Personalized News Recommender System

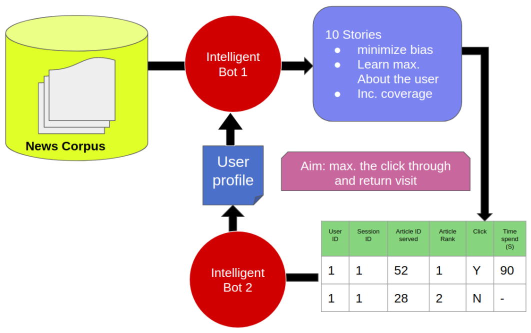
Machine Learning - Final Report

Asrith Krishna(Ms16033)

June 1, 2020

1 Introduction

Recommender systems are used at numerous platforms like shopping (Amazone), movies(Netflix), music(Spotify), and the list go on, personalizing our web experience. When loads of information is available online, it will require an extensive search to find what we really want, a reliable recommender system makes our job easier by showing us the most relevant items and thus avoid the extensive search.
In this project, we are assigned to build a recommender system for news articles. Our task is to create a model which can categorize the corpus of news stories into classes like sports, technology, fashion, politics etc. We can display 10 stories on the screen without a scroll. In the first visit we need to reduce bias in our data display and maximise coverage of the news corpus. Then in the first visit once the user consume the news we need to create the user profile by collecting some data as per the table given.Then from the second use onward we should use the user profile we have created to feed the user with the news from the area of interest of the user.



User ID	Session ID	Article ID served	Article Rank	Click	Time spend (S)
1	1	52	1	Y	90
1	1	28	2	N	-

Figure 1: A pictorial representation of the problem

We obtained data that is about 10K news articles from scroll.in. We used BeautifulSoup for web scraping and generated a corpus of news stories for the recommender system. We have a sequence of words which make up the document, we need to vectorize these word sequences to be able to do some machine learning on it. One of the popular models is the Bag-of-word approach. In the BoW model firstly we define a vocabulary with fixed size which is the set of all unique words available in all the documents of the corpus. The size of the vector equals the size of the vocabulary. Then, for expressing a text document using this vector, we count how many times each word of our vocabulary appears in the text document and we put this number in the corresponding vector entry. Now we know that words like 'is', 'was', 'the', etc repeat very frequently in a document. We create a list of these words called the stopword and get rid of them from our vector, thus now the size of our vector is reduced.

Term frequency-inverse document frequency (TF-IDF) is a well-known and the most common representation for text vectorization. It is used to estimate how significant a word is to a document in a corpus. So here we did the TF-IDF vectorization and convert the text data in the vector form, the data in the array form appears to be simple and quick to use.We used k-means clustering to cluster the dataset into 10 categories to recommend the 10 different news articles, one from each category to the new user.

2 Click-stream data

We are constrained by the fact that we show only 10 stories at a time. And our objective with those 10 stories is to minimize bias, learn maximum about the user interests, and to maximize coverage. To recommend the news articles to the new user, we create intelligent bot 1 which recommends 10 news stories. To minimizes the bias we can use the information of the previous user interest and purge the unused articles by the new articles from that category. And to maximize the coverage we recommend the different articles for the new user so that we can use the maximum from our corpus.

For creating a "User Profile" we need the clickstream as given below:

User ID	Session ID	Article ID Served	Article Rank	Click	Time Spend(s)

The challenge here is that we are generating the click-stream data using probability distribution functions.We thought how each of the dimensions of our click-stream data can be modelled using a suitable function to match the real world user profile.
We assume that we have three types of users: Type I- land and bounce, Type II- land and browse for some time, and Type III - land and browse for a long time. The users who are feeding on our data for the first time are users with session-id:1, they are our first time users. After recommending them news from 10 categories, we will randomly generate a click for each article as we do not have prior data about user interests. For a time-spent generation, we used a mixture of normal distribution where we assume that most of the first time users are either from type-I and type-II. Here there is a rare chance of meeting type-III. Thus in this way, if there is a click for a category, we will get time spent, article rank, article id for it, and the same way we get above data(exclude time spent)

for not clicking a category in a CSV file.

From the set of first-time users, we assume that a portion of users will visit again. We predicted whether a user will visit again or not by choosing a suitable threshold time. If a user's total time-spent is above the threshold then they will visit again otherwise not. From the data collected for the first-time users, we calculated the user interest in each category

$$u_i = \frac{(\frac{t_i}{N_i})}{\sum_i(\frac{t_i}{N_i})}$$

t_i =time spent in each category, N_i =no of clicks in each category.

User ID	Click	Time Spend(s)	Session ID	Article Rank	Article ID Served	Rating
1.0	1	9.556711	1.0	0	2063	0
1.0	1	3.489446	1.0	1	3435	0
1.0	1	45.781897	1.0	2	4772	4
1.0	0	0.00000	1.0	3	5148	0
1.0	0	0.00000	1.0	4	5350	0
1.0	1	36.20536	1.0	5	5794	3

For later visits of the user that is for 2^{nd} and 3^{rd} visit, we used a sigmoid curve to generate the data points for click as we expect the click rate should go high or stabilize for each category. Then we randomly choose a point from it and if it goes above 'u_i' then we will get a click otherwise not. For time-spent, we will generate data from normal Gaussian as we assume most of the users will spend moderate time feeding on the article and belongs ti type-III users.

3 The Article Recommender System

3.1 Item-based collaborative filtering using similarity between items

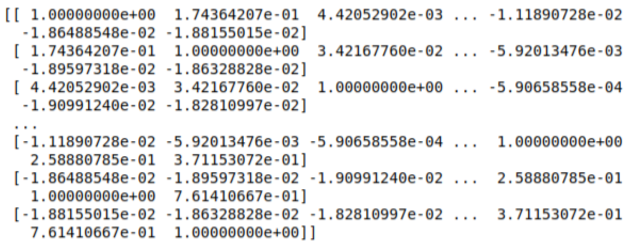
The similarity values between the two documents are measured by observing all the readers who have rated both the documents.
Now let's assume we have a set U of users and set D of news documents and R be the rating matrix of size u × d.

Cosine similarity : $sim(u_i, u_k) = \frac{r_i . r_k}{\|r_i\|\|r_k\|} = \frac{\sum_{j=1}^{d} r_{ij} r_{kj}}{\sqrt{\sum_{j=1}^{d} r_{ij}^2 \sum_{j=1}^{d} r_{kj}^2}}$

Pearson (correlation) similarity : $sim(u_i, u_k) = \frac{\sum_j (r_{ij}-r_i)(r_{kj}-r_k)}{\sqrt{\sum_j (r_{ij}-r_i)^2 \sum_j (r_{kj}-r_k)^2}}$

Using centered cosine similarity for our article recommender system was not a beneficial approach because it resulted in poor rating predictions, i.e with very low accuracy.

On applying the centered cosine similarity on our dataset we obtain the matrix shown below:



```
[[ 1.00000000e+00  1.74364207e-01  4.42052902e-03 ... -1.11890728e-02
  -1.86488548e-02 -1.88155015e-02]
 [ 1.74364207e-01  1.00000000e+00  3.42167760e-02 ... -5.92013476e-03
  -1.89597318e-02 -1.86328828e-02]
 [ 4.42052902e-03  3.42167760e-02  1.00000000e+00 ... -5.90658558e-04
  -1.90991240e-02 -1.82810997e-02]
 ...
 [-1.11890728e-02 -5.92013476e-03 -5.90658558e-04 ...  1.00000000e+00
   2.58880785e-01  3.71153072e-01]
 [-1.86488548e-02 -1.89597318e-02 -1.90991240e-02 ...  2.58880785e-01
   1.00000000e+00  7.61410667e-01]
 [-1.88155015e-02 -1.86328828e-02 -1.82810997e-02 ...  3.71153072e-01
   7.61410667e-01  1.00000000e+00]]
```

Figure 2:

3.2 Matrix Factorisation using Stochastic gradient descent (SGD)

In a news recommender system, there is a set of users and a set of news articles. The rating given by the user to some item in the system can be used to predict how the users would rate the items that they have not yet rated, such that we can make recommendations to the users. We assume that there exist certain latent features that determine how a user rates an item and the number of features would be smaller than the number of users and the number of items.

Now let's assume we have a set U of users and set D of news documents and R be the rating matrix of size U $\times$ D. We assume that there are K latent features. Now our task is to find matrix P(size U $\times$ K) and Q (size D $\times$ K) such that their product approximates R. The rating prediction of an item d_i by user u_i can be calculated as : $e_{ij}^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik}q_{kj})$

Regularization is done to by adding a parameter β and modify the squared error to avoid overfitting.

$$e_{ij} = (r_{ij} - \sum_{k=1}^{K} p_{ik}q_{ki})^2 + \frac{\beta}{2}\sum_{k=1}^{K}(\|P\|^2 + \|Q\|^2)$$

Formula to update p_{ik} and q_{kj} :

$$p_{ik}^{'} = p_{ik} + \alpha(2e_{ij}q_{kj} - \beta p_{ik})$$

$$q_{kj}^{'} = q_{kj} + \alpha(2e_{ij}p_{ki} - \beta q_{kj})$$

where α is a constant that determines the rate of approaching the minimum
β is used to control the magnitudes of the user-feature and item-feature vectors.

In the work we have performed we used K=100, α = 0.1, β = 0.01 and got the total mean square error = 7.5679.

3.3 Collaborative filtering using Singular Value Decomposition (SVD)

Any real matrix R can be decomposed into 3 matrices U, Σ, and V. Here in our news recommender system U is an n $\times$ k user-latent feature matrix, V is an m

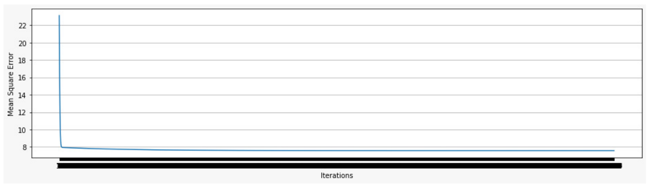


Figure 3: Mean square error vs iteration plot for SGD

$\times$ k news article-latent feature matrix.   is a k $\times$ k diagonal matrix containing the singular values of the original matrix, representing how important a specific feature is to predict user preference.

$$R = U\Sigma V^T$$

4 Discussion and Conclusion

In this report, we present our research on developing an efficient news recommender system. We started by generating a clickstream dataset for the user profile.

To build a news recommender system we have tried some of the popular approaches. We tried using cosine similarity approach but it was observed that it was not predicting higher ratings accurately. We further examined the Matrix Factorisation approach using Stochastic gradient descent (SGD). Earlier using cosine similarity we got coverage of 450 out of 10,000 which is a weak coverage, SGD could give better coverage of 956 articles. We also worked with SVD.