AO3 Reading Recommender

Machine Learning [CS7CS4] Project of Group 27

Teona Banu	19326456
Catalin Gheorghiu	22305257
Stefan Iscru-Togan	22317185

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science & Statistics
Trinity College Dublin

August 6, 2023

1 Introduction

Archive of Our Own (AO3) is an online repository that hosts about 10 million pieces of fanfiction and non-fandom literary works. AO3 is not only an important artwork vault but also the most popular way for people to interact with this kind of artwork for free. In spite of - and contributing to - its popularity with readers, AO3 brands itself as strictly an archive. This means that it does not plan on employing recommender algorithms associated with other content-based websites such as Youtube, Netflix, or Pinterest, instead functioning more like a library. While praised by some users, this policy leaves many readers and authors wishing there was an easier way for people to find works they enjoy. That being said, due to the massive amount of works, authors and readers quickly adopted an incredibly detailed tagging system. Each author tags the rating, content warnings, characters, relationships, and fandoms relevant to the work. Furthermore, there is a virtually unlimited[1] number of freeform tags that an author can use or create, such as "Angst" or "Humour". Finally, an author can specify the number of chapters that the work will contain, or mark the work as ongoing, without a set limit of chapters. We made use of this tagging system to create an AO3 recommender algorithm. We sifted through the publicly available data on AO3 and tested out multiple content-based models as well as a collaborative model against a content-based baseline. We found that KNN, Decision Trees, and the collaborative approach do not manage to outperform the baseline, MLP performs slightly better, and Logistic Regression significantly outperforms the baseline, both when prioritising recall and precision.

2 Data

For any single work on AO3, one may see its title, the set of text tags meant to help creators catalog their writing, its word count and number of chapters, its summary, its number of hits (or views), its number of bookmarks reflecting how many people saved it for easy return access, its number of comments and its number of kudos (or likes). Figure 1 should give the reader an idea of the page's layout and the way this information is presented. Note that clicking on the number of kudos or bookmarks leads to a list of users that have engaged with the function for the specific work.

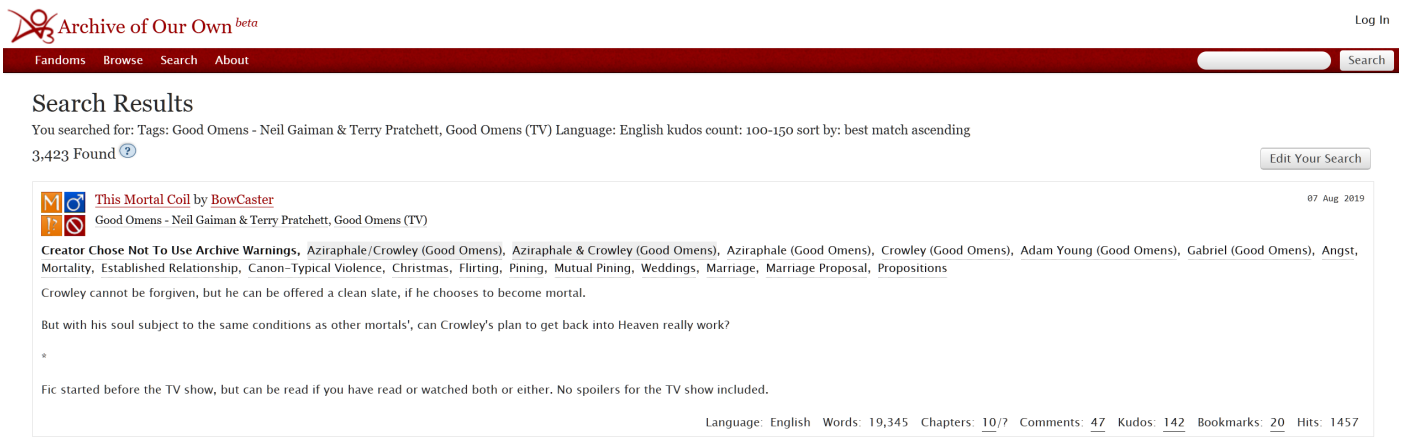

Figure 1: Sample work on the AO3 search results page, featuring most of the data available to us.

The perspicacious reader may have noticed the search criteria in the previous figure, such as a fixed range of total kudos and a restriction to a couple of specific tags: "Good Omens - Neil Gaiman & Terry Pratchett" and "Good Omens (TV)". While the tags themselves were chosen for our enjoyment, we are deliberately limiting the scope of this project to a specific fandom and subset of works for two scientific reasons. The first is to make the workload feasible. All data is scraped directly off the website – an operation that takes between 2 and 15 seconds per work (depending on how much information is requested) even on the best computers available to us. The second is to increase the likelihood of common tags and common readers in our training sets, such that we may learn more information on average from the profile of an arbitrary user. This information forms the core of at least one of the methods we will present in the next section.

As the scraping turned out to be incredibly time-consuming both because of the aforementioned long time spent on a single work and because of the website's terms of use forcing an extra 5 seconds between requests,

[1]The number of tags was limited to 75 in 2021 due to a controversial explicit work that used 1700 tags and would either crash the website when opened or take a very long time to scroll past. The number 75 was chosen because less than 0.5% works user more than 75 tags, with the mean and mode being 17 and 11 respectively. Archive of Our Own (2021)

we only retrieved the information we believed users would naturally use to choose the next work to read. Thus we scraped the word count and list of tags for each work to be used in content-based approaches, and the list of users who left kudos for collaborative filtering approaches.

Information such as the number of hits of bookmarks seems inconsequential to the actual literary preferences of users. It is true that the most popular ways of sorting through works are currently date based and popularity based. However, the purpose of our algorithm is to make it easier for less popular works to be discovered and enjoyed. Thus, not including hits, bookmarks, or comments, and limiting the scope to works with a medium-small number of kudos not only makes analysis easier, but makes logical sense for our purpose. The title or summary would be rather voluminous and difficult to work with. Finally, the number of chapters is already highly correlated with the word count (although, further attempts might find value in taking the completion status of a work as a feature). Before discussing our approaches in-depth, we note that while a list of users who had seen and disliked the work would have substantially increased the effectiveness of collaborative filtering, such information was not available for scraping. We believe a lot of research like ours would benefit from AO3 collecting this information in the future, even in some compressed form, but this is unlikely to happen since the concept of a "dislike" is somewhat contrary to the AO3 ethos.

The data preprocessing is described in Appendix A. After this process, we are left with two main tables acting as the pillars of the analysis. We begin with the more simple one, which maps a subset of 359 users (chosen by selecting for users who gave kudos to at least 50 works) on rows to 1531 different works across the columns (chosen by selecting works with a minimum of 5 kudos remaining after removing the kudos-stingy users). Each cell contains a 1 if the user left kudos on the respective work, and a 0 otherwise. Figure 2 is a snippet of this table, reflecting the insight offered thus far. The second, more complex table employs user-work combinations as unique row identifiers, making for the repeated username entries seen in figure 3. The kudos column is a binary reflecting whether the user left kudos on the corresponding work. The tag preferences that occupy the rest of the table are calculated using formula 1.

$$x_{uwt} = \left(\frac{UT_{ut}}{\sum_{w \in W} UW_{uw}} - \frac{\sum_{w \in W} TW_{tw}}{|W|}\right) \times TW_{tw} \tag{1}$$

This formula uses notation defined and explained in Appendix A. In short, U, W, and T are the sets of users, works, and tags, while UT, UW, and TW are matrices that map these elements to eachother. The fist element represents a user's preference for a tag computed as the probability that a work liked by the user contains the tag. The second element is the usage rate of that tag. The higher the difference between these, the more a user can be said to actively select for/against this tag. The third element takes value -1 or 1 depending whether work w contains tag t. Note that while this matrix usually contains value of 0 and 1, for this formula -1 is used instead to signal the work does not use the tag. The product of this element with the previous difference results in positive values if a user's preference for/against that tag is fulfilled by the work, and negative values otherwise. The closer the value is to 0, the more ambivalent the user is. Higher word counts reflect a propensity to appreciate longer works.

	useri	19883812	20058244	20458610	20961095	21040970	21181403	21430618	21527128	21698359	...	26646586	27642413	27691817	28047846
0	HolRose	0	0	1	0	0	0	0	0	1	...	1	0	0	0
1	Starwolf23	0	0	0	0	0	0	1	1	0	...	0	0	0	0
2	Eigon	0	0	1	0	0	0	1	0	0	...	0	0	0	0
3	Sassy_Angel	0	0	0	0	0	0	0	0	0	...	0	0	0	0
4	Shadowfang44	0	0	0	0	0	0	0	0	1	...	0	0	0	0

Figure 2: One-hot encoded table showing which works received kudos from individual users.

1	user	id	kudos	wordcount	General Au	No Archive	M/M	Aziraphale/	Aziraphale
2	HolRose	19149922	0	0.000598	0.1449176	0.0315744	0.1020397	0.0394225	-0.01571
3	HolRose	19152154	0	0.000436	0.1449176	0.0315744	-0.10204	0.0394225	-0.01571
4	HolRose	19209664	0	0.0016043	-0.144918	0.0315744	0.1020397	0.0394225	-0.01571
5	HolRose	19210612	0	0.00042	0.1449176	0.0315744	0.1020397	0.0394225	-0.01571
6	HolRose	19212322	0	0.0001843	0.1449176	-0.031574	0.1020397	0.0394225	-0.01571
7	HolRose	19212925	0	0.0013395	0.1449176	0.0315744	0.1020397	0.0394225	-0.01571

Figure 3: Table of relative kudos inclination across tags for each unique user-work combination.

3 Methodology

To explain a bit of common terminology for recommendation systems first, content-based approaches make use of the similarity between works to find a recommendation the user is likely to offer kudos to. Collaborative-filtering approaches, on the other hand, use the similarity between users to recommend the target individual a work that a similar person appreciated, working off the assumption that the target individual will give kudos to it as well. In broad strokes, we aim to compare these two approaches to each other, as well as a baseline guesser that is 100% matrix multiplication and 0% machine learning – if one assumes there is a difference between them to begin with...

3.1 Content-based

Using the table designed specifically for the content-based approaches that we elaborated on previously, a "content-based champion" will be the best performer between logistic regression and its regularisation-stabilised sibling *Ridge* regression, k-Nearest-Neighbours (kNN) and a multi-layer perceptron (MLP) neural network. We shall of course show the cross-validation processes leading to the choices of optimal parameters in each of these approaches, it is only fair to compare them at their best. The baseline algorithm produces a likelihood for each user-work combination to yield kudos based on matrix multiplications. The methodology for this (content-based) baseline recommender is described in Appendix B.

3.1.1 Logistic Regression

The central idea of logistic regression, given our training labels of kudos $y^{(i)} = \{0, 1\}$ and feature set $x^{(i)}$ containing the relative word count and augmented tag appreciations, is to minimise the cost function in the equation below with respect to parameter vector θ. The *Ridge* regression adds an L2 penalty of the form $\frac{\theta^T\theta}{C}$, whose harshness is intuitively regulated by hyperparameter C. We shall cross-validate the model for different values of C to strike a balance between precision (adversity to false positives) and recall (adversity to false negatives) when faced with new data. For both versions of the logistic regression, SAGA is our optimisation algorithm of choice as its steady performance in the face of large, sparse data sets seems to intuitively fit our goals better than the default LBFGS alternative.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} log(1 + e^{-y^{(i)}\theta^T x^{(i)}})$$

3.1.2 K Nearest Neighbours

We implement kNN both before and after re-balancing the dataset using the *Imbalanced-Learn* library. Random works that received kudos - normally outnumbered 20 to 1 by those cumulatively passed by - are duplicated systematically until the minority class attains a significantly higher fraction of entries: 25% in our case. In order to determine the optimal number of neighbours k, we cross-validate with 4 and 5 folds respectively before and after re-balancing – we shamefully admit the former to be a result of impatience, however we believe the results section will show the rebalancing was substantially more impactful than 5% more training points would have been to prevent virtually non-existent overfitting.

3.1.3 Multi-Layer Perceptron

The MLP is indeed traditional black-box neural network, yet under no means does that relieve us of the duty of fine-tuning its architecture to the problem at hand. We perform two rounds of cross-validation for the two core hyperparameters: the number of nodes in the hidden layer n and the regularisation term C of the logistic loss cost function to be minimised for optimal performance – see formula below. For the activation function meant to translate the hidden weights to the output classification (i.e. kudos or lack thereof), we stand by the tried and tested rectified linear unit (ReLU) as there is little indication in the problem itself that another method may be more suitable. The empirical trustworthiness of ReLU should be enough to carry our results through.

$$\text{Log Loss} = \sum_{(x,y)\in D} -y\log(y') - (1-y)\log(1-y')$$

- $(x, y) \in D$ is the data set containing many labeled examples, which are (x, y) pairs.
- y is the label in a labeled example. Since this is logistic regression, every value of y must either be 0 or 1.
- y' is the predicted value (somewhere between 0 and 1), given the set of features in x.

3.2 Collaborative Approach

Shifting our focus to the information on the kudos given by each user, an item-based algorithm in the style of (Sarwar et al., 2001) will compete against the content-based approaches. Given that this algorithm strand is specifically designed for recommenders, we are excited to test its mettle. We will use cosine similarity to quantify the element-to-element relationships of our works, aiming to create a new element-by-element array containing the weights (relationships) between each work, where perfect correlation equals 1 and no correlation corresponds to a score of 0. Its robustness and simplicity are strong arguments for employing cosine similarity – the formula of which is given below. A never-before-seen work would thus be recommended to a user if similar users had appreciated similar works in the past.

$$cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|} = \frac{\sum_i A_i B_i}{\sqrt{\sum_i A_i^2}\sqrt{\sum_i B_i^2}}$$

3.3 Measuring performance

In order to compare the performances of these algorithms, we must carefully choose which measures to use. There are three main challenges inherent to our data. First, users can only scroll through a relatively small number of works in a given time, and can read even fewer. Second, partly stemming from the first issue, there is a huge class imbalance; users only give kudos to a small subset of works. Third, related to the previous two, since users cannot dislike a work, the real number of negatives is completely unknown, and true positives are only partly known. A user not giving kudos to a work does not necessarily mean the user did not like the work. Chances are they never came across it, scrolled past it, or read it and only mildly enjoyed it (since most people will not keep actively engaging with something they dislike). Due to the second and third challenge, measuring performance by looking at accuracy would be a deeply flawed approach. Precision might also be inaccurate since it it based on the number of false positives, which consists of real negatives, which are unknown. Using recall seems to mitigate the third challenge, but has its own issues. Namely, the more works we end up recommending, the higher the recall score, but the less likely it is that a user will actually interact with all of them, decreasing the recall score reliability. We will keep these things in mind moving forward, and we will base our results discussion on all four performance measurements (accuracy, recall, precision, and F1).

4 Results

4.1 Baseline Recommender

Table 1 shows the performance of our baseline content-based recommender described in Appendix B. The threshold represents the percentage of works that the algorithms predicts the user will like. For a threshold of n%, the algorithm predicts that the user will like the top n% best works for that user. The thresholds of 0.1% and 34% correspond to the resulting threshold of the ridge regression with default and balanced class weights respectively. Thus, it will be particularly interesting to compare the results of those models compared to the baseline model with the same number of predicted kudos.

Threshold	Accuracy	Recall	Precision	F1
0.1%	94.7%	0.6%	40.9%	1.2%
1%	94%	4%	21%	6%
5%	91%	13%	13%	12%
10%	87%	21%	11%	14%
34%	67%	52%	8%	14%

Table 1: Baseline performance at various thresholds

4.2 Content-Based Approaches

4.2.1 Logistic Regression & Adjacents

Table 2 shows the results of ridge regression with $C \in 0.01, 0.03, 0.05, 0.5, 1$ as well as logit regression. Because our data is so imbalanced, we also trained each of these models with balanced class weights. This resulted in slightly lower accuracy, but skyrocketed the recall measure and significantly improved the F1 score in spite of tanking the precision. The time it took to train and evaluate the models is also shown, in seconds. Using balanced class weights leads to a 35-45% increase in this time.

class_weights	C	Accuracy	Recall	Precision	F1
None	0.0005	94.689%	0.000%	0.000%	0.000%
	0.01	94.7%	0.5%	62.6%	1.0%
	0.005	94.7%	1.2%	44.0%	2.3%
	0.1	94.6%	1.5%	38.5%	2.8%
	0.5	94.6%	1.9%	31.4%	3.6%
	None	94.5%	2.2%	28.3%	0
None	0.00005	63.2%	60.8%	8.6%	15.0%
	0.001	63.5%	60.9%	8.7%	15.2%
	0.0005	62.9%	62.0%	8.7%	15.2%
	0.01	62.3%	61.6%	8.6%	15.0%
	0.05	62.3%	61.6%	8.6%	15.0%
	None	59.5%	59.6%	7.8%	13.7%

Table 2: Performance of ridge regression with various C weights as well as logit regression with no penalty.

Figures 4 and 5 show the 5-fold cross-validation process of selecting a penalty weight. The weaker the regularisation, the worse the performance. This is because the weaker the regularisation, the more the model overfits when training. For the ridge regression with no class weights, C = 0.01 achieves the best performance. For the ridge regression with balanced class weights, C = 0.001 achieves the best performance and the smallest error bars.

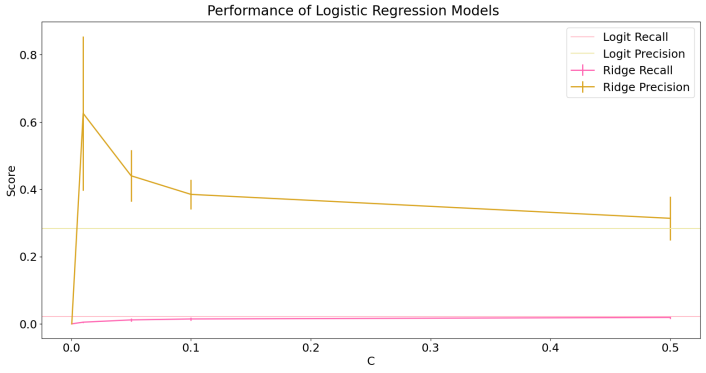

Figure 4: Cross-validation for ridge regression with default class weights compared to the logit

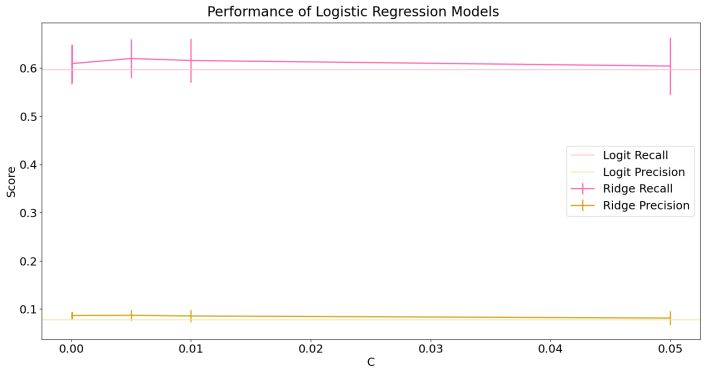

Figure 5: Cross-validation for ridge regression with balanced class weights compared to the logit

Compared to the baseline, the default class weight ridge regression achieves very similar accuracy, recall and F-score. However, the precision is 25% higher, showing the merits of the logistic regression over the baseline. For the balanced class ridge regression, the accuracy, precision, and F1 scores are very similar, but the logistic regression achieves a 20% better recall score. This shows that logistic regression manages to outperform the pure content-based baseline.

4.2.2 k-Nearest-Neighbours

This method was perhaps the most likely to benefit from oversampling as multiple identical neighbours form solid clusters, bound to encourage the algorithm to assign positive labels to similar, close-by works in the *359*1388*155* - dimensional space of user-work-tag interactions. Results before re-balancing the sample are condensed in figure 7. It is immediately apparent that the choice of neighbour count matters little, as the high accuracy in contrast with all other metrics is a strong indicator of the algorithm's bias towards negative labels. This intuition is corroborated by the confusion matrix in figure 7 and should strike one as completely unsurprising given the large imbalance in favour of negative - or more accurately neutral - labels in the data. A high false negative rate is thus observed to also drag down recall and F1 score. As a result, kNN performs overall much worse than even the dummy classifier on the unbalanced dataset, raising expectations for the next part of this analysis.

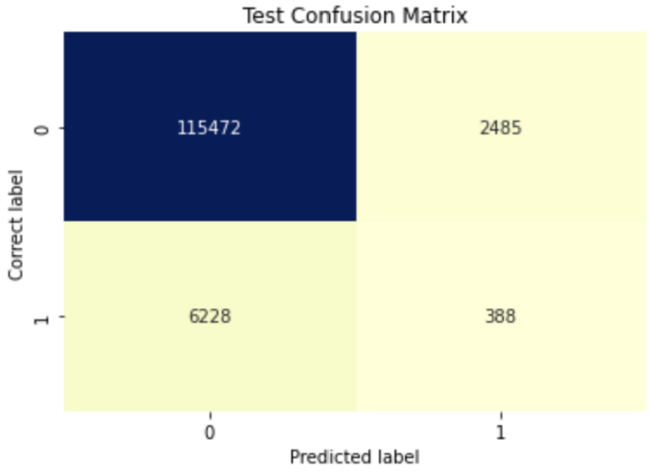

Figure 6: Confusion matrix of a signle 5-nearest-neighbour cross-validation fold.

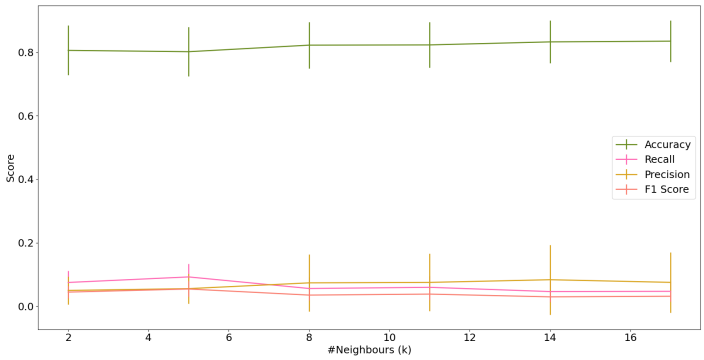

Figure 7: Cross-validation results of kNN and their quantification through various common methods.

Oversampling the works that have received kudos up to covering 25% of the dataset has a significant impact on performance metrics, best seen in figure 8. For a low number of neighbours, precision increases over 25% on average, likely the result of previous false positives turning intro true positives thanks to the oversampling. For higher numbers of neighbours however, the algorithm seems to balance back to near 0 precision, perhaps because the previously-mentioned clusters of positive labels stretch out far enough to begin wrongly assigning additional positive labels once again. To wrap up this method, we would say kNN remains unconvincing as a method for this dataset, seemingly being eventually overwhelmed by the large dataset with complex neighbour interactions.

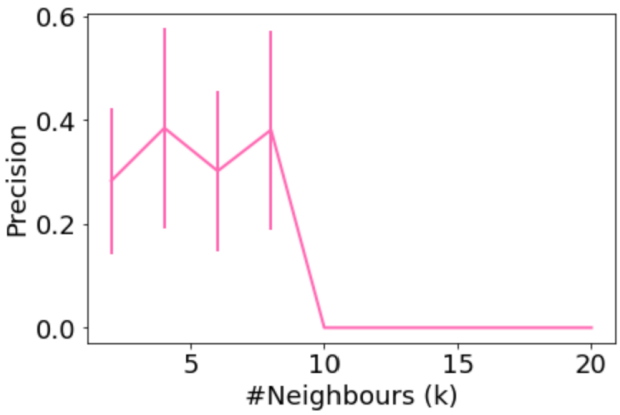

Figure 8: Cross-validation kNN results in the dataset adjusted with an oversampled minority class.

4.2.3 Multi-Layer Perceptron

Results from the first round of cross-validation have been summarised in figure 10. Note a trend similar to what we observed with kNN as the size of the hidden layers increasing, with accuracy decreasing and recall rising as the algorithm becomes bold enough to shift off the safe choice of never recommending works. Given the existing imbalance, it is intuitively understandable that true positives increase faster than false negatives, resulting in the trend we see. The trade-off does not seem more damaging to accuracy than it is beneficial to recall, and we would prefer an algorithm actually making recommendations than always playing it safe, however larger hidden layer lengths increased the time taken by the algorithm exponentially. As a result, we move forward with a balanced hidden layer length of 50. The confusion matrix in figure 9 shows the status of one fold belonging to this hidden layer length, making the shift towards positive labels obvious in comparison to the kNN confusion matrix in figure 6.

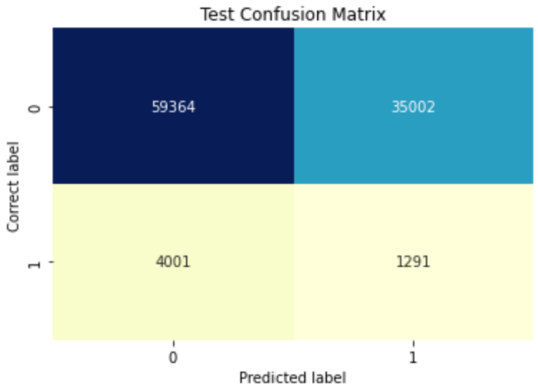

Figure 9: Confusion matrix of a single 50 hidden layer MLP cross-validation fold.

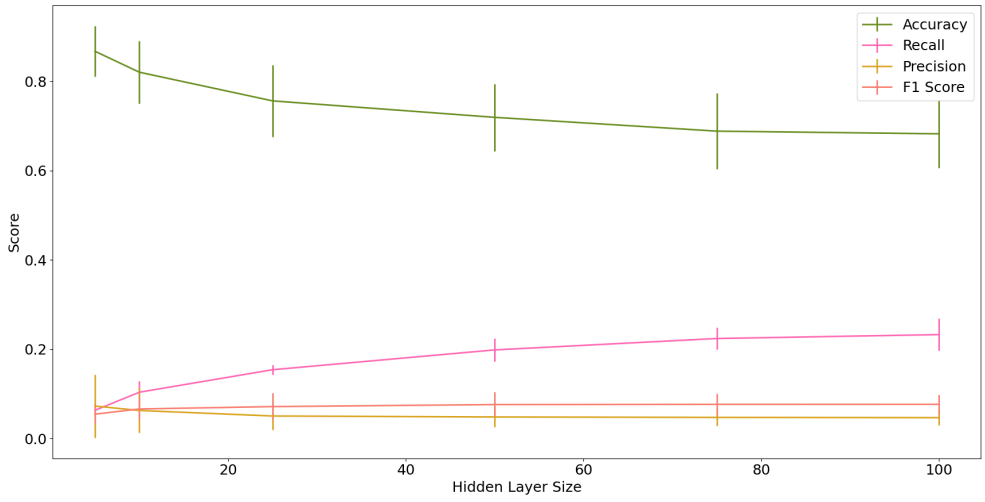

Figure 10: MLP cross-validation process to identify the optimal number of hidden layers.

Cross-validation over the regularisation hyperparameter C was far less eventful - so much so that we decided it was not worth the space to plot it. Low values of C lead to virtually no positive label predictions, and as the value of C increases toward 100, the network becomes braver in its recommendations, reaching up to 400 true positives. The algorithm took over an hour to progress through a single fold of cross-validaiton when C=100, forcing us to abort before obtaining complete results. Nevertheless, it is clear that the algorithm would perform better as C increases until the soft recall limit of the previously observed regularisation-free framework is reached. We thus move forward without regularisation.

Let us then compare this best version of MLP to a strongly regularised L2 penalty logistic regression at C=0.001, which proved somewhat more effective than the baseline in its previous section. The ROC in figure 11 makes it clear that the neural network is visibly superior to both the dummy classifier that never recommends anything and the logistic regression at every level of the false positive rate.

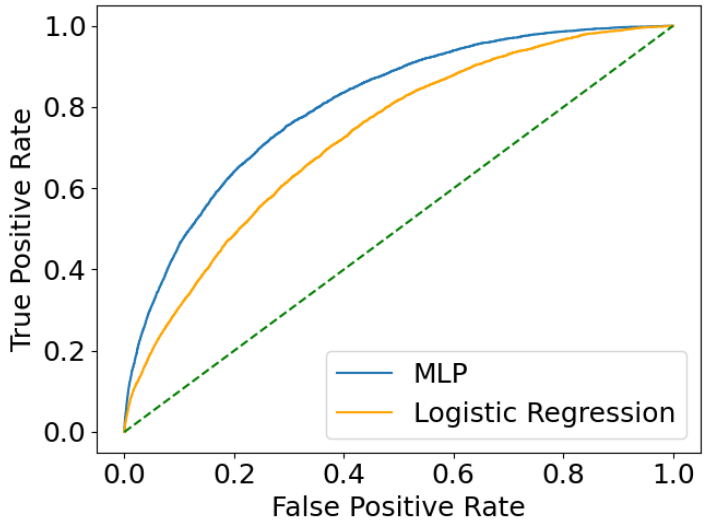

Figure 11: Receiving operating characteristics (ROC) curve comparing 50-layer MLP to a dummy classifier and baseline logistic regression.

4.2.4 Decision Trees

We end the content-based section with a secret small execution of a random forest framework. We did not have the time to flesh this approach out, but we felt its results may be worth discussing in comparison to the other algorithms in the concluding discussion on the best methods available.

Decision trees are powerful deep-learning tools that arrange decisive class feature values in a tree hierarchical structure and perform this tree recursively, each step defining feature values strongly correlated to one of the classes. We trained a random forest classifier and performed grid search to deduce the best parameter values – most importantly the maximum number of child nodes per level. Unfortunately, the model performed worse than anticipated, attaining a 55% accuracy score after the dataset has been downsampled to ensure class equilibrium.

4.3 Collaborative-Filtering (CF) Approach

We aim to make the item-item matrix as representative as possible for the general user pool. That is why the user vectors need to be normalized beforehand. In other words, it is important that the matrix displays the works appreciated by more selective users as more representative than the ones appreciated by users that would give kudos to every piece that comes around. This goal is achieved by normalizing the user vectors in the following manner:

- Compute the magnitude of the user ratings

$$magnitude = \sqrt{(x^2 + y^2 + z^2 + ...)}$$

- Compute the unit vector based on individual magnitudes

$$\textbf{vector} = \begin{bmatrix} \frac{x}{magnitude} \\ \frac{y}{magnitude} \\ \frac{z}{magnitude} \\ ... \end{bmatrix}$$

As this normalization step is applied before the similarity computations are done, instead of binary ratings we have a matrix with values that reside between 0 and 1 that correspond to the importance of each user rating. With the similarity matrix in place, the kudos are ready for analysis; thus we create an item-user score using the formula below, where W_{ij} represents the weight associated with the element on row i and column j, and r_{ui} is the actual kudos score the user u gave to work i. The n highest scores shall determine the works to be shown to the user.

$$S(u,i) = \frac{\sum_{j \epsilon N} W_{ij} r_{ui}}{\sum_j \mid W_{ij} \mid}$$

The code implementation starts off by computing the magnitude and unit vectors. The newly formatted data is passed as an argument to the *compute-similarity* method. The sparse matrix is generated using the transpose of the previously shown vector matrix and SciPy's *sparse* instance.

The non-null matrix values are associated with their row/column indices. These elements are used to compute the cosine similarity scores. Having all the elements in place, we test this approach on a random user.

1. We retrieve the IDs of the appreciated works by the specific user.
2. We use the function presented above to calculate the score for each individual item in the user vector.
3. We drop the user's known liked work IDs and recommend the top n works.

```
36436963    0.014229
28814523    0.014182
30822749    0.013534
31919452    0.013299
21856060    0.013180
```

We notice that even the highest scores don't indicate a strong confidence degree in the recommendation that has been made suggesting that some improvements could be performed for this approach. One intuitive strategy is to rely solely on the top 10 values from every sparse matrix column. Only the strongest correlations are thus weighed.

```
for i in range(0, len(data_matrix.columns)):
    data_neighbours.iloc[i,:10] = data_matrix.iloc[0:,i].sort_values(ascending=False)[:10].index
```

For this newly created dataframe, we reiterate the initial processing steps. Two elements should be discussed when comparing the two results:

```
36436963    0.022372
21856060    0.022343
28814523    0.022328
20018326    0.021727
21949201    0.021587
```

1. Despite the fact that the mean correlation score increases, the score increases for all the elements (no matter their ranking in the scores list) so we cannot conclude that the algorithm is more confident when distinguishing items that should not be recommended from the ones that should be.
2. The IDs of the fiction associated with larger scores are slightly changing for the top prediction. In other words, this approach could be a better solution when assessing the order in which items must be recommended to the user.

5 Conclusion

5.1 Discussion - What Worked Best?

Let us boil down a summary comparison to the tradeoffs between accuracy, precision and recall, since F1 score remained consistently low across methods and hyperparameter adjustments. Given the tensions specifically between precision and recall that we described in the "Measuring performance" section - born out of the way users interact with the kudos system on AO3 - it is not worrisome that F1, a measure tailored to balance them, makes no progress when one increases at the cost of another.

Logistic regression performed best when class balance was introduced, forcing the algorithm over its bias to never label any work as likely to get kudos. Recall and accuracy balanced out at 60% while sacrificing precision almost completely, hinting at the algorithm's achievement to boost its true positive rate while keeping false negatives born out of excessive caution under control. These results show a fair advantage over the baseline's best balance of accuracy and recall, which reached 67% and 52% respectively. This performance corresponds to an average of 34% of user-work pair being predicted to result in a user giving kudos. This is 6 times higher than the status quo. This suggests that users have only found, on average, one in every 6 works they might end up enjoying. Given that one can never read enough, we find this threshold to be plausible enough. This means that we manage to avoid the biggest challenge posed by using recall as a performance measure - our set of suggested works is small enough for a user to realistically be able to peruse it.

kNN gets nowhere near beating the baseline and logistic regression even in individual metrics, let alone any meaningful fusion of them. The complexity of the dataset just proved to be too much for neighbour identification to properly form helpful clusters of similar works. The multi-layer perceptron, on the other hand, fared rather well, capable in its own right to balance accuracy and recall in taking risks to recommend a relatively large number of works. This algorithm achieves one of the highest number of false positives, driving precision very low; however, we would argue the additional true positives and fewer false negatives compared to kNN for example make it a more valuable recommender system overall, shown to trump even logistic regression (without class balancing, admittedly) on an ROC curve. Random forests failed to make a mark on this project, and we hope future work may give them another chance in the algorithm competition.

Finally, the collaborative approach failed to capture meaningful correlations even when all but the most similar and active users are considered. Collaborative filtering thus lands on the promising future research side instead of showing impressive results, leaving balanced-class logistic regression and MLP as leading algorithms for this task.

5.2 Limitations & Further Research

We insisted enough thus far on the limits of what could be scraped and how they forced various pivots in the project. Smaller limitations worth mentioning include lacking the harmonious time-computing power balance needed to push the rigour of heavy methods like MLP further. We had to cut corners like lowering the maximum iterations in the aforementioned MLP algorithms and reducing the number of folds when cross-validating the number of neighbours in kNN in order to finish on time - projects on a more relaxed timeline may correct this. Another well-felt limitation was the class imbalance that, while we made some efforts to alleviate, needs a more systematic tackling and warrants some analysis of its own to determine the best strategy to work around it.

It would obviously be interesting for these algorithms to be applied on larger scales, cross-fandom, and generally spanning more works and users. Furthermore, incorporating more advanced techniques like sequential recommendations may help narrow down the very sparse matrices one ends up working with after at least a moderate volume of data collection. We would have loved to try combining computationally-heavy methods to see how peak deep learning would compare to the more traditional methods we have employed, for instance applying k-means clustering and training neural networks on specific clusters may be a powerful hybrid approach we would be very excited to see implemented in the future. Having decided on balanced-class logistic regression and MLP as the strongest algorithms to move forward, it would of course be especially interesting to see how the neural networks performs in a balanced dataset and whether it will be capable to take over the elegant simplicity of the logistic regression.

6 Contributions

- **Teona Banu**
  - Data processing
  - Baseline recommender
  - Logistic regression
  - Scraping (support)
- **Catalin Gheorghiu**
  - KNN
  - MLP
  - Logistic regression (support)
  - Baseline (support)
- **Stefan Iscru-Togan**
  - Scraping
  - Collaborative Approach
  - Decision Trees
  - KNN (support)

We all collaborated on all the parts of the report. All code available on GitHub: `https://github.com/stefanTogan/MachineLearningProject`.

References

Archive of Our Own (2021). Upcoming limit on tags per work.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, page 285–295, New York, NY, USA. Association for Computing Machinery.

7 Appendix A: Preprocessing

Data preprocessing consists of generating two tables. The first one, $UW = users \times works$, maps each user to each work with value 1 if the user gave kudos to the work and 0 otherwise. The second, $TW = tags \times works$ maps each tag to each work with value 1 if the work contains the tag and 0 otherwise. By performing the matrix multiplication $UW \times TW^T = UT$ the resulting matrix $UT = users \times tags$ maps each user to each tag using the number of times a user gave kudos to a work containing that tag. This can be interpreted as the extent to which a user enjoys that type of content, and is analogous to the extent to which someone enjoys a movie or music genre. The next example shows this process:

$$\text{Raw data} = \begin{bmatrix} Work & Tags & Users \\ w_1 & t_1, t_2, t_3 & u_1, u_2, u_3, u_4 \\ w_2 & t_2, t_3, t_4 & u_1, u_3 \\ w_3 & t_1, t_4 & u_4 \end{bmatrix}$$

$$UW \times TW^T = \begin{bmatrix} & w_1 & w_2 & w_3 \\ u_1 & 1 & 1 & 0 \\ u_2 & 1 & 0 & 0 \\ u_3 & 1 & 1 & 0 \\ u_4 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} & t_1 & t_2 & t_3 & t_4 \\ w_1 & 1 & 1 & 1 & 0 \\ w_2 & 0 & 1 & 1 & 1 \\ w_3 & 1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & t_1 & t_2 & t_3 & t_4 \\ u_1 & 1 & 2 & 2 & 1 \\ u_2 & 1 & 1 & 1 & 0 \\ u_3 & 1 & 2 & 2 & 1 \\ u_4 & 2 & 1 & 1 & 1 \end{bmatrix} = UT$$

8 Appendix B: Pure content-based work enjoyment likelihood

By performing the matrix multiplication $UT \times TW = Baseline$, we obtain our baseline recommendation matrix. This maps each user to each work, but instead of zeroes and ones, it contains numbers which can be interpreted as the likelihood that a user will like a work. The following example illustrates this process:

$$UT \times TW = \begin{bmatrix} & t_1 & t_2 & t_3 & t_4 \\ u_1 & 1 & 2 & 2 & 1 \\ u_2 & 1 & 1 & 1 & 0 \\ u_3 & 1 & 2 & 2 & 1 \\ u_4 & 2 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} & w_1 & w_2 & w_3 \\ t_1 & 1 & 0 & 1 \\ t_2 & 1 & 1 & 0 \\ t_3 & 1 & 1 & 0 \\ t_4 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} & w_1 & w_2 & w_3 \\ u_1 & 5 & 5 & 2 \\ u_2 & 3 & 1 & 1 \\ u_3 & 5 & 5 & 2 \\ u_4 & 4 & 3 & 3 \end{bmatrix} = Baseline$$

This example shows how this technique ends up not only attributing high values to the works that the users did enjoy, but results in a plausible looking prediction: user 4 might enjoy work 2, since work 2 has tags that user 4 has liked in the past. Also noteworthy is that the end values are not normalised. No inter-user comparisons can be drawn, because lower numbers overall might simply mean the user gave fewer kudos. However, for each user, ordering by "likelihood" leads to the order in which the baseline suggests works, with works that the user is more likely to enjoy being suggested first. This approach poses a challenge when evaluating - the higher the threshold of works to be suggested, the more likely it is that the user has given kudos to the works, and the higher the recall measure. However, the higher the threshold, the lower the chance a user will actually read all the suggestions, or even scroll through all of them. The pros and cons of using other performance measures are discussed in the methodology section.