# Introduction

Hello, today I’m going to show you how to **recommend movies** using **Collaborative Filtering**. And we will implement it using **Python Jupyter Notebook**.

In Collaborative Filtering, we have **two entities**, in our case, **the users** and **the movies**.

**A score** is given to **quantity the relationship** between these two entities. For examples, a user will **rate a score** of 1 to 5to a movie, to quantify how much he likes it.

However, a user will **not rate every movie**, and a movie will **not be rated by** **all users**. There are some **missing scores** and our task is to **predict** those missing scores from the existing ones.

# Loading Data

Here’s the codes. First, we will **load data** from the files.

Our dataset has about **900 users on 1600 movies**. We have **3 variables**; the **movie list** stores the **names of the movies**. Given a movie ID, we can enquire its name.

The other two variables are **matrices Y and R**. Y represent the **users’ preferences to movies**, it has values **ranging from 0 to 5**. Its **dimension** is the no. of users times the no. of movies.

**R has the same shape** as Y, but its values are **only 1 or 0.** Rindicates if there is an **existing score** **or not** for a user to a movie. **1 means** there is an **existing score on Y**, 0 means that is **missing**.

# Data Pre-processing

In the next step, we will do some data **pre-processing**. In our previous experience, the system often returns some **rarely known** movies. So, here we want to **shortlist** only the **100 mostly rated movies**.

Our movie list, matrices Y and R now trim down to 100 movies. Here we list the top 10 of them, they are now more recognizable.

Another important step in data pre-processing is **normalization**. We want an **average score** for the movies that have **not been given scores by a user** **at the beginning**.

Now, they are **all zeros**, which **means the worst**, when 5 means the best. So, we want to normal the scores, and have them **centred on zero**, then **zero will means the average**.

These are our normalised result, they now have **position and negative** values, and **the mean is zero**.

# Parameters Initialization

The next step is to **initiate our parameters**. First, we **define the dimensions**, m is the **no. of users**; n is the **no. of movies**. Here, we introduce k, that is the **no. of latent factors**, or so called **hidden features**.

We are using **one of the Collaborative Filtering methods** that depends on **latent factors**. Here, we assume there are **3 important features** that can **characterise all the movies** in our dataset. And we **don’t need know in advance** what these 3 factors look like.

We also assume there are **two matrices**, **Theta and X**. Theta represents the **users’ preference to the k hidden features.** And X represents the **movies’ tendency to the k hidden features**.

The dimension of Theta is a **m x k** matrix; and dimensions of X is **n x k**.

Suppose, we have an algorithm that can **tune Theta and X to have some optimal values**. Then **Theta multiples the transpose of X** will **produces a (m x n) matrix**, that will be the **same dimension** as Y!

We want that matrix **predicts the values of Y**, including the **missing scores**.

Back to our codes, we **initialize** Theta and X with some standard normal random numbers.

L2 is the constant for **regularization**, which is a **technique in machine learning to prevent overfitting**.

Here we do an action called “unroll” to **flatten Theta and X** into a 1D array. You will see why we do that later.

# Cost Function

To get the optimal Theta and X, we will use **gradient descent algorithm** to **minimize** a cost function**.**

Here, we define **the cost function**. First, we roll back the Theta and X from the flatten parameters.

Theta multiplies the transpose of X gives the prediction. **Our cost** is the **sum square error between the prediction and Y**.

But, we only **count the error** on **existing scores**, that’s why we **element-wisely multiply** this term to **R**.

And these terms are for **regularization**. This is to prevent overfitting.

This part, we **calculate the gradients** for gradient descent. The cost function returns both the cost and the unrolled form of **gradients**.

# Training

Here we use the **external optimizer** to **minimize** the cost function. The algorithm is called **LBFGS**, which is an **advance gradient descent** algorithm.

The optimizer takes the **flattened 1-D array** as parameter. That’s why we kept unrolling and rolling back our Theta, X and their gradients.

It took a few seconds for the optimization to **finish**. Here, we get the **optimal Theta and X**. In the next session, I will show you what is the **implication** of these optimal Theta and X.

# Latent Factors Analysis

Remember **X represents the tendency of movies to the hidden features**? We are interested in **which movies gave the high scores** in each of the components in X.

For the first component, those movies include criminal films, related drugs or alcohol addiction.

“A Clockwork Orange”, “Pulp Fiction”, “Dead Man Walking”, and “The Godfather” are all **criminal films**. “Trainspotting” is about **drugs taking**. “Leaving Las Vegas” is about **alcohol addiction**.

For the second component, we see movies that are **more feminine**.

“English Patient”, “The Graduate”, “Leaving Las Vegas”, and “Sense and Sensibility” are all **romance stories**. “Evita” is the musical about the **female president in Argentina**.

Movies for the last components are those **commercial**, **sci-fi** or **action** movies, like these classics.

You see **we haven’t informed** the system how to classify movies, but at the end it gained some **intelligence to classify** movies like these. That’s the characteristic of **unsupervised learning**!

# Recommending Movies

Finally, we try to **recommend movies** to a **“Terminator” fans** in our dataset. There are **two Terminator movies** in our samples, the original Terminator and Terminator II.

Frist, we find the **user who gave high scores** in these two movies and has the **least ratings activities**. That’s the user.

These are **what he had rated**. And we **recommend these movies** to him.

You see these movies are all about **killing people or a disaster**. “Independence Day” is about **aliens’ attack**, that’s an **analogue to AI robots attacking** people in Terminator.

And for those **Sci-fi fans**, they **may also like conspiracies**, maybe that’s why the system recommends “Conspiracy Theory”.

In this demonstration, you have seen Collaborative Filtering successfully classify movies automatically, and recommending movies. I hope you enjoyed it.