Professor‘s Project 2 Bias and rating

June 22, 2025

[1]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error

class GlobalBaselineRecommender:
    """
    Global Baseline Estimate Recommender System

    Formula: b_ui =  + b_u + b_i
    Where:
    -  (mu): global mean rating
    - b_u: user bias (user u's deviation from global mean)
    - b_i: item bias (item i's deviation from global mean)
    """

    def __init__(self, reg_user=0.1, reg_item=0.1):
        """
        Initialize the recommender with regularization parameters

        Args:
            reg_user: regularization parameter for user biases
            reg_item: regularization parameter for item biases
        """
        self.reg_user = reg_user
        self.reg_item = reg_item
        self.global_mean = 0
        self.user_biases = {}
        self.item_biases = {}
        self.trained = False

    def fit(self, ratings_matrix):
        """
        Train the model on the ratings matrix

        Args:
            ratings_matrix: pandas DataFrame with columns ['user_id',␣
↪'item_id', 'rating']
```

```
        """
        # Calculate global mean
        self.global_mean = ratings_matrix['rating'].mean()

        # Initialize biases
        users = ratings_matrix['user_id'].unique()
        items = ratings_matrix['item_id'].unique()

        # Calculate user biases (regularized)
        for user in users:
            user_ratings = ratings_matrix[ratings_matrix['user_id'] ==␣
↪user]['rating']
            n_ratings = len(user_ratings)
            user_mean = user_ratings.mean()

            # Regularized user bias
            self.user_biases[user] = (user_mean - self.global_mean) * n_ratings␣
↪/ (n_ratings + self.reg_user)

        # Calculate item biases (regularized)
        for item in items:
            item_ratings = ratings_matrix[ratings_matrix['item_id'] == item]

            adjusted_ratings = []
            for _, row in item_ratings.iterrows():
                user = row['user_id']
                rating = row['rating']
                bu = self.user_biases.get(user, 0)
                adjusted_ratings.append(rating - self.global_mean - bu)

            n_ratings = len(adjusted_ratings)
            if n_ratings > 0:
                self.item_biases[item] = sum(adjusted_ratings) / (n_ratings +␣
↪self.reg_item)

        self.trained = True

    def predict(self, user_id, item_id):
        """
        Predict rating for a user-item pair

        Args:
            user_id: ID of the user
            item_id: ID of the item

        Returns:
            Predicted rating
```

```
        """
        if not self.trained:
            raise ValueError("Model must be trained before making predictions")

        # Get biases (default to 0 for unseen users/items)
        user_bias = self.user_biases.get(user_id, 0)
        item_bias = self.item_biases.get(item_id, 0)

        # Apply the global baseline formula
        prediction = self.global_mean + user_bias + item_bias

        # Clamp to valid rating range (assuming 1-5 scale)
        return max(1, min(5, prediction))

    def predict_batch(self, user_item_pairs):
        """
        Predict ratings for multiple user-item pairs

        Args:
            user_item_pairs: list of tuples [(user_id, item_id), ...]

        Returns:
            List of predicted ratings
        """
        return [self.predict(user, item) for user, item in user_item_pairs]

    def get_top_n_recommendations(self, user_id, all_items, n=10,␣
↪exclude_rated=None):
        """
        Get top N item recommendations for a user

        Args:
            user_id: ID of the user
            all_items: list of all available item IDs
            n: number of recommendations to return
            exclude_rated: set of item IDs already rated by the user

        Returns:
            List of (item_id, predicted_rating) tuples, sorted by rating
        """
        if exclude_rated is None:
            exclude_rated = set()

        # Predict ratings for all unrated items
        recommendations = []
        for item_id in all_items:
            if item_id not in exclude_rated:
```

```
                predicted_rating = self.predict(user_id, item_id)
                recommendations.append((item_id, predicted_rating))

        # Sort by predicted rating (descending) and return top N
        recommendations.sort(key=lambda x: x[1], reverse=True)
        return recommendations[:n]

def create_sample_data():
    """Create sample movie ratings data"""
    np.random.seed(42)

    # Create sample data
    users = [f'User_{i}' for i in range(1, 21)]  # 20 users
    movies = [f'Movie_{i}' for i in range(1, 16)]  # 15 movies

    data = []
    for user in users:
        # Each user rates 5-12 movies randomly
        n_ratings = np.random.randint(5, 13)
        rated_movies = np.random.choice(movies, n_ratings, replace=False)

        # Simulate user preferences (some users rate higher/lower on average)
        user_bias = np.random.normal(0, 0.5)

        for movie in rated_movies:
            # Simulate movie quality (some movies are generally better)
            movie_idx = int(movie.split('_')[1]) - 1
            movie_bias = np.sin(movie_idx / 5) * 0.8  # Some pattern in movie␣
 ↪quality

            # Generate rating with noise
            base_rating = 3.0 + user_bias + movie_bias + np.random.normal(0, 0.
 ↪3)
            rating = max(1, min(5, round(base_rating)))

            data.append({
                'user_id': user,
                'item_id': movie,
                'rating': rating
            })

    return pd.DataFrame(data)

def evaluate_model(model, test_data):
    """Evaluate model performance"""
    actual_ratings = test_data['rating'].values
    predicted_ratings = []
```

```
    for _, row in test_data.iterrows():
        pred = model.predict(row['user_id'], row['item_id'])
        predicted_ratings.append(pred)

    rmse = np.sqrt(mean_squared_error(actual_ratings, predicted_ratings))
    mae = np.mean(np.abs(np.array(actual_ratings) - np.
 ↪array(predicted_ratings)))

    return rmse, mae

def main():
    # Create sample dataset
    print("\n***********")
    print("Creating sample movie ratings dataset...")
    df = create_sample_data()
    print(f"Dataset shape: {df.shape}")
    print(f"Number of users: {df['user_id'].nunique()}")
    print(f"Number of movies: {df['item_id'].nunique()}")
    print(f"Rating distribution:\n{df['rating'].value_counts().sort_index()}")
    print("\nSample of the dataset:")
    print(df.head())

    # Split data into train/test (80/20)
    print("\n***********")
    print("Creating train and test dataset...")
    from sklearn.model_selection import train_test_split
    train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)
    print(f"\nTrain size: {len(train_data)}, Test size: {len(test_data)}")

    # Initialize and train the model
    print("\n***********")
    print("\nTraining Global Baseline Recommender...")
    recommender = GlobalBaselineRecommender(reg_user=0.1, reg_item=0.1)
    recommender.fit(train_data)

    # Display model statistics
    print("\n***********")
    print(f"\nModel Statistics:")
    print(f"Global mean rating: {recommender.global_mean:.3f}")
    print(f"Number of user biases: {len(recommender.user_biases)}")
    print(f"Number of item biases: {len(recommender.item_biases)}")

    # Show some user and item biases
    print("\n***********")
    print(f"\nSample User Biases:")
    for i, (user, bias) in enumerate(list(recommender.user_biases.items())[:5]):
```

```
        print(f"  {user}: {bias:+.3f}")

    print(f"\nSample Item Biases:")
    for i, (item, bias) in enumerate(list(recommender.item_biases.items())[:5]):
        print(f"  {item}: {bias:+.3f}")

    # Evaluate model
    print("\n***********")
    print(f"\nEvaluating model on test data...")
    rmse, mae = evaluate_model(recommender, test_data)
    print(f"RMSE: {rmse:.3f}")
    print(f"MAE: {mae:.3f}")

    print("\n***********")
    print(f"\nInference")

    # Example predictions
    print("\n***********")
    print(f"\nExample Predictions")
    sample_user = 'User_1'
    all_movies = df['item_id'].unique()
    user_rated_movies = set(train_data[train_data['user_id'] ==␣
↪sample_user]['item_id'])

    print("\n***********")
    # Show actual vs predicted for some test cases
    print(f"\nActual vs Predicted (sample from test set):")
    sample_test = test_data.head(10)
    for _, row in sample_test.iterrows():
        actual = row['rating']
        predicted = recommender.predict(row['user_id'], row['item_id'])
        print(f"  {row['user_id']} -> {row['item_id']}: Actual={actual},␣
↪Predicted={predicted:.2f}")

    # Get recommendations
    print("\n***********")
    recommendations = recommender.get_top_n_recommendations(
        sample_user, all_movies, n=5, exclude_rated=user_rated_movies
    )
    print(f"\nTop 5 recommendations for {sample_user}:")
    for movie, predicted_rating in recommendations:
        print(f"  {movie}: {predicted_rating:.2f}")


if __name__ == "__main__":
    main()
```

```
**********
Creating sample movie ratings dataset…
Dataset shape: (172, 3)
Number of users: 20
Number of movies: 15
Rating distribution:
rating
2     3
3    77
4    86
5     6
Name: count, dtype: int64

Sample of the dataset:
  user_id   item_id  rating
0  User_1   Movie_1       3
1  User_1   Movie_2       3
2  User_1   Movie_6       4
3  User_1  Movie_15       3
4  User_1  Movie_14       3

**********
Creating train and test dataset…

Train size: 137, Test size: 35

**********

Training Global Baseline Recommender…

**********

Model Statistics:
Global mean rating: 3.569
Number of user biases: 20
Number of item biases: 15

**********

Sample User Biases:
  User_12: +0.425
  User_6: +0.143
  User_5: +0.129
  User_10: +0.030
  User_7: +0.284

Sample Item Biases:
```

```
  Movie_12: +0.169
  Movie_13: -0.091
  Movie_10: +0.239
  Movie_2: -0.264
  Movie_5: +0.047

**********

Evaluating model on test data…
RMSE: 0.491
MAE: 0.410

**********

Inference

**********

Example Predictions

**********

Actual vs Predicted (sample from test set):
  User_9 -> Movie_2: Actual=2, Predicted=2.51
  User_16 -> Movie_13: Actual=3, Predicted=3.90
  User_18 -> Movie_4: Actual=3, Predicted=3.72
  User_6 -> Movie_11: Actual=4, Predicted=3.68
  User_18 -> Movie_6: Actual=4, Predicted=3.82
  User_19 -> Movie_13: Actual=3, Predicted=3.25
  User_10 -> Movie_15: Actual=3, Predicted=3.26
  User_17 -> Movie_9: Actual=3, Predicted=3.87
  User_13 -> Movie_3: Actual=3, Predicted=2.80
  User_3 -> Movie_9: Actual=3, Predicted=3.55

**********

Top 5 recommendations for User_1:
  Movie_7: 3.69
  Movie_8: 3.67
  Movie_4: 3.55
  Movie_11: 3.46
  Movie_13: 3.41
```

[ ]:
```python
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
```

```python
class GlobalBaselineRecommender:
    """
    Global Baseline Estimate Recommender System

    Formula: b_ui =   + b_u + b_i
    Where:
    -   (mu): global mean rating
    - b_u: user bias (user u's deviation from global mean)
    - b_i: item bias (item i's deviation from global mean)
    """

    def __init__(self, reg_user=0.1, reg_item=0.1):
        """
        Initialize the recommender with regularization parameters

        Args:
            reg_user: regularization parameter for user biases
            reg_item: regularization parameter for item biases
        """
        self.reg_user = reg_user
        self.reg_item = reg_item
        self.global_mean = 0
        self.user_biases = {}
        self.item_biases = {}
        self.trained = False

    def fit(self, ratings_matrix):
        """
        Train the model on the ratings matrix

        Args:
            ratings_matrix: pandas DataFrame with columns ['user_id',␣
↪'item_id', 'rating']
        """
        # Calculate global mean
        self.global_mean = ratings_matrix['rating'].mean()

        # Initialize biases
        users = ratings_matrix['user_id'].unique()
        items = ratings_matrix['item_id'].unique()

        # Calculate user biases (regularized)
        for user in users:
            user_ratings = ratings_matrix[ratings_matrix['user_id'] ==␣
↪user]['rating']
            n_ratings = len(user_ratings)
            user_mean = user_ratings.mean()
```

```
            # Regularized user bias
            self.user_biases[user] = (user_mean - self.global_mean) * n_ratings␣
↪/ (n_ratings + self.reg_user)

        # Calculate item biases (regularized)
        for item in items:
            item_ratings = ratings_matrix[ratings_matrix['item_id'] == item]

            adjusted_ratings = []
            for _, row in item_ratings.iterrows():
                user = row['user_id']
                rating = row['rating']
                bu = self.user_biases.get(user, 0)
                adjusted_ratings.append(rating - self.global_mean - bu)

            n_ratings = len(adjusted_ratings)
            if n_ratings > 0:
                self.item_biases[item] = sum(adjusted_ratings) / (n_ratings +␣
↪self.reg_item)

        self.trained = True

    def predict(self, user_id, item_id):
        """
        Predict rating for a user-item pair

        Args:
            user_id: ID of the user
            item_id: ID of the item

        Returns:
            Predicted rating
        """
        if not self.trained:
            raise ValueError("Model must be trained before making predictions")

        # Get biases (default to 0 for unseen users/items)
        user_bias = self.user_biases.get(user_id, 0)
        item_bias = self.item_biases.get(item_id, 0)

        # Apply the global baseline formula
        prediction = self.global_mean + user_bias + item_bias

        # Clamp to valid rating range (assuming 1-5 scale)
        return max(1, min(5, prediction))
```

```
    def predict_batch(self, user_item_pairs):
        """
        Predict ratings for multiple user-item pairs

        Args:
            user_item_pairs: list of tuples [(user_id, item_id), ...]

        Returns:
            List of predicted ratings
        """
        return [self.predict(user, item) for user, item in user_item_pairs]

    def get_top_n_recommendations(self, user_id, all_items, n=10,␣
↪exclude_rated=None):
        """
        Get top N item recommendations for a user

        Args:
            user_id: ID of the user
            all_items: list of all available item IDs
            n: number of recommendations to return
            exclude_rated: set of item IDs already rated by the user

        Returns:
            List of (item_id, predicted_rating) tuples, sorted by rating
        """
        if exclude_rated is None:
            exclude_rated = set()

        # Predict ratings for all unrated items
        recommendations = []
        for item_id in all_items:
            if item_id not in exclude_rated:
                predicted_rating = self.predict(user_id, item_id)
                recommendations.append((item_id, predicted_rating))

        # Sort by predicted rating (descending) and return top N
        recommendations.sort(key=lambda x: x[1], reverse=True)
        return recommendations[:n]

def create_sample_data():
    """Create sample movie ratings data"""
    np.random.seed(42)

    # Create sample data
    users = [f'User_{i}' for i in range(1, 21)]  # 20 users
    movies = [f'Movie_{i}' for i in range(1, 16)]  # 15 movies
```

```
    data = []
    for user in users:
        # Each user rates 5-12 movies randomly
        n_ratings = np.random.randint(5, 13)
        rated_movies = np.random.choice(movies, n_ratings, replace=False)

        # Simulate user preferences (some users rate higher/lower on average)
        user_bias = np.random.normal(0, 0.5)

        for movie in rated_movies:
            # Simulate movie quality (some movies are generally better)
            movie_idx = int(movie.split('_')[1]) - 1
            movie_bias = np.sin(movie_idx / 5) * 0.8  # Some pattern in movie␣
 ↪quality

            # Generate rating with noise
            base_rating = 3.0 + user_bias + movie_bias + np.random.normal(0, 0.
 ↪3)
            rating = max(1, min(5, round(base_rating)))

            data.append({
                'user_id': user,
                'item_id': movie,
                'rating': rating
            })

    return pd.DataFrame(data)

def evaluate_model(model, test_data):
    """Evaluate model performance"""
    actual_ratings = test_data['rating'].values
    predicted_ratings = []

    for _, row in test_data.iterrows():
        pred = model.predict(row['user_id'], row['item_id'])
        predicted_ratings.append(pred)

    rmse = np.sqrt(mean_squared_error(actual_ratings, predicted_ratings))
    mae = np.mean(np.abs(np.array(actual_ratings) - np.
 ↪array(predicted_ratings)))

    return rmse, mae

def main():
    # Create sample dataset
    print("\n***********")
```

```
    print("Creating sample movie ratings dataset...")
    df = create_sample_data()
    print(f"Dataset shape: {df.shape}")
    print(f"Number of users: {df['user_id'].nunique()}")
    print(f"Number of movies: {df['item_id'].nunique()}")
    print(f"Rating distribution:\n{df['rating'].value_counts().sort_index()}")
    print("\nSample of the dataset:")
    print(df.head())

    # Split data into train/test (80/20)
    print("\n**********")
    print("Creating train and test dataset...")
    from sklearn.model_selection import train_test_split
    train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)
    print(f"\nTrain size: {len(train_data)}, Test size: {len(test_data)}")

    # Initialize and train the model
    print("\n**********")
    print("\nTraining Global Baseline Recommender...")
    recommender = GlobalBaselineRecommender(reg_user=0.1, reg_item=0.1)
    recommender.fit(train_data)

    # Display model statistics
    print("\n**********")
    print(f"\nModel Statistics:")
    print(f"Global mean rating: {recommender.global_mean:.3f}")
    print(f"Number of user biases: {len(recommender.user_biases)}")
    print(f"Number of item biases: {len(recommender.item_biases)}")

    # Show some user and item biases
    print("\n**********")
    print(f"\nSample User Biases:")
    for i, (user, bias) in enumerate(list(recommender.user_biases.items())[:5]):
        print(f"  {user}: {bias:+.3f}")

    print(f"\nSample Item Biases:")
    for i, (item, bias) in enumerate(list(recommender.item_biases.items())[:5]):
        print(f"  {item}: {bias:+.3f}")

    # Evaluate model
    print("\n**********")
    print(f"\nEvaluating model on test data...")
    rmse, mae = evaluate_model(recommender, test_data)
    print(f"RMSE: {rmse:.3f}")
    print(f"MAE: {mae:.3f}")

    print("\n**********")
```

```
    print(f"\nInference")

    # Example predictions
    print("\n***********")
    print(f"\nExample Predictions")
    sample_user = 'User_1'
    all_movies = df['item_id'].unique()
    user_rated_movies = set(train_data[train_data['user_id'] ==␣
↪sample_user]['item_id'])

    print("\n***********")
    # Show actual vs predicted for some test cases
    print(f"\nActual vs Predicted (sample from test set):")
    sample_test = test_data.head(10)
    for _, row in sample_test.iterrows():
        actual = row['rating']
        predicted = recommender.predict(row['user_id'], row['item_id'])
        print(f"  {row['user_id']} -> {row['item_id']}: Actual={actual},␣
↪Predicted={predicted:.2f}")

    # Get recommendations
    print("\n***********")
    recommendations = recommender.get_top_n_recommendations(
        sample_user, all_movies, n=5, exclude_rated=user_rated_movies
    )
    print(f"\nTop 5 recommendations for {sample_user}:")
    for movie, predicted_rating in recommendations:
        print(f"  {movie}: {predicted_rating:.2f}")


if __name__ == "__main__":
    main()
```