# matrix_factorization_limitations

June 9, 2024

## 0.1 Investigating limitations of sklearn's non-negative matrix factorization library

The following code investigates the limitations of sklearn's non-negative matrix factorization library.

```
[ ]: # Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import mean_squared_error
from math import sqrt
```

Part 1

```
[ ]: # Load the data
users = pd.read_csv('movie_data/users.csv')
movies = pd.read_csv('movie_data/movies.csv')
train = pd.read_csv('movie_data/train.csv')
test = pd.read_csv('movie_data/test.csv')

# Create a user-movie matrix
train_data = pd.pivot_table(train, index='uID', columns='mID', values='rating')

# Fill missing values with zeros
train_data_filled = train_data.fillna(0)
```

```
# Perform SVD
svd = TruncatedSVD(n_components=50, random_state=42)
train_data_svd = svd.fit_transform(train_data_filled)

# Create a DataFrame from the SVD results
train_data_svd_df = pd.DataFrame(train_data_svd, index=train_data.index)

# Predict the ratings for the test data
def predict(row):
    try:
        return train_data_svd_df.loc[row['uID']].dot(svd.components_[:,␣
 ↪row['mID']-1])
    except IndexError:
        return np.nan

# Apply the predict function to each row in the test data
test['predicted_rating'] = test.apply(predict, axis=1)

# Calculate the global average rating
global_average_rating = train['rating'].mean()

# Fill NaN values in the predicted ratings with the global average rating
test['predicted_rating'].fillna(global_average_rating, inplace=True)

# Calculate the RMSE
rmse = sqrt(mean_squared_error(test['rating'], test['predicted_rating']))
print(f'RMSE: {rmse}')
```

```
RMSE: 3.4530633627495573

C:\Users\gtnov\AppData\Local\Temp\ipykernel_4792\3417046265.py:34:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test['predicted_rating'].fillna(global_average_rating, inplace=True)
```

RMSE stands for Root Mean Square Error. It is a commonly used metric to measure the error of a model in predicting quantitative data.

The RMSE gives a relatively high weight to large errors because the differences are squared before they are averaged. This means that RMSE is more useful when large errors are particularly undesirable.

In the context of this code, the RMSE is being used to measure the accuracy of the predicted movie ratings. A lower RMSE indicates a better fit of the model to the data.

Part 2

The result is a relatively high RMSE. This did not work well compared to simple baseline or similarity-based methods. This could be due to several reasons: * Cold Start Problem. The SVD method might not work well when there are new users or new movies that were not in the training data. This is known as the cold start problem in recommendation systems. The SVD method relies on having some existing data about users and movies to make predictions. If a user or a movie is not in the training data, the method might not be able to make accurate predictions. * Number of Components. The number of components used in the SVD (50 in this case) might not be optimal. Using too few components might result in loss of important information, while using too many might include noise. You could try experimenting with different numbers of components to see if it improves the RMSE. * Global Average Imputation. Filling missing predictions with the global average rating is a simple imputation method, but it might not always give the best results. It assumes that the average rating is a good substitute for missing ratings, which might not be the case. For example, some users might consistently give higher or lower ratings than the average.