Building Content-Based Recommender System

(For an article based platform)

- **Understanding Content Based Recommendations :**

  i) For these kinds of recommender systems the data of the article is used for calculating the ***cosine similarity*** by creating a ***vector-space model*** between two different articles. The value of this cosine similarity indicates the degree of similarity between two articles. It can range from -1 to +1, where +1 indicates the articles to be exactly similar to each other and vice-versa.

  ii) To calculate this correlation the concept of ***tf-idf*** is used. Where :

     - ***Term Frequency*** – This measures how frequently a term occurs in a document. It is defined as the no. of times a term occurs in a particular document. Mathematically :
       TF (t) = (# of times t occurs in the doc) / (Total no. of terms in the doc)
     - ***Inverse Document Frequency*** – This measures how important a term is. Since we need to weigh up the effects of less frequently occurring terms and weigh down the more frequently occurring terms (like : if, that etc.) we use logarithms. Mathematically:
       IDF (t) = log_e (Total no. of docs / no. of docs with term t in it)
     - Tf-Idf value of a particular term in an article will be = TF * IDF
     - Calculating the tf-idf of each term in an article gives us the ***feature vector*** of the respective article. A feature vector can be viewed as a vector representation of a particular article in the vector-space model.

  iii) Next step would be calculating the cosine-similarity between two articles with the help of their feature vectors (f1, f2). Cosine of two vectors in a vector space model is defined as the dot product between the two vectors divided by the product of their magnitude. Mathematically:

  Cos Ø = (f1.f2) / (|f1| * |f2|)

- **Building the Actual Model :**

  i) The first step would be to create the ***corpus*** of all the articles and a ***bag of words*** which contains all the unique words across all articles. The size of this bag of words will be the dimensionality of all the tf-idf vectors.

  ii) Then for all the articles present in the corpus evaluate and assign to them their respective tf-idf feature vector. Let's call this as a ***Document Object.***

  iii) ***Recommending Articles***: For an individual user obtain a list of document objects (articles) read by that user. Calculate the Cosine Similarity of each document in the list with the rest of the documents of the corpus not present in the list.

iv) ***Filter*** out the cosine similarity comparisons: E.g. if a1 was compared to D and a2 was compared to D (where a1, a2 are the articles from the user's history) then eliminate the comparison with D which resulted in the lower score.

v) ***Arrange*** all the article comparisons in the decreasing order of their respective similarity score and recommend the articles from the top of the list.

- **Techniques to Improve the Accuracy :**

i) Building a ***Hybrid Recommender System*** using Collaborative Filtering : If you can gather user response for articles like a rating from 0-5 or a like/dislike option we can use the concept of ***collaborative filtering*** along with content based filtering for higher accuracy.

    - Implementing ***user-user similarity***: Using the information how one user rated a particular article can be used to recommend a user with other users with similar tastes. This is achieved by calculating the similarity using ***Pearson Correlation*** coefficient.
    - If we don't wish to interfere with the user experience we can use ***boolean values*** instead of ratings to develop a similar structure. Like, did the user click on the article, did the user spend sufficient time to completely read the article, did the user share the article.

ii) ***Dimensionality Reduction*** of Bag of Words: Since the bag of words approach leads to a large size of the vectors, when a tf-idf vector is created for an article it is very sparsely populated. That is, a significant no. of terms have tf-idf value=0. We can use techniques like ***Latent Semantic Indexing*** to reduce the dimensionality of tf-idf vector***.***

iii) Instead of taking one term at a time we can create the tf-idf vector by using ***ngrams***.
E.g. :
"The many applications of machine learning."
Unigrams – The, many, applications, of, machine, learning
Bigrams – The many, many applications, applications of, of machine, machine learning.
And so on and so forth. We can test our model on various ngrams and choose the one which performs the best. For e.g. the user may prefer articles that contain the term "***Machine Learning***" rather than the articles which contain "Machine" or "Learning". This approach generally leads to a better accuracy in such cases.

iv) ***Pre-Processing*** the text: Before creating our bag of words we can clean our text by removing words which generally don't determine the preference of a user, for e.g. removing the stop words. We can further enhance this by using the ***Natural Language Processing*** techniques to eliminate organization names, verbs, noun phrases etc. from the articles as per our discretion.

v) Use ***meta data*** of the articles like the genre along with the user boolean response : For e.g. :

Articles	Technology	Art	Politics
Article1	1	1	0
Article2	0	0	1
Article3	0	1	1

	User1	User2
Article1	1	-1
Article2	1	1
Article3	0	1

- ***Normalize the attributes***: We can perform normalization of attributes by dividing the term occurrence (1/0) by the square root of number of attributes in the article. E.g. for Article1 normalized attribute value = 1/√2.
- Generate ***User Profile Vectors***: Calculate the dot product between the user ratings vector and the genre vector to evaluate the score of user preference for each genre.
- E.g. : The dot product between User1 and Technology vector gives the result = $(1 \div \sqrt{2})$ *1 + 0*1 + 0*0 = .707. This is the score of the preference of User1 for Technology related articles. Similarly, other scores can be calculated.

- **Recommending Content to a New User :**

i) This has been a perennial problem in the field of Recommender Systems and is in fact a hot topic of research these days. This problem is often referred to as the ***cold start*** problem.

ii) Although there are ways to recommend items to a new user it is often discouraged to do so because of a high probability of making an inappropriate recommendation.

iii) Other incentives like coupons, free subscription, etc. can be offered to a new user to engage them on the platform till sufficient data on them can be gathered.

iv) If you still wish to recommender the user something following techniques can be used :

a) Recommend the most popular articles in various genres.

b) Gather the meta-data of the user like age, gender, country etc. and compare with the meta-data of the existing users. Then recommend the articles most popular within that section of the viewership.

c) Provide the user with prompts of various articles and observe the users' actions. Whether the user skips it, clicks on it, if the user clicks on it then how much time does the user spend on it, etc. to gather data.

- **Technologies Used :**
  i) Although the algorithm is simple enough to be implemented without the use of any libraries, for ease in coding and implementation of further advanced techniques for improved accuracy we can use ***Python*** along with the following libraries:
     a) ***NumPy*:** A Python library suitable for mathematical computation of matrices, arrays, etc.
     b) ***Scikit-Learn*:** A machine learning oriented for Python. It can be used to create a recommender model which will involve creating feature vectors, calculating similarity and recommending items with relative ease.
     c) ***Gensim:*** A Python module to implement dimensionality reduction on the vector space model. It implements 3 methods for the same –
        - ***Latent semantic indexing*** – It is Principal Component Analysis for binary or multinomial data.
        - ***Latent dirichlet allocation*** – Uses Bayesian techniques to sparser data than LSI.
        - ***Random projections*** – It has less computational requirements and sometimes can provide accuracy as good as LSI or LDA.

- **Algorithm for Different Calculations:**
  i) Create a class '***Document***' :
     Attributes:
     - Text
     - Fields for Meta-Data
     - Tf-Idf Vector<Float>

  ii) Create a class '***Corpus***' :
     Attributes:
     - List<Document>

  iii) Create a class '***User***':
     Attributes:
     - Fields for Meta-Data
     - List<Document >

  iv) Evaluating ***tf-idf Vector*** for each Document 'd' in Corpus 'C' :

```
Loop in 'C':
    For 'd' in 'C' loop in its terms:
            For T in Bag of Words:
                    If T is in 'd':
                            For term 't' in 'd':
                                    If t==T:
```

```
                              Tf = count of ‘t’ in ‘d’ / total no. of terms in ‘d’
                              Count =0
                              Loop in ‘C’:
                                  If ‘ d’ ’ contains ‘t’:
                                      Count = count+1
                              Idf = log_e(size of ‘C’ /count)
                              Tfidf = Tf*Idf
                              d.tf-idf vector.add(Tfidf)
                  else :
                      d.tf-idf vector.add(0)
```

v) Evaluating ***Dot Product*** :

Query Doc: (Q)

Refer Doc: (D)

```
Dot product=0
Index=0
For val1 at index in Q.tf-df vector:
    For val2 at index in in D.tf-idf vector:
        Dot product = dot product + val1*val2
```

vi) Evaluating ***Vector Magnitude*** :

Document: D

```
Magnitude=0
Square sum=0
For val in D.tf-idf vector:
    Square sum = square sum + $val^2$
Magnitude = √square sum
```

vii) Evaluating ***Cosine Similarity*** :

Query Doc – Q

Refer Doc – D

```
Dot product = dot product (Q, D)
||Q|| = Vector Magnitude (Q)
||D|| = Vector Magnitude (D)
Similarity = Dot product / (||Q|| * ||D||)
```