You and Meme: A Meme Recommender System

by Matthew Ottomano and Ammar Almahdy
# Table of Contents

- Introduction
- Collecting Data
- Creating a Webpage
- Finding Memes
- Advertising
- Getting Hacked; A Lesson in Security
- Randomizing Memes
- Responding to Complaints
- Using the Right Libraries: SURPRISE
- Explaining Root-Mean-Square-Error
- User-User Collaborative Filtering
- Item-Item Collaborative Filtering
- Matrix Factorization Using SVD
- Comparing our Performance to Netflix
- Real-Time Recommendations
- Conclusion
Introduction

In our changing world where information is being transferred at an extremely fast rate, memes are at the forefront of this cultural phenomena. On social media websites such as Reddit, Facebook, Instagram and Twitter, memes dominate the social circle, leading users to scroll endlessly throughout memes that make them laugh, cry and cringe. There are ways to narrow one’s search of memes; one way is categorizing memes in different genre’s such as wholesome, sad, funny, and happy. This is similarly done with movies, however one feature that memes are missing is a personalized feature that allows memes to be recommended to users based on their likelihood of liking the meme. This is where we come in. We decided that Netflix users are not the only ones who deserve recommendations, so we worked on creating a recommender system for memes and investigated three different algorithms for implementing this system.

Collecting Data

One of the biggest problems we faced creating a recommender system was collecting user ratings for memes. Unfortunately, unlike movies, memes do not have large datasets with user ratings so to solve this, we decided to take data ourselves. We recognized the implications of doing this and the amount of data we would be able to collect in a short amount of time, however this didn’t stop us from attempting to collect as much data as we could. We discussed potential efficient ways to collect user ratings including google docs, however we agreed that creating a webpage would be way more efficient.

Creating a Webpage

The advantage of creating our own webpage is that we could have it be completely personalized to provide the most seamless experience for our users. For us, what mattered most was to quicken and facilitate the process of gathering data as much as possible. Thus, we created our website to work in the following order: user creates an account or logs-in to an already-created account. Once the log-in is successful, the user is redirect to the main website page which contains simply an image ‘a meme’ and 5 little buttons labeled 1-5 for the rating. Immediately as the user presses a button of the 5, the meme, rating and username of the user get stored in our database and another meme seamlessly appears to the user. Thus, for the user, the process is as easy as just clicking 1 of 5 buttons.

Finding Memes

We found large datasets of memes on Kaggle; one had about 3,000 memes while the other had well over 80,000 memes. The problem with this is, we never actually looked inside the datasets because we thought that a meme dataset was self-explanatory. We later realized that the “meme” datasets just composed pictures and was supposed to be used in a neural network
that creates memes. So we did the next best thing; we collected memes ourselves. We
dedicated hours to simply going through Reddit and Facebook downloading memes. To avoid
bias, we included every meme we saw, regardless of how vulgar it was. Our reasoning for this is
that if we just included good memes, there would be no need for a recommender system and
judging whether a meme is good or not is already something subjective that our recommender
system tries to iron out. In the end we managed to get over 400 memes.

Advertising

Our next step was to get people to go on our webpage and start rating memes. We thought of
places we could advertise that would yield the greatest participation. The first place we
advertised was the Facebook group for Middlebury’s memes. This is a hub for Middlebury
students and memes and we thought that there couldn’t be a better place to advertise this
webpage than here. We also advertised on the Middlebury class of 2020 Facebook page and to
friends from other colleges in hopes to get a wide variety of students. These other colleges
include Hunter College and Queens College. A friend in LA also managed to get a few high
school students to rate some memes.

Getting Hacked; A lesson in security

During our process of gathering meme-ratings, we woke up once to an SQL Injection attack on
our website. While our website is written entirely in PHP and JavaScript, both platforms talk to
our database which is written and modified using MySQL. Through the log-in textboxes
available on our website, the hackers were trying to inject malicious code to our database that
would have done the following: grant them a full record of the usernames and passwords of
our users, and even worse, delete the entire database of ratings.
While that would not have hurt us for we have backed up our data, we were very surprised to
see someone devoting their time to hack a simple website that is aiming for nothing but to
grant people some fun. Consequently, we spent several hours implementing security roles and
injection-prevention procedures on the website to ensure the security of our database. Thus
far, nothing malicious has managed to get through.

Randomizing Memes

Once our webpage was running smoothly, we opened the discussion up to how we should be
collecting the user ratings. With careful consideration to how we collect the memes, we
thought that the process that we collect user ratings is just as if not more important. In the
preliminary stages, we realized that our recommender system wouldn’t be able to calculate
recommendations because we were giving the same memes to all users in the same order. For
a recommender system to work, there must be a variety of different user ratings from different
memes to accurately recommend a meme to a user. To fix this, we randomized the memes so
that the order of memes that a user received was always different. Due to an average of 20
ratings from each user, we didn’t put a limit to the number of memes that a user could rate
since we predicted they would stop after a few. This allowed us to get a variety of different memes rated to allow accurate predictions of ratings for memes that users have not rated.

Responding to Complaints

Throughout data collection, we received complaints about the content of some of the memes on our webpage. These complaints discussed the vulgar and profanity of some memes. To address this, we added a note to our second launch of advertising which explained how we didn’t discriminate against any meme to avoid bias in our data. We also went on to explain that without these vulgar memes, our recommender system would not be able to predict that a user wouldn’t like these sorts of memes which would do a disservice to everyone. I think we the response was overall understanding and positive.

Using the Right Libraries: SURPRISE

To implement a recommender system, we spent some time looking for useful libraries to help us. Through our extensive research, Surprise, a python library, caught our eyes. This library, a library meant for recommender systems, allowed us to extract algorithms like n-nearest neighbors and SVD to implement our recommender system algorithms.

Explaining Root-Mean-Square-Error

The root-mean-square-error (RMSE), was the unit we used to measure the accuracy of our recommender system. We realized that a recommender system is very different from algorithms like logistic regression or neural networks in the way that labels and values are predicted. In a neural network, classifiers are clear and consistent and can match labels flawlessly. A recommender system however doesn’t have specific classifiers. For example, our rating system goes from 1-5, however a recommender system doesn’t predict an integer from 1-5, rather it predicts a real number from this range. This means that we were not able to do a traditional accuracy assessment that measures correctly guessed vs incorrectly guessed. A predicted rating of 4.99 on a actual rating of 5 is really close and a great prediction, but with the strict guidelines of an accuracy assessment, this would classify it as an incorrect guess since 4.99 does not equal 5. To combat this, we used RMSE, which one can think of as a standard deviation, to measure the accuracy of our system.

User-User Collaborative Filtering

Our first algorithm that we implemented for our recommender system was User-User collaborative filtering. This is a general collaborative filtering algorithm that uses distinctions between users to help create features and thetas for other users. The main algorithm implemented for our User-User collaborative filtering was n-nearest neighbors which tried to look for similarities between users in order to make good predictions for unrated memes. With over 150 users and over 4000 user ratings, we separated our data into a training set with 80% of the data and a test set with 20% of our data. With no tunable parameters, we didn’t need a
cross-validation set. Using this, we got an average RMSE of 1.31 over several iterations of the system as seen in the table below. Our training RMSE was at a solid .22 RMSE which meant that we were overfitting. We postulated that the reason for overfitting was our lack of data, however we tried other algorithms to see if we could close this gap.

<table>
<thead>
<tr>
<th>Item-Item Collaborative Filtering</th>
</tr>
</thead>
</table>
| Item-Item collaborative filtering is the same thing as our first algorithm, except instead of using user similarities when running n-nearest neighbors, it uses item (meme) similarities instead. We didn’t think this would address the overfitting too much, but we wanted to test it out. We managed to get a slightly lower average RMSE of 1.30, which is not a significant decrease, but it furthered our belief that this was an overfitting problem due to our lack of data. Below is a graph of the different RMSE’s from this algorithm.
Matrix Factorization Using SVD

Aside from getting more data, we remembered that regularizing features was another way to combat overfitting. We didn’t understand how to regularize a recommender system, so we researched possible ways and found Matrix Factorization. Matrix Factorization uses Singular Value Decomposition (SVD) to lower the dimensionality of the features for the memes. By lowering the dimensionality, it can create matrices of users and items that are in its simplest form. This is the closest one can get to regularizing for a recommender system, so we tried it out. We used the Surprise library for SVD and cross-validation where we used a 5-fold cross-validation due to the tunable parameters for SVD. Using the cross-validation, we were able to get a 1.23 RMSE for our test set which is much lower than the other algorithms we used. However, Matrix Factorization is computationally expensive and took about 15 minutes to run on a Macbook Pro. In real-time, we want our recommender system to work quicker so that it gets smarter as users are rating memes. We don’t think that using Matrix Factorization is feasible since the time it takes for it to run is far larger than the time it takes for a user to rate numerous memes and get bored because the recommendations are not that good. We do think that if we were able to run this on a higher GPU in real-time over and over again, then this may be a feasible option to take for real-time recommendations.

Comparing our Performance to Netflix

Netflix’s famous recommender system for recommending movies performs at around a 0.8 RMSE which is far lower than our best RMSE of 1.23. However, Netflix also has tons more data than we do which is a huge factor in comparing our RMSE’s. We think that although Netflix’s recommender system performs at around 0.8 RMSE, our system’s RMSE is great for just over 4000 user ratings. Also, Netflix is the lower bound for RMSE’s; typical recommender systems perform at 0.90-0.95 RMSE. We ran our user-user recommender system on a dataset with over 100k movie ratings and achieved an RMSE of 0.92. This leads us to believe that with more carefully obtained user ratings, our meme recommender system can perform at the level of industry grade recommender systems.

Real-Time Recommendations

While we are very satisfied with what we have managed to achieve thus far, our ambitions and hopes for this project are almost limitless. For the last few days, we have been focusing our energy on getting Real-Time Recommendations on our website. We are planning to transform our website from a data-gathering platform to a proof-of-concept platform. Not only should our API allow us to gather ratings but, it should also allow the user a taste of what You & Meme will be like in the future. Mainly, we would like the user to rate at least 20 memes then redirect them to our recommender which will then calculate the user’s predicted ratings for all other memes and present the user with the memes it thinks they will love the most. This is a two-fold problem. Our machine learning algorithm is written and almost only available in Python using the Surprise library. We are aiming to achieve asynchronous Python and PHP execution. We have managed to adjust the scripts and to prepare our website to receive the
recommendation from the Python client. However, the Surprise library is not very accessible or available in several platforms. We tried running our client on Reclaim Hosting’s middcreate.net servers and on our CS department’s Basin server with no success on either. Currently, we are setting our PHP script to trigger a signal to the Python client to update the predictions each time the user finishes rating a series of 5 presented memes. Once we manage to install the proper libraries on either Basin or middcreate.net, we would happily start the alpha testing and refinement period over winter break. Consequently, we will keep moving forward to gathering more data, adding more security procedures, refining the user interface and the API, and potentially moving our website to a more resourceful hosting server.

Conclusion

Overall, the meme recommender system was a huge success. Despite our data limitations, we were able to experiment with different Machine Learning techniques and achieve the best RMSE that our system can currently get with the data we have. We were methodical in our process, working on every component diligently and carefully since we knew that any fabrication or bias in our data could have drastic effects on the performance of our recommender system. Our webpage had its ups and downs, but it was able to provide us with user ratings in a clear and fast way. We used known algorithms such as collaborative filtering to train our recommender system and even managed to regularize our features using a new researched method: Matrix Factorization. We think that the best thing to do now is keep advertising to get more data and see if our system begins to perform better. This was a rewarding experience into the world of data collection and using a system for a meaningful and new function; recommending memes. We have worked hard to get this far, and we are not planning on stopping before every “memer” out there is completely satisfied, including ourselves.