Jämförelse av maskininlärningsalgoritmer för prediktion av matchresultat inom elektronisk sport

Erik Lumme
Kandidatavhandling i datateknik
Handledare: Åke Syysloiste
Fakulteten för naturvetenskaper och teknik
Åbo Akademi
29 mars 2015

Referat

Maskininlärning omfattar de algoritmer som med hjälp av inlärningsdata försöker anpassa en modell så att resultaten av nya data kan förutses. Den här avhandlingen behandlar deras användningsområden inom elektronisk sport, och deras förmåga att förutse matcher testas på en samling professionella matcher från datorspelet Dota 2. Elektronisk sport har valts eftersom den förändras och utvecklas betydligt snabbare än traditionell sport, vilket försvårar algoritmernas modellanpassning. De algoritmer som behandlas är neuronnät, k-NN och logistisk regression.

Nyckelord: Maskininlärning, neuronnät, k-NN, logistisk regression, elektronisk sport, e-sport.

Innehållsförteckning

1 Inledning **1**
2 Resultatets tillämpningsområden **2**
2.1 Vadslagning 2
2.2 Organiserande av turneringar 2
3 Algoritmerna **3**
3.1 Artificiella neuronnät 3
3.2 K-närmaste grannar 3
3.3 Logistisk regression 4
4 Dota 2 ..... **5**
4.1 Spelet 5
4.2 Testdata 5
4.2.1 Uppdelning 5
4.2.2 Uppbyggnad 6
5 Neuronnät **8**
5.1 Begrepp 8
5.1.1 Nätets typ 9
5.1.2 Inlärningsregeln 9
5.1.3 Överföringsfunktionen 10
5.2 Anpassning 11
5.2.1 Anpassning av mellanliggande lager och neuroner 11
5.2.2 Anpassning av inlärningshastighet och rörelsemängd 12
5.3 Resultat 13
6 k-NN **14**
6.1 Begrepp 14
6.2 Anpassning av inlärningsdata 14
6.3 Resultat 16
7 Logistisk regression **17**
7.1 Begrepp 17
7.1.1 Logit och sannolikhet 17
7.1.2 Anpassning av modellen 17
7.1.3 Koefficienter, standardfel och signifikans 17
7.2 Anpassning av data och preliminära resultat 18
7.3 Resultat 19
8 Sammanfattning och diskussion **20**
8.1 Sammanfattning 20
8.2 Diskussion 20
9 Källförteckning **22**
10 Bilagor **24**

1 Inledning

Maskininlärning är en gren inom artificiell intelligens som omfattar algoritmer som lär sig från data. Dessa algoritmer tar emot indata och producerar ett resultat, till exempel en sannolikhet eller ett beslut. Den här avhandlingen behandlar övervakad maskininlärning, där det önskade resultatet är känt och algoritmen korrigerar sin modell för att enligt bästa möjliga förmåga kunna förutse detta.

Med elektronisk sport, eller e-sport, avses datorspel som i stor utsträckning spelas professionellt. Dessa datorspel utvecklas ofta kontinuerligt för att göra vinstförutsättningarna så jämna som möjligt, och för att göra matcherna så intressanta som möjligt. Detta utgör en utmaning för maskininlärningsalgoritmer, då deras modeller ständigt måste korrigeras på grund av ändrade förhållanden.

Tre algoritmer för övervakad maskininlärning behandlas inom denna avhandling. Dessa är artificiella neuronnät, k-NN och logistisk regression. Samtliga algoritmer får lära sig från samma inlärningsdata bestående av matcher från datorspelet Dota 2, och därefter får de försöka förutse vinnaren i annan en samling matcher.

Syftet med avhandlingen är att se om en hög noggrannhet i förutsägelserna kan uppnås med de använda algoritmerna. Detta kan till exempel vara av intresse för vadslagning eller då turneringar organiseras, så att man kan garantera att de bästa lagen deltar. Den centrala hypotesen är att maskininlärning kan användas för att med hög noggrannhet förutse resultaten av matcher inom elektronisk sport.

Avhandlingen beskriver de grundläggande begreppen inom de algoritmer som används, och är avsedd att fungera som en grund för vidare försök inom området. Vilken exakthet som krävs för att man skall kunna dra praktisk nytta av prediktionerna behandlas endast ytligt. På grund av dess relativa komplexitet läggs större vikt på beskrivning av neuronnät än på de andra algoritmerna.

2 Resultatets tillämpningsområden

Det finns flera tillämpningsområden där man kan dra nytta av att med tillräckligt hög säkerhet kunna förutse resultatet av en match. Ett av dessa är vadslagning, där stora internetbaserade företag har tagit steget från traditionell sport till elektronisk sport [1]. Ett annat område är organiserande av turneringar, där man med hjälp av detta kan se till att de bästa lagen deltar, och strukturera turneringen så att matcherna blir så intressanta som möjligt för att locka en stor publik.

2.1 Vadslagning

Vadslagning på internationella e-sportsmatcher är ett växande fenomen [2]. Allt fler företag erbjuder vadslagning också inom e-sport. Pinnacle Sports, som är en av världens största vadslagningsfirmor [3], hävdar att det är deras snabbast växande sport, och att antalet vad har fördubblats varje år. Under 2014 lades det mer än 350 000 insatser på e-sport på deras sida, och i mitten av december 2014 nådde de över en miljon insatser på e-sport totalt [4, 5].

2.2 Organiserande av turneringar

Elektronisk sport utövas ofta av lag utan en specificerad hemmaort, då majoriteten av matcherna spelas online och spelarna således inte behöver befinna sig på samma plats. Detta leder i hög grad till att lag som presterar bra har flera supportrar än lag som presterar dåligt, då understödandet inte är lokalt i samma grad som i traditionell sport.

Det är i turneringsorganisatörernas intresse att de bästa lagen är kvar i turneringen så länge som möjligt, då detta resulterar i flera åskådare. I så kallade elimineringsmatcher, där det förlorande laget inte längre är med i turneringen, är det därför önskvärt att de bästa lagen ställs emot varandra så sent som möjligt. Om man med hjälp av maskininlärning med hög noggrannhet kan förutse vinnaren av en match leder detta till att organisatörerna lättare kan planera turneringen så att de önskade lagen spelar så mycket som möjligt.

3 Algoritmerna

3.1 Artificiella neuronnät

Artificiella neuronnät (hädanefter neuronnät eller nät) är ett försök att efterlikna neuronnätet i hjärnan. Neuronnätet består av ett lager för indata, ett eller flera gömda mellanliggande lager, samt ett lager för utdata. Varje lager har en eller flera noder. I lagret för indata representerar varje nod någon parameter från de data som matats in. Dessa noder skickar sina värden vidare till noderna i nästa lager, varvid varje värde ges en koefficient som anger värdets signifikans. Beroende på hur de utdata som produceras skiljer sig från de förväntade så förändras koefficienterna så att modellen förbättras [6].

För jämförelserna i denna avhandling används övervakad inlärning, som innebär att neuronnätet har tillgång till inlärningsdata där de utdata som önskas är kända. Det är möjligt för nätet att behandla alla inlärningsdata flera gånger, det vill säga göra flera iterationer, för att uppnå bättre resultat. Huruvida detta ger ett bättre resultat är av speciellt intresse för prediktion av e-sportsresultat, på grund av dess snabba utveckling och förändring. Ett antagande är att detta försämrar nätets prestanda, då det fäster för stor vikt vid föråldrade data.

I klassificerande neuronnät är de slutgiltiga utdata som ges den klass som nätets indata motsvarar. Denna klass kan då det kommer till prediktion av matchresultat till exempel vara ”hemmalaget vinner” eller ”bortalaget vinner”. Utdata kan också förekomma i form av reella värden som representerar sannolikheten för att det ena laget vinner.

3.2 K-närmaste grannar

K-närmaste grannar (k-NN) är en klassificeringsmetod där objekt får koordinater i ett n-dimensionellt koordinatsystem baserat på värdet av deras parametrar (n stycken). Även k-NN är en metod för övervakad maskininlärning, där varje punkt i koordinatsystemet har ett inlärt binärt värde. Nya objekt klassificeras genom att de får det värde som majoriteten av dess *k* närmaste grannar har.

Värdet på *k* påverkar hur stor del av de inlärningsdata som används som klassificeras fel. Om värdet 1 används klassificeras inga inlärningsobjekt fel, och en ökning av *k*

leder till en större andel felklassificeringar [6]. Väljs *k* litet tas således stor hänsyn till små variationer i de indata som används, också slumpmässiga sådana. Ett stort värde på *k* minimerar påverkan av slumpmässiga variationer, men ett för stort värde på *k* utesluter också sådana samlingar av punkter som inte är slumpmässiga [7].

3.3 Logistisk regression

Logistisk regression är en metod för att utifrån kvantitativa eller kvalitativa data bestämma oddsen för att ett objekt hör till en klass. Utfallet är binärt, således hör objektet antingen till den ena klassen eller till den andra. Logistisk regression kräver till skillnad från linjär regression inte normalfördelade indata, och inte heller indata med linjära samband [8].

Olikt linjär regression, där funktionen för sannolikheten är linjär, är funktionen i logistisk regression linjär endast i avseende på naturliga logaritmen av utfallets odds, hädanefter kallat logitfunktionen eller endast logit. Oddset för ett utfall med sannolikheten *p* fås av figur 1.

$$odds = \left(\frac{p}{1-p} \right)$$

Figur 1. Formeln för konvertering från sannolikhet till odds.

Således är formeln för sannolikhet inom logistisk regression den i figur 2, där β_i betecknar koefficienten för parametern x_i.

$$ln\left(\frac{p}{1-p} \right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$$

Figur 2. Formeln för sannolikheten p.

4 Dota 2

4.1 Spelet

Dota 2 är ett onlinebaserat datorspel som spelas i två lag om fem personer. Personerna kontrollerar varsin hjälte, och målet är att förstöra en av motståndarlagets byggnader. En match tar i regel mellan 15 och 60 minuter. De två lagen försvarar varsin sida, den ena kallad *Radiant* och den andra *Dire*. Med lag ett avses i den här avhandlingen det lag som spelar på, och försvarar, sidan *Radiant*, och med lag två det lag som spelar på sidan *Dire*. Sidornas struktur på spelplanen är inte identisk, och således kan det i prediktioner förekomma att ena sidan anses ha ett övertag.

Likt många andra datorspel är Dota 2 dynamiskt i det avseende att det ständigt utvecklas och uppdateras av utvecklarna, och strategier som tidigare inte var gångbara kan med tiden bli väldigt populära. Förutom detta är orsaken till valet av Dota 2 för denna avhandling att det är det datorspel som spelarna har vunnit mest pengar på [9], och att skribenten är bekant med spelet sedan tidigare.

Dota 2 har en stadig och växande professionell krets som drivs av en mängd årliga turneringar organiserade av många olika organisationer. Spelets skapare, Valve Corporation (Valve), håller årligen turneringen The International, som år 2014 hade en sammanlagd prispott på 10,9 miljoner dollar [10].

4.2 Testdata

Som testdata används en samling matcher som spelades mellan 25.9.2014 och 16.3.2015. Alla matchdata har tillhandahållits av Datdota [11], en webbsida som samlar och visar upp allmänt tillgängliga data.

4.2.1 Uppdelning

De utgångsdata som används består av 3492 matcher. Detta är alla professionella matcher som spelats mellan de i avsnitt 4.2 nämnda datum. Med en professionell match avses en match som har spelats i en turnering som har registrerats av Valve. Av dessa 3492 matcher uteslöts 490 olämpliga matcher enligt kriterierna i avsnitt 4.2.2. Ytterligare uteslöts de två sista matcherna för att få jämnt 3000 matcher.

Denna samling delas upp i två delar, inlärningsdata och provdata. De 2800 första matcherna är inlärningsdata, och används således av algoritmerna under inlärningsprocessen. De sista 200 används för att jämföra algoritmernas prestanda efter att inlärningsprocessen har avslutats.

4.2.2 Uppbyggnad

Alla data som används är i form av numeriska värden. Majoriteten av de data som används är sådana som inte är tillgängliga innan matchen spelas. Därför är respektive lags data ersatta av lagets medeltal från de tio senaste matcherna, den aktuella matchen är inte inkluderad. För varje lags tio första matcher används medeltalet av dessa. De matcher som exkluderats enligt avsnitt 4.2.1 är de matcher där åtminstone ett av lagen inte har spelat tio eller flera matcher under den använda tidsperioden.

I tabell 1 ges en kort beskrivning av de datafält, alternativt parametrar eller invärden, som används. De värden som räknas per spelare kombineras till ett medeltal för laget. Vissa värden anges per minut, medan andra anges per match. Det är upp till algoritmerna att hitta det korrekta sambandet mellan dessa.

Tabell 1. Beskrivning av de datafält som används av algoritmerna

Engelskt namn	**Beskrivning**
Game number	Matchens ordningsnummer, till exempel i en bäst av tre-serie
Duration	Matchens varaktighet i minuter
Deaths	Antal gånger spelare på de olika lagen har dött
Gold per minute	Hur mycket spelpengar spelarna har tjänat per minut
Experience per minute	Hur många erfarenhetspoäng spelarna har tjänat per minut
Level	Vilken nivå spelarnas hjältar har nått
Creep score	Hur många monster en hjälte har dödat
Denies	Hur många monster en hjälte har hindrat andra laget från att döda
Damage	Hur mycket skada en hjälte har angjort andra laget
Tower damage	Hur mycket skada en hjälte har angjort andra lagets byggnader
Healing	Hur mycket en hjälte har helat det egna laget
Radiant win	Huruvida lag ett vann eller inte

För enkelhetens skull anges inte lagens värden var för sig då data matas in i algoritmerna, utan enbart differensen mellan lag ett och lag två. Ett negativt värde

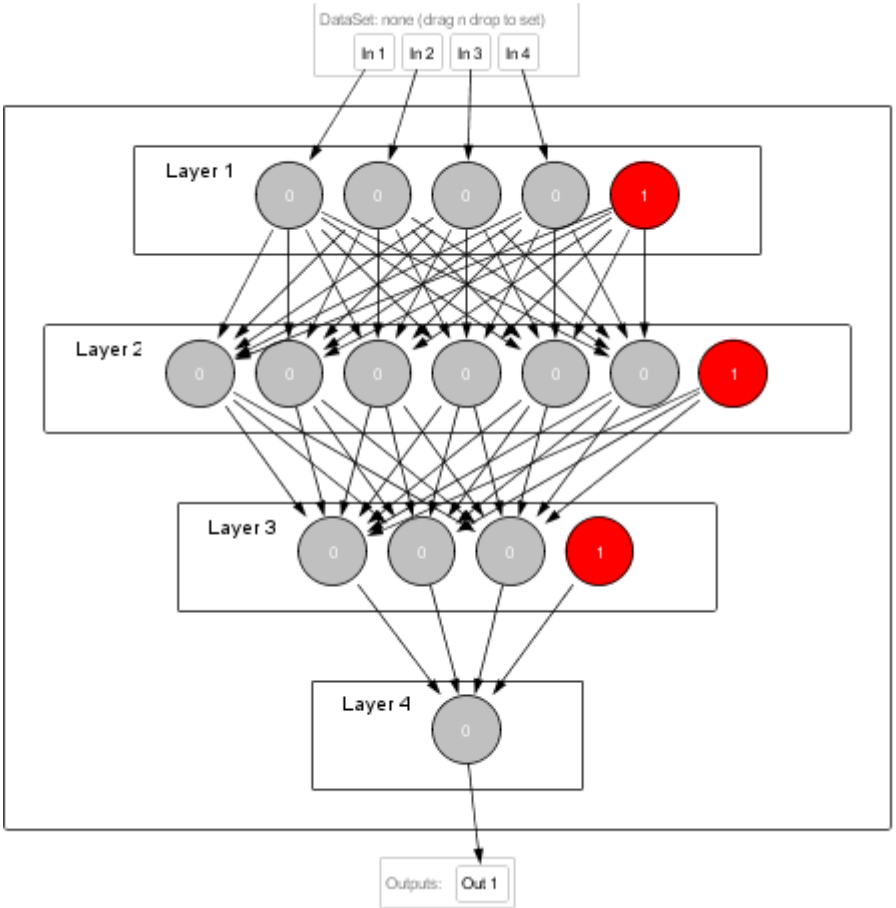
innebär således att värdet för lag två var högre, och ett positivt värde innebär att värdet för lag ett var högre.

5 Neuronnät

För neuronnäten används programmet NeurophStudio [12] för Windows. För att få bästa möjliga resultat anpassades näten och de indata som användes beroende på hur de olika näten presterade, hur denna anpassning gjordes tas upp i 5.2.

5.1 Begrepp

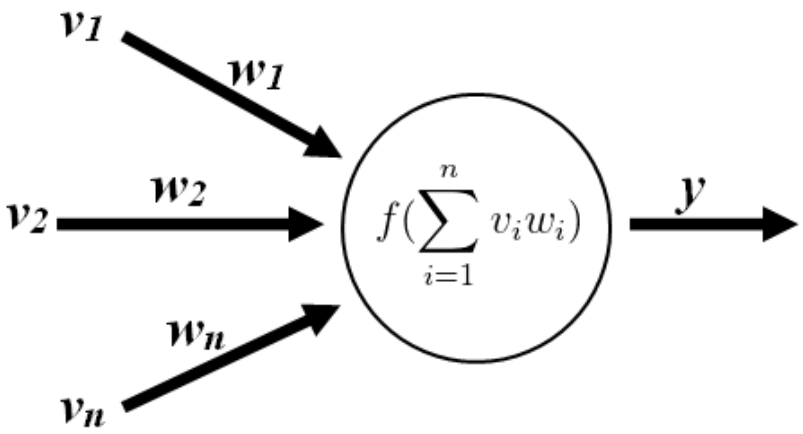
Avsikten med denna del är att förklara de centrala termer som används inom neuronnät i NeurophStudio, och vilka inställningar som använts. I figur 3 visas ett exempel på ett nät i NeurophStudio. Nätet har fyra lager, varav två är mellanliggande. Nätet tar in fyra parametrar, och har således fyra neuroner i lagret för indata. De mellanliggande lagren har sex respektive tre neuroner, och lagret för utdata har en. De röda cirklarna representerar ytterligare så kallade vinklande neuroner, som skickar vidare det konstanta värdet 1 för att förbättra prestandan. Dessa används i samtliga nät inom denna avhandling.



Figur 3. Ett exempel på ett nät i NeurophStudio.

5.1.1 Nätets typ

Det finns flera olika typer av neuronnät, som fungerar olika bra på olika sorters data. I den här avhandlingen används nät av typen *multilayer perceptron* (MLP), det vill säga ett nät bestående av perceptroner i flera lager. En perceptron är en algoritm som associerar en koefficient med alla värden i dess indata. Varje värde multipliceras med dess koefficient, och resultaten summeras sedan ihop. Summan matas in i en överföringsfunktion (se 5.1.3) och resultatet skickas vidare till nästa lager. Beroende på hur utvärdet skiljer sig från det väntade modifieras sedan koefficienterna för att i framtiden ge ett bättre resultat [13]. I figur 4 visas ett neuron med perceptronalgoritmen, där invärden betecknade med *v* och koefficienter betecknade med *w* multipliceras, summeras och slutligen matas in i överföringsfunktionen *f*. Resultatet *y* skickas vidare till nästa lager.



Figur 4. Ett neuron med perceptronalgoritmen.

5.1.2 Inlärningsregeln

Inlärningsregeln bestämmer hur ett neuronnäts prestanda förbättras under inlärningen. I avhandlingens neuronnät används *backward propagation of errors with momentum*, det vill säga baklänges felspridning med rörelsemängd.

Baklänges felspridning syftar på att felet, det vill säga hur mycket resultatet skiljer sig från det förväntade, beräknas från det sista lagret först. Med hjälp av detta lager och föregående beräknas felens gradient, som anger i vilken riktning koefficienterna bör ändras för att minska på felet så mycket som möjligt. Detta görs i tur och ordning

för alla lager, så att de utdata som neuronerna i varje lager skickar vidare ger ett bättre resultat i nästa lager [6].

Rörelsemängd innebär att det till koefficienternas förändring adderas en del av föregående iterations förändring. Således är förändringen större om den är i samma riktning som tidigare. Till inlärningsregeln hör också en inlärningshastighet, som anger i hur stora steg koefficienterna skall förändras.

5.1.3 Överföringsfunktionen

I hjärnan sänds elektriska signaler mellan neuroner, och med vilken frekvens dessa sänds beror på hur aktiva neuronerna är. Hur aktiv ett neuron är modelleras inom neuronnät med överföringsfunktionen, även kallad aktiveringsfunktionen [13]. Denna funktions graf är vanligtvis s-formad med utvärden mellan 0 och 1.

I denna avhandlings neuronnät används överföringsfunktionen i figur 5. *Input* betecknar summan av neuronens invärden multiplicerade med koefficienterna, och *slope* betecknar hur brant sluttningen är på den gradient som beräknats enligt 5.1.2.

$$y = \frac{1}{1 + e^{(-slope \cdot input)}}$$

Figur 5. Neuronnätets överföringsfunktion.

5.2 Anpassning

För att få bästa möjliga resultat bör nätets parametrar anpassas genom tester. De parametrar som ändrats för det nät som här behandlas är inlärningshastigheten, antal mellanliggande lager och neuroner, rörelsemängden, antal invärden och antal iterationer.

En liten prestandaförbättring noterades då invärdena normaliserades till värden mellan -1 och 1, och således är det dessa värden som här har använts. Däremot blev felet större då antalet invärden minskades, och alla tester här använder således alla 11 invärden.

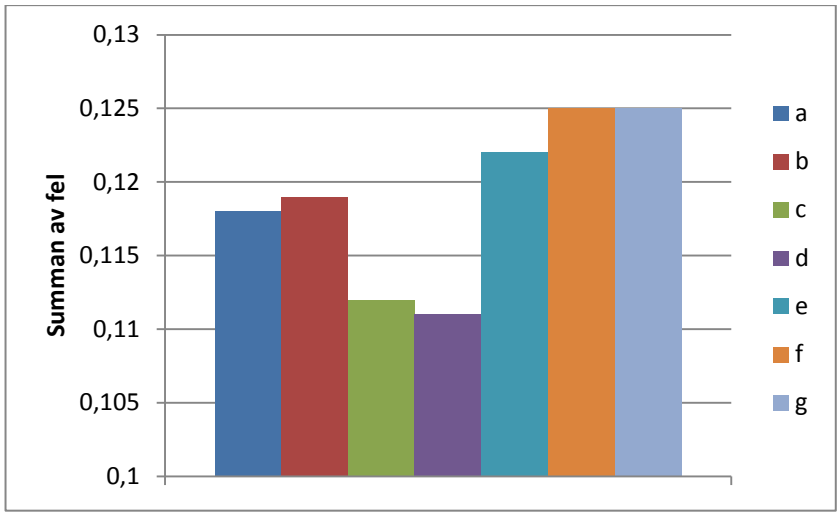
5.2.1 Anpassning av mellanliggande lager och neuroner

Några tester gjordes med olika antal mellanliggande lager och olika många neuroner i dessa. Detta för att minimera summan av fel i nätverket. Figur 6 visar felets storlek i de sju olika konfigurationer som visas i tabell 2. Samtliga värden är det minsta fel som uppnåtts inom 3000 iterationer. Konfigurationerna av mellanliggande lager är följande:

Tabell 2. De mellanliggande lagrens uppsättning

a	ett lager, 3 neuroner
b	ett lager, 6 neuroner
c	ett lager, 16 neuroner
d	ett lager, 30 neuroner
e	två lager, 16 och 8 neuroner
f	tre lager, 10, 10 och 10 neuroner
g	tre lager, 30, 16 och 8 neuroner

Av detta kan man dra slutsatsen att den bästa prestandan ges av ett mellanliggande lager med många neuroner. Dessutom är prestandaförbättringen försumbar då antalet neuroner är större än 30.

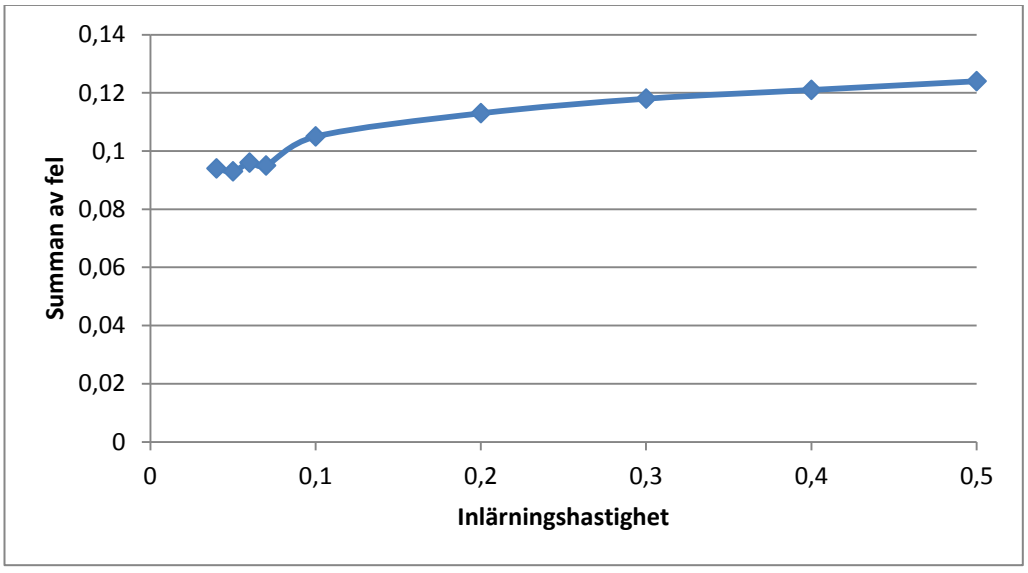


Figur 6. Summan av nätets fel då de mellanliggande lagren har olika konfigurationer enligt tabell 2. Mindre är bättre. Märk att skalan börjar vid 0,1.

5.2.2 Anpassning av inlärningshastighet och rörelsemängd

Genom att öka på inlärningshastigheten anpassar sig nätet snabbare, men risken är också större att för stora förändringar görs, och att nätet således hoppar över bra värden. Ett större värde på rörelsemängden gör att nätet anpassar sig snabbare då koefficienterna efter varje iteration ändras åt samma håll, men precis som med inlärningshastigheten ökas risken av att bra värden missas.

Figur 7 visar det minsta värdet på summan av felen som uppnåtts under 3000 iterationer med olika värden på inlärningshastigheten på x-axeln.



Figur 7. Summan av nätets fel i förhållande till inlärningshastighet.

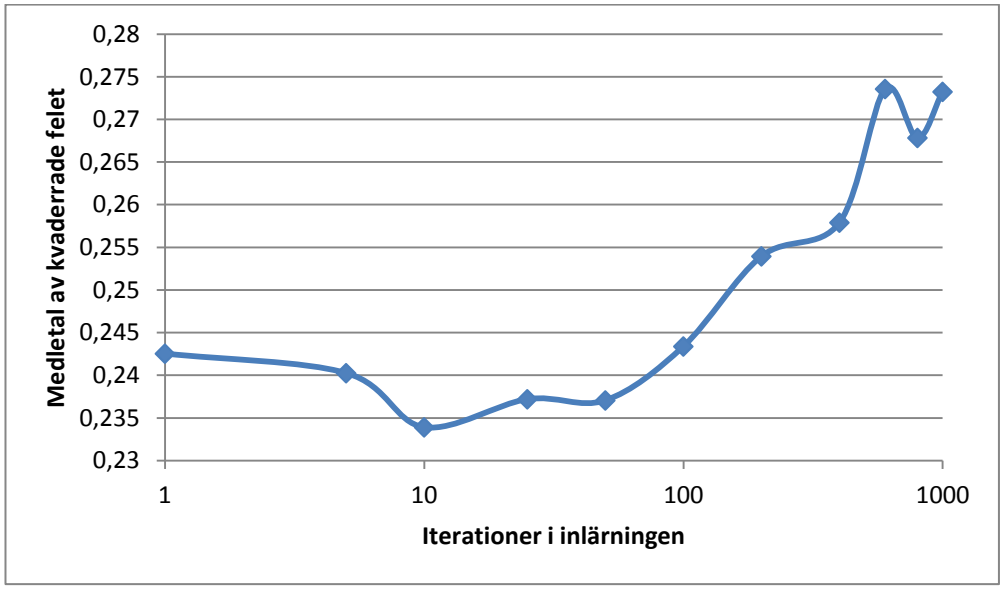
Det minsta felet påträffas då inlärningshastigheten var 0,05.

Värdet på rörelsemängden var 0,7 i samtliga tester. Vid ytterligare tester visades att en ändring av värdet till 0,8 ökade felet med 0,006, och om värdet ändrades till 0,6 ökade felet med 0,012. Således kan 0,7 anses vara ett optimalt värde.

5.3 Resultat

I fältet för utdata innebär en etta att lag ett vann, medan en nolla innebär motsatsen. Vid körning av testdata märktes att det antal iterationer som gett det lägsta felet på nätets inlärningsdata inte gav det bästa resultatet på dess testdata. Detta stämmer överens med hypotesen i 3.1, som förutsade att för många iterationer leder till att nätet anpassar sig för mycket till gamla data, och att spelets snabba utveckling leder till att denna anpassning inte mera är aktuell på nya data.

Grafen i figur 8 visar medeltalet av de kvadrerade felen på de testade matcherna då inlärningsprocessen har gjorts med olika många iterationer. Minsta felet ficks då endast tio iterationer gjordes på nätets inlärningsdata.



Figur 8. Förhållande mellan fel på testdata och antal iterationer i inlärningen.

10 iterationer på nätets inlärningsdata gav följande resultat på testmatcherna:

Medeltal av kvadrerade felet: 0,2339094
Antal korrekta matcher: 125 av 200

6 k-NN

K-NN är lätt att implementera, och för den här avhandlingen används en enkel implementation i Java som bifogats som bilaga 1.

6.1 Begrepp

Ett k-NN-systems prestanda beror på hur avståndet till en punkts närmaste grannar beräknas, och vilket värde på *k* som väljs.

Här används för avståndet euklidiskt avstånd, som för två punkter är längden av en rak linje mellan dem. För en vektor i *n* dimensioner ges avståndet av följande formel, där x_i betecknar en punkt i vektor x och y_i betecknar motsvarande punkt i vektor y.

$$distance = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2}$$

Figur 9. Formeln för euklidiskt avstånd.

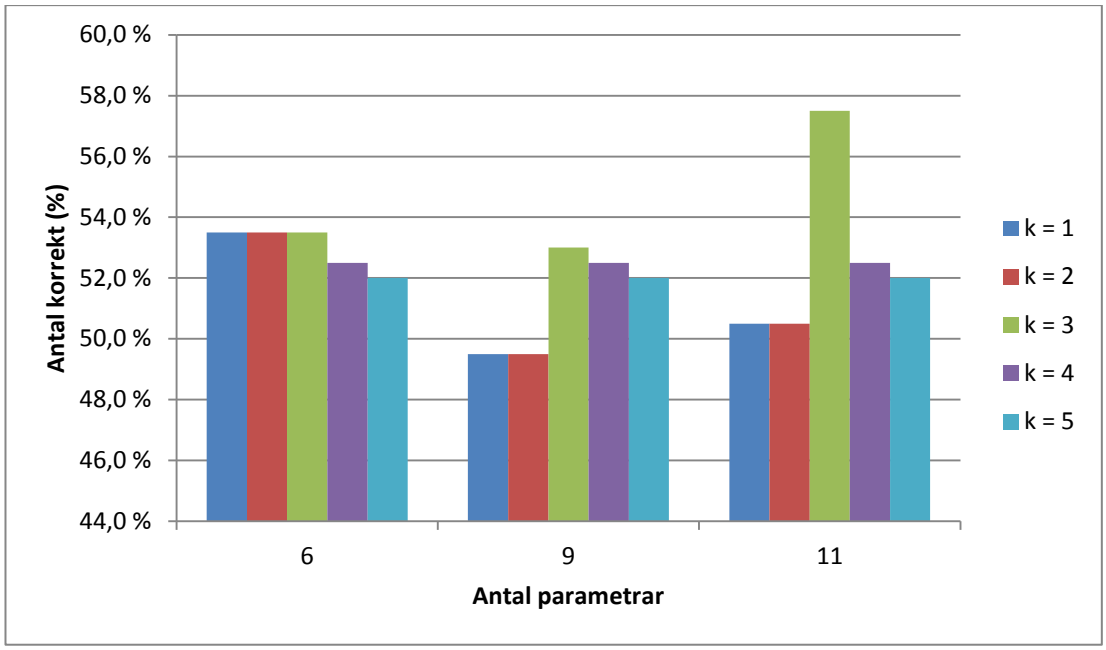
Värdet på *k*, det vill säga hur många närmaste grannar som skall tas i beaktande, kan väljas så att det ger bästa möjliga resultat på de inlärningsdata som finns tillgängliga. I avsnitt 6.3 ges resultat för olika värden på *k*.

K-NN kan användas som klassificerare eller för regression. Ett klassificerande k-NN-system ger som utvärde en klass, här ett binärt värde som motsvarar om lag ett vann eller inte. Vid regression är värdet det numeriska medeltalet av objektets n närmaste grannar [6].

6.2 Anpassning av inlärningsdata

Systemets prestanda testades med olika antal parametrar för dess indata. Förutom de maximala 11 testades även 9 och 6. I de första av fallen hade *Duration* och *Game Number* tagits bort, och i det andra även *Damage*, *Tower Damage* och *Healing*, se tabell 1 för förklaringar.

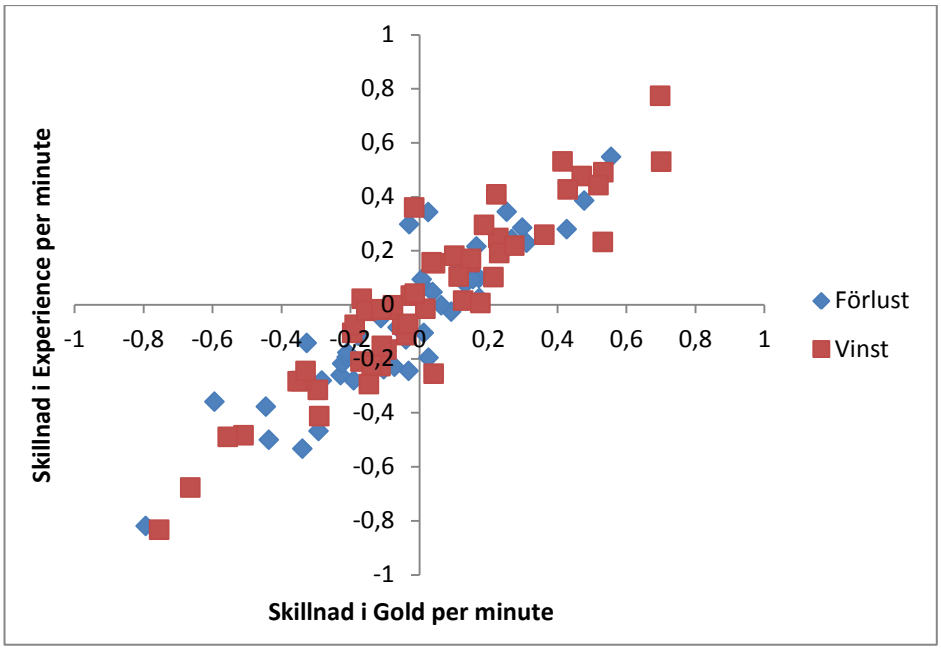
I figur 10 visas procentuellt hur många matcher som klassificerats korrekt då *k* har värden mellan 1 och 5, och med 11, 9 eller 6 parametrar i systemets inlärnings- och testdata.



Figur 10. Antal rätt förutsedda matcher med olika värden på k, och med olika antal parametrar.

Av detta kan ses att bäst prestanda sannolikt fås med alla 11 parametrar, och då *k* har värdet 3. Värdet på *k* bör väljas udda då det handlar om klassificering, för att undvika situationer där det finns lika många närmaste grannar av var klass.

I figur 11 har matcher plottas då endast två parametrar har tagits i beaktande: *Gold per minute* och *Experience per minute*. Från detta ser vi att samtliga punkter ligger i stort sett samma linje, detta eftersom bägge parametrarnas värden i Dota 2 kommer från samma källor. Från grafen är det svårt att urskilja ett mönster som med hög noggrannhet kunde klassificera matcherna korrekt, även om vinsterna till något högre grand befinner sig uppe till höger som förväntat.



Figur 11. Vunna och förlorade matcher ur lag etts synpunkt enligt skillnad i Gold per minute och Experience per minute.

6.3 Resultat

För den slutgiltiga körningen har inlärningsdata med 11 parametrar använts, och *k* har haft värdet 3, i enlighet med resultatet från figur 10.

Detta gav följande resultat:

Antal korrekta matcher: 115 av 200.

7 Logistisk regression

För logistisk regression används programmet gretl [14] för Windows.

7.1 Begrepp

I det här avsnittet förklaras de viktigaste begreppen och funktionerna inom beräkningar med logistisk regression.

7.1.1 Logit och sannolikhet

Enligt avsnitt 3.3 är logit den naturliga logaritmen av det positiva utfallets odds, och definieras i figur 12.

$$logit(p) = ln\left(\frac{p}{1-p}\right)$$

Figur 12. Logitfunktionen.

Sannolikheten *p* är således definierad enligt formeln i figur 13, där β står för anpassningskoefficienter och *x* är parametrarna som bestämmer utfallet [8].

$$p = \frac{exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n)}{exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n) + 1}$$

Figur 13. Definitionen av sannolikheten p.

7.1.2 Anpassning av modellen

För att anpassa modellen så att rätt koefficienter hittas används Maximum-likelihood-metoden (ML-metoden). Ett antagande görs att de stickprov som används, här modellens inlärningsdata, är fördelade enligt samma fördelningsfunktion, som förutom på stickprovet beror på den okända parametern θ. Enligt ML-metoden väljs denna parameter så att sannolikheten för att ha fått de stickprov som används maximeras [15].

7.1.3 Koefficienter, standardfel och signifikans

Då modellen har anpassats ges för invärdesparametrarna koefficienter, som betecknar den naturliga logaritmen av oddsets förändring då parameterns värde ökar med ett. Således innebär en positiv koefficient att en ökning av motsvarande

parameter resulterar i en större sannolikhet för att modellen ger ett positivt utfall. Det anges också ett standardfel, som är en uppskattning av standardavvikelsen för parametern i fråga.

Signifikans anger om parametern är statistiskt signifikant för modellen. Ett så kallat p-värde ges, som anger sannolikheten för att nollhypotesen förkastas även om den är sann. Nollhypotesen är hypotesen att det inte finns ett samband mellan två observationer, och således anger p-värdet sannolikheten att ett samband mellan parametern och utfallet observeras fast ett sådant inte finns. I dessa tester används en signifikansnivå på 5 %, alltså måste p-värdet vara mindre än detta för att sambandet skall anses vara signifikant.

7.2 Anpassning av data och preliminära resultat

Inlärningsdata kan anpassas så att parametrar som inte är signifikanta väljs bort. Med alla parametrar ger modellens inlärningsdata resultatet i tabell 3, där *Experience per minute* har förkortats *Xp per minute*.

Tabell 3. Parametrarnas koefficienter, standardfel och p-värden då modellen anpassats

	Koefficient	**Standardfel**	**p-värde**
Duration	-0,852603	0,392722	0,0299
Game number	-0,526683	0,276195	0,0565
Deaths	-1,03907	0,229529	0,00000598
Gold per minute	0,68626	0,616842	0,2659
Xp per minute	0,582402	0,60766	0,3378
Level	-0,368826	0,52843	0,4852
Creep score	0,448078	0,419249	0,2852
Denies	0,328444	0,186226	0,0778
Damage	0,0831888	0,284806	0,7702
Tower damage	0,32584	0,283664	0,2507
Healing	0,130277	0,162695	0,4233

Av denna figur kan avläsas att få värden är signifikanta enligt en signifikansnivå på 5

%. Detta innebär att p-värdet bör vara lägre än 0,05, vilket är fallet endast för *Duration* och *Deaths*. *Deaths* är den parameter som med störst sannolikhet är signifikant, då den till och med är signifikant med en signifikansnivå på 0,001 %. Parametern *Damage* är sannolikt osignifikant.

Vidare kan ses att om lag ett i medeltal har spelat längre spel än lag två ger detta lag två en fördel. Med en signifikansnivå på 6 % skulle det även vara signifikant vilket ordningsnummer matchen har, då det spelas en serie av flera matcher. Matcher som spelas senare i serien ger lag två en fördel. Som väntat är de flesta parametrar positiva, vilket innebär att lag ett har en fördel då deras värden är högre, till exempel då de tjänat mera spelpengar per minut eller då de dödat flera monster per match. Av de 2800 matcher som har analyserats skulle den givna modellen förutse 61 % av matcherna korrekt.

I tabell 4 visas resultat som motsvarar de i tabell 3, då parametrar som inte är signifikanta med en signifikansnivå på 10 % har tagits bort. Parametern *Death*:s p-värde är skrivet i en notation där värdet är det samma som $7{,}03 \cdot 10^{-21}$. Modellen som genererats förutser däremot endast 59 % av matcherna korrekt, och kan därför anses vara sämre än den föregående.

Tabell 4. Parametrarnas koefficienter, standardfel och p-värden då endast signifikanta parametrar används enligt signifikansnivån 10 %

	Koefficient	**Standardfel**	**p-värde**
Duration	−0,00631019	0,0031473	0,045
Game number	−0,105536	0,0544644	0,0527
Deaths	−0,0588864	0,0062824	7,03E-21
Denies	0,0707873	0,020022	0,0004

7.3 Resultat

För det slutgiltiga testet har alla 11 parametrar använts, och testet har körts på de 200 matcher som ingår i de testdata som brukats i övriga resultat. Då dessa data inte ingår i de data modellen anpassats för, är prestandan något sämre.

Antal korrekta matcher: 115 av 200.

8 Sammanfattning och diskussion

Detta avsnitt behandlar jämförelser av de olika algoritmerna, både då det gäller funktion och då det gäller resultat. Dessutom ges reflektioner kring hur resultatet kunde förbättras, och hur bra denna noggrannhet kan anses vara.

8.1 Sammanfattning

Av resultaten i tabell 5 kan ses att samtliga algoritmer med hög sannolikhet presterar bättre än slumpen. För samtliga har p-värden beräknats, vilka visar att samtliga algoritmers prestanda är signifikant på signifikansnivån 5 %. Vidare är neuronnätets prestanda signifikant på signifikansnivån 0,05 %.

Tabell 5. Algoritmernas prestanda och p-värde

Algoritm	**Antal matcher**	**Antal korrekta**	**Procent**	**p-värde**
Neuronnät	200	125	62,5 %	0,000249713
k-NN	200	115	57,5 %	0,0200186
Logistisk regression	200	115	57,5 %	0,0200186

För jämförelse kontrollerades hur många matcher åskådarna har förutsett rätt på GosuBet [16], en sida där spelare tillåts satsa virtuella objekt av visst monetärt värde på matcher. De 200 senaste matcherna kontrollerades, och det visade sig att i 141 av matcherna hade de flesta tippat rätt, det vill säga i 70,5 % av fallen.

Av algoritmerna är neuronnät i sitt grundutförande den mest omfattande, medan k-NN och logistisk regression är mindre omfattande. För samtliga finns fri programvara tillgänglig.

8.2 Diskussion

Av 8.1 ser vi att den noggrannhet algoritmerna uppnådde inte är lika bra som den hos personer som satsar virtuella objekt på matcher. I nuvarande form är det osannolikt att algoritmernas förutsägningsförmåga är av stort värde.

Det bevisades dock att samtliga presterar bättre än slumpen, och då det gäller neuronnät kunde rätt vinnare förutses i 62,5 % av fallen. Detta ger hopp om att vidare arbete med algoritmernas konfigurationer och de data som används kan ge

tillräckliga resultat. I de lagbeskrivande data som användes togs inte i beaktande vilka motståndare de hade spelat mot, vilket har en stor inverkan på värdena. Dessutom finns många andra data tillgängliga om en bra metod används för att sammanställa dem till en form som är lämplig för analys. Hypotesen att maskininlärning kan användas för att med hög noggrannhet förutse matcher inom elektronisk sport har således inte motbevisats.

9 Källförteckning

[1] Sheldon, D. 02/24 2015-last update, *Are eSports the Next Big Betting Thing?* [Homepage of Casino.org], [Online]. Available: http://www.casino.org/blog/are-esports-the-next-big-betting-thing/ [2015, 03/31].

[2] Martin, A. 01/29 2014-last update, *Fair play and fixing: The growing pains of eSports* [Homepage of RedBull], [Online]. Available: http://www.redbull.com/en/games/stories/1331630452105/fair-play-and-fixing-the-growing-pains-of-esports [2015, 02/12].

[3] *PinnacleSports*. Available: http://www.pinnaclesports.com/ [2015, 02/12].

[4] Wagstaff, K. 2014, 11/07-last update, *Game On: Gambles Want part of Rising E-Sports* [Homepage of NBC News], [Online]. Available: http://www.nbcnews.com/tech/video-games/game-gamblers-want-part-rising-e-sports-n242086 [2015, 02/12].

[5] *Pinnacle Sports road to one million eSports bets*2014, [Homepage of eSport betting], [Online]. Available: http://esportbetting.eu/news/pinnaclesports-road-to-one-million-eSport-bets [2015, 02/24].

[6] Hastie, T., Tibshirani, R. & Friedman, J. 2008, *The Elements of Statistical Learning*
Data Mining, Inference, and Prediction, Second edn, Springer, Stanford, California.

[7] Maier, M., Hein, M. & von Luxburg, U. 2009, *Optimal construction of k-nearest neighbor graphs for identifying noisy clusters*, Institute for Biological Cybernetics;Saarland University.

[8] Bjerling, J. & Ohlsson, J. 2010, *En introduktion till logistisk regressionsanalys*, Göteborg.

[9] *Top 50 Games Awarding Prize Money* [Homepage of E-Sports Earnings], [Online]. Available: http://www.esportsearnings.com/games [2015, 03/31].

[10] *The International Dota 2 Championships* [Homepage of Valve Corporation], [Online]. Available: http://www.dota2.com/international/overview/ [2015, 02/12].

[11] *datdota* [Homepage of datdota], [Online]. Available: http://www.datdota.com [2015, 03/21].

[12] *Java Neural Network Framework.* Available: http://neuroph.sourceforge.net/ [2015, 03/28].

[13] Daumé III, H. 2012, *A Course in Machine Learning*, version 0.8.

[14] Cottrell, A. & Lucchetti, R. 2014, 09/20-last update, *Gnu Regression, Econometrics and Time-series Library*. Available: http://gretl.sourceforge.net/ [2015, 03/31].

[15] Gustafsson, T. 2010, "Maximum-likelihood-metoden" in *Sannolikhetslära och statistik*, pp. 111.

[16] *GosuBet* [Homepage of GosuGamers], [Online]. Available: http://www.gosugamers.net/dota2/gosubet [2015, 03/31].

10 Bilagor

Bilaga 1 Kod för Java-program för k-NN

Bilaga 1: Kod för det Java-program som beräknade k-NN-resultaten.

```
import org.apache.commons.math3.ml.distance.EuclideanDistance;
import org.apache.commons.lang3.ArrayUtils;

public class Knn {
    static TreeMap<Double[], Double> trainingData = new TreeMap<Double[], Double>(new
DoubleComparator());
    static TreeMap<Double[], Double> testData = new TreeMap<Double[], Double>(new
DoubleComparator());

    public static void main (String[] args) {
        // Data, tab separated, one row for each case with the result last
        readTrainingData("knn_optim_2800.txt");
        readTestData("knn_optim_200.txt");
    }

    public static void readTrainingData(String path) {
        Scanner sc = null;
        Double[] data;
        try {
            sc = new Scanner(new FileReader(path));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        while(sc.hasNextLine()) {
            String line = sc.nextLine();
            String[] pieces = line.split("\t");
            data = new Double[pieces.length-1];
            Double result;

            for(int i = 0; i < pieces.length-1; i++) {
                data[i] = Double.parseDouble(pieces[i]);
            }
            result = Double.parseDouble(pieces[pieces.length-1]);

            trainingData.put(data, result);
        }
        sc.close();
    }

    public static void readTestData(String path) {
        Scanner sc = null;
        Double[] data;
        int k = 0;
        try {
            sc = new Scanner(new FileReader(path));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        EuclideanDistance ed = new EuclideanDistance();
        int right = 0;
        int wrong = 0;

        while(sc.hasNextLine()) {
            String line = sc.nextLine();
            String[] pieces = line.split("\t");
            data = new Double[pieces.length-1];
            Double result;
```

Bilaga 1: Kod för det Java-program som beräknade k-NN-resultaten.

```
            for(int i = 0; i < pieces.length-1; i++) {
                data[i] = Double.parseDouble(pieces[i]);
            }

            result = Double.parseDouble(pieces[pieces.length-1]);

            testData.put(data, result);

            double guess = -1;
            k = 1; // Define k

            double[] min = new double[k];
            Double[][] values = new Double[k][];
            for(int i = 0; i < k; i++) {
                min[i] = 999;
                values[i] = new Double[]{0.0};
            }

            for(Double[] comp: trainingData.keySet()) {
                double dist = ed.compute(ArrayUtils.toPrimitive(comp),
                                         ArrayUtils.toPrimitive(data));

                int x;
                for(x = 0; x < k; x++) {
                    // Found a lower
                    if(min[x] > dist) {
                        int y = x;
                        // Move everything up
                        for(x = k-1; x > y; x--) {
                            min[x] = min[x-1];
                            values[x] = values[x-1];
                        }
                        // Save new
                        min[y] = dist;
                        values[y] = comp;
                        break;
                    }
                }
            }

            // Get the guess
            double win = 0;
            double loss = 0;
            for(int i = 0; i < k; i++) {
                if(trainingData.get(values[i]) == 1)
                    win = win + 1 + (1/Math.pow(2, (i+1)));
                else loss = loss + 1 + (1/Math.pow(2, (i+1)));
            }
            guess = (win > loss) ? 1 : 0;

            if(guess == result) right++;
            else wrong++;
        }
        System.out.println("Out of 200 games, we had " + right + " right and "
                           + wrong + " wrong with " + k + "-nearest-neighbor");
        sc.close();
    }

    private static class DoubleComparator implements Comparator<Double[]> {
```

Bilaga 1: Kod för det Java-program som beräknade k-NN-resultaten.

```
        @Override
        public int compare(Double[] o1, Double[] o2) {
            if(o1.length == 0) {
                if(o2.length == 0) return 0;
                return -1;
            }
            if(o2.length == 0) return 1;

            if(o1[0] < o2[0]) return -1;
            if(o1[0] > o2[0]) return 1;
            return 0;
        }
    }
}
```