WID2002 Computing Maths II

Semester 2 (2022/2023)

Group Project Report - Student's

Productivity Apps Recommendation System

Lecturer: Dr. Saw Shier Nee

Tutorial: Occurrence 2 (**Group Apple Pi**)

NAME	MATRIC NUMBER
Samiha Tasnim Dristy	**S2112287**
Peter Siow Wei Chun	**U2102775**
Muhammad Showherda Ad-Din	**S2117118**
Mohammed Al Aubasani	**S2102848**
Isaiah Disimond	**U2000487**

Table of Contents

Problem Statement

In the present technological age, students frequently experience an overwhelming number of distractions and an endless number of options when it comes to using technology for educational purposes. Our goal is to create a student productivity recommendation system that offers personalized suggestions for useful apps and resources based on each student's ratings of the apps that they already use or are familiar with.

A broad spectrum of digital tools, applications, and study resources are available in the modern educational environment for students to use as they progress through their learning process. However, students may find it challenging and exhausting to properly handle the abundance of choices. Additionally, students might not be aware of the resources that are most suited to their educational needs, interests, and goals.

The main goal of this project is to develop and deploy a student productivity apps recommendation system that uses user ratings on a variety of study aids and apps to produce customized suggestions for additional useful apps. The system seeks to enhance and increase the level of accuracy of its recommendations through the gathering and analysis of user ratings, eventually improving students' academic productivity, engagement, and overall learning results.

Proposed Solution

Data Description

The data for the Student's Productivity Recommendation System was collected from Google Play using the Google Play Scraper library. The objective was to gather reviews and ratings for a list of productivity apps. The data collection process involved filtering reviews by specific user names and extracting the corresponding ratings.

The collected data was then cleaned and preprocessed to ensure data quality. The cleaned data was saved as a DataFrame in a CSV file named "df_reviews.csv". The DataFrame contains user names as index and app packages as columns, with the corresponding ratings for each user-app combination.

To facilitate further analysis and modeling, the data was rearranged into a new DataFrame with three columns: UserID, AppID, and Rating. This rearranged data is stored in a CSV file named "df_rearranged.csv". The UserID and AppID represent unique identifiers for users and apps, respectively, while the Rating represents the user's rating for a specific app.

In addition, a DataFrame and CSV file named "df_app_names.csv" were created to map AppIDs to their corresponding App names. This mapping allows for easier interpretation and presentation of the recommendation results.

	Microsoft One	Trello	Wunderlist	Todoist	Google Keep	Any.do	Workflowy	Dropbox	Google Docs	Google Sheets	Google Slides	Google Drive	Google Classr
Aadit	1	0	0	0	5	1	0	0	0	5	0	0	0
Aahan	0	0	0	0	0	0	0	0	0	0	1	0	5
Aaliyah	0	0	0	0	0	0	0	0	1	1	4	0	1
Aarav	0	0	0	0	4	4	0	0	5	3	0	5	1
Aarush	0	0	0	5	5	0	0	0	0	0	1	0	5
Abbas	5	5	0	1	3	4	0	5	5	5	5	5	1
Abbott	2	0	0	5	5	0	0	0	0	0	5	5	1
Abby	1	0	0	5	3	1	0	5	1	0	3	0	2
Abdel	5	0	5	1	5	5	0	5	5	5	4	4	1
Abdelrahman	0	0	0	1	5	5	0	0	0	0	4	0	0

Figure 1a

Mendeley	GitHub	TED	Udemy	Duolingo	Quizlet	Memrise	Anki	RescueTime	Boosted Produ	StudyBlue	Remind	Grammarly Ke
0	4	0	4	5	0	5	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	5	0	0	0	0	0	5	5
0	0	0	0	5	5	0	0	0	0	0	0	0
0	0	0	1	5	1	0	0	0	5	0	0	5
1	5	5	5	5	5	5	5	0	5	0	0	5
0	3	0	5	0	0	0	0	0	0	0	0	0
0	5	1	5	5	1	5	4	0	0	0	1	5
0	5	1	1	5	5	5	0	0	5	0	5	5
0	5	3	2	5	5	5	0	0	0	0	0	5

Figure 1b

Figure 1a and Figure 1b show the first ten rows of the "df_reviews.csv" table after the data cleaning process.

UserID	AppID	Rating
1	1	1
1	5	5
1	6	1
1	10	5
1	15	4
1	17	4
1	18	5
1	20	5
2	11	1
2	13	5

Figure 2

Figure 2 shows the first ten rows of the "df_rearranged.csv" table after rearranging the data into three columns: UserID, AppID, and Rating.

AppID	AppName
1	Microsoft OneNote
2	Trello
3	Wunderlist
4	Todoist
5	Google Keep
6	Any.do
7	Workflowy
8	Dropbox
9	Google Docs
10	Google Sheets

Figure 3

Figure 3 shows the first ten rows of the "df_app_names.csv" table, where each AppID is mapped to its corresponding AppName.

Data Analysis

The data analysis phase of the Student's Productivity Recommendation System project involved exploring and analyzing the collected data to gain insights into user behavior, app ratings, and potential patterns that could inform the recommendation system. Several key aspects were examined during the data analysis process.

1. **Data Overview:**

   The dataset utilized in this analysis was obtained directly from Google Play, leveraging the powerful capabilities of the Google Play Scraper library. It represents a comprehensive collection of user-generated reviews and ratings specifically pertaining to a curated list of applications. To facilitate in-depth examination and exploration, the dataset was seamlessly imported into a widely used data analysis environment, such as Python, enriched with the versatility of the popular pandas library.

   Within this rich dataset, an array of crucial information is encapsulated, encompassing not only the users themselves but also the diverse range of applications under scrutiny. Each application is uniquely identified by its app package name, acting as a vital reference point for subsequent analyses. Complementing these user and application details are the corresponding ratings provided by the users, which serve as a fundamental indicator of their satisfaction and overall experience with the respective applications.

   To ensure the robustness and integrity of the analysis, a critical step involved meticulous data cleaning procedures. These procedures were applied to the dataset, eliminating any instances of duplicated rows and diligently addressing any missing or null values. By undertaking this essential cleansing process, the resulting dataset emerged as a refined and pristine foundation, poised to deliver invaluable insights into the diverse range of user ratings that have been bestowed upon the targeted applications.

   Within the framework of this comprehensive dataset, a well-defined structure comes to life, presenting data in an organized and easily navigable tabular format. Users are effectively represented as rows, with applications assuming the role of distinct columns. It is within the intersecting cells of this well-structured table that the ratings, indicative of user satisfaction and preferences, find their rightful place. Thus, armed with this meticulous arrangement of data, analysts are primed to embark on a journey of exploration, discovering trends, patterns, and novel perspectives that can help unravel the intricate dynamics between users and the applications they engage with.

```
In [2]: from google_play_scraper import reviews, Sort
        import pandas as pd

        # List of app package names
        app_packages = [
            'com.microsoft.office.onenote',
            'com.trello',
            'com.wunderkinder.wunderlistandroid',
            'com.todoist',
            'com.google.android.keep',
            'com.anydo',
            'com.workflowy.android',
            'com.dropbox.android',
            'com.google.android.apps.docs.editors.docs',
            'com.google.android.apps.docs.editors.sheets',
            'com.google.android.apps.docs.editors.slides',
            'com.google.android.apps.docs',
            'com.google.android.apps.classroom',
            'com.mendeley',
            'com.github.android',
            'com.ted.android',
            'com.udemy.android',
            'com.duolingo',
            'com.quizlet.quizletandroid',
            'com.memrise.android.memrisecompanion',
            'com.anki',
            'com.rescuetime.android',
            'com.boostedproductivity.app',
            'com.studyblue',
            'com.remind101',
            'com.grammarly.android.keyboard',
        ]



        # List of user names
        user_names = ['Aadit', 'Aahan', 'Aaliyah', 'Aarav', 'Aarush', 'Abbas', 'Abbotsford', 'Abbott', 'Abby',


        # Create an empty DataFrame to store the reviews
        df_reviews = pd.DataFrame(index=user_names, columns=app_packages)

        # Create a set to keep track of usernames
        seen_usernames = set()

        # Iterate through each app
        for app_package in app_packages:
            # Get app reviews for the specified app
            result, _ = reviews(
                app_package,
                lang='en',
                country='us',
                sort=Sort.NEWEST,
                count=10000,
            )

            # Iterate through each user
            for user_name in user_names:
                # Check if the username is already seen
                if user_name.lower() in seen_usernames:
                    # Skip if the username is duplicated
                    continue

                # Filter reviews based on the user name
                filtered_reviews = [review for review in result if user_name.lower() in review['userName'].lowe

                # Extract the scores from the filtered reviews
                scores = [review['score'] for review in filtered_reviews]

                # Get the latest score if available
                score = scores[0] if scores else 0

                # Add the score to the DataFrame
                df_reviews.loc[user_name, app_package] = score

                # Delete rows where all values are empty
        df_reviews.dropna(how='all', inplace=True)

        # Print the DataFrame
        print(df_reviews)

        df_reviews.to_csv('df_reviews.csv', index=True)
```

Figure 4: shows how the dataset obtained from Google Play using the Google Play Scraper library

```
In [3]: # Read the CSV file into a DataFrame
        df_reviews = pd.read_csv('df_reviews.csv', index_col=0)

        # Check if all values from column B until AA are 0 and delete the rows
        df_reviews = df_reviews.loc[(df_reviews.iloc[:, 1:] != 0).any(axis=1)]

        # Remove duplicate rows
        df_reviews = df_reviews.drop_duplicates()


        # Save the updated DataFrame to a new CSV file
        df_reviews.to_csv('df_reviews_updated.csv', index=True)
```

Figure 5: shows how the dataset is being cleaned

2. User Ratings Distribution:

A rearranged table was ingeniously devised in order to present the distribution of ratings in a more visually appealing and comprehensible manner, enabling the swift identification of any potential biases or discernible trends within the user ratings. The resulting table consists of three distinctive columns, namely UserID, AppID, and Rating, which meticulously document the ratings assigned by diverse users to a myriad of applications. The primary objective behind this arrangement is to conduct a meticulous examination of the user ratings, thereby unraveling profound insights into user preferences and behavioral patterns. To accomplish this task, an extensive frequency analysis was conducted, meticulously tallying the occurrences of each rating value, ranging from the lowest denominator of 1 star to the highest accolade of 5 stars. This painstaking analysis serves as an invaluable resource for comprehending the most prevalent rating values and, in turn, furnishes critical information pertaining to user inclinations towards distinct applications.

```
In [1]: import pandas as pd

        # Read the CSV file into a DataFrame
        df_reviews = pd.read_csv('df_reviews_updated.csv', index_col=0)

        # Create a dictionary to map usernames to sequential IDs
        username_to_id = {name: i+1 for i, name in enumerate(df_reviews.index)}

        # Create a dictionary to map app packages to sequential IDs
        app_package_to_id = {package: i+1 for i, package in enumerate(df_reviews.columns)}

        # Create lists to store the rearranged data
        user_ids = []
        app_ids = []
        ratings = []

        # Iterate over each row in the DataFrame
        for index, row in df_reviews.iterrows():
            # Get the username and corresponding ID
            username = index
            user_id = username_to_id[username]

            # Iterate over each column in the row
            for column, rating in row.iteritems():
                # Skip columns where the rating is 0
                if rating == 0:
                    continue

                # Get the app package and corresponding ID
                app_package = column
                app_id = app_package_to_id[app_package]

                # Append the data to the lists
                user_ids.append(user_id)
                app_ids.append(app_id)
                ratings.append(rating)

        # Create a new DataFrame with the rearranged data
        df_rearranged = pd.DataFrame({'UserID': user_ids, 'AppID': app_ids, 'Rating': ratings})

        # Print the rearranged DataFrame
        print(df_rearranged)

        # Save the rearranged DataFrame to a CSV file
        df_rearranged.to_csv('df_rearranged.csv', index=False)
```

Figure 6: shows the process of rearranging the dataset to achieve the desired arrangement

```
In [1]: import pandas as pd

        # List of app packages
        app_packages = [
            'com.microsoft.office.onenote',
            'com.trello',
            'com.wunderkinder.wunderlistandroid',
            'com.todoist',
            'com.google.android.keep',
            'com.anydo',
            'com.workflowy.android',
            'com.dropbox.android',
            'com.google.android.apps.docs.editors.docs',
            'com.google.android.apps.docs.editors.sheets',
            'com.google.android.apps.docs.editors.slides',
            'com.google.android.apps.docs',
            'com.google.android.apps.classroom',
            'com.mendeley',
            'com.github.android',
            'com.ted.android',
            'com.udemy.android',
            'com.duolingo',
            'com.quizlet.quizletandroid',
            'com.memrise.android.memrisecompanion',
            'com.anki',
            'com.rescuetime.android',
            'com.boostedproductivity.app',
            'com.studyblue',
            'com.remind101',
            'com.grammarly.android.keyboard'
        ]

        # Create a dictionary to map app packages to AppIDs
        app_package_to_id = {package: i+1 for i, package in enumerate(app_packages)}

        # Create a dictionary to map AppIDs to App names
        app_id_to_name = {
            1: 'Microsoft OneNote',
            2: 'Trello',
            3: 'Wunderlist',
            4: 'Todoist',
            5: 'Google Keep',
            6: 'Any.do',
            7: 'Workflowy',
            8: 'Dropbox',
            9: 'Google Docs',
            10: 'Google Sheets',
            11: 'Google Slides',
            12: 'Google Drive',
            13: 'Google Classroom',
            14: 'Mendeley',
            15: 'GitHub',
            16: 'TED',
            17: 'Udemy',
            18: 'Duolingo',
            19: 'Quizlet',
            20: 'Memrise',
            21: 'Anki',
            22: 'RescueTime',
            23: 'Boosted Productivity',
            24: 'StudyBlue',
            25: 'Remind',
            26: 'Grammarly Keyboard'
        }

        # Create a DataFrame for AppID and App Name
        df_app_names = pd.DataFrame({
            'AppID': list(app_id_to_name.keys()),
            'AppName': list(app_id_to_name.values())
        })

        # Save the AppID and App Name DataFrame to a CSV file
        df_app_names.to_csv('df_app_names.csv', index=False)
```

Figure 7: Figure 9: Process of Creating AppID-to-AppName Mapping

Figure 7 illustrates the process of creating a dictionary to establish a mapping between app packages and their respective AppIDs, followed by the creation of another dictionary to map AppIDs to their corresponding App names. The code snippet further generates a DataFrame containing the AppID and App Name information. Subsequently, the DataFrame is saved as a CSV file named 'df_app_names.csv'.

Solution used

A basic primer

We came across a fairly simple method that could be used for recommendation systems. And that is **Matrix factorization**, Understanding this requires you to be familiar with matrices and partial differentiation.

But firstly, when we say factorization in a general sense in math we mean to break down a number into let's say 2 of its factors, for example;

$$34 = 17 \cdot 2$$

The basic idea of Matrix factorization is similar to this where we intend to break down a big matrix into 2 lower dimensional rectangular matrices. These matrices can be called as the factors of the big matrix, because if we perform matrix multiplication on them we are able to get back a good approximation of the original matrix.

Some background

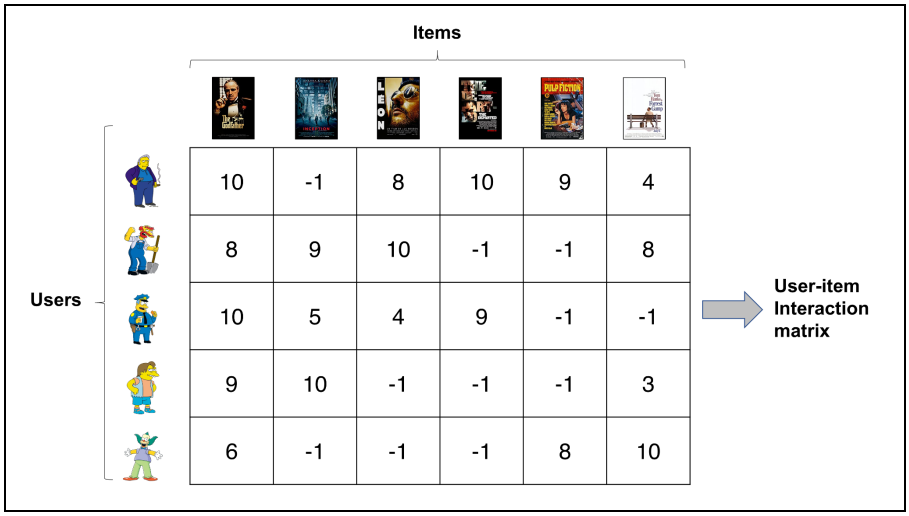
Before we delve into the Math behind matrix factorization, here's some information about it. Matrix factorization is a class of collaborative filtering algorithms used in recommendation systems. Matrix factorization works by decomposing the user-item matrix (rows for users and columns for items) into the product of two lower dimensionality rectangular matrices.

Collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if person A has the same opinion as person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person.

This family of methods became widely known during the Netflix prize challenge due to its effectiveness as reported by Simon Funk in his 2006 blog post, where he shared his findings with the research community.

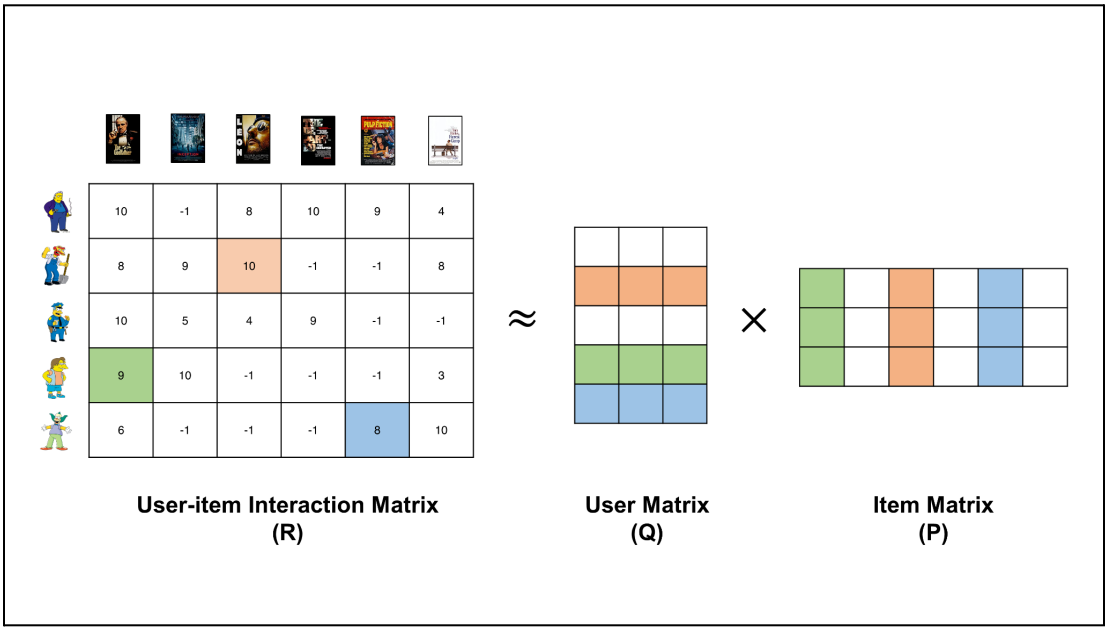
How does it recommend?

Now that we know a few things about matrix factorization, we can try to understand how it recommends. Imagine you have a user-movie interaction matrix (with user ratings of the movies) with dimension (i x j). Here the cells correspond to the rating given by the i-th user to the j-th movie. Cells with the value -1 represent an absense of an interaction (i.e. the user didn't rate the movie).



After we run this through matrix factorization we end up with two lower dimensionality rectangular matrices Q and P. With the dimensions (i x k) and (k x j) respectively

Here k is the size of the latent (hidden) features and is usually chosen by the person training the model. It is usually in $k \leqslant min(i,\ j)$.The example provided has k = 3. The dot product of the i-th row in the Q matrix and j-th column in the P matrix gives us the rating value the i-th user has given to the j-th movie. For example, the dot product of the 2nd row in the user matrix (Q) and the 3rd column in the item matrix (P) gives us the rating of 10 (highlighted in light red). So the 2nd user gave a 10 rating to the 3rd movie.



The reason why K is referred to as the hidden features or latent features is that it is not defined explicitly like the user feature or the movie feature, it only comes up when we factorize the ratings matrix. So the

user matrix (Q) becomes (user x latent features) and the movies matrix (P) becomes (movie x latent features).

The nice thing about Matrix factorization is that if there were no user-movie interaction (the user didn't rate the movie). Let's say the i-th user didn't rate movie j. When we factorize the rating matrix R and perform matrix multiplication with the factor matrices P and Q we get a new rating matrix ($\hat{R}$). In this matrix we would have a predicted value for what the i-th user might have rated for the j-th movie.

$$R \approx Q \times P = \hat{R}$$

This predicted rating value can be used to recommend movies that the i-th user has not seen yet. This is how recommendation works with matrix factorization.

How do we factorize the rating matrix R?

Firstly, we have the rating matrix R, with 2,632 users as its rows and 26 apps as its columns
Here some cells are empty (indicating that the user didn't rate the app) we fill in all empty slot with zeroes.

	Microsoft One	Trello	Wunderlist	Todoist	Google Keep	Any.do	Workflowy	Dropbox	Google Docs	Google Sheets	Google Slides	Google Drive	Google Classr
Aadit	1	0	0	0	5	1	0	0	0	5	0	0	0
Aahan	0	0	0	0	0	0	0	0	0	0	1	0	5
Aaliyah	0	0	0	0	0	0	0	0	1	1	4	0	1
Aarav	0	0	0	0	4	4	0	0	5	3	0	5	1
Aarush	0	0	0	5	5	0	0	0	0	0	1	0	5
Abbas	5	5	0	1	3	4	0	5	5	5	5	5	1
Abbott	2	0	0	5	5	0	0	0	0	0	5	5	1
Abby	1	0	0	5	3	1	0	5	1	0	3	0	2
Abdel	5	0	5	1	5	5	0	5	5	5	4	4	1
Abdelrahman	0	0	0	1	5	5	0	0	0	0	4	0	0

We then choose a value for K, let's say 26 (as we have 26 apps in our data).

$$R^{(user \times app)} \approx X^{(user \times K)} \times Y^{(K \times app)} = \hat{R}^{(user \times app)}$$

After substituting the values;

$$R^{(2632 \times 26)} \approx X^{(2632 \times 26)} \times Y^{(26 \times 26)} = \hat{R}^{(2632 \times 26)}$$

Now we randomly guess some values for the factor matrices $X^{(2632 \times 26)}$ and $Y^{(26 \times 26)}$.

We perform matrix multiplication on these to get $\hat{R}$.
Here we have,

$$\widehat{r_{ij}} \;=\; x_i \; y_j \;=\; \sum_{k=1}^{k\;=K} x_{ik} \; y_{kj}$$

Where,

$\widehat{r_{ij}} \;\epsilon\; \hat{R},\; x_{ik} \;\epsilon\; X,\; and\; y_{kj} \;\epsilon\; Y$

and

$x_i \;=\; i^{th}\; row\; in\; X,$

$y_j \;=\; j^{th}\; column\; in\; Y$

Now we calculate the difference between R and $\hat{R}$ for values in the matrix that are not 0.

$$e_{ij} \;=\; (r_{ij} \;-\; \widehat{r_{ij}}) \quad where\; r_{ij} \;\neq\; 0$$

$$e_{ij} \;=\; r_{ij} \;-\; \sum_{k=1}^{k\;=K} x_{ik} \; y_{kj}$$

Using this we can calculate the squared error.

$$e_{ij}^2 \;=\; \left(r_{ij} \;-\; \sum_{k=1}^{k\;=K} x_{ik} \; y_{kj} \right)^2$$

When it comes to a single instance of k the equation for error becomes.

$$e_{ij}^2 \;=\; (r_{ij} \;-\; x_{ik} \; y_{kj})^2$$

At this instance we try to minimize the error therefore we take partial derivatives with respect to x_{ik} and y_{kj}. So we get,

$$\frac{\partial e_{ij}^2}{\partial x_{ik}} \;=\; -2\,(r_{ij} \;-\; \widehat{r_{ij}})\, y_{kj}$$

and

$$\frac{\partial e_{ij}^2}{\partial y_{kj}} \;=\; -2\,(r_{ij} \;-\; \widehat{r_{ij}})\, x_{ik}$$

These can also be written as;

$$\frac{\partial e_{ij}^2}{\partial x_{ik}} \ = \ -2 \ e_{ij} \ y_{kj}$$

$$\frac{\partial e_{ij}^2}{\partial y_{kj}} \ = \ -2 \ e_{ij} \ x_{ik}$$

Now we can use these partial derivatives to update the values for x_{ik} and y_{kj}. We are basically doing gradient descent here.

$$x'_{ik} \ = \ x_{ik} \ + \ 2 \ \alpha \ e_{ij} \ y_{kj}$$

and

$$y'_{kj} \ = \ y_{kj} \ + \ 2 \ \alpha \ e_{ij} \ x_{ik}$$

After we have done all the updates we can calculate the total error by summing the error e_{ij} over all the values where $r_{ij} \ \neq \ 0$.

$$Total \ Error \ = \sum_{(r_{ij} \ \epsilon \ R, \ r_{ij} \ \neq \ 0)} e_{ij}^2$$

If we expand it, we get,

$$Total \ Error \ = \sum_{(r_{ij} \ \epsilon \ R, \ r_{ij} \ \neq \ 0)} \left(r_{ij} \ - \ \sum_{k=1}^{k \ = K} x_{ik} \ y_{kj} \right)^2$$

Keep in mind that this is the total error after only one iteration of updates. The total error is calculated at the end of every update. We keep on updating x_{ik} and y_{kj} until the total error goes below a threshold say 0.001 (as per convention).

This is the bare bones of matrix factorization, actual implementations of it have some regularization parameters like β to avoid overfitting.

After regularization, the equation for squared error becomes,

$$e_{ij}^2 \ = \ \left(r_{ij} \ - \ \sum_{k=1}^{k \ = K} x_{ik} \ y_{kj} \right)^2 \ + \ \frac{\beta}{2} \ \sum_{k=1}^{k \ = K} \left(||X||^2 \ + \ ||Y||^2 \right)$$

And the equation for the updating becomes,

$$x'_{ik} = x_{ik} + 2\,\alpha\,(e_{ij}\,y_{kj} - \beta\,x_{ik})$$

$$y'_{kj} = y_{kj} + 2\,\alpha\,(e_{ij}\,x_{ik} - \beta\,y_{kj})$$

Here's the code implementation of what we just discussed:

```
import numpy as np

def matrix_factorization(R, X, Y, K, iterations = 5000, alpha = 0.0002, beta = 0.02,
error_treshhold = 0.001):

    Y = Y.T

    for _ in range(iterations):
        for i in range(len(R)):
            for j in range(len(R[i])):

                #if there exists a value
                if R[i][j] > 0:
                    #we calculate the difference (error)
                    eij = R[i][j] - np.dot(X[i,:], Y[:,j])

                    for k in range(K):
                        X[i][k] = X[i][k] + alpha * (2 * eij * Y[k][j] - beta * X[i][k])
                        Y[k][j] = Y[k][j] + alpha * (2 * eij * X[i][k] - beta * Y[k][j])


        total_error = 0
        for i in range(len(R)):
            for j in range (len(R[i])):

                #Adding up all the errors after one iteration
                if R[i][j] > 0:
                    total_error = total_error + pow(R[i][j] - np.dot(X[i, :], Y[:, j]), 2)

                    #As per the regularization
                    for k in range(K):
                        total_error = total_error + (beta/2) * ( pow(X[i][k],2) + pow(Y[k][j],2)
)


        if total_error < error_treshhold:
            break

    return X, Y.T
```

We ran 10 user reviews through this code. Here's the output

```
PS C:\Users\hp> python -u "c:\Users\hp\OneDrive\Desktop\Apple_Pi\test.py"
Before Matrix Factorization
[1, 0, 0, 0, 5, 1, 0, 0, 0, 5, 0, 0, 0, 0, 4, 0, 4, 5, 0, 5, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 4, 0, 1, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 5, 5]
[0, 0, 0, 0, 4, 4, 0, 0, 5, 3, 0, 5, 1, 0, 0, 0, 0, 5, 5, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 5, 5, 0, 0, 0, 0, 0, 1, 0, 5, 0, 0, 0, 1, 5, 1, 0, 0, 0, 5, 0, 0, 5]
[5, 5, 0, 1, 3, 4, 0, 5, 5, 5, 5, 5, 1, 1, 5, 5, 5, 5, 5, 5, 5, 0, 5, 0, 0, 5]
[2, 0, 0, 5, 5, 0, 0, 0, 0, 0, 0, 5, 5, 1, 0, 3, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 5, 3, 1, 0, 5, 1, 0, 3, 0, 2, 0, 5, 1, 5, 5, 1, 5, 4, 0, 0, 0, 1, 5]
[5, 0, 5, 1, 5, 5, 0, 5, 5, 5, 4, 4, 1, 0, 5, 1, 1, 5, 5, 5, 0, 0, 5, 0, 5, 5]
[0, 0, 0, 1, 5, 5, 0, 0, 0, 0, 4, 0, 0, 0, 5, 3, 2, 5, 5, 5, 0, 0, 0, 0, 0, 5]

After Matrix Factorization
array([1. , 3.7, 3.1, 4.3, 4.9, 1. , 3.5, 4.7, 1.9, 5. , 3.6, 3.5, 2.8,
       1.4, 4. , 1.3, 4. , 5. , 2.7, 5. , 4. , 3.6, 5.7, 2.3, 2.8, 5. ])
array([2.3, 3.6, 4.3, 3.9, 5.1, 2.9, 2.7, 4.9, 2. , 4. , 1. , 3.7, 5. ,
       2.4, 3.2, 0.9, 1.4, 4.5, 1.8, 4.5, 3.9, 3.2, 5.1, 3.7, 2.8, 5. ])
array([1.5, 3.3, 3.5, 3.4, 5.6, 3.1, 2.2, 4.1, 1.1, 1. , 4. , 4.5, 1. ,
       2.3, 2. , 1.4, 2.7, 5.1, 4.9, 4. , 4. , 2.6, 4.9, 2.6, 5. , 5. ])
array([3.8, 4.5, 4. , 2. , 4. , 4. , 3.6, 4.4, 4.9, 3.1, 4.9, 4.9, 1. ,
       1.2, 3.7, 3.5, 3.9, 4.9, 5.1, 4.3, 4.4, 3.9, 4.8, 3.7, 5. , 5.2])
array([2.1, 3.1, 4.1, 5. , 5. , 2.1, 2.3, 5.1, 1.1, 3.8, 1. , 3. , 4.9,
       2. , 2.8, 0.3, 1. , 5. , 1. , 3.9, 3.4, 2.7, 5. , 3.2, 2.7, 5. ])
array([5. , 5. , 4. , 1.1, 3.1, 4. , 4.2, 5. , 5. , 4.9, 5. , 5. , 0.9,
       1. , 5. , 5. , 4.9, 5. , 4.9, 5. , 5. , 3.6, 5. , 3.8, 4.4, 4.9])
array([2. , 4.5, 4.3, 5. , 4.9, 2.9, 4. , 5. , 2.3, 3.3, 5. , 5. , 1. ,
       2.3, 3. , 2. , 5. , 6.1, 4.6, 5.2, 4.5, 4.1, 5.8, 3.2, 4.4, 5.8])
array([1. , 3.4, 2.7, 5. , 3. , 1. , 3.5, 5. , 1. , 5.4, 3. , 2.9, 2. ,
       1.3, 4.9, 1. , 5. , 5. , 1. , 4.9, 4. , 2.8, 5.3, 2.5, 1. , 5. ])
array([5. , 4.3, 5. , 1.1, 5. , 5. , 3.8, 5. , 5. , 5. , 4. , 4. , 1. ,
       1.1, 5. , 1.1, 1. , 5. , 4.9, 5. , 5. , 3.9, 5. , 3.5, 5. , 4.9])
array([4.9, 4.4, 4.6, 0.9, 4.9, 4.9, 3.5, 5. , 4.8, 4.4, 4. , 4.6, 1.8,
       1.4, 4.9, 2.9, 2. , 5. , 5. , 5. , 5.2, 3.5, 5. , 3.9, 4.9, 5.1])
PS C:\Users\hp>
```

You can see in the output that already existing values have not been altered much, but values which were previously zero are now varied some having a rating close to 1 and some close to 5.

The recommendations from this small matrix of 10 users won't be helpful. We need to run it on a bigger matrix (preferably on all 2632 user reviews), but the code we wrote is very inefficient :/

So we had to use a library to train our data on it. We used the ***cmfrec*** library and ran our data through the classic model for matrix factorization.

2.1 Classic model

Usual low-rank matrix factorization model with no user/item attributes:

$$\mathbf{X} \approx \mathbf{A}\mathbf{B}^T + \mu + \mathbf{b}_A + \mathbf{b}_B$$

Where

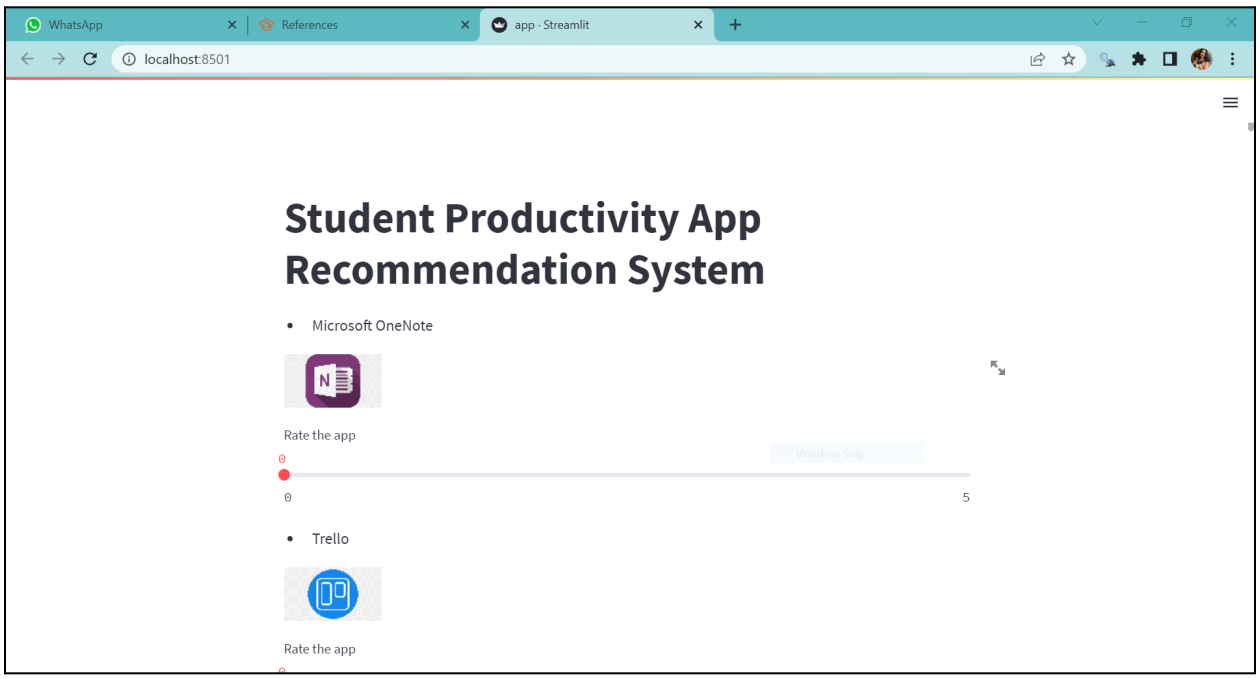
- $\mathbf{X}$ is the ratings matrix, in which users are rows, items are columns, and the entries denote the ratings.
- $\mathbf{A}$ is the user-factors matrix.
- $\mathbf{B}$ is the item-factors matrix.
- μ is the average rating.
- $\mathbf{b}_A$ are user-specific biases (row vector).
- $\mathbf{b}_B$ are item-specific biases (column vector).

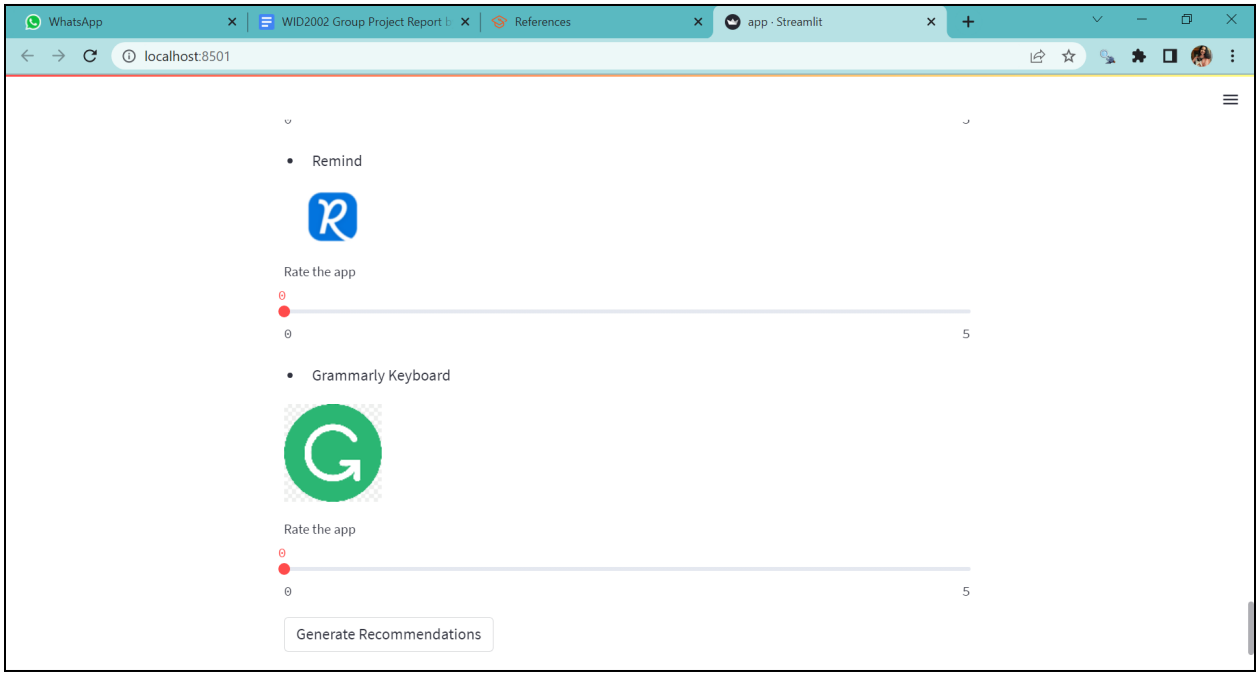
We then used the built-in topN function from the library to suggest productivity apps to the students.

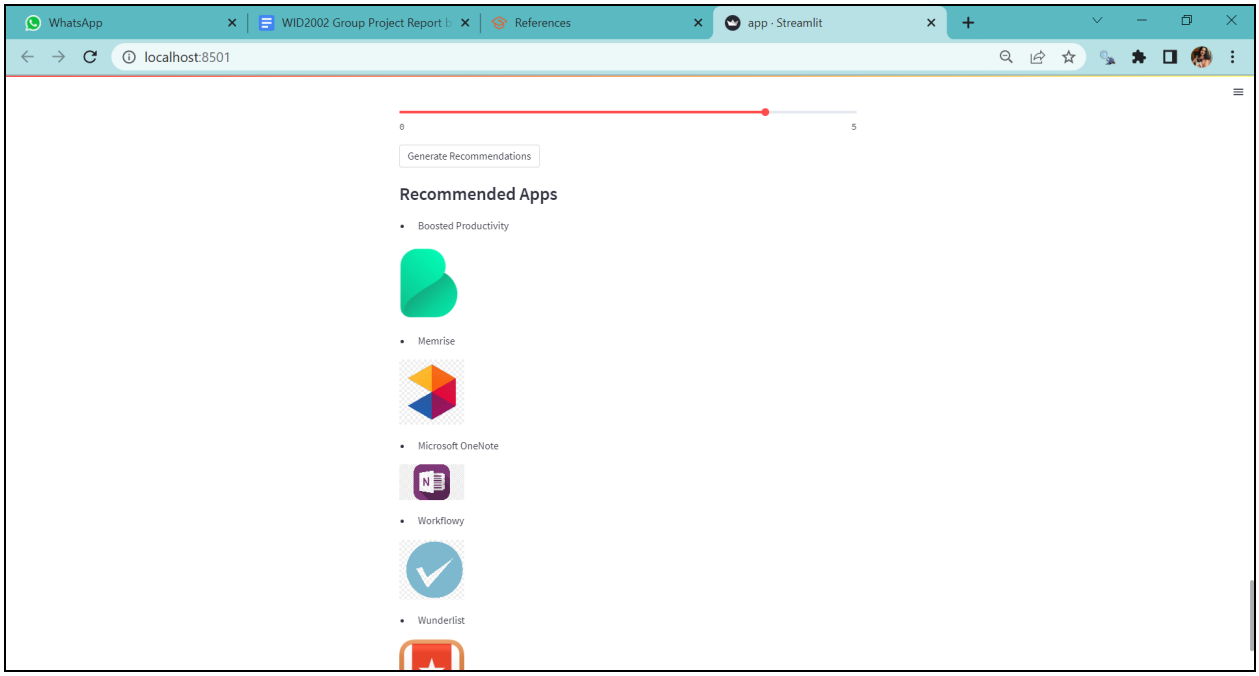
It is somewhat ingenious how we are gonna recommend the productivity apps to the new user. Firstly, they will be asked to rate the apps they already used. Then we are gonna append the new user's ratings to the already existing database. And perform matrix factorization.

When we get back the new matrix $\hat{R}$, it would contain predicted ratings for apps the user has not used. Based on these predicted ratings we will recommend apps to the new user.

Website


WhatsApp
References
app · Streamlit
localhost:8501
Student Productivity App Recommendation System
Microsoft OneNote
Rate the app
0
0
5
Trello
Rate the app


WhatsApp
WID2002 Group Project Report
References
app · Streamlit
localhost:8501
Remind
Rate the app
0
0
5
Grammarly Keyboard
Rate the app
0
0
5
Generate Recommendations



Our website's goal is to help students become more productive by suggesting helpful study-related apps. The following simply describes how our website works:

- Display of the Top 26 Productive Apps: Users to our website are given a choice of 26 productive apps when they initially enter. The apps come with their names and logos.
- User Rating Input: A slider is provided underneath each app's image to allow users to input their ratings for the apps they are most familiar with. Each app can be rated on a scale of 0-5 by users based on how beneficial or relevant it seems to be for their education.
- Generate Response: The user can go on by clicking the "Generate Recommendation" button when they have done rating the apps they are familiar with. The recommendation process is initiated by this step.
- Recommended Apps Display: After a user clicks the generate recommendation button, our website evaluates their ratings and presents them with five potential app choices. These suggestions are shown on the website, giving users a list of apps that might be useful and productive for their education.

Github Link

Website: https://github.com/samiha26/Computing-Mathematics-2.git

Strength and Limitations

Strengths:

- Simple recommendation system using matrix factorization.
- Easy learning curve.
- High effectiveness as reported by Simon Funk in his 2006 blog post.
- Easy to use & simple user interface design.

Limitations:

- Data source limitation: Limited to apps available on Google Play.
- Sparse data challenges: Impact on recommendation accuracy.
- Cold start problem not addressed.
- Evaluation metrics not specified.
- Lack of validation and real-world testing.

Future Work

1. **Enhanced Data Collection**: By extracting more data from the Google Play Store, such as app descriptions, user reviews, and app metadata (such as category, developer, and release date), the data collecting process can be improved. The original matrix can be enhanced with this new information, and the recommendations will be of higher quality.

2. **Handling Sparse Data**: Because collaborative filtering techniques like matrix factorization might struggle with sparse matrices, it is important to develop strategies for handling sparse data. We intend to investigate methods like "matrix completion," which uses low-rank matrix approximation to fill in blanks in the user-item matrix.

3. **Dynamic Recommendation**: Put in place a dynamic recommendation system that changes to match users' evolving tastes. This can be accomplished by including time-dependent variables in the matrix factorization model, taking into account variables like app popularity, user activity levels, or temporal trends in app ratings.

4. **Cold Start Problem**: The cold start issue, which happens when a new user or program enters the system with little data available, has to be addressed. We will investigate hybrid approaches to recommendations that combine collaborative filtering with content- or knowledge-based techniques to offer preliminary recommendations up till adequate user feedback is gathered.

5. **Evaluation Metrics**: Utilize relevant assessment criteria, such as precision, recall, F1-score, or mean average precision, to assess and compare the effectiveness of various recommendation algorithms. This will enable us to evaluate the performance of our recommendation system and contrast it with other methods.

6. **Regularization Techniques**: We intend to try out several regularization methods to manage model complexity and avoid overfitting. Improved generalization and reduced reliance on a small number of dominating elements can be achieved with the aid of methods like L1 or L2 regularization.

7. **Exploration vs. Exploitation**: We want to utilize both exploitation and exploration tactics to strike a balance between promoting well-liked products (exploitation) and promoting unique or diverse products (exploration). This can be accomplished by combining strategies like Thompson sampling and multi-armed bandits into the recommendation system.

8. **Domain-Specific Considerations**: In the future, we may apply limits to protect privacy, diversity, or fairness in the recommendations as parts of domain-specific-knowledge.

9. **Interpretability and Explainability**: Create methods to help people better understand and interpret the recommendation process. User confidence and engagement can be increased by employing strategies like model introspection, explaining recommendations, or showing the factors discovered by the matrix factorization model. Thus, in the future we will update the UI to include these factors that were discovered during the model training stage.

10. **Contextual Information**: In the future, we will include background knowledge in the suggestion process; such as: user demographic data, temporal context (such as time of day or week), or location-based context, for instance, to further tailor recommendations and make them more pertinent to the user's present situation.

Conclusion

The project demonstrates the effectiveness of matrix factorization in developing a productivity app recommendation system. Through the decomposition of the user-app ratings matrix, personalized recommendations can be generated. The data collected from Google Play using the Google Play Scraper library proved valuable for building the recommendation system. Future work recommendations, such as exploring advanced matrix factorization techniques, incorporating contextual information, and evaluating algorithm performance, provide avenues for further improvement. The project highlights the importance of user feedback, real-time updates, and addressing challenges like sparse data. Enhancements in user interface design and visualization contribute to better user engagement and understanding. Overall, this project lays a foundation for a robust and user-centric productivity app recommendation system

Contribution of Each Team Member

Name	Contribution
Samiha Tasnim Dristy	1. Decided the project topic 2. Problem statement 3. Evaluate the model 4. Website development
Peter Siow Wei Chun	1. Decided the project topic 2. Data analysis 3. Data description 4. Find suitable dataset 5. Filter dataset
Muhammad Showherda Ad-Din	1. Decided the project topic 2. Future work 3. Evaluate the model 4. Website development
Mohammed Al Aubasani	1. Decided the project topic 2. Solution used 3. Train the model 4. Strengths and Limitations
Isaiah Disimond	1. Decided the project topic 2. Find suitable dataset 3. Filter dataset 4. Strengths and Limitations 5. Conclusion

References

Chen, D. (2021, December 15). Recommender System — Matrix Factorization - Towards Data Science. *Medium*. https://towardsdatascience.com/recommendation-system-matrix-factorization-d61978660b4b

"Introduction to Matrix Factorization - Collaborative Filtering with Python 12 · Buomsoo Kim." *Github.io*, 2022, buomsoo-kim.github.io/recommender%20systems/2020/09/25/Recommender-systems-collab-filtering-12.md/ Accessed 9 June 2023.

"Jupyter Notebook Viewer." *Nbviewer.org*, nbviewer.org/github/david-cortes/cmfrec/blob/master/example/cmfrec_movielens_sideinfo.ipynb. Accessed 9 June 2023.

"Matrix Factorization: A Simple Tutorial and Implementation in Python @ Quuxlabs." *Quuxlabs.com*, 2010, www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/.

Wikipedia Contributors. "Collaborative Filtering." *Wikipedia*, Wikimedia Foundation, 5 May 2019, en.wikipedia.org/wiki/Collaborative_filtering.

---. "Matrix Factorization (Recommender Systems)." *Wikipedia*, Wikimedia Foundation, 18 Feb. 2019, en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems).

Funk, Simon. "Netflix Update: Try This at Home." *Sifter.org*, 2006, sifter.org/~simon/journal/20061211.html