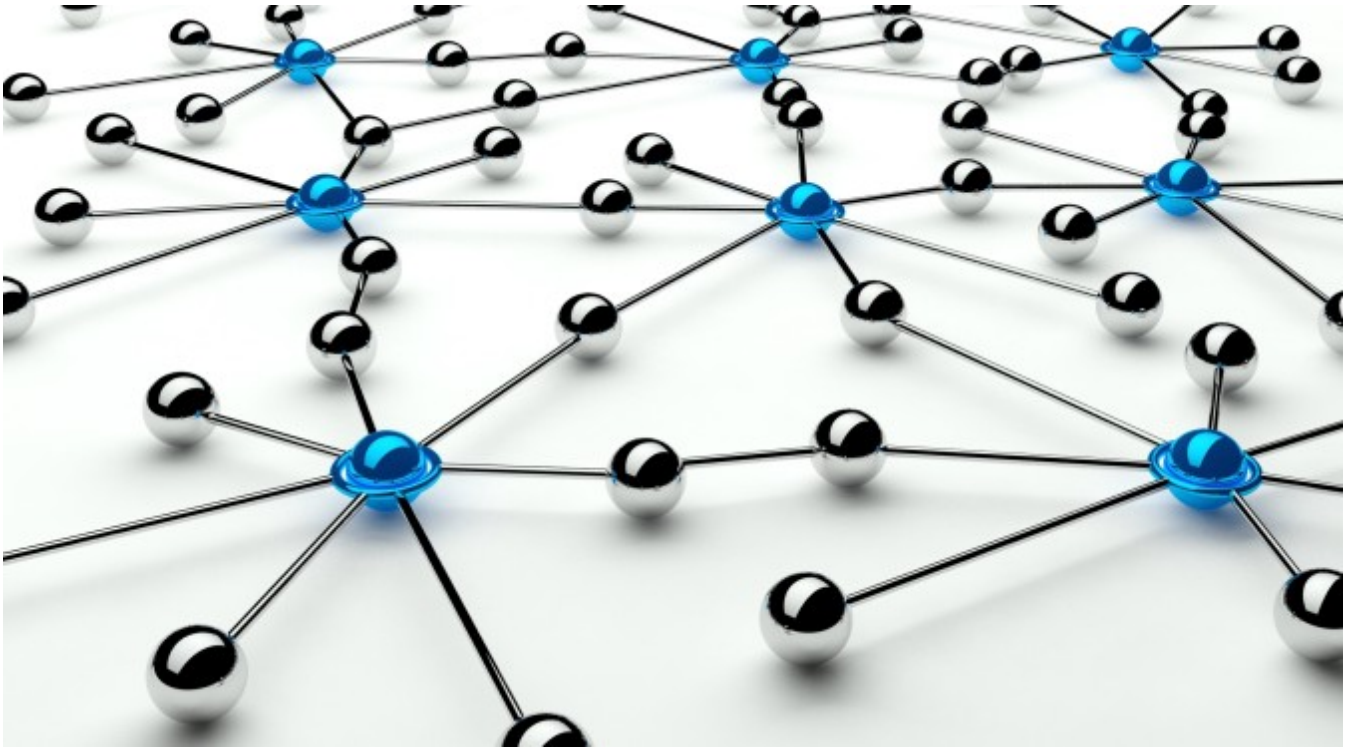
Introduction To Recommendation system In Javascript

Stephen Oni
Apr 18, 2018 · 9 min read



https://consumervaluecreation.com/tag/recommendation-system

I bet you've come across a recommendation system in diverse ways, ranging from a commerce website where you buy goods,like Amazon;social network like Facebook; video/movies site like You-tube and Netflix. These site use the past record of movies,goods and friends to recommend new ones for you.
In this post I will be introducing you briefly on how the system work in javascript. Will we look at different way of representing the case so that they all work with the same set of algorithm and will we create and example with a small dataset on movies critics and using real movie data called MovieLens.

Collaborative Filtering

We all know that the best way to get recommendation for a products,movies is to ask your friends. And I will tell you that the only way, i have ever watch the best movies is through friends that have a good taste for good movies, like Game of throne, which I think was boring at first,was assured to me by a friend that it's interesting ,so I watch it again and see the beauty, and sure that most of us had experienced this before.

What of, if you have a group of friend, who have a good sense for Fashion and you are confused of the cloth to wear to a date. Each one of them make a suggestion on which of the cloth is better than one and suitable for the date. Each one of them have different option and this can confused you the more on which cloth to wear for the date.

As more option become available,then it become less practical to decide what you want by asking a small group of people. This why the technique called collaborative filtering was developed.

Collaborative filtering algorithm skim through a large set of people and finding a smaller set with taste similar to yours. It look at things they enjoy and combine them to create a ranked list of suggestion.

Top 3 Most Popular Ai Articles:

1. Cheat Sheets for AI, Neural Networks, Machine Learning, Deep Learning & Big Data

2. Data Science Simplified Part 1: Principles and Process

3. Getting Started with Building Realtime API Infrastructure

We create a nested object in javascript to store the movies critics , containing different users and the movies they like which is ranked from 1–5.

```
var dataset={
    'Lisa Rose': {
    'Lady in the Water': 2.5,
    'Snakes on a Plane': 3.5,
    'Just My Luck': 3.0,
    'Superman Returns': 3.5,
    'You, Me and Dupree': 2.5,
    'The Night Listener': 3.0},

'Gene Seymour': {'Lady in the Water': 3.0,
    'Snakes on a Plane': 3.5,
    'Just My Luck': 1.5,
    'Superman Returns': 5.0,
    'The Night Listener': 3.0,
    'You, Me and Dupree': 3.5},

    'Michael Phillips': {'Lady in the Water': 2.5,
    'Snakes on a Plane': 3.0,
    'Superman Returns': 3.5,
    'The Night Listener': 4.0},
    'Claudia Puig': {'Snakes on a Plane': 3.5,
    'Just My Luck': 3.0,
    'The Night Listener': 4.5,
    'Superman Returns': 4.0,
    'You, Me and Dupree': 2.5},

    'Mick LaSalle': {'Lady in the Water': 3.0,
    'Snakes on a Plane': 4.0,
    'Just My Luck': 2.0,
    'Superman Returns': 3.0,
    'The Night Listener': 3.0,
    'You, Me and Dupree': 2.0},

    'Jack Matthews': {'Lady in the Water': 3.0,
```

```
'Snakes on a Plane': 4.0,
'The Night Listener': 3.0,
'Superman Returns': 5.0,
'You, Me and Dupree': 3.5},

'Toby': {'Snakes on a Plane':4.5,
'You, Me and Dupree':1.0,
'Superman Returns':4.0}};
```

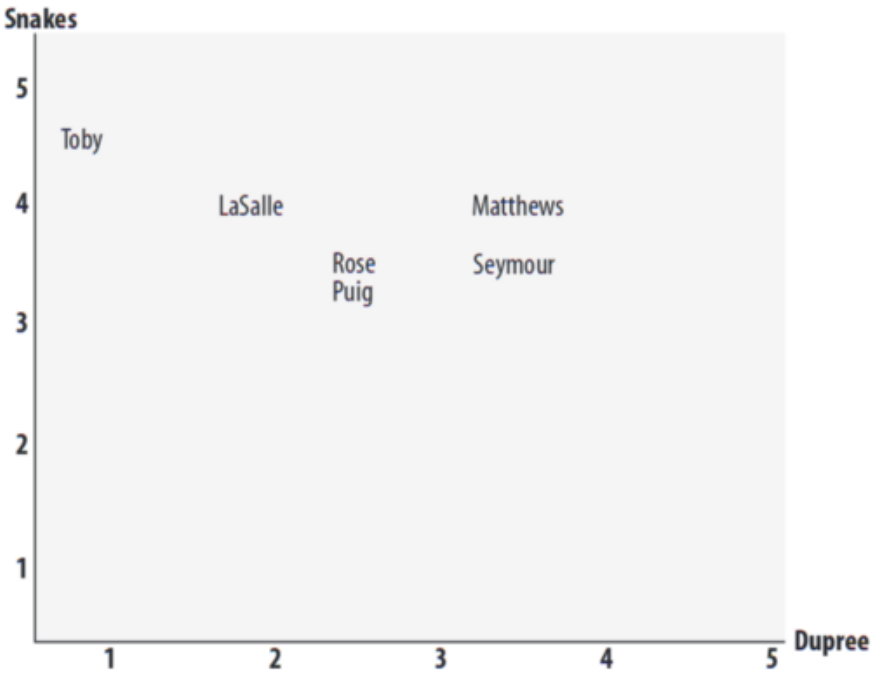
Finding similar user

After we've collect the information of what the users like, we find a way to determine similar people who also like the same thing. We do this by calculating the similarity score which involve comparing the user with all other person. We can either use Euclidean distance or Pearson correlation.

Euclidean distance

$$\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\ldots+(p_n-q_n)^2} = \sqrt{\sum_{i=1}^{n}(p_i-q_i)^2}$$

One of the simplest way to calculate similarity score between two user. It involve representing the data as a point on a chart(x-y axis), and each user having a point e.g from our movie critics we can draw the chart of people who watch `Dupree (x-axis)` and `snakes on the plane(y-axis)` .such has



programming collective intelligence(Toby segaran)

The figure above show that `Rose` has been plotted at `3.5` on the `snake` axis and `2.5` of `Dupree` axis. The closer a set of people are in the preference space the closer their preference are.since the chart is two dimensional you can only look at few ranking at a time

for example to calculate the distance between Lisa Rose and Mathew

```
var euclid = Math.sqrt(Math.pow(3.5-2.5,2)+Math.pow(4.0-3.5,2));
//1.118033989
```

This calculate the distance, which will be smaller for people who are more similar. since we want the value to be high for similar people and the value should range between 0 to 1. so similar user will have a value close to 1.Hence we change the way we implement the formula.

```
var reuclid = 1/(1+euclid);
//0.472135954
```

so let create a function to calculate the euclidean distance for our movie critics

```
//calculate the euclidean distance btw two item
var euclidean_score  = function(dataset,p1,p2){

    var existp1p2 = {};//store item existing in both item

//if dataset is in p1 and p2
    //store it in as one
    for(var key in dataset[p1]){
        if(key in dataset[p2]){
            existp1p2[key] = 1
        }

if(len(existp1p2) ==0) return 0;//check if it has a data

var sum_of_euclidean_dist = [];//store the  euclidean distance

        //calculate the euclidean distance
        for(item in dataset[p1]){
            if(item in dataset[p2]){
```

```
                    sum_of_euclidean_dist.push(Math.pow(dataset[p1]
[item]-dataset[p2][item],2));
                }

}
        var sum=0;
        for(var i=0;i<sum_of_euclidean_dist.length;i++){
            sum +=sum_of_euclidean_dist[i]; //calculate the sum of
the euclidean
        }

//since the sum will be small for familiar user
        // and larger for non-familiar user
        //we make it exist btwn 0 and 1

        var sum_sqrt = 1/(1 +Math.sqrt(sum));

    return sum_sqrt;
    }

}
```

we create an helper function to calculate the length of an object

```
var len  = function(obj){
    var len=0;
    for(var i in obj){
        len++
    }
    return len;
}
```

Let compute the overall euclidean distance between Lisa Rose and Mathews

```
euclidean_score(dataset,"Lisa Rose","Jack Mathews");

//0.3405424265831667
```

Pearson Correlation score

The correlation coefficient is a measure of how well two set of data fit into a straight line.Although the formula is more complex to Euclidean score but gives a better result where data isn't normalized

$$r = \frac{\sum XY - \frac{\sum X \sum Y}{N}}{\sqrt{\left(\sum X^2 - \frac{(\sum X)^2}{N}\right)\left(\sum Y^2 - \frac{(\sum Y)^2}{N}\right)}}$$

```
var pearson_correlation = function(dataset,p1,p2){

var existp1p2 = {};

for(item in dataset[p1]){
            if(item in dataset[p2]){
                existp1p2[item] = 1
            }
        }
        var num_existence = len(existp1p2);

if(num_existence ==0) return 0;

//store the sum and the square sum of both p1 and p2
        //store the product of both
        var p1_sum=0,
            p2_sum=0,
            p1_sq_sum=0,
            p2_sq_sum=0,
            prod_p1p2 = 0;
        //calculate the sum and square sum of each data point
        //and also the product of both point
        for(var item in existp1p2){
            p1_sum += dataset[p1][item];
            p2_sum += dataset[p2][item];

p1_sq_sum += Math.pow(dataset[p1][item],2);
            p2_sq_sum += Math.pow(dataset[p2][item],2);

prod_p1p2 += dataset[p1][item]*dataset[p2][item];
        }
        var numerator =prod_p1p2 - (p1_sum*p2_sum/num_existence);

var st1 = p1_sq_sum - Math.pow(p1_sum,2)/num_existence;
        var st2 = p2_sq_sum -Math.pow(p2_sum,2)/num_existence;

var denominator = Math.sqrt(st1*st2);

if(denominator ==0) return 0;
        else {
            var val = numerator / denominator;
            return val;
        }

}
```

The function will return a value between -1 and 1.A value of -1 means they are not close or similar, and a value 1 means that the two people have exactly the same rating for every item.

```
pearson_correlation(dataset,'Lisa Rose','Jack Matthews')
//0.7470178808339965
```

Ranking the Critics

After the similarity score has been created,we create a function to ranked user in the movie critic in relation to the user we want i.e find other user that have the same taste of movie .

```
var similar_user = function(dataset,person,num_user,distance){

var scores=[];

for(var others in dataset){
        if(others != person && typeof(dataset[others])!="function"){
            var val = distance(dataset,person,others)
            var p = others
            scores.push({val:val,p:p});
        }
    }
    scores.sort(function(a,b){
        return b.val < a.val ? -1 : b.val > a.val ? 1 : b.val >=
a.val ? 0 : NaN;
    });
    var score=[];
    for(var i =0;i<num_user;i++){
        score.push(scores[i]);
    }

return score;

}
```

This function check if the user is not compare to itself when comparing the users.Then it return an `n` number of user of the sorted result.

```
similar_user(dataset,'Jack Matthews',3,pearson_correlation);
//[ { val: 0.963795681875635, p: 'Gene Seymour' },
```

```
{ val: 0.7470178808339965, p: 'Lisa Rose' },
{ val: 0.66284898035987, p: 'Toby' } ]
```

From this we know that `Jack Matthews` should be reading movie reviews by `Gene Seymour` ,since their taste are similar.

Recommending Items

What we just created is not what user will actually want to be engage in. for instance we can just expect user like `Jack Matthews` to start searching for movies that `Gene Seymour` had watched, that will be too tedious,apart from that it could also return reviewer who strangely liked a movie that got bad reviews from all other critics return by `similar_user` .

To solve these problem we need to score the items by producing weighted score that ranks the critics, then take the votes of all other critics and multiply how similar they are to the user we want by the score they gave each movie

Critic	Similarity	Night	S.xNight	Lady	S.xLady	Luck	S.xLuck
Rose	0.99	3.0	2.97	2.5	2.48	3.0	2.97
Seymour	0.38	3.0	1.14	3.0	1.14	1.5	0.57
Puig	0.89	4.5	4.02			3.0	2.68
LaSalle	0.92	3.0	2.77	3.0	2.77	2.0	1.85
Matthews	0.66	3.0	1.99	3.0	1.99		
Total			12.89		8.38		8.07
Sim. Sum			3.84		2.95		3.18
Total/Sim. Sum			3.35		2.83		2.53

programming collective intelligence(Toby)

The table above shows the correlation score for each critics and the rating they gave the three movies. The above table is a score matrix for `Toby` . since Toby has not rated some movies like `(The Night Listener,Lady in the Water, and Just My Lick)` . win the column `S.x` we weight every rating by multiplying it with their similarity to `Toby` . Then we divide the total of each `S.x` column by the total of the Similarity score of user that rate such movie.

The reason for dividing, is because a movie reviewed by more people would have a big advantage. for example `The Night Listener` was reviewed by everyone,so it total is

divided by the sum of all the similarities.While `Lady in the Water` was not reviewed by `Puig` ,so the movie is not divided by the sum of all other similarity.

```
var recommendation_eng = function(dataset,person,distance){

    var totals = {
        //you can avoid creating a setter function
        //like this in the object you found them
        //since it just check if the object has the property if not
create
        //and add the value to it.
        //and  because of this setter that why a function property
        // is created in the dataset, when we transform them.

        setDefault:function(props,value){
            if(!this[props]){
                this[props] =0;
            }

this[props] += value;
        }
    },
        simsum = {
            setDefault:function(props,value){
                if(!this[props]){
                    this[props] =0;
                }

                this[props] += value;
            }
        },
        rank_lst =[];

for(var other in dataset){

if(other ===person) continue;

var similar = distance(dataset,person,other);

        if(similar <=0) continue;

for(var item in dataset[other]){
                if(!(item in dataset[person]) ||(dataset[person]
[item]==0)){
                 //the setter help to make this look nice.
                    totals.setDefault(item,dataset[other]
[item]*similar);
                    simsum.setDefault(item,similar);


                }

        }
```

```
    }

   for(var item in totals){
  //this what the setter function does
  //so we have to find a way to avoid the function in the object
        if(typeof totals[item] !="function"){

            var val = totals[item] / simsum[item];
            rank_lst.push({val:val,items:item});
        }
    }
    rank_lst.sort(function(a,b){
        return b.val < a.val ? -1 : b.val > a.val ?
        1 : b.val >= a.val ? 0 : NaN;
    });
    var recommend = [];
      for(var i in rank_lst){
        recommend.push(rank_lst[i].items);
     }
   return [rank_lst,recommend];
}
```

Item/Content Based Filtering

In the next tutorial we learn how to know which customer/user s will or should use the item next, and we practice with the MovieLens dataset.

Based on programming collective Intelligence by Toby Segaran