branching-factor-and-depth

September 1, 2020

[1]:
```
import pandas as pd
```

1 Branching Factor und Depth für "Chamäleon Schach"

Dieses Dokument versucht die Komplexität für das Spiel "Chamäleon Schach" zu ermitteln. Im Ordner `scripts` gibt es eine Datei namens `branching-factor-and-depth.ts`. Dieses Script ist bereits ausgeführt worden. Es hat sehr viele Spiele mit dem Sieger-Algorithmus gespielt. Zusätzlich ist mit unterschiedlichen Wahrscheinlichkeiten ein Zufallszug ausgeführt worden.

Alle Ergebnisse von diesen Spielen sind in eine CSV-Datei im Ordner `data` gespeichert worden. Diese Datei wird nun analysiert:

[2]:
```
data = pd.read_csv('../data/branching-factor-and-depth.csv')
data
```

[2]:

	numOfPlayers	chanceOfRandom	branchingFactorAvg	branchingFactorMedian	depth
0	2	0.01	15.090909	15.0	22
1	2	0.01	22.000000	24.0	27
2	2	0.01	17.961538	16.0	26
3	2	0.01	16.294118	15.5	34
4	2	0.01	20.074074	19.0	27
…	…	…	…	…	…
1495	4	1.00	21.227273	22.0	110
1496	4	1.00	16.811111	17.0	
1497	4	1.00	15.100000	15.0	
1498	4	1.00	18.400000	17.0	
1499	4	1.00	14.131579	13.0	

```
1496     90
1497    100
1498    140
1499     76

[1500 rows x 5 columns]
```

Insegsamt sind 1500 Spiele gespielt worden. Zu jedem Spiel ist angegeben:

- `numOfPlayers`: wieviele Spieler mitgespielt haben
- `chanceOfRandom`: die Wahrscheinlichkeit für eine Zufallszug, statt für einen Zug des Algorithmus
- `branchingFactorAvg`: der durschnittliche Branching Factor des Spieles
- `branchingFactorMedian`: der Median des Branching Factors des Spieles
- `depth`: und die Länge des Spieles, also wieviele Züge dieses Spiel gedauert hat.

[3]:
```
data.describe()
```

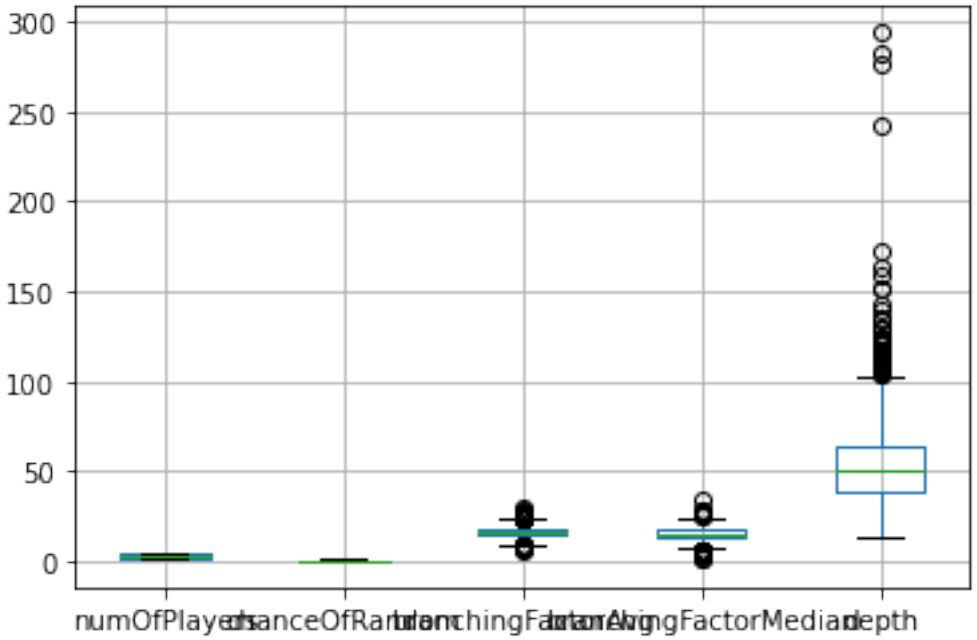
[3]:
```
       numOfPlayers  chanceOfRandom  branchingFactorAvg  \
count   1500.000000     1500.000000         1500.000000
mean       3.000000        0.362000           16.261504
std        0.816769        0.359252            2.965053
min        2.000000        0.010000            5.872093
25%        2.000000        0.100000           14.428571
50%        3.000000        0.200000           16.000000
75%        4.000000        0.500000           17.902439
max        4.000000        1.000000           30.370370

       branchingFactorMedian        depth
count            1500.000000  1500.000000
mean               15.050333    54.164667
std                 3.809021    25.493401
min                 2.000000    13.000000
25%                13.000000    38.000000
50%                14.500000    51.000000
75%                17.500000    64.000000
max                33.500000   294.000000
```

Es ist aufällig, dass die Werte über ein sehr breites Spektrum gestreut sind. Die meisten Spiele dauern um die 54 Züge, aber es gibt einen Ausreißer mit 294 Zügen. Der durchschnittliche Branching Factor liegt bei 16. Im Median liegt er etwas niedriger.

Als nächstes wird ein Boxplot gezeichnet, um die Verteilung der Werte besser visualisieren zu können

[4]:
```
data.boxplot()
```

[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b26b8c250>
```



Offenbar gibt es in der Depth sehr viele hohe Ausreißer. Diese sollen nun genauer betrachtet werden:

[5]:
```
outliers = data[data.depth > 100]
outliers.describe()
```

[5]:

	numOfPlayers	chanceOfRandom	branchingFactorAvg	\
count	65.000000	65.000000	65.000000	
mean	3.707692	0.782615	14.357015	
std	0.605265	0.400567	3.734314	
min	2.000000	0.010000	5.872093	
25%	4.000000	1.000000	11.935780	
50%	4.000000	1.000000	15.108527	
75%	4.000000	1.000000	17.000000	
max	4.000000	1.000000	21.227273	

	branchingFactorMedian	depth
count	65.000000	65.000000
mean	13.076923	126.846154
std	4.608273	41.408872
min	2.000000	101.000000
25%	11.500000	105.000000
50%	14.000000	114.000000
75%	16.500000	125.000000
max	22.000000	294.000000

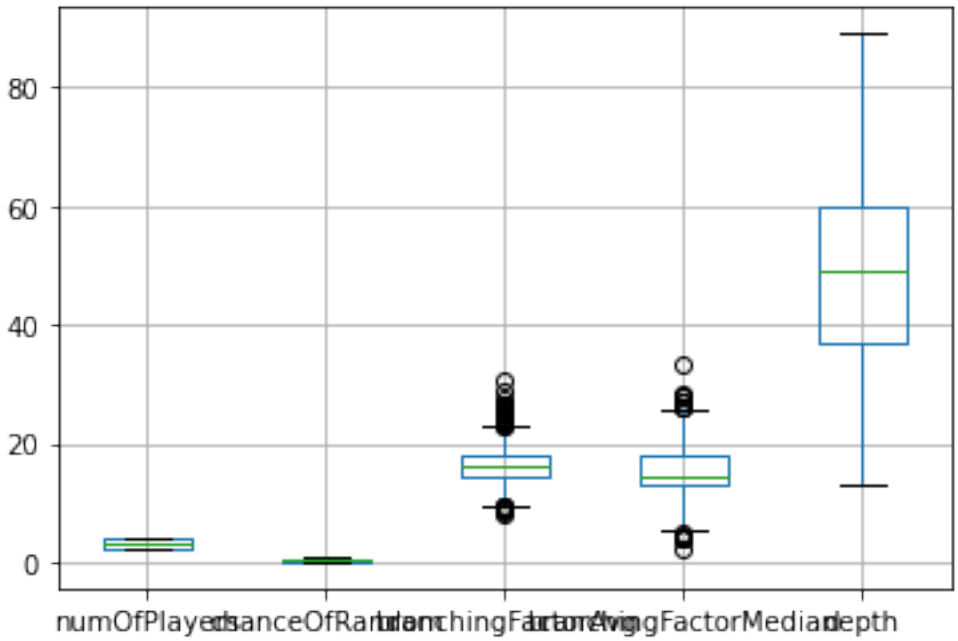
Die Ausreißer treten vor allem bei Spielen mit vier Spielern auf und besonders bei Spielen mit

100% Zufallszügen. Das macht Sinn, da nur zufällig gezogen wird und keine aktiven Schritte unternommen werden, um einen Sieg zu erringen. Die Werte drücken außerdem den Branching Factor nach unten, da der Durchschnitt hier bei 14 liegt.

Daher werden diese Ausreißer nun im Folgenden entfernt.

[6]:
```
cleaned = data[data.depth < 90]
cleaned.boxplot()
```

[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b26c01990>
```



Mit der Beseitigung der Ausreißer sieht der Boxplot wieder deutlich aufgeräumter aus. Trotzdem werden die Werte nun immer getrennt nach der Anzahl der Spieler und nach der Wahrscheinlichkeit für Zufallszüge betrachtet.

[7]:
```
cleaned.groupby(['numOfPlayers','chanceOfRandom']).mean().round()
```

[7]:

numOfPlayers	chanceOfRandom	branchingFactorAvg	branchingFactorMedian	depth
2	0.01	20.0	18.0	28.0
	0.10	17.0	16.0	31.0
	0.20	17.0	15.0	29.0
	0.50	16.0	14.0	34.0
	1.00	17.0	16.0	58.0
3	0.01	17.0	18.0	44.0
	0.10	16.0	15.0	50.0
	0.20	16.0	15.0	52.0

	0.50	16.0	15.0	53.0
	1.00	16.0	15.0	73.0
4	0.01	16.0	14.0	56.0
	0.10	16.0	14.0	56.0
	0.20	15.0	14.0	57.0
	0.50	16.0	14.0	63.0
	1.00	16.0	15.0	80.0

Die obige Tabelle zeigt die durchschnittlichen Werte für die Spiele. Sie sind jeweils gruppiert worden nach der Anzahl der Spieler sowie nach der Wahrscheinlichkeit für einen Zufallszug.

Der Branching Factor ist in fast allen Versionen in etwa 16. Vor allem bei zwei Spielern geht die Tendenz eher zu 17. Der Median liegt aber stets drunter. Daher wird der Branching Factor für "Chamäleon Schach" nun pauschal mit 16 gewichtet.

Bei der Depth ist eine Unterscheidnug nach der Anzahl der Spieler sinnvoll. Da mit mehr Spielern auch mehr Figuren auf dem Brett sind, ist klar, dass die Spiellänge entsprechend unterschiedlich ausfallen muss. Am meisten Gewicht wird dabei auf die Spiele mit einer recht geringen Wahrscheinlichkeit für einen Zufallszug gelegt. Diese Spiele sind ja deutlich "zielorientierter".

Für zwei Spieler wird die Depth mit 30 angesetzt. Drei Spieler brauchen um die 50 Züge und vier Spieler ungefähr 56.

Mit diesen Werten ist nun die Komplexität von "Chämleon Schach" je nach Anzahl der Spieler ermittelt worden:

[8]:
```
def row(name, b, d):
    return [name, b, d, '{:e}'.format(b ** d) ]

rows = [
    row('2 Players', 16, 30),
    row('3 Players', 16, 50),
    row('4 Players', 16, 56),
]
pd.DataFrame(rows, columns=['players','branching factor','depth','complexity'])
```

[8]:

	players	branching factor	depth	complexity
0	2 Players	16	30	1.329228e+36
1	3 Players	16	50	1.606938e+60
2	4 Players	16	56	2.695995e+67

Damit ist die Komplexität gefunden. Sie reicht von 10^{36} bei zwei Spielern über 10^{60} bei drei bis hin zu 10^{67} bei vier Spielern. Bei allen drei Versionen handelt es sich um sehr hohe Komplexitäten.