BBC News Classification Part 2: Limitations of sklearn's Non-Negative Matrix Factorization (NMF) Library

Non-negative matrix factorization (NMF) is a popular technique for dimensionality reduction and topic modeling. However, the `sklearn` implementation of NMF has some limitations:

1. **Scalability**: `sklearn` 's NMF may not scale well with very large datasets due to its computational complexity.
2. **Sparsity**: `sklearn` 's NMF does not handle sparse matrices efficiently, which is a common characteristic of recommender system data.
3. **Initialization**: `sklearn` 's NMF relies on initialization methods like 'random' or 'nndsvd', which can impact the convergence and final solution quality.
4. **Missing Values**: `sklearn` 's NMF cannot handle missing values directly, which is crucial for recommendation systems where the matrix is typically incomplete.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.decomposition import NMF
from sklearn.metrics import mean_squared_error
```

In [2]:
```
# Load the movie ratings data (as in the HW3-recommender-system)
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

In [3]:
```
train.head()
```

Out[3]:

	uID	mID	rating
0	744	1210	5
1	3040	1584	4
2	1451	1293	5
3	5455	3176	2
4	2507	3074	5

In [4]:
```
test.head()
```

Out[4]:

	uID	mID	rating
0	2233	440	4
1	4274	587	5
2	2498	454	3
3	2868	2336	5
4	1636	2686	5

In [5]:
```
# Load the data (example data frame structure)
# Assume df has columns: userId, movieId, rating
df = pd.read_csv('train.csv')
```

In [6]:
```
# Split the data into train and test sets
train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)

# Create the user-item matrix for training
train_matrix = train_data.pivot(index='uID', columns='mID', values='rating').fil

# Prepare test data
test_matrix = test_data.pivot(index='uID', columns='mID', values='rating').filln
```

In [7]:
```
# Apply NMF
n_factors = 15  # Number of latent factors
model = NMF(n_components=n_factors, init='random', random_state=42, max_iter=500
W = model.fit_transform(train_matrix)
H = model.components_
```

In [8]:
```
# Predict ratings
predicted_ratings = np.dot(W, H)

predicted_ratings
```

Out[8]:
```
array([[1.71325973e+00, 5.45407300e-01, 1.89827749e-02, ...,
        9.01326720e-03, 3.75894878e-03, 8.17556609e-02],
       [1.03758588e+00, 3.74016438e-01, 1.23073281e-01, ...,
        2.36025102e-02, 2.57335592e-03, 1.05551191e-01],
       [1.02032732e+00, 1.35829686e-01, 2.99129466e-02, ...,
        1.88961079e-03, 8.15878670e-04, 1.57826943e-02],
       ...,
       [4.27294176e-01, 2.40787832e-02, 5.66683049e-04, ...,
        9.58219575e-04, 4.72233003e-04, 4.66842776e-03],
       [9.22806894e-01, 2.66902181e-01, 7.13577392e-02, ...,
        2.92729513e-02, 9.72259203e-03, 1.80512531e-02],
       [8.31913262e-01, 6.48540433e-02, 3.65613280e-04, ...,
        9.10483514e-02, 5.67328288e-02, 4.30515657e-01]])
```

In [9]:
```
# Align test data indices with the train matrix
user_index_map = {user_id: index for index, user_id in enumerate(train_matrix.in
movie_index_map = {movie_id: index for index, movie_id in enumerate(train_matrix
```

In [10]:
```python
# Get actual and predicted values
actual_ratings = []
predicted_ratings_list = []

for _, row in test_data.iterrows():
    user_id = row['uID']
    movie_id = row['mID']
    actual_rating = row['rating']

    if user_id in user_index_map and movie_id in movie_index_map:
        user_index = user_index_map[user_id]
        movie_index = movie_index_map[movie_id]

        predicted_rating = predicted_ratings[user_index, movie_index]

        actual_ratings.append(actual_rating)
        predicted_ratings_list.append(predicted_rating)
```

In [11]:
```python
# Calculate RMSE
actual_values = np.array(actual_ratings)
predicted_values = np.array(predicted_ratings_list)

rmse = np.sqrt(mean_squared_error(actual_values, predicted_values))
print(f'RMSE: {rmse:.4f}')
```

```
RMSE: 3.0435
```

Discussion of Results

The obtained RMSE of 3.0435 for the NMF model is significantly higher than the RMSEs of various baseline and similarity-based methods. Here's a detailed discussion on why `sklearn` 's NMF did not perform as well and potential ways to improve the model.

Why `sklearn` 's NMF Did Not Work Well

1. **Scalability Issues**: The `sklearn` NMF implementation might not scale well with larger datasets or higher dimensions. MovieLens datasets can be quite large, and NMF may struggle to capture the complex interactions between users and items efficiently.
2. **Sparsity Handling**: Recommender system data is usually sparse. While NMF can handle sparse data, the `sklearn` implementation may not be optimized for very sparse matrices. The model may not be able to fill in the missing values effectively.
3. **Initialization Sensitivity**: NMF relies on initialization methods, and poor initialization can lead to suboptimal factorization. The random initialization used may not have been suitable for this data.
4. **Overfitting**: NMF can overfit the training data, especially if the number of components is not chosen carefully. Overfitting reduces the model's ability to generalize to unseen data, leading to poor performance on the test set.
5. **Hyperparameter Selection**: The choice of hyperparameters (e.g., the number of latent factors) significantly impacts the performance of NMF. In this case, 15 latent factors might not have been optimal.

Comparison with Baseline and Similarity-Based Methods

- **Baseline Methods**: Simple baseline methods like predicting the global mean ($Y_p = 3$) or user mean ($Y_p = \mu_u$) are straightforward and often provide reasonable benchmarks. These methods don't require complex computations and are less prone to overfitting, hence their lower RMSE.
- **Content-Based and Similarity-Based Methods**: Methods like item-item collaborative filtering or similarity-based approaches leverage user and item similarities, which can effectively capture user preferences and item characteristics. These methods tend to handle sparsity better and can be more interpretable.

Ways to Improve the NMF Model

1. **Better Initialization**:
    - Use more advanced initialization techniques like `nndsvd` which can provide a better starting point for factorization.
1. **Regularization**:

1. **Regularization**:

    - Adding regularization can help prevent overfitting. Regularization terms in the objective function penalize large factor values, leading to more generalizable models.

1. **Parameter Tuning**:

    - Conduct a grid search or use cross-validation to find the optimal number of latent factors and regularization parameters.

1. **Hybrid Methods**:

    - Combine NMF with other models. For instance, use NMF to initialize a collaborative filtering model or blend NMF predictions with those from baseline or similarity-based methods.

1. **Alternative Matrix Factorization Techniques**:

    - Use other matrix factorization techniques like SVD (Singular Value Decomposition) which might perform better in certain cases.

Conclusion

The high RMSE for the `sklearn` NMF model compared to baseline and similarity-based methods indicates potential limitations in handling sparsity, initialization sensitivity, and scalability issues. By improving initialization, adding regularization, tuning hyperparameters, and exploring hybrid or alternative methods, we can enhance the performance of matrix factorization models for recommendation systems.