Predicting kills in Game of Thrones using network properties

Jaka Stavanja, Matej Klemen

The presentation

1. Introduction to the problem and related work
2. Data and methods used
3. Results
4. Discussion

What we were working on

- **Data**: kills for the Game of Thrones series ranging from season 1-6.
- **Challenge**: predict <u>who kills whom</u> using:
  - Link prediction
  - Standard machine learning approaches (with added features, computed on networks)

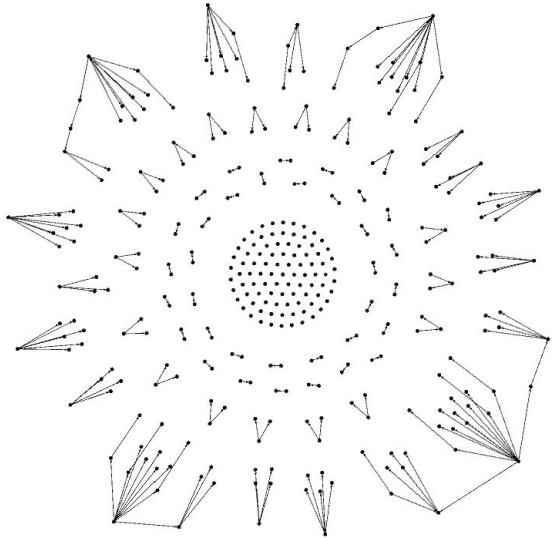
	Character	Killed by	Time	House	Status	Importance	Episode ID
0	White Rat	a member of the Sons of the Harpy	15:20	undefined	undefined	1	41
1	Mance Rayder	Jon Snow	49:18	undefined	undefined	2	41
2	Son of Harpy	Mossador	46:30	undefined	undefined	1	42
3	Mossador	Daario Naharis	50:24	undefined	undefined	2	42
4	Janos Slynt	Jon Snow	38:58	undefined	undefined	2	43

What others were working on

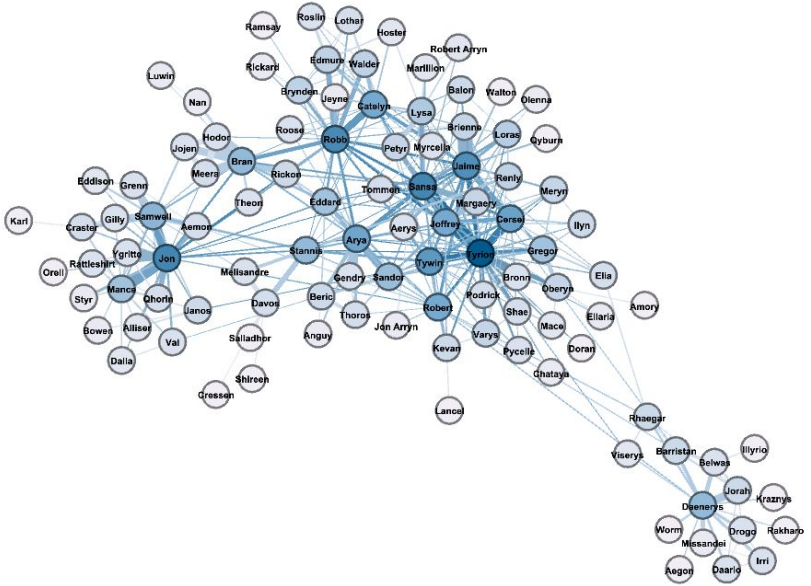
- Best strategic positions for characters
- Who is more likely to die (not who kills whom)
- Using standard machine learning or statistical approaches (no network science)

2 networks

- Collection of kills for seasons 1-6
  - Kill network
  - nodes = characters, directed links = kills

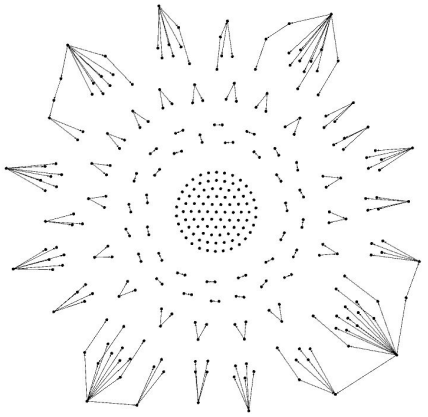


- Social network of characters
  - link if two characters appear within a 15 word span in the books



Link prediction approach

- Using kills network
- Link prediction:
  - using episode up to certain number to train and all later episodes to predict (start with episode 30)
  - Predicting one episode at a time



Indices

- Indices:
  - Preferential attachment $s_{ij} = k_i k_j$
  - Adamic-Adar $s_{ij} = \sum_{x \in \Gamma_i \cap \Gamma_j} \frac{1}{\log k_x}$
  - Community (Leiden modularity optimization)

$$s_{ij} = \left\{ \begin{array}{ll} m_i / \binom{n_i}{2}, & \text{where } c_i = c_j \\ m_{i,j} / m_i * m_j, & \text{where } c_i \neq c_j \end{array} \right\}$$

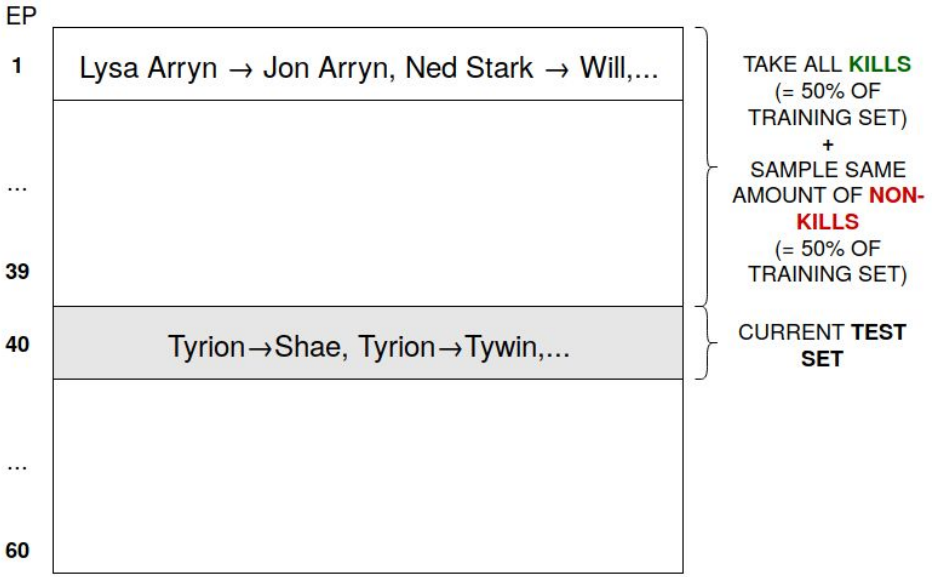
  - Alive index (custom)
    - If i and j alive 1 else 0

Standard ML approach

- K Nearest-Neighbors
- Logistic Regression
- SVM

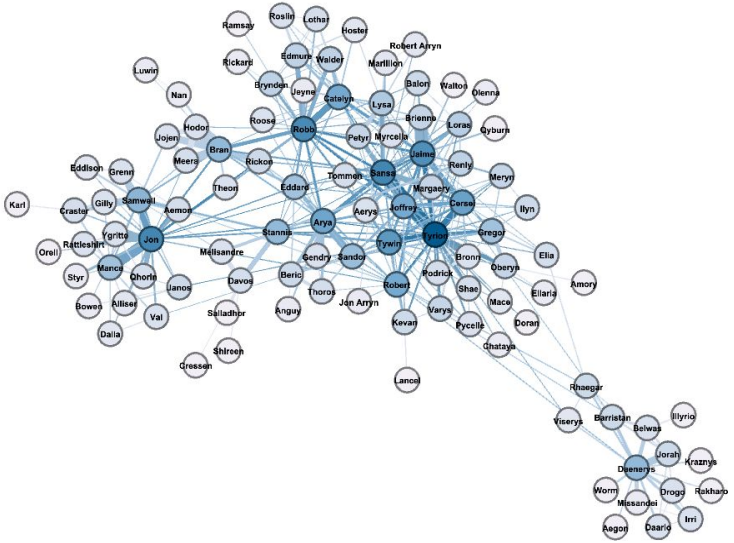
- Using original kills dataset with new features, computed on the network

Sampling:



Network feature extraction

- Using the social network of characters
- PageRank
- Betweenness
- Find communities for each character



Results (link prediction)

Method	AUC	Precision	Recall
Preferential attachment	0.503 (0.02)	0.522 (0.113)	0.087 (0.000)
Adamic-Adar	0.500 (0.000)	0.000 (0.000)	0.000 (0.000)
Community	0.500 (0.000)	0.000 (0.000)	0.000 (0.000)
Alive index	0.863 (0.032)	0.822 (0.020)	0.930 (0.000)

Results (Standard ML approaches)

Basic dataset only:

Method	AUC	Precision	Recall
KNN	0.632 (0.048)	0.641 (0.024)	0.453 (0.020)
Logistic Regression	0.596 (0.036)	0.797 (0.010)	0.400 (0.013)
SVM	0.556 (0.066)	0.688 (0.024)	0.523 (0.007)

Results (Standard ML approaches)

Added PageRank, betweenness and community IDs:

Method	AUC	Precision	Recall
KNN	0.659 (0.035)	0.725 (0.016)	0.453 (0.004)
Logistic Regression	0.658 (0.033)	0.816 (0.016)	0.418 (0.013)
SVM	0.686 (0.058)	0.719 (0.058)	0.456 (0.056)

Conclusion

- Usual link prediction approaches perform poorly because of the way the edges are created
- The alive index works, but only because of how our network is constructed
- Nothing much better than baseline

Conclusion

- Standard machine learning approaches perform well but not well enough that we could actually get somewhat accurate predictions
- Network features definitely help but not as much as we would hope

If anyone is curious ...

- https://github.com/matejklemen/got-link-prediction