Predicting kills in Game of Thrones using network properties

Jaka Stavanja[a, b] **and Matej Klemen**[a, b]

[a]University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia; [b]{js8927|mk3141}@student.uni-lj.si

TV series such as HBO's most popular show Game of Thrones have seen a high number of dedicated followers, who watch and thoroughly analyze every minute of the show. Largely discussed aspect of the show between viewers seems to be the dramatic murders of the most important characters, the thing that the series is most known for. In our work, we try to predict characters' kills (killer and victim pairs) using data about previous kills by the characters and additional metadata. We construct a network with characters as nodes, where two nodes are linked if one killed the other. Then we use a link prediction framework and evaluate different techniques to predict the next possible kills. Lastly, we construct features from various network properties on a social network of characters, which we use in conjunction with classic data mining techniques. We see that due to the small size of the kills dataset and the somewhat random distribution of kills, we cannot predict much with standard indices. However, we show that we can construct an index which is very customized for the exact network we create for Game of Thrones but might not work on other series. We also see that the features we compute on the social network of characters help with standard machine learning approaches as well but do not yield as accurate predictions as we would hope. The best results overall are achieved by using a custom index for link prediction, which is fitting for our type of network and gives us an Area Under the ROC Curve (AUC) of 0.863.

game of thrones | link prediction | network analysis | feature extraction

1. Introduction

With the ever increasing popularity of the TV series Game of Thrones, coming mostly from it's incredible plot twists and deaths of main characters, the question arises whether we can predict those deaths from a network analysis point of view. If we are able to predict the deaths from the data, collected from previous episodes, that means that the author is very predictable, which might not be the best thing in terms of the show being entertaining. To predict the deaths, we construct a network (illustrated in Figure 1), where nodes are characters in the show (along with other entities, that are able to kill another character such as a horse or a dragon) and connect two if one has murdered the other. We then try to predict whether a certain link between two nodes in the network happened (removing the link from the network beforehand). Because we are dealing with temporal data, we also remove links that appear in the network after the link currently being predicted. We use different approaches to assign scores to the pairs, for example different link prediction indices such as the preferential attachment index (1) or the Adamic-Adar index (2). We also construct additional features based on various network properties and additional metadata and see how this influences the results. The network properties for the features used in machine learning approaches are then computed on a different, social network of the Game of Thrones
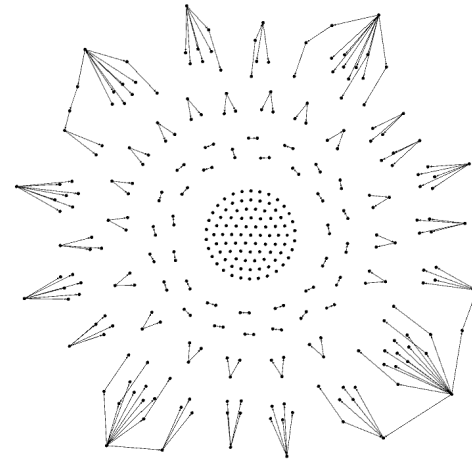

Fig. 1. Game of Thrones kills network.

characters, that are connected if they appear somewhat close in the original books' story.

The rest of the paper is structured as follows. In section 2 we provide an overview of existing literature on our studied topic. In section 3 we describe the methods that we use in our work. In section 4 we provide the results of our work, which we then discuss in section 5. In section 6 we summarize the work that was done and provide some possible future improvements.

2. Related work

For Game of Thrones specifically, some research has already been done in terms of finding which aspects of the show resonate with the viewer count the most and how real the characters' interactions are, done by modeling the show's houses with a network and exploring structural balance (3). Further studies determined who has the best strategic position in the show's world (4) etc., but the only article touching on death prediction studies (5) for the show used Cox's proportional hazard model (6), which didn't explore any network structure properties but rather used a regression approach (mostly used in medicine) to determine which factors introduce a higher risk of mortality through time. But the killers were not taken into consideration here, we just know how likely a character is to die, thus we cannot really compare this approach with ours. This gives us a unique opportunity to try and use network properties along with metadata to better predict kills in the mythical world of the series, but instead of only predicting
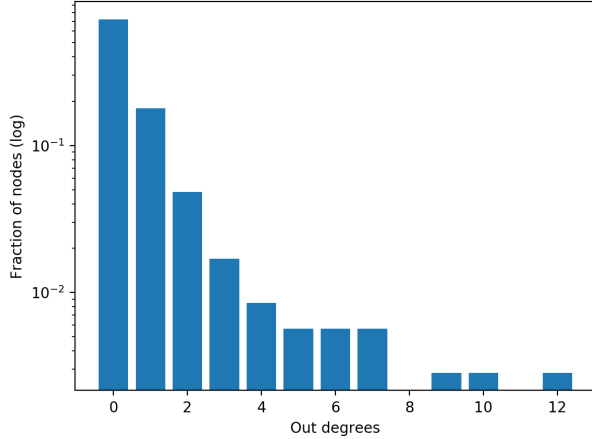
Fig. 2. Out-degree distribution for the kills network.

how likely someone is to die, we can predict both the killers and victims, on which there has not yet been much research.

3. Methods and network properties

In this section we describe the framework used to test our link prediction methods. We also provide descriptions of some classic link prediction methods based on well-known properties of networks. Then, we introduce a new social network, described in subsection B, which connects characters if they appear close in the original story from the books. We compute various network properties from that network and use them as additional features in the more traditional machine learing approach for classification.

A. Link prediction using the kills network. The network of kills is constructed of directed links between nodes u and v, where node u represents a character that killed the character represented by node v. The network is very small and also very sparse. It has 353 nodes and 194 links. By looking at a visualization of the network in Figure 1, we can see that most kills appear outside the somewhat bigger connected components and do not seem to be attributed to hubs. We can still observe a few hubs and connected components around them, where a lot of kills seem to occur, but in the majority of cases there are just two nodes involved in the kill, which on their own form a small two-node connected component. That gives us a clue as to which methods might predict kills better than others. We can construct an index that predicts links based on nodes' out degrees. Since the in-degrees in the networks could be either zero or one (meaning alive or dead) we cannot get any additional information from that besides whether a person can still kill someone or not (when they are already dead). The out-degree distribution can potentially be of use, since people that kill a lot of people might also tend to kill more people, and those who never killed anyone might not be inclined to murder. A plot of the out-degree distribution (using a logarithmic scale for the fractions of nodes) is shown on Figure 2.

In this subsection, we propose an evaluation framework and different indices that we use for the link prediction.

A.1. Evaluation framework for link prediction. To test how well our link prediction techniques work on our network, we construct a framework and provide a brief description of it here. It takes our prediction index function and the network as the input and outputs an Area Under the ROC Curve (AUC) value. The prediction index function assigns a score s_{ij} to every link between two nodes i and j that is being tested. A high score implies a high likelihood that the link exists in the network and a low score implies a small likelihood of the link's existence.

The core of the testing framework is the logic, which removes links from the network and then tries to predict how likely the links we have removed are to form as the network evolves through time. To use as much information as we can, we choose an episode and remove links from that episode (e.g. episode 30) and onwards from the graph. Then, we predict the links (kills) at that time using the information about murders from the previous episodes. We can then clone the original graph, remove links from the next episode in the chronology (e.g. 31) and predict links for that episode using the information from all the episodes before (including information from episode 30 in this example). By predicting links in this way, we are not using the data from the future to predict past links and we are using a lot more information than if we were to remove links from a certain episode onwards and just try to predict links for multiple episodes in one single iteration.

The idea of the framework's core implementation is as follows:

1. Iterate through every episode in the range from 30 to 60, and at each step do:

   (a) $L_P \leftarrow$ remove node links $((i, j) \in L)$ after current episode.

   (b) Compute s_{ij} for $(i, j) \in L_P$ at the current episode time.

   (c) Save the scores to S_P.

2. $L_N \leftarrow$ randomly sample $|S_p|$ unlinked nodes $((i, j) \notin L)$.

3. Compute s_{ij} for $(i, j) \in L_N$ and save scores to S_N.

4. Compute AUC by comparing the scores assigned to positive samples from S_P and scores assigned to negative samples S_N. When comparing scores, we can assign to b the amount of times when the score from S_P is bigger than the one from S_N, and assign to e the number of times when the two scores are equal. Then, we can compute the AUC as $\frac{b+e/2}{|L_P|}$.

A.2. Alive index. We create a type of a baseline index by looking at the network's high level properties. We check if the killer has an in-degree of zero and the target has an in-degree of zero (i.e. killer is alive and target has not been killed yet). If the endpoints of a link satisfy these conditions, the value of the index is 1, otherwise it is 0.

A.3. Preferential attachment index. Real world networks tend to have a scale-free degree distribution due to a phenomenon known as preferential attachment. The preferential attachment index (1) is defined as $s_{ij} = k_i k_j$, where k_i is the degree of node i. For our problem, we will use out-degrees only, as only they provide useful information. The idea behind the index is in the preferential attachment process - nodes are more likely

to connect with nodes that have a high degree, thus a link between two nodes with high degrees should be assigned a high score s_{ij}. We modify this definition slightly and define a modified preferential attachment index as $s_{ij} = k_i^{(out)} k_j^{(out)}$, where $k_i^{(out)}$ is the out-degree of node i. The logic behind that is that the more kills one has to their record, the more they are inclined to murder and vice-versa. We also make this modification because a node having an in-degree larger than 0 means that the character represented by the node is already dead and then this index does not only rely on the preferential attachment process. If this method gives a very bad score, we can invert the classifier to predict links between lower degree nodes with a higher probability, which would make sense if the authors were trying to make the killers less predictable.

A.4. Adamic-Adar similarity index. In real world networks, links tend to appear between nodes that have a lot of common neighbors. But due to preferential attachment, nodes tend to connect to higher degree nodes more likely, thus making that neighbor less useful for predicting new links between two nodes. The Adamic-Adar similarity index (2) takes the high degree neighbors into account and is defined as $s_{ij} = \sum_{x \in \Gamma_i \cap \Gamma_j} \frac{1}{\log k_x}$, where Γ_i is the neighborhood of node i, Γ_j is the neighborhood of the second node j and k_x is the degree of node x.

A.5. Community index. New links in e.g. social networks tend to appear between members that are inside of the same community and only rarely between members of different communities (7). We can find densely linked communities where people are killing each other the most and treat the new links in those communities as more likely to occur than the ones outside communities. So that we do not ignore links between communities, we can also count the amount of links that occur between two communities and model the inter-community densities. Let $\{C\}$ be the set of communities output by the Leiden modularity optimization algorithm (8), and c_i the community of node i. Then,

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

where n_c is the number of nodes in the community, m_i the number of links within community i in question and $m_{i,j}$ is the number of links between communities i and j.

B. Feature extraction for machine learning approaches.
Besides using the classic index-based link prediction techniques, we also make use of a machine-learning-based approach, in which we construct features from network properties and additional metadata and use them to train a classifier. The data used for training the classifier includes properties of the kills network such as PageRank scores for all characters and the basic kills from the original dataset. Computing a score like PageRank (9) on the kills network would be pointless as the nodes have an in-degree of at most one, so the scores would be high for those who already died. We can take a different network of characters into account for this particular case. An online repository of sample networks, found on the URL https://github.com/melaniewalsh/sample-social-network-datasets, contains a sample Game of Thrones social network, which creates an edge between two character nodes if they appear within a 15-word distance in the original books. Since the shows are
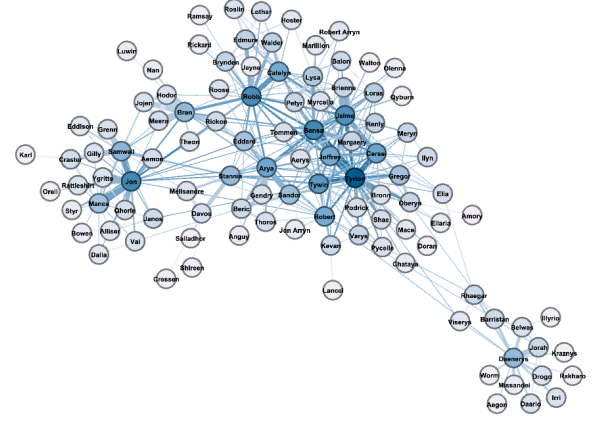

Fig. 3. Game of Thrones social network.

vastly influenced by the books (at least very strongly up to episode 60, to which our kills data is collected) we can use properties from that network as well to gather some additional information. This network has 107 nodes and 352 edges, and is shown on Figure 3.

B.1. Standard machine learning framework. The framework we use for predicting links using machine learning is similar to the framework that is described in section A.1. We split the procedure for getting scores for test links into two parts - first we obtain the scores for positive examples and then the negative examples.

To obtain the scores for positive examples, we first choose some episode, whose kills we are currently trying to predict. These links make up our current test set. Links that appear in episodes after the selected one are ignored, as we must not predict the past based on future events. The kills that happened prior to the selected episode make up our current training set. In addition to that, we sample the same amount of negative examples in order to make the training set balanced. The classifier is then trained on the training set and used to predict scores for chosen test examples. This is repeated for multiple chosen episodes and at the end we obtain scores for $|P|$ positive examples.

For negative examples, we take all the kills from our dataset and sample the same amount of non-kills to form a training set. The test set consists of $|P|$ randomly sampled non-kills from the entire dataset. These test examples are sampled in a way that they do not overlap with negative examples in the training set.

For obtaining the scores, we use three different models: K-nearest neighbors (KNN), logistic regression and support vector machine (SVM). The rest of the framework remains the same as the one described in section A.1.

B.2. PageRank. The first feature we use is the PageRank score (9). We can calculate it to find the most important characters as they will hopefully have the highest scores. Being more important could mean two things: either you are very important and thus have a lot of security by guards and other helpers, so you are very unlikely to die, or the exact opposite. The opposite would imply that since this series is

oriented around taking the power from others, you are more likely to die if you are very important, which seems more plausible since the show thrives on the sudden deaths of the more popular characters. A very low PageRank score of some character would then also mean that they are very unlikely to have their death portrayed since the viewers and readers don't care or have forgotten about the least important people in the story. We use two PageRank-based features, the first being *killer_pagerank* and the second *victim_pagerank*. The top 5 scoring characters after PageRank calculations are Tyrion (0.055), Jon (0.045), Daenerys (0.041), Jaime (0.037) and Sansa (0.036). From our knowledge of the show, we can safely claim that these are some of the most, if not the most important characters, so we can then be sure that these scores make sense in terms of importance. We assign a mean of all the PageRank scores for each character that is found in the kills dataset but is not present in the social network.

B.3. Betweenness centrality. Another measure that we extract from the social network as a feature is the betweenness centrality (10). This score measures how many shortest paths from two different nodes go through a certain other node. That means that the higher the score, the more control over a big portion of the network a node has (i.e. is one of the nodes that can make the network split quickly). That should also yield a measure of importance - if one character wants to reduce the power of a part of a network, they can disconnect it from the biggest component by eliminating nodes with high betweenness centrality. Again, we observe that the five top scoring people are also the most important characters in the story: Jon (0.230), Robert (0.209), Tyrion (0.198), Daenerys (0.157), Robb (0.127). For every character from the kills dataset that is not in the social network, we use a mean of betweenness centrality scores, similarly to the PageRank scores.

B.4. Community detection. Simply using the house that a character belongs to as a feature could perhaps help. The intuition behind this is that there might be more kills occuring between members of different houses than between members of the same house. Since our dataset has a lot of undefined values for the houses, we can find communities in the social network using an algorithm such as the Leiden modularity optimization (8). We can assign a label of the community to each character and see if that helps us with the prediction by creating some sort of indicator between which groups the kills occur. After running the community detection algorithm on the social network, we can see that we obtain meaningful results (meaningful to people who watch the show) in terms of alliances. The network gets partitioned into 5 big communities. For example, we can see that Khal Drogo, Daenerys Targaryen, Aegon Targaryen and Jorah Mormont all fall into the same community even though they do not originate from the same houses, but as we know from watching the show, they are allies. Every character from the kills dataset that is not included in the social network gets assigned to a dummy community.

4. Results

For our experiments we select and remove links that are associated with kills that happened in seasons four to six. There are 114 of those, to which we add 114 randomly selected unlinked nodes and compute the AUC based on these examples. We

Table 1. Results for 4 **link prediction methods: preferential attachment index, Adamic-Adar index and community index. The table shows the mean AUC, precision and recall and their standard deviation over** 5 **runs.**

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)

Table 2. Results for 3 **classic machine learning methods (using the basic dataset): K-nearest neighbors, logistic regression and support vector machine (SVM). The table shows the mean AUC, precision and recall and their standard deviation over** 5 **runs.**

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)

repeat this process 5 times (to account for the randomness in selection of negative examples) and provide the mean AUC and its standard deviation. We support the AUC scores with precision and recall scores as well, since AUC only measures how well randomly selected positive examples can be distinguished from randomly selected negative examples. For the classic link prediction techniques, we classify examples with scores strictly higher than zero as positive and the others as negative. For the machine learning approach, every example with a score greater than or equal to 0.5 is classified as positive and the others as negative. We then calculate the precision as

$$precision = \frac{\#true\ positives}{\#true\ positives + \#false\ positives}$$

and recall as

$$recall = \frac{\#false\ positives}{\#true\ positives + \#false\ positives}.$$

The obtained results are shown in Table 1.

Results show that the alive index achieves best results, achieving a mean AUC of 0.863. Second best results are obtained by preferential attachment index. It achieves a mean AUC of 0.503, which is slightly better than random guessing. However, this preferential index was inverted as described in section 3, giving a higher chance to links occuring between lower degree nodes. The other two methods achieve an AUC around 0.5. This implies that they perform no better than if links were classified randomly.

The standard machine learning approaches came out to be a little bit better than random when using the base kills dataset using only the out-degrees of characters as a feature, with AUC scores ranging from 0.556 to 0.632 as shown in Table 2.

By adding the new network features (PageRank, Betweenness and Community ID for each character) we improve the general performance of all the classifiers and obtain a better top score of 0.686 by using SVM. All the final results for the standard machine learning approaches are shown in Table 3.

Table 3. Results for 3 **classic machine learning methods (using the augmented dataset): K-Nearest-Neighbors, Logistic regression and Support Vector Machine (SVM). The table shows the mean AUC, precision and recall and their standard deviation over** 5 **runs.**

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)

5. Discussion

We see that the sparsity of the network and its size (less that 400 nodes) make link prediction on such a small network very inaccurate in most cases.

Since the original network of kills does not seem to have any community-like structure, it is very hard to predict kills based on community-based link prediction methods, such as the community index. The modularity optimization algorithm finds more than 100 communities with no links between them, which means that the modeling of densities between communities does not give us any information, since the probability of a link occuring between communities is zero. That is a solid foundation for the claim that the authors have done a good job by not creating a very obvious structure of the kills, where someone would kill a lot of people from e.g. their opposing house, making it easy to predict that there will be another similar kill occuring in the following episodes that the viewers have not yet seen.

We can not achieve good results by constructing indices that try to predict kills using out-degrees only since a majority of nodes have a very low out-degree (as can be observed on Figure 2).

The alive index achieves the best results because of the way the network is constructed. Nodes either have an in-degree of zero or one (depending on whether they were already killed or not). When sampling positive examples in our framework, we remove the edge for that positive example, decreasing the in-degree of the target node from one to zero. When sampling negative examples, we do not remove any edges. However, because our original network is constructed from deaths in the series, most of the characters in our network have died at some point. Therefore, the target node of a negative example is quite likely to have an in-degree of one. The alive index makes use of the fact that when we sample a positive example, we are going to decrease the in-degree of the target to zero, and when we sample a negative example, the in-degree of the target is likely to still be one (i.e. that the target was killed by somebody else at some point). We try to account for this by adding some additional isolated nodes (corresponding to characters that did not kill anyone and did not die), but the index still performs best on the expanded network. So although this baseline index achieves good results, it does not do this by using any structural properties of the network but rather just by abusing the way our network is constructed. The results from the standard machine learning approaches are bad, since we only use the out degree as the basic feature from the original dataset to predict kills. When we augment it with different centrality measures and community identifiers, the performance is improved, mostly because of the fact that we do not only have one feature anymore and because these features are not equally weighted anymore. However, there should be some added value to the features due to what the features represent, which is explained at each index description in section 3.

6. Conclusion

Through our work we have acknowledged that the Game of Thrones kills don't have a particularly detectable pattern, since all the kills appear between two nodes that have not yet killed anyone before in most cases. But since our network is small, our test samples are even smaller and that can give deceivingly high AUC scores for indices that would potentially fail on bigger networks and in different scenarios. By using classic machine learning and link prediction techniques, we have found that on this dataset no index or feature works better than a simple baseline index (the alive index) which does not model some useful property, but rather abuses the way the network is formed. Most of the techniques used to predict kills in Game of Thrones gave us a fairly good AUC score, but the predictions that our approaches get right do not have a high "shock factor". For example, our classifier might be able to predict that Jon Snow will kill a wight, but likely fails on less obvious kills. For future work, our approaches could be tested on different TV shows, books or even movies to see how predictable the kills are. The alive index could potentially fail on bigger networks with a bit more diverse structure (e.g. having more bigger connected components) and that would give other more basic indices higher accuracy. We could also construct edges using some other information besides kills, e.g. who had a relationship with whom in some show or who scammed whom in a criminal series.

Source code. The source code to reproduce results, presented in this paper, is available at https://github.com/matejklemen/got-link-prediction.

1. David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.
2. Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3): 211–230, 2003.
3. Dianbo Liu and Luca Albergante. Balance of thrones: a network study on "Game of Thrones" that unveils predictable popularity of the story. *arXiv preprint arXiv:1707.05213*, 2017.
4. Andrew Beveridge and Jie Shan. Network of thrones. *Math Horizons*, 23(4):18–22, 2016.
5. Suveen Angraal, Ambika Bhatnagar, Suraj Verma, Sukhman Shergill, Aakriti Gupta, and Rohan Khera. Risk factors associated with mortality in Game of Thrones: A longitudinal cohort study. *arXiv preprint arXiv:1802.04161*, 2018.
6. David R Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–202, 1972.
7. Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.
8. Vincent Antonio Traag, Ludo Waltman, and Nees Jan van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233, 2019. .
9. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.
10. Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.