MODULE 4: RECOMMENDATION SYSTEMS

CASE STUDY ACTIVITY TUTORIAL

CASE STUDY 3 – NEW PRODUCT RECOMMENDATION

CASE STUDY ACTIVITY TUTORIAL

CASE STUDY 3 – NEW PRODUCT RECOMMENDATION

Problem: Make new Product Recommendations (e.g. Amazon.com)

- **Dataset**: Amazon Reviews data (http://jmcauley.ucsd.edu/data/amazon/)
    - You will need to seek permission to use any/all of the datasets in this repository
    - The repository has several datasets. You can choose any. For this example we will use the Electronics dataset.
    - The host page has several pointers to scripts and other examples that can help with parsing the datasets.
    - The data set consists of:
        - 7,824,482 Ratings (1-5) for Electronics products.
        - Other metadata about products. Please see the description of the fields available on the webpage cited above.
    - For convenience of future use, parse the raw data file (using Python, for example) and extract the following fields: 'product/productId' as **prod_id**, 'product/title' as **prod_name**, 'review/userId' as **user id**, 'review/score' as **rating**
    - Save these to a tab separated file. We will refer to this file as **product_ratings.csv**
- **Read/View the Dataset:**
    - The first task is to explore the dataset. You can do so using a programming environment of your choice, e.g. Python or R.
    - In R, you can read the data by simply calling the read.table() function:

      ```
      data = read.csv('product_ratings.csv')
      ```

        - You can rename the column names as desired:

          ```
          colnames(data) = c("prod_id",  "prod_name", "user_id", "rating"
          ```

        - Now you can look at the data properties by using:

          ```
          str(data)
          summary(data)
          ```

        - Plot a histogram of the data:

          ```
          hist(data$rating)
          ```

    - In Python, you can convert the data to a pandas dataframe to organize the dataset (http://pandas.pydata.org/)

      For plotting in Python, you can use MatplotLib: http://matplotlib.org/
    - The dataset sparsity can be calculated as:

      Sparsity = <Number of Ratings in the Dataset> / (Number of Products x Number of Users) * 100%

- **Sub-setting the data:**
  - If you want the data to be less sparse, for example, a good way to achieve that is to subset the data where you only select Users/Products that have at least a certain number of observations in the dataset.
  - In R, for example, if you wanted to subset the data such that only users with 50 or more ratings remained, you would do the following:

    ```
    data = data[ data$user_id %in%
    names(table(data$user_id))[table(data$user_id) > 50] , ]
    ```

- **Recommenders:**
  - If you want to build your own Recommenders from scratch, you can consult the vast amounts of academic literature available freely. There are also several self-help guides which can be useful, such as these:

    http://www.salemmarafi.com/code/collaborative-filtering-r/ ,

    http://blogs.gartner.com/martin-kihn/how-to-build-a-recommender-system-in-python/

  - On the other hand, why build a recommender from scratch when there is a vast array of publicly available Recommenders (in all sorts of programming environments) ready for use? Some examples are:
    - RecommenderLab in R (https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf) ,
    - Graphlab-Create for Python (has a free license for persona and academic use)(https://dato.com/products/create/docs/graphlab.toolkits.recommender.html),
    - Apache Spark's Recommendation module (https://spark.apache.org/docs/1.4.0/api/python/pyspark.mllib.html#module-pyspark.mllib.recommendation),
    - Apache Mahout (https://mahout.apache.org/users/recommender/userbased-5-minutes.html)

- **Splitting Data Randomly (Train/Test):**
  - A random split can be created in R and Pandas (Python) .
    - In R, you can do the following to create a 70/30 split for Train/Test:

      ```
      library(caTools)
      spl = sample.split(data$rating, 0.7)
      train = subset(data, spl == TRUE)
      test = subset(data, spl == FALSE)
      ```

- In Pandas, using the SciKit-Learn library:

```
import pandas as pd
import numpy as np
from sklearn.cross_validation import train_test_split

# assuming pdf is the pandas dataframe with the data
train, test = train_test_split(pdf, test_size = 0.3)
```

  - Alternatively, one can use the Recommender libraries (discussed earlier) to create the data splits.
    - For RecommenderLab in R, the documentation in Section 5.6 provides examples that will allow random data splits (https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf)
    - Graphlab's Sframe objects also have a random_split function which works similarly (https://dato.com/products/create/docs/generated/graphlab.SFrame.random_split.html)

- **Popularity Recommender:**
  - The RecommenderLab in R, for example, provides a popularity recommender out of the box. Section 5.5 of the RecommenderLab guide provides examples and sample code to help do this: https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf
  - GraphLab-Create also provides a Popularity Recommender (in python). If the dataset is in Pandas, it can easily integrate with GraphLab's Sframe datatype as noted here: https://dato.com/products/create/docs/generated/graphlab.SFrame.html. Some more information on the Popularity Recommender and its usage is provided here: https://dato.com/products/create/docs/generated/graphlab.recommender.popularity_recommender.PopularityRecommender.html

- **Collaborative Filtering:**
  - Most recommender libraries will provide an implementation for Collaborative Filtering methods. The RecommenderLab in R and GraphLab in Python both provide implementations of Collaborative Filtering methods.
    - In RecommenderLab, use the "UBCF" (user-based collaborative filtering) to train the model.
    - In GraphLab, use the "Factorization Recommender" (https://dato.com/products/create/docs/generated/graphlab.recommender.factorization_recommender.FactorizationRecommender.html)

- Often, a regularization parameter is used with these models. The best value for this regularization parameter is chosen using a Validations set. Here is how this can be done:
    - Split the Test set further 75%/25% in to Train/Validation sets.
    - Now we have three sets: Train, Validation, Test.
    - Train several models, each using a different value of the regularization parameter (usually in the range: (1e-5, 1e-1).
    - Use the Validation set to determine which model results in the lowest RMSE.
    - Use the regularization value that corresponds to the lowest Validation set RMSE.
    - Finally, with that parameter value fixed, use the trained model to get a final RMSE value on the Test set.
    - In R and Python, it can also help plotting the Validation set RMSE values vs the Regularization parameter values to determine the best one.

- **Evaluation (RMSE):** Once the model is trained on the Training data, it can be used to compute the error (RMSE) on predictions made on the Test data.
    - RecommenderLab in R uses the **predict()** and **calcPredictionAccuracy()** functions to compute the predictions (based on the trained model) and evaluate RMSE (and MSE and MAE).
    - Graphlab in Python also has a **predict()** function to get predictions. It provides a suite of functions to evaluate metrics such as rmse (**evaluation.rmse()**, for example).
- **Top-K Recommendations:**
    - Since our goal is to recommend new products to each user based on his/her habits, we will recommend K new products.
    - Based on scores assigned to User-Item pairs, each recommender algorithm makes available functions that will provide a sorted list of top-K items most highly recommended for each user (from among those items **not** already rated by the user).
        - In RecommenderLab, the parameter **type='topNlist'** to the **evaluate**() function will produce such a list.
        - In GraphLab, the **recommend(K)** function for each type of recommender will do the same.