Vergelijking van rating-systemen voor voetbalteams

door

C. Kerkmeester

ter verkrijging van de graad van Bachelor of Science
aan de Technische Universiteit Delft,
in het openbaar te verdedigen op woensdag 10 juli 2024 om 14:00 uur.

Studentnumner:	5291666	
Begin- en einddatum project:	22 April 2024 – 10 Juli 2024	
Beoordelingscommissie:	Dr. F. J. van de Bult,	TU Delft, begeleider
	Dr. N. de Kleijn,	TU Delft

Een elektronische versie van dit verslag is beschikbaar op `http://repository.tudelft.nl/`.

TU Delft

Lekensamenvatting

In schaken wordt een systeem gebruikt om de sterktes van spelers bij te houden. Deze sterktes worden aangegeven met een getal, de rating, en hoe hoger deze rating is, hoe beter we verwachten dat de speler is. Op basis van de ratings kunnen de spelers dan gerangschikt worden. Dit kan toegepast worden op allerlei dingen, maar voornamelijk sport. In dit verslag worden drie verschillende systemen getoond, die gebruikt kunnen worden om ratings te geven aan voetbalteams (specifiek Eredivisie en Eerste Divisie), om ze te kunnen rangschikken. Iedere maand zullen de ratings van de teams veranderd worden op basis van hun resultaten in die maand. Hoe de ratings veranderen hangt af van welk van de drie systemen gebruikt wordt. Deze systemen hangen allemaal af van bepaalde constanten: de optimale waarden voor deze constanten kunnen met behulp van bepaalde methoden gevonden worden. De drie systemen zijn het Elo-systeem, het Glicko-systeem en het MOV-systeem. Het Elo-systeem is het meest simpele van de drie, en een waar veel mensen al van weten. Het Glicko-systeem is een ingewikkeldere versie van het Elo-systeem, waarbij er meer rekening wordt gehouden met hoe er onzekerheid is over de sterkte van een team. Het MOV-systeem is een uitbreiding van het Glicko-systeem, waarbij er rekening wordt gehouden met de marges waarmee een team wint, want bij voetbal is er uiteraard een groot verschil tussen een 1-0 en een 6-0 overwinning. De drie systemen worden met elkaar vergeleken met behulp van een bepaalde functie, en op basis van deze functie kunnen we concluderen dat het MOV-systeem hier het beste werkt, gevolgd door het Glicko-systeem en dan het Elo-systeem.

Samenvatting

Om de sterkte van een voetbalteam bij te houden kunnen we kijken naar de rating van een team. De rating is een bepaald getal, waarvan we willen dat hij hoger is, hoe beter een team speelt. Om voetbalteams te rangschikken, kunnen we hun ratings uitrekenen met behulp van bepaalde systemen. Iedere maand waarin een team wedstrijden speelt, wordt hun rating geüpdatet op basis van hun resultaten in die maand.
In dit verslag wordt naar drie van dit soort systemen gekeken: ten eerste het meest bekende systeem, het Elo-systeem, waarbij de rating altijd proportioneel aan het verschil tussen het resultaat en het verwachte resultaat van een wedstrijd verandert. Deze proportie staat bekend als de K-waarde, en wordt vast gekozen. Ten tweede is er het Glicko-systeem, een ingewikkeldere versie van het Elo-systeem, waarbij de K-waarde niet vast is, maar steeds verandert, waardoor er meer rekening wordt gehouden met de onzekerheid over de ratings. De systemen worden gemodelleerd met behulp van een normale verdeling, waarbij de μ als de rating wordt gekozen en de σ^2 als de onzekerheid. Deze onzekerheid zal steeds omlaag gaan (en hoe lager de onzekerheid, hoe minder de rating zal veranderen), hoe meer wedstrijden een team speelt, maar om ervoor te zorgen dat hij niet te laag wordt, wordt iedere maand een bepaalde waarde aan σ^2 toegevoegd, de toename in variantie. Ten slotte is er het MOV-systeem, een uitbreiding van het Glicko-systeem, waarbij er rekening wordt gehouden met de echte eindstand van een wedstrijd. In voetbal is er namelijk een groot verschil tussen een 1-0 overwinning en een 6-0 overwinning, ook al worden die resultaten in het Elo-systeem en het Glicko-systeem als hetzelfde gezien.
Alle drie de systemen worden met elkaar vergeleken met behulp van een bepaalde functie, de discrepantie-functie. Elk systeem hangt af van bepaalde constanten, zoals bijvoorbeeld de K-waarde. Deze constanten zullen met behulp van de secant methode op zo een manier worden gekozen, dat de discrepantie-functie geminimaliseerd wordt. Het systeem waarbij de discrepantie-functie het laagst is, zal gekozen worden als het beste systeem. De verwachting is dat het Elo-systeem de hoogste discrepantie zal hebben, gevolgd door het Glicko-systeem en dan het MOV-systeem.
Met behulp van Python-codes worden de ratings van Eredivisie en Eerste Divisie teams over de laatste tien seizoenen uitgerekend, en daarmee worden de discrepanties van de systemen ook uitgerekend, en wat blijkt is dat het Elo-systeem inderdaad de hoogste discrepantie heeft, en dus het minst goed werkt, gevolgd door het Glicko-systeem en dan het MOV-systeem. Het MOV-systeem werkt hier dus inderdaad het beste.

Inhoudsopgave

1

Inleiding

Ranglijsten kunnen op veel verschillende manieren worden opgezet: ze kunnen gemaakt worden op basis van absolute getallen, bijvoorbeeld de meest verkochte automerken; we kunnen om de meningen van mensen vragen, en de ranglijsten van die mensen combineren in een ranglijst, maar waar het in dit verslag over gaat zijn gevallen waarin steeds twee dingen met elkaar vergeleken worden, en waarbij een van de twee dan een bepaalde voorkeur krijgt. Een voorbeeld waar dit handig is om te gebruiken is in sport, en hier zal dan ook het voorbeeld van voetbal gebruikt worden.
Specifiek zal er gekeken worden naar alle wedstrijden die over de laatste tien jaar in de Eredivisie en de Eerste Divisie zijn gespeeld, inclusief de wedstrijden in de nacompetitie en de play-offs voor Europees voetbal, en op basis van hun resultaten worden de teams gerangschikt. Niet alleen wordt er gekeken naar welke teams boven welke staan, maar ook wat de ratings van alle teams zijn, om een beter beeld te geven van hoe goed elk team ongeveer is.
In Hoofdstuk 2 worden diverse rating-systemen uitgelegd, die gebruikt kunnen worden voor het maken van zo een ranglijst; in Hoofdstuk 3 wordt uitgelegd hoe deze systemen met elkaar vergeleken zullen worden; in Hoofdstuk 4 worden de resultaten van deze vergelijkingen gepresenteerd; in Hoofdstuk 5 worden op basis van deze resultaten conclusies getrokken over deze rating-systemen; en tot slot worden bepaalde aspecten van het model en de resultaten gediscussieerd.
Aan het eind van dit verslag staat er nog een Appendix met de Python-codes die gebruikt werden voor het maken van de ranglijsten.

2

Rating-systemen

Voor het geven van ratings aan teams bestaan er veel verschillende systemen. Waarschijnlijk het bekendste voorbeeld hiervan is het systeem gemaakt door Elo (1978), wat gebruikt wordt voor allerlei sporten, games en voornamelijk schaken. Mensen hebben over de jaren systemen ontwikkeld om de ratings van teams te schatten op een betere manier dan het Elo-systeem. Een van de voornaamste van deze systemen is het systeem gemaakt door Glickman (1999), het Glicko-systeem. Daarnaast zal er nog naar een derde systeem gekeken worden. Die heeft te maken met het feit dat ze bij Elo en Glicko alleen onderscheid maken tussen een overwinning, een verlies of een gelijkspel, wat niet goed bij voetbal past, aangezien er wel onderscheid is tussen een 6-0 overwinning en een 2-1 overwinning. De Elo en Glicko-systemen kunnen worden uitgebreid om de 'Margin Of Victory' (MOV) te implementeren, zoals gedaan is door Kovalchik (2020) en door Hvattum en Arntzen (2010). Dit systeem zal in dit verslag het MOV-systeem genoemd worden. Alle drie deze systemen werken op een soortgelijke manier, maar er zijn wel verschillen in hoe de ratings precies veranderen.

2.1. Algemene idee

Om een goed beeld te kunnen geven van de sterkte van een team, willen we van ieder team twee dingen weten: hun rating uiteraard, en de onzekerheid over die rating, want hoe onzekerder de rating van een team is, hoe meer die rating moet veranderen op basis van de uitslag van een wedstrijd. Om dit te kunnen modelleren nemen we aan dat de sterkte van een team een normale verdeling heeft, met een gemiddelde μ en een variantie σ^2. Voor dit model zullen we zeggen dat de μ de rating van het team is en de σ^2 de onzekerheid is. Aan het begin heeft elk team een bepaalde μ en σ^2. Vervolgens worden voor ieder team iedere maand de μ's en σ^2's geüpdatet op basis van de resultaten van dat team in die maand. Daarna wordt er gekeken naar de volgende maand, en worden de variabelen weer geüpdatet. Dit wordt dan herhaald tot de laatste maand. Hoe deze variabelen iedere maand worden geüpdatet is anders op basis van welk systeem wordt gebruikt, maar er is wel iets wat ze allemaal gemeen hebben: bij elk systeem wordt er gekeken naar het verschil tussen het resultaat van een wedstrijd en de verwachte waarde van dat resultaat. Als dit verschil positief is (dus als een beter resultaat wordt behaald dan verwacht), dan gaat de rating omhoog, en anders gaat hij omlaag.

2.2. Initialisatie

Om te beginnen krijgt elk team een bepaalde rating: er wordt gekeken naar de wedstrijden van Eredivisie en Eerste Divisie clubs over tien jaar: het begint in het seizoen 2014-2015 en eindigt in het seizoen 2023-2024. De teams die in het eerste seizoen in de Eredivisie speelden beginnen met een μ van 1600 en de Eerste Divisie teams beginnen met een μ van 1400. Stel dat elk team met 1500 zou beginnen: als een Eredivisie-team veel wedstrijden zou winnen tegen andere Eredivisie-teams, zou dat team een hoge rating krijgen. Als een Eerste Divisie team veel wedstrijden tegen andere Eerste Divisie teams zou winnen, zou dat team ook een hoge rating krijgen. Deze twee teams zouden dan dus allebei hoge ratings hebben, aangezien hun tegenstanders soortgelijke ratings hadden, ook al heeft het Eredivisie-team tegen veel betere teams gespeeld. Om ervoor te zorgen dat de Eerste Divisie teams geen te hoge ratings krijgen beginnen ze met lagere ratings. De slechtste teams van de Eredivisie gaan aan het eind van het seizoen naar de Eerste Divisie en de beste teams van de Eerste Divisie naar de Eredivisie, en dat zal er ook voor zorgen dat ze meer tegen elkaar spelen en dat de ratings voor de slechtere teams dus ook niet te hoog worden.

Naast de μ begint elk team ook met een bepaalde variantie σ^2: voor het Glicko en het MOV-systeem zal een begin-variantie van 40000 worden gekozen. Deze is aan het begin heel hoog om te compenseren voor het feit dat de teams die tegen elkaar spelen allemaal met dezelfde ratings beginnen, en er dus veel onzekerheid over hun ratings moet zijn.

2.3. Elo-systeem

Het Elo-systeem is het enige waarbij de σ^2 voor elk team niet echt zal veranderen. Iedere maand wordt de μ voor elk team geüpdatet op basis van hun resultaten. Voor het Elo-systeem wordt het als volgt veranderd:

$$\mu' = \mu + K \sum_{j=1}^{n} [r_j - We(\mu, \mu_j)] \tag{2.1}$$

met

$$We(\mu, \mu_j) = \frac{1}{1 + 10^{-(\mu - \mu_j)/400}}$$

Bij het updaten van de μ van het team wordt er gesommeerd over elke wedstrijd die in die maand door dat team gespeeld wordt. Een team speelt n wedstrijden in een maand, r_j is het resultaat van wedstrijd j (1 als het team wint, 1/2 bij een gelijkspel en 0 bij een nederlaag) en $We(\mu, \mu_j)$ is het verwachte resultaat van die wedstrijd, afhankelijk van de μ van het team en μ_j, de μ van de tegenstander in wedstrijd j. In het originele model van Elo (1978) werd een andere functie voor $We(\mu, \mu_j)$ gebruikt, maar sindsdien is er een betere functie gevonden voor het representeren van het verwachte resultaat van een wedstrijd en dus zal die gebruikt worden. K is een constante die bepaalt hoeveel de ratings ongeveer moeten veranderen. K is een positief getal en daarom is het volgende handig om te zien: als $r_j - We(\mu, \mu_j)$ positief is (dus als het resultaat beter is dan verwacht), dan gaat de μ omhoog, en anders gaat hij omlaag.
In de praktijk wordt de waarde van K meestal niet altijd hetzelfde gehouden. Sporten die gebruik maken van Elo, gebruiken vaak ietwat aangepaste versies. De Wereldschaakbond FIDE gebruikt bijvoorbeeld een Elo-systeem waarbij spelers een K-waarde van 20 hebben, totdat ze boven een bepaalde rating komen. Dan gaat hun K-waarde omlaag naar 10. In het model in dit verslag zal dit niet gebeuren en zal de K-waarde hetzelfde blijven. Hoe deze K-waarde precies gekozen zal worden, wordt in Hoofdstukken 3 & 4 uitgelegd.
Omdat er niet echt rekening wordt gehouden met de onzekerheid in sterkte van teams, hebben mensen systemen verzonnen, die soortgelijk aan Elo zijn, maar wel de onzekerheid modelleren.

2.4. Glicko-systeem

Een van dit soort systemen is het Glicko-systeem (Glickman, 1999). In dit systeem wordt voor elk team iedere maand niet alleen de μ, maar ook de σ^2 geüpdatet. Iedere wedstrijd, die een bepaald team speelt, weten we meer over de sterkte van een team en is het dus logisch als de σ^2 omlaag gaat. Het is echter niet echt realistisch als de onzekerheid over de rating van een team alleen maar omlaag gaat, aangezien de sterktes van teams nogal veel veranderen over de tijd, vooral in voetbal waar er iedere zomer een paar maanden niet wordt gespeeld en clubs nieuwe spelers halen. Om hiervoor te compenseren wordt er aan elke σ^2 iedere maand een bepaalde waarde toegevoegd, zodat de σ^2 niet te laag wordt. Deze waarde wordt aangegeven met ν^2 en is de toename in variantie over de tijd. Met dat in gedachten worden de variabelen als volgt geüpdatet:

$$\mu' = \mu + \frac{q}{\frac{1}{\sigma^2} + \frac{1}{\delta^2}} \sum_{j=1}^{n} g(\sigma_j^2)[r_j - E(r|\mu, \mu_j, \sigma_j^2)] \tag{2.2}$$

$$\sigma^{2'} = (\frac{1}{\sigma^2} + \frac{1}{\delta^2})^{-1} + \nu^2 \tag{2.3}$$

met

$$q = \frac{\ln(10)}{400}$$

$$g(\sigma^2) = \frac{1}{\sqrt{1 + 3q^2\sigma^2/\pi^2}}$$

$$E(r|\mu, \mu_j, \sigma_j^2) = \frac{1}{1 + 10^{-g(\sigma_j^2)(\mu - \mu_j)/400}}$$

$$\delta^2 = [q^2 \sum_{j=1}^{n} g(\sigma_j^2) E(r|\mu, \mu_j, \sigma_j^2)(1 - E(r|\mu, \mu_j, \sigma_j^2))]^{-1}$$

Als we (2.1) en (2.2) vergelijken, zien we dat ze nogal op elkaar lijken. In tegenstelling tot Elo, waarbij we een vaste K-waarde hadden, verandert de K-waarde nu op basis van de σ^2 van het team. Daarnaast zitten er nu de functies g en E in om rekening te houden met de variantie. Als we naar (2.2) kijken zien we het volgende: als σ_j^2, de onzekerheid over de rating van tegenstander j, gelijk is aan 0 (dus als er geen onzekerheid over de sterkte van tegenstander j is), dan is $g(\sigma_j^2)$ gelijk aan 1 en is $E(r|\mu, \mu_j, \sigma_j)$ gelijk aan $We(\mu, \mu_j)$: dus dan hebben we formule (2.1) met een veranderende K.

2.5. MOV-systeem

Het Glicko-systeem is dus een manier om de onzekerheid over de sterkte van een team te implementeren in het Elo-systeem, maar waar het geen rekening mee houdt is de marge waarmee een team wint, iets dat vooral bij voetbal relevant is. Om deze marges te implementeren voegen we een variabele toe aan de update-formule van het Glicko-systeem. Dit kan op meerdere manieren gedaan worden (Kovalchik, 2020), maar hier zal het als volgt worden gedaan:

$$\mu' = \mu + \frac{q}{\frac{1}{\sigma^2} + \frac{1}{\delta^2}} \sum_{j=1}^{n} s(1 + d_j)^{\alpha} g(\sigma_j^2)[r_j - E(r|\mu, \mu_j, \sigma_j^2)] \tag{2.4}$$

De formule voor σ^2 blijft hetzelfde. d_j is het doelpunten-verschil van wedstrijd j: die is dus 3 bij een 6-3 overwinning of een 1-4 overwinning. Er wordt geen onderscheid gemaakt tussen een 2-0 en een 4-2 overwinning en er wordt ook geen onderscheid gemaakt tussen thuis- en uitwedstrijden (Davidson & Beaver, 1977). Voor de rest zijn s en α bepaalde constanten die gekozen zullen worden. α beslist hoeveel meer de rating moet veranderen op basis van hoeveel meer doelpunten er worden gemaakt en s is een constante die ervoor zorgt dat de ratings niet te veel veranderen. Iets om op te merken is dat als α gelijk is aan 0 en s gelijk is aan 1, dat we de functies voor het Glicko-systeem krijgen.
Voor alle drie de systemen hebben we nu wat onbekende constanten: voor het Elo-systeem hebben we de K, voor het Glicko-systeem hebben we de v^2 en voor het MOV-systeem hebben we de v^2, de s en de α. Hoe de waarden voor deze constanten gekozen worden, wordt in het volgende hoofdstuk uitgelegd. Daarnaast wordt er uitgelegd hoe deze systemen precies met elkaar vergeleken zullen worden.

3

Analyse

Nu dat we deze drie systemen hebben, is het handig om ze te kunnen vergelijken om te zien welke het beste werkt. De verwachting is dat het Elo-systeem het het slechtste zal doen, gevolgd door het Glicko-systeem en dan het MOV-systeem, aangezien de modellen realistischer lijken te worden, maar of dit ook echt zo is zal blijken.

3.1. Discrepantie

Om te beslissen welke van de systemen het beste werkt, zal een bepaalde variabele gebruikt worden: de discrepantie tussen de uitslag van een wedstrijd en de verwachte waarde van die uitslag:

$$d_{ij} = -s_{ij}\ln(p_{ij}) - (1-s_{ij})\ln(1-p_{ij}) \tag{3.1}$$

met

$$p_{ij} = \frac{1}{1+10^{-g(\sigma_i^2+\sigma_j^2)(\mu_i-\mu_j)/400}}$$

Hier is s_{ij} de uitslag van een wedstrijd tussen team i en team j, en p_{ij} is de kans dat team i van team j wint. Iets om op te merken is het feit dat de functie van de discrepantie komt van de log-likelihood van een Bernoulli model:

$$\ln(\ell(p|s)) = s\ln(p) + (1-s)\ln(1-p) \tag{3.2}$$

Deze lijkt heel erg op (3.1), op een min-teken na. Omdat de $\ln(p_{ij})$ altijd negatief is (omdat p_{ij} tussen 0 en 1 zit), is het handig om het met -1 te vermenigvuldigen, zodat de discrepantie een positief getal is.
Stel dat we gaan kijken naar de som van een boel discrepanties. Die som zal er dan ongeveer zo uit zien (als we gelijkspelen even negeren):

$$-\ln(p_1) - \ln(p_2) - \ldots - \ln(p_n)$$

met p_1 de kans dat de winnaar van wedstrijd 1 ook echt zou winnen, enzovoorts. Omdat het logaritmes zijn, is deze som hieraan gelijk:

$$-\ln(p_1 p_2 \ldots p_n)$$

Het is dus de ln van het product van een boel kansen, en dit product van kansen is uiteraard gelijk aan de kans dat alle teams die gewonnen hadden, hun wedstrijden ook zouden winnen, aannemende dat de wedstrijden onafhankelijk zijn. Dit is een reden voor waarom de log-likelihood handig is om te gebruiken.

Stel dat we kijken naar de verwachting van de discrepantie: we zeggen hier dat q de daadwerkelijke kans is dat team i wint (en we negeren hier even gelijkspelen):

$$\begin{aligned} E[\ln(\ell(p|s))] &= P(\text{Team i wint}) * \ln(\ell(p|s=1)) + P(\text{Team i verliest}) * \ln(\ell(p|s=0)) \\ &= -q * \ln(p) - (1-q) * \ln(1-p) \end{aligned}$$

Als we deze verwachting nu willen minimaliseren, kunnen we de afgeleide nemen en die dan gelijk zetten aan nul, en dan krijgen we het volgende:

$$0 = -\frac{q}{p} + \frac{1-q}{1-p}$$

Deze vergelijking klopt als p gelijk is aan q, dus de verwachting van de discrepantie is minimaal als de kans p gelijk is aan de daadwerkelijke kans q.
Omdat de discrepantie de log-likelihood is, is het volgende logisch om te zien: stel dat team i wint, dan wordt de formule voor de discrepantie $d_{ij} = -\ln(p_{ij})$. Hoe hoger de kans nu was dat i van j zou winnen (dus hoe hoger p_{ij}), hoe lager het verschil tussen het verwachte resultaat en het echte resultaat, en dus hoe lager de discrepantie moet worden. Stel nu aan de andere kant dat team i de wedstrijd verliest, dan wordt de formule voor de discrepantie $d_{ij} = -\ln(1-p_{ij})$. Hoe hoger nu de kans was dat i van j zou winnen, hoe groter het verschil tussen de echte uitkomst en de verwachte uitkomst, en dus hoe groter de discrepantie. Bij een gelijkspel wordt een gemiddelde van de twee waarden voor d_{ij} genomen, waarbij de discrepantie lager wordt, hoe dichter p_{ij} bij een half zit. Om te kunnen bepalen welk van de drie systemen het beste werkt zullen de discrepanties voor elk van de systemen met elkaar vergeleken worden, en in het specifiek zal gekeken worden naar welke de laagste discrepanties geeft, omdat die dan het beste de uitslagen van wedstrijden voorspelt.
De p_{ij} hangt af van de μ en de σ^2 van de teams en deze hangen dan weer deels af van de onbekende constanten in de systemen: K voor het Elo-systeem, v^2 voor het Glicko-systeem, en v^2, s en α voor het MOV-systeem. Deze constanten zullen op zo een manier gekozen worden dat de discrepantie voor het hele systeem geminimaliseerd wordt. Hoe deze constanten gekozen worden, wordt in sectie 3.3 uitgelegd.

3.2. Berekening van de Discrepantie

De totale discrepantie voor een model zal als volgt uitgerekend worden: omdat de teams grotendeels met dezelfde ratings beginnen, zijn de waarden voor p_{ij} en dus ook de waarden voor de discrepantie aan het begin niet heel betrouwbaar, en daarom zal er de eerste twee seizoenen niet gekeken worden naar de discrepanties voor die wedstrijden, maar de ratings zullen op basis van deze twee seizoenen wel iedere maand geüpdatet worden. Beginnende met het derde seizoen worden de discrepanties voor iedere wedstrijd in die maand gesommeerd. Vervolgens worden de μ's en de σ^2's op basis van de uitslagen in die maanden geüpdatet. Dit wordt herhaald voor elke maand in het vierde en het vijfde seizoen. Dan hebben we dus een som van discrepanties over drie seizoenen. Deze som wordt geminimaliseerd door de waarden van de onbekende constanten te veranderen. Wanneer we de optimale waarden voor deze constanten hebben gevonden, zullen deze waarden gebruikt worden voor het uitrekenen van de som van de discrepanties van de laatste vijf seizoenen. Het rating-systeem waarbij deze som het laagst is, is het systeem waarvan we hier zullen zeggen dat die het beste werkt.
Het opsplitsen van de seizoenen wordt gedaan om overfitten te voorkomen. Als we de waarden van de constanten zo kiezen dat de discrepantie van de laatste acht seizoenen samen geminimaliseerd wordt, dan werken die waarden goed voor deze data-set, maar het kan zijn dat het niet goed werkt als we naar een andere data-set willen kijken, of als we aan deze data-set meer wedstrijden willen toevoegen, bijvoorbeeld meer seizoenen. Daarom is het belangrijk dat er geen overlap in de seizoenen zit, dus de eerste twee seizoenen worden gebruikt om ervoor te zorgen dat de ratings niet allemaal hetzelfde zijn, de volgende drie seizoenen worden gebruikt voor het vinden van optimale waarden voor de constanten, en de laatste vijf seizoenen worden gebruikt voor het uitrekenen van de discrepanties van de systemen, waaruit de conclusie zal worden getrokken welk het systeem het beste werkt.

3.3. Optimalisatie van de Constanten

Voor het vinden van optimale waarden voor de constanten zal voor alle drie de systemen hetzelfde idee worden gebruikt. We gaan kijken naar de functie $d(K)$ voor het Elo-systeem en de functie $d(v^2, s, \alpha)$ voor het Glicko en het MOV-systeem. De functie d is de discrepantie en we kunnen voor Glicko en MOV dezelfde functie gebruiken aangezien MOV en Glicko hetzelfde zijn als α gelijk is aan 0 en s gelijk is aan 1. Om het minimum van de functie $d(K)$ te vinden, wordt het nulpunt van $d'(K)$ gevonden met behulp van een numerieke methode. Er zijn twee methoden waar hier naar gekeken worden: de Newton-Raphson methode en de secant methode. Bij de Newton-Raphson methode beginnen we met een punt x_0 niet al te ver van het nulpunt en vervolgens wordt de volgende stap toegepast:

$$x_{n+1} = x_n - \frac{d'(x_n)}{d''(x_n)}$$

Bij de secant methode gebruiken we een soortgelijke stap, maar we beginnen wel met twee punten x_0 en x_1:

$$x_{n+1} = x_n - d'(x_n)\frac{x_n - x_{n-1}}{d'(x_n) - d'(x_{n-1})}$$

Deze stappen kunnen herhaald worden tot we een punt krijgen dat dicht genoeg bij het nulpunt in de buurt zit. Allebei deze methoden werken goed voor het vinden van nulpunten en ze hebben allebei hun voordelen: de Newton-Raphson methode convergeert sneller naar het nulpunt en bij de secant methode is het niet nodig om $d''(x)$ uit te rekenen. Om precies deze reden zal de secant methode gebruikt worden, aangezien het uitrekenen van $d''(x)$ voor veel extra berekeningen zorgt. Neem bijvoorbeeld $d'(x)$; deze wordt benaderd met behulp van de definitie van de afgeleide:

$$d'(x) = \frac{d(x+h) - d(x)}{h}$$

met h een heel klein getal. Voor het uitrekenen van $d'(x)$ moeten we dus twee keer de discrepantie uitrekenen. $d''(x)$ is dan weer de afgeleide hiervan:

$$d''(x) = \frac{d'(x+h) - d'(x)}{h} = \frac{d(x+2h) - 2d(x+h) + d(x)}{h^2}$$

Hiervoor moeten we dus drie keer de discrepantie uitrekenen. Het is dus sneller om de secant methode te gebruiken, aangezien we daar iedere stap minder vaak discrepanties moeten uitrekenen, wat een groot verschil maakt, omdat het uitrekenen van de discrepanties een lange tijd duurt vanwege de grote hoeveelheid berekeningen om de ratings te vinden.
De secant methode kan dus gebruikt worden om de optimale waarde voor K te vinden bij het Elo-systeem en de waarde voor v^2 voor het Glicko-systeem, maar voor het MOV-systeem is het iets anders, aangezien de functie d van meerdere variabelen afhangt. Er bestaan meerdere numerieke methoden voor het minimaliseren van een functie met meerdere variabelen. Die zijn echter nogal ingewikkeld en daarom zal een wat simpelere methode gebruikt worden: we kiezen bepaalde waarden voor s en α en gebruiken dan de secant methode om de optimale waarde voor v^2 te vinden. Vervolgens kiezen we deze waarde voor v^2 en een bepaalde waarde voor α, en gebruiken we de secant methode om een optimale waarde voor s te vinden. Dan kiezen we de waarde voor v^2 en s, en vinden we de optimale waarde voor α. Dit proces wordt vervolgens herhaald: we vinden steeds nieuwe optimale waarden voor v^2, daarna s, α en dan weer terug naar v^2 totdat de waarden niet meer veel veranderen. Zo kunnen we ook $d(v^2, s, \alpha)$ minimaliseren. In het volgende hoofdstuk worden de resultaten hiervan laten zien

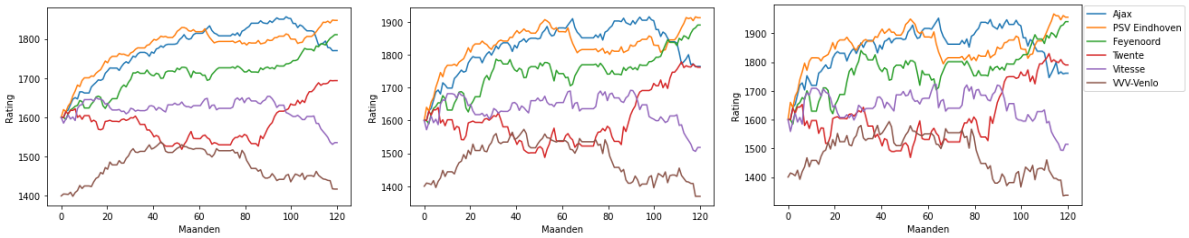
4

Resultaten

Met alle drie de rating-systemen kunnen nu de ratings van de teams over de loop van de jaren uitgerekend worden. Dit wordt gedaan met Python-codes, die in Appendix A staan. Eerst wordt de nodige data verzameld: voor dit model zal de site fbref.com gebruikt worden, waar de uitslagen van alle wedstrijden, die de teams over de laatste tien seizoenen hebben gespeeld, op staan. Er kunnen naar meer seizoenen worden gekeken, maar tien zal genoeg zijn. Vervolgens worden de ratings van de teams op basis van die uitslag iedere maand aangepast. Hoe deze aangepast worden ligt dus aan het systeem en de constanten. Deze constanten worden gevonden met behulp van de methoden uit Hoofdstuk 3. Met behulp van deze constanten worden dan de discrepanties van de systemen uitgerekend. De verwachting is dat het Elo-systeem de hoogste discrepantie zal hebben, gevolgd door het Glicko-systeem en dan het MOV-systeem.

4.1. Elo-systeem

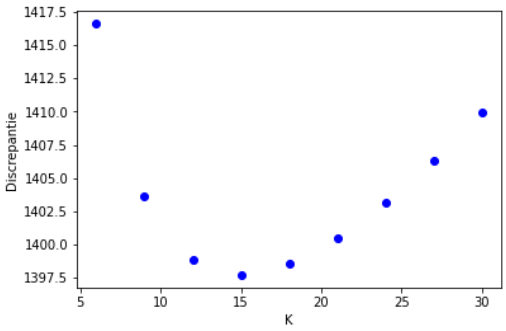
Het Elo-systeem hangt af van de waarde van de constante K, dus de ratings zien er op basis van K anders uit. De ratings veranderen voor bepaalde waarden voor K voor een paar van de teams op de volgende manier:



(a) $K = 10$ (b) $K = 20$ (c) $K = 30$

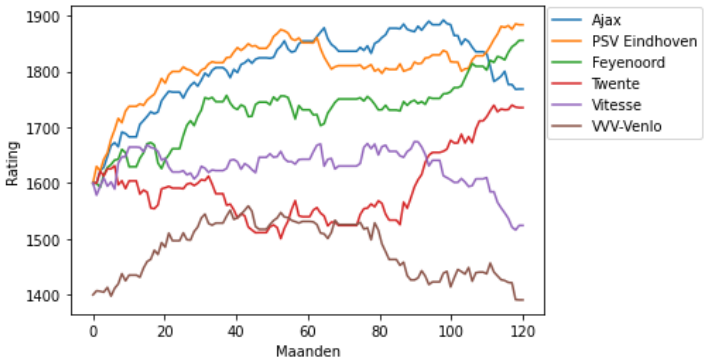
Figuur 4.1: Ratings voor Elo-systeem

Er is te zien dat de Eredivisie teams aan het begin een rating van 1600 hebben en de Eerste Divisie teams een rating van 1400. Vervolgens gaan de ratings van de beste teams (Ajax, PSV en Feyenoord) redelijk snel omhoog. Hoe hoger de K-waarde, hoe meer de rating zal veranderen, wat de hoge verandering in Figuur (c) verklaart. Elk van de teams lijkt een logische progressie te hebben: Ajax was een lange tijd een van de beste teams van het land, maar is de laatste twee seizoenen iets achteruit gegaan; PSV is een soortgelijk geval, maar zij zijn het laatste seizoen juist veel beter geworden; Feyenoord was lange tijd de derde club, maar is de laatste twee seizoenen een stuk beter geworden; Twente en Vitesse zijn normaal goede teams, maar hebben lagere ratings in de jaren dat ze degraderen, en VVV Venlo begon in de Eerste Divisie, maar is sinds die tijd meerdere keren gepromoveerd en gedegradeerd.
Alle ratings lijken dus logisch, maar om erachter te komen voor welke K het systeem hier het best werkt, wordt de methode van Hoofdstuk 3 toegepast: er wordt uitgerekend voor welke waarde van K de som van de discrepanties van seizoenen 3, 4 en 5 minimaal is:



Figuur 4.2: Discrepanties voor bepaalde waarden van K

Dit zijn dus de waarden voor de discrepanties met bepaalde K's. De minimale discrepantie lijkt bij een K rond de 15 te zijn, en als we de optimalisatie toepassen op het Elo-model, is dat ook inderdaad de K waarvoor de discrepanties het laagst is: deze discrepantie is dan gelijk aan 1397,79. De ratings zien er met een K-waarde van 15 nu zo uit:

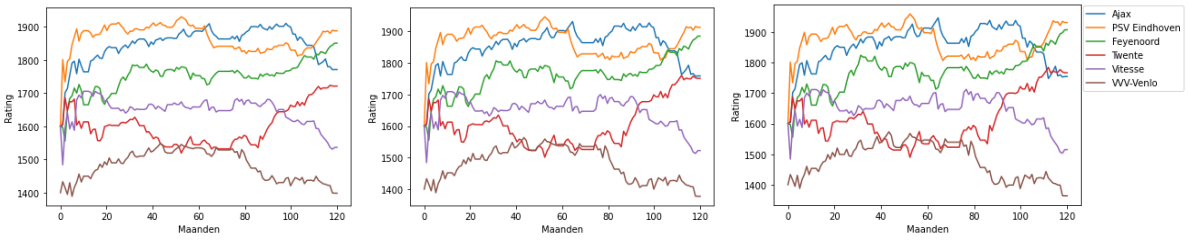


Figuur 4.3: Ratings voor $K = 15$

Nu kunnen we de discrepantie voor de laatste vijf seizoenen uitrekenen. Dit is dan de discrepantie die vergeleken zal worden met de discrepanties van de andere systemen. Voor een K van 15 is deze discrepantie gelijk aan 2031,42. Deze getallen voor de discrepantie betekenen nu niet veel, maar het belangrijke is hoe ze vergelijken met de discrepanties van de andere systemen.

4.2. Glicko-systeem

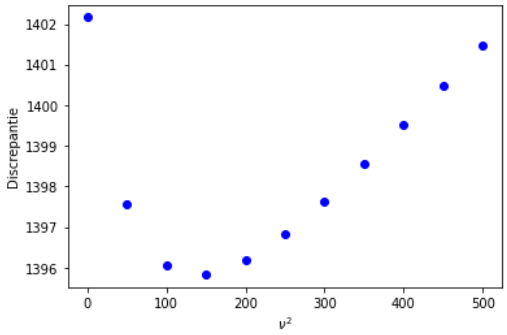
Bij het Glicko-systeem is er geen vaste K, maar wel een v^2 die gekozen zal worden. Voor bepaalde waarden voor v^2 ziet het er zo uit:



(a) $v^2 = 100$ (b) $v^2 = 200$ (c) $v^2 = 300$

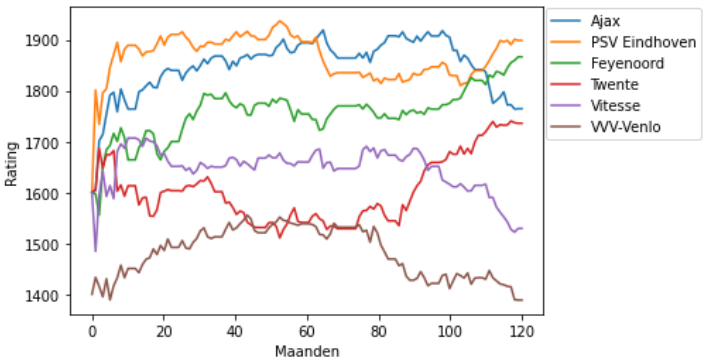
Figuur 4.4: Ratings voor Glicko-systeem

Het lijkt wel op het Elo-systeem, maar er zijn een paar dingen die er anders uit zien, voornamelijk het begin: omdat de start-variantie zo hoog is (40000), is er aan het begin veel verandering in de ratings. Op dat moment is er namelijk veel onzekerheid over de ratings. De variantie gaat wel redelijk snel omlaag, en dan veranderen de ratings een stuk minder. Door de v^2 is er geen moment dat de variantie te klein wordt en de ratings niet snel genoeg veranderen. Wat betreft de discrepantie ziet het er als volgt uit op basis van de v^2:



Figuur 4.5: Discrepanties voor bepaalde waarden van v^2

Hier lijkt de minimale discrepantie bij een v^2 van 150 te liggen. Als we de optimalisatie toepassen, zien we dat de discrepantie geminimaliseerd word bij een v^2 van ongeveer 140: de discrepantie is dan gelijk aan 1395,82. Deze is al iets lager dan die van het Elo-systeem: in sectie 4.4 zullen de discrepanties helemaal vergeleken worden. De ratings zien er bij een v^2 van 140 zo uit:

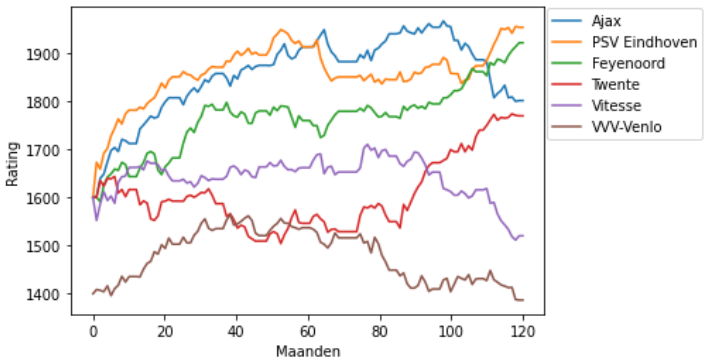


Figuur 4.6: Ratings voor $v^2 = 140$

De discrepantie van de laatste vijf seizoenen komt bij een v^2-waarde van 140 uit op 2027,9.

4.3. MOV-systeem

Voor het MOV-systeem zijn er drie variabelen die geoptimaliseerd moeten worden: de v^2, de s en de α. Omdat er meerdere zijn, is het niet heel handig om grafieken te laten zien, dus er zal eerst verteld worden wat de optimale waarden voor de constanten zijn: de minimale discrepantie kan gevonden worden met behulp van de optimalisatie methode uit Hoofdstuk 3, waarmee een discrepantie van 1390,83 gevonden wordt bij v^2 gelijk aan 2000, s gelijk aan 0,25 en α gelijk aan 0,3. De v^2 is hier veel hoger dan in het Glicko-systeem, maar dit wordt gecompenseerd met een lage s en een lage α, die ervoor zorgen dat de ratings niet te veel veranderen. Hoe ze dan wel veranderen is te zien op de volgende pagina in Figuur 4.7.



Figuur 4.7: Ratings voor MOV-systeem

Zoals te zien is, veranderen de ratings aan het begin nog wel veel, maar zeker niet zo veel als bij het Glicko-systeem. Voor de rest is er niet zo veel over op te merken, maar het belangrijke is de discrepantie. Voor de laatste vijf seizoenen is de discrepantie voor het MOV-systeem met de eerder genoemde constanten gelijk aan 2024,58. Nu dat we voor ieder team de ratings hebben, kunnen we een ranglijsten maken van deze ratings. Op de volgende pagina staat de ranglijst van de ratings na de laatste maand. Deze kunnen we nu vergelijken met de eindstanden van de Eredivisie en de Eerste Divisie van het laatste seizoen, die ook op de volgende pagina staan.
De plaatsingen zijn grotendeels soortgelijk: de grootste verschillen zitten in de teams die vooral in het laatste seizoen het een stuk beter of slechter deden dan normaal, zoals verwacht, aangezien hun ratings nog niet te veel zijn veranderd.

4.4. Vergelijking

Nu dat we de discrepanties voor alle drie de systemen hebben, kunnen we ze met elkaar vergelijken:

Discrepantie	Elo	Glicko	MOV
Seizoenen 3-5	1397,79	1395,82	1390,83
Seizoenen 6-10	2031,42	2027,9	2024,58

Zoals verwacht wordt de discrepantie steeds lager. Bij de seizoenen 3 tot en met 5 is het discrepantie-verschil tussen MOV en Glicko ruim twee keer zoveel als het verschil tussen Glicko en Elo, maar bij de laatste vijf seizoenen is het verschil dus heel vergelijkbaar. Het verschil tussen Elo en MOV blijft ongeveer hetzelfde maar het Glicko-systeem lijkt het wat beter te doen als het gaat om de laatste vijf seizoenen. Blijkbaar is het verschil tussen hoe goed Elo en Glicko werken redelijk vergelijkbaar met het verschil tussen hoe goed Glicko en MOV werken. Iets om op te merken aan de data is het volgende: als we kijken naar de grafieken van de ratings, bijvoorbeeld Figuur 4.7, is het te zien dat de ratings in seizoenen 3-5 niet veel veranderen, maar in seizoenen 6-10 juist wel, omdat de teams nu minder consistent lijken te zijn. Blijkbaar werkt het Glicko-systeem dus beter als de ratings wat meer veranderen, wat logisch is, aangezien het Glicko-systeem zo gemaakt is om goed te werken, zelfs als de ratings redelijk veel veranderen: iets wat iets minder het geval is bij het MOV-systeem, aangezien de s en de α redelijk klein zijn.

	Team	μ
1	PSV	1945,5
2	Feyenoord	1922,4
3	Ajax	1802,1
4	AZ	1776,8
5	FC Twente	1770,3
6	FC Utrecht	1662,6
7	Sparta Rotterdam	1634,4
8	NEC	1617,2
9	Go Ahead Eagles	1597,9
10	SC Heerenveen	1559,7
11	PEC Zwolle	1546,1
12	Willem II	1546,0
13	Heracles Almelo	1533,1
14	Fortuna Sittard	1528,0
15	Vitesse	1520,6
16	FC Groningen	1513,6
17	RKC Waalwijk	1508,7
18	Almere City FC	1504,7
19	Excelsior	1485,5
20	Nac Breda	1451,7
21	Roda JC Kerkrade	1449,6
22	FC Emmen	1441,5
23	Fc Volendam	1436,8
24	ADO Den Haag	1433,6
25	De Graafschap	1425,7
26	SC Cambuur	1394,4
27	MVV Maastricht	1388,5
28	VVV Venlo	1386,8
29	FC Dordrecht	1386,2
30	Jong AZ	1375,6
31	FC Eindhoven	1370,6
32	Jong Ajax	1334,4
33	Helmond Sport	1333,9
34	Jong PSV	1313,5
35	Telstar	1306,6
36	TOP Oss	1264,6
37	FC Den Bosch	1261,2
38	Jong FC Utrecht	1206,4

	Team	W	G	V	Pt.
1	PSV	29	4	1	91
2	Feyenoord	26	6	2	84
3	FC Twente	21	6	7	69
4	AZ	19	8	7	65
5	Ajax	15	11	8	56
6	NEC	14	11	9	53
7	FC Utrecht	13	11	10	50
8	Sparta Rotterdam	14	7	13	49
9	Go Ahead Eagles	12	10	12	46
10	Fortuna Sittard	9	11	14	38
11	SC Heerenveen	10	7	17	37
12	PEC Zwolle	9	9	16	36
13	Almere City FC	7	13	14	34
14	Heracles Almelo	9	6	19	33
15	RKC Waalwijk	7	8	19	29
16	Excelsior	6	11	17	29
17	FC Volendam	4	7	23	19
18	Vitesse	6	6	22	6
1	Willem II	23	10	5	79
2	FC Groningen	22	9	7	75
3	Roda JC Kerkrade	21	12	5	75
4	FC Dordrecht	18	15	5	69
5	ADO Den Haag	17	12	9	63
6	De Graafschap	19	6	13	63
7	FC Emmen	17	6	15	57
8	NAC Breda	15	11	12	56
9	MVV Maastricht	16	8	14	56
10	Jong AZ	16	8	14	56
11	Helmond Sport	14	9	15	51
12	VVV-Venlo	13	9	16	48
13	SC Cambuur	13	8	17	47
14	FC Eindhoven	9	16	13	43
15	Jong Ajax	10	10	18	40
16	Jong PSV	11	7	20	40
17	Telstar	9	8	21	35
18	TOP Oss	10	4	24	34
19	FC Den Bosch	8	9	21	33
20	Jong FC Utrecht	5	11	22	26

Tabel 4.1: Tabellen van de ranglijst op basis van de μ na de laatste maand (links) en eindstanden Eredivisie en Eerste Divisie van het seizoen 2023/2024 (rechts). NAC werd gepromoveerd via de nacompetitie en Vitesse kreeg 18 punten in mindering.

5

Conclusie/Discussie

In dit verslag is er gekeken naar drie systemen voor het geven van ratings aan voetbalteams: het Elo-systeem, het Glicko-systeem en het MOV-systeem. Om te kunnen beslissen welke van de drie systemen het beste werkt is er gekeken naar de discrepantie van de systemen, en zijn die met elkaar vergeleken. Deze discrepanties hangen af van bepaalde constanten, die gevonden kunnen worden met de secant methode. Met behulp van deze constanten zijn de minimale discrepanties van de systemen gevonden. De resultaten lieten zien dat het Elo-systeem de hoogste discrepantie had, gevolgd door het Glicko-systeem en dan het MOV-systeem, waardoor we kunnen zeggen dat het MOV-systeem hier het beste werkt, zoals verwacht. Sterker nog, het leek erop dat het verschil tussen Elo en Glicko vergelijkbaar was met het verschil tussen Glicko en MOV.
Het model uit dit verslag kan worden uitgebreid om rekening te houden met bepaalde factoren: al zal de code wel wat aangepast moeten worden, zullen het model en de optimalisatie-methoden ook werken voor teams uit bijvoorbeeld Engeland of Duitsland. Op dit moment hangen de ratings van de teams alleen af van de wedstrijden die ze in de maanden ervoor gespeeld hebben, maar een 'parameter smoothing' functie kan worden toegepast om de ratings ook van wedstrijden te laten afhangen die later worden gespeeld (Glickman, 1999). Er bestaan meerdere manieren om MOV in een Glicko-systeem te implementeren: naast degene gebruikt in dit verslag is er bijvoorbeeld ook een logistisch model (Kovalchik, 2020). Bij deze veranderingen is het niet duidelijk wat er met de discrepantie zal gebeuren, maar bij de volgende uitbreidingen is het wel de verwachting dat de discrepanties omlaag zullen gaan, aangezien ze het model wat realistischer lijken te maken. Er wordt nu bijvoorbeeld geen onderscheid gemaakt tussen thuis- en uitwedstrijden (Davidson & Beaver, 1977), ook al is er een duidelijk voordeel voor de thuisploeg in voetbal. De start-variantie werd voor het Glicko-systeem en het MOV-systeem als 40000 gekozen, maar deze kan net zo goed geoptimaliseerd worden met behulp van de methoden uit Hoofdstuk 3 en de codes. Dit is niet gedaan, simpelweg omdat dat te veel zou zijn en te lang zou duren, maar tenzij 40000 toevallig de optimale waarde is voor de start-variantie, zullen de discrepanties voor bepaalde andere waarden lager worden.

A

Code

Run eerst BEP Scraper, daarna BEP Glicko of BEP Optimize en als laatste BEP Results. BEP Scraper haalt de uitslagen van de website fbref.com, om ze dan in BEP Glicko te gebruiken om de ratings uit te rekenen. Met BEP Optimize kunnen de waarden van de constanten geoptimaliseerd worden. Hier worden ook de relevante discrepanties uitgerekend. Ten slotte kan BEP Results gebruikt worden om de ratings of de discrepanties te plotten.

A.1. BEP Scraper

```
import time
import datetime
import requests
from bs4 import BeautifulSoup
import pandas as pd


# Beslis hoeveel seizoenen worden gebruikt

years = 10

current_date = datetime.date.today()
if current_date.month < 8:
    current_year = current_date.year - 1
else:
    current_year = current_date.year


starting_year = current_year


def get_web_page():

    # Ontvang de URL's van de webpagina's van de relevante seizoenen

    url1 = "https://fbref.com/en/comps/23/schedule/" + \
           "Eredivisie-Scores-and-Fixtures"

    url2 = "https://fbref.com/en/comps/51/schedule/"+ \
           "Eerste-Divisie-Scores-and-Fixtures"

    urllist = [url1, url2]
```

```
    for year in range(1,years):
        url3 = "https://fbref.com/en/comps/23/" + \
                str(starting_year-year) + "-" + str(starting_year-year+1) + \
                "/schedule/" + \
                str(starting_year-year) + "-" + str(starting_year-year+1) + \
                "-Eredivisie-Scores-and-Fixtures"

        url4 = "https://fbref.com/en/comps/51/" + \
                str(starting_year-year) + "-" + str(starting_year-year+1) + \
                "/schedule/" + \
                str(starting_year-year) + "-" + str(starting_year-year+1) + \
                "-Eerste-Divisie-Scores-and-Fixtures"

        urllist.append(url3)
        urllist.append(url4)

    # Gebruik de URL's om de webpagina's zelf te krijgen

    soup = []
    for index, url in enumerate(urllist):
        if len(urllist) >= 20:
            time.sleep(4)
        soup.append(BeautifulSoup(
                    requests.get(url, verify=True, timeout=20)
                    .content, "html.parser"))

        print(f"Webpagina Nummer {index+1} Ontvangen")

    return soup


def fbref_data(soup):

    # Zet de relevante data uit de webpagina in een dataframe

    stats = [pd.DataFrame()]*len(soup)
    stat_columns1 = {"sched": {"Date": 1, "Home": 3, "Score": 5, "Away": 7}}
    stat_columns2 = {"sched": {"Date": 1, "Home": 3, "Score": 4, "Away": 5}}
    s_columns = [stat_columns1, stat_columns2]*years


    for index in range(len(s_columns[12:])):
        s_columns[index+12] = stat_columns2

    for i, stat_columns in enumerate(s_columns):
        for table_name, table_dict in stat_columns.items():

            table = soup[i].select_one('table[id^="' + table_name + '"]')

            for col_name, index in table_dict.items():
                single_stat = pd.DataFrame(columns=[col_name])
                rows = list(table.tbody.find_all("tr"))
```

```
                for row in rows:
                    j = 1
                    if row.find_all("td")[j].text.strip()[:4] != "":
                        try:
                            row_year = int(row.find_all("td")[j].text[:4])
                        except ValueError:
                            j += 1
                            row_year = int(row.find_all("td")[j].text[:4])
                    else:
                        row_year = 0

                    if current_year - row_year < years:
                        columns = row.find_all("td")

                        all_values = [columns[i].text for i in range(1,7)]

                        if "" not in all_values:
                            col = columns[index+j-1].text.strip()
                            single_stat = pd.concat(
                                [single_stat,
                                 pd.DataFrame.from_records(
                                     [{col_name: col}]
                                     )],
                                 ignore_index=True,
                            )
                stats[i] = pd.concat([stats[i], single_stat], axis = 1)

    return pd.concat(stats, ignore_index = True)

if __name__ == "__main__":
    fbref_data(get_web_page()).\
        to_csv("BEP_Data.csv", sep=',', index=False, encoding='utf-8')
```

A.2. BEP Glicko

```
import pandas as pd
import math
import numpy as np
import json

# Definieer wiskundige functies

q = math.log(10)/400
g = lambda sigma2: 1/(math.sqrt(1+3*sigma2*q*q/math.pi**2))
E = lambda mu1, mu2, sigma2: 1/(1+10**(-g(sigma2)*(mu1-mu2)/400))

# Ontvang de dataframe uit de vorige code en pas hem aan
# zodat er makkelijk mee gerekend kan worden

df = pd.read_csv('BEP_Data.csv')

def marginalize(df):
    scores = df.split("  ")
    if scores[0][0] == "(":
        return 0
    return int(scores[0])-int(scores[1])
```

```
def months_only(df):
    return df[:7]

df["Score"] = df["Score"].apply(marginalize)
df["Date"] = df["Date"].apply(months_only)


for index in range(len(df["Home"])):
    if df["Home"][index] == "Sparta␣Rotterdam":
        df["Home"][index] = "Sparta␣R'dam"
    elif df["Home"][index] == "Roda␣JC␣Kerkrade":
        df["Home"][index] = "Roda␣JC"
    elif df["Home"][index] == "Go␣Ahead␣Eagles":
        df["Home"][index] = "Go␣Ahead␣Eag"
    if df["Away"][index] == "Sparta␣Rotterdam":
        df["Away"][index] = "Sparta␣R'dam"
    elif df["Away"][index] == "Roda␣JC␣Kerkrade":
        df["Away"][index] = "Roda␣JC"
    elif df["Away"][index] == "Go␣Ahead␣Eagles":
        df["Away"][index] = "Go␣Ahead␣Eag"

# Definieer de teams en splits ze op tussen Eredivisie en
# Eerste Divisie teams

teams = list(set(df["Home"]))
teams.sort()

eerste_divisie_teams = list(set(df["Home"][-100:]))
eerste_divisie_teams.extend(['Jong␣Utrecht', 'Jong␣AZ'])
edt = set(eerste_divisie_teams)

eredivisie_teams = [x for x in teams if x not in edt]

# Definieer de constanten

s_var = 40000
k = 15
nu2 = 2000
alpha = 0.3
s = 0.25

# Gebruik de dataframe om erachter te komen naar welke maanden
# gekeken moeten worden

months = list(set(df["Date"]))
months.sort()

first_month = months[0]
last_month = months[-1]

current_year = int(last_month[:4])
years = current_year-int(first_month[:4])
```

```
month_numbers = ['01','02','03','04',
                 '05','06','07','08',
                 '09','10','11','12',]


months = [f"{current_year-year}-{month_number}"
          for year in range(years+1)
          for month_number in month_numbers]

months.sort()

months = months[months.index(first_month):]
months = months[:(months.index(last_month)+1)]


def update_ratings(s_var, k, nu2, alpha, s):
    # Initialiseer alle teams en de ratinglijst
    edt_ratings = {team: [(1400, s_var)] for team in eerste_divisie_teams}
    rating_list = {team: [(1600, s_var)] for team in eredivisie_teams}
    rating_list.update(edt_ratings)

    for month in months:

        for team in teams:
            # Pak de variabelen die ge pdatet moeten worden

            mu, sigma2 = rating_list[team][-1]

            # Ga door met het volgende team als het team
            # in deze maand geen wedstrijden heeft gespeeld

            if not np.any(((df["Home"] == team) | (df["Away"] == team)) \
            & (df["Date"] == month)):
                print(f"Geen wedstrijden van {team} in {month}")
                r_list = rating_list[team]
                r_list.append((mu, sigma2 + nu2))
                rating_list[team] = r_list
                continue

            # Kijk naar elke wedstrijd in de maand en reken
            # de delta2 uit, en sla de uitslagen en marges
            # van de wedstrijden op

            d = 0
            results = []
            opponents = []
            margins = []
```

```
        for i in df.index:

            if month != df["Date"][i]:
                continue
            elif team != df["Home"][i] and team != df["Away"][i]:
                continue
            else:
                result = 0.5 + 0.5*np.sign(df["Score"][i])* \
                        (-1)**(1-(team == df["Home"][i]))
                results.append(result)

                if team == df["Home"][i]:
                    # uncomment de line hieronder en de line daaronder,
                    # zet de nu naar 0 en verander k naar een constante
                    # voor het ELO-systeem

                    mu_j, sigma2_j = rating_list[df["Away"][i]][-1]#[0], 0
                    opponents.append((df["Away"][i], mu_j, sigma2_j))
                else:
                    mu_j, sigma2_j = rating_list[df["Home"][i]][-1]#[0], 0
                    opponents.append((df["Home"][i], mu_j, sigma2_j))

                margins.append(abs(df["Score"][i]))

                d += g(sigma2_j)**2*E(mu, mu_j, sigma2_j)* \
                        (1-E(mu, mu_j, sigma2_j))


        delta2 = 1/(q*q*d)


        # Update sigma2 en mu op basis van de resultaten

        sigma2 = 1/(1/sigma2 + 1/delta2)

        k_list = list(k*np.ones(len(results)))

        # Voor het Elo-systeem is de volgende line gecomment

        k_list = list(q*s*sigma2*(1+np.array(margins))**alpha)


        mu += sum([k_list[l]*g(opponents[l][2])* \
                        (results[l]-E(mu, opponents[l][1], opponents[l][2]))
                        for l in range(len(results))])


        r_list = rating_list[team]
        r_list.append((mu, sigma2 + nu2))
        rating_list[team] = r_list

    print(f"{month} compleet.")
```

```
    # Sla de volledige ratinglijst op zodat hij later gebruikt kan worden

    with open("rating_list.json", "w") as outfile:
            json.dump(rating_list, outfile)

    return rating_list


if __name__ == "__main__":
     update_ratings(s_var, k, nu2, alpha, s)
```

A.3. BEP Optimize

```
from BEP_Glicko import update_ratings
import json
import math
import numpy as np
import pandas as pd

# Zie de BEP_Glicko code voor uitleg over de volgende lines

q = math.log(10)/400
g = lambda sigma2: 1/(math.sqrt(1+3*sigma2*q*q/math.pi**2))
E = lambda mu1, mu2, sigma2: 1/(1+10**(-g(sigma2)*(mu1-mu2)/400))


df = pd.read_csv('BEP_Data.csv')

def marginalize(df):
    scores = df.split("  ")
    if scores[0][0] == "(":
        return 0
    return int(scores[0])-int(scores[1])

df["Score"] = df["Score"].apply(marginalize)

for index in range(len(df["Home"])):
    if df["Home"][index] == "Sparta Rotterdam":
        df["Home"][index] = "Sparta R'dam"
    elif df["Home"][index] == "Roda JC Kerkrade":
        df["Home"][index] = "Roda JC"
    elif df["Home"][index] == "Go Ahead Eagles":
        df["Home"][index] = "Go Ahead Eag"
    if df["Away"][index] == "Sparta Rotterdam":
        df["Away"][index] = "Sparta R'dam"
    elif df["Away"][index] == "Roda JC Kerkrade":
        df["Away"][index] = "Roda JC"
    elif df["Away"][index] == "Go Ahead Eagles":
        df["Away"][index] = "Go Ahead Eag"


teams = list(set(df["Home"]))
teams.sort()
```

```
months = list(set([date[:7] for date in df["Date"]]))
months.sort()

first_month = months[0]
last_month = months[-1]

current_year = int(last_month[:4])
years = current_year-int(first_month[:4])

month_numbers = ['01','02','03','04',
                 '05','06','07','08',
                 '09','10','11','12',]


months = [f"{current_year-year}-{month_number}"
          for year in range(years+1)
          for month_number in month_numbers]

months.sort()

months = months[months.index(first_month):]
months = months[:(months.index(last_month)+1)]

numbered_months = {month:i for month, i in zip(months,range(len(months)))}


def discrepancy(s_var, k, nu2, alpha, s):

    # Functie van de discrepantie voor bepaalde waarden voor constanten

    total_discrepancy = 0
    rating_list = update_ratings(s_var, k, nu2, alpha, s)

    for month in months[24:60]:

        for i in df.index:
            if month != df["Date"][i][:7]:
                continue
            else:
                result = 0.5 + 0.5*np.sign(df["Score"][i])

                mu, sigma2 = rating_list[df["Home"][i]][numbered_months[month]]
                mu_j, sigma2_j = rating_list[df["Away"][i]][numbered_months[month]]

            p_ij = 1/(1+10**(-g(sigma2+sigma2_j)*(mu-mu_j)/400))
            total_discrepancy += (-result*np.log(p_ij)-(1-result)*np.log(1-p_ij))


    return total_discrepancy
```

```
# Kies welke variabele geoptimaliseerd moet worden
# Bijvoorbeeld wrt = [0,0,1,0,0] voor nu2

wrt = [0,0,1,0,0]
s_var = 5800
k = 20
nu2 = 140
alpha = 0
s = 1

h = 1e-3


def secant(f, x0, x1, m):

    # Secant methode

    check = f(x1)

    print(check)
    if abs(check) < m:
        return x1
    else:
        print('x0 = ', x0, '\nx1 = ', x1)

        return secant(f, x1, x1-(x1-x0)/(check-f(x0))*check, m)


# Functie voor het uitrekenen van de afgeleide van discrepantie

discrepancies = {}

def ddisc_sec(x):

    # Verander de volgende line op basis van welke
    # constante geoptimaliseerd moet worden

    nu2 = x

    # Als een functiewaarde al is uitgerekend
    # reken hem dan niet opnieuw uit.
    # Als hij nog niet is uitgerekend, reken hem dan uit

    try:
        normal_disc = discrepancies[(s_var, k, nu2, alpha, s)]
    except KeyError:
        normal_disc = discrepancy(s_var, k, nu2, alpha, s)
        discrepancies[(s_var, k, nu2, alpha, s)] = normal_disc
```

```
    try:
        deriv_disc = discrepancies[(s_var+h*wrt[0], k+h*wrt[1], nu2+h*wrt[2], \
        alpha+h*wrt[3], s+h*wrt[4])]
    except KeyError:
        deriv_disc = discrepancy(s_var+h*wrt[0], k+h*wrt[1], nu2+h*wrt[2], \
        alpha+h*wrt[3], s+h*wrt[4])
        discrepancies[(s_var+h*wrt[0], k+h*wrt[1], \
        nu2+h*wrt[2], alpha+h*wrt[3], s+h*wrt[4])] = deriv_disc


    return (deriv_disc-normal_disc)/h

# Calling van de functie

print(secant(ddisc_sec, 15, 20, 1e-6))

# Het volgende kan ge-uncomment worden om grafieken van de
# discrepanties te maken voor bepaalde constanten


# d_list = []
# x_list = range(0,550,50)
# disc_dict = {}

# for nu2 in x_list:
#     d = discrepancy(40000, k, nu2, 0, 1)
#     disc_dict[nu2] = d
#     print(nu2, 'Done')

# with open("disc_dict.json", "w") as outfile:
#     json.dump(disc_dict, outfile)
```

A.4. BEP Results

```
import matplotlib.pyplot as pt
import json
import numpy as np

# Ontvang data uit andere codes

with open("rating_list.json", "r") as infile:
    rating_list = json.load(infile)

with open("disc_dict.json", "r") as infile:
    disc_dict = json.load(infile)

# Zet de ratings na de laatste maand in een gesorteerde lijst

final_ratings = {team: values[-1] for team, values in rating_list.items()}
sorted_list = sorted(final_ratings.items(), key = lambda x:x[1], reverse = True)
print([(elt[0], elt[1][0], elt[1][1]) for elt in sorted_list])

# Maak een gesorteerde lijst van de gemiddelde mu's over de seizoenen

mean_ratings = {team : np.mean([rating[0] for rating in rating_list[team]]) \
                for team in list(rating_list.keys())}
sorted_means = sorted(mean_ratings.items(), key = lambda x:x[1], reverse = True)

# Kies welke teams geplot moeten worden

teams = ['Ajax', 'PSV␣Eindhoven', 'Feyenoord', 'Twente', 'Vitesse', 'VVV-Venlo']
team_ratings = [rating_list[team] for team in teams]

# Plot hoe de ratings van teams over de jaren veranderen

x = np.linspace(0, len(team_ratings[0]), len(team_ratings[0]))
for i in range(len(teams)):
    y = [rate[0] for rate in team_ratings[i]]
    pt.plot(x, y)


pt.xlabel('Maanden')
pt.ylabel('Rating')
pt.legend(teams, bbox_to_anchor=(1.35, 1.02))
pt.show()

# Plot hoe de discrepanties eruit zien voor verschillende waarden voor de constanten

for i in range(len(disc_dict.keys())):
    pt.scatter(int(list(disc_dict.keys())[i]), list(disc_dict.values())[i], c = 'blue')

pt.xlabel('Variabele')
pt.ylabel('Discrepantie')
pt.show()
```

Bibliografie

Hvattum, L. M., & Arntzen, H. (2010). Using ELO ratings for match result prediction in association football. *International Journal Of Forecasting*, 26(3), 460–470.https://doi.org/10.1016/j.ijforecast.2009.10.002

Kovalchik, S. (2020). Extension of the Elo rating system to margin of victory. *International Journal Of Forecasting*, 36(4), 1329–1341. https://doi.org/10.1016/j.ijforecast.2020.01.006

Elo, A. E. (1978) *The Rating of Chess Players Past and Present.* New York: Arco

Davidson, R. R., & Beaver, R. J. (1977). On Extending the Bradley-Terry Model to Incorporate Within-Pair Order Effects. *Biometrics*, 33(4), 693. https://doi.org/10.2307/2529467

Eredivisie scores & fixtures | FBref.com. (z.d.). FBref.com. https://fbref.com/en/comps/23/schedule/Eredivisie-Scores-and-Fixtures

Eerste divisie scores & fixtures | FBref.com. (z.d.). FBref.com. https://fbref.com/en/comps/51/schedule/Eerste-Divisie-Scores-and-Fixtures

Glickman, M. E. (1999). Parameter estimation in large dynamic paired comparison experiments. *Applied Statistics/Journal Of The Royal Statistical Society. Series C, Applied Statistics, 48*(3), 377–394. https://doi.org/10.1111/1467-9876.00159