Recommendation Engine

billschreiber111, 9 Mar 2018

This article describes a recommendation engine or collaborative filter that is written in C#.

Editorial Note

This article is an entry in our Machine Learning and Artificial Intelligence Challenge. Articles in this sub-section are not required to be full articles so care should be taken when voting.

Download Recommender.Console.zip - 1,011.3 KB

Introduction

This entry is my entry to the "Birds of a Feather" Machine Learning and Artificial Intelligence Challenge. This article describes a recommendation engine or collaborative filter that is written in C#.

Background

This article is my entry to the "Birds of a Feather" competition.

A recommendation engine is software that can predict what a user may or may not like based on previous expressed likes or dislikes. It can be used as an alternative or in conjunction with searches since it helps users discover products or content that they may not have otherwise come across. Recommendation engines are a big part of Amazon, Facebook, movie and many, many content sites across the internet.

The challenge given here was to take a set of data given and to come up with recommendations for users based on that data. I was familiar with the results of recommendation engines or collaborative filters since I use Amazon but had never actually written or used one. I felt it would be an interesting challenge to take on and give me an opportunity to learn more about machine learning. I wasn't sure if in three weeks of working on weekends I could complete it, but the subject was interesting and a challenge.

I started my research on how to design and build a recommendation engine. I read a number of posts, blogs and articles on different approaches to the problem since this is a common problem that many seem to have interest in. The approaches and techniques used in these articles all had several things in common and did two sets of calculations.

First, the similarity between two raters in the system were calculated. Based on the similarities between raters, the probability that a user would like any particular item were calculated. The recommendation engine is based on the idea that if A is similar to B, C is similar to B, A must be similar to C as well. But just because I like an article or several articles that you like, does not necessarily mean that I like every other article that you like. So, the correlation between these data points is weak.

I saw several approaches, but I settled on using this approach.

https://www.toptal.com/algorithms/predicting-likes-inside-a-simple-recommendation-engine

He does a nice job outlining where and how these equations came about. Feel free to read his article to better understand this approach. This approach also gave us a similarity rating between -1.0 and 1.0 with -1 being completely dissimilar and 1 being completely similar. Users with not enough data to calculate a similarity between them will have similarity of 0 and the results will be discarded. I ran some numbers to test his math and verify his formulas which seemed to come out the way I wanted, so I decided to use them as a basis for this article.

Similarity equation

S(U1, U2) = (L1 intersection L2) + (D1 intersection D2) – (L1 intersection D2) – (L2 intersection D1) / (L1 union L2 union D1 union D2)
S(U1, U2) = similarity between user one and user two – a value from -1 to 1 – a zero probability means there isn't enough data to make a calculation
L1 intersection L2 – Number of likes of user one that user two also likes
D1 intersection D2 – Number of dislikes that user one and user two also dislike
L1 intersection D2 – Number likes that user one has that user two dislikes
L2 intersection D1 – Number of likes that user two has that user one dislikes
L1 union L2 union D1 union D2-Number of likes for user one plus number of likes for user two plus number of dislikes for user one plus number of dislikes for user two.

Similarity Equation - using Tags to define similarities within system

I also tried using tags associated to articles as a way to calculate likes and dislikes for all users in the system. I have separated out the code enough that it is fairly easy to plug in a different formula to calculate similarities between users.

Using the currently loaded user's Likes and Dislikes, the number of times each tag occurs in those articles is collated to create the numbers below that are plugged into the similarity formula above. Below, I outline what values are being plugged into those formulas. I include an example of how the various values might be calculated.

As an example of a simple calculation:

So, if a

User1			
	Likes -	Article One	
			Tags = A,B,C
		Article Two	
			Tags-A,B,E,F
User2			
	Likes		
		Article Three	
			Tags=A,B,C
		Article Four	
			Tags=A,G,H
User1 A-2, B – 2,C – 1,E- 1,F -1 Total = 7			
User2 =A-2, B-1, C-1, G-1,H-1 Total = 6			
			Total=13

U1 intersection U2 tags = A,B,C = A-4, B-3, C-2 = 9

S(U1, U2) = (L1 intersection L2) + (D1 intersection D2) – (L1 intersection D2) – (L2 intersection D1) / (L1 union L2 union D1 union D2)
(L1 intersection L2) – This starts by generating the intersection of all tags that were associated with each article that user1 and user2

both liked. The final number is the number of occurrences for those tags for both user1 and user2.
(D1 intersection D2) – This starts with the intersection of all tags that were associated with each article that user1 and user2 both disliked. The final number is the number of occurrences for those tags for both user1 and user2.
(L1 intersection D2) – This starts with the intersection of all tags that were associated with each article that user1 liked and user2 disliked. The final number is the number of occurrences for those tags for both user1 and user2.
(L2 intersection D1) – This starts with the intersection of all tags that were associated with each article that user1 disliked and user2 liked. The final number is the number of occurrences for those tags for both user1 and user2.
(L1 union L2 union D1 union D2)– This starts with the union of all tags that were associated with each article that user1 liked, user1 disliked, user2 liked and user2 disliked. The final number is the number of occurrences for those tags for both user1 liked and disliked, and user2 liked and disliked.

Probability Equation

This generates a value from -1.0 to 1.0 with -1.0 being 100 probability user or rater will dislike the ratee, and 1.0 being 100 % probability user or rater will like the article.

P(U,A) = (Zl – Zd) / (Ml + Md)
P(U,A) – the probability that a user or rater will like a particular ratee or article.
Zl-sum of similarity indices of users or raters who have liked the article
Zd- sum of similarity indices of users who have disliked the article
Ml-total number of users or raters who have liked the article or ratee.
Md-total number of users or raters who have disliked the article or rate.

Using the Code

In this case, we are given a set of users, articles, tags and user actions which include View, Download, UpVote, or DownVote.

Since the example data consists of 3000 users and 3000 articles and each user has to have similarity calculation for each user in the system, and then probability the user will like for each article in the system, that is 18,000,000 calculations per run. So, each of these took a while. I do run the probability calculations in parallel to speed things up which cuts the time roughly by half for those calculations.

Base Classes

RaterBase – base class for any rater within the system. I overload with the User class.

Likes – list of ratees that rater liked
Dislikes - list of ratees that rater disliked
Similarities – list of all users in system with value from -1.0 to 1.0
Ratees – dictionary of ratees with probabilities rater will like from -1.0 to 1.0

RateeBase – base class for any ratee. I overload with the Article class.

Likes- list of raters that liked ratee
Dislikes list of raters that liked ratee

Similarity – saved as a list in the RaterBase::Similarities class.

Rater
Value

UserAction – a user or rater action with in the sytem

Rater -
Ratee
Action – user could have done either UpVote, DownVote, View or Download

RecommenderBase

Raters – all users or raters
Ratees– all ratees or articles
Ratings – list of all UserActions

Likes – all user actions that were likes
Dislikes-all user actions that were dislikes
AddUserActionsToLikes – can add a class of UserActions to data – allows me to add Views and Downloads easily to test dataset
InitializeStatistics – housekeeping method to build stats for later consumption
GenerateSimilaritiesValuesForUsers – pulls values from Likes and Dislike to generate a similarity value between -1.0 and 1.0
GenerateRatingsProbabilitiesForUsers – generates probabilities that each user will like an article with a value between -1.0 and 1.0

Derived Classes

ArticleRecommender : RecommenderBase

Tags – list of all tags-
Downloads – UserAction list of all Downloads
Views – UserAction list of all Views
LoadData(string filename,string actionlike,string actionDislike)

User: RaterBase

Views –Ratees viewed by user
Downloads – ratees downloaded by user
GenerateSimilaritiesByTags-this uses tags to generate similarities to users. It uses their likes and dislikes and counts the number of time each occurs in that list to weight what tags are important.

Tag

Id
Name

Article derives from RateeBase

Tags – tags associated with article
Views – raters who viewed this ratee
Downloads – raters who downloaded this ratee

The basic idea is to load in data to the Likes and Dislikes for users then call `GenerateSimilaritiesValuesForUsers()`, then `GenerateRatingsProbabilitiesForUsers()`. To use tags to correlate on the loaded data, use `GenerateSimilaritiesByTags()` then `GenerateRatingsProbabilitiesForUsers()`. `AddUserActionsToLikes()` and `RemoveUserActionsToLikes()` enables one to add and remove UserActions from the dataset. Refer to the *Program.cs* file for the full listing of all the code involved as it takes the data through a workout.

```
// example usage…

ArticleRecommender recommendationEngine = new ArticleRecommender("Userbehaviour.txt");

// optional to add more UserAction data to test
recommendationEngine.AddUserActionsToLikes("Download");
recommendationEngine.GenerateSimilarityValuesForUsers();
recommendationEngine.GenerateRatingsProbabilitiesForUsers();

// usage to use tags for the similarities correlations
recommendationEngine.GenerateSimilaritiesByTags ();
recommendationEngine.GenerateRatingsProbabilitiesForUsers();
```

Points of Interest

What is interesting is that adding the views and downloads to the overall likes, didn't change substantially the overall correlation or similarities between users. That says to me, that the articles that are viewed and downloaded are different from the articles that users take the time to either like or dislike.

The highest values I got were around 0.25 which is very low. It seems to me that there may likely be something wrong with the way I am calculating similarities although looking through the code and looking at what I am given, it seems the values calculated are accurate (ie the values going into the calculations seem to go along with the article I am using documentation. I ran a number of tests to verify formulas and code and it seems correct). It may be that there aren't a lot of users who do a lot on the site so trying to correlate between users like I have is not the right approach. There may also be a lot of content on the site which is all pretty evenly read, upvoted, downvoted and downloaded. The result is that there are no clear favorites on the site, since most articles have roughly even statistics. That would be an interesting avenue to explore.

I think a more likely fruitful approach is using the tags associated to articles as I have. By weighting the views, I come back with values from 1 to -1 for each variety of cut on the data. There are only 30 tags which makes it easier to correlate which might skew results higher. But, based on my first results, this appears to be the best results garnered yet.

I found this interesting and a lot of fun. If I really want to do more machine learning things, I think I need to get better with Python and/or learn R. C# is a bit cumbersome for doing some of the things I did.

My code isn't as well tested or verified as I would have liked. The concepts are there, but the actual code is probably buggy given the time frame. There are definitely some things I could do to speed things up.

The last few years, I have spent my winters skiing but surgery for torn cartilage in my knee has laid me up the past several weeks. I used the extra time in my schedule to do this challenge. Thank you CodeProject for running this challenge and keeping my mind off of the skiing I am missing!

I hope someone finds it useful and educational.

History

3/4/2018 – first cut

3/9/2018 - added UserBehavior.txt file to project so will compile and use that file

License

About the Author

billschreiber111

United States

I am a nineteen year professional who spent my first 8 years in C and C++, dabbled in Java and has landed the last several years mostly in C# although can't seem to get away from writing Javascript and SQL. My robotics hobby has lead me into learning AI, machine learning, Linux and ROS while playing with Raspberry Pis, Arduinos and ESP8266 wifi chips. I love learning new things and am always fascinated by technology and what these devices can do to make our lives better.

Comments and Discussions

2 messages have been posted for this article Visit **https://www.codeproject.com/Articles/1233227/Recommendation-Engine**

to post and view comments on this article, or click **here** to get a print view with messages.