τ ist

$$y_r = \frac{1}{N_\tau} \sum_{x_n \in R_\tau} t_n. \tag{3.7}$$

Der Wert für λ wird über Kreuzvalidierung herausgefunden. In Gleichung 3.5 wird die Summe der Fehler, dargestellt durch Q_τ, mit der Komplexität des Baumes summiert. Als Maß der Komplexität gilt dafür alleine die Anzahl der Blattknoten.

Bei einem Klassifikations-Verfahren wird die mittlere quadratische Abweichung durch ein passenderes Verfahren ausgetauscht. Bishop [1] beschreibt dafür die Kreuzentropie und den Gini-Index. Zweiteres wurde bereits erwähnt (siehe Abschnitt 3.2.3). Die Kreuzentropie lässt sich folgendermaßen definieren: Angenommen $p_{\tau k}$ ist der Anteil an Objekten der Klasse k in der Region R_τ, so gilt

$$Q_\tau(T) = \sum_{k=1}^{K} p_{\tau k} \ln(p_{\tau k}). \tag{3.8}$$

Kapitel 4

Implementierung

4.1 Einleitung

Dieses Kapitel widmet sich der Projektarbeit, die im Rahmen dieser Arbeit entstanden ist. Ein Hauptaugenmerk liegt dabei auf dem Arbeiten mit "echten" Daten. Zu Beginn dieser Arbeit stand die Frage: Kann die Software die Daten genauso verarbeiten wie ein Mensch? Woher kommen also diese Daten? Hier dient Riot Games als Lieferant. Wie bereits erwähnt, ist Riot Games der Hersteller des Spiels League of Legends, das sich seit einiger Zeit an großer Beliebtheit erfreut. Das Unternehmen hat kürzlich ihre eigene API (Riot-Api) vorgestellt, mit der es möglich ist, auf sämtliche Spiele und deren zugehörige Information zuzugreifen. Dies deshalb, weil sich Websites schon vorher darauf spezialisiert hatten, dass Nutzer andere Spieler suchen können, um über deren Präferenzen und deren Können Bescheid zu wissen. Auch die Ranglisten-Platzierung anderer Spieler steht oft im Vordergrund, um über die Talente des Gegners Bescheid zu wissen. Um an diese Daten zu gelangen, benutzten einige Sites "Scraping"-Algorithmen, die die Website von Riot verlangsamten und für erhöhten Traffic sorgten. Um dieses Problem zu unterbinden, stellte Riot eine eigene Schnittstelle [20] zur Verfügung.

Diese Api sollte also die Grundlage für das Projekt sein. Was soll allerdings damit gemacht werden? In Abschnitt 3.1.7 wurde bereits über das derzeit bestehende Matchmaking geschrieben. Wie darin bereits erwähnt wurde, hat das System einige Schwächen. Es kommt sehr häufig zu Diskussionen und Streit, die leicht zu verhindern wären. Bei einem weiteren Spielmodus von LoL, wählt der Spieler bereits vorher seinen Champion und seine Rolle und wird anschließend mit anderen Spielern und anderen Rollen zusammengeführt. Dabei gibt es keinerlei Streitereien, was welcher Spieler macht. Allerdings gibt es bei diesem Modus keine Rangliste. Die Idee war also, das Matchmaking für Ranglisten-Spiele (siehe Abschnitt 3.1.6) zu erweitern.

Aufgrund unterschiedlicher Strategien, wie League of Legends gespielt

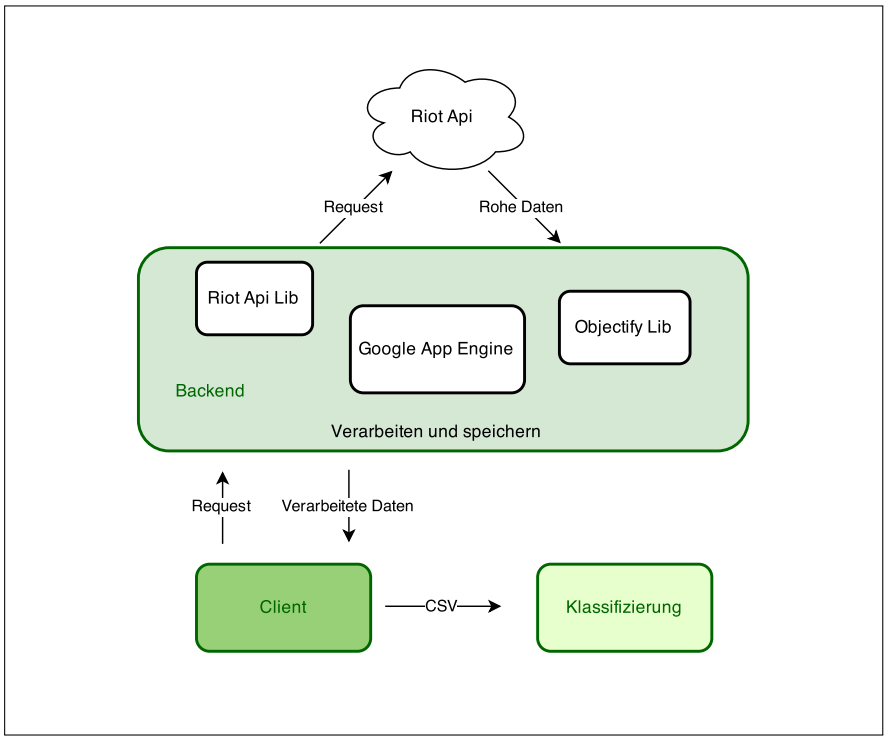


Abbildung 4.1: Skizzierte Darstellung des Projekts.

wird, lassen sich bei Spielern Rollen festlegen, die sie bevorzugen. Diese Rollen lassen sich als Mensch, der sich mit dem Spiel beschäftigt, anhand einiger Regeln und anhand der Position des Spielers relativ leicht bestimmen. Doch kann auch ein Programm diese Rollen ermitteln?

Anschließend an eine Einteilung der Präferenz, kann diese gewonnene Information genutzt werden. Spieler, die sich eingereiht haben und auf ein Spiel warten, können entsprechend mit anderen Spielern gruppiert werden. Somit sollten auch bei Ranglisten-Spielen die Streitereien etwas unter Kontrolle gehalten werden können.

Soweit der Gedanke und die Idee des Projekts. Für die Umsetzung wurden mehrere Teil-Programme entwickelt (siehe Abbildung 4.1):

- Ein Backend, zum Abrufen, Bearbeiten und Speichern der Daten,
- eine Android App, als Interface zu dem Backend und
- ein Python-Programm, das die Klassifizierung der Daten vornimmt.

Diese drei Programme arbeiten vernetzt untereinander. Das Backend bietet den Zugriff auf die Daten, die mit Hilfe der App als CSV-File gespeichert werden können. Dieses wiederum wird mittels Python Programm verwendet und analysiert, um letztendlich zu einem Ergebnis zu kommen.

Als Entwicklungsumgebung wurde Android Studio gewählt, sowie Py-

Charm. Beide stammen vom Software-Hersteller Jetbrains. Beziehungsweise ist Android Studio die von Google erstellte IDE (integrated development environment), die, wie der Name schon sagt, zur Entwicklung von Android-Apps dient. Sie basiert auf IntelliJ von Jetbrains und sollte verwendet werden, da sie dem Entwickler die Arbeit mit Android sehr erleichtert. Die Google App Engine, die für das Backend verwendet wird, ist auch integriert und lässt sich sehr leicht mit einer App kombinieren. PyCharm ist eine Umgebung für Python Programmierung.

4.2 Backend

Hier folgt eine Erklärung des Backends: Wie gearbeitet wurde und welche Mittel zur Hilfe genommen wurden, um zu einem funktionierenden Programm zu gelangen. Als Programmiersprache wurde Java gewählt. Als Plattform für das Backend sollte eine Möglichkeit gewählt werden, die unterschiedliche Client-Varianten zulässt, da zu Beginn noch nicht klar war, welche Art von Client erstellt werden soll. Aufgrund der hohen Vielfalt und der unterschiedlichsten Vorteile, war dies durchaus hersaufordernd.

Letztendlich fiel die Entscheidung auf die Google App Engine Plattform. Dieses von Google zur Verfügung gestellte Framework lässt den Nutzer auf einfache Art und Weise ein Web-Service erstellen. Der große Vorteil dieses Frameworks zeigt sich bei einem Blick auf die benötigte Infrastruktur. Der Benutzer braucht keinerlei Server oder ähnliches, um das Service zu hosten. All dies wird von Google übernommen. Natürlich muss dafür bezahlt werden, jedoch steht die Bezahlung im Verhältnis zu den Zugriffszahlen. Das bedeutet, dass bei einem Projekt zu einer Masterarbeit keine Kosten anfallen sollten, da sich die Zugriffe sehr niedrig halten.

Zur Kommunikation zwischen Client und Backend stellt die App Engine unterschiedliche Möglichkeiten zur Verfügung. Einerseits kann eine RESTful[1] Api erzeugt werden. Jedoch stellt die App Engine auch die Möglichkeit zur Verfügung, Client Libraries zu generieren. So kann für Javascript, iOS und auch Android eine Client Library erzeugt werden, die in dem jeweiligen Projekt eingebunden wird und somit eine Kommunikation erlaubt. Dabei werden die benötigten Objekte und Klassen erstellt und können verwendet werden. Der große Vorteil dieser Libraries ist die Zeitersparnis. Beim Arbeiten mit JSON Objekten und einer RESTful Api müsste zuerst noch Arbeit in das Parsen usw. gesteckt werden.

Das Backend stellt das Bindeglied zwischen Riot-Api und der Verarbeitung der Daten dar. Daher war einer der wichtigsten Schritte die Riot-Api einzubinden. Diese ist eine RESTful Api die mit JSON Objekten arbeitet.

[1] ***RE****presentational* ***S****tate* ***T****ransfer* ist ein Architekturprinzip für Web Services. Jede Resource des Services ist über eine eindeutige URI erreichbar. Der Client muss zum Austausch der Daten nicht über den Service Bescheid wissen.

Eine sehr gute Dokumentation steht ebenfalls zur Verfügung. Wie bereits erwähnt, setzt das Arbeiten mit JSON Objekten ein Parsen und ein Erstellen von Java-Objekten voraus, um effizient mit den Daten umzugehen. Um auch hier effizient zu arbeiten wurde eine freie Library von Github [23] verwendet. Diese übernimmt das Erstellen der Objekte und das Absetzen der Requests. Da dieses Projekt nicht zu kommerziellen Zwecken dient ist die Abhängigkeit von einer anderen Software-Bibliothek nicht weiter tragisch. Auch etwaige Performance Optimierungen sind nicht vonnöten.

Somit sind die Zugriffe sicher gestellt und die Daten können gesammelt werden. Als nächsten Schritt gilt es, die Daten entsprechend zu filtern und auszusortieren und für eine Weiterverarbeitung aufzubereiten. Als kurze Erklärung dazu: Abgefragt wurde eine *Matchhistory*. Dabei erhält man Zugriff auf die letzten zehn Ranglisten-Spiele des gesuchten Spielers. Für jedes dieser *Matches* wurden die Details einzeln abgefragt, um Zugriffe auf Daten, wie etwa die Zahl der getöteten gegnerischen Champions, oder den gesamten verursachten Schaden zu erhalten.

Dieser Schritt ist nun sehr entscheidend. Für eine gute Klassifikation benötigt man die richtigen Daten. Es werden die wesentlichen Attribute gesammelt und kleine Berechnungen durchgeführt. Beispielsweise ist es interessant zu wissen, wie viele Minions ein Spieler getötet hat, um über die Rolle Bescheid zu wissen. Allerdings kann ein Spiel unterschiedlich lange dauern, weshalb es zu einer Verzerrung kommt. Daher wurde das Verhältnis der getöteten Minions im Vergleich zum restlichen Team berechnet. Dabei sollte ein Schütze und ein Mid-Laner eine relativ hohe Zahl als Ergebnis haben. Mehr zu den gewählten Attributen wird in Abschnitt 5.2 geschildert.

Nach der Aufbereitung der Daten werden diese gespeichert. Die verwendete Datenbank dafür ist ein Open Source Projekt *objectify* [25]. Objectify bietet eine Schnittstelle auf den Google App Engine Datastore. Der Datastore bietet eine Möglichkeit, Objekte als solche zu speichern. Allerdings gibt es dazu einige Einschränkungen. Objectify erleichtert dies sehr. Über Annotationen können eigene Klassen bearbeitet werden. Diese können anschließend gesichert und abgefragt werden. Zusätzlich zu den Basisfunktionen bietet Objectify noch viele weitere Vorteile und ist einfach erlernbar.

Zusammenfassend dient das Backend als Schnittstelle zwischen Daten und Client und arbeitet in drei Schritten:

1. Zugriff und Abfragen der gewünschten Daten,
2. Filtern und Verarbeiten der erhaltenen Informationen und
3. Speichern und Ablegen der erstellten Objekte.

4.3 Client – Android App

Aufgrund der guten Integration zwischen App Engine und Android im Android Studio, war die Entscheidung für den Client einfach: eine Android

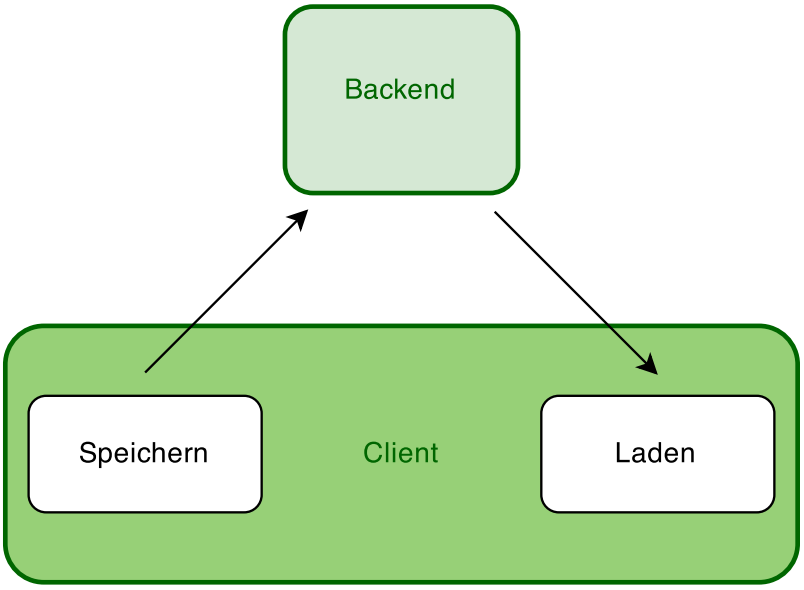


Abbildung 4.2: Darstellung des Ablaufs im Client.

Applikation. Die Aufgabe des Clients ist, eine visuelle und einfach bedienbare Schnittstelle zwischen User und Backend herzustellen.

Das Backend lässt zwei Zugriffe zu: Einerseits das Speichern eines Spielers (im englischen *Summoner*). Andererseits das Laden eines Spielers aus der Datenbank, sofern dieser schon vorhanden ist. Für beide Operationen wird eine Name sowie die zugehörige Region der Person benötigt. Aufgrund der großen Beliebtheit von League of Legends und der Tatsache, dass es global gespielt wird, können Spieler zwischen unterschiedlichen Regionen wählen, in denen sie spielen möchten. Diese richten sich nach der geographischen Lage der Person und der Lage des Servers für die Region. Als Beispiel existieren die Regionen North America, Europe West, Europe Nordic & East, Latin America North, Republic of Korea und einige weitere. Mit Hilfe dieser zwei Parameter lässt sich der Summoner eindeutig finden. Die für das Speichern benötigten Prozesse werden bis auf die Eingabe des Namens und der Region vollständig in dem Backend abgehandelt. Beim Laden wird die erhaltene Antwort als CSV-File auf dem Smartphone abgelegt. Dazu werden lediglich die Daten in das File geschrieben und dieses mit einem Timestamp versehen und abgelegt. Dadurch kann es nicht verwechselt werden und ist für die weitere Auswertung bereit.

Das Aufgabengebiet des Clients ist sehr limitiert (siehe Abbildung 4.2) und auch die Ausführung beschränkt sich nur auf das Wesentliche. Wie in Screenshot der Applikation 4.3 zu sehen ist, verfügt diese nur über eine

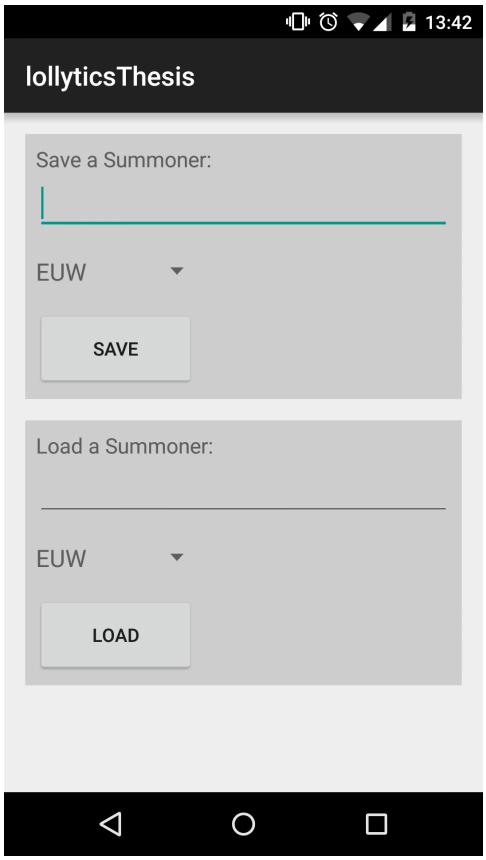


Abbildung 4.3: Screenshot der minimalistischen Client-App.

View, auf der die beiden Aufgaben abgehandelt werden können.

Beide Requests, sowohl der zum Speichern, als auch der zum Laden eines Spielers aus der Datenbank, werden asynchron abgesetzt. Wenn das Backend die jeweilige Aufgabe erledigt hat, sendet es eine Antwort zurück zum Client. Für eine bessere Handhabung wurden hier zwei kleine Pop-Ups eingebaut, welche dem User mitteilen, dass die Aufgaben erfolgreich (oder auch nicht) erledigt wurden.

4.4 Python Programm zur Klassifizierung

Die Klassifizierung der Daten wird vollständig in Python durchgeführt. Für die Klassifizierung und Machine Learning existieren eine Reihe von Bibliotheken. Diese sind zumeist gut getestet und hoch optimiert. Recherchen zeigten, dass es in den meisten Programmiersprachen ein existierendes Projekt gibt, welches die gängigen Algorithmen implementiert. Als Voraussetzung war wichtig, das die Api klar zu verstehen sein muss und es genügend

Unterlagen für ein Selbststudium geben sollte.

Die Wahl fiel dabei auf *scikit-learn*, eine Python-Library, welche ein Open Source Projekt ist. Scikit hat eine sehr gute Dokumentation und es lassen sich viele Lernunterlagen finden. Weiters lässt es sich gut einbinden und schnell verwenden.

Scikit bringt eine Vielzahl an Algorithmen mit sich. Für das Projekt standen viele Klassifizierungs-Algorithmen zur Auswahl. Die Wahl fiel auf die Entscheidungsbäume (engl. Decision Trees). Scikit verwendet dabei laut eigenen Angaben eine optimierte Version des CART Algorithmus [26]. Die Entscheidung fiel auf Decision Trees, weil diese ein leicht verständliches Modell bilden und nach Regeln arbeiten. Dabei ging es um die Frage: Wie würde eine Person, die sich mit League of Legends beschäftigt, andere Spieler kategorisieren? Dabei wurden einige Regeln festgestellt. Interessant war zu sehen, welche Regeln der Algorithmus aufstellt, um zu einem Ergebnis zu kommen und wie diese den Regeln einer Person gegenüber stehen. Außerdem lassen sich die einzelnen Schritte, die für eine Klassifizierung benötigt werden, gut nachvollziehen. Im Vergleich zu anderen Algorithmen, wo Vektorräume betrachtet werden usw., ist ein Entscheidungsbaum verhältnismäßig leicht zu verstehen.

Bezüglich der Regeln und der Entscheidungen, die getroffen werden, folgt ein kleines Beispiel. Betrachtet man die Datensätze, so wird eine hohe Anzahl an getöteten Minions die möglichen Klassen in zwei Gruppen teilen: AD Carry, Top-Laner und Mid-Laner, sowie Jungler und Supporter. Erstere sollten über eine hohe Zahl, Zweitere über eine verhältnismäßig niedrige Zahl verfügen. Nimmt man nun die Unterteilung in Jungler und Supporter her, so lassen sich diese anhand der gewählten Beschwörerfähigkeiten leicht unterscheiden. Ein Jungler wird immer *Zerschmettern* wählen, während ein Supporter sehr oft *Erschöpfen* vorzieht. Auch die Anzahl der platzierten Wards kann hier verwendet werden. Diese sollte beim Supporter sehr hoch sein. Genau aufgrund solcher Festellungen und Regeln ist es sehr interessant, wie ein Entscheidungsbaum bzw. der vorhandene Algorithmus vorgeht, um die entsprechenden Unterscheidungen zu machen. Diese kurzen Beispiele waren selbstverständlich sehr optimierte Varianten. In niedrigeren Ranglisten, in denen mit weniger Strategie vorgegangen wird, wird es ungleich schwieriger werden, die richtigen Entscheidungen zu treffen.

Was die Umsetzung betrifft, so ist die Klassifizierung sehr leicht programmiert. Wichtig ist zu Beginn, dass ein CSV-File eingelesen wird. Dem Programm muss mitgeteilt werden, um welche Datentypen es sich handelt. Als nächster Schritt werden die Klassen, welche klassifiziert werden sollen, festgelegt. In diesem konkreten Fall wird in dem CSV-File eine Spalte erstellt, die *Target* heißt, in der die jeweiligen Klassen aufgelistet sind.

Nach dem Benutzen des Clients erhält der User ein CSV in dem pro Zeile ein Datensatz steht, welcher ein Spiel des gesuchten Summoners repräsentiert. Anschließend muss eine Kategorisierung der Rolle durchgeführt

werden. Dazu wird die entsprechende Rolle in der Spalte notiert. Dies ist für den weiteren Verlauf sehr wichtig.

Zurück zur Implementierung in Python. Die eingelesen Daten werden zu einem Array gewandelt, welches für den Baum benötigt wird. Der Klassifizierer arbeitet in zwei Schritten: Zuerst wird gelernt und als zweites das Gelernte verwendet, um neue Daten zu klassifizieren. Im Programm wird hierzu ein Teil der gesamten Daten herangezogen um den Baum lernen zu lassen. Anschließend wird den Verbliebenen eine Klasse zugewiesen. Scikit lässt den Benützer beim Erstellen des Baumes die Wahl zwischen Gini und Entropie als Split-Kriterium.

Abschließend lässt Scikit den User noch einige statistische Daten auslesen. So lässt sich eine *Confusion Matrix* ausgeben und auch der Baum lässt sich drucken, wodurch man eine grafische Darstellung des erstellten Baumes erhält. Dazu wird eine .dot Datei erstellt. Für diese gibt es diverse Möglichkeiten, um daraus beispielsweise ein png zu erstellen. Im Rahmen dieser Arbeit wird dafür ein einfaches Konsolen-Kommando gewählt. Dazu muss Graphviz installiert sein. Graphviz ist eine Open-Source Visalisierungs-Software[2]. In Abbildung 4.4 ist ein während der Arbeit entstandener Baum zu sehen. Dieser zeigt einen kleinen, mittlerweile schon älteren Zwischenstand. Die fertigen Bäume sind sehr groß, weshalb sie nicht in der Arbeit abgebildet werden. Sie befinden sich auf der, der Arbeit beigelegten, CD.

[2]http://www.graphviz.org

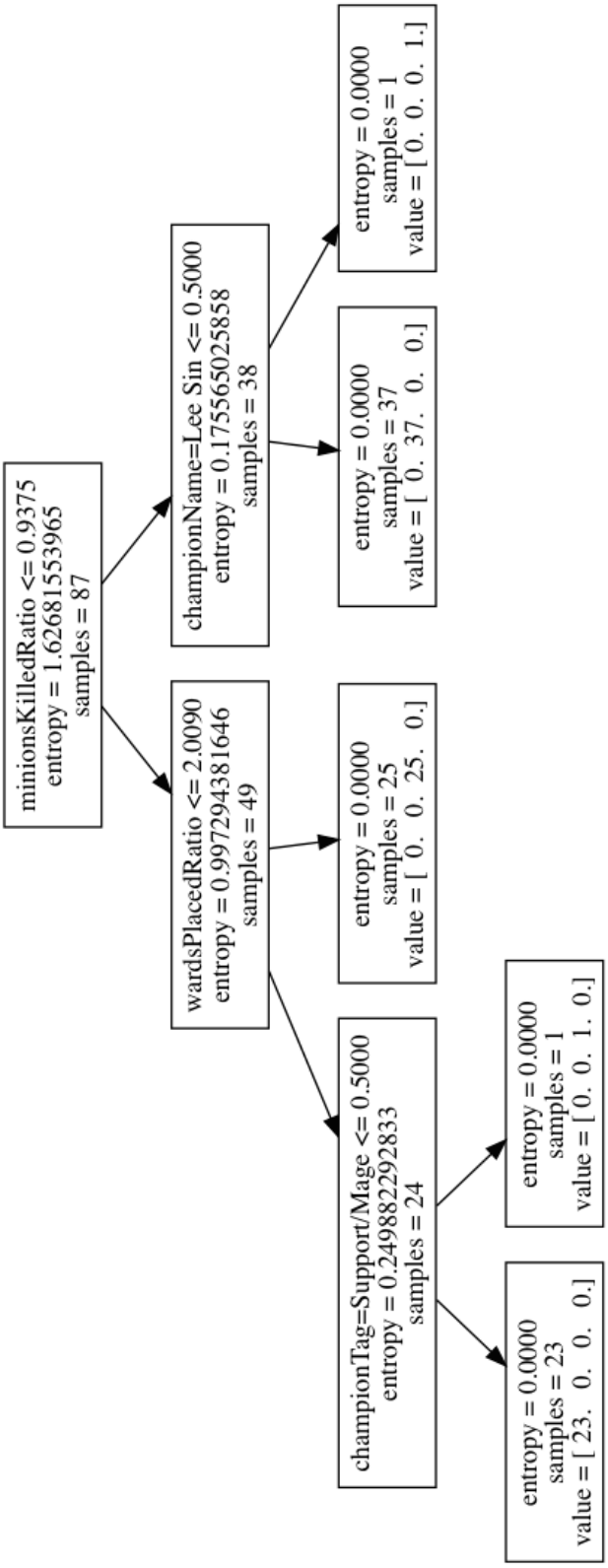


Abbildung 4.4: Grafische Darstellung eines Baumes

Kapitel 5

Ergebnisse

Dieser Teil der Arbeit widmet sich der Auswertung der Ergebnisse und den Erkenntnissen. Zu der bereits dargelegten Theorie erfolgt der praktische Teil und die Erprobung des Projekts. Es folgen Ausführungen über Auswahl, Filterung, sowie Auswertung der Daten. Zu guter Letzt werden Schlüsse über das vorhandene Ergebnis gezogen.

Ziel der Auswertung ist es herauszufinden, ob mit den gegebenen Daten eine Klassifizierung möglich ist und wie exakt diese arbeitet. Weiters ist es interessant zu sehen, wie die unterschiedlichen Spielstärken die Ergebnisse verändern und ob es sinnvoll ist, zwischen guten und schlechten Spielern Rückschlüsse zu ziehen.

Das Programm wird verwendet um die Rolle eines Spielers in einem einzelnen Spiel zu bestimmen. Für das weitere Vorgehen, müssten alle vorhergesagten Spiele betrachtet werden und dem Spieler seine Vorlieben zugeteilt werden. Dieser weitere Schritt wird nicht vollzogen. Es wird nur die Einteilung der Rollen in den einzelnen Spiele betrachtet und daraus Schlüsse gezogen. Wichtig ist eine exakte Arbeitsweise des Programms.

5.1 Auswahl der Daten

Ein sehr wichtiger Punkt in der gesamten Analyse findet bereits zu Beginn statt. Welche Daten werden verwendet? Aufgrund der Rangliste lässt sich bei League of Legends eine Einteilung der Spielstärke der einzelnen Personen erkennen. Die Spieler sind in die unterschiedlichen Ligen eingeteilt. Auf Websites wie op.gg[1] oder lolking[2] lassen sich Spieler suchen und deren Platzierung lässt sich ablesen. Außerdem kann man die Liga betrachten und somit weitere Spieler mit ähnlicher Platzierung finden.

Um eine möglichst repräsentative Stichprobe zu nehmen, wurden jeweils 250 Spiele aus den Ligen Bronze, Silber, Gold, Platinum, Diamant sowie 250

[1] http://www.op.gg
[2] http://www.lolking.com

aus Meister und Herausforderer gemeinsam. Das bedeutet insgesamt werden 1500 Spiele betrachtet. Um zu diesen Informationen zu kommen, wurden jeweils 25 Spieler gesucht und deren zehn letzte Ranglisten-Spiele entnommen. 25 Spieler sind Profis, die in der aktuellen offiziellen E-Sports-Liga spielen, die weiteren 125 wurden zufällig ausgewählt. Dafür wurden willkürlich Spielernamen aus den jeweiligen Ligen genommen. Alle Spieler befinden sich in der Region Europe West. Dies war wichtig, um sicher zu gehen, dass nicht unterschiedliche Meta-Strategien aktuell sind. Gerade in Regionen wie Korea kommt es öfters zu anderen Strategien. Eine Untersuchung diesbezüglich wäre sehr interessant, würde aber den Rahmen dieser Arbeit übersteigen.

Die Profis wurden ausgewählt, um eine Gruppe an Daten zu haben, die der professionellen Meta entsprechen. Die Annahme dahinter ist, dass jeder Spieler versucht, die beste Strategie zu wählen und der aktuellen Meta zu entsprechen, in der Hoffnung, dadurch einen Vorteil zu haben. Das würde bedeuten, dass viele Spieler versuchen, ähnlich zu den Profis zu spielen, weil diese als ausgezeichnete Spieler gelten. In der Auswertung wird auf diesen Punkt eingegangen. Weiters gilt die Hypothese, dass es in den unteren Ligen zu einer deutlicheren Abweichung kommt. Woraus sich schließen ließe, dass schlechtere Spieler auch aufgrund der falschen Herangehensweise an das Spiel schlechter sind. Aber hierzu dürften auch noch viele weitere Gründe kommen. Auch eine Erforschung in diese Richtung wäre spannend.

Zusätzlich zu der Klassifizierung an sich, finden sich also noch weitere interessante Themen, die mit den Daten evaluiert werden können.

5.2 Filterung der Daten

Riot stellt dem Benutzer ihrer Api eine große Zahl an Informationen zur Verfügung. Über query-Parameter lässt sich bestimmen, welche Daten inkludiert werden. Beispielsweise lässt sich auch die In-Game Position des Spielers zu jedem Zeitpunkt während des Spiels erfragen. Bei dieser Datenflut ist es daher sehr wichtig, diese auszusortieren. Aufbauend auf die Art, mit der ein erfahrener LoL-Spieler die Kategorisierung vornehmen würde, wurden die Attribute ausgewählt. Dabei wurden folgende Informationen gefiltert und verwendet:

- **Name des Champions**: Dieser dient der Identifizierung des gespielten Champions.
- **Champion Schlagwörter**: Die Schlagwörter geben Informationen zur Art des Champions, z.B. Mage, Tank, Marksman usw. Diese sollen dem Programm helfen. Als Spieler weiß man von einem Champion, wofür dieser genutzt werden kann.
- **Beschwörerfähigkeit 1 und 2**: Die beiden Beschwörerfähigkeiten geben sehr viel Auskunft über die Rolle. Sie sind für die Kategorisierung durch einen Menschen sehr wichtig.

- **Gegenstand 1–7**: Die gekauften Gegenstände des Spielers bestimmen die Art, wie der Champion gespielt wird.
- **Anteil der getöteten Minions verglichen mit dem Team**: Diese Kennzahl bestimmt, ob der Spieler in einer Rolle ist, die viele Minions töten darf.
- **Anteil der getöteten neutralen Minions verglichen mit dem Team**: Damit ist es möglich, einen Jungler zu identifizieren, da dieser vorwiegend neutrale Minions töten sollte.
- **Anteil des magischen Schadens verglichen mit dem Team**: Dies zeigt, ob der Champion magischen oder physischen Schaden verursacht.
- **Anteil des physischen Schadens verglichen mit dem Team**: siehe oben.
- **Gesamter verursachter Schaden verglichen mit dem Team**: Dies zeigt, ob der Champion einer der Hauptschaden-Verursacher ist.
- **Anteil der platzierten Wards verglichen mit dem Team**: Ein Spieler, der in einer Unterstützungsfunktion agiert, sollte tendenziell mehr Wards platzieren, als andere Rollen. Zum Beispiel der Jungler und der Supporter.
- **Anteil des erlittenen Schadens verglichen mit dem Team**: Zeigt an, wie viel Schaden ein Spieler während eines Spieles erlitten hat. Ein Tank (meist Top-Laner oder Jungler) sollte mehr Schaden einstecken als beispielsweise ein ADCarry, auch schon alleine deswegen, weil er über mehr Leben verfügt.

Insgesamt wurden also 18 verschiedene Attribute ausgewählt und für die Klassifizierung aufbereitet. Kurz noch die Erklärung wie ein Spieler vorgehen würde:

1. Zu Beginn wird der Champion betrachtet und anhand des Wissens über die Fähigkeiten, bzw. aus der Erfahrung einiger Spiele, entschieden, welche Rollen für den Champion am geeignetsten sind.
2. Die Beschwörerfähigkeiten werden als nächstes herangezogen. Die Beschwörersprüche Teleportation, Zerschmettern, Heilen und Erschöpfen sind sehr charakteristisch für gewisse Rollen. Ein ADCarry verwendet sehr oft Heilen. Ein Jungler üblicherweise Zerschmettern, da dieses viel Schaden an Minions verursacht usw.
3. Als nächstes werden die Gegenstände analysiert. Welche Werte diese erhöhen und über welche Fähigkeiten diese verfügen.
4. Die Anzahl der getöteten Minions lässt auch noch weitere Schlüsse zu.
5. Zu guter Letzt, wenn die Rolle noch nicht geklärt werden kann, muss die gesamte Teamkomposition betrachtet werden, um endgültig zu einem Ergebnis zu gelangen.

Klarerweise wird sich die Art, wie das Programm vorgeht, von dieser Weise unterscheiden. Spannend ist, welches Attribut von dem Algorithmus als wichtigstes eingestuft wird. Die erwähnte Erfahrung mit den Champions und das Wissen über deren Fähigkeiten soll das Programm anhand des verursachten Schadens und der Schlagwörter kompensieren. Der letzte Schritt ist für das Programm auch nicht möglich, da es die Teamkomposition nicht erfährt. Dies muss bei der Auswertung berücksichtigt und analysiert werden, wie groß der Nachteil dadurch ist.

5.3 Qualität der Klassifikation

Bevor die Ergebnisse in Zahlen und Grafiken gezeigt werden, ist es wichtig, ein paar Kennzahlen zu erklären. Um eine Klassifikation beurteilen zu können, werden im Fall dieser Arbeit vier Zahlen herangezogen. Scikit lässt den Benutzer nach der Klassifikation einen Report erstellen. In diesem kommen *precision*, *recall*, *f1-score* und *support* vor.

Bei der Klassifikation unterscheidet man klassifizierte Objekte in vier Kategorien: *falsch negativ, falsch positiv, richtig positiv* und *richtig negativ*. Dabei ist gemeint, dass ein Objekt als positiv oder als negativ erkannt wird, was eine Klasse anbelangt und dabei falsch oder richtig liegen kann. Betrachtet man beispielsweise eine Gruppe von Patienten und versucht diese nach Krankheit zu klassifizieren, so gilt einer der an einer Grippe leidet und bei dem diese diagnostiziert wird als richtig positiv. Wird Grippe diagnostiziert, aber es handelt sich um eine andere Krankheit, so handelt es sich um eine falsch positive Klassifikation. Wird bei einem Patient nicht Grippe diagnostiziert, aber dieser hat die Grippe, so handelt es sich um einen falsch negativ Fall. Und zu guter Letzt, hat ein Patient keine Grippe und es wird auch keine Grippe diagnostiziert, so ist dies richtig negativ.

Recall oder auch *Trefferquote* bezeichnet den Anteil der als richtig positiv klassifizierten Objekte aller richtig positiven Objekte im Datensatz. Anders ausgedrückt, zeigt der Recall die Stärke des Klassifizierers beim Finden aller positiven Beispiele. Berechnet wird diese Kennzahl r durch

$$r = \frac{\mathrm{r_p}}{\mathrm{r_p} + \mathrm{f_n}}, \tag{5.1}$$

wobei $\mathrm{r_p}$ für alle *richtig positiven* Objekte und $\mathrm{f_n}$ für alle *falsch negativen* Objekte steht.

Die **precision** oder *Relevanz* zeigt den Anteil der richtig positiv klassifizierten Objekte aller positiv klassifizierten Objekte. Unter precision wird verstanden, wie gut der Klassifizierer positive von negativen Objekten unterscheiden kann. Die Formel für die precision p lautet

$$p = \frac{\mathrm{r_p}}{\mathrm{r_p} + \mathrm{f_p}}. \tag{5.2}$$

r_p steht dabei wieder für alle *richtig positiven* Objekte und f_p für alle *falsch positiven* Objekte.

Der **f1-score** f_1 ist das harmonische Mittel von precision und recall, also

$$f_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{5.3}$$

Und unter **support** verbirgt sich die Anzahl der Objekte, die in dieser Klasse klassifiziert wurden.

5.4 Resultat

Für diese Arbeit wurden zwölf verschiedene Auswertungen vorgenommen. Zuerst wurden jede der Ligen für sich betrachtet. Wie weiter oben beschrieben wurden 250 Datensätze aus jeder Liga entnommen. Von diesen Datensätzen wurden 80% zum Lernen des Algorithmus und 20% für die Auswertung verwendet. Das bedeutet, dass 25 Spiele und die darin eingenommene Rolle pro Liga klassifiziert wurden. Auch die Profis wurden auf diese Art und Weise behandelt. Das genauen Ergebnisse können in den Tabellen im Anhang betrachtet werden.

Weiters wurden die Datensätze folgendermaßen kombiniert. Gestützt auf die Hypothese, dass die besseren Spieler (Profis und Diamant Liga) der aktuellen Meta am meisten folgen, wurden die weiteren vier Ligen auf ihre "Meta-Tauglichkeit" geprüft. Das heißt, der Klassifizierer hat mit den Datensätzen der Profis und der Diamant Liga gelernt und anschließend die Platinum, Gold, Silber und Bronze Liga Datensätze klassifiziert. Hierzu wurden 510 Spiele zum Trainieren und 240 zum Klassifizieren verwendet.

Und als letztes wurden dieses Verfahren umgedreht und wird im weiteren Verlauf als "Reverse Meta" bezeichnet. Das heißt, dass die Datensätze der Platinum, Gold, Silber und Bronze Liga kombiniert wurden. Anschließend wurden sowohl die Profis, als auch die Diamant Liga Spiele klassifiziert. Bei dieser Variante wurden 1000 Spiele verwendet, um den Algorithmus zu trainieren und 250 Spiele wurden anschließend ausgewertet. Dabei stellte sich die Frage: Lässt sich von der Art und Weise, wie schlechtere Spieler spielen, auf die Besseren rückschließen.

Diese sechs Auswertungen der einzelnen Gruppen, sowie die vier Auswertungen der Meta und die zwei Reverse Meta, ergeben insgesamt die bereits genannten zwölf Ergebnisse.

Um die Tabellen besser zu verstehen, werden diese anhand der Gold Liga Resultate erklärt. Bei jedem Ergebnis erstellt der Algorithmus einen Report mit den Kennzahlen Precision, Recall, f1-score und Support. Außerdem wurde eine Confusion Matrix (Wahrheitsmatrix) ausgegeben. In Tabelle 5.1 sind die Kennzahlen zu sehen und in Tabelle 5.2 die Confusion Matrix.

Die Matrix zeigt in den Reihen die Anzahl der Objekte, deren die Klasse zugeteilt wurden, an. Und in den Spalten lässt sich ablesen, welcher Klasse

	precision	recall	f1-score	support
ADCarry	1.00	1.00	1.00	4
Jungler	1.00	1.00	1.00	8
Mid	0.60	1.00	0.75	3
Support	1.00	0.67	0.80	6
Top	0.75	0.75	0.75	4
avg / total	0.91	0.88	0.83	25

Tabelle 5.1: Auswertung Gold Liga.

	ADCarry	Jungler	Mid	Support	Top
ADCarry	4	0	0	0	0
Jungler	0	8	0	0	0
Mid	0	0	3	0	0
Support	0	0	1	4	1
Top	0	0	1	0	6

Tabelle 5.2: Confusion Matrix Gold Liga.

das Objekt wirklich angehört. In diesem Beispiel lässt sich in Zeile 4 ablesen, dass insgesamt sechs Objekte als Supporter klassifiziert wurden. Jedoch sind nur vier dieser Objekte richtig klassifiziert, da ein Objekt in Wahrheit ein Mid-Laner und ein weiteres Objekt ein Top-Laner ist. Anders ausgedrückt, wurden bei diesem Ergebnis zwei falsch positive Objekte als Supporter eingeteilt.

Da bei diesem Testfall nur 25 Spiele verwendet wurden, sind die Ergebnisse selbstverständlich mit etwas Vorsicht zu betrachten. Sie erscheinen aber sehr gut. In dem Fall der Gold Liga werden zwei Klassen vollkommen richtig zugeordnet und bei den weiteren drei kommt es nur zu kleinen Abweichungen. c

In Abbildung 5.1 ist ein Ausschnitt des erstellten Baumes für die Spiele der Gold Liga zu sehen. Die Bäume werden relativ groß, weshalb sie in dieser Arbeit schwer abzubilden sind, deshalb wird nur ein Ausschnitt gezeigt. Die vollständigen Bäume finden sich auf der beigelegten CD dieser Arbeit. Doch bereits ein kleiner Teil des Baumes lässt schon sehr viel erkennen: In jedem der Knoten steht zu Beginn der Test, der darin durchgeführt wird. Beispielsweise steht im Wurzel-Knoten $minionsKilledRatio \leq 0.4940$. Wie weiter oben beschrieben, handelt es sich dabei um einen Test auf eines der gefilterten Attribute. Ist die Aussage für ein Objekt korrekt, also hat dieses Objekt ein minionsKilledRatio von kleiner oder gleich 0.4940, so wandert es nach links. Anderen Falls wandert es nach rechts. Weiters lässt sich der jeweilige Gini Index des Attributs ablesen und in der untersten Zeile die

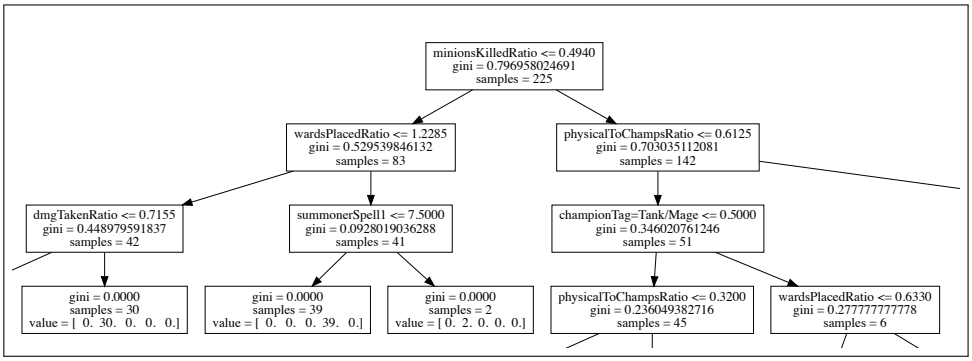


Abbildung 5.1: Entscheidungsbaum der Gold Liga.

Anzahl der Samples, die in diesem Knoten noch vorhanden sind.

Betrachten wir einen Blatt-Knoten, so sehen wir, dass darin kein Attribut mehr getestet wird, weil eine Klassifikation bereits stattgefunden hat. Jedes Objekt, das in einem Blattknoten landet wurde einer Klasse zugewiesen. Der *value* Vektor der abzulesen ist, zeigt dabei an, um welche Klasse es sich handelt. Dieser wurde zuvor im Programm festgelegt und in diesem Fall lautet er [Jungler, ADCarry, Support, Top, Mid]. Die jeweilige Stelle in dem Vektor bestimmt also, was das Objekt ist. Links in dem Bild wurden 30 Beispiele als ADCarry klassifiziert. Auch weitere Blätter sind noch zu erkennen. Die großen Vorteile der Entscheidungsbäume, die gute Lesbarkeit und Verständlichkeit, sind hier deutlich zu erkennen.

Wie in dieser Arbeit schon erwähnt wurde, ist es interessant zu sehen, welche Regeln der Algorithmus bildet. Und gleich das erste Attribut ist eine deutliche Abweichung von der Art, die präsentiert wurde. Wobei für eine Person der erste Schritt Wissen über Champions voraussetzt, das der Algorithmus in dieser Art nicht haben kann. Die Unterscheidung anhand der getöteten Minions ist aber nachvollziehbar, da sie die Rollen grob in zwei Kategorien teilt. Jungler und Supporter sollten eine eher geringe Zahl haben, während Top, Mid und AD Carry über eine höhere Anzahl an getöteten Minions verfügen sollen. Auch die zweite Auswahl, die rechts zu sehen ist, anhand des ausgeteilten physischen Schadens, wirkt sehr vernünftig.

In Abbildung 5.2 zeigt ein Diagramm die Güte der Klassifizierung im Bezug auf die einzelnen Ligen. Grundsätzlich gilt, je höher die Balken, desto besser funktioniert der Algorithmus für die gegebenen Daten. Erstaunlich ist dabei das Ergebnis der Bronze Liga. Vor der Auswertung war der Verfasser der Meinung, dass die Bronze Liga im Vergleich zu den anderen Ligen verhältnismäßig am schwierigsten zu verarbeiten sei. Auch beim Bestimmen der Rollen vor der Bearbeitung mit dem Algorithmus war die Bronze Liga am schwierigsten. Das Ergebnis zeigt, dass alle Ligen über 0.75 liegen, also über 75%. Die Platinum Liga sticht mit einem sehr guten Ergebnis heraus,

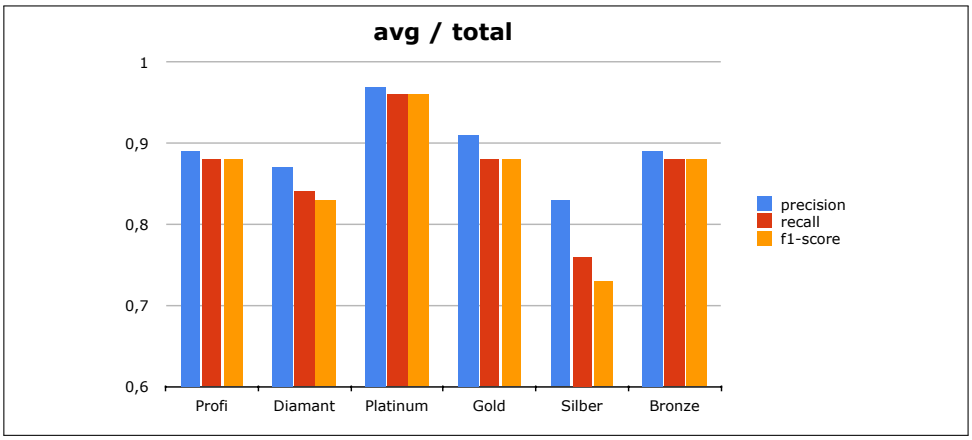


Abbildung 5.2: Durchschnitt der Kennzahlen der einzelnen Ligen.

aber auch alle anderen sind nicht zu verachten. Das spricht dafür, dass zumindest innerhalb einer Liga die Rollen ähnlich und gut zu erkennen sind.

Kommen wir nun zu konkreten Ergebnissen. Für eine gute Übersicht wurden die Ergebnisse der einzelnen Klassen gewertet. Schließlich ist diese Arbeit der Versuch, die Rollen der Spieler zu erkennen. Wie gut funktioniert also das Erkennen der einzelnen Rollen, die ein Spieler einnehmen kann? Hierfür wurden Diagramme erstellt, die den precision, den recall und den f1-score aller Auswertungen gegenüberstellen. Somit erhält man eine gute Übersicht über die Gesamtheit der Ergebnisse.

Auch die erstellen Wahrheitsmatrizen wurden zusammengefasst und in ein Diagramm gepackt. Dieses ist in Abbildung 5.3 zu sehen. In jedem der sichtbaren Balken wird angezeigt, wie viele Objekte dieser Klasse klassifiziert wurden. Außerdem zeigt die farbliche Unterteilung der Balken die wahre Zugehörigkeit der Objekte an. Betrachtet man den Balken des ADCarrys, so sieht man, dass es eine sehr geringe Verwechslung dieser Klasse gibt. Zwischen Jungler und Supporter lässt sich eine größere Verwechslung feststellen. Dies ist nachvollziehbar, da beide Rollen eine unterstützende Funktion inne haben. Beide Rollen nehmen ähnliche Strategien wahr. Für eine definitive Unterscheidung müsste hier die Position der Spieler betrachtet werden. Auch der Anteil der Mid-Laner, die für einen Top-Laner gehalten werden, lässt sich gut erklären. Top-Laner können und werden sehr unterschiedlich interpretiert und es ist durchaus der Fall, dass der Top-Laner sich lediglich durch eine Beschwörerfähigkeit unterscheidet. Auch der Trend, Champions, die ursprünglich als Mid-Laner brillierten, in der Rolle des Top-Laners zu spielen, trägt einiges hierzu bei.

Zwar lässt sich in Abbildung 5.4 nicht ablesen, welche Verwechslungen vorliegen, aber ein Überblick über die Anzahl der Verwechslungen ist zu

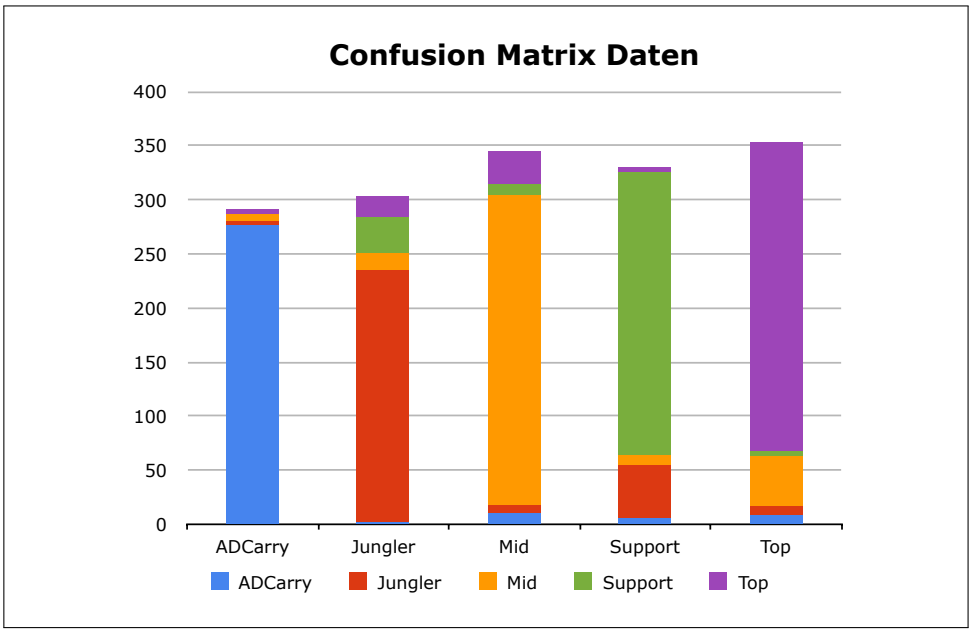


Abbildung 5.3: Confusion Matrix aller Auswertungen.

erkennen. Es ist darin zu sehen, dass die Rate der richtig klassifizierten Objekte weit über 70% liegt, bzw. beinahe überall bei 80%. Das ist in Bezug auf das Matchmaking ein sehr wesentlicher Bestandteil. Die Kategorisierung dient alleine dem Zweck, eine Ahnung zu erhalten, welche Rollen der Spieler präferiert und gerne spielen möchte. Selbstverständlich könnte die Rate aber immer noch höher sein, um mehr Sicherheit zu geben.

5.4.1 ADCarry

Das Diagramm in Abbildung 5.5 zeigt die Auswertung der ADCarry-Rolle. Diese Rolle lässt sich für den Experten gut und einfach erkennen. Auch gibt es wenig Verwechslungsmöglichkeiten. Am ehesten dürfte es zu Verwechslungen zwischen ADCarry und dem Mid-Laner kommen. Beide stecken wenig Schaden ein und teilen viel Schaden aus. Und beide sollten eine hohe Rate an getöteten Minions aufweisen. Jedoch ist der ADCarry anhand der Champions leicht zu kategorisieren. Das Diagramm zeigt, dass die auch der Algorithmus keine großen Schwierigkeiten mit dieser Rolle hat. Lediglich in der Meta Auswertung der Bronze Liga fällt der ADCarry etwas zurück, wobei eine Precision Rate von über 73% immer noch sehr gut erscheint. Zurückgehen dürfte dieses leichte Abfallen auf die Spielstärke der Spieler in der Bronze Liga, da die getöteten Minions eine große Rolle in der Klassifizierung spielen (jeder erstellte Baum wählt dieses Attribut zuerst) und diese Rate mit der Spielstärke zunimmt. Das effiziente Töten der Minions ist eine

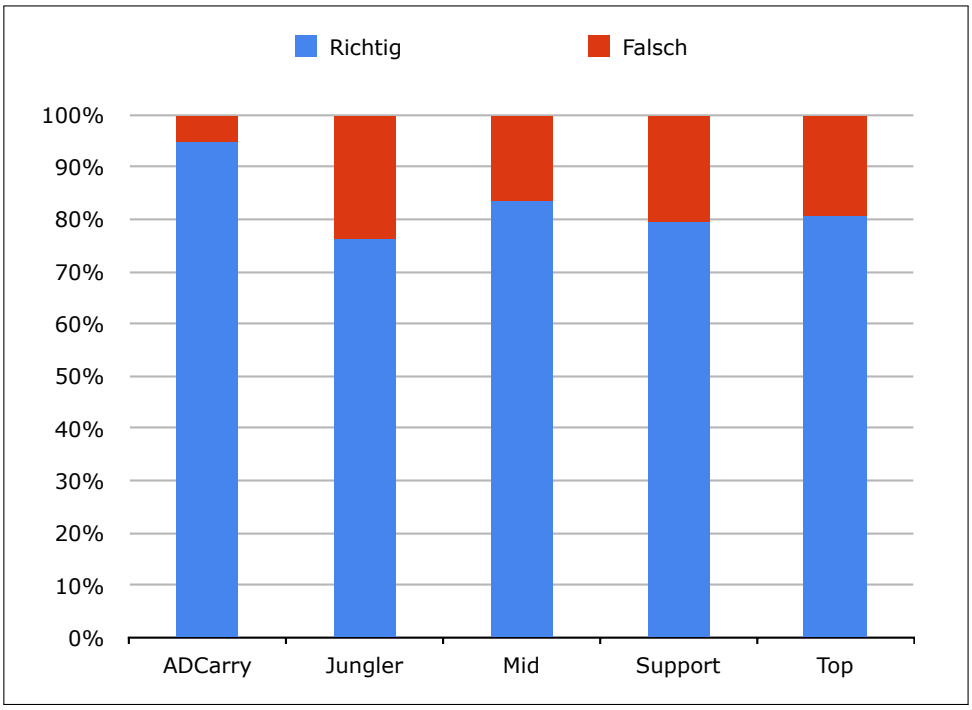


Abbildung 5.4: Falsch und richtig klassifizierte Objekte aller Auswertungen.

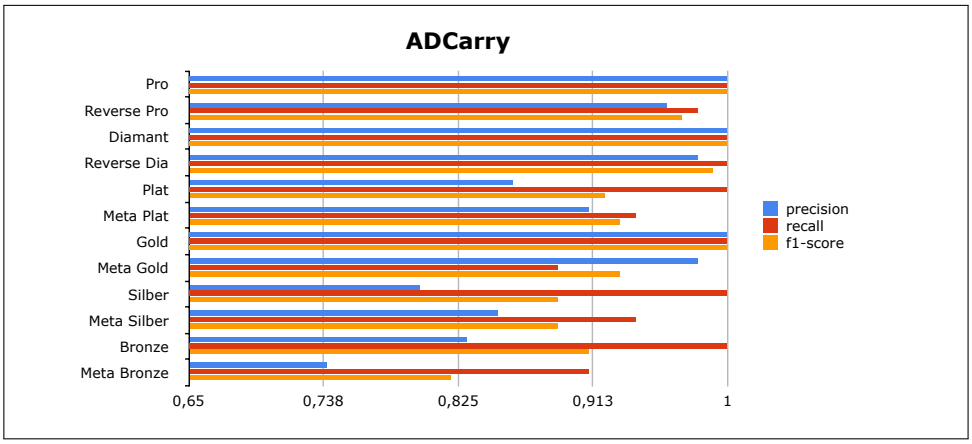


Abbildung 5.5: Auswertung aller ADCarry Klassifikationen.

der größeren Schwierigkeiten für Neulinge und spielmechanisch schlechteren Spieler.

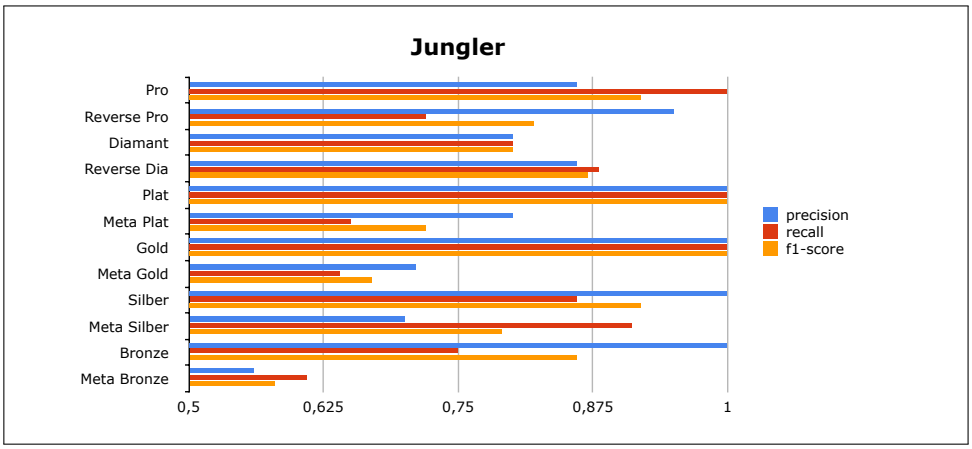


Abbildung 5.6: Auswertung aller Jungler Klassifikationen.

5.4.2 Jungler

Abbildung 5.6 zeigt die Auswertung der Spiele, die mit der Jungler Rolle assoziiert wurden. Die Rolle des Junglers ist sehr vielfältig. Bezogen auf die gesamte Komposition des Teams, kann der Jungler sehr leicht unterschiedliche Strategien verfolgen. Durch die Auswahl des Champions und der Gegenstände kann die Rolle des Jungler sehr variieren. Jedoch gibt es Attribute, die den Jungler eine Alleinstellung geben sollten. Die Anzahl der getöteten neutralen Minions ist hier definitiv das Wichtigste. Bedingt dadurch, dass der Jungler einen Großteil seiner Spielzeit damit verbringen sollte, gegen neutrale Minions zu kämpfen, sollte der Algorithmus hier anknüpfen können.

Wie das Diagramm zeigt, sind die Zahlen unterschiedlicher als beim AD-Carry. Die Skala beginnt nun bei ungefähr 55%, was bedeutet, dass die Klassifizierung schlechter arbeitet. Bei einem Wert von knapp über der Hälfte bedeutet das, dass in der Bronze Liga jeder zweite Jungler verwechselt wird. Für eine Person lässt sich der Jungler auch noch anhand der Beschwörerfähigkeit “Zerschmettern” erkennen. Allerdings arbeitet der Algorithmus hier anders, weil das Attribut als Zahl genommen wurde. Als Verbesserung könnte man hier die Fähigkeiten als Text und nicht deren ID speichern und filtern, um dem Algorithmus zu helfen. Diese Erkenntnis stellte sich leider erst beim Auswerten der Ergebnisse ein. Auch die Meta Auswertungen zeigen, dass sich die Strategie des Jungler in den höheren Ligen von den niedrigeren unterscheidet. Alle Meta Ergebnisse sind durchwegs niedriger als die der Ligen. Hierfür könnte eine größere Effizienz der guten Spieler eine Erklärung geben: mehr platzierte Wards, mehr getötete Minions (neutral, feindlich) und auch die Auswahl der Champions. Für eine detaillierte Analyse müsste man allerdings alle Datensätze betrachten.

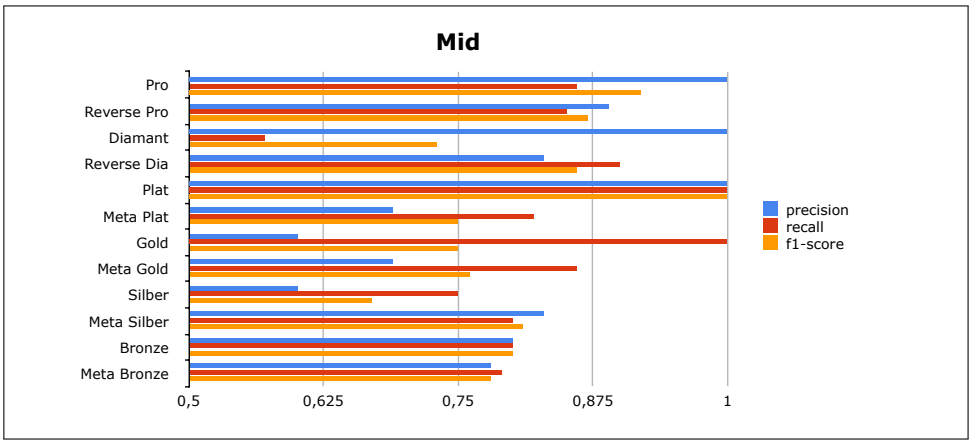


Abbildung 5.7: Auswertung aller Mid-Laner Klassifikationen.

5.4.3 Mid-Laner

Die Rolle des Mid-Laners ist eine sehr schwierige und sehr spielprägende innerhalb von LoL. Der Mid-Laner hat eine enorm große Präsenz auf der Spielkarte durch seine zentrale Positionierung. Außerdem sollte er immer einer der größten Schadensverursacher sein. Viele Spiele werden durch einen besseren Mid-Laner entschieden. Was die Klassifizierung betrifft, so ist es mit abnehmender Spielstärke immer schwieriger den Mid-Laner zu identifizieren. Grundsätzlich ist der Championpool nahezu uneingeschränkt und ändert sich je nach Spieler.

Wie sich in Abbildung 5.7 ablesen lässt, nimmt die Genauigkeit der Klassifikation mit der Spielstärke ab. Wie bereits erwähnt, unterscheiden sich Mid-Laner sehr stark voneinander. Jedoch existieren in der gültigen Meta jeweils beste Varianten. Diese lassen sich in den höheren Ligen leichter feststellen. Grundsätzlich ist das Ergebnis nicht schlecht, da fast jede Kennzahl bei mindestens 75% steht, mit leichten Ausreißern in der Gold und Silber Liga. Hier kann es aufgrund der Größe der Datensätze durchaus an einer schlechten Stichprobe liegen, dass die Varianz etwas höher ausfällt. Was die Meta Ergebnisse betrifft, so sind diese durchwegs beachtenswert und es scheint, dass nicht nur die besten der Besten wissen, was und wie gespielt werden sollte.

5.4.4 Top-Laner

Die Rolle des Top-Laners gestaltet sich sehr unterschiedlich und variiert stark von Zeit zu Zeit. Während in der Meta zu einem Zeitpunkt Tanks bevorzugt werden – also ein Champion, der viel aushält und eine Strategie, die darauf abzielt, dass der Spieler viel Schaden einsteckt und die schwächeren Mitglieder schützt – so wechselt dies schnell hin zu Kämpfern oder

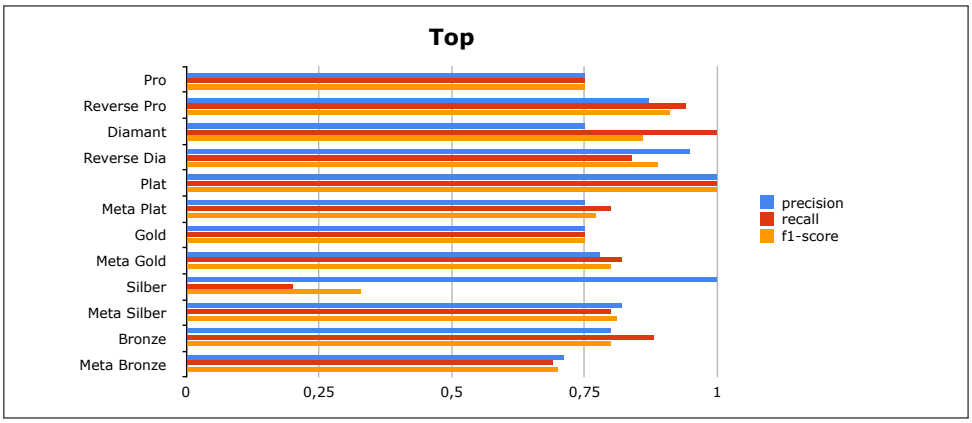


Abbildung 5.8: Auswertung aller Top-Laner Klassifikationen.

Magiern. Das bedeutet, dass die Strategie und Komposition der Gruppe dem Top-Laner seine Spielweise vorgibt. Betrachtet man die Attribute, so sind eine hohe Anzahl an getöteten Minions, die Beschwörerfähigkeit Teleport und (zur Zeit) viel erlittener Schaden ein Indiz für einen Top-Laner. Schwierigkeiten dürfte es bei der Unterscheidung zwischen Top- und Mid-Laner geben, wie bereits erwähnt wurde.

Betrachtet man das Diagramm in Abbildung 5.8, so lassen sich sehr unterschiedliche Ergebnisse betrachten. Gesamt betrachtet kann man mit der Klassifizierung weniger zufrieden sein, da im Schnitt nur etwa drei von vier Spielern erkannt werden. Für eine gute Vorhersage dürfte das zu wenig sein. Auch hier muss dem Algorithmus zu Gute gehalten werden, dass der Umgang mit den Beschwörerfähigkeiten bei der Auswahl der Attribute falsch interpretiert wurde. Interessant ist das Ergebnis der Silber Liga. Während die Bearbeitung der Liga Daten ein sehr schlechtes Ergebnis zeigt, mit einem sehr niedrigen Recall und f1-score, ist das Meta Ergebnis erstaunlich gut. Scheinbar fehlten hier dem Algorithmus ein paar Datensätze, um eine gute Klassifizierung zu gewährleisten. Allgemein muss man natürlich sagen, je mehr Datensätze, desto besser.

Die Platinum Liga sticht erneut hervor, der Datensatz dürfte hier sehr homogen und gut zu verarbeiten sein. Durchwegs sind die Meta Ergebnisse sehr gut, woraus sich schließen lässt, dass die unterschiedliche Art, die Rolle des Top-Laners zu interpretieren, schwer zu klassifizieren ist, bzw. erst mit zunehmender Anzahl und Daten zu erlernen ist.

5.4.5 Supporter

Der Supporter ist ein Rolle, die sehr spielentscheidend ist, sich jedoch über geringere Beliebtheit erfreut. Aufgrund der unterstützenden Funktion, die eingenommen wird, kann der Supporter in manchen Spielen sehr hilflos wer-

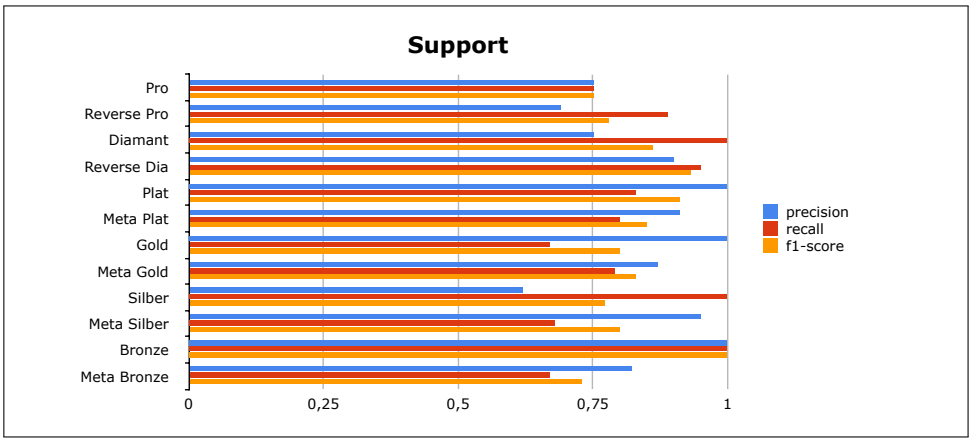


Abbildung 5.9: Auswertung aller Supporter Klassifikationen.

den. Auch der geringe Schadensoutput ist ein Faktor, weshalb viele Spieler lieber andere Rollen einnehmen. Die Stärke eines Supporters liegt im strategischen Vorgehen, sowie für Übersicht und Unterstützung der anderen Spieler zu sorgen.

Die Rolle lässt sich gut kategorisieren, kann jedoch zu Verwechslungen mit dem Jungler führen. Eine hohe Anzahl an platzierten Wards, eine geringe Anzahl an getöteten Minions, sowie eine niedrige Anzahl and verursachtem Schaden helfen, den Supporter zu identifizieren. In manchen Spielen kommt es dazu, dass ein Spieler gegen seinen Willen die Rolle des Supporters einnehmen muss. Dies tritt häufiger auf, als bei anderen Rollen. Das führt dazu, dass einige Spieler einen ursprünglich für den Mid-Laner konzipierten Champion auswählen und sich in Richtung Schaden-Verursachen entwickeln. Das führt zu Verwechslungen zwischen Mid-Laner und Supporter. Einzig die Anzahl der gesetzten Wards dient dann zur Alleinstellung, da die meisten Spieler doch so weit sind, zumindest die wichtigste Aufgabe des Supporters zu übernehmen, wenn sie schon die restlichen vernachlässigen. In der Abbildung 5.9 sieht man eine mittelmäßige Auswertung. Im Vergleich zum Top-Laner erscheinen die Werte gut. Jedoch in den unteren zwei Ligen wandern die Werte unter 0.75. Hier spielt nun die abnehmende Lust, einen Supporter zu spielen, sicherlich eine große Rolle. Betrachtet man vor allem die Meta Ergebnisse, so wird einem klar, dass der Supporter in den höheren Ligen anders gespielt wird. Hier ist man sich bewusst, welche besondere und wichtige Rolle der Supporter einnimmt und welche Aufgabe erfüllt werden müssen.

Erstaunlicherweise sind auch die Reverse Meta Ergebnisse sehr gut, woraus sich schließen lässt, dass der Algorithmus auch mit den Spielen der niedrigeren Klassen zu einigen guten Regeln kommt, mit denen sich gut arbeiten lässt.

5.4.6 Zusammenfassung

Abschließend lässt sich sagen, dass die Klassifizierung und Kategorisierung im Durchschnitt in mehr als 75% aller Fälle funktioniert. Dies ist kein schlechter Wert. Für eine Verbesserung des Wertes müsste schon eine Erweiterung der Daten reichen. Mehr Datensätze für ein besseres Training dürften dem Algorithmus auf jeden Fall helfen. Auch werden dadurch kleine Ausreißer, wo ein Spieler keiner der Regeln gefolgt ist, besser minimiert. Insgesamt ist der Verfasser der Meinung, dass sich auf diesen Ergebnissen aufbauen lässt. Bedenkt man die Anzahl an Personen, die Riot für solch ein Unterfangen zur Verfügung hätte und auch die Daten und Rechenleistung, so könnte man meinen, dass eine Verbesserung des Matchmakings kein unmögliches Unterfangen sein kann. Selbstverständlich wird Riot wissen, welche Projekte Priorität haben, aber eine Umsetzung erscheint machbar.

Eine Rate von ungefähr 90% dürfte wohl reichen, um ein zufriedenstellendes Ergebnis zu erhalten. Bei der Auswertung wurde festgestellt, dass viele Spieler zu sehr unterschiedlichen Rollen neigen und erst in höheren Ligen klare Präferenzen abzulesen sind. Eine Nachforschung in diese Richtung könnte sehr spannend sein: Wie stark präferieren Spieler eine gewisse Rolle, bzw. wie hoch ist die Bereitschaft, andere Rollen einzunehmen? Hierzu gibt es keine Veröffentlichungen seitens Riot.

Zum Schluss bleibt noch zu sagen, dass durch eine genaue Betrachtung der Daten sehr viel Wissen gewonnen werden kann. Interessant wäre zu erfahren, welche Forschungen Riot mit diesen Daten betreibt und welchen Profit sie daraus ziehen. Sei es wirtschaftlicher Art, oder zum Verbessern des Spiels.

Kapitel 6

Fazit

Diese Arbeit hat sich mit dem Online Spiel League of Legends und den darin enthaltenen Spieldaten auseinander gesetzt: Welche Informationen können genutzt werden, um einen Vorteil zu gewinnen? Als Beispiel, welches wertvolle Wissen erlernt werden kann, wurde das bestehende Matchmaking hergenommen und ein Versuch gestartet, dieses um den Aspekt der spielbaren Rollen zu erweitern. Der Gedanke dabei war, unterschiedliche Spielertypen zu homogenen Gruppen zu vereinen.

Insgesamt betrachtet zeigen die gesamte Arbeit, sowie der Versuch, Spieldaten auf diese Art zu verwenden, sehr gute Ergebnisse. Die Klassifizierung funktioniert über weite Teile sehr gut und die Verwechslungsrate ist überschaubar.

Während der Arbeit entstanden laufend neue Ideen, die mit den zur Verfügung gestellten Daten weiterentwickelt werden könnten. Viele Nachforschungen und interessante Themen ließen sich damit eruieren. Erwähnt wurde beispielsweise schon eine Analyse der Strategien nach Regionen. Unterscheiden sich die Rollen je nach geografischer Lage und Umfeld? Auch das Betrachten der unterschiedlichen Teamkompositionen und deren Vor- und Nachteile, wäre sehr interessant. Ebenfalls könnte man versuchen, die gesamten Daten zu verwenden, um Gründe für die Spielstärke eines Spielers zu finden. Also warum ist ein guter Spieler gut und ein schlechter Spieler schlecht, abgesehen von persönlichen Merkmalen, wie etwa Reaktionszeit und dergleichen. Dies würde rein auf die Strategie und Ausführung einer Rolle und eines Spieltypen bezogen werden.

6.1 Ausblick / Verbesserungen

Was könnte erweitert werden, bzw. was könnte man an dem jetzigen Versuch verbessern? Die erste Verbesserung liegt klar auf der Hand: Viel mehr Datensätze. 100 000 oder noch mehr Spiele würden eine wesentlich repräsentativere Menge darstellen und möglicherweise die Klassifizierung erleichtern

und viele Ausreißer abschwächen.

Die Verwendung von Machine Learning Methoden zur Attributsauswahl wäre ein weiterer Schritt, welcher helfen könnte. Werden die Attribute nicht per Hand gefiltert, sondern ein Algorithmus tut dies, so werden eventuell gänzlich andere Attribute als wichtig eingestuft und möglicherweise ändert dies den gesamten restlichen Vorgang.

Auch wurde bereits erwähnt, dass der Umgang mit dem Attribut der Beschwörerfähigkeiten nicht ideal war. Bei der Filterung war der Gedanke, dass der Algorithmus vergleichen kann, ob der Datensatz diese Fähigkeit besitzt, im Sinne eines programmiertechnischen "==". Allerdings wurden von dem Algorithmus "kleiner gleich"-Vergleiche ("<=") angestellt, da die Beschwörerfähigkeiten als Zahl gespeichert sind. Dadurch wird keine eindeutige Prüfung auf genau eine Fähigkeit vollzogen. Wie beschrieben wurde unterscheiden sich manche Rollen großteils anhand der Beschwörerfähigkeiten, weshalb es Schade ist, das dieses Attribut so etwas verloren ging.

Eine Normalisierung aller numerischen Attribute würde eine gute Vergleichbarkeit bringen. Beispielsweise wüsste man bei der Anzahl der getöteten Minions schneller, ob diese nun hoch oder niedrig ist, wenn sie sich nur zwischen 0-1 bewegen würde.

Die Arbeit wurde im Bezug auf das Matchmaking geschrieben. Hier wäre es schön, einen Prototypen zu erstellen, der die klassifizierten Daten tatsächlich verwendet und damit geeignete Gegner sucht.

Was die Umsetzung des Projekts betrifft, so könnte anstatt der benützten Library eine eigene Implementierung der Riot-Api helfen, die Geschwindigkeit zu erhöhen. Dazu müsste man JSON-Parser testen und den geeignetsten implementieren.

Alles in allem handelt es sich dabei durchwegs um sehr zeitaufwendige und große Aufgaben, welche aber einen Mehrwert bringen würden. Selbstverständlich ist es nicht Aufgabe dieser Arbeit alle Möglichkeiten auszuloten, weshalb das Ergebnis in dieser Art und Weise passend und gerechtfertigt ist.

Anhang A

Auswertung

	precision	recall	f1-score	support
ADCarry	0.83	1.00	0.91	5
Jungler	1.00	0.75	0.86	4
Mid	0.80	0.80	0.80	5
Support	1.00	1.00	1.00	6
Top	0.80	0.88	0.80	5
avg / total	0.89	0.88	0.88	25

Tabelle A.1: Auswertung Bronze Liga.

	precision	recall	f1-score	support
ADCarry	1.00	1.00	1.00	4
Jungler	0.80	0.80	0.80	5
Mid	1.00	0.57	0.73	7
Support	0.75	1.00	0.86	3
Top	0.75	1.00	0.86	6
avg / total	0.87	0.84	0.83	25

Tabelle A.2: Auswertung Diamant Liga.

	precision	recall	f1-score	support
ADCarry	1.00	1.00	1.00	4
Jungler	1.00	1.00	1.00	8
Mid	0.60	1.00	0.75	3
Support	1.00	0.67	0.80	6
Top	0.75	0.75	0.75	4
avg / total	0.91	0.88	0.83	25

Tabelle A.3: Auswertung Gold Liga.

	precision	recall	f1-score	support
ADCarry	0.86	1.00	0.92	6
Jungler	1.00	1.00	1.00	5
Mid	1.00	1.00	1.00	4
Support	1.00	0.83	0.91	6
Top	1.00	1.00	1.00	4
avg / total	0.97	0.96	0.88	25

Tabelle A.4: Auswertung Platinum Liga.

	precision	recall	f1-score	support
ADCarry	1.00	1.00	1.00	4
Jungler	0.86	1.00	0.92	6
Mid	1.00	0.86	0.92	7
Support	0.75	0.75	0.75	4
Top	0.75	0.75	0.75	4
avg / total	0.89	0.88	0.96	25

Tabelle A.5: Auswertung Profi Spieler.

	precision	recall	f1-score	support
ADCarry	0.80	1.00	0.89	4
Jungler	1.00	0.86	0.92	7
Mid	0.60	0.75	0.67	4
Support	0.62	1.00	0.77	5
Top	1.00	0.20	0.33	5
avg / total	0.83	0.76	0.88	25

Tabelle A.6: Auswertung Silber Liga.

	precision	recall	f1-score	support
ADCarry	0.98	1.00	0.99	42
Jungler	0.86	0.88	0.87	34
Mid	0.83	0.90	0.86	60
Support	0.90	0.95	0.93	40
Top	0.95	0.84	0.89	74
avg / total	0.91	0.90	0.90	25

Tabelle A.7: Auswertung Reverse Meta Diamant.

	precision	recall	f1-score	support
ADCarry	0.96	0.98	0.97	45
Jungler	0.95	0.72	0.82	58
Mid	0.89	0.85	0.87	65
Support	0.69	0.89	0.78	46
Top	0.87	0.94	0.91	36
avg / total	0.88	0.86	0.86	250

Tabelle A.8: Auswertung Reverse Meta Pro.

	precision	recall	f1-score	support
ADCarry	0.74	0.91	0.82	35
Jungler	0.56	0.61	0.58	33
Mid	0.78	0.79	0.78	57
Support	0.82	0.67	0.73	54
Top	0.71	0.69	0.70	61
avg / total	0.73	0.73	0.73	250

Tabelle A.9: Auswertung Meta Bronze.

	precision	recall	f1-score	support
ADCarry	0.98	0.89	0.93	55
Jungler	0.71	0.64	0.67	45
Mid	0.69	0.86	0.76	49
Support	0.87	0.79	0.83	52
Top	0.78	0.82	0.80	39
avg / total	0.81	0.80	0.81	240

Tabelle A.10: Auswertung Meta Gold.

	precision	recall	f1-score	support
ADCarry	0.91	0.94	0.93	53
Jungler	0.80	0.65	0.72	43
Mid	0.69	0.82	0.75	40
Support	0.91	0.80	0.85	49
Top	0.75	0.80	0.77	55
avg / total	0.81	0.81	0.81	240

Tabelle A.11: Auswertung Meta Platinum.

	precision	recall	f1-score	support
ADCarry	0.85	0.94	0.89	35
Jungler	0.70	0.91	0.79	55
Mid	0.83	0.80	0.81	44
Support	0.95	0.68	0.80	60
Top	0.82	0.80	0.81	46
avg / total	0.83	0.82	0.81	240

Tabelle A.12: Auswertung Meta Silber.

Anhang B

Confusion Matrix

	ADCarry	Jungler	Mid	Support	Top
ADCarry	4	0	0	0	0
Jungler	0	4	0	1	0
Mid	0	1	4	0	2
Support	0	0	0	3	0
Top	0	0	1	0	6

Tabelle B.2: Confusion Matrix Diamant Liga.

	ADCarry	Jungler	Mid	Support	Top
ADCarry	4	0	0	0	0
Jungler	0	8	0	0	0
Mid	0	0	3	0	0
Support	0	0	1	4	1
Top	0	0	1	0	6

Tabelle B.3: Confusion Matrix Gold Liga.

	ADCarry	Jungler	Mid	Support	Top
ADCarry	5	0	0	0	0
Jungler	1	3	0	0	0
Mid	0	0	4	0	1
Support	0	0	0	6	0
Top	0	0	1	0	4

Tabelle B.1: Confusion Matrix Bronze Liga.

	ADCarry	Jungler	Mid	Support	Top
ADCarry	6	0	0	0	0
Jungler	0	5	0	0	0
Mid	0	0	4	0	0
Support	1	0	0	5	0
Top	0	0	0	0	3

Tabelle B.4: Confusion Matrix Platinum Liga.

	ADCarry	Jungler	Mid	Support	Top
ADCarry	4	0	0	0	0
Jungler	0	6	0	0	0
Mid	0	0	6	0	1
Support	0	1	0	3	0
Top	0	0	0	1	4

Tabelle B.5: Confusion Matrix Profi-Spieler.

	ADCarry	Jungler	Mid	Support	Top
ADCarry	4	0	0	0	0
Jungler	0	6	0	1	0
Mid	0	0	3	1	0
Support	0	0	0	5	0
Top	1	0	2	1	3

Tabelle B.6: Confusion Matrix Silber Liga.

	ADCarry	Jungler	Mid	Support	Top
ADCarry	42	0	0	0	0
Jungler	0	30	0	4	0
Mid	1	2	54	0	3
Support	0	0	2	38	0
Top	0	3	9	0	37

Tabelle B.7: Confusion Matrix Reverse Meta Diamant.

	ADCarry	Jungler	Mid	Support	Top
ADCarry	44	0	1	0	0
Jungler	0	42	3	13	0
Mid	2	0	55	5	3
Support	0	1	2	41	2
Top	0	1	1	0	62

Tabelle B.8: Confusion Matrix Reverse Meta Pro.

	ADCarry	Jungler	Mid	Support	Top
ADCarry	32	1	0	0	2
Jungler	0	20	04	3	6
Mid	2	1	45	2	7
Support	2	13	1	36	2
Top	7	1	8	3	42

Tabelle B.9: Confusion Matrix Meta Bronze.

	ADCarry	Jungler	Mid	Support	Top
ADCarry	49	2	3	0	1
Jungler	0	29	7	5	4
Mid	1	1	42	1	4
Support	0	9	2	41	0
Top	0	0	7	0	42

Tabelle B.10: Confusion Matrix Meta Gold.

	ADCarry	Jungler	Mid	Support	Top
ADCarry	50	0	2	0	1
Jungler	0	28	2	4	9
Mid	2	0	33	0	5
Support	3	6	1	39	0
Top	0	1	10	0	32

Tabelle B.11: Confusion Matrix Meta Plat.

	ADCarry	Jungler	Mid	Support	Top
ADCarry	33	0	0	0	2
Jungler	2	50	0	2	1
Mid	3	1	35	0	5
Support	0	18	1	41	0
Top	1	2	6	0	44

Tabelle B.12: Confusion Matrix Meta Silber.

Anhang C

Inhalt der CD-ROM

C.1 PDF-Dateien

Pfad: /

Florian_Eckerstorfer_2015.pdf Masterarbeit (Gesamtdokument)

C.2 Abbildungen

Pfad: /Abbildungen

*.pdf	Vektorgrafiken
*.jpg	Rasterbilder und Screenshots
*.png	Rasterbilder

C.3 Auswertung

Pfad: /Auswertung

/Bäume	Enthält alle erstellten Entscheidungsbäume als PDF-Dateien.
/Diagramme	Enthält alle erstellen Diagramme.
/Ergebnis	Enthält alle erstellen .txt-Dateien, in denen sich die Ergebnisse der Auswertungen befinden.
/Spieldaten	Enthält alle Datensätze, die im Rahmen dieser Arbeit, erstellt wurden. Weiters befinden sich hier Unterordner mit den einzelnen Dateien zu allen Spielern, die abgefragt wurden, eingeteilt in deren Liga.

C.4 Literatur

Pfad: /Literatur

*.pdf	Kopien der verwendeten Literatur. Die PDFs sind jeweils nach dem Namen der Arbeit benannt.

C.5 Literatur-Online

Pfad: /Literatur-Online

*.html	Kopien der verwendeten Online-Quellen.

C.6 Projekt

Pfad: /Projekt

readme_lollytics.rtf . .	Dokument mit kurzer Instruktion zum Aufsetzen des Projekts.
readme_python.rtf . . .	Dokument mit kurzer Instruktion zum Aufsetzen des Projekts.
/lollytics	Enthält den Source Code, sowie alle benötigten Dateien, für das Backend und die Client-Applikation.
/python	Enthält den Source Code für die Auswertung der Spieldaten.
/python/data	Enthält alle csv-Dateien, die im Rahmen des Projekts ausgewertet wurden.

Quellenverzeichnis

Literatur

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006 (siehe S. 22, 26, 27).

[2] Jeremy Blackburn und Haewoon Kwak. „STFU NOOB!: Predicting Crowdsourced Decisions on Toxic Behavior in Online Games“. In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW '14. Seoul, Korea: ACM, 2014, S. 877–888 (siehe S. 3).

[3] Leo Breiman u. a. *Classification and Regression Trees*. Statistics/Probability Series. Belmont, California, U.S.A.: Wadsworth Publishing Company, 1984 (siehe S. 24).

[4] Neven Caplar, Mirko Suznjevic und Maja Matijasevic. „Analysis of player's in-game performance vs rating: Case study of Heroes of Newerth“. In: *CoRR* abs/1305.5189 (2013). URL: http://dblp.uni-trier.de/db/journals/corr/corr1305.html#abs-1305-5189 (siehe S. 4).

[5] Colin Feo und Allen Sirolly. *Predicting Winning Teams in League of Legends*. Techn. Ber. University of Pennsylvania, 2014 (siehe S. 5).

[6] Lynn Gao u. a. *Classifying Dota 2 Hero Characters Based on Play Style and Performance*. Techn. Ber. University of Utah Laboratory for Experimental Economics und Finance, 2013 (siehe S. 5).

[7] Jorge Jiménez-Rodríguez, Guillermo Jiménez-Díaz und Belén Díaz-Agudo. „Matchmaking and Case-based Recommendations“. In: *Proceedings of the ICCBR 2011 Workshops*. Greenwich, London, United Kingdom: Cia. Española de Reprografía y Servicios, 2011, S. 53–62 (siehe S. 5).

[8] Angela Kim, Afsheen Ghorashy und Paul van der Vecht. *Machine Learning Based Prediction of League of Legends Matches*. Techn. Ber. University of Waterloo, 2014 (siehe S. 7).

[9] Yubo Kou und Bonnie Nardi. „Governance in League of Legends: A Hybrid System“. In: *Proceedings of the 9th International Conference on the Foundations of Digital Games*. Fort Lauderdale, FL, 2014 (siehe S. 4).

[10] Tom M. Mitchel. *Machine Learning*. New York, NY: McGraw-Hill, 1997 (siehe S. 22, 24).

[11] Mateusz Myślak und Dominik Deja. „Developing Game-Structure Sensitive Matchmaking System for Massive-Multiplayer Online Games“. English. In: *Social Informatics*. Hrsg. von Luca Maria Aiello und Daniel McFarland. Bd. 8852. Lecture Notes in Computer Science. Springer International Publishing, 2015, S. 200–208. URL: http://dx.doi.org/10.1007/978-3-319-15168-7_25 (siehe S. 4).

[12] Kweku-Muata Osei-Bryson. „Evaluation of decision trees: a multi-criteria approach“. In: *Computers & Operations Research* 31.11 (2004), S. 1933–1945 (siehe S. 20).

[13] J. R. Quinlan. „Induction of decision trees“. In: *Machine Learning* 1.1 (1986), S. 81–106 (siehe S. 23).

[14] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993 (siehe S. 23).

[15] Xindong Wu u. a. „Top 10 Algorithms in Data Mining“. In: *Knowledge and Information Systems* 14.1 (Dez. 2007), S. 1–37. URL: http://dx.doi.org/10.1007/s10115-007-0114-2 (siehe S. 24).

Online-Quellen

[16] Paul Tassi / Forbes. 2014. URL: http://www.forbes.com/sites/insertcoin/2014/01/27/riots-league-of-legends-reveals-astonishing-27-million-daily-players-67-million-monthly/ (siehe S. 9).

[17] Riot Games. 2014. URL: http://www.cnet.com/news/video-game-industry-grew-4-times-faster-than-us-economy-in-2012-study-says/ (siehe S. 3).

[18] Riot Games. 2015. URL: http://www.riotgames.com/articles/20141201/1628/worlds-2014-numbers (siehe S. 14).

[19] Riot Games. 2015. URL: http://na.lolesports.com/articles/what-are-roles (siehe S. 14).

[20] Riot Games. *The Riot Api*. 2015. URL: http://na.leagueoflegends.com/en/news/riot-games/announcements/dev-blog-riot-api (siehe S. 28).

[21] Jeffrey Lyn. 2015. URL: http://na.leagueoflegends.com/en/news/game-updates/player-behavior/upgrading-tribunal (siehe S. 3, 4).

[22] Christian Nutt. 2015. URL: http://www.gamasutra.com/view/news/239318/More_carrot_less_stick_Jeffrey_Lin_on_tweaking_League_of_Legends_player_behavior.php (siehe S. 3).

[23] Rithms. *riot api github*. 2015. URL: https://github.com/rithms/riot-api-java (siehe S. 31).

[24] Daniel Ross. *Arpad Elo and the Elo Rating System*. 2007. URL: http://en.chessbase.com/post/arpad-elo-and-the-elo-rating-system (siehe S. 18).

[25] Jeff Schnitzer. 2015. URL: https://github.com/objectify/objectify (siehe S. 31).

[26] scikit-learn. 2015. URL: http://scikit-learn.org/stable/modules/tree.html (siehe S. 34).

[27] Nick Statt. 2015. URL: http://gameinfo.euw.leagueoflegends.com/de/game-info/champions/ (siehe S. 14).

[28] WolframAlpha. 2015. URL: http://mathworld.wolfram.com/ExternalPathLength.html (siehe S. 21).