Using Data Mining & Recommender Techniques in Clouds

November 12, 2020
Online Lecture
Review of Lecture 11 – Big Data Analytics & Cloud Data Mining

### Association Rule Mining

- Deep Learning to ‘mine’ product tags for DBs

- (customer question in store: I want to have a perfume that looks like a ‘gold bar’)

- (name unknown, but shape and color known by customer)

- (databases - DBs – have no color or shape info)

- (shapes)

- (colors with another script)

- (APRIORI uses frequent itemsets & iterative approach)

- (FP-GROWTH uses frequent pattern tree-based approach)

- [12] Big Data Tips, Association Rules

- [13] Performance Evaluation Apriori vs. FP-Growth
Outline of the Course

1. Cloud Computing & Big Data Introduction
2. Machine Learning Models in Clouds
3. Apache Spark for Cloud Applications
4. Virtualization & Data Center Design
5. Map-Reduce Computing Paradigm
6. Deep Learning driven by Big Data
7. Deep Learning Applications in Clouds
8. Infrastructure-As-A-Service (IAAS)
10. Software-As-A-Service (SAAS)
11. Big Data Analytics & Cloud Data Mining
12. Docker & Container Management
13. OpenStack Cloud Operating System
14. Online Social Networking & Graph Databases
15. Big Data Streaming Tools & Applications
16. Epilogue

+ additional practical lectures & Webinars for our hands-on assignments in context

- Practical Topics
- Theoretical / Conceptual Topics

Practical Lecture 11.1 – Using Data Mining & Recommender Techniques in Clouds
Outline

- Using Data Mining Techniques in Clouds
  - Using Helmholtz Data Federation (HDF) Cloud with Jupyter @ JSC
  - Install MLxtend Library in HDF Environment via Pip Package Installer
  - Using Apriori Algorithm with Retail Shopping Data Example
  - Using FP-Growth Algorithm with Retail Shopping Data Example
  - Understanding Recommender Systems & Surprise Package Example

- Using Recommender Systems in Clouds
  - Google Colab Cloud & Movie Rental Recommendation Example
  - Using Collaborative Filtering Recommender & Optimization Approach
  - Understanding Hand-crafted vs. Automatically Learned Embeddings
  - Using Matrix Factorization Approaches as Simple Embedding Model
  - CRISP-DM Process in Context & Understanding different Phases

Promises from previous lecture(s):
- Practical Lecture 0.1: Lecture 11 will provide more insights into how the algorithm works & how they scale for big datasets in cloud computing environments
- Practical Lecture 0.1: Lecture 11 will provide insights about using the real dataset of the pieper perfume stores and its big data mining processing challenges
- Practical Lecture 0.1: Lecture 11 will provides more insights how to use configuration options in data mining algorithms to perform fine tuned data analysis
- Practical Lecture 3.1: Lecture 11 provides more details on using recommender engines & that are partly considered as data mining technique
- Lecture 5: Lecture 11 will provide more details on data analytics techniques using parallel computing for data mining applications in Clouds today
Using Data Mining Techniques in Clouds
Remote Access to HPC Systems: Jupyter @ Juelich Supercomputing Centre (JSC)

- Startup Remote Jupyter (Jupyter @ JSC)
  - Understanding differences between local laptop vs. remote cloud or HPC system
  - Understanding differences Jupyter vs. JupyterLab

[3] Jupyter @ JSC
Practical Lecture 11.1 – Using Data Mining & Recommender Techniques in Clouds

[DEMONSTRATION] Jupyter @ JSC – (cf. Lecture 0.1)

(EOSC service B2DROP, cf. Lecture 10)
Jupyter @ JSC – Register & Access

After login press ‘join project’ and fill out the information as below:

- Project ID: [your email]
- Optional additional information: 
  - Student in class

We will send an email to this address containing a link to complete the registration process.

[5] JuDoor
Helmholtz Data Federation (HDF) Cloud Computing Platform @ JSC

- Comprises OpenStack compute, network, and volume services as well as an integration with the DATA file system also available on the HPC systems
- Includes links to other services relevant for the EOSC Cloud (e.g., B2DROP academic dropbox)

Lecture 13 provides more details on the Cloud operating system OpenStack and its various forms of core building blocks for services
Jupyter @ JSC using the HDF-Cloud – Startup & Install MLxtend Library

```
(base) jsoyan@005f97a19787:~$ pip install mlxtend
Collecting mlxtend
  WARNING: Retrying (Retry(total=3, connect=None, read=None, redirect=None, status=None)) after connection broken by 'ReadTimeoutError('HTTPSConnectionPool(host='pypi.org', port=443): Read timed out. (read timeout=15)\n\nWARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after connection broken by 'ReadTimeoutError('HTTPSConnectionPool(host='pypi.org', port=443): Read timed out. (read timeout=15)
Collecting mlxtend
  WARNING: Retrying (Retry(total=5, connect=None, read=None, redirect=None, status=None)) after connection broken by 'ReadTimeoutError('HTTPSConnectionPool(host='pypi.org', port=443): Read timed out. (read timeout=15)
  Downloading mlxtend-0.17.3-py3-none-any.whl (1.3 MB)
Requirement already satisfied: joblib>=0.13.2 in /opt/conda/lib/python3.7/site-packages (from mlxtend) (0.14.1)
Requirement already satisfied: scipy>=1.2.1 in /opt/conda/lib/python3.7/site-packages (from mlxtend) (1.4.1)
Requirement already satisfied: mlkit-learn>=0.20.3 in /opt/conda/lib/python3.7/site-packages (from mlxtend) (0.22)
Requirement already satisfied: matplotlib>=3.0.0 in /opt/conda/lib/python3.7/site-packages (from mlxtend) (3.3.3)
Requirement already satisfied: requests in /opt/conda/lib/python3.7/site-packages (from mlxtend) (2.23.0)
Requirement already satisfied: numexpr>=2.16.2 in /opt/conda/lib/python3.7/site-packages (from mlxtend) (2.2.8)
Requirement already satisfied: pandas>=0.24.2 in /opt/conda/lib/python3.7/site-packages (from mlxtend) (1.0.5)
Requirement already satisfied: pillow>=0.4.4; python_version>='2.6',<='2.7.12' in /opt/conda/lib/python3.7/site-packages (from matplotlib>=3.0.0; python_version>='2.6') (7.0.1)
Requirement already satisfied: scikit-learn<0.13.0,>=0.12 in /opt/conda/lib/python3.7/site-packages (from mlxtend) (0.22)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from mlxtend) (1.15.0)
Successfully installed mlxtend-0.17.3
```

Practical Lecture 11.1 – Using Data Mining & Recommender Techniques in Clouds

10 / 50
Using Apriori Algorithm with Retail Shopping Data Example – cf. Lecture 11

(Use configuration parameter to finetune the results)

(load notebook & example data)

(Practical Lecture 11.1 – Using Data Mining & Recommender Techniques in Clouds)

[7] mlxtend lib, Apriori
Using FP-Growth Algorithm with Retail Shopping Data Example – cf. Lecture 11

### Code Snippet 1
```python
from mlxtend.preprocessing import TransactionEncoder
df = pd.DataFrame(df, columns='itemsets')

t = TransactionEncoder()
t.dataset = df
items = t.to_array()
df = pd.DataFrame(items, columns='itemsets')
```

### Code Snippet 2
```python
from mlxtend.frequent_patterns import fp_growth

data = [['Milk', 'Banana', 'Kidney Beans', 'Eggs', 'Yogurt', 'Tomato', 'Onion', 'Potato', 'Ice cream', 'Cereal'],
        ['Milk', 'Banana', 'Kidney Beans', 'Eggs', 'Yogurt', 'Tomato', 'Onion', 'Potato', 'Ice cream', 'Cereal'],
        ['Milk', 'Banana', 'Kidney Beans', 'Eggs', 'Yogurt', 'Tomato', 'Onion', 'Potato', 'Ice cream', 'Cereal'],
        ['Milk', 'Banana', 'Kidney Beans', 'Eggs', 'Yogurt', 'Tomato', 'Onion', 'Potato', 'Ice cream', 'Cereal'],
        ['Milk', 'Banana', 'Kidney Beans', 'Eggs', 'Yogurt', 'Tomato', 'Onion', 'Potato', 'Ice cream', 'Cereal']]

t = fp_growth(data, min_support=0.6)
support = t.support
```

### Code Snippet 3
```python
[0] 1.0 (Kidney Beans)
[1] 0.8 (Milk)
[2] 0.6 (Eggs)
[3] 0.6 (Yogurt)
[4] 0.6 (Tomato)
[5] 0.8 (Onion, Kidney Beans)
[6] 0.6 (Yogurt, Kidney Beans)
[7] 0.6 (Tomato, Milk)
[8] 0.6 (Onion, Eggs, Kidney Beans)
[9] 0.6 (Tomato, Milk, Kidney Beans)
```
1. Some pattern exists
2. No exact mathematical formula
3. Data exists
   - Idea ‘Learning from Big Data’
     - Shared with a wide variety of other disciplines
     - E.g. signal processing, big data mining, etc.
   - Challenges
     - Data is often complex
     - Requires ‘Big Data analytics’
     - Learning from data requires processing time → Clouds or High Performance Computing

- Machine learning is a very broad subject and goes from very abstract theory to extreme practice (‘rules of thumb’)
- Training machine learning models needs processing time (clouds or high performance computing)
- While data analysis is more describing the process of analysing the data, the term data analytics also includes and the necessary scalable or parallel infrastructure to perform analysis of ‘big data’
Other Mllib Algorithms – Partly Overlap with Data Mining Techniques – Revisited

- **Example: K-Means Clustering**

- **Example: Recommendation Engine**
  - **Using Collaborative Filtering via Google ‘Colab’**

[10] YouTube video, Visualization of K-Means Clustering

[8] Collaborative Filtering Recommender System on Colab
Another Form of ‘Data Mining’ using Recommender Systems – Overview

- **Content-based / Product-based Recommendation Systems**
  - E.g. Netflix user has watched many **cowboy movies** in the past (focus on product feature)
  - Recommendation: movie classified in the database as **having the ‘cowboy’ genre tag**
  - (not covered here as relatively straightforward to implement: e.g., DB lookup)
  - Might be still useful in combination with more elaborate systems if space in GUI is available

- **Collaborative Filtering-based / Customer-based Recommendation Systems**
  - E.g. Similarity of the customer ratings for products (not focus on product feature)
  - Identify: looking at **other customers** that are most similar to this customer
  - Recommendation: products that are liked or preferred by the other ‘similar customers’
  - Focus in this lecture and on one concrete algorithm: **matrix factorization**
Collaborative Filtering – Methodology

- **Methodology**
  - Recommendation systems that leverage existing shopping/watching/listening behaviour patterns
  - Predicts what customers could like in future based on previous customers behavior patterns
  - Assumes that customers like products similar to other products they like, but also products that are liked by other people with similar taste

- **Approach**
  - Uses different machine learning methods
  - Collaborative filtering is a general concept and there are many algorithms (e.g., singular vector decomposition, neural networks, etc.)
  - Two main techniques: memory & model-based collaborative filtering
**Collaborative Filtering – Memory and Model-based Techniques**

- **Two quite different approaches for the same problem**
  - Popular approaches are based on low-dimensional factor models these days
  - Different approaches have different advantages and disadvantages and could be used both (if needed)

**Memory-based approach**
- Find similar users based on cosine similarity or pearson correlation and take weighted avg. of ratings
  - **Advantage**: Easy creation and explanability of results
  - **Disadvantage**: Performance reduces when data is sparse. So, non scalable

**Model-based approach**
- Use machine learning to find user ratings of unrated items, e.g, PCA, SVD, Neural Nets, Matrix Factorization
  - **Advantage**: Dimentionality reduction deals with missing/sparse data
  - **Disadvantage**: Inference is intractable because of hidden/latent factors

- **Memory-based approaches** for Collaborative Filtering can quickly become computationally expensive, but enables better explainability
- **Model-based approaches** work with dimensionality reduction, but results are not easy to explain to store managers

*(this lecture will focus on Matrix Factorization – with a simple demonstration)*
Collaborative Filtering – Famous Dataset Example & Challenges

- **Famous Example in Retail**
  - Illustrating the underlying assumption that if a customer A has the same opinion/rating as a customer B on a certain product...
  - ... A is more likely to have B’s opinion on a different product as well than that of a randomly chosen customer

- **Challenges**
  - In real datasets millions or billions of transactions are used, including ratings if possible (otherwise buy & not buy only)
  - Unfortunately in practice not always ratings are existing

- **Algorithms Benefit**
  - Automation of the process using collaborative filtering algorithms
  - Patterns help to identify new opportunities and ways for cross-selling products to customers
Matrix Factorization-based Algorithms & Tool Support Examples

- Deep Learning techniques change the experiences over the last decades and become more popular with very good accuracies and good packages (e.g., fast.ai) using innovative HPC & Cloud computing

- Scikit-surprise is a package specifically designed for recommendation systems and includes a variety of algorithms in Python useful for data mining tasks

Practical Lecture 11.1 – Using Data Mining & Recommender Techniques in Clouds

[16] Towards Data Science, Various CFs

[19] fast.ai library

[18] scikit-surprise library
Systematic Process to Support Learning From Data – Revisited

- Systematic data analysis guided by a ‘standard process’
  - Cross-Industry Standard Process for Data Mining (CRISP-DM)

A data mining project is guided by these six phases:
1. Problem Understanding;
2. Data Understanding;
3. Data Preparation;
4. Modeling;
5. Evaluation;
6. Deployment

Learning takes place during the modeling and evaluation phases.

CRISP-DM Model
Using Recommender Systems in Clouds
Movie Recommendation Example using Collaborative Filtering Techniques

- **Given movie feedback matrix**
  - Row represents a user
  - Column represents a movie

- **Feedback encoding**
  - One of two categories: explicit & implicit feedback
  - Example: feedback matrix is binary with a value of 1 that indicates interest in the movie

- **Embeddings Approach**
  - Can be learned automatically (no need for hand-engineering of features)
  - 1D Embedding Example

- **Collaborative filtering uses similarities between users and items simultaneously to provide recommendations**
- **Collaborative filtering models can recommend an item to user A based on the interests of a similar user B**
- **Explicit feedback in collaborative filtering means that users specify how much they liked a particular movie by providing a numerical rating**
- **Implicit feedback in collaborative filtering means that if a user watches a movie, the system infers that the user is interested**
- **The goal of collaborative filtering systems in movie ratings are to recommend (1) similarity to movies the user has liked in the past, and (2) movies that similar users liked and are not seen yet**

---

[1] Google Colab Exercise

- product of movie embedding & user embedding should be higher (closer to 1) for movies that we expect the user to like
- users watched these movies & preferences are well explained by this feature
Collaborative Filtering Techniques & Automatically Learned Embeddings

- **Embeddings Approach**
  - 1D feature not enough to explain preferences well
  - 2D Embedding Example: add a second feature
  - E.g. the degree to which each movie is a blockbuster or an arthouse movie

- The embedding space is an abstract representation common to both items and users, in which we can measure similarity or relevance using a similarity metric

- Embeddings can be learned automatically, which is the power of collaborative filtering models
  - Embeddings of users with similar preferences will be close together
  - Embeddings of movies liked by similar users will be close in the embedding space

Example: $(0.1 \times 1) + (1 \times -1) = -0.9$
Matrix Factorization Approaches as Simple Embedding Model

- **Given movie feedback matrix** $A$
  - Row represents a user: $m$ users
  - Column represents a movie: $n$ movies
- **Model learns automatically:**
  - User embedding matrix $U$
    (row $i$ is the embedding for user $i$)
  - Movie embedding matrix $V$
    (row $j$ is the embedding for movie $j$)
  - Learning = minimize ‘errors’
- **Embeddings**
  - Have an embedding dimension $d$
    (here we have a 2D example)
  - Learned such that the product $UV^T$ is a good approximation of matrix $A$

---

**Objective function** (cf. Lecture 6)

$$
\min_{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}} \sum_{(i,j) \in \text{obs}} (A_{ij} - (U_i \cdot V_j))^2
$$

(optimization problem can be solved with SGD for example, cf. Lecture 6 & 7)

**Optimization**

- (minimize the sum of squared errors over all pairs of observed entries = Observed Only MF)
- (treat the unobserved values as zero, and sum over all entries in the matrix)

- $\text{SVD} = \min_{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}} \|A - UV^T\|_F^2$
  - (SVD = singular value decomposition, poor generalization in sparse movie rating matrix setup)

---

**Google Colab Exercise**

Practical Lecture 11.1 – Using Data Mining & Recommender Techniques in Clouds
Google Colab – Movie Rental Recommendation Notebook & Create Own Copy

[1] Google Colab Exercise

[9] Google Colaboratory

Practical Lecture 11.1 – Using Data Mining & Recommender Techniques in Clouds
Recommendation engine is built using TensorFlow deep learning package
Recommendation engine script requires imports from various useful libraries: e.g., pandas, numpy, sklearn, etc.

Manipulating Pandas DataFrame options and adding selected convenient functions

Installing Altair and importing the library
Altair is a library for declarative visualization in Python and offers very good interactive visualizations for data analysis

Installing and importing gspread that is a Python API for Google Sheets
Optional as this enables own ratings to inject into the recommendation system (if needed)
Movielens Dataset

- http://files.grouplens.org/datasets/movielens/ml-100k.zip

```
# Title Load the Movielens data (run this cell).

# Download Movielens data.
print("Downloading movielens data...")
from urllib.request import urlretrieve
import zipfile

urlretrieve("http://files.grouplens.org/datasets/movielens/ml-100k.zip", "movielens.zip")
zipref = zipfile.ZipFile("movielens.zip", "r")
zipref.extractall()
print("

```
Google Colab – Movie Rental Recommendation Notebook – Check Dataset

- Check available data
  - Users
  - Ratings
  - Genre
  - Movies

- The MovieLens dataset consists of data about users, ratings about movies, the genre of movies and information about movies.

```python
# Load each data set (users, movies, and ratings).
users_cols = ['user_id', 'age', 'sex', 'occupation', 'zip_code']
users = pd.read_csv('ml-100k/u.user', sep='\|', names=users_cols, encoding='latin-1')

ratings_cols = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_csv('ml-100k/u.data', sep='\t', names=ratings_cols, encoding='latin-1')

# The movies file contains a binary feature for each genre.
movies_cols = ['movie_id', 'title', 'release_date', 'video_release_date', 'imdb_url']
+ genre_cols
movies = pd.read_csv('ml-100k/u.item', sep='|', names=movies_cols, encoding='latin-1')
```

[1] Google Colab Exercise
Like in previous examples of machine learning and data mining, also for learning the recommendation engine we split the available dataset into two disjunct datasets: train and test datasets.

### Training Examples

\( (x_1, y_1), \ldots, (x_N, y_N) \)

(historical records, groundtruth data, examples)

---

(understanding some basic statistics describing numeric user features)

---

Downloading movielens data...

Done. Dataset contains:

b'943 users\n1682 items\n100000 ratings\n'

---

Warning: you are connected to a GPU runtime, but not utilizing the GPU. 

[1] Google Colab Exercise
Interactive data visualization techniques are an important approach to have a better data understanding:

- E.g. histograms to further understand the distribution of the users, distribution of ratings per user, occupation in the right chart filters the data by that occupation
- E.g. data understanding observed: very high number of ratings from students → recommendation bias

(students have a lot of records compared to other users who rated movies)
Google Colab – Movie Rental Recommendation Notebook – Movie Data

```python
movies_ratings = movies.merge(
    ratings
    .groupby('movie_id', as_index=False)
    .agg({'rating': ['count', 'mean']})
    .flatten_rows(),
    on='movie_id')

genre_filter = alt.selection_multi(fields=['genre'])
genre_chart = alt.Chart().mark_bar().encode(
    x='count()',
    y=alt.Y('genre'),
    color=alt.condition(
        genre_filter,
        alt.Color('genre:N'),
        alt.value('lightgray'))
).properties(height=300, selection=genre_filter)
```

```python
movies_ratings[['title', 'rating count', 'rating mean']].
.sort_values('rating count', ascending=False)
.head(10)
```

[1] Google Colab Exercise
Starting the Modeling Approach – Matrix Factorization – Revisited

- Factorize the ratings matrix $A$
  - Into the product of a user embedding matrix $U$ and movie embedding matrix $V$

$$A \approx UV^\top$$

- Using TensorFlow (cf. Lecture 6 & 7) SparseTensor
  - Sparse representation of the rating matrix $A$ required
  - Rating matrix could be very large (many users and many movies)
  - Example:

<table>
<thead>
<tr>
<th>user_id</th>
<th>movie_id</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5.0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3.0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1.0</td>
</tr>
</tbody>
</table>

- A rating matrix used in collaborative filtering with matrix factorization in the movie recommendation example shows that most of the entries are unobserved, since users will only rate a small subset of all movies

- Example:

$$A = \begin{bmatrix} 5.0 & 3.0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

- EfficientSparseTensor representation example

Practical Lecture 11.1 – Using Data Mining & Recommender Techniques in Clouds

Google Colab Exercise
(a function that maps from ratings DataFrame to a TensorFlow SparseTensor)

(solution)

(solution, but infeasible for 'big data')
Google Colab – Movie Rental Recommendation Notebook – Mean Squarer Error

- Error to guide the learning process
  - Calculating the error to improve learning and to measure the approximation error
  - Model approximates the ratings matrix $A$ by a low-rank product $U^T V$
- First Approach (last slide)
  - Initially Mean Squared Error (MSE) of observed entries only
    
    \[
    MSE(A, U^T V) = \frac{1}{|\Omega|} \sum_{ij \in \Omega} (A_{ij} - (U^T V)_{ij})^2 \\
    = \frac{1}{|\Omega|} \left( \sum_{ij \in \Omega} A_{ij}^2 - \frac{1}{|\Omega|} \left( \sum_{ij \in \Omega} A_{ij} \right)^2 \right) \\
    \]
    
    $|\Omega|$ is the set of observed ratings, and $|\Omega|$ is the cardinality of $\Omega$.

- Second Approach (above)
  - Only gather the embeddings of the observed pairs, then compute their dot products
  - More efficient to fit into memory for ‘big data’
    
    \[O(|\Omega|d)\] where $d$ is the embedding dimension. (Note: next notebook steps in adding new ratings via spreadsheet is not used here)

- Practical Lecture 11.1 – Using Data Mining & Recommender Techniques in Clouds

1. Google Colab Exercise (learning goal)
2. Memory footprint
3. Simple example here ok since it fits into memory
4. Embedding dimension on order of 10

[34] / 50
**Approach**

- **Class to train a matrix factorization model using stochastic gradient descent (cf. Lecture 6)**

```python
def train(self, num_iterations=100, learning_rate=1.0, plot_results=False, optimizer=GradientDescentOptimizer):
    # Training loop
    ...  
    # Plot the metrics
    ...  
    return results
```

- **After training a matrix factorization model using stochastic gradient descent (SGD)** we obtain the trained embeddings via a `model.embeddings` dictionary that in turn is used to perform recommendations.
Latent features are learned, but are hard to explain.

Latent features are known as ‘hidden features’ to distinguish them from observed features.

Latent features are computed from observed features using matrix factorization techniques.

(looks like we lower the train and test error successfully)

(iteration 1000: train_error=0.372467, test_error=1.334507)
Google Colab – Movie Rental Recommendation Notebook – Model Evaluation

- **Evaluation Viewpoints**
  - Movie recommendation
  - Nearest neighbors of some movies
  - Norms of the movie embeddings
  - (Visualizing the embedding in a projected embedding space)

- Computes the scores of the candidates
  - Different similarity measures will yield different results

```
[22] def user_recommendations(model, measure='dot', exclude_rated=False, k=5):
    scores = compute_scores(model, measure=model.embeddings[user_id],
                            scores_key='score',
                            movie_id=model.embeddings[movie_id],
                            norm=model.embeddings['norm']
                            df = pd.DataFrame({
                                'score': list(scores),
                            'movie_id': movies['movie_id'],
                            'title': movies['title'],
                            'genres': movies['all_genres']})

[23] def movie_neighbors(model, title_substring, measure='dot', k=5):
    ids = movies[movies['title'].str.contains(title_substring)].index.values
    if len(ids) == 0:
        raise ValueError("Found no movies with title \"{}\" \& title_substring")
    if len(ids) > 1:
        print("Found more than one matching movie. Other candidates: \{}").format(ids)

[21] dot_product: the score of item i is \( <w_i, V_j> \).
    cosine: the score of item i is \( \frac{w_i \cdot V_j}{||w_i|| ||V_j||} \).
```

(1) Google Colab Exercise

Practical Lecture 11.1 – Using Data Mining & Recommender Techniques in Clouds
Validating recommendations

- Using different score measures

Using the dot-product score for model evaluation in training a matrix factorization model, the model tends to recommend popular movies. Popular movies are explained by the fact that in matrix factorization models, the norm of the embedding is often correlated with popularity. Popular movies have a larger norm that makes the model more likely to recommend more popular movies.

(Confirm this by sorting the movies by their embedding norm)

(manual validation, would you assume this to be right?)

(how good is the model really?)

[23] user_recommendations(model, measure=cosine, k=5)

[24] movie_neighbors(model, 'Aladdin', k=5)

[25] Embedding Visualization code (run this cell)

[26] movie_embedding_norm(model)

[1] Google Colab Exercise
Working on Hyperparameters

- Change initial standard deviation hyperparameter `init_stddev`
- How does this affect the embedding norm distribution, and the ranking of the top-norm movies?

(new model training after tuning hyperparameter)

(manual validation, changes in recommendation observed)

(remember much more record counts for drama in movies than others → bias)

(how good is the model?)

(looks like we lower the train and test error successfully)

[1] Google Colab Exercise
**Evaluation Viewpoints**

- Movie recommendation
- Nearest neighbors of some movies
- Norms of the movie embeddings
- Visualizing the embedding in a projected embedding space
- Example t-SNE approach

---

- It is hard to visualize model embeddings in a higher-dimensional space (>3, here in this example the embedding dimension is 30) so one idea is to project the embeddings to a lower dimensional space
- t-distributed Stochastic Neighbor Embedding (t-SNE) is an algorithm that projects the embeddings while attempting to preserve their pairwise distances
- t-SNE is used for visualization of models but should be handled with care (embeddings hard to visualize correctly)

---

[26] t-SNE information

[1] Google Colab Exercise

Practical Lecture 11.1 – Using Data Mining & Recommender Techniques in Clouds
Learned insights from Evaluation

- Poor model quality

In learning a recommendation engine with matrix multiplication a potential poor model quality can occur when learning only on the observed part of the rating matrix and not using regularization.

- The reason for poor model quality in this case is known as ‘folding’: the model does not learn how to place the embeddings of irrelevant movies.

Regularization for Matrix Factorization

- Regularization often inherent in model building (e.g., logistic regression, or support vector machines)
- Add two types of regularization terms that will address this issue:

  - Regularization of the model parameters. This is a common $\ell_2$ regularization term on the embedding matrices, given by $r(U, V) = \frac{1}{N} \sum_i \|U_i\|^2 + \frac{1}{M} \sum_j \|V_j\|^2$.
  - A global prior that pushes the prediction of any pair towards zero, called the gravity term. This is given by $g(U, V) = \frac{1}{MN} \sum_{i=1}^{N} \sum_{j=1}^{M} (U_i, V_j)^2$.

The total loss with two new hyper-parameters for tuning: the amount of regularization is:

$$\frac{1}{|O|} \sum_{(i,j) \in O} (A_{ij} - (U_i, V_j))^2 + \lambda_U r(U, V) + \lambda_V g(U, V)$$

where $\lambda_U$ and $\lambda_V$ are two regularization coefficients (hyper-parameters).
Hyper-Parameter Tuning

- Most complex aspect in machine learning & data mining
- Takes massive human time with a lot of possibilities to choose from
- E.g. regularization_coeff
- E.g. gravity_coeff
- E.g. embedding_dim
- E.g. init_stddev
- E.g. num_iterations (fitting over time)
- E.g. learning_rate (relevant for SGD)

Several techniques have been established to help with a more systematic hyper-parameter tuning, like AutoML techniques or genetic algorithms for example

Still many modeling activities require human intervention to really tune a machine learning or data mining model really right so that it generalizes well
Back to Model Evaluation

- After Regularization and using different learning scheme
  - E.g. dot-product, cosine, norms

[32] user_recommendations(reg_model, DOT, exclude_rate=True, k=10)
[33] movie_neighbors(reg_model, "Aladdin", DOT)
    movie_neighbors(reg_model, "Aladdin", COSINN)

(exclude_rate: own rated ones via spreadsheet)

[34] movie_embedding_norm(model, model_lowinit, reg_model)
    (comparing norms between model & new regularized model)

We seem to improve the modeling.

Practical Lecture 11.1 – Using Data Mining & Recommender Techniques in Clouds
Continue Model Evaluation

- Visualizing the embedding in a projected embedding space
- Example t-SNE approach

The final model evaluation after regularization reveals that the embeddings have a lot more structure than the unregularized model case.

Examples include different genres where one can observe how they tend to form clusters (e.g., Horror, Animation and Children), but ‘never perfect’

More specialized algorithms such as Alternating Least Squares (ALS) might improve the modeling.
Lecture Bibliography
Lecture Bibliography (1)

- [3] Jupyter @ Juelich Supercomputing Centre, Online: https://jupyter-jsc.fz-juelich.de/
- [10] YouTube Video, ‘Visualization of k-means clustering’, Online: https://www.youtube.com/watch?v=nXY6PxAaOk0
Lecture Bibliography (2)

- [14] Pieper.de Duefte, Online: https://www.pieper.de/duefte/
- [16] Various Implementations of Collaborative Filtering, Online: https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0
- [17] Collaborative Filtering and Embeddings — Part 1, Online: https://towardsdatascience.com/collaborative-filtering-and-embeddings-part-1-63b00b9739ce
- [18] Python scikit-surprise, Online: http://surpriselib.com/
- [19] fast.ai library, Online: https://www.fast.ai/
- [22] MovieLens Dataset, Online: https://grouplens.org/datasets/movielens/
Lecture Bibliography (3)

- [24] Project Jupyter, Online: https://jupyter.org/
- [25] Python Programming Language, Online: https://www.python.org/
- [26] How to Use t-SNE Effectively, Online: https://distill.pub/2016/misread-tsne/
Acknowledgements – High Productivity Data Processing Research Group

This research group has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 763558 (DEEP-EST EU Project)