基于隐式反馈的 Top-10 推荐列表预测

吴建军
学号 2024140899
roshibang@bupt.edu.cn

摘要： 本实验通过使用提供的训练集构建了推荐模型，并使用测试集中的数据对推荐系统进行了 Top-10 推荐列表的预测。完成了数据处理、模型构建、训练与验证的过程，并生成了每个用户的推荐结果。通过实验验证了推荐算法的效果和可行性，并对推荐系统的性能进行了评估。

关键字： 推荐系统；Top-10 推荐；隐式反馈

一　实验概述

training.txt 是用户-物品-隐式反馈的交互对，一共有四万多条交互信息。在代码中将其拆分为训练集和验证集。 test.txt 是真实的测试集，只有用户 ID，我们最终需要在该测试集上进行 Top-N 推荐任务。 result.txt 是算法得到的结果，即对 test.txt 中的用户一一进行 Top-10 推荐。

1.1　实验设计

本实验包括以下几个步骤，每个部分都有明确的功能，最终实现了一个基于矩阵分解的推荐系统。

1. 数据加载与预处理。这部分代码加载训练数据，进行负采样来生成负样本，并将正负样本结合后打乱顺序。
2. 数据集划分和稀疏度计算。计算了数据的稀疏度，并划分了训练集和验证集，计算了平均评分。
3. 初始化矩阵参数。初始化了用户和电影的特征矩阵及其增量矩阵。
4. 训练模型。实现了模型的训练过程，包括特征矩阵的更新、训练误差和验证误差的计算，并绘制了误差曲线。
5. Precision@10 和 Recall@10 计算。这部分代码计算了 Precision@10 和 Recall@10 两个评价指标。
6. 推荐结果生成与保存。这部分代码对测试集用户生成推荐结果，并将结果保存为 txt 文件。

1.2　代码清单

本项目的代码目录如下：

```
project/
│
├── data/
│   └── training.txt
```

```
│   └── test.txt
│   └── result.txt
│
├── images/
│   └── num_feats_6.png
│   └── num_feats_8.png
│   └── num_feats_10.png
│   └── baseline.png
│
├── baseline.py
└── main.py
```

data 目录包含输入、输出数据文件。 images 目录包含本报告中所有的图。 baseline.py 文件为基准模型的实现代码，用于与主模型进行比较。 main.py 文件为主程序文件，包含项目的核心逻辑。

1.3 实验环境

本实验使用的环境如下：

- Ubuntu 22.04 LTS
- Python 3.11.9

二 实现细节

2.1 数据加载与预处理

加载训练数据并设定列名为"user_id", “item_id”, “click"，方便后续处理。将所有点击行为设为 5，简化了评分机制，使模型更关注于交互关系。通过负采样生成负样本（即用户未点击的项目），并将正负样本结合，打乱顺序以增加数据的多样性。

这样做确保数据格式正确，同时通过引入负样本，提升模型的泛化能力，使其在预测未见数据时表现更好。

```python
# 读取训练数据
train_data = pd.read_csv("data/training.txt", sep=" ", header=None,
names=["user_id", "item_id", "click"])
train_data["click"] = 5
rating = train_data["click"]
movie = train_data["item_id"]
user = train_data["user_id"]
data_num = len(rating)
movie_num = movie.nunique()  # 电影的数量
user_num = user.nunique()  # 用户的数量

# 负采样
negative_sample = np.random.choice(user_num * movie_num, data_num, replace=True)
record_user = []
record_movie = []
```

```
record_rating = [1] * data_num

for i in negative_sample:
    movie_i = i // user_num
    user_i = i % user_num + 1
    record_user.append(user_i)
    record_movie.append(movie_i)

negative_data = pd.DataFrame({"user_id": record_user, "item_id": record_movie,
"click": record_rating})

# 数据结合
data = pd.concat([train_data, negative_data])
rating = data["click"]
movie = data["item_id"]
user = data["user_id"]
data_num = len(rating)
movie_num = movie.nunique()  # 使用.nunique()确保电影数量的准确性
user_num = user.nunique()  # 使用.nunique()确保用户数量的准确性
data = data.sample(frac=1).reset_index(drop=True)  # 打乱数据集
```

2.2 数据集划分和稀疏度计算

计算数据的稀疏度，通过用户和物品的数量以及评分数据的数量来确定数据的稀疏程度。这一步有助于理解数据的特点，选择合适的模型和优化方法。随后将数据划分为训练集和验证集，训练集用于训练模型，验证集用于评估模型在未见过的数据上的表现。

```
# 稀疏度
sparsity = data_num / (movie_num * user_num)
print(f"Sparsity: {sparsity:.6f}")

# 划分训练集和验证集
train_num = 80000
train_vec = data.iloc[:train_num]
probe_vec = data.iloc[train_num:84000]
mean_rating = train_vec["click"].mean()
pairs_tr = len(train_vec)
pairs_pr = len(probe_vec)
numbatches = 8
num_m = movie_num
num_p = user_num
```

2.3 初始化矩阵参数

初始化用户特征矩阵 w1_P1 和电影特征矩阵 w1_M1，以及它们的增量矩阵 w1_P1_inc 和 w1_M1_inc，这些矩阵用于动量优化。这些特征矩阵以随机小数初始化，确保模型在开始训练时有一个合理的初始状态。增量矩阵用于记录前几次更新的趋势，动量优化有助于加速训练过程，提高模型的收敛速度和稳定性，减少训练过程中的震荡。

```
# 初始化矩阵参数
w1_M1 = 0.1 * np.random.rand(num_m, num_feat)  # 电影特征矩阵
w1_P1 = 0.1 * np.random.rand(num_p, num_feat)  # 用户特征矩阵
w1_M1_inc = np.zeros((num_m, num_feat))  # 电影特征矩阵增量
w1_P1_inc = np.zeros((num_p, num_feat))  # 用户特征矩阵增量
```

2.4 训练模型

通过多次 epoch 和 batch 迭代更新特征矩阵。在每个 batch 中，提取当前批次的用户和物品 ID，计算预测评分、误差平方和、梯度，并更新特征矩阵。分批次训练使得模型可以处理大规模数据，避免内存溢出。梯度更新和动量优化可以加速模型训练，提高模型的准确性和稳定性。每个 epoch 结束时计算训练误差和验证误差，有助于监控模型的训练过程，及时调整策略。

```
# 训练模型
for epoch in range(maxepoch):
    for batch in range(numbatches):
        # 计算当前批次的起始和结束索引
        start = batch * N
        end = start + N
        if end > pairs_tr:  # 防止越界
            end = pairs_tr

        # 提取当前批次的用户和物品 ID，并将它们的索引减 1 以匹配矩阵
        aa_p = train_vec.iloc[start:end]["user_id"].values - 1
        aa_m = train_vec.iloc[start:end]["item_id"].values - 1
        rating = train_vec.iloc[start:end]["click"].values.astype(float)  # 确保
rating 为浮点数
        rating -= mean_rating  # 减去平均评分

        # 计算预测评分
        pred_out = np.sum(w1_M1[aa_m] * w1_P1[aa_p], axis=1)
        f = np.sum((pred_out - rating) ** 2)  # 计算误差平方和

        # 计算梯度
        IO = 2 * (pred_out - rating)
        IO = np.tile(IO[:, None], num_feat)  # 将 IO 扩展到特征数量的维度

        Ix_m = IO * w1_P1[aa_p]  # 对电影特征的梯度
        Ix_p = IO * w1_M1[aa_m]  # 对用户特征的梯度

        # 初始化梯度增量矩阵
        dw1_M1 = np.zeros((movie_num, num_feat))
        dw1_P1 = np.zeros((user_num, num_feat))

        # 累加每个样本的梯度
        for ii in range(N):
            dw1_M1[aa_m[ii]] += Ix_m[ii]
            dw1_P1[aa_p[ii]] += Ix_p[ii]
```

```
        # 更新特征矩阵
        w1_M1_inc = momentum * w1_M1_inc + epsilon * dw1_M1 / N
        w1_M1 -= w1_M1_inc
        w1_P1_inc = momentum * w1_P1_inc + epsilon * dw1_P1 / N
        w1_P1 -= w1_P1_inc

    # 计算训练误差
    pred_out = np.sum(w1_M1[aa_m] * w1_P1[aa_p], axis=1)
    f_s = np.sum((pred_out - rating) ** 2)
    err_train[epoch] = np.sqrt(f_s / N)  # 计算训练集上的 RMSE

    # 在验证集上进行预测并计算误差
    aa_p = probe_vec["user_id"].values - 1
    aa_m = probe_vec["item_id"].values - 1
    rating = probe_vec["click"].values

    pred_out = np.sum(w1_M1[aa_m] * w1_P1[aa_p], axis=1) + mean_rating
    pred_out = np.clip(pred_out, 1, 5)  # 将预测评分限制在 1 到 5 之间

    err_valid[epoch] = np.sqrt(np.sum((pred_out - rating) ** 2) / pairs_pr)  # 计
算验证集上的 RMSE

    # 打印每个 epoch 的训练和验证误差
    print(f"Epoch {epoch + 1}/{maxepoch}, Train RMSE: {err_train[epoch]:.4f},
Test RMSE: {err_valid[epoch]:.4f}")

# 绘制 Loss 曲线
plt.plot(range(1, maxepoch + 1), err_train, label="Train Error", color="blue")
plt.plot(range(1, maxepoch + 1), err_valid, label="Validation Error",
color="red")
plt.xlabel("Epoch")
plt.ylabel("Error")
plt.legend()
plt.show()
```

2.5 计算 Precision@10 和 Recall@10

计算两个关键的推荐系统评估指标：Precision@10 和 Recall@10。Precision@10 衡量推荐结果中前 10 个推荐项中相关项的比例，反映推荐结果的准确性。Recall@10 衡量前 10 个推荐项中覆盖了所有相关项的比例，反映推荐结果的全面性。

```
# 定义常量
j = 10

# 计算 Precision@10
precisions = []
for i in range(user_num):
    user_i_rating_real = probe_vec[probe_vec["user_id"] == i + 1]
    user_i_rating_real = user_i_rating_real.sort_values(by="click",
ascending=False)
    user_i_rating = (
        np.dot(w1_P1[i, :], w1_M1[user_i_rating_real["item_id"].values - 1].T)
```

```
        + mean_rating
    )

    if len(user_i_rating_real) > j:
        top_j_pred = np.argsort(-user_i_rating)[:j]
        precision = np.sum(np.isin(top_j_pred, np.arange(j))) / j
        precisions.append(precision)
    else:
        ti = np.sum(user_i_rating_real["click"] >= 4)
        if ti != 0:
            precision = np.sum(user_i_rating[:ti] >= 4) / ti
            precisions.append(precision)

Pre = np.mean(precisions)

# 计算 Recall@10
recalls = []
for i in range(user_num):
    user_i_rating_real = probe_vec[probe_vec["user_id"] == i + 1]
    user_i_rating_real = user_i_rating_real.sort_values(by="click",
ascending=False)
    user_i_rating = (
        np.dot(w1_P1[i, :], w1_M1[user_i_rating_real["item_id"].values - 1].T)
        + mean_rating
    )

    if len(user_i_rating_real) > j:
        user_i_rating[user_i_rating < 4] = 0
        user_i_rating[user_i_rating >= 4] = 1
        ti = np.sum(user_i_rating)
        if ti != 0:
            bigerthan4 = np.sum(user_i_rating_real["click"] >= 4)
            tinri = np.sum(user_i_rating[:bigerthan4])
            recall = tinri / ti
            recalls.append(recall)
    else:
        ti = np.sum(user_i_rating_real["click"] >= 4)
        if ti != 0:
            recall = np.sum(user_i_rating[:ti] >= 4) / ti
            recalls.append(recall)

Re = np.mean(recalls)

print(f"Recall@10: {Re:.4f}, Precision@10: {Pre:.4f}")
```

2.6 推荐结果生成与保存

读取测试集数据，对每个测试用户生成推荐结果。通过计算用户对每个物品的预测评分，筛选出评分最高的前 10 个物品作为推荐结果。将推荐结果与测试集用户结合，生成推荐结果的 DataFrame，并保存为 txt 文件。

```
# 读取测试集数据
test = pd.read_csv("data/test.txt", header=None, sep=" ")
```

```
test.columns = ["user_id"]  # 设置列名为'user_id'

# 定义常量
k = 10  # 推荐的数量

# 初始化变量
record = []  # 存储推荐结果的列表

# 对每个测试用户进行推荐
for i in test["user_id"]:
    user_i_rating = np.dot(w1_P1[i - 1, :], w1_M1.T) + mean_rating
    used = data[data["user_id"] == i]["item_id"].values - 1
    user_i_rating[used] = 0  # 将已评分的电影的评分设为 0
    top_k_movies = np.argsort(user_i_rating)[-k:][::-1] + 1
    record.append((i, top_k_movies))  # 将用户 ID 和推荐结果加入列表

# 将推荐结果转换为指定格式的字符串
result_lines = []
for user_id, movies in record:
    movies_str = ",".join(map(str, movies))
    result_lines.append(f"{user_id}: {movies_str}")

# 将结果保存到 TXT 文件
with open("data/result.txt", "w") as file:
    for line in result_lines:
        file.write(line + "\n")
```

三　结果分析

3.1　参数设定

本实验的所用的参数如下，学习率（epsilon=50）控制每次参数更新的步长，动量参数（momentum=0.7）加速收敛并减少震荡；初始 epoch（epoch=1）和总训练次数（maxepoch=50）决定训练迭代的轮数；训练误差和验证误差数组（err_train 和 err_valid）分别用于记录每个 epoch 结束时的训练和验证误差，以监控模型表现；隐因子数量（num_feat=8）决定特征向量的维度，影响模型复杂度和推荐效果；每次训练三元组的数量（N=10000）设定了每个 batch 的大小，确保有效利用内存处理大规模数据。

```
epsilon = 50  # Learning rate 学习率
momentum = 0.7  # Momentum parameter 动量优化参数
epoch = 1  # Initial epoch 初始化 epoch
maxepoch = 50  # Total number of training epochs 总训练次数
err_train = np.zeros(maxepoch)  # Training error 训练误差
err_valid = np.zeros(maxepoch)  # Validation error 验证误差
err_random = np.zeros(maxepoch)  # Random error 随机误差
```

```
num_feat = 8  # Number of latent factors 隐因子数量
N = 10000  # Number of training triplets per epoch 每次训练三元组的数量
```

3.2 训练集表现

分别设定 num_feat 的值为 6，8，10 来探究训练集上的表现，如图 1、图 2、图 3。可以看到训练误差（Train Error）随着训练次数的增加，训练误差逐渐下降并趋于稳定，说明模型在训练数据上的拟合效果越来越好。但是随着隐因子数量的增加，验证误差（Validation Error）在初期也随训练次数的增加而下降，但在某个点之后开始趋于平稳甚至略有上升。这表明增加隐因子数量虽然可以提高训练数据上的拟合效果，但对验证数据的泛化能力提升有限，甚至可能导致过拟合。

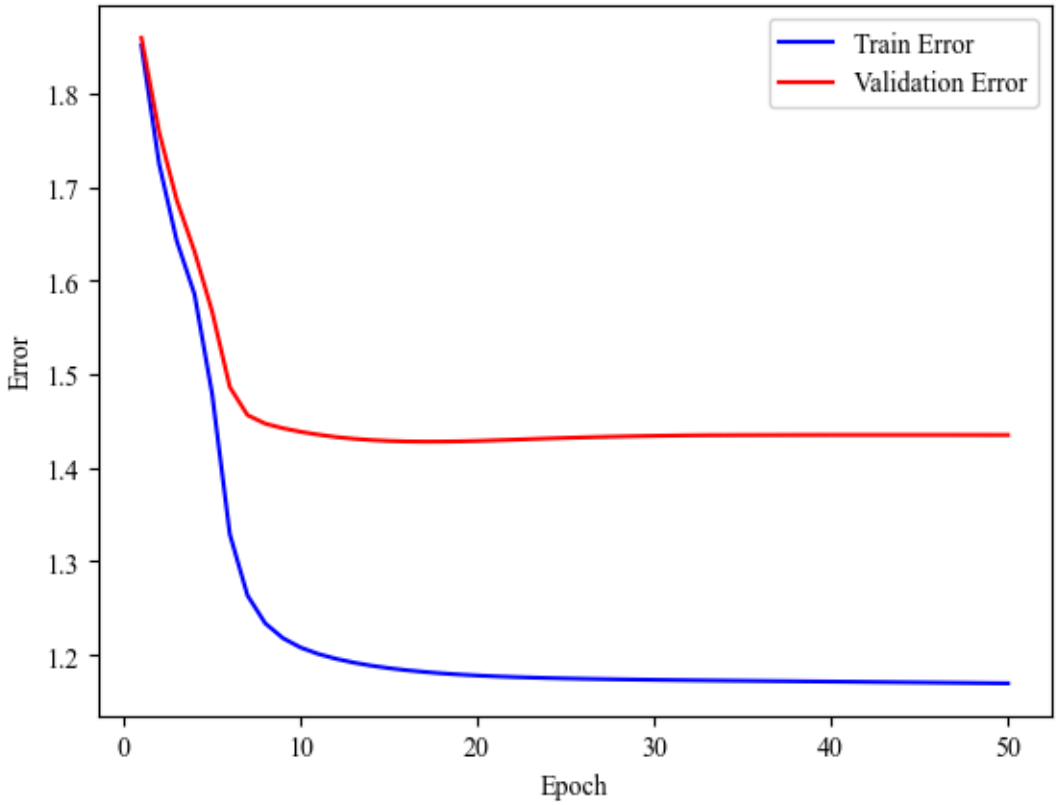

图 1　num_feat = 6 的 Loss 曲线

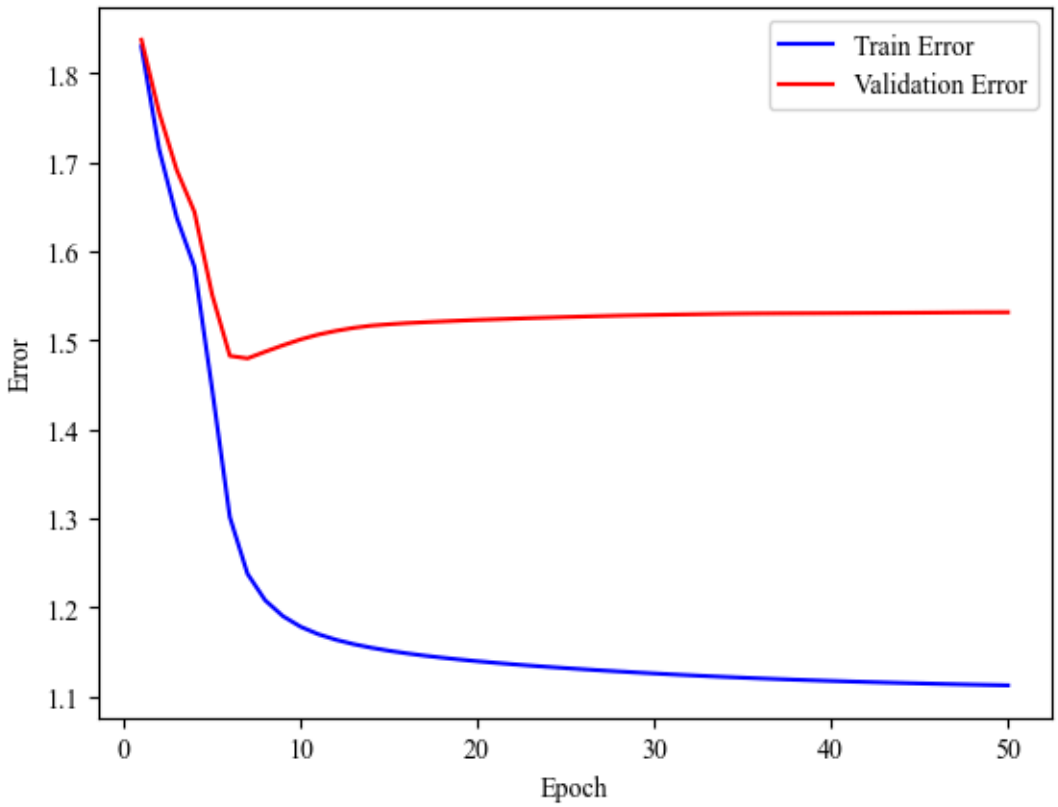

图 2　num_feat = 8 的 Loss 曲线

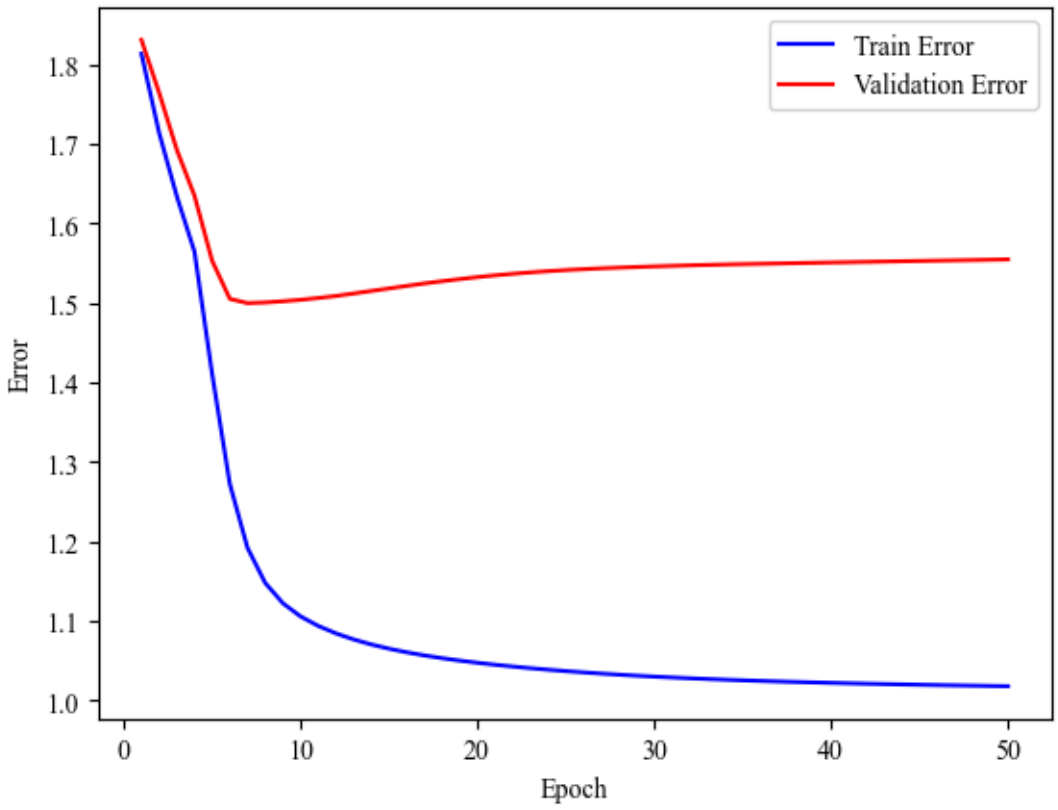

图 3　num_feat = 10 的 Loss 曲线

如表 1 所示，在五个随机种子下进行了不同测试以消除偶然性。隐因子数量为 6 时，模型在 Pre@10 和 Re@10 两个指标上的表现最好，表明此时模型的推荐效果较优。增加隐因子数量到 8 或 10 并未显著提升模型性能，反而有所下降。

隐因子数量	Pre@10	Re@10
6	0.5876	0.5903
8	0.5394	0.5415
10	0.5669	0.5689

表 1　不同隐因子的 Pre@10 和 Re@10

3.3　与其他模型的对比

采用 ALS 与 BPR 模型作为 baseline，与本模型进行对比。从图 4 中可以观察到 ALS（Alternating Least Squares）和 BPR（Bayesian Personalized Ranking）两种模型在 Precision@10 和 Recall@10 指标上的表现。

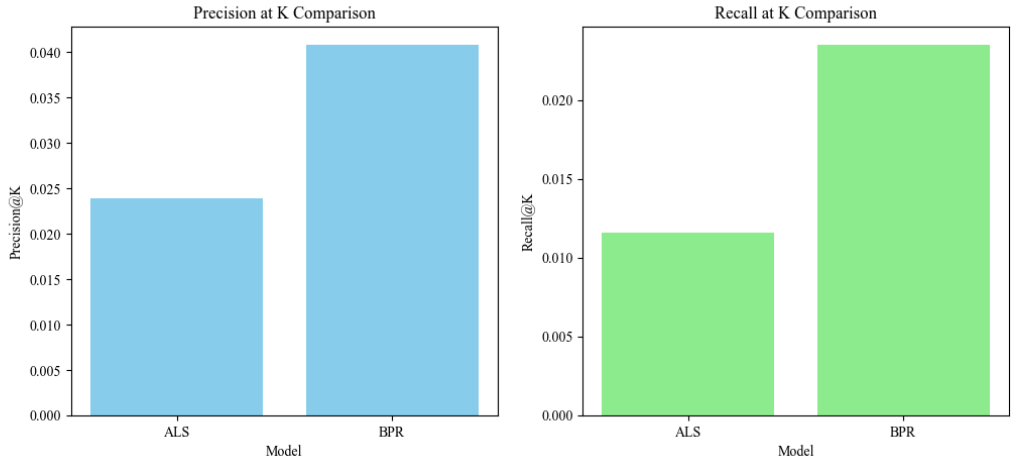

图 4　ALS 与 BPR 模型的表现

3.3.1　Pre@10 比较

- 本模型：隐因子数量为 6 时的 Pre@10 最高，为 0.5876；隐因子数量为 10 时，Pre@10 也较高，为 0.5669。
- ALS 模型：Pre@10 值较低，约为 0.025。
- BPR 模型：Pre@10 值较高，约为 0.04，但仍远低于本模型的表现。

3.3.2　Re@10 比较

- 本模型：隐因子数量为 6 时的 Re@10 最高，为 0.5903；隐因子数量为 10 时，Re@10 也较高，为 0.5689。
- ALS 模型：Re@10 值较低，约为 0.01。
- BPR 模型：Re@10 值较高，约为 0.022，但仍远低于本模型的表现。

从对比中可以明显看出，本模型在不同隐因子数量下的 Pre@10 和 Re@10 指标均显著高于 ALS 和 BPR 模型。这表明在推荐系统的前 10 个推荐项中，本模型的推荐准确性和召回率远高于 ALS 和 BPR 模型，特别是在隐因子数量为 6 和 10 时表现最佳。

四　实验总结

在本次实验中，我对比了不同推荐算法和模型在隐性反馈数据上的表现，重点评估了本模型、ALS（Alternating Least Squares）模型和 BPR（Bayesian Personalized Ranking）模型在 Precision@10 和 Recall@10 两个指标上的表现。

从实验结果中可以明显看出，本模型在不同隐因子数量配置下的 Precision@10 和 Recall@10 指标均显著优于 ALS 和 BPR 模型。特别是在隐因子数量为 6 和 10 时，本模型的推荐准确性和召回率表现最佳。这表明本模型在处理隐性反馈数据时具有较强的优势，能够提供更为精准和全面的推荐结果。