IMDb Movies Recommender System and Genre Prediction

Created by Maria Fain, Veronija Kodovska and Fabijan Bošnjak

Our project

Movie Recommender System

- uses the ratings dataset
- gives a ranked list of movies for a user

Text classification Genre Prediction

- uses the movies dataset
- predicts movie genres based on the movie overview

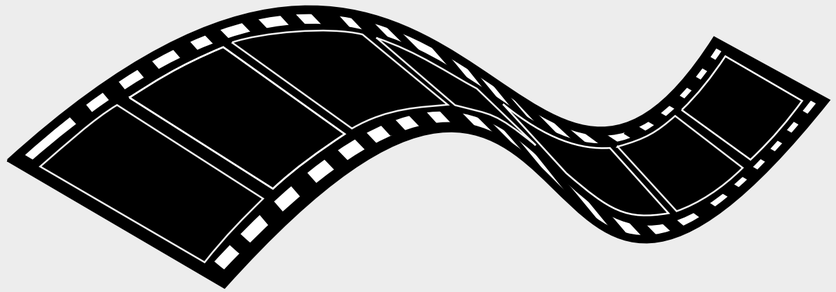
Our datasets

Movies

- genres
- movieId
- overview
- title

Ratings

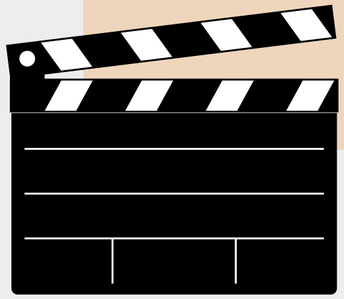
- userId
- movieId
- rating
- timestamp



Movies dataset description & preprocessing

Datasets found on kaggle

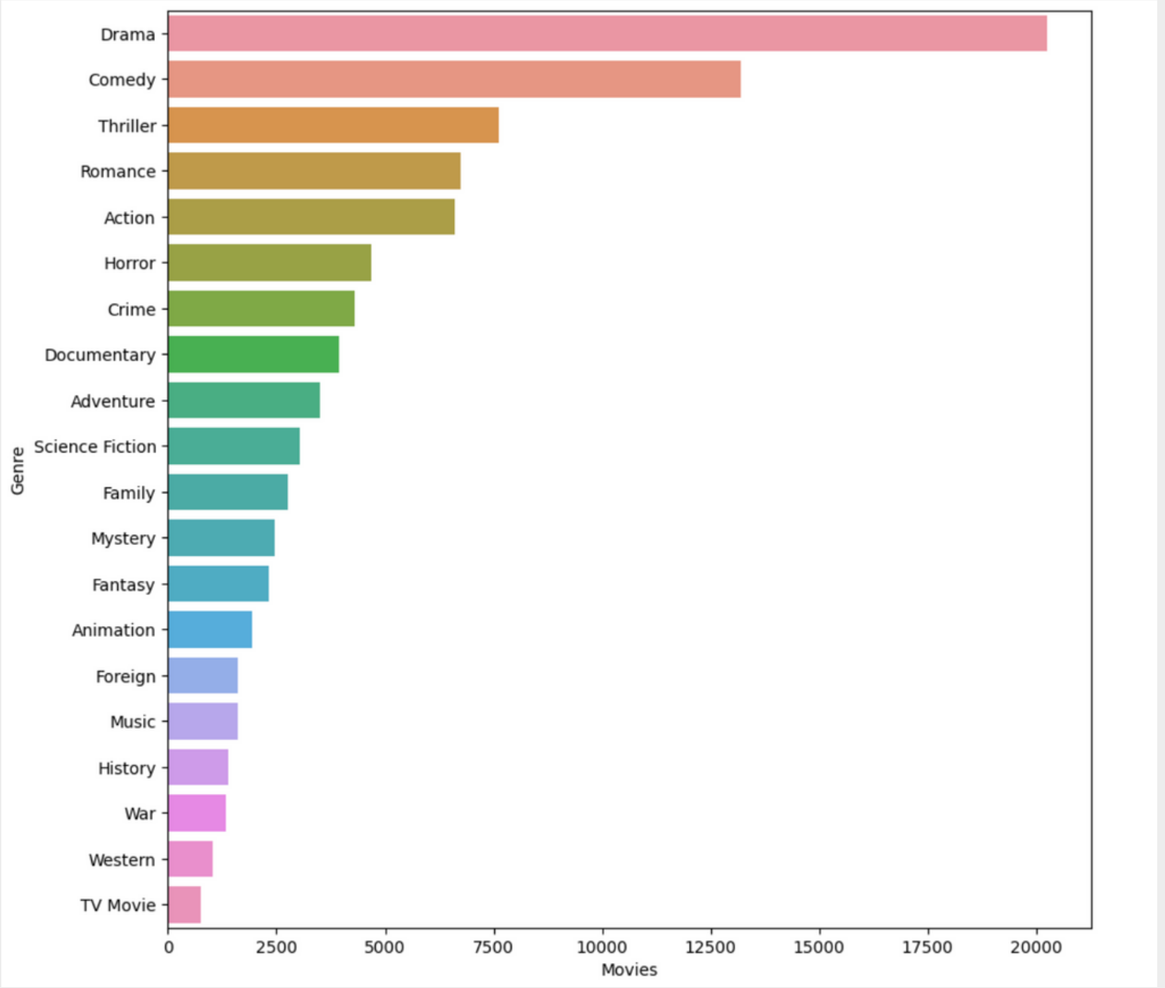
- converted genres from JSON to a list of strings, ex. [Drama]
- dropped movies without genres
- dropped movies with empty titles and overviews or overviews without meaning
- total of 42183 rows × 4 columns



```
movies.describe()
```

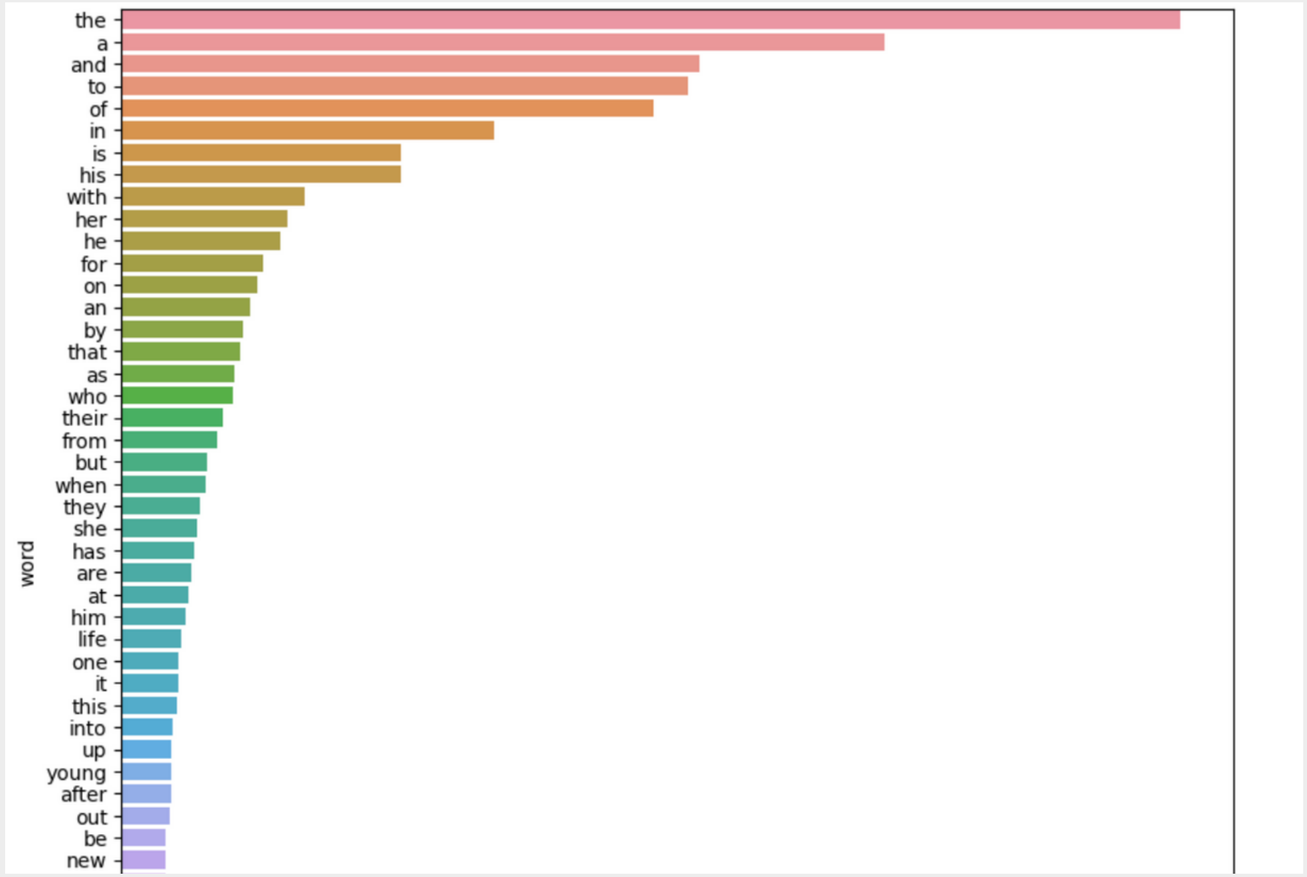
	genres	movieId	overview	title
count	45466	45466	44512	45460
unique	4069	45436	44307	42277
top	[{'id': 18, 'name': 'Drama'}]	141971	No overview found.	Cinderella
freq	5000	3	133	11

Chart of unique genres

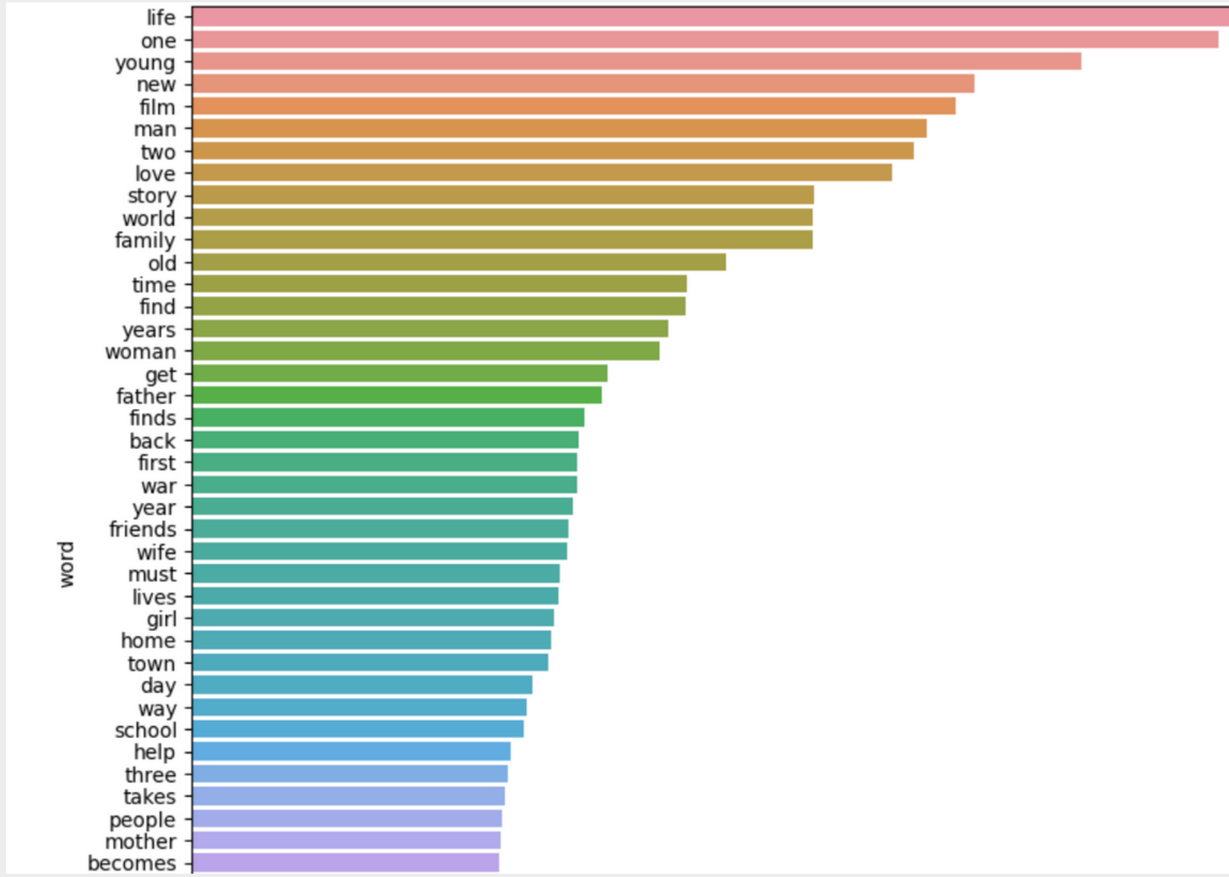


Visualization of our data

Barplot before cleaning the data and removing the stopwords



Barplot after cleaning the data and removing the stopwords



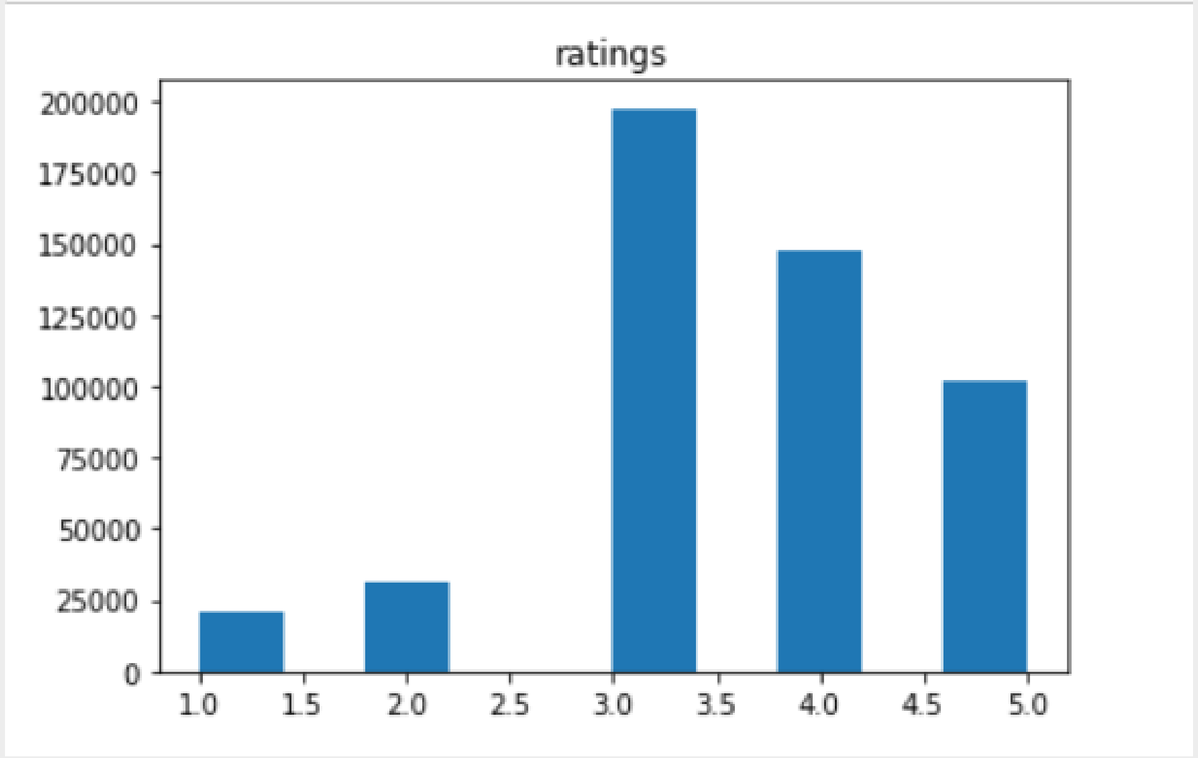
Word Cloud for overviews

Ratings dataset description & preprocessing

- converted timestamp to ISO format, ex. 1995-01-09 11:46:44
- decreased the size of the original dataset which was 26,024,289 rows and exported the smaller dataset to .csv for faster loading
- there were no empty values
- dropped duplicates
- 500 000 rows × 4 columns

	userId	movieId	rating	timestamp
0	1	110	1.0	1425941529
1	1	147	4.5	1425942435
2	1	858	5.0	1425941523
3	1	1221	5.0	1425941546
4	1	1246	5.0	1425941556

Visualization of ratings



- class imbalance between the lower rated movies and those with higher ratings

Text classification

Movie genre prediction based on given overview

Steps

1. Convert each **genre** to a separate **binary class**
2. Extract **most frequent words** from overivews
3. Train and test the model
4. Custom tests

Used models

- logistic regression
- random forest
- naive bayes

Python packages or modules used

- **sklearn MultiLabelBinarizer**: Used to convert each genre to a separate binary class
- **sklearn TfidfVectorizer**: Used to extract the top 10000 most frequent words from overviews and transform text data into TF-IDF vectors
- **sklearn train_test_split**: Used to split the dataset into training and test sets
- **sklearn multiclass OneVsRestClassifier**: Used to handle multi-label classification. It decomposes a multiclass problem into multiple binary classification subproblems.
- **sklearn LogisticRegression, MultinomialNB, RandomForestClassifier**

Training models for text classification

20 different output classes and using one vs rest approach caused:

- lower recall
- class imbalance
- high precision
- blank predictions

Our solution to this problem is a function that:

- handle cases where all predicted probabilities are **below** the threshold
- choose the class with the **highest** probability
- ensures that at least one class is predicted

Results

	Precision	Recall	F1-score
Logistic Regression	0.70	0.49	0.54
Random Forest	0.65	0.40	0.47
Naive Bayes	0.65	0.39	0.46

Example of use

- Unseen example: Movie - "Waiting to Exhale"
  - genres: Comedy, Drama, Romance
- Models were given a cleaned overview

- Random Forest
  - predicted only Drama

- Logistic Regression
  - predicted Comedy and Drama

- Naive Bayes
  - predicted only Drama

Movie Recommender System

- combining collaborative filtering and content based approaches
- avoiding "cold-start" - users with min 20 rated items
- train and test split
  - 70% - 30%
  - stratified splitting strategy
- utility matrix -> sparse matrix

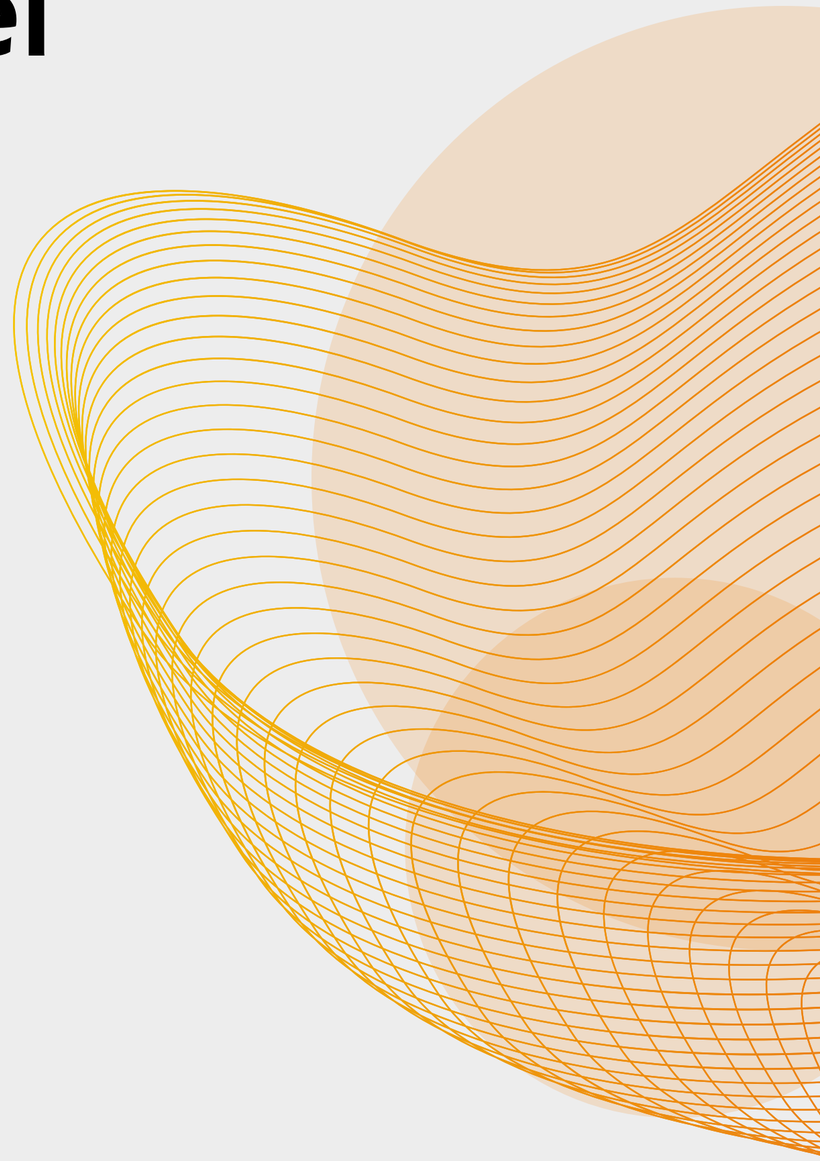
Python packages or modules used

- NumPy, Pandas
- Recommenders - for stratified splitting
- SciPy - for sparse matrixes
- Surprise - for SVD and Grid Search
- Sklearn - for evaluation metrics
- XGBoost

Preparing data for our model

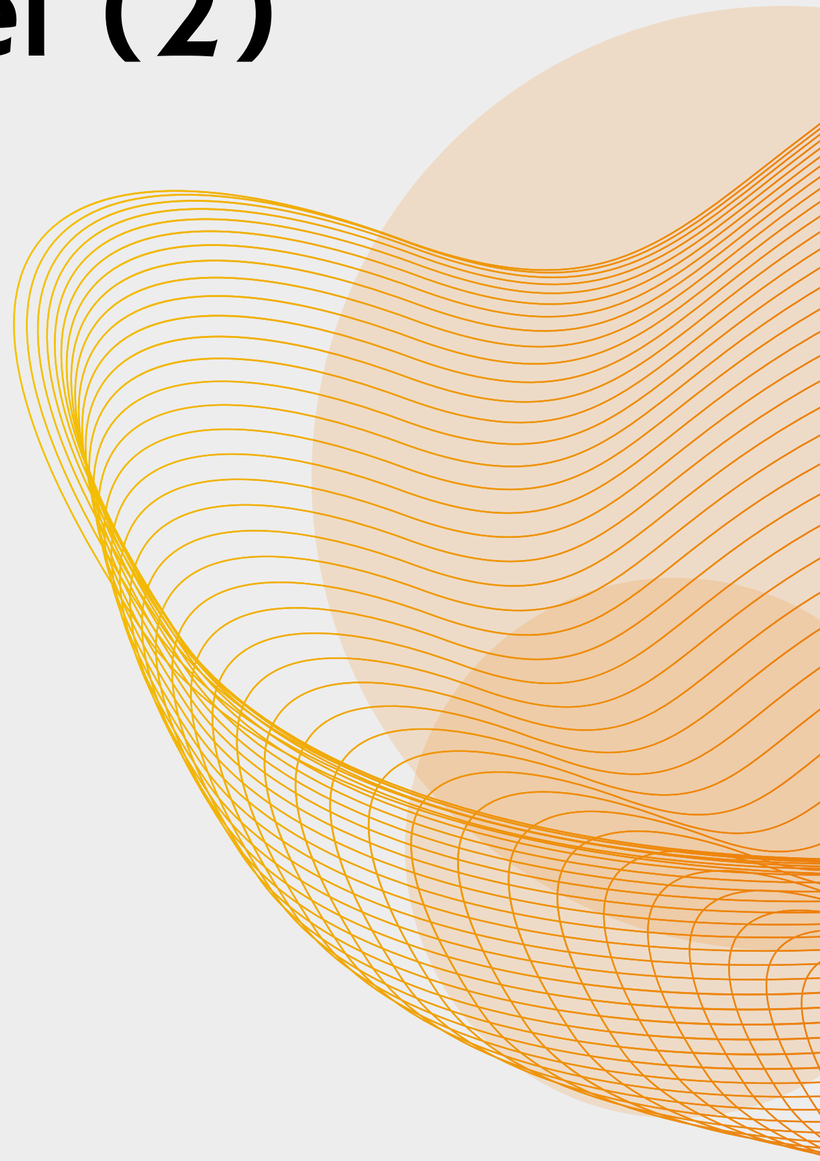
- Collaborative filtering method
  - matrix factorization - SVD
  - predictions used as a column in the final data

- Content based method
  - finding similar movies with cosine similarity



Preparing data for our model (2)

- implemented a function `get_final_data()`
- the function returns a Dataframe that will be used for training the model
- we called the function separately for the train and test set to avoid data leakage

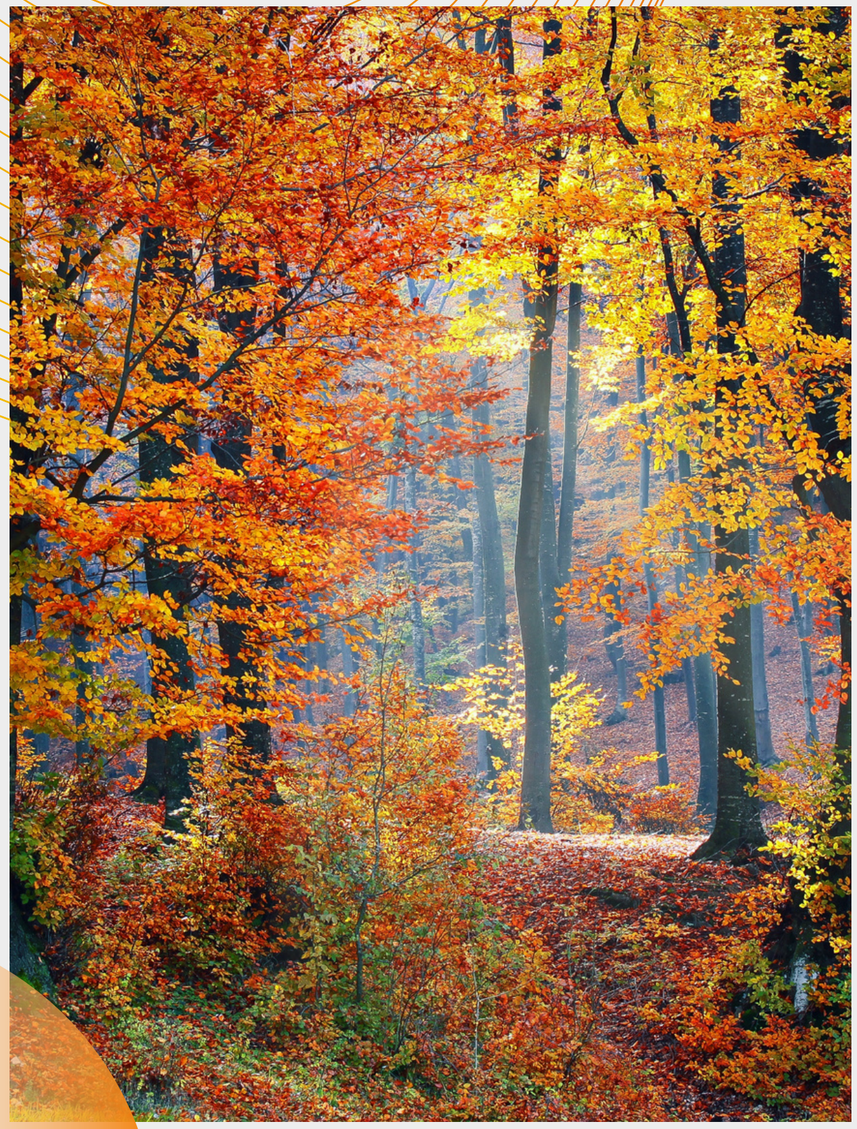


Data for the model

	userId	movieId	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	mf_svd
0	2	0	3.718811	3.0	3.0	4.0	4.0	3.0	2.0	5.0	5.0	4.0	4.0	3.780488	3.46	3.0	4.027541
0	16	0	3.718811	4.0	2.0	4.0	4.0	4.0	3.0	5.0	4.0	5.0	5.0	3.979592	3.46	2.0	3.650898
0	51	0	3.718811	3.0	4.0	3.0	3.0	3.0	5.0	5.0	5.0	5.0	3.0	3.816092	3.46	3.0	3.475944
0	72	0	3.718811	3.0	4.0	3.0	5.0	3.0	3.0	4.0	4.0	3.0	5.0	3.967742	3.46	3.0	3.563394
0	126	0	3.718811	3.0	4.0	3.0	3.0	4.0	3.0	4.0	4.0	3.0	4.0	3.392405	3.46	3.0	2.864823

Final time passed: 1:16:19.795123

- GAvg - global average rating
- sur[1-5] – top 5 similar user ratings of the movie for the user-movie pair
- smr[1-5] – top 5 similar movie ratings of the movie for the user-movie pair
- UAvg – user average rating
- MAvg – movie average rating
- mf_svd – the SVD predictions calculated previously



Model

- Extreme Gradient Boosting - XGBoost
- Trained with 244961 rows and 14 features from the final data

Results

- RMSE: 0.815
- MAE: 0.636

- converted predictions to a binary scale
  - >= 3.0 "good ratings" - 1
  - <= 3.0 "bad ratings" - 0
- classification_report
- class imbalance causes lower precision and recall for the class with less examples

	precision	recall	f1-score	support
0	0.36	0.55	0.44	11209
1	0.94	0.88	0.91	93904
accuracy			0.85	105113
macro avg	0.65	0.72	0.67	105113
weighted avg	0.88	0.85	0.86	105113

Giving recommendations

- implemented a function for predicting top n movies for a user from the test set

	userId	movieId	title
0	198	153	Lost in Translation
1	198	339	Night on Earth
2	198	160	The Arrival of a Train at La Ciotat
3	198	553	Dogville
4	198	186	Lucky Number Slevin
5	198	592	The Conversation
6	198	19	Metropolis
7	198	185	A Clockwork Orange
8	198	225	Man of Iron
9	198	344	Bang, Boom, Bang

userId	movieId	rating	title
198	553	4.0	Dogville
198	225	4.0	Man of Iron
198	344	3.0	Bang, Boom, Bang
198	185	4.0	A Clockwork Orange
198	153	3.0	Lost in Translation
198	160	2.0	The Arrival of a Train at La Ciotat
198	339	4.0	Night on Earth
198	19	1.0	Metropolis
198	592	1.0	The Conversation
198	186	3.0	Lucky Number Slevin

Evaluating recommendations

- implemented a function for calculating precision for the top n predicted ratings

- $$\text{precision} = \frac{TP}{TP+FN}$$

$$= \frac{|\text{good movies recommended}|}{|\text{all recommendations}|}$$

- for our example, precision = 0.7

Giving unrated movies recommendation

- implemented a function for predicting top n unrated movies for a user

	userId	movieId	title
0	198	580	Jaws: The Revenge
1	198	338	Good bye, Lenin!
2	198	254	King Kong
3	198	389	12 Angry Men
4	198	390	Lisbon Story
5	198	342	Summer Storm
6	198	271	Ronja Robbersdaughter
7	198	241	Natural Born Killers
8	198	211	Berlin is in Germany
9	198	503	Poseidon

Thank You