Building Recommender Systems with Machine Learning and AI

https://www.udemy.com/course/building-recommender-systems-with-machine-learning-and-ai/learn/lecture

---

Planning

- **w1: S1 - S4**
- **w2: S5**
- **w3: S6**
- **w4: S7**
- **w5: S8**
- **w6: S9**
- **w7: S10**
- **w8: S11, S12, S13 and S14**

S1 - Introduction

There will be presented in the course:

- Tensorflow
- DSSTNE (amazon's tool)
- Sagemaker
- Apache Spark

Recommender systems is:

- a system that predicts ratings or preferences a user might give to an item
- present "top N" recommendations
- also known as recommender engines, recommendation systems and recommendation platforms

Types of recommenders:

- things (similar or related items)
- content
- music
- people (online dating)
- search results (recommended web pages)

There are different types of behavior:

- explicitly signs (ratings, like/unlike, follow)
- implicitly signs (clicks, purchase, consume)

S2 - Skipped since it was an introduction to python

S3 - Evaluating Recommender Systems

Top-N hit rate is a metric used to analyse model's performance

- hit rate = hits on recommendations / number of users
- average reciprocal hit rate (ARHR) = 1/rank / number of users
- cumulative hit rate (cHR) = remove hit rate for ratings below a certain threshold
- rating hit rate (rHR) = hit rate by rating

Coverage is the percentage of (user, item) that can be predicted

Diversity is given by the formula (1 - S), where S is the average similarity between recommendation pairs

Novelty is the average popularity rank of recommended items

Churn is how sensitive your system is to users behavior (if the recommendation changes too much depending on the behavior's change)

Responsiveness is how quickly does new user behavior influence your recommendations

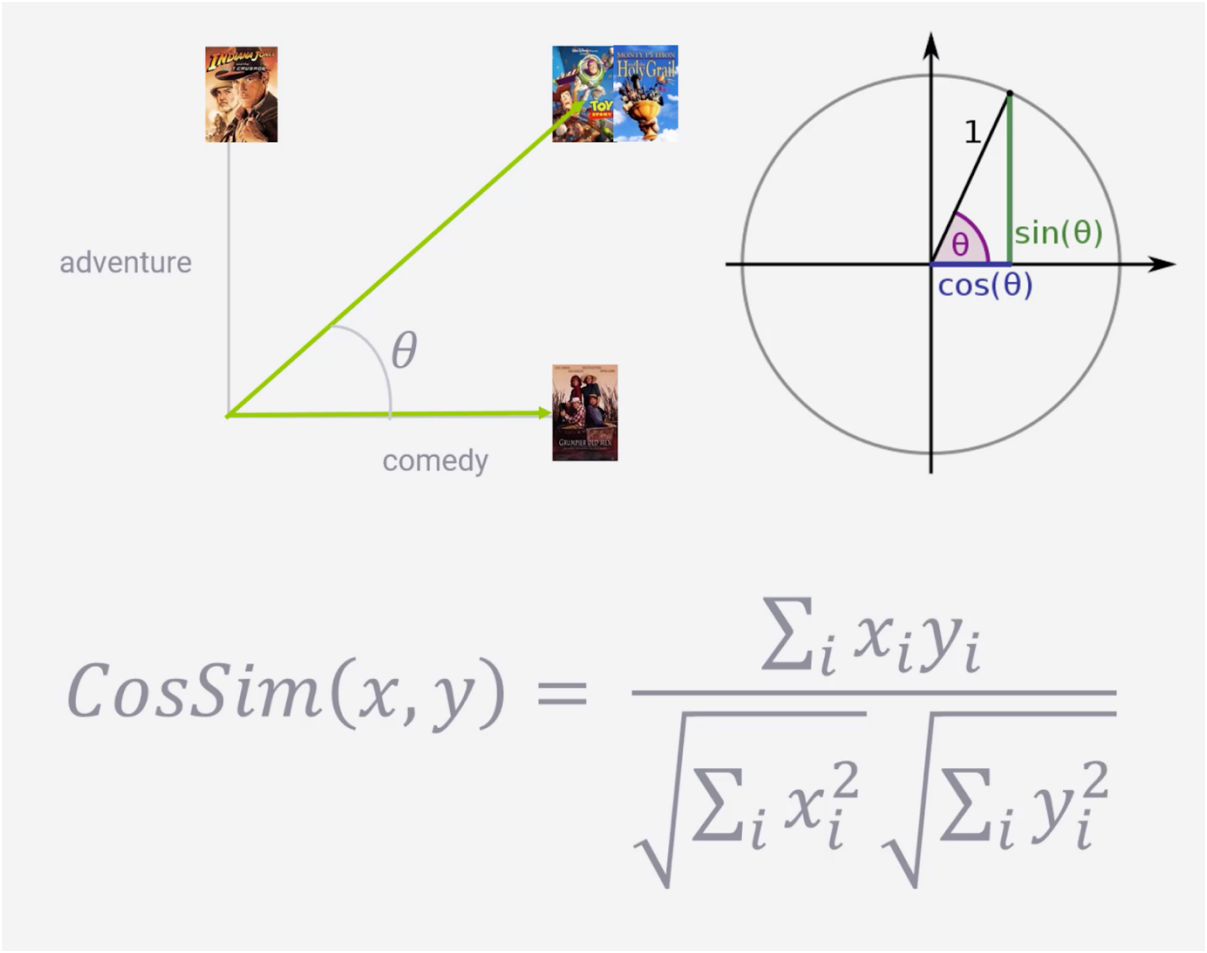
A/B tests to be used as online evaluation

S4 - Recommender Engine Framework

A walk-through the code and framework for evaluating different algorithms. In one part it made a comparison between SVD algorithm (created in the Netflix's competition) and a random model.

S5 - Content-Based Filtering

Content-based recommendations using the cosine similarity metric (illustrated below).



$$CosSim(x,y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

The codes are presented in the ContentBased folder.

S6 - Neighborhood-Based Collaborative Filtering

In summary, finding other people like you and recommending stuff they liked OR finding other things you might like based on what you liked.

One way is to calculate the cosine similarity coefficient between attributes/variables from people, e.g., “has bought nike snickers model Z”. The main difference is that the collaborative filtering takes into account the attributes from people and not from the items.

Sparsity is a problem, since there are too many items and few people who has a relationship with them. Sparse matrix is a problem, like in the image below.

	Indiana Jones	Star Wars	Shape of Water	Incredibles	Casablanca
Bob	4				
Ted					
Alice				5	

Presents several similarities metrics to use for defining the similarity of two users instead of cosine, some of them are:

Adjusted cosine

$$CosSim(x,y) = \frac{\sum_i \left((x_i - \bar{x})(y_i - \bar{y})\right)}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}}$$

Uses the user's average ratings. Attempts to normalize the users' rating, but presents a bigger problem with sparsity.

Pearson

$$CosSim(x,y) = \frac{\sum_i \left((x_i - \bar{\imath})(y_i - \bar{\imath})\right)}{\sqrt{\sum_i (x_i - \bar{\imath})^2}\sqrt{\sum_i (y_i - \bar{\imath})^2}}$$

Uses the item's average ratings. The main problem here is if two users are completely different, for certain items they might look similar.

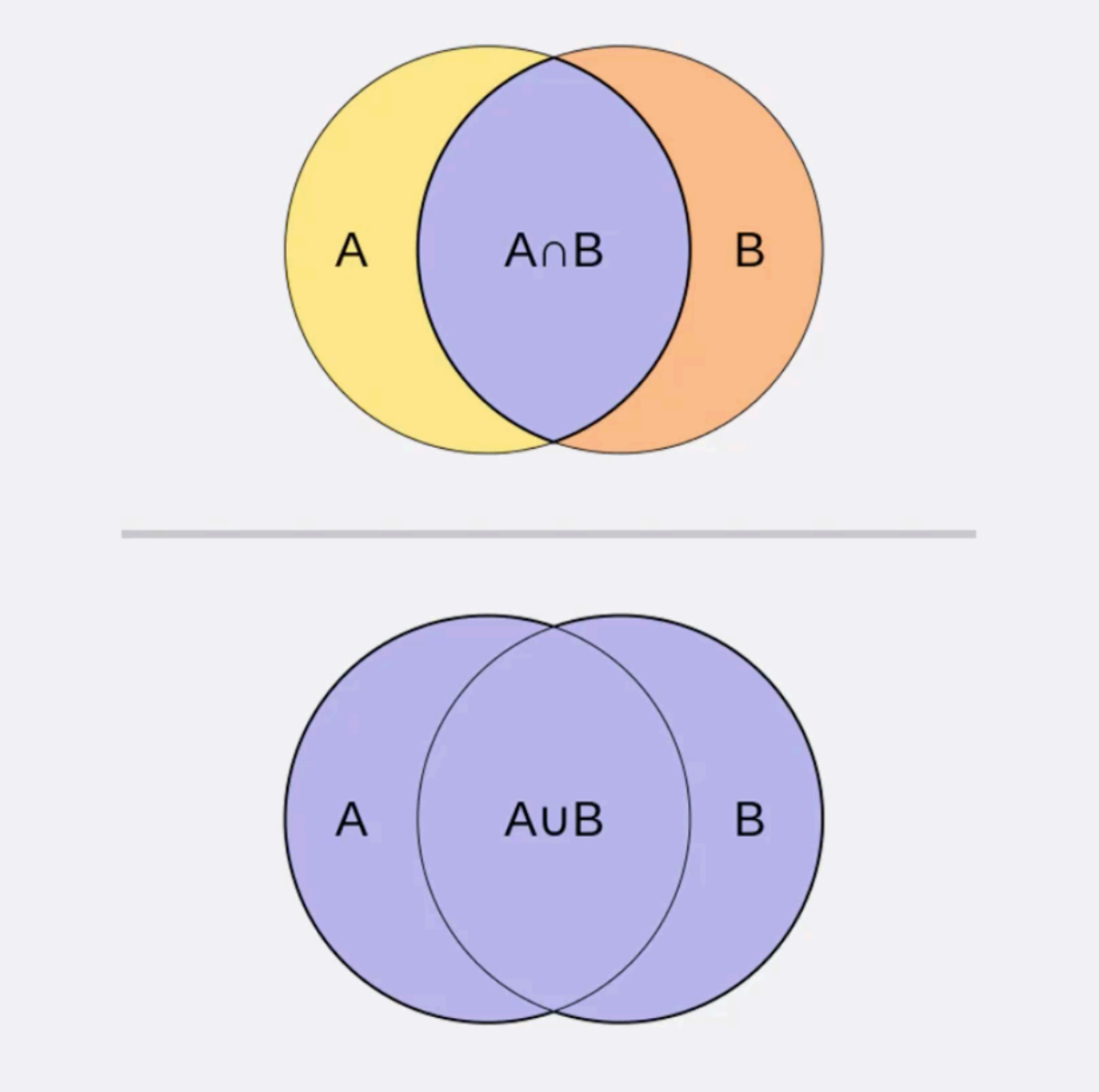
Mean Squared Difference

$$MSD(x,y) = \frac{\sum_{i \in I_{xy}} (x_i - y_i)^2}{|I_{xy}|}$$

$$MSDsim(x,y) = \frac{1}{MSD(x,y) + 1}$$

Takes into account number of items users have in common.

Jaccard



Relies on the intersection of items across users.

The default metric is **cosine similarity**, since it usually presents the best performance.

User-based collaborative filtering follows the following steps:

- user -> item rating matrix
- user -> user similarity matrix
- look up similar users
- candidate generation
- candidate scoring
- candidate filtering

Item-based collaborative filtering (might produce better results than user-based collaborative filtering) focus on getting items similar to the ones that were used/liked by the user.

The matrix is flipped from the previous case (shown below).

	Bob	Ted	Ann
Indiana Jones	4		
Star Wars	5		5
Empire Strikes Back			5
Incredibles			5
Casablanca		1	

	Indiana Jones	Star Wars	Empire Strikes Back	Incredibles	Casablanca
Indiana Jones	1	1	0	0	0
Star Wars	1	1	1	1	0
Empire Strikes Back	1	1	1	1	0
Incredibles	1	1	1	1	0
Casablanca	0	0	0	0	1

The user-based KNN formula for predicting a rating for user u and item i is as follows:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} sim(u, v) \cdot r_{vi}}{\sum_{v \in N_i^k(u)} sim(u, v)}$$

Same formula for item-based, but changing the similarity for item to item, instead of user to user (sim(i, j) instead of sim(u, v)).

KNN might not be the best solution even getting the best performance (RMSE), since the ranking might be worse.

S7 - Matrix Factorization Methods

Motivation for finding different methods for matrix factorization is the computational cost for calculating matrix similarities.

Principal Component Analysis (PCA) - method to reduce dimensionality and so reduce computational cost.

R

	Indiana Jones	Star Wars	Empire Strikes Back	Incredibles	Casablanca
Bob	4	5	5	4	4
Ted	3	3	3	5	4
Ann	4	5	5	5	2

PCA

	"Action"	"Sci-Fi"	"Classic"
Bob	0.3	0.5	0.2
Ted	0.1	0.1	0.8
Ann	0.3	0.6	0.1

U

R^T

	Bob	Ted	Ann
Indiana Jones	4	3	4
Star Wars	5	3	5
Empire Strikes Back	5	3	5
Incredibles	4	5	5
Casablanca	4	4	2

PCA

	"Action"	"Sci-Fi"	"Classic"
Indiana Jones	0.6	0.3	0.1
Star Wars	0.4	0.6	0
Empire Strikes Back	0.4	0.6	0
Incredibles	0.8	0.2	0
Casablanca	0.2	0	0.8

M

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

Bleeding edge
SLIM (Sparse Linear Methods) methodology seems to outperform most of the current methods in the HR (hitting rate) metric.

Calculates the ratings of each user times a sparse matrix of weights, the algorithm below can be used for learning the weights.

B. *Learning W for* SLIM

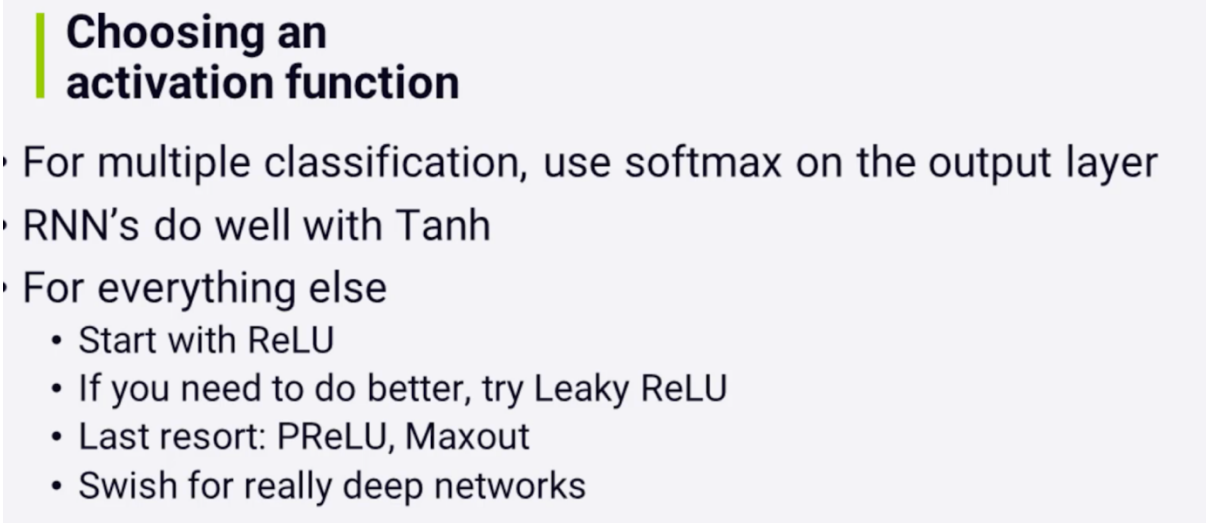
We view the purchase/rating activity of user u_i on item t_j in A (i.e., a_{ij}) as the ground-truth item recommendation score. Given a user-item purchase/rating matrix A of size $m \times n$, we learn the sparse $n \times n$ matrix W in Equation 2 as the minimizer for the following regularized optimization problem:

$$\begin{aligned} \underset{W}{\text{minimize}} \quad & \frac{1}{2}\|A - AW\|_F^2 + \frac{\beta}{2}\|W\|_F^2 + \lambda\|W\|_1 \\ \text{subject to} \quad & W \geq 0 \\ & \text{diag}(W) = 0, \end{aligned} \tag{3}$$

where $\|W\|_1 = \sum_{i=1}^{n}\sum_{j=1}^{n}|w_{ij}|$ is the entry-wise ℓ_1-norm of W, and $\|\cdot\|_F$ is the matrix Frobenius norm. In Equation 3, AW is the estimated matrix of recommendation scores (i.e., $\tilde{A}$) by the sparse linear model as in Equation 2. The first term $\frac{1}{2}\|A - AW\|_F^2$ (i.e., the residual sum of squares) measures how well the linear model fits the training data, and $\|W\|_F^2$ and $\|W\|_1^2$ are ℓ_F-norm and ℓ_1-norm regularization

S8 - Introduction to Deep Learning

There was a good introduction about ANN (structure, activation functions, optimizer methods, etc.), which was a review to me since I studied this before. No further notes about it other than the print below.



Tensorflow was developed by Google not exactly for deep learning, mostly for graph numerical operations. What Tensorflow does is to optimize the processing of that graph and distribute its processing across a network of GPUs (for example). Very similar to Apache Spark and in some ways competitors.

Tensorflow is written in C++ and it can run anywhere.

Keras, some characteristics:

- high level api on top of Tensorflow
- has a scikit_learn integration
- less to think about, fast trials

Brief explanation about CNN
In summary, it is used when you have data that doesn't easily align into columns, and it uses a convolutional method where it basically iterates through the data.

- main usage are images, find features in an image, image related problems
- machine translation
- sentence classification
- sentiment analysis

Brief explanation about RNN
In summary, it is used for sequence models and with data that consists of sequences of arbitrary length (machine translation, image captions, machine-generated music, etc.)

- Sequence to sequence
  - i.e., predict stock prices based on series of historical data
- Sequence to vector
  - i.e., words in a sentence to sentiment
- Vector to sequence
  - i.e., create captions from an image
- Encoder -> Decoder
  - Sequence -> vector -> sequence
  - i.e., machine translation

Training RNN is really hard and its what they call real deep neural networks.

- State from earlier time steps get diluted over time
  - This can be a problem, for example when learning sentence structures
- LSTM Cell
  - Long Short-Term Memory Cell
  - Maintains separate short-term and long-term states
- GRU Cell
  - Gated Recurrent Unit
  - Simplified LSTM Cell that performs about as well

On tuning neural networks:

- Small batch sizes tend to not get stuck in local minima
- Large batch sizes can converge on the wrong solution at random
- Large learning rates can overshoot the correct solution
- Small learning rates increase training time

S9 - Deep Learning for Recommender Systems

Restricted Boltzmann Machines (RBM) is being used since 2007 and is still considered one of the state of the art algorithm (Netflix published they use it too).

RBM structure is somehow simple as it's composed by a neural network with only two layers (one visible and one hidden). +?

Bleeding edge
DeepFM: A Factorization-Machine based Neural Network for CTR Prediction

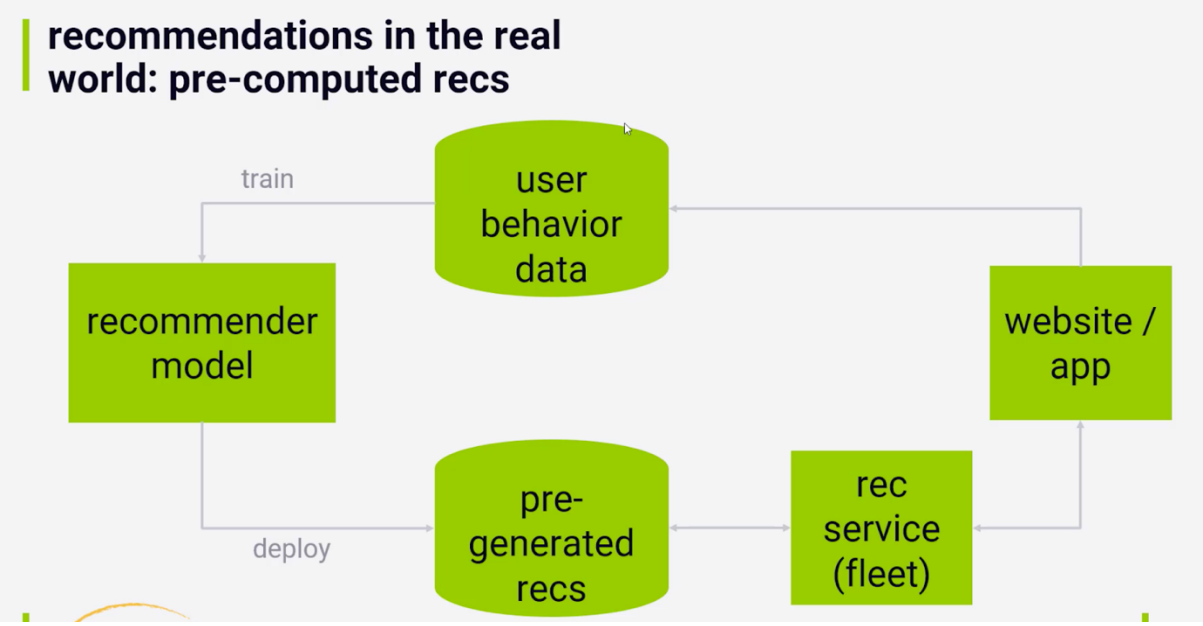
S10 - Scaling it Up

Spark can be used to split the process in several different CPUs.

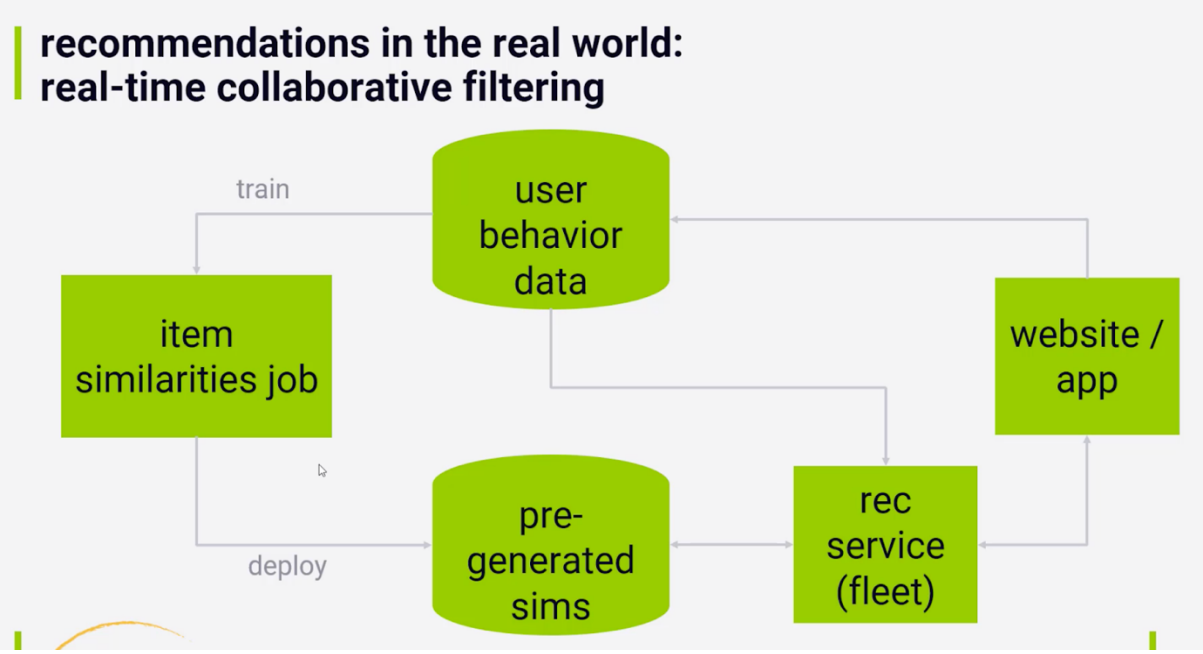
Amazon DSSTNE (Deep Scalable Sparse Tensor Neural Engine) is an engine developed by Amazon and is now open source (since 2016). It's used to produce recommender systems in Amazon Scale.

Convert training data into the format that is accepted in DSSTNE and it should create the models without one single line of code, since with only one simple json config file the model is up and running. It can run in GPUs and split the process among them.

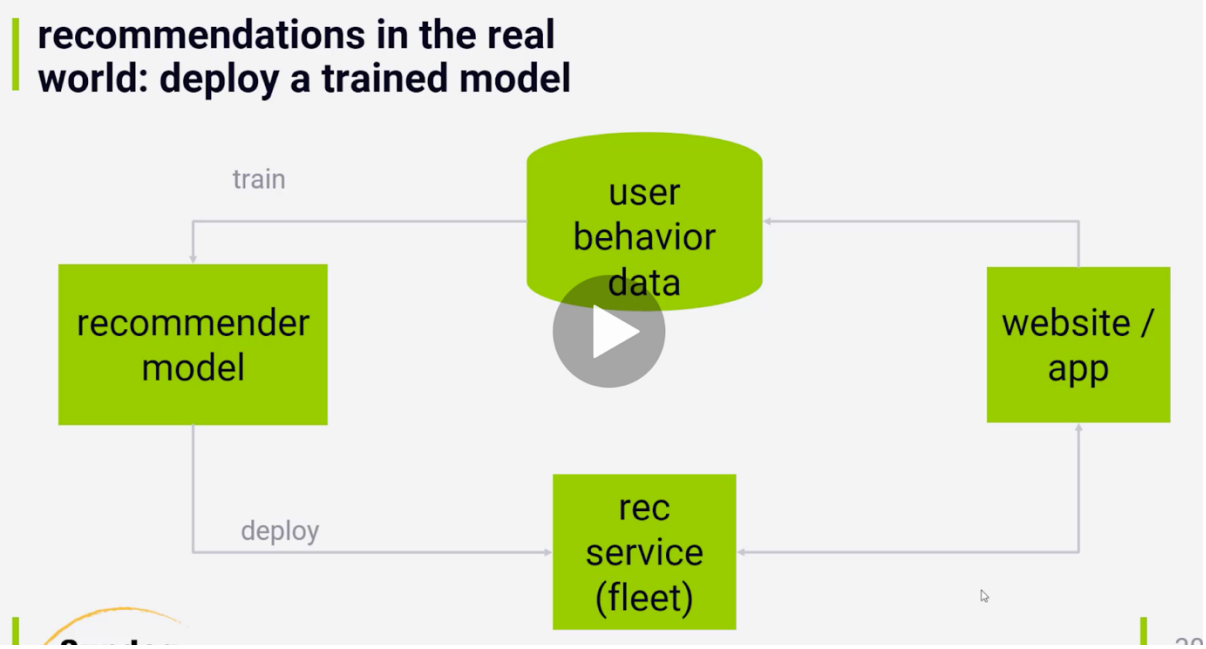
How to actually deploy a recommender system in a real live environment is a very broad subject, there are several ways of doing so, one example is pre-computed recs (shown below).



Another example, for real-time collaborative filtering (shown below).

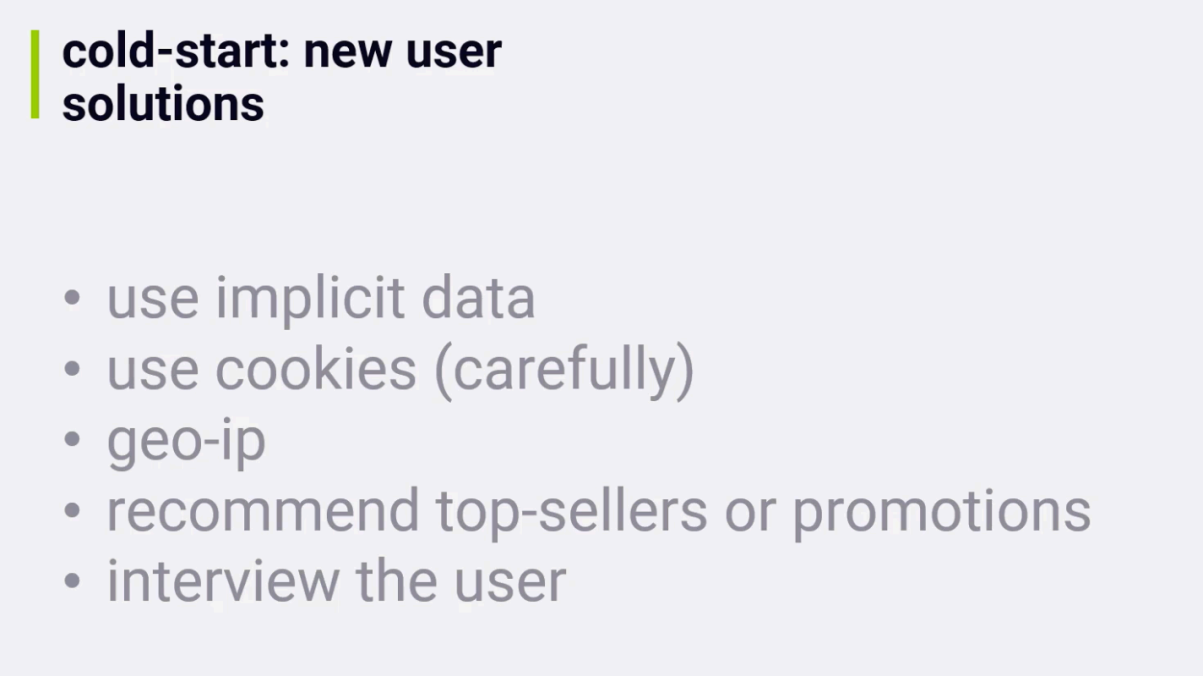


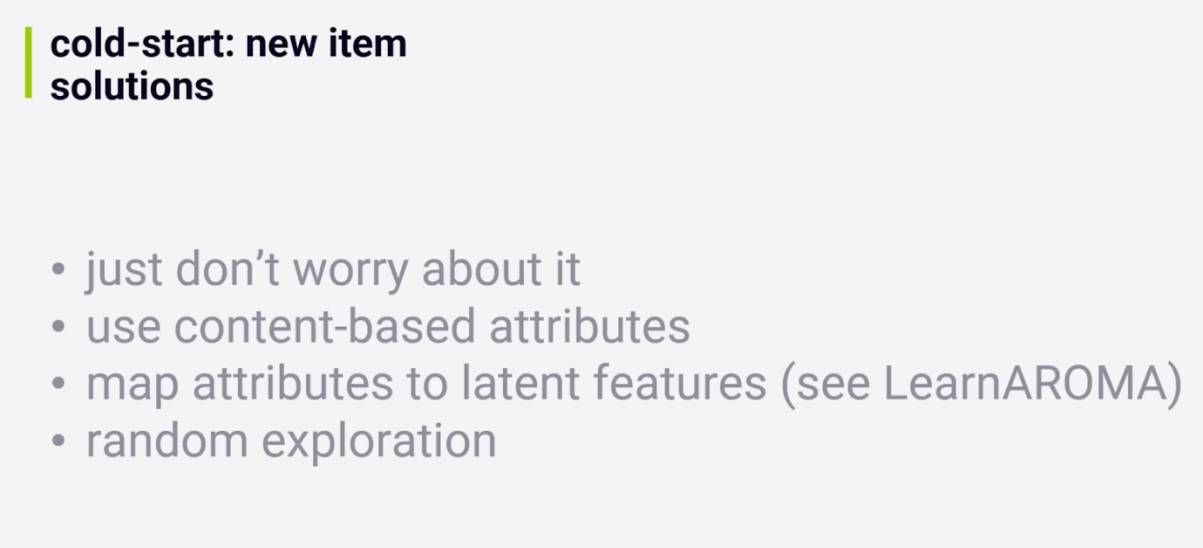
Another example, for real world very used nowadays including the way SageMaker works (shown below).



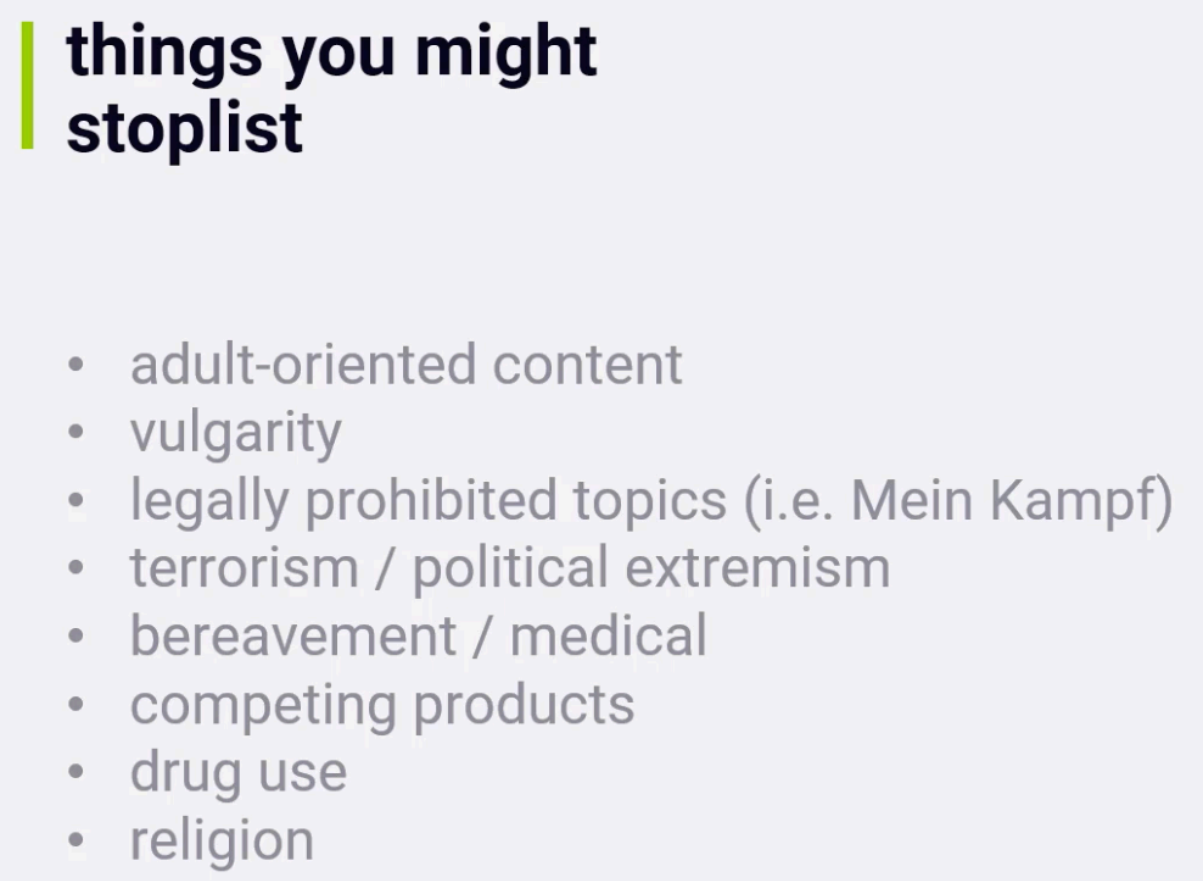
S11 - Real-World Challenges of Recommender Systems

Cold-start problem is when you don’t have any information about the user or item because it’s completely new.





Stoplists are useful to remove recommendations that might be not ethical, some examples are:

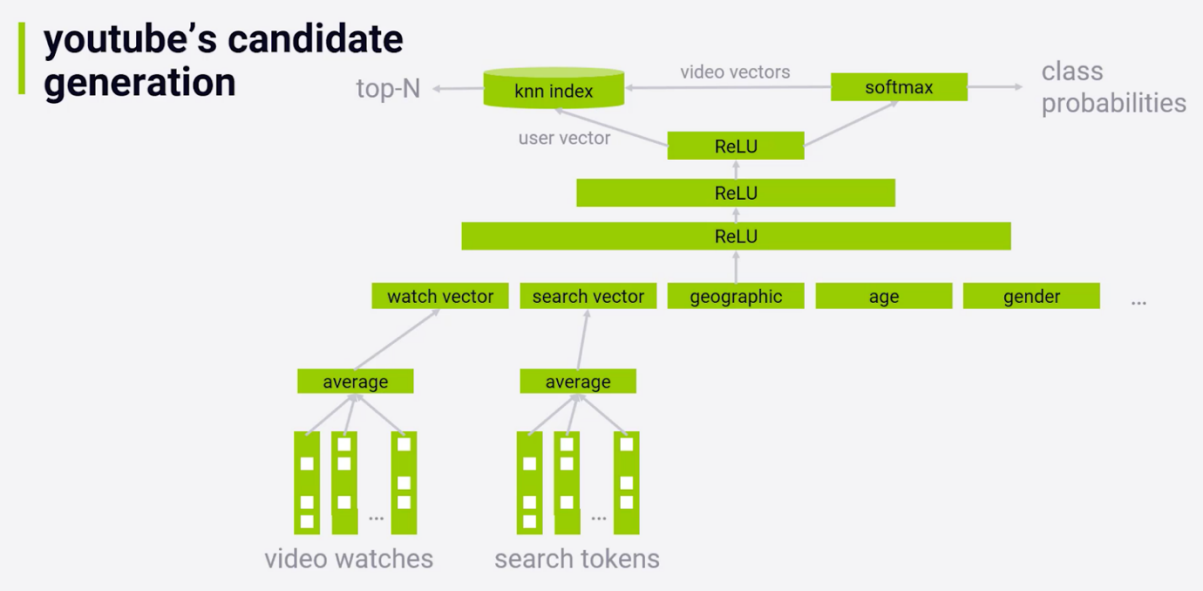


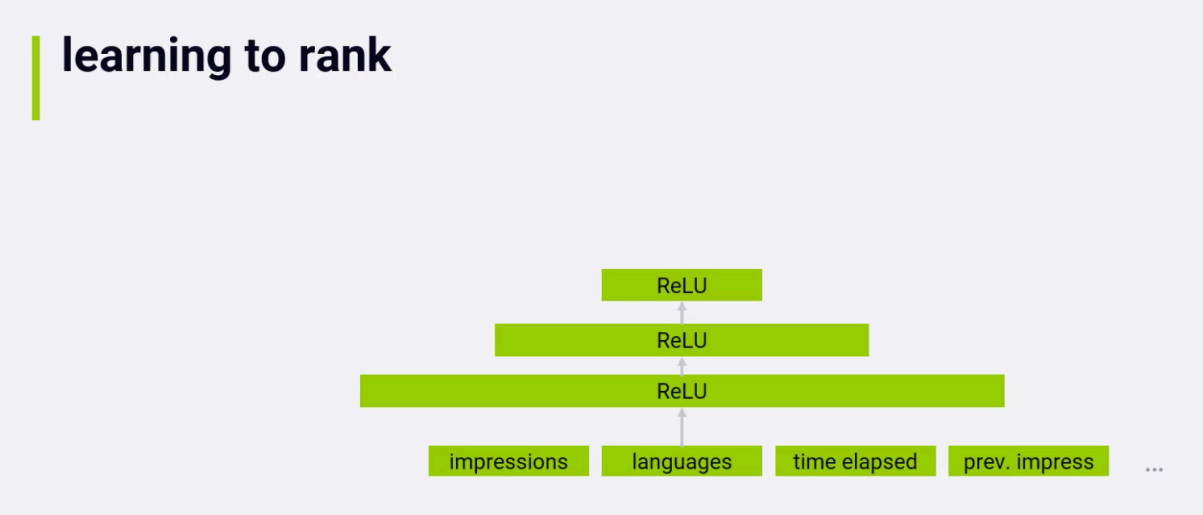
Filter bubbles is the proposal to try to solve a problem generated by recommender systems. Sometimes recommender systems can create an environment that influence users to stay on a certain bubble, thus, making the world more polarised.

S12 - Case Studies

Talks about two market cases: YouTube and Netflix

YouTube has the challenge of having too many users, events and products (videos being uploaded all the time), and has to work with scaling and updating recommendations very frequently. Mainly works with implicit data.





Netflix doesn't say much about their methodologies, but has to deal with the challenge of not having the same items in the front page when recommending.

S13 - Hybrid Approaches

There is the possibility to use hybrid approaches in the recommender systems, and that can stand as:

- ensemble of different methods
- combining behavior and semantic data

It can be the union of RBM algorithm and Content KNN algorithm.

S14 - Wrapping Up

One suggestion from the course is to attend SIGKDD, a conference about Recommender Systems.