**CAPSTONE PROJECT**

**INTRODUCTION TO DATA SCIENCE REPORT**

1. **Overview**

One of the major families of applications of machine learning in the information technology sector is the ability to make recommendations of items to potential users or customers. In this project, we attempt to build various kinds of recommendation engines.

1. **Introduction**

Recommendation systems use ratings that users have given to items to make specific recommendations. For example, Amazon sells many products to many customers and permit these customers to rate their products so they are able to collect massive datasets. The same could be done for other items as movies in our case. Recommendation systems are one of the most used models in machine learning algorithms. For this project, we will focus on creating a movie recommendation system using Full MovieLens Dataset.

1. **Aim of the project**

The aim of this project is to train a machine learning algorithm …

The value used to evaluate algorithm performance is the Root Mean Square Error (RMSE). RMSE is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are, RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. The effect of each error on RMSE is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSE.

**3. Dataset**

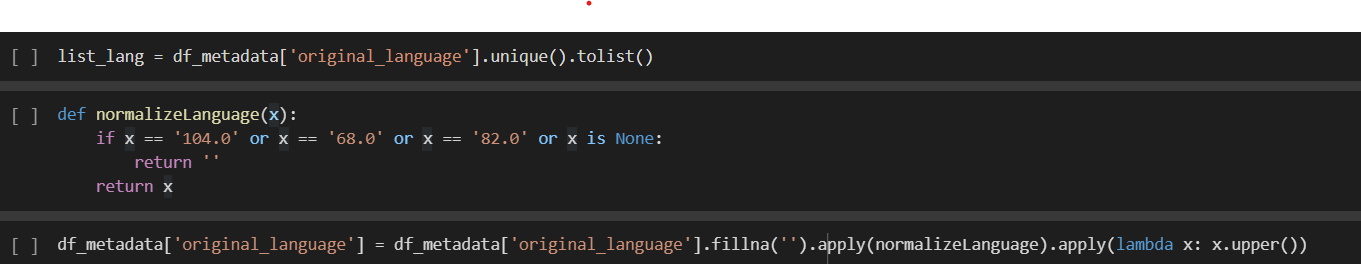
- The data used in this project has been obtained from MovieLens. It has a publicity dataset that contains 26 million ratings from 270000 users for all 45000 movies. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB (The Movie Database) vote counts and vote averages.

- The following files were used in the project:

* **credits.csv:** Consists of Cast and Crew Information for all our movies.
* **links.csv:** Contains the TMDB and IMDB IDs of all the movies featured in the Full MovieLens dataset.
* **movies\_metadata.csv:** Contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.
* **keywords.csv:** Contains the movie plot keywords for our MovieLens movies.
* **ratings.csv:** Contains Cast and Crew Information for all movies in the movies\_metadata.csv file.

- Cleaning data:

Because this dataset contains some features which had null data, called NaN, so we will clean this dataset by using *fillna()* with some specific columns. For example, with “original\_language” column, any fields which contains NaN value, we will change it to blank field. With fields which have good value, we convert them to uppercase.



Most of the features were converted into a Python basic type such as integer, string, float by removing all unclean values.

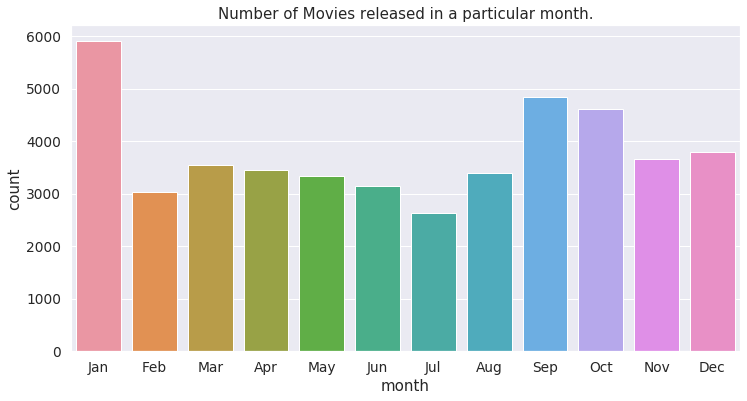
- Some analysis:

* Most popular destination for shooting movies:



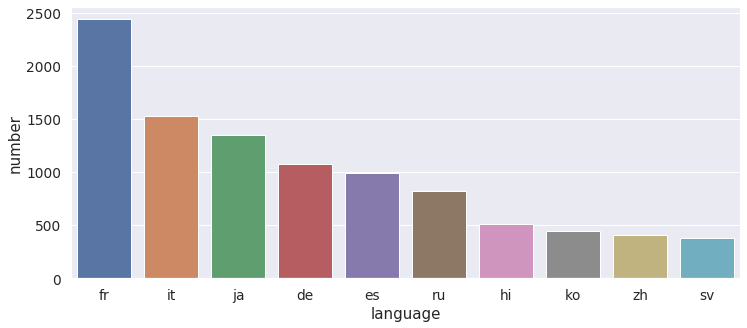
As you can see, the United States of America is the most popular destination of production for movies given that this dataset largely consists of English movies. With the UK, France, Germany & Italy in top 5, Europe is also an popular location. Japan & India are the most popular Asian countries with positions 7 and 10.

* Number of movies released in a particular month:



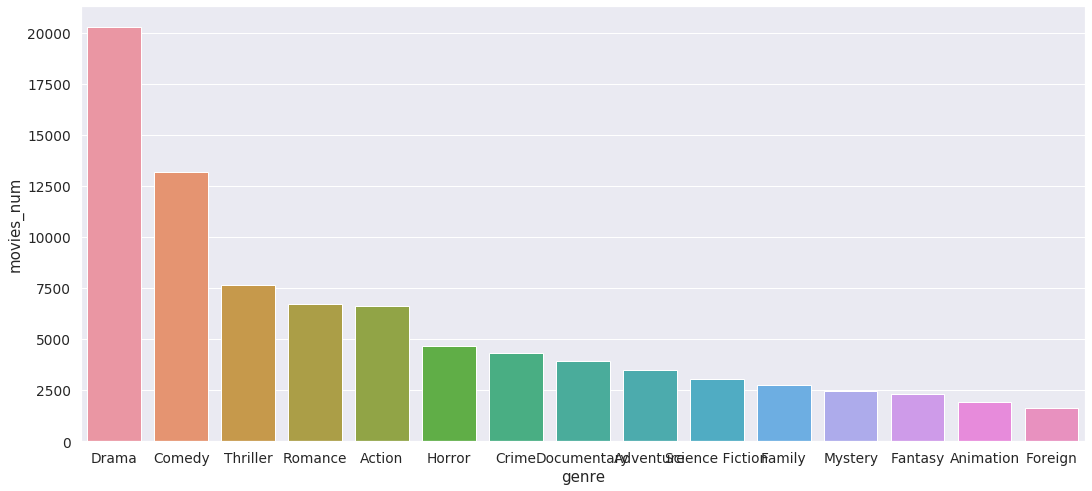
It can easily see that January is the most popular month with nearly 6000 movies has been released, followed by September and October. From February to August, this number are around 3000 to 3500.

* Most popular language apart from English:



After the analysis of the most popular destination for shooting movies, we had state that English is the most used language. If we consider the others, French and Italian are the most commonly occurring languages after English.

* Top genres:



Drama is the most common genre with almost half of the number of movies in the dataset followed by Comedy with 25%. Other major genres represented in the top 10 are Action, Horror, Crime, Mystery, Science Fiction, Animation and Fantasy.

1. **Methods**
2. **Loss function**

The function that computes the loss function (RMSE) for vectors of ratings and their corresponding predictors will be the following:

With: - N being the number of users over all these combinations

- u: user u

- i: item i

- : real rating of user

- : expected rating of user

RMSE is measure of model accuracy, it is a typical error we make when predicting a movie rating. If the result is larger than 1, it means that our typical error is larger than one star, which is a not good result. The lower RMSE the better.

**2. Method & Techniques**

Recommendation systems generally produce recommendations using either Content Based Filtering or Collaborative Filtering approaches and while Collaborative Filtering methods builds models based on users’s past decisions and decisions made by other similar uses, Content-based filtering methods use a series of discrete characteristics of an item in order to recommend additional items with similar properties. Among the two major categories, Collaborative Filtering is today the more popular approach in building Recommender Systems and we will focus on Collaborative filtering techniques in this project.

* **Collaborative filtering:**

There are three major processes in the recommendation system: object data collections and representations, similarity decisions and recommendation computations. Collaborative filtering aims at finding the relationships among the new individual and the existing data in order to further to determine the similarity and provide recommendations. How to define the similarity is an important issue. Similarity decisions are concluded differently by collaborative filtering techniques. For example, people that like and dislike movies in the same categories would be considered as the ones with similar behavior.

Collaborative filtering techniques collect and establish profiles, and determine the relationships among the data according to similarity models. Collaborative filtering solves several limitations in content-based filtering techniques which decides user preference only based on the individual profile.

* **Matrix Factorizations**

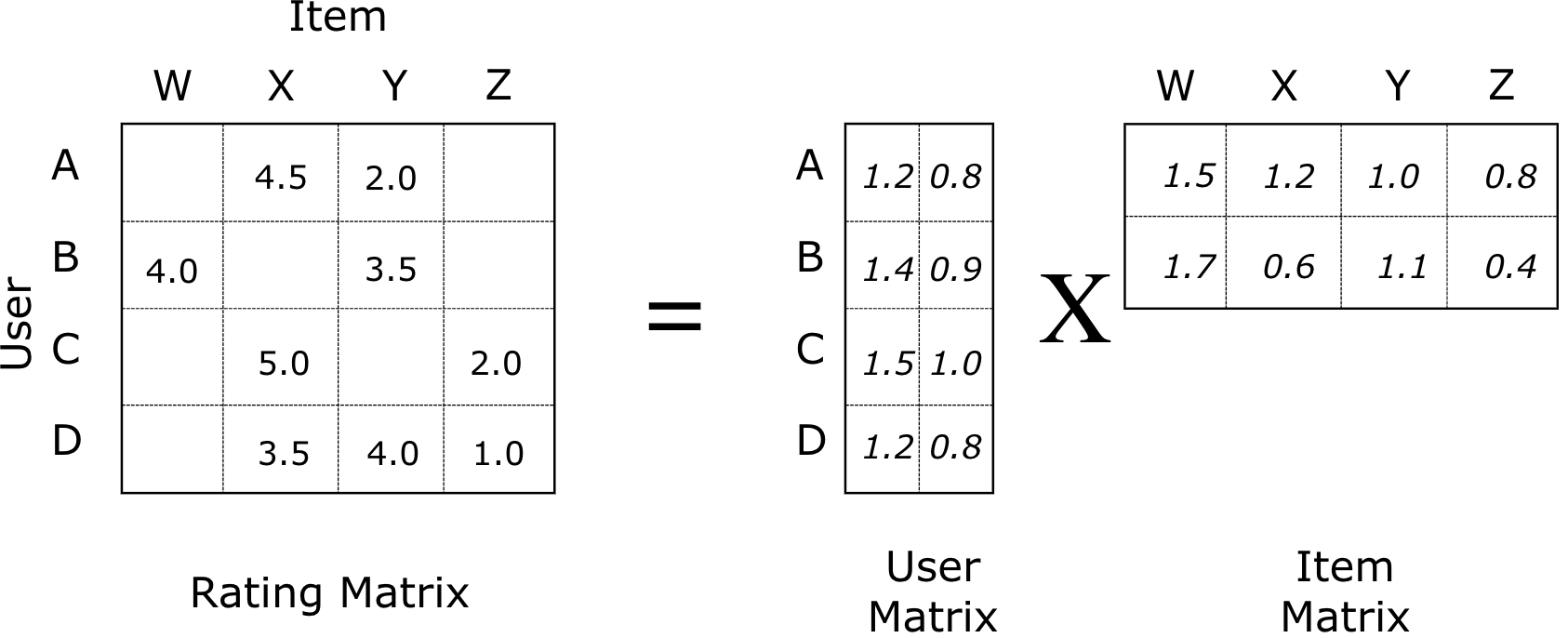
Some drawbacks that we might face using collaborative filtering is:

- Popularity Bias: Movies with the most interactions are likely to be recommended all the time.

- Item cold-start problem: When the new movie is added to dataset, it has either none or very little interactions while recommender rely on the movie’s interactions to make recommendations.

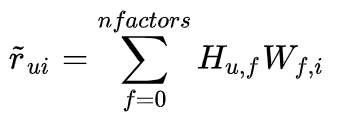
- Scalibility issue: refers to lack of the ability to scale to much larger sets of data when more and more users and movies added into our database.

Therefore, a state-of-the-art method comes out as a improved solution for Collaborative Filtering, which became widely known since Netflix Prize Challenge, Matrix Factorization. This method handle sparse data problem much better than Collaborative Filtering.



Matrix Factorization is a factorization of a matrix into a product of matrices. An User-item matrix is decomposed in to the product of 2 lower dimension matrices. One matrix can be seen as the user matrix where rows represent users and columns are latent factors. The other matrix is the item matrix where rows are latent factors and columns represent items.

In the sparse user-item interaction matrix, the predicted rating user u will give item i is computed as:



where H is user matrix, W is item matrix

Rating of item i given by user u can be expressed as a dot product of the user latent vector and the item latent vector.

- Matrix Factorization cost function:

The first term in this cost function is the Mean Square Error (MSE) distance measure between the original rating matrix R and its approximation . The second term is called a “regularization term” which is added to govern a generalized solution (to prevent overfitting to some local noisy effects on ratings).

* **Alternating Least Squares (ALS)**

- Look at the Matrix Factorization’s cost function, it appears that we aim at learning two types of variables, U and P.

- The idea behind ALS is that if we can fix the value of either U or P, the cost function is simply reduced to the problem of linear regression.

- So, here is how ALS do the job:

- In every iteration it first fixes P and solves for U, and following that it fixes U and solves for P.

- Alternating between the two steps guarantees reduction of the cost function, until convergence.

* **ALS with SparkML**

ALS in SparkML is a matrix factorization algorithm in parallel fashion. ALS is implemented in Apache Spark and built for large-scale collaborative filtering problems. ALS is doing a pretty good job at solving scalability and sparseness of the ratings data, and it’s simple and scales well to very large datasets.

Most important hyper-params in Alternating Least Square (ALS):

- maxIter: the maximum number of iterations to run (defaults to 10).

- rank: the number of latent factors in the model (defaults to 10).

- regParam: the regularization parameter in ALS (defaults to 1.0).

* **Elasticsearch**

Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows you to store, search and analyze big volumes of data quickly and in near real time. Elasticsearch is standing as a  NOSQL DB because: it easy to use, has a great community, compability with JSON and broad use cases.

Main backend components of elasticsearch:

* Node: a single server that is part of a cluster, stores out data and participates in the cluster’s indexing and search capabilities.
* Cluster: a collection of one or more nodes that together holds your entire data and provides federated indexing and search capabilities.
* Index: a collection of documents that have similar characteristics.
* Document: a basic unit of information that can be indexed.

**3. Main component of the code**

- **src/config/config.py:** contains all the configs of elasticsearch.

- **src/core/search.py:** core search.

- **src/helper:** module supports connection with elasticsearch.

- **src/model:** runs model and returns the result.

**- src/services:** initialization of routing/api of flask.

1. **Conclusion**

Recommender systems have become ubiquitous. People use them to find books, music, news, smart phones, vacation trips, and romantic partners. Nearly every product, service, or type of information has recommenders to help people select from among the myriad alternatives the few they would most appreciate. In our project, we have used Collaborative filtering method, the RMSE obtained was around 0.74. This approach has some advantages such as we don't need domain knowledge because the embeddings are automatically learned. To some extent, the system needs only the feedback matrix to train a matrix factorization model so that it will have a great starting point. However, it also has some drawbacks and one of the challenges is the cold-start problem. The prediction of the model for a given (user, item) pair is the dot product of the corresponding embeddings. So, if an item is not seen during training, the system can't create an embedding for it and can't query the model with this item.