Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Urša Krevs

Računalniška analiza človekovih napak pri igranju šaha

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00040/2012
Datum: 13.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **URŠA KREVS**

Naslov: **RAČUNALNIŠKA ANALIZA ČLOVEKOVIH NAPAK PRI IGRANJU ŠAHA**

COMPUTER ANALYSIS OF HUMAN ERRORS IN CHESS PLAYING

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Glavno vprašanje pri tej diplomski nalogi je, ali je možno iz množice danih šahovskih partij posameznega igralca odkriti, kdaj najbolj pogosto dela napake oz. kaj so njegove slabosti ter splošne lastnosti pri igranju? Na voljo imamo zbirke pozicije iz partij posameznih igralcev in velikost morebitnih napak pri odigrani potezi za vsako pozicijo. Napaka se izračuna kot razlika med računalniško oceno najboljše poteze in oceno dejansko odigrane poteze. Naloga je poskusiti s tehnikami strojnega učenja za vsakega igralca odkriti razloge za izbiro napačne poteze oz. bolj splošne korelacije med napakami in lastnostmi pozicij ali potez. Pri analizi naj bodo izpeljani poskusi z raznimi metodami učenja v paketu Orange, rezultati poskusov pa naj bodo interpretirani v luči naslednjih vprašanj: Ali je možno odkriti značilne lastnosti igralca? Kje so prednosti in slabosti posameznega igralca? Ali je možno dobljene rezultate uporabiti za računanje podobnosti med igralci?

Mentor:

akad. prof. dr. Ivan Bratko

Fakulteta za računalništvo in informatiko
UNIVERZA V LJUBLJANI
2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Urša Krevs, z vpisno številko **63090074**, sem avtorica diplomskega dela z naslovom:

Računalniška analiza človekovih napak pri igranju šaha

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom akad. prof. dr. Ivana Bratka,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 19. septembra 2012 Podpis avtorja:

Zahvaljujem se mentorju Ivanu Bratku, Mateju Guidu, Martinu Možini in vsem ostalim v laboratoriju za umetno inteligenco, ki so mi pomagali pri tej diplomi. Zahvaljujem se tudi svoji družini in Igorju, ker mi ves čas stojijo ob strani.

Kazalo

Povzetek

Namen tega diplomskega dela je bila računalniška analiza podatkov o šahovskih partijah tako, da preko ocenjevanja posameznih odigranih potez poskušamo ugotoviti in na človeku razumljiv način opisati, kdaj igralci delajo napake in kdaj ne.

Analiza je izvedena tako, da so na podlagi podatkov o partijah in ocenah potez zgenerirani diskretni atributi, ki opisujejo lastnosti šahovskih pozicij in potez, ki so predmet računalniške analize. Na teh atributih smo izvedli χ^2 test in generirali pravila z metodo CN2 z EVC (ang. *Extreme Value Correction*). S χ^2 testom smo poskušali ugotoviti kateri atributi so najbolj informativni. Namen učenja pravil pa je bil združiti več atributov na smiseln način tako, da bi se iz tega dalo ugotoviti značilnosti posameznih igralcev ali skupin igralcev.

Analizo smo izvajali na podatkih o odigranih partijah dveh šahovskih velemojstrov - nekdanjih svetovnih prvakov, da bi ugotovili značilnosti posameznih igralcev. Analizirali smo tudi podatke dveh skupin igralcev z različnimi ELO ratingi, pri čemer smo ugotavljali lastnosti skupin igralcev, ki se razlikujejo po šahovski moči.

Rezultati kažejo, da je s pomočjo analize podatkov s χ^2 testom in z učenjem pravil možno ugotoviti, da nekatere lastnosti posamezne pozicije pri določenih igralcih nakazujejo, da bo odigral slabšo ali boljšo potezo. Učenje pravil je potrdilo rezultate χ^2 testa, saj so bile pri obeh metodah ugotovitve glede lastnosti igralcev podobne. Rezultati, ki smo jih dobili, so logični in jih je mogoče smiselno interpretirati.

Ena bolj zanimivih ugotovitev, do katere smo prišli z analizo je bila, da igralci delajo manj slabih potez, ko imajo kompenzacijo za pomanjkanje materiala.

Abstract

In this thesis we carried out a computer analysis of chess games played by a number of human players. We evaluated each move in these games and tried to find out when players are likely to make mistakes and when not. We tried to induce understandable error prediction rules with a rule learning program.

In the analysis we generated discrete attributes for each position and its evaluation. These attributes describe chess positions and moves analysed. The analysis included a χ^2 test of the dependency between the attributes and human error and rule induction with the CN2 algorithm with EVC (Extreme Value Correction). With the χ^2 test we tried to determine which individual attributes were associated with errors. By generating rules we tried to find combinations of several attributes associated with errors. Using these two methods we tried to find the main characteristics of a player or groups of players.

We analysed game data from two chess grandmasters - once world chess champions to try and determine the characteristics of individual players. We also analysed data from two groups of players which differed in the players ELO ratings. Here we tried to determine the characteristics of groups of players of different strengths.

The results show that an analysis with the χ^2 test and rule generation can show that some characteristics of a chess position can indicate that certain players are more likely to make a good or a bad move in that position. Rule generation confirmed the results of the χ^2 test. Both methods yielded similar results regarding the characteristics of players. The results we generated were

logical and could be interpreted.

One of our more interesting conclusions was that players make fewer bad moves when they have compensation for having less material.

Poglavje 1

Uvod

Namen tega diplomskega dela je analiza podatkov o šahovskih partijah z analizo posameznih pozicij in potez. Poteze so bile, glede na pozicijo v kateri so bile izvedene, z računalnikom ocenjene kako dobre so. Na podlagi teh ocen smo določili, ali je posamezna poteza slaba ali dobra. Da smo posamezno pozicijo in pripadajočo potezo lažje opisali, smo zgenerirali opisne atribute, kot na primer: ali igralec ima kraljico, ali ima lovski par in ga nasprotnik nima, ali je center šahovnice blokiran...

Analizirani podatki so izvirali iz iger dveh velemojstrov, Fischerja in Karpova, in iz partij šibkejših igralciev, ki so bili razdeljeni na skupine glede na njihov ELO rating.

Podatke smo ocenjevali z računalniškim programom Houdini 1.5a.

Podatke smo analizirali na dva načina. Najprej smo izvedli χ^2 test z dvema različnima ničelnima hipotezama:

- H_0: Vrednost nekega atributa ni povezana z verjetnostjo da je poteza dobra.

- H_0: Vrednost nekega atributa ni povezana z verjetnostjo da je poteza slaba.

Pri tem smo iskali hipoteze, ki so bile ovržene ali potrjene. χ^2 test smo uporabljali kot hevristiko za iskanje uporabnih atributov oziroma takih, pri

katerih pride do statistično pomembnih odstopanj pri verjetnosti za določen razred. Ugotavljali smo tudi kakšen vpliv na igro imajo določeni atributi oziroma pod katerimi pogoji določen igralec igra bolje ali slabše.

Na podlagi atributov smo tudi generirali pravila. Za generiranje pravil smo preizkušali več načinov. Pravila smo generirali z algoritmom CN2 z EVC (ang. *Extreme Value Correction*). Pravila, ki smo jih dobili, smo poskušali razlagati. Ugotavljali smo tudi, ali se naše ugotovitve, ki izhajajo iz pravil, skladajo s tistimi, ki izhajajo iz χ^2 testa.

Na podlagi analize smo ugotovili, da se isti atributi izkažejo za pomembne tako pri χ^2 testu kot tudi pri generiranju pravil. χ^2 test je dober način za ugotavljanje posameznih pomembnih atributov. Pravila, ki jih generiramo, pa logično združujejo več atributov.

Kot najpomembnejši atribut se izkaže *CompensatingLessMaterial*, saj prav vsi analizirani igralci delajo manj napak, ko imajo kompenzacijo za pomanjkanje materiala.

Programska koda, uporabljena za izdelavo diplomskega dela je bila napisana v programskem jeziku *Python*. *Python* je interpretiran programski jezik, ki uporablja dinamične podatkovne tipe. Za obdelavo podatkov in generiranje pravil smo v veliki meri uporabljali *Pythonov* modul *Orange*. *Orange* je odprtokodni projekt Laboratorija za bioinformatiko na Fakulteti za računalništvo in informatiko Univerze v Ljubljani.

Poglavje 2

Priprava podatkov

2.1 O podatkih

Podatki, ki smo jih uporabljali pri analizah, so bili pridobljeni iz podatkovne baze *Chessbase Megabase*.

Ti podatki za vsako potezo vsebujejo zaporedno številko poteze. V FEN notaciji je zapisana pozicija figur na šahovnici pred odigrano potezo. Podano imamo tudi potezo, ki jo je igralec odigral. Prav tako so izračunani ocena poteze, ki jo je odigral igralec, poteza, ki bi bila najbolje ocenjena, in ocena najboljše poteze. Ti podatki so izračunani z računalniškim šahovskim programom Houdini 1.5a na globini preiskovanja 20 polpotez.

Za to, da je bila analiza podatkov lažja, smo iz teh podatkov za vsako potezo definirali dodatne atribute.

Podatki, ki smo jih analizirali so dveh vrst:

- Podatki razdeljeni po ELO ratingih igralcev, ki so partije odigrali.
- Podatki o partijah dveh velemojstrov: Karpova in Fischerja.

2.1.1 Podatki razdeljeni po ELO ocenah igralcev

Podatki razdeljeni po ELO ocenah igralcev so bili zgenerirani z namenom primerjave boljših in slabših igralcev.

Igralci so ocenjeni po sistemu ocenjevanja ELO (Povzeto po [5]). Pri tem sistemu so igralci ocenjeni tako, da se njihova ocena izboljša, če premagajo igralca, ki je boljši od njih, in poslabša, če jih premaga slabši igralec. Razlika med ocenama igralcev tudi določi za koliko se ocena spremeni.

Podatke smo razdelili v dve skupini:

- Podatki o partijah igralcev, ocenjenih z ocenami med 1600 in 1999. Te igralce smo imenovali *amaterji*.
- Podatki o partijah igralcev, ocenjenih z ocenami med 2000 in 2299. Te igralce smo imenovali *klubski igralci*.

Podatki so podrobno opisani v tabeli 2.1. Atributi in njihove vrednosti, ki so podane v tabeli, so opisani v poglavju 2.5 na strani 8.

2.1.2 Podatki velemojstrov

Ti podatki vsebujejo pozicije iz partij dveh velemojstrov:

- Bobby Fischer, velemojster in svetovni šahovski prvak v letih 1972-1975.
- Anatolij Karpov, velemojster, svetovni šahovski prvak v letih 1975-1985 in svetovni šahovski prvak FIDE v letih 1993-1999.

Podatki so podrobno opisani v tabeli 2.2. Atributi in njihove vrednosti, ki so podane v tabeli, so opisani v poglavju 2.5 na strani 8.

	Amaterji	Klubski igralci
Vsi primeri	6339	5354
Class=Bad	827	478
Class=Good	3906	3735
Class=Dubious	1606	1141
Side=BLACK	2972	2565
Side=WHITE	3367	2789
CompensatingLessMaterial=No	5484	4672
CompensatingLessMaterial=Yes	855	682
CompensatingMoreMaterial=No	5981	5085
CompensatingMoreMaterial=Yes	358	269
PawnSymmetry=No	5360	4540
PawnSymmetry=Yes	979	814
Material=High	3006	2532
Material=Low	3333	2822
HasQueen=No	1808	1363
HasQueen=Yes	4531	3991
HasBishopPair=No	5784	4738
HasBishopPair=Yes	555	616
BlockedCenter=No	6007	4982
BlockedCenter=Yes	332	372
DoublePawnsPlayer=No	5573	4603
DoublePawnsPlayer=Yes	766	751
DoublePawnsOpponent=No	5369	4542
DoublePawnsOpponent=Yes	970	812
IsolatedQueensPawnPlayer=No	6290	5279
IsolatedQueensPawnPlayer=Yes	49	75
IsolatedQueensPawnOpponent=No	6291	5277
IsolatedQueensPawnOpponent=Yes	48	77
Imbalance=No	5582	4753
Imbalance=Yes	757	601

Tabela 2.1: Število primerov za skupine za določene vrednosti atributov.

	Fischer	Karpov
Vsi primeri	1338	1165
Class=Bad	161	130
Class=Good	1017	872
Class=Dubious	160	163
Side=BLACK	631	633
Side=WHITE	707	532
CompensatingLessMaterial=No	1094	958
CompensatingLessMaterial=Yes	244	207
CompensatingMoreMaterial=No	1186	1135
CompensatingMoreMaterial=Yes	152	30
PawnSymmetry=No	1211	982
PawnSymmetry=Yes	127	183
Material=High	521	491
Material=Low	817	674
HasQueen=No	578	351
HasQueen=Yes	760	814
HasBishopPair=No	1147	1016
HasBishopPair=Yes	191	149
BlockedCenter=No	1303	1095
BlockedCenter=Yes	35	70
DoublePawnsPlayer=No	1203	1035
DoublePawnsPlayer=Yes	135	130
DoublePawnsOpponent=No	1117	1068
DoublePawnsOpponent=Yes	221	97
IsolatedQueensPawnPlayer=No	1338	1161
IsolatedQueensPawnPlayer=Yes	0	4
IsolatedQueensPawnOpponent=No	1316	975
IsolatedQueensPawnOpponent=Yes	22	190
Imbalance=No	1155	1048
Imbalance=Yes	183	117

Tabela 2.2: Število primerov za igralce za določene vrednosti atributov.

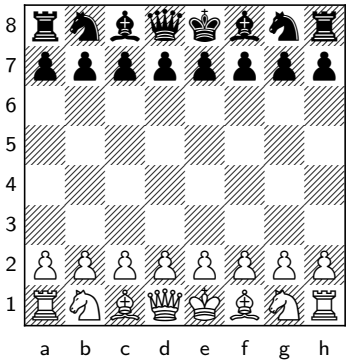
2.2 FEN

Ta atribut je že podan v podatkih, vendar se ne uporablja direktno za analizo. Iz njega se računa večina atributov. Ta atribut je zapis podatkov o poziciji in igri s Forsyth-Edwardsovo notacijo.

Za naše potrebe je pomemben predvsem prvi del zapisa, ki nam da podatke o pozicijah figur. Podatki za vsako vrsto na šahovnici so ločeni z znakom: /. Figure so označene s črkami: trdnjava z r, lovec z n, tekač z b, kraljica s q, kralj s k in kmet s p. Velike črke označujejo bele figure, male pa črne figure. Prazna mesta na šahovnici so označena s številkami oziroma s številom praznih mest.

FEN začetne postavitve figur na šahovnici (glej sliko 2.1) je tako naslednji:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1



Slika 2.1: Začetna postavitev šahovnice

2.3 Ocena

Podatek o oceni obsega tri atribute:

- *MovePlayedValue* - ocena poteze, ki jo je igralec odigral.

- *BestMove* - poteza, ki jo je program izbral kot najboljšo.
- *BestMoveValue* - ocena poteze, ki jo je program izbral kot najboljšo.

Ti podatki so bili izračunani z računalniškim programom Houdini 1.5a 64-bit pri globini preiskovanja 20 polpotez, z vklopljenim iskanjem mirovanja (ang. *quiescence search*). Houdini je trenutno eden izmed vodilnih računalniških šahovskih programov.

Ocena poteze je številka, kjer je 100 vrednost enega kmeta. Torej, če je razlika med *BestMoveValue* in *MovePlayedValue* enaka 100, je napaka enaka izgubi enega kmeta, brez nadomestila za pomanjkanje materiala.

Tu opisani atributi se ne uporabljajo direktno za analizo ampak se iz njih računajo nadaljni atributi.

2.4 Vrednost poteze

Za atribut vrednost poteze se uporablja ime *RelativeMoveValue*. Ta atribut se ne uporabi direktno pri kateri izmed analiz, ampak se uporabi za računanje drugih atributov, predvsem razreda. Atribut vrednost poteze ima številsko vrednost, ki pove, koliko slabša je poteza igralca v primerjavi s potezo, ki bi jo v tej situaciji naredil računalnik in naj bi bila najboljša.

Atribut izračunamo po formuli 2.1:

$$RelativeMoveValue = |MovePlayedValue - BestMoveValue| \quad (2.1)$$

MovePlayedValue je ocena poteze, ki je bila odigrana. *BestMoveValue* je ocena poteze, ki naj bi bila najboljša.

2.5 Opisni atributi

V tem delu so opisani atributi, ki se uporabljajo za analizo podatkov. Programska koda, s katero so bili atributi generirani je v dodatku B.

2.5.1 Barva figur

Ta atribut nam pove na kateri strani igra igralec, torej če igra s črnimi ali z belimi figurami. Zanj uporabljamo ime *Side*. Ta atribut ima dve možni vrednosti: *BLACK*, če igralec igra s črnimi figurami in *WHITE*, če igra z belimi figurami.

2.5.2 Prisotnost kraljice

Za ta atribut se uporablja ime *HasQueen*. Atribut ima dve možni vrednosti: *Yes*, če je igralčeva kraljica prisotna na šahovnici in *No*, če igralčeve kraljice na šahovnici ni več.

Kraljica je najmočnejša figura na šahovnici zaradi svoje gibljivosti. Premika se lahko po diagonalah, vertikalah in horizontalah.

2.5.3 Prisotnost lovskega para

Za ta atribut se uporablja ime *HasBishopPair*. Atribut ima dve možni vrednosti: *Yes* in *No*. Atribut ima vrednost *Yes* le v primeru, ko ima igralec na šahovnici še oba lovca, nasprotnik pa le enega ali nobenega. Drugače ima atribut vrednost *No*.

Lovski par je močnejša kombinacija kot par katerih drugih lahkih figur. Vsak lovec pokrije tista polja, ki jih drugi ne more, za to sta v kombinaciji zelo močna.

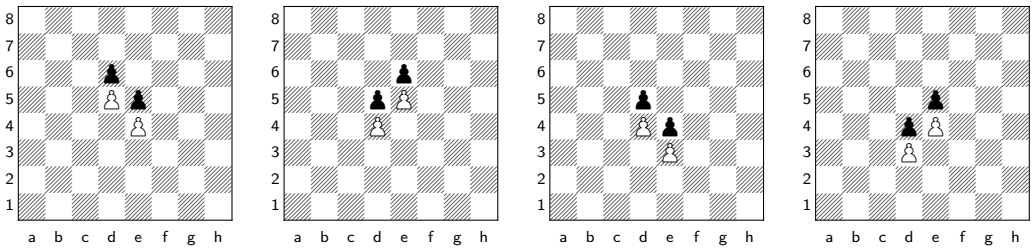
2.5.4 Blokiran center

Za ta atribut se uporablja ime *BlockedCenter*. Ta atribut ima dve možni vrednosti: *Yes*, ko je center šahovnice blokiran in *No*, ko ni. Center je lahko blokiran na enega izmed naslednjih načinov (glej sliko 2.2):

- Bela kmeta sta na *E4* in *D5*, črna kmeta sta na *E5* in *D6*.
- Bela kmeta sta na *E5* in *D4*, črna kmeta sta na *E6* in *D5*.

- Bela kmeta sta na *E3* in *D4*, črna kmeta sta na *E4* in *D5*.
- Bela kmeta sta na *E4* in *D3*, črna kmeta sta na *E5* in *D4*.

Pri blokiranem centru se igra tipično odvija po krilih šahovnice.



Slika 2.2: Pozicije pri katerih je center blokiran.

2.5.5 Osamljen damin kmet

Ta podatek je predstavljen z dvema atributoma:

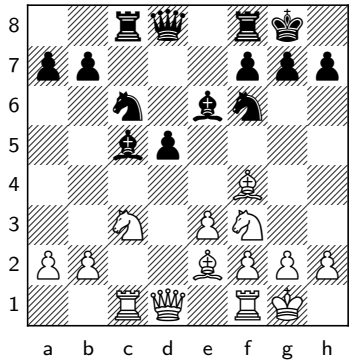
- *IsolatedQueensPawnPlayer*: ta atribut ima vrednost *Yes*, ko ima osamljenega daminega kmeta igralec, ki ga analiziramo, in *No*, kadar ga nima.
- *IsolatedQueensPawnOpponent*: ta atribut ima vrednost *Yes*, ko ima osamljenega daminiga kmeta nasprotnik igralca, ki ga analiziramo, in *No*, ko ga nima.

Vrednost posameznega atributa ugotovimo s pomočjo barve igralčevih figur in z analizo pozicij figur. Osamljeni damin kmet se lahko pojavi na enega izmed naslednjih načinov (glej sliko 2.3):

- Beli ima osamljenega daminega kmeta, ko je beli kmet na *D4*, črni kmet je na *E6* in/ali na *C6*, črni nima kmeta na liniji *D*, beli nima kmetov na linijah *D* in *E*.

- Črni ima osamljenega daminega kmeta, ko je črni kmet na *D5*, beli kmet je na *E3* in/ali na *C3*, beli nima kmeta na liniji *D*, črni nima kmetov na linijah *D* in *E*.

Osamljen damin kmet je pomemben, ker zaseda eno izmed centralnih linij na šahovnici. Takega kmeta bo igralec pogosto prisiljen zaščititi, vendar ker nima kmetov na linijah *C* in *E*, bo moral to storiti s katero izmed močnejših figur. Vendar lahko osamljeni kmet predstavlja tudi nevaren dinamični potencial, če nasprotnik ne uspe zaustaviti njegovega napredovanja. Igralec z osamljenim kmetom ima pogosto tudi več prostora za ostale figure.



Slika 2.3: Primer pozicije, kjer je prisoten osamljen damin kmet na strani črnega.

2.5.6 Dvojni kmetje

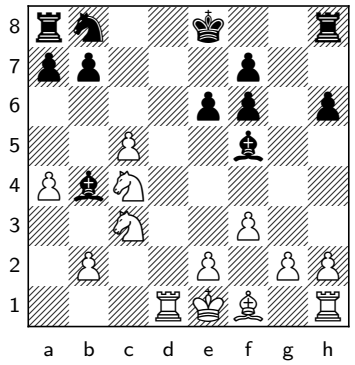
Ta podatek opišemo z dvema atributoma:

- *DoublePawnsPlayer*: ta atribut ima vrednost *Yes*, kadar so dvojni kmetje prisotni na strani igralca, ki ga analiziramo, in *No*, kadar jih ni.

- *DoublePawnsOpponent*: ta atribut ima vrednost *Yes*, kadar so dvojni kmetje na strani igralčevega nasprotnika, in *No*, kadar jih ni.

Dvojni kmetje so prisotni kadar sta dva kmeta iste barve na isti liniji (glej sliko 2.4).

Dvojni kmetje običajno predstavljajo slabost.



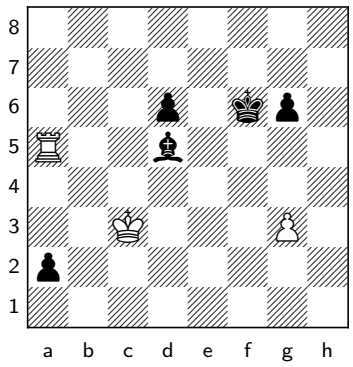
Slika 2.4: Primer pozicije, kjer so dvojni kmetje na liniji *F*, na strani črnega.

2.5.7 Materialna neravnovesja

Za ta atribut se uporablja ime *Imbalance*. Ta atribut ima dve možni vrednosti: *Yes*, kadar neravnovesje je, in *No*, kadar neravnovesja ni.

Pri računanju neravnovesja figuram pripišemo vrednosti. Kraljica dobi vrednost 9, trdnjava 5, lovec in skakač sta vredna 3 in kmet je vreden 1. To so vrednosti figur v kmetih: kraljica je vredna 9 kmetov. Pogledamo koliko katerih figur ima vsak izmed igralcev. Odstranimo tiste, ki se prekrivajo, npr.: če ima črni 5 kmetov in beli 3 kmete nam ostaneta 2 kmeta. Lovca in skakača štejemo kot enako figuro. Seštejemo vse vrednosti figur, ki nam ostanejo. Če je seštevek več ali enako 5, potem ima ta atribut vrednost *Yes*, drugače ima vrednost *No*.

Na sliki 2.5 je primer materialnega neravnovesja. Na primeru ima črni dva kmeta več od belega. Beli ima trdnjavo, ki je črni nima, črni pa lovca, ki ga beli nima.



Slika 2.5: Primer pozicije, kjer je prisotno materialno neravnovesje.

2.5.8 Material

Za ta atribut uporabljamo ime *Material*. Ta atribut ima dve možni vrednosti: *High*, kadar ima igralec veliko materiala in *Low*, kadar ga ima malo.

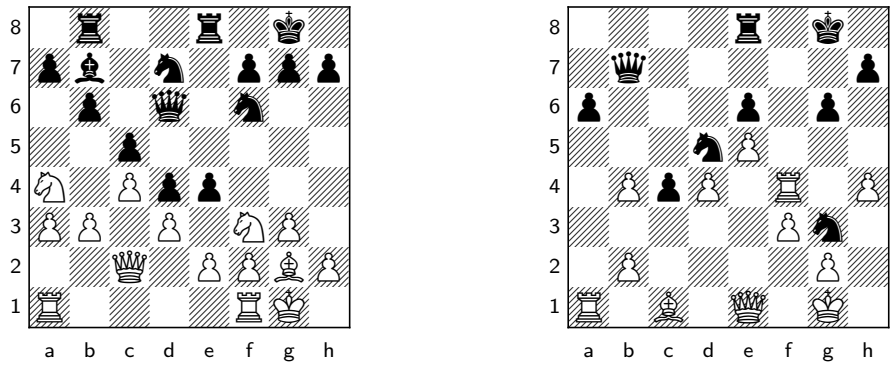
Količino materiala izračunamo tako, da seštejemo figure po njihovih vrednostih. Pri tem je kraljica vredna 9, trdnjava 5, skakač in lovec sta vredna 3. Vrednosti kmetov pri tej oceni nismo upoštevali.

Maksimalna količina materiala, ki jo igralec lahko ima je 31. Za končnico smo upoštevali, da se začne, ko je količina materiala 15. Pozicij, ki so v končnici, ne analiziramo. Za mejo med *high* in *low* smo zato vzeli aritmetično sredino med tema dvema vrednostima, kar je 23. Če je količina materiala, ki ga ima igralec, večja od 23, ima ta atribut vrednost *High*, če pa je manjša ali enaka 23, ima ta atribut vrednost *Low*.

2.5.9 Simetrična kmečka struktura

Za ta atribut uporabljamo ime *PawnSymmetry*. Ta atribut ima dve možni vrednosti: *Yes*, kadar so kmetje postavljeni simetrično, in *No*, kadar niso.

Kmečka struktura je simetrična, kadar je na vsaki liniji enako število kmetov vsake barve. Torej na liniji lahko sploh ni kmetov ali pa je en črn kmet in en bel kmet. Glej sliko 2.6.



(a) Simetrična kmečka struktura. (b) Nesimetrična kmečka struktura.

Slika 2.6: Primer pozicij s simetrično in nesimetrično kmečko strukturo.

2.5.10 Kompenzacija za materialni primanjkljaj

Ta podatek opišemo z dvema atributoma:

- *CompensatingMoreMaterial*: ta atribut ima vrednost *Yes*, kadar ima igralec več materiala kot nasprotnik in ima atribut *BestMoveValue* vrednost na intervalu $(-50, 50)$, v nasprotnem primeru ima vrednost *No*.

- *CompensatingLessMaterial*: ta atribut ima vrednost *Yes*, kadar ima igralec manj materiala kot nasprotnik in ima atribut *BestMoveValue* vrednost na intervalu $(-50, 50)$, v nasprotnem primeru ima vrednost *No*.

Pri računanju materiala za ta dva atributa se upoštevajo tudi kmetje, ki imajo vrednost 1.

Več materiala ponavadi pomeni prednost. Igralec lahko pomanjkanje materiala kompenzira z boljšimi pozicijami figur ali z aktivnejšo igro. Pozicije morajo imeti vrednost atributa *BestMoveValue* na intervalu $(-50, 50)$ - to pomeni, da je pozicija kljub materialnemu primanjkljaju približno izenačena.

2.5.11 Generiranje razreda

Za generiranje pravil je potrebno določiti razred poteze. Torej moramo vedeti ali je poteza dobra ali slaba, če jo hočemo analizirati.

Atribut za razred smo poimenovali *Class*. Ta atribut ima tri možne vrednosti *Good*, ko je poteza dobra, *Bad*, ko je poteza slaba in *Dubious*, ko za potezo ne moremo trditi ne, da je dobra in ne, da je slaba. Pri raznih analizah nas zanima predvsem kdaj ime pri poteza atribut *Class* vrednost *Good* ali *Bad*.

Razred smo generirali tako, da smo določili zgornjo in spodnjo meje vrednosti atributa *RelativeMoveValue*, pri katerem je poteza dobra ali slaba.

Za amaterje in klubske igralce smo te meje doloečili pri 10 in pri 50. Torej, če je bil *RelativeMoveValue* za nek primer manjši od 10 smo temu primeru določili razred *Good*, če je imel atribut *RelativeMoveValue* vrednost večjo od 50 smo primeru dali razred *Bad*, drugače pa smo mu določili razred *Dubious*. Za Karpova in Fischerja smo mejo za *Good* postavili pri 5 in za *Bad* pri 15.

2.6 Uporabljeni podatki

Vseh podatkov nismo uporabili za analizo. Nekatere smo izločili zaradi manjkajočih atributov, nekatere pa zaradi lažje analize ali tehničnih razlogov.

2.6.1 Faza igre

Odločili smo se, da bomo analizirali le središnjico, ker so v tem delu igre računalniki najboljši, in torj najbolj primerni kot ocenjevalci.

Za začetek središnjice smo vzeli 12. potezo v partiji. Konec središnjice oziroma začetek končnice pa smo definirali s pomočjo definicije, ki jo uporablja šahovski računalniški program Crafty. Končnica je, ko je skupna vrednost vseh figur na šahovnici manjša od 15, pri čemer ima dama vrednost 9, trdnjava 5, lovec in skakač pa 3, kmetje pa se pri tem izračunu ne upoštevajo.

Vse poteze, ki niso bile v središnjici, smo izločili.

2.6.2 Izločanje pozicij glede na oceno poteze

Če sta obe poteza, ki jo je igralec naredil in poteza, ki jo je računalnik izbral kot najboljšo pod -200 ali nad 200, potem primer izločimo. V tem primeru domnevamo, da je igralec namenoma naredil slabšo potezo, saj je hotel do zmage priti na, za človeka, lažji način, kjer je potrebno manj kompleksnega preračunavanja potez. Lahko pa da se je igralec poskušal rešiti iz zelo težke situacije, vendar je bilo pri tem potrebno odigrati nekaj, za računalnik neoptimalnih potez.

Za take poteze torej ne moremo trditi, da so slabe, čeprav bi jih računalnik za take označil, za to jih iz analize izločimo.

Poglavje 3

Uporabljene metode

V tem poglavju so opisane metode, ki smo jih uporabili za analizo podatkov.

3.1 χ^2 test

χ^2 test je namenjen preverjanju ali sta dve spremenljivki odvisni ali ne. To naredimo tako, da pojavitve vrednosti spremenljivk primerjamo s pričakovanim številom pojavitev.

Postavimo si ničelno hipotezo:

$$H_0\text{: spremenljivki sta neodvisni}$$

Sedaj izračunamo koeficient χ^2 po formuli 3.1, kjer je vseh vrednosti n, pričakovane vrednosti so označene z E, izmerjene pa z O.

$$\chi^2 = \sum_{i=1}^{n} \frac{(E_i - O_i)^2}{E_i} \tag{3.1}$$

Poznati moramo tudi število prostorskih stopenj k, ki je (formula 3.2):

$$k = n - 1 \tag{3.2}$$

Sedaj primerjamo izračunano vrednost χ^2 z distribucijo $\chi^2(k)$ (distribucija χ^2 s k prostorskimi stopnjami). S tem dobimo verjetnost, da H_0 drži, ki jo označimo s p. Izbrati si moramo še stopnjo značilnosi, običajno $\alpha = 0.05$.

Če je $p \leq \alpha$ potem je H_0 ovržena, če je $p \geq (1 - \alpha)$ je H_0 potrjena, če pa je $\alpha < p < (1 - \alpha)$ potem ničelne hipoteze ne moremo ne potrditi ne ovreči.

Pogoj, da je χ^2 test veljaven je, da nobena izmed pričakovanih in izmerjenih vrednosti ni manjša od 5, ker χ^2 test slabo deluje pri majhnih vrednostih.

Računali smo tudi smer odstopanja vrednosti po formuli 3.3:

$$Smer_i = \frac{E_i}{O_i} \tag{3.3}$$

Če je $Smer \geq 1$ smo izmerili manj vrednosti, kot smo jih pričakovali, če pa je $Smer \leq 1$ smo izmerili več vrednosti, kot smo jih pričakovali.

Ko smo izvajali χ^2 test smo za vsakega igrallca ali skupino igralcev testirali 26 hipotez. Ko testiramo tako število hipotez, se lahko zgodi, da je kakšna hipoteza potrjena ali ovržena po naključju. Eden izmed načinov, da ta problem rešimo je, da zmanjšamo α tako, da jo delimo s številom testiranih hipotez. Popravljeni α za naš primer je 0.002.

χ^2 test je v kodi implementiran na sledeč način:

```
import scipy.stats as s
import numpy
...
E=numpy.array([20, 30, 40])      #pričakovane vrednosti
O=numpy.array([22, 16, 52])      #izmerjene vrednosti
...
p=s.chisquare(O, E)[1]  # p je 0.00570
...                     #ničelna hipoteza je ovržena
```

3.2 Generiranje pravil

Za generiranje pravil smo uporabili metodo CN2 z EVC.

3.2.1 CN2

Opis je povzet po [1].

CN2 algoritem za generiranje pravil sta razvila Peter Clark in Tim Niblett. S svojim algoritmom sta poskušala izboljšati algoritem AQ tako, da bi deloval bolje na podatkih s šumom. Želela sta tudi, da preišče večji prostor pravil.

Algoritem deluje tako, da iterativno pregleduje množico učnih primerov. V vsaki iteraciji poišče pravilo, ki pokriva čim večje število primerov s ciljnim razredom in čim manjše število primerov z drugimi razredi. Novo zgenerirano pravilo doda na konec seznama pravil, iz učne množice pa odstrani vse primere, ki jih novo pravilo pokriva.

S tem postopkom na začetku dobimo bolj enostavna pravila nato pa vedno bolj kompleksna.

Pravila, ki jih dobimo učnih primerov ne klasificirajo popolno vendar delujejo bolje na novih podatkih.

3.2.2 CN2 z EVC

Opis je povzet po [2].

Pri večini algoritmov za generiranje pravil se kvaliteta pravil oceni z istimi podatki s katerimi so bila pravila zgenerirana. Posledično so pravila vedno zelo optimistična glede svoje točnosti. EVC (ang. *Extreme Value Correction*) to korigira s permutacijskim testom. Kvaliteto pravila popravi glede na to, kako verjetno je, da se bo to pravilo pojavilo pri naključni razporeditvi vrednosti pri danih podatkih.

Za implementacijo CN2 z EVC se uporablja *Orangeova* implementacija ABCN2. ABCN2 relevantnost pravil ugotavlja ne samo z EVC ampak tudi s podanimi argumenti. Argumentov mi ne podamo zato deluje enako kot CN2 ki mu dodamo EVC. Pri tem načinu generianja pravil lahko tudi določimo minimalno število primerov, ki naj jih pravilo pokriva. Ker želimo čim bolj relevantna pravila smo to omejitev postavili 10. Želeli smo tudi da vsak

dodaten atribut porazdelitev primerov ki jih pravilo pokriva signifikantno spremeni. Za to smo parameter att_sig nastavili na 0.05.

```
import Orange
...
data=...      #podatkovna tabela tipa Orange.data.Table
abcn2_learner=Orange.classification.rules.ABCN2(
                                        min_coverage=10, att_sig=0.05)
abcn2_classifier = cn2_learner(data)
...
#izpis dobljenih pravil
for r in abcn2_classifier.rules:
    print Orange.classification.rules.rule_to_string(r)
...
```

Poglavje 4

Rezultati

V tem poglavju so predstavljeni rezultati χ^2 testov in generiranja pravil.

Rezultate smo razlagali s pomočjo strokovnjaka za šah[1].

Pri rezultatih χ^2 testov so pri primerih, kjer je hipoteza ovržena ali potrjena, odebeljeno napisane verjetnosti s katero hipoteza drži. Rezultati χ^2 testov, ki niso veljavni zaradi premajhnega števila izmerjenih ali pričakovanih primerov, niso prikazani. Da je χ^2 test veljaven število pričakovanih in izmerjenih primerov ne sme biti manjše od 5.

Pri rezultatih generiranja pravil so navedena samo pravila, ki jih je naš strokovnjak za šah izbral kot zanimiva ali pomembna. Ostala pravila so v dodatku A na strani 41.

Ko opisujemo pravila so le ta v sledečem formatu:

```
IF Atribut=["Vrednost atributa"]
AND Atribut2=["Vrednost atributa2"] AND ...
THEN Class=Razred, ki ga pravilo napoveduje
<Razredi primerov, ki jih pravilo pokriva>
```

Prvi del pravila so pogoji, da nek primer spada pod neko pravilo, nato sledi razred, ki ga to pravilo napoveduje. Na koncu, v lomljenih oklepajih je napisana porazdelitev učnih primerov, ki jih to pravilo pokriva, po razredih. Pri tem je prva številka število učnih primerov, ki ima razred *Bad*, druga je

[1] V vlogi šahovskega eksperta je bil Matej Guid, FIDE mojster

število učnih primerov, ki ima razred *Good* in tretja število učnih primerov, ki ima razred *Dubious*

Primer pravila:

```
IF CompensatingLessMaterial=['Yes'] AND PawnSymmetry=['Yes']
THEN Class=Good<0.000, 57.000, 3.000>
```

Njegov pomen:

Če ima atribut *CompensatingLessMaterial* vrednost *Yes* in atribut *PawnSymmetry* vrednost *Yes* potem je razred *Good*. Med učnimi primeri, ki jih to pravilo pokriva je 57 primerov z razredom *Good*, 3 primere z razredom *Dubious* in niti enega primera z razredom *Bad*. Pravilo izraža, da se analizirani igralci izvrstno odrežejo v pozicijah v katerih imajo kompenzacijo za pomanjkanje materiala in za katere velja, da je kmečka struktura simetrična.

4.1 Amaterji

Pri teh igralcih smo poteze označili za slabe, ko je imel atribut *RelativeMoveValue* vrednost večjo od 50, za dobre pa, ko je imel vrednost manjšo od 10.

4.1.1 Rezultati χ^2 testa

S χ^2 testom smo testirali dve hipotezi:

- H_0: `Verjetnost slabe poteze ni povezana z vrednostjo atributa` (Rezultati v tabeli 4.1)

  Za $\alpha = 0.05$ je hipoteza ovržena pri atributih *CompensatingLessMaterial*, *PawnSymmetry*, *Material*, *HasQueen*, *BlockedCenter* in *Imbalance*.

  Za popravljeni $\alpha = 0.002$ je hipoteza ovržena pri atributih *CompensatingLessMaterial*, *PawnSymmetry*, *HasQueen*, *BlockedCenter* in *Imbalance*.

- H_0: `Verjetnost dobre poteze ni povezana z vrednostjo atributa` (Rezultati v tabeli 4.2)

  Za $\alpha = 0.05$ je hipoteza ovržena pri atributih *CompensatingLessMaterial*, *Material* in *HasQueen*.

  Za popravljeni $\alpha = 0.002$ je hipoteza ovržena pri atributu *CompensatingLessMaterial*.

Atribut	P	Smer	
Side	0.875	BLACK: 0.994	WHITE: 1.005
CompensatingLessMaterial	**<0.001**	No: 0.938	Yes: 1.743
CompensatingMoreMaterial	0.056	No: 0.984	Yes: 1.374
PawnSymmetry	**<0.001**	No: 0.918	Yes: 1.965
Material	**0.012**	High: 1.102	Low: 0.923
HasQueen	**0.001**	No: 1.229	Yes: 0.931
HasBishopPair	0.154	No: 1.016	Yes: 0.862
BlockedCenter	**<0.001**	No: 0.972	Yes: 2.063
DoublePawnsPlayer	0.744	No: 1.004	Yes: 0.970
DoublePawnsOpponent	0.667	No: 1.006	Yes: 0.966
IsolatedQueensPawnPlayer	0.301	No: 1.003	Yes: 0.710
IsolatedQueensPawnOpponent	0.272	No: 1.003	Yes: 0.696
Imbalance	**<0.001**	No: 1.065	Yes: 0.691

Tabela 4.1: Rezultati χ^2 testa za amaterje za H_0: Verjetnost slabe poteze ni povezana z vrednostjo atributa.

Ko ima igralec kompenzacijo za manj materiala dela manj napak, kot je pričakovano. Prav tako v takih situacijah odigra več dobrih potez, kot jih pričakujemo. Za pozicije s kompenzacijo za materialni primanjkljaj pogosto velja, da ima igralec z manj materiala pobudo. Prav tako je tipično za te pozicije, da imajo forsiran značaj. Razlika med dobrimi in slabimi potezami je zato pogosto tako velika, da se je slabim potezam lažje izogniti.

Atribut	P	Smer	
Side	0.570	BLACK: 0.990	WHITE: 1.009
CompensatingLessMaterial	**<0.001**	No: 1.038	Yes: 0.808
CompensatingMoreMaterial	0.390	No: 1.003	Yes: 0.947
PawnSymmetry	0.514	No: 1.004	Yes: 0.976
Material	**0.010**	High: 1.045	Low: 0.962
HasQueen	**0.023**	No: 0.946	Yes: 1.023
HasBishopPair	0.429	No: 0.996	Yes: 1.043
BlockedCenter	0.644	No: 1.002	Yes: 0.970
DoublePawnsPlayer	0.694	No: 1.002	Yes: 0.983
DoublePawnsOpponent	0.442	No: 1.005	Yes: 0.972
IsolatedQueensPawnPlayer	0.608	No: 1.001	Yes: 0.915
IsolatedQueensPawnOpponent	0.317	No: 1.001	Yes: 0.845
Imbalance	0.132	No: 1.009	Yes: 0.939

Tabela 4.2: Rezultati χ^2 testa za amaterje za H_0: Verjetnost dobre poteze ni povezana z vrednostjo atributa.

Tudi pri simetričnih kmečkih strukturah ti igralci delajo manj napak, kot bi jih od njih pričakovali, glede na apriorno verjetnost napake. Znan primer tega je menjalna francoska obramba, ki se je boljši igralci v partijah proti slabšim izogibajo, saj vodi v simetrično kmečko strukturo. Simetrične pozicije so lažje za igranje kot nesimetrične, saj le-te zahtevajo več znanja in izkušenj.

V situacijah, ko imajo več materiala ti igralci delajo manj napak. To morda ni presenetljivo. Znano je, da boljši igralci v partijah proti slabšim raje zavijejo v končnico. To pa še ne pomeni, da v pozicijah z več materiala blestijo. Izkaže se, da delajo manj napak in hkrati manj dobrih potez, kot bi pričakovali glede na apriorno verjetno dobre in slabe poteze pri tem atributu.

Na igro vpliva tudi to ali igralec ima ali nima kraljice. Rezultati kažejo na to, da delajo manj napak in hkrati več dobrih potez kot bi pričakovali, ko

njihove kraljice ni več na šahovnici. Prisotnost kraljice namreč poveča število možnih potez in s tem tudi možnosti za napake. Pri tem podatku pa pridemo tudi do neskladja z rezultati o količini materiala. Če kraljice na šahovnici ni, je materiala malo in naj bi ti igralci delali več napak, vendar naj bi jih zaradi odsotnosti kraljice delali manj. Tu obstaja več možnih razlag. Lahko, da s kraljico igrajo slabo, ko imajo materiala že malo zaradi izgube drugih figur. Lahko pa da na začetku igre, ko je materiala veliko, igrajo dobro ne le zaradi količine materiala ampak tudi zaradi boljšega poznavanja otvoritev in nadaljnjih potez za ta del igre ter zaradi odsotnosti časovne stiske, ki se pojavi v nadaljevanju partije.

V pozicijah z blokiranim centrom amaterji delajo manj napak kot bi pričakovali. Za to je lahko več možnih razlogov. Če do take pozicije pride na začetku igre, je lahko razlog za manj slabih potez več časa za razmislek in boljše poznavanje teorije na začetku igre. Drug možen razlog je lahko to, da je v pozicijah z blokiranim centrom več manevriranja, ker igra poteka bolj po kraljevem ali daminem krilu. Igralec ima v takih pozicijah tipično več časa in nekoliko manj možnosti za napake.

Neravnovesne situacije tipično veljajo za bolj zahtevne, zato je logično, da ti igralci v takih situacijah delajo več napak, kot bi pričakovali.

4.1.2 Rezultati generiranja pravil

V tem poglavju so razlage za izbrana pravila. Celoten seznam pravil za amaterje je v dodatku A.1 na strani 41.

```
IF CompensatingLessMaterial=['Yes'] AND PawnSymmetry=['Yes']
THEN Class=Good<0.000, 57.000, 3.000>
```

V tem primeru sta združena dva atributa, ki se izkažeta za pomembna že pri χ^2 testu. Ti igralci igrajo dobro, ko imajo kompenzacijo za manj materiala in ko ima pozicija simetrično kmečko strukturo.

```
IF CompensatingLessMaterial=['Yes'] AND Material=['Low']
THEN Class=Good<39.000, 427.000, 72.000>
```

Čeprav χ^2 test kaže na to, da igralci delajo več napak in manj dobrih potez, ko imajo materiala manj pa se izkaže, da manj materiala v kombinaciji s kompenzacijo za manj materiala kaže, da bo igralec igral dobro.

```
IF CompensatingLessMaterial=['Yes']
THEN Class=Good<64.000, 652.000, 139.000>
```

Atribut *CompensatingLessMaterial* se je izkazal za zelo pomembnega pri skoraj vsaki analizi in to odražajo tudi pravila.

```
IF Imbalance=['Yes'] AND HasQueen=['Yes']
AND CompensatingLessMaterial=['No']
THEN Class=Bad<106.000, 216.000, 69.000>
```

V tem pravilu je združenih veliko atributov, ki že sami kažejo na povečano število napak. Ko sestavimo pravilo z vsemi skupaj pa se verjetnost za slabo potezo še poveča.

```
IF Imbalance=['Yes'] AND CompensatingMoreMaterial=['No']
AND PawnSymmetry=['No'] THEN Class=Bad<135.000, 396.000, 96.000>
```

To pravilo nam pove, da ta skupina igralcev igra slabše v situacijah, kjer obstaja materialno neravnovesje in kjer ni simetričnih kmečkih struktur. To lahko razložimo s tem, da so take pozicije bolj zapletene in jih je zato težje odigrati brez napak.

```
IF CompensatingLessMaterial=['Yes'] AND Imbalance=['Yes']
THEN Class=Good<14.000, 160.000, 11.000>
```

V pozicijah, kjer obstaja materialno neravnovesje in kjer imajo igralci kompenzacijo za manj materiala amaterski igralci igrajo dobro.

```
IF Imbalance=['Yes'] AND PawnSymmetry=['Yes']
THEN Class=Good<2.000, 36.000, 1.000>
```

Kljub temu, da ti igralci v pozicijah z materialnim neravnovesjem večkrat igrajo slabše pa se to spremeni, če ima pozicija hkrati tudi simetrično kmečko strukturo.

4.2 Klubski igralci

Pri teh igralcih smo poteze označili za slabe, ko je imel atribut *RelativeMoveValue* vrednost večjo od 50, za dobre pa, ko je imel vrednost manjšo od 10.

4.2.1 Rezultati χ^2 testa

Pri tej skupini pričakujemo, da bodo rezultati podobni kot pri amaterjih, saj obe skupini vsbujeta podatke večih igralcev in pri obeh skupinah zato najdemo lastnosti, ki niso specifične posamezniku. To se tudi zgodi. χ^2 testov sicer ne moremo primerjati tako da bi rekli, da je ena hipoteza "bolj ovržena" ali "bolj potrjena" od druge, vidimo pa, da so hipoteze, ki so ovržene pri klubskih igralcih ovržene tudi pri amaterjih. Nekaj hipotez pa je takih, ki so ovržene samo pri amaterjih.

S χ^2 testom smo testirali dve hipotezi:

- H_0: `Verjetnost slabe poteze ni povezana z vrednostjo atributa` (Rezultati v tabeli 4.3)

  Za $\alpha = 0.05$ je hipoteza ovržena pri atributih *CompensatingLessMaterial*, *PawnSymetri*, *HasQueen* in *Imbalance*.

  Za popravljeni $\alpha = 0.002$ je hipoteza ovržena pri atributih *CompensatingLessMaterial*, *PawnSymetri*, *HasQueen* in *Imbalance*.

- H_0: `Verjetnost dobre poteze ni povezana z vrednostjo atributa` (Rezultati v tabeli 4.4)

  Za $\alpha = 0.05$ je hipoteza ovržena pri atributih *CompensatingLessMaterial* in *HasQueen*.

  Za popravljeni $\alpha = 0.002$ je hipoteza ovržena pri atributu *CompensatingLessMaterial*.

Tako kot se je izkazalo pri slabših igralcih tudi klubski igralci delajo manj napak in več dobrih potez kot bi pričakovali, ko imajo kompenzacijo za manj

Atribut	P	Smer	
Side	0.170	BLACK: 1.070	WHITE: 0.943
CompensatingLessMaterial	**0.002**	No: 0.948	Yes: 1.602
CompensatingMoreMaterial	0.294	No: 0.989	Yes: 1.264
PawnSymmetry	**<0.001**	No: 0.917	Yes: 2.019
Material	0.198	High: 1.066	Low: 0.947
HasQueen	**<0.001**	No: 1.540	Yes: 0.893
HasBishopPair	0.473	No: 1.012	Yes: 0.917
BlockedCenter	0.563	No: 0.993	Yes: 1.107
DoublePawnsPlayer	0.688	No: 0.993	Yes: 1.048
DoublePawnsOpponent	0.180	No: 1.027	Yes: 0.873
IsolatedQueensPawnPlayer	0.370	No: 1.005	Yes: 0.744
IsolatedQueensPawnOpponent	0.471	No: 0.996	Yes: 1.375
Imbalance	**<0.001**	No: 1.064	Yes: 0.679

Tabela 4.3: Rezultati χ^2 testa za klubske igralce za H_0: Verjetnost slabe poteze ni povezana z vrednostjo atributa.

materiala. Podobno se zgodi tudi, ko kraljice ni na šahovnici. Tudi oni takrat igrajo dobro in delajo manj napak.

Pri simetričnih kmečkih strukturah delajo manj napak kot v povprečju.

Hipoteza je ovržena tudi pri atributu *Imbalance*. Ko so pozicije v materialnem neravnovesju delajo več napak kot bi jih pričakovali glede na apriorno verjetnost.

Iz rezultatov χ^2 testa za atribute *Imbalance*, *PawnSymmetry* in *HasQueen* lahko sklepamo, da tudi ti igralci delajo več napak v bolj zapletenih in manj napak v manj zapletenih pozicijah, podobno kot amaterji.

Atribut	P	Smer	
Side	0.439	BLACK: 0.987	WHITE: 1.012
CompensatingLessMaterial	**<0.001**	No: 1.026	Yes: 0.851
CompensatingMoreMaterial	0.396	No: 1.003	Yes: 0.943
PawnSymmetry	0.519	No: 1.004	Yes: 0.976
Material	0.065	High: 1.033	Low: 0.972
HasQueen	**0.008**	No: 0.930	Yes: 1.026
HasBishopPair	0.363	No: 0.995	Yes: 1.043
BlockedCenter	0.499	No: 0.997	Yes: 1.042
DoublePawnsPlayer	0.921	No: 1.001	Yes: 0.996
DoublePawnsOpponent	0.734	No: 0.998	Yes: 1.013
IsolatedQueensPawnPlayer	0.547	No: 0.999	Yes: 1.090
IsolatedQueensPawnOpponent	0.652	No: 1.001	Yes: 0.942
Imbalance	0.614	No: 1.003	Yes: 0.977

Tabela 4.4: Rezultati χ^2 testa za klubske igralce za H_0: Verjetnost dobre poteze ni povezana z vrednostjo atributa.

4.2.2 Rezultati generiranja pravil

V tem poglavju so razlage za izbrana pravila. Celoten seznam pravil za klubske igralce je v dodatku A.2 na strani 45.

Pravila, ki smo jih zgenerirali s podatki klubskih igralcev so podobna tistim, ki smo jih zgenerirali s podatki amaterjev.

```
IF CompensatingLessMaterial=['Yes']
THEN Class=Good<38.000, 559.000, 85.000>
```

Tudi pri teh igralcih je *CompensatingLessMaterial* eden izmed najpomembnejših atributov. To odraža tudi zgornje pravilo.

```
IF CompensatingLessMaterial=['Yes'] AND PawnSymmetry=['Yes']
THEN Class=Good<0.000, 67.000, 3.000>
```

Verjetnost za dobro potezo se še izboljša ko ima pozicija simetrično kmečko strukturo poleg tega, da ima igralec kompenzacijo za pomanjkanje materiala.

IF CompensatingLessMaterial=['Yes'] AND HasQueen=['No'] THEN Class=Good<4.000, 168.000, 15.000>

Veliko dobrih potez se pojavlja tudi, ko ima igralec kompenzacijo za pomanjkanje materiala in hkrati nima kraljice.

IF HasQueen=['No'] THEN Class=Good<79.000, 1022.000, 262.000>

Tudi že samo odsotnost kraljice močno pripomore k igranju boljših potez. Brez kraljice je namreč veliko manj možnih potez torej je v povprečju tudi manj možnosti za napake.

Večino ugotovitev iz pravil smo videli že pri rezultatih χ^2 testa, zanimive so kombinacije atributov, ki jih vidimo v pravilih.

4.3 Fischer

Pri tem igralcu smo poteze označili za slabe, ko je imel atribut *RelativeMoveValue* vrednost večjo od 15, za dobre pa, ko je imel vrednost manjšo od 5.

Te meje so veliko nižje kot pri podatkih o amaterjih in o klubskih igralcih vendar je to potrebno, saj velemojstri igrajo veliko bolje.

4.3.1 Rezultati χ^2 testa

S χ^2 testom smo testirali dve hipotezi:

- H_0: `Verjetnost slabe poteze ni povezana z vrednostjo atributa` (Rezultati v tabeli 4.5)

  Za $\alpha = 0.05$ je hipoteza ovržean pri atributih *Side*, *CompensatingLessMaterial* in *PawnSymmetry*.

  Za popravljeni $\alpha = 0.002$ je hipoteza ovržena pri atributu *CompensatingLessMaterial*.

- H_0: `Verjetnost dobre poteze ni povezana z vrednostjo atributa` (Rezultati v tabeli 4.6)

Atribut	P	Smer	
Side	**0.004**	BLACK: 0.808	WHITE: 1.270
CompensatingLessMaterial	**0.002**	No: 0.896	Yes: 2.097
CompensatingMoreMaterial	0.671	No: 1.012	Yes: 0.914
PawnSymmetry	**0.026**	No: 0.946	Yes: 2.183
Material	0.593	High: 0.950	Low: 1.035
HasQueen	0.174	No: 1.140	Yes: 0.914
HasBishopPair	0.819	No: 1.007	Yes: 0.958
DoublePawnsPlayer	0.646	No: 1.012	Yes: 0.902
DoublePawnsOpponent	0.609	No: 1.018	Yes: 0.917
Imbalance	0.815	No: 0.993	Yes: 1.049

Tabela 4.5: Rezultati χ^2 testa za Fischerja za H_0: Verjetnost slabe poteze ni povezana z vrednostjo atributa.

Iz rezultatov χ^2 se da sklepati, da je Fischer s črnimi figurami delal več napak, kot bi pričakovali glede na apriorno verjetnost. Ena izmed možnih razlag za to je, da je Fischer igral zapletene otvoritve s črnimi figurami. Te otvoritve vodijo v bolj zapletene pozicije v kasnejših fazah igre.

Tako kot pri ostalih rezultatih analize se tudi pri Fischerju pokaže, da je delal manj napak, ko je imel kompenzacijo za pomanjkanje materiala. Prav tako je v takih situacijah odigral več dobrih potez.

Odstopajoči rezultati so tudi pri atributu *PawnSymmetry*. Pri pozicijah s simetrično kmečko strukturo Fischer dela manj napak kot bi pričakovali glede na apriorno verjetnost.

Atribut	P	Smer	
Side	0.724	BLACK: 1.012	WHITE: 0.990
CompensatingLessMaterial	0.056	No: 1.029	Yes: 0.887
CompensatingMoreMaterial	0.885	No: 1.002	Yes: 0.987
PawnSymmetry	0.870	No: 0.998	Yes: 1.016
Material	0.368	High: 1.037	Low: 0.978
HasQueen	0.627	No: 0.983	Yes: 1.013
HasBishopPair	0.666	No: 1.006	Yes: 0.968
BlockedCenter	0.753	No: 0.998	Yes: 1.064
DoublePawnsPlayer	0.724	No: 1.004	Yes: 0.968
DoublePawnsOpponent	0.801	No: 0.997	Yes: 1.018
IsolatedQueensPawnOpponent	0.502	No: 0.997	Yes: 1.194
Imbalance	0.174	No: 1.017	Yes: 0.903

Tabela 4.6: Rezultati χ^2 testa za Fischerja za H_0: Verjetnost dobre poteze ni povezana z vrednostjo atributa.

4.3.2 Rezultati generiranja pravil

V tem poglavju so razlage za izbrana pravila. Celoten seznam pravil za Fischerja je v dodatku A.3 na strani 47.

```
IF Side=['BLACK'] AND PawnSymmetry=['No']
THEN Class=Bad<91.000, 421.000, 52.000>
```

To, da Fischer igra slabše s črnimi figurami smo ugotovili že iz χ^2 testa. Zgornje pravilo nam pove, da je imel še dodatne težave pri nesimetričnih kmečkih strukturah.

```
IF DoublePawnsPlayer=['Yes'] AND DoublePawnsOpponent=['Yes']
THEN Class=Good<0.000, 28.000, 2.000>
```

To pravilo je zanimivo, ker nam pove, da je Fischer igral zelo dobro, ko so bili na šahovnici dvojni kmetje. Zanimivo je tudi, ker se ta dva atributa nista pojavila med zanimivimi atributu pri χ^2 testu.

```
IF Imbalance=['Yes'] AND DoublePawnsPlayer=['Yes']
THEN Class=Good<0.000, 20.000, 0.000>
```

Zanimiv je tudi ta rezultat, ki pokaže, da je bil dober ko je imel dvojne kmete in ko je bila pozicija neravnovesna.

Zelo zanimivo je tudi naslednje pravilo:

```
IF Imbalance=['Yes'] THEN Class=Good<21.000, 154.000, 8.000>
```

To pravilo je zanimivo predvsem v smislu primerjave s praviom, ki je bilo naučeno pri Karpovu:

```
IF Imbalance=['Yes'] THEN Class=Bad<16.000, 95.000, 6.000>
```

Ti dve pravili namreč pravita, da je Fischer v situacijah z materialnim neravnovesjem igral dobro, med tem ko je Karpov v takih situacijah igral slabo.

4.4 Karpov

Pri tem igralcu smo poteze označili za slabe, ko je imel atribut *RelativeMoveValue* vrednost večjo od 15, za dobre pa, ko je imel vrednost manjšo od 5.

4.4.1 Rezultati χ^2 testa

S χ^2 testom smo testirali dve hipotezi:

- H_0: `Verjetnost slabe poteze ni povezana z vrednostjo atributa` (Rezultati v tabeli 4.7)

  Za $\alpha = 0.05$ je hipoteza ovržena pri atributu *CompensatingLessMaterial*.

- H_0: `Verjetnost dobre poteze ni povezana z vrednostjo atributa` (Rezultati v tabeli 4.8)

Za $\alpha = 0.05$ je hipoteza ovržena pri atributu *CompensatingLessMaterial*.

Za $\alpha = 0.05$ je hipoteza potrjena pri atributu *HasBishopPair*.

Atribut	P	Smer	
Side	0.677	BLACK: 0.968	WHITE: 1.041
CompensatingLessMaterial	**0.020**	No: 0.914	Yes: 1.777
PawnSymmetry	0.732	No: 0.987	Yes: 1.075
Material	0.501	High: 1.074	Low: 0.952
HasQueen	0.238	No: 1.187	Yes: 0.936
HasBishopPair	0.251	No: 1.040	Yes: 0.792
BlockedCenter	0.239	No: 1.027	Yes: 0.710
DoublePawnsPlayer	0.070	No: 0.947	Yes: 1.813
DoublePawnsOpponent	0.709	No: 1.010	Yes: 0.902
IsolatedQueensPawnOpponent	0.087	No: 0.938	Yes: 1.514
Imbalance	0.390	No: 1.026	Yes: 0.816

Tabela 4.7: Rezultati χ^2 testa za Karpova ocenjenega s Houdinijem za H_0: Verjetnost slabe poteze ni povezana z vrednostjo atributa.

Tako kot veliko drugih igralcev tudi Karpov dela manj napak in hkrati več dobrih potez, ko ima kompenzacijo za pomanjkanje materiala. Razlogi za to so podobni kot pri ostalih igralcih. V pozicijah s forsiranim značajem, kot to tipično velja za ta atribut, je lažje poiskati boljšo potezo.

Pri tem igralcu je bila pri atributu *HasBishopPair* hipoteza, da verjetnost dobre poteze ni povezana z vrednostjo atributa. To pomeni da posedovanje lovskega para ne vpliva na kvaliteto igre.

4.4.2 Rezultati generiranja pravil

V tem poglavju so razlage za izbrana pravila. Celoten seznam pravil za Karpova je v dodatku A.4 na strani 49.

Atribut	P	Smer	
Side	0.532	BLACK: 0.981	WHITE: 1.024
CompensatingLessMaterial	**0.013**	No: 1.041	Yes: 0.847
CompensatingMoreMaterial	0.586	No: 1.003	Yes: 0.898
PawnSymmetry	0.928	No: 0.999	Yes: 1.007
Material	0.287	High: 1.044	Low: 0.970
HasQueen	0.494	No: 0.966	Yes: 1.015
HasBishopPair	**0.957**	No: 0.999	Yes: 1.005
BlockedCenter	0.629	No: 0.996	Yes: 1.069
DoublePawnsPlayer	0.297	No: 1.013	Yes: 0.909
DoublePawnsOpponent	0.844	No: 0.998	Yes: 1.023
IsolatedQueensPawnOpponent	0.699	No: 0.994	Yes: 1.031
Imbalance	0.403	No: 1.010	Yes: 0.922

Tabela 4.8: Rezultati χ^2 testa za Karpova ocenjenega s Houdinijem za H_0: Verjetnost dobre poteze ni povezana z vrednostjo atributa.

Pri Karpovu je bilo naučenih le 6 pravil. Edino, ki se nam je zdelo izstopajoče je bilo tisto, ki smo ga omenili že pri rezultatih generiranja pravil za Fischerja.

To pravilo je zanimivo predvsem iz stališča iskanja razlik med Fischerjem in Karpovom, saj pravilo za Karpova pravi, da dela napake v pozicijah z materialnim neravnovesjem medtem, ko Fischer v takih pozicijah igra bolje.

Karpov:

```
IF Imbalance=['Yes'] THEN Class=Bad<16.000, 95.000, 6.000>
```

Fischer

```
IF Imbalance=['Yes'] THEN Class=Good<21.000, 154.000, 8.000>
```

Poglavje 5

Sklepne ugotovitve

Iz rezultatov smo ugotovili, da tako pri χ^2 testu kot pri pravilih izstopajo enaki atributi. Atributi, ki se ne pojavljajo kot signifikantni pri χ^2 testu pa se tudi zelo redko pojavljajo v pravilih. Iz tega lahko sklepamo, da je χ^2 test dobra hevristika za ugotavljanje pomembnih atributov.

Pravila, ki so zgenerirana s CN2 z EVC se izkažejo za uporaben način za združevanje atributov in pravila, ki smo se jih naučili s to metodo, so s stališča šahista videti smiselna.

Med vsemi atributi, ki smo jih uporabljali, se je za najpomembnejšega izkazal atribut *CompensatingLessMaterial*. Vsi igralci zelo konsistentno delajo manj napak in hkrati več dobrih potez, ko imajo kompenzacijo za pomanjkanje materiala.

Rezultati računalniške analize so pokazali, da so amaterji slabši v bolj kompleksnih situacijah in boljši v manj kompleksnih situacijah. Slabši so, ko ima pozicija nesimetrično kmečko strukturo, ko imajo na šahovnici kraljico in ko je prisotno materialno neravnovesje.

Pri klubskih igralcih so bili rezultati zelo podobno kot pri amaterjih, le da prvi igrajo nekoliko bolje. χ^2 test je bil ovržen pri manj atributih.

Za Fischerja se je izkazalo, da je nekoliko slabši s črnimi figurami. Precej boljši, zlasti v primerjavi s Karpovom, pa je v pozicijah z materialnim neravnovesjem.

Za Karpova lahko s pomočjo χ^2 testa ugotovimo, da igra zelo uravnoteženo. Za edini izstopajoči atribut se je izkazal *CompensatingLessMaterial*, ki je sicer izstopal tudi pri ostalih.

Pri nadaljnjih raziskavah bi bilo dobro upoštevati še več atributov. Analiza bi bila lažja in bolj natančna, če bi imeli na voljo več podatkov še zlasti za velemojstre, saj le-ti delajo zelo malo napak. Zanimivo bi bilo tudi, če bi imeli podatke o času igranja, saj bi tako lahko spoznali značilnosti igre v časovnih stiskah.

Literatura

[1] P. Clark, T. Niblett. "The CN2 Induction Algorithm", *Machine Learning*, št. 4, zv. 3, str. 261-283, 1988.

[2] M. Možina, J. Demšar, J. Žabkar, I. Bratko. "Why is Rule Learning Optimistic and How to Correct It", v zborniku *Proceedings of 17th European Conference on Machine Learning (ECML 2006)*, Berlin, 2006, str. 330-340.

[3] T. Curk, J. Demšar, Q. Xu, G. Leban, U. Petrovič, I. Bratko, G. Shaulsky, B. Zupan. "Microarray data mining with visual programming", v zborniku *Bioinformatics*, št. 3, zv. 21, str. 396-398, 2005.

[4] M. Guid, I. Bratko. "Računalniška primerjava svetovnih prvakov", *Šahovska misel*, št. 1, str. 36-44, 2007.

[5] P. C. H. Albers, H. de Vries. "Elo-rating as a tool in the sequential estimation of dominance strengths", *Animal Behaviour*, št. 2, zv. 61, str. 489-495, 2001.

Dodatek A

Celotni rezultati generiranja pravil

A.1 Amaterji

Celoten seznam pravil zgeneriranih za skupino amaterji.

```
IF Imbalance=['Yes'] AND HasBishopPair=['Yes']
AND CompensatingLessMaterial=['No']
THEN Class=Bad<19.000, 25.000, 3.000>

IF Imbalance=['Yes'] AND DoublePawnsPlayer=['Yes']
AND CompensatingLessMaterial=['No']
AND CompensatingMoreMaterial=['No']
THEN Class=Bad<27.000, 31.000, 9.000>

IF Imbalance=['Yes'] AND HasBishopPair=['Yes']
THEN Class=Bad<22.000, 45.000, 5.000>

IF Imbalance=['Yes'] AND HasQueen=['Yes']
AND CompensatingLessMaterial=['No']
AND CompensatingMoreMaterial=['No']
```

```
THEN Class=Bad<101.000, 176.000, 56.000>

IF HasBishopPair=['Yes'] AND Material=['Low']
AND Side=['WHITE'] THEN Class=Bad<27.000, 51.000, 12.000>

IF HasBishopPair=['Yes'] AND Material=['Low']
AND CompensatingLessMaterial=['No']
THEN Class=Bad<33.000, 67.000, 21.000>

IF Imbalance=['Yes'] AND HasQueen=['Yes']
AND CompensatingLessMaterial=['No']
THEN Class=Bad<106.000, 216.000, 69.000>

IF Imbalance=['Yes'] AND HasQueen=['Yes']
AND CompensatingMoreMaterial=['No'] AND PawnSymmetry=['No']
THEN Class=Bad<111.000, 245.000, 62.000>

IF Imbalance=['Yes'] AND CompensatingLessMaterial=['No']
AND CompensatingMoreMaterial=['No']
THEN Class=Bad<123.000, 271.000, 86.000>

IF Imbalance=['Yes'] AND CompensatingMoreMaterial=['No']
AND PawnSymmetry=['No']
THEN Class=Bad<135.000, 396.000, 96.000>

IF Material=['Low'] AND HasQueen=['Yes']
AND CompensatingLessMaterial=['No']
AND CompensatingMoreMaterial=['No']
THEN Class=Bad<237.000, 665.000, 228.000>

IF Material=['Low'] AND BlockedCenter=['Yes']
```

```
AND Side=['BLACK'] THEN Class=Good<0.000, 24.000, 1.000>

IF CompensatingLessMaterial=['Yes']
AND PawnSymmetry=['Yes']
THEN Class=Good<0.000, 57.000, 3.000>

IF Imbalance=['Yes'] AND BlockedCenter=['Yes']
THEN Class=Good<1.000, 13.000, 0.000>

IF Imbalance=['Yes'] AND PawnSymmetry=['Yes']
THEN Class=Good<2.000, 36.000, 1.000>

IF Material=['Low'] AND HasQueen=['Yes']
AND PawnSymmetry=['No']
THEN Class=Bad<264.000, 843.000, 270.000>

IF CompensatingLessMaterial=['Yes'] AND Imbalance=['Yes']
THEN Class=Good<14.000, 160.000, 11.000>

IF BlockedCenter=['Yes'] AND DoublePawnsOpponent=['Yes']
THEN Class=Good<0.000,35.000, 7.000>

IF Material=['Low'] AND BlockedCenter=['Yes']
AND PawnSymmetry=['No']
THEN Class=Good<5.000, 50.000, 6.000>

IF CompensatingLessMaterial=['Yes'] AND Material=['Low']
THEN Class=Good<39.000, 427.000, 72.000>

IF Imbalance=['Yes'] AND HasQueen=['No']
AND DoublePawnsPlayer=['No']
```

```
THEN Class=Good<15.000, 174.000, 38.000>

IF CompensatingLessMaterial=['Yes']
THEN Class=Good<64.000, 652.000, 139.000>

IF Imbalance=['Yes'] AND HasQueen=['No']
THEN Class=Good<25.000, 188.000, 41.000>

IF Material=['Low'] AND BlockedCenter=['Yes']
THEN Class=Good<9.000, 96.000, 27.000>

IF Material=['Low'] AND PawnSymmetry=['Yes']
THEN Class=Good<21.000, 198.000, 55.000>

IF BlockedCenter=['Yes'] AND PawnSymmetry=['No']
THEN Class=Good<14.000, 91.000, 23.000>

IF DoublePawnsPlayer=['No'] AND Imbalance=['Yes']
THEN Class=Good<113.000, 421.000, 106.000>

IF DoublePawnsPlayer=['No'] AND HasQueen=['No']
THEN Class=Good<165.000, 1033.000, 385.000>

IF HasQueen=['No']
THEN Class=Good<192.000, 1178.000, 438.000>

IF Material=['Low']
THEN Class=Good<471.000, 2134.000, 728.000>

IF BlockedCenter=['Yes']
THEN Class=Good<21.000, 211.000, 100.000>
```

```
IF DoublePawnsOpponent=['Yes']
THEN Class=Good<131.000, 615.000, 224.000>

IF PawnSymmetry=['No'] AND Material=['Low']
THEN Class=Good<450.000, 1936.000, 673.000>

IF PawnSymmetry=['Yes']
THEN Class=Good<65.000, 618.000, 296.000>

IF Side=['BLACK']
THEN Class=Good<390.000, 1849.000, 733.000>

IF TRUE THEN Class=Good<827.000, 3906.000, 1606.000>
```

A.2 Klubski igralci

Celoten seznam pravil zgeneriranih za skupino klubski igralci.

```
IF Imbalance=['Yes'] AND CompensatingLessMaterial=['No']
AND PawnSymmetry=['Yes']
THEN Class=Bad<6.000, 3.000, 3.000>

IF Imbalance=['Yes'] AND HasQueen=['Yes']
AND CompensatingLessMaterial=['No']
AND CompensatingMoreMaterial=['No']
THEN Class=Bad<52.000, 119.000, 28.000>

IF HasQueen=['Yes'] AND Material=['Low']
AND HasBishopPair=['Yes']
THEN Class=Bad<18.000, 39.000, 15.000>
```

```
IF Imbalance=['Yes'] AND HasQueen=['Yes']
AND CompensatingLessMaterial=['No']
THEN Class=Bad<56.000, 152.000, 36.000>

IF Imbalance=['Yes'] AND HasQueen=['Yes']
THEN Class=Bad<62.000, 219.000, 47.000>

IF CompensatingLessMaterial=['Yes']
AND PawnSymmetry=['Yes']
THEN Class=Good<0.000, 67.000, 3.000>

IF HasQueen=['No'] AND DoublePawnsPlayer=['No']
AND Side=['WHITE'] AND DoublePawnsOpponent=['Yes']
THEN Class=Good<2.000, 63.000, 5.000>

IF CompensatingLessMaterial=['Yes'] AND HasQueen=['No']
THEN Class=Good<4.000, 168.000, 15.000>

IF DoublePawnsOpponent=['Yes'] AND HasQueen=['No']
AND Side=['WHITE'] THEN Class=Good<4.000, 81.000, 13.000>

IF CompensatingLessMaterial=['Yes']
THEN Class=Good<38.000, 559.000, 85.000>

IF DoublePawnsPlayer=['Yes'] AND BlockedCenter=['Yes']
THEN Class=Good<0.000, 49.000, 11.000>

IF CompensatingMoreMaterial=['Yes'] AND Side=['WHITE']
THEN Class=Good<6.000, 121.000, 26.000>

IF CompensatingMoreMaterial=['Yes']
```

```
AND DoublePawnsPlayer=['No']
THEN Class=Good<12.000, 161.000, 35.000>

IF Side=['WHITE'] AND HasQueen=['No']
THEN Class=Good<39.000, 509.000, 125.000>

IF DoublePawnsPlayer=['No'] AND HasQueen=['No']
THEN Class=Good<72.000, 876.000, 217.000>

IF HasQueen=['No']
THEN Class=Good<79.000, 1022.000, 262.000>

IF CompensatingMoreMaterial=['Yes']
THEN Class=Good<19.000, 199.000, 51.000>

IF Material=['Low']
THEN Class=Good<266.000, 2025.000, 531.000>

IF PawnSymmetry=['Yes']
THEN Class=Good<36.000, 582.000, 196.000>

IF DoublePawnsPlayer=['Yes']
THEN Class=Good<64.000, 526.000, 161.000>

IF TRUE THEN Class=Good<478.000, 3735.000, 1141.000>
```

A.3 Fischer

Celoten seznam pravil zgeneriranih za Fischerja.

```
IF Side=['BLACK'] AND PawnSymmetry=['No']
THEN Class=Bad<91.000, 421.000, 52.000>
```

```
IF Imbalance=['Yes'] AND DoublePawnsPlayer=['Yes']
THEN Class=Good<0.000, 20.000, 0.000>

IF DoublePawnsPlayer=['Yes']
AND DoublePawnsOpponent=['Yes']
THEN Class=Good<0.000, 28.000, 2.000>

IF CompensatingLessMaterial=['Yes']
AND DoublePawnsOpponent=['No']
THEN Class=Good<4.000, 148.000, 14.000>

IF DoublePawnsPlayer=['Yes'] AND Side=['WHITE']
THEN Class=Good<4.000, 55.000, 4.000>

IF CompensatingLessMaterial=['Yes']
THEN Class=Good<14.000, 209.000, 21.000>

IF HasBishopPair=['Yes'] AND Side=['WHITE']
THEN Class=Good<11.000, 90.000, 5.000>

IF Imbalance=['Yes'] THEN Class=Good<21.000, 154.000, 8.000>

IF HasBishopPair=['Yes']
THEN Class=Good<24.000, 150.000, 17.000>

IF DoublePawnsPlayer=['Yes']
THEN Class=Good<18.000, 106.000, 11.000>

IF Side=['WHITE'] THEN Class=Good<67.000, 543.000, 97.000>
```

```
IF DoublePawnsOpponent=['No']
THEN Class=Good<132.000, 852.000, 133.000>

IF TRUE THEN Class=Good<161.000, 1017.000, 160.000>
```

A.4 Karpov

Celoten seznam pravil zgeneriranih za Karpova.

```
IF HasQueen=['Yes'] AND Material=['Low']
AND IsolatedQueensPawnOpponent=['No']
THEN Class=Bad<44.000, 192.000, 16.000>

IF Imbalance=['Yes'] THEN Class=Bad<16.000, 95.000, 6.000>

IF CompensatingLessMaterial=['Yes']
THEN Class=Good<13.000, 183.000, 11.000>

IF DoublePawnsPlayer=['Yes']
THEN Class=Good<8.000, 107.000, 15.000>

IF Material=['Low']
THEN Class=Good<79.000, 520.000, 75.000>

IF TRUE THEN Class=Good<130.000, 872.000, 163.000>
```

Dodatek B

Programska koda

Dodatek vsebuje programsko kodo za generiranje atributov in razred, za interpretacijo FEN-a, ki ga uporabljamo pri generiranju atributov.

B.1 Generiranje atributov

```
from fen import FEN
import Orange
from operator import itemgetter

class AttributeGenerator(object):
  QUEEN=9
  ROOK=5
  BISHOP=3
  KNIGHT=3

  def __init__(self, data):
    if data:
      self.data=data

  def generate_has_queen(self, data=None):
```

```
    #generate HasQueen
    if data:
      self.data=data

    def check_has_queen(inst, return_what):
      if inst["Side"]=="WHITE":
        queen="Q"
      else:
        queen="q"
      if queen in str(inst["FEN"]).split(" ")[0]:
        return has_queen("Yes")
      else:
        return has_queen("No")

    has_queen = Orange.feature.Discrete(
      "HasQueen", values=["No", "Yes"])
    has_queen.get_value_from = check_has_queen
    self.data = Orange.data.Table(Orange.data.Domain(
      self.data.domain, has_queen, False), self.data)
    return self.data

  def generate_has_bishops(self, data=None):
    #generate HasBishopPair
    if data:
      self.data=data

    def check_has_bishops(inst, return_what):
      if inst["Side"]=="WHITE":
        bishop="B"
        opp_bishop="b"
      else:
```

```
            bishop="b"
            opp_bishop="B"
          if str(inst["FEN"]).split(" ")[0].count(bishop)==2 \
              and str(inst["FEN"]).split(" ")[0].count(opp_bishop)<2:
            return has_bishops("Yes")
          else:
            return has_bishops("No")

        has_bishops=Orange.feature.Discrete("HasBishopPair",\
          values=["No", "Yes"])
        has_bishops.get_value_from=check_has_bishops
        self.data=Orange.data.Table(Orange.data.Domain(
          self.data.domain, has_bishops, False), self.data)
        return self.data

    def generate_game_phase(self, data=None):
        if data:
          self.data=data

        def get_game_phase(inst, return_what):
          f=str(inst["FEN"]).split(" ")[0]
          if int(inst["MoveNo"].value)<12:
            print int(inst["MoveNo"].value)
            raise Exception("aaaaaargh")
            return game_phase("Opening")
          elif ((f.count("Q")+f.count("q"))*self.QUEEN)+\
              ((f.count("R")+f.count("r"))*self.ROOK)+\
              ((f.count("B")+f.count("b"))*self.BISHOP)+\
              ((f.count("N")+f.count("n"))*self.KNIGHT)<15:
            return game_phase("EndGame")
          else:
```

```
            return game_phase("MiddleGame")
        game_phase=Orange.feature.Discrete("GamePhase",\
          values=["Opening", "MiddleGame", "EndGame"])
        game_phase.get_value_from=get_game_phase
        self.data=Orange.data.Table(Orange.data.Domain(
          self.data.domain, game_phase, False), self.data)
        return self.data

    def generate_relative_move_value(self, data=None):
        #in centipawns
        if data:
            self.data=data

        def calculate_relative_move_value(inst, return_what):
            try:
                val=abs(inst["MovePlayedValue"]-inst["BestMoveValue"])
            except:
                val=0
            if val>300:
                val=300
            return Orange.data.Value(relative_move_value, val)

        relative_move_value=Orange.feature.Continuous(
          "RelativeMoveValue", number_of_decimals=0)
        relative_move_value.get_value_from=
          calculate_relative_move_value
        self.data=Orange.data.Table(Orange.data.Domain(
          self.data.domain, relative_move_value, False), self.data)
        return self.data

    def generate_move_validity_by_value(self, data=None):
```

```
    if data:
      self.data=data

    def check_move_validity(inst, return_what):
      try:
        if (inst["MovePlayedValue"]<=-200 and\
            inst["BestMoveValue"]<=-200) or\
            (inst["MovePlayedValue"]>200 and\
            inst["BestMoveValue"]>200):
          return move_validity("Invalid")
        else:
          return move_validity("Valid")
      except:
        return move_validity("Invalid")

    move_validity=Orange.feature.Discrete("MoveValidity",\
      values=["Invalid", "Valid"])
    move_validity.get_value_from=check_move_validity
    self.data=Orange.data.Table(Orange.data.Domain(
      self.data.domain, move_validity, False), self.data)
    return self.data

  def generate_blocked_center(self, data=None):
    #generate BlockedCenter
    if data:
      self.data=data
    def get_blocked_center(inst, return_what):
      f=FEN(str(inst["FEN"]))
      try:
        if f.get_piece_by_position("e4")=="P" and\
            f.get_piece_by_position("d5")=="P" and\
```

```
                f.get_piece_by_position("e5")=="p" and\
                f.get_piece_by_position("d6")=="p":
            return blocked_center("Yes")
        elif f.get_piece_by_position("d4")=="P" and\
                f.get_piece_by_position("e5")=="P" and\
                f.get_piece_by_position("d5")=="p" and\
                f.get_piece_by_position("e6")=="p":
            return blocked_center("Yes")
        elif f.get_piece_by_position("e3")=="P" and\
                f.get_piece_by_position("d4")=="P" and\
                f.get_piece_by_position("e4")=="p" and\
                f.get_piece_by_position("d5")=="p":
            return blocked_center("Yes")
        elif f.get_piece_by_position("d3")=="P" and\
                f.get_piece_by_position("e4")=="P" and\
                f.get_piece_by_position("d4")=="p" and\
                f.get_piece_by_position("e5")=="p":
            return blocked_center("Yes")
        else:
            return blocked_center("No")
    except:
        return blocked_center("error")
    blocked_center=Orange.feature.Discrete("BlockedCenter",\
      values=["No", "Yes", "error"])
    blocked_center.get_value_from=get_blocked_center
    self.data=Orange.data.Table(Orange.data.Domain(
      self.data.domain, blocked_center, False), self.data)
    return self.data

def generate_isolated_queens_pawn(self, data=None):
    #generate IsolatedQueensPawnPlayer and
```

```
#IsolatedQueensPawnOpponent
if data:
  self.data=data

def get_isolated_queens_pawn_player(inst, return_what):
  f=FEN(str(inst["FEN"]))
  if f.get_piece_by_position("d4")=='P':
    if not any([f.get_piece_by_position(
        "d"+str(i))=='p' for i in range(1, 9)]):
      if not any([f.get_piece_by_position(
          "c"+str(i))=='P' for i in range(1, 9)]) and\
          not any([f.get_piece_by_position(
          "e"+str(i))=='P' for i in range(1, 9)]):
        if f.get_piece_by_position("c6")=='p' or\
            f.get_piece_by_position("e6")=='p':
          if str(inst["Side"])=="WHITE":
            return isolated_queens_pawn_player("Yes")
  elif f.get_piece_by_position("d5")=='p':
    if not any([f.get_piece_by_position(
        "d"+str(i))=='P' for i in range(1, 9)]):
      if not any([f.get_piece_by_position(
          "c"+str(i))=='p' for i in range(1, 9)]) and\
          not any([f.get_piece_by_position(
          "e"+str(i))=='p' for i in range(1, 9)]):
        if f.get_piece_by_position("c3")=='P' or\
            f.get_piece_by_position("e3")=='P':
          if str(inst["Side"])=="BLACK":
            return isolated_queens_pawn_player("Yes")
  return isolated_queens_pawn_player("No")
```

```
def get_isolated_queens_pawn_opponent(inst, return_what):
  f=FEN(str(inst["FEN"]))
  if f.get_piece_by_position("d4")=='P':
    if not any([f.get_piece_by_position(
        "d"+str(i))=='p' for i in range(1, 9)]):
      if not any([f.get_piece_by_position(
          "c"+str(i))=='P' for i in range(1, 9)]) and\
          not any([f.get_piece_by_position(
          "e"+str(i))=='P' for i in range(1, 9)]):
        if f.get_piece_by_position("c6")=='p' or\
            f.get_piece_by_position("e6")=='p':
          if str(inst["Side"])=="BLACK":
            return isolated_queens_pawn_opponent("Yes")
  elif f.get_piece_by_position("d5")=='p':
    if not any([f.get_piece_by_position(
        "d"+str(i))=='P' for i in range(1, 9)]):
      if not any([f.get_piece_by_position(
          "c"+str(i))=='p' for i in range(1, 9)]) and\
          not any([f.get_piece_by_position(
          "e"+str(i))=='p' for i in range(1, 9)]):
        if f.get_piece_by_position("c3")=='P' or\
         f.get_piece_by_position("e3")=='P':
          if str(inst["Side"])=="WHITE":
            return isolated_queens_pawn_opponent("Yes")
  return isolated_queens_pawn_opponent("No")


isolated_queens_pawn_player=Orange.feature.Discrete(
  "IsolatedQueensPawnPlayer", values=["No", "Yes"])
isolated_queens_pawn_player.get_value_from=
  get_isolated_queens_pawn_player
```

```
    isolated_queens_pawn_opponent=Orange.feature.Discrete(
      "IsolatedQueensPawnOpponent", values=["No", "Yes"])
    isolated_queens_pawn_opponent.get_value_from=
      get_isolated_queens_pawn_opponent

    self.data=Orange.data.Table(Orange.data.Domain(
      self.data.domain, isolated_queens_pawn_player, False),\
      self.data)
    self.data=Orange.data.Table(Orange.data.Domain(
      self.data.domain, isolated_queens_pawn_opponent, False),\
      self.data)
    return self.data

  def generate_double_pawns(self, data=None):
    #generate DoublePawnsPlayer and
    #DoublePawnsOpponent
    if data:
      self.data=data
    def get_double_pawns_player(inst, return_what):
      if FEN(str(inst["FEN"])).detect_double_pawns()\
          [inst["Side"].value]:
        return double_pawns_player("Yes")
      else:
        return double_pawns_player("No")

    def get_double_pawns_opponent(inst, return_what):
      if FEN(str(inst["FEN"])).detect_double_pawns()["BLACK"] and\
          inst["Side"].value=="WHITE":
        return double_pawns_opponent("Yes")
      elif FEN(str(inst["FEN"])).detect_double_pawns()["WHITE"] and\
          inst["Side"].value=="BLACK":
```

```
            return double_pawns_opponent("Yes")
        else:
            return double_pawns_opponent("No")

    double_pawns_player=Orange.feature.Discrete(
        "DoublePawnsPlayer", values=["No", "Yes"])
    double_pawns_player.get_value_from=get_double_pawns_player

    double_pawns_opponent=Orange.feature.Discrete(
        "DoublePawnsOpponent", values=["No", "Yes"])
    double_pawns_opponent.get_value_from=get_double_pawns_opponent

    self.data=Orange.data.Table(Orange.data.Domain(
        self.data.domain, double_pawns_player, False), self.data)
    self.data=Orange.data.Table(Orange.data.Domain(
        self.data.domain, double_pawns_opponent, False), self.data)
    return self.data

def generate_imbalance(self, data=None):
    #generate Imbalance
    if data:
        self.data=data
    def get_imbalance(inst, return_what):
        if FEN(str(inst["FEN"])).measure_imbalance()<5:
            return imbalance("No")
        else:
            return imbalance("Yes")

    imbalance=Orange.feature.Discrete(
        "Imbalance", values=["No", "Yes"])
    imbalance.get_value_from=get_imbalance
```

```
    self.data=Orange.data.Table(Orange.data.Domain(
      self.data.domain, imbalance, False), self.data)
    return self.data

  def generate_material(self, data=None):
    #generate Material
    if data:
      self.data=data
    def get_material(inst, return_what):
      if FEN(str(inst["FEN"])).measure_material(
          inst["Side"].value)>23:
        return material("High")
      else:
        return material("Low")

    material=Orange.feature.Discrete(
      "Material", values=["High", "Low"])
    material.get_value_from=get_material
    self.data=Orange.data.Table(Orange.data.Domain(
      self.data.domain, material, False), self.data)

    return self.data

  def generate_pawn_symmetry(self, data=None):
    #generate PawnSymmetry
    if data:
      self.data=data
    def get_pawn_symmetry(inst, return_what):
      if FEN(str(inst["FEN"])).symmetrical_pawns():
        return pawn_symmetry("Yes")
      return pawn_symmetry("No")
```

```
    pawn_symmetry=Orange.feature.Discrete(
      "PawnSymmetry", values=["No", "Yes"])
    pawn_symmetry.get_value_from=get_pawn_symmetry
    self.data=Orange.data.Table(Orange.data.Domain(
      self.data.domain, pawn_symmetry, False), self.data)

    return self.data

  def generate_compensation(self, data=None):
    #generate CompensatingLessMaterial and
    #CompensatingMoreMaterial
    if data:
      self.data=data
    def get_compensating_more_material(inst, return_what):
      f=FEN(str(inst["FEN"]))
      if inst["BestMoveValue"].value<50 and\
          inst["BestMoveValue"].value>-50:
        if inst["Side"].value=="WHITE" and\
            f.measure_material("WHITE", pawns=True)>\
            f.measure_material("BLACK", pawns=True):
          return compensating_more_material("Yes")
        if inst["Side"].value=="BLACK" and\
            f.measure_material("BLACK", pawns=True)>\
            f.measure_material("WHITE", pawns=True):
          return compensating_more_material("Yes")
      return compensating_more_material("No")

    def get_compensating_less_material(inst, return_what):
      f=FEN(str(inst["FEN"]))
      if inst["BestMoveValue"].value<50 and\
```

```
            inst["BestMoveValue"].value>-50:
        if inst["Side"].value=="WHITE" and\
            f.measure_material("WHITE", pawns=True)<\
            f.measure_material("BLACK", pawns=True):
          return compensating_less_material("Yes")
        if inst["Side"].value=="BLACK" and\
            f.measure_material("BLACK", pawns=True)<\
            f.measure_material("WHITE", pawns=True):
          return compensating_less_material("Yes")
      return compensating_less_material("No")


compensating_more_material=Orange.feature.Discrete(
  "CompensatingMoreMaterial", values=["No", "Yes"])
compensating_more_material.get_value_from=
  get_compensating_more_material

compensating_less_material=Orange.feature.Discrete(
  "CompensatingLessMaterial", values=["No", "Yes"])
compensating_less_material.get_value_from=
  get_compensating_less_material

self.data=Orange.data.Table(Orange.data.Domain(
  self.data.domain, compensating_more_material,\
  False), self.data)
self.data=Orange.data.Table(Orange.data.Domain(
  self.data.domain, compensating_less_material,\
  False), self.data)

return self.data
```

```
    """remove data instances"""

    def remove_by_game_phase(self,\
        phase_to_keep="MiddleGame", data=None):
      if data:
        self.data=data
      if not "GamePhase" in self.data.domain:
        self.generate_game_phase()
        remove_after=True
      else:
        remove_after=False

      game_phase_filter=Orange.data.filter.SameValue()
      game_phase=self.data.domain["GamePhase"]
      game_phase_filter.value=Orange.data.Value(
        game_phase, phase_to_keep)
      game_phase_filter.position=self.data.domain.index(
        "GamePhase")

      if remove_after:
        self.data=Orange.data.Table(Orange.data.Domain(
          [d for d in self.data.domain if not d.name=="GamePhase"],\
          False), game_phase_filter(self.data))
      else:
        self.data=game_phase_filter(self.data)
      return self.data

    def remove_invalid_moves(self, data=None):
      if data:
        self.data=data
      if not "MoveValidity" in self.data.domain:
```

```
        self.generate_move_validity_by_value()
        remove_after=True
    else:
        remove_after=False

    move_validity_filter=Orange.data.filter.SameValue()
    move_validity=self.data.domain["MoveValidity"]
    move_validity_filter.value=Orange.data.Value(
        move_validity, "Valid")
    move_validity_filter.position=self.data.domain.index(
        "MoveValidity")

    if remove_after:
        self.data=Orange.data.Table(Orange.data.Domain(
            [d for d in self.data.domain if not d.name=="MoveValidity"],\
            False), move_validity_filter(self.data))
    else:
        self.data=move_validity_filter(self.data)
    return self.data

"""generate class variable"""

def generate_class_custom_3(self, limit_good, limit_bad, data=None):
    """limit[0 - 300] in centipawns"""
    #Fischer, Karpov: limit_good=5, limit_bad=15
    #amaterji, klubski igralci: limit_good=10, limit_bad=50
    if data:
        self.data=data

    if not "RelativeMoveValue" in self.data.domain:
        self.generate_relative_move_value()
```

```
    remove_after=True
  else:
    remove_after=False

  def get_class_custom_3(inst, return_what):
    if inst["RelativeMoveValue"]>limit_bad:
      return class_variable("Bad")
    elif inst["RelativeMoveValue"]<limit_good:
      return class_variable("Good")
    else:
      return class_variable("Dubious")

  class_variable=Orange.feature.Discrete(
    "Class", values=["Bad", "Good", "Dubious"])
  class_variable.get_value_from=get_class_custom_3
  if remove_after:
    self.data=Orange.data.Table(Orange.data.Domain(
      [d for d in self.data.domain if not\
      d.name=="RelativeMoveValue"], class_variable, True),\
      self.data)
  else:
    self.data=Orange.data.Table(Orange.data.Domain(
      self.data.domain, class_variable, True), self.data)
  return self.data
```

B.2 FEN

```
class FEN(object):
  a=0
  b=1
  c=2
  d=3
```

```
e=4
f=5
g=6
h=7
fen_str=""
fen_table=[['R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R'],
        ['P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'],
        ['.', '.', '.', '.', '.', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.', '.', '.'],
        ['p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'],
        ['r', 'n', 'b', 'q', 'k', 'b', 'n', 'r']]
other_things=['w', 'KQkq', '-', '0', '1']

piece_value={'q':9, 'Q':9, 'r':5, 'R':5, 'b':3, 'B':3,
  'n':3, 'N':3, 'p':1, 'P':1, 'k':0, 'K':0}

def __init__(self, fen_string):
  if fen_string:
    self.fen_str=fen_string
    fen_split=fen_string.split(" ")
    self.other_things=fen_split[1:]
    out_counter=0
    for line in reversed(fen_split[0].split("/")):
      in_counter=0
      for char in line:
        try:
          char_int=int(char)
          for i in range(char_int):
            self.fen_table[out_counter][in_counter+i]='.'
```

```
            in_counter+=char_int
          except:
            self.fen_table[out_counter][in_counter]=char
            in_counter+=1
        out_counter+=1

  def get_piece_by_position(self, position):
    try:
      return self.fen_table[int(position[1])-1]\
        [eval("self."+position[0])]
    except:
      raise Exception("Wrong position for FEN.\
        get_piece_by_position(position). position="+position)

  def __str__(self):
    return "\n".join([" ".join(t_line) for\
      t_line in reversed(self.fen_table)])

  def get_move(self, position):
    try:
      piece=self.get_piece_by_position(position)
      if piece=='K' or piece=='k':
        if abs(eval("self."+position[0])-\
            eval("self."+position[2]))>1:
          return "CASTLING"
      return piece
    except:
      raise Exception("Wrong position for \
        FEN.get_move(position). position="+position)

  def detect_double_pawns(self):
```

```
    """
    Returns [black, white] <- bool array
    """
    black=False
    white=False
    for pawn in ['p', 'P']:
      for i in range(8):
        one=False
        for line in self.fen_table:
          if line[i]==pawn:
            if one:
              if pawn=='p':
                black=True
              else:
                white=True
            else:
              one=True
    return {"BLACK":black, "WHITE":white}

  def measure_imbalance(self):
    black=[]
    white=[]
    for line in self.fen_table:
      for f in line:
        if f!='.':
          if f.isupper():
            white.append(self.piece_value[f])
          else:
            black.append(self.piece_value[f])
    sum_list=[]
    for b in black:
```

```
      if b in white:
        white[white.index(b)]=0
      else:
        sum_list.append(b)
    return sum(sum_list)+sum(white)

  def measure_material(self, side, pawns=False):
    pieces={"WHITE":['Q', 'R', 'B', 'N'],
      "BLACK":['q', 'r', 'b', 'n']}
    if pawns:
      pieces["WHITE"].append('P')
      pieces["BLACK"].append('p')
    fs=self.fen_str.split(" ")[0]
    m=0
    for p in pieces[side]:
      m+=fs.count(p)*self.piece_value[p]
    return m

  def symmetrical_pawns(self):
    pawn='pP'
    for i in range(8):
      pawn_count={'p':0, 'P':0}
      for line in self.fen_table:
        if line[i] in pawn:
          pawn_count[line[i]]+=1
      if pawn_count['p']!=pawn_count['P']:
        return False
    return True
```