十月组队学习_推荐系统

https://pic1.zhimg.com/80/v2-4c7edc94af4f7f40eddc4cb949f89cf8_720w.jpg

mind by h1astro

2 推荐系统概述

2.4 课后思考

3 协同过滤

4 矩阵分解

4.1 隐语义模型与矩阵分解

31

本节以推荐场景为例来介绍如何利用机器学习来解决实际产品中的问题。以在线电影网站为例，一位用户（下文简称为小王）在观看完电影《Star Wars: Episode IV - A New Hope》（下文简称为星球大战）后，**会倾向于观看同样是太空题材的电影《Star Wars: Episode V - The Empire Strikes Back》**（下文简称为帝国反击战）的概率会大幅提高。这个例子中，**星球大战和帝国反击战**之间的关联性，一定程度上决定了**小王观看帝国反击战的概率**。

本节将以这个例子为引子，介绍一种**隐语义模型**（Latent Factor Model，LFM）来解决这个问题的方法。

31

本节以隐语义模型（LFM）的数学原理为引子，**对矩阵分解（Matrix Factorization）**这一经典问题进行深入讨论。

矩阵分解是一个**经典的机器学习问题**，**在推荐系统、自然语言处理、计算机视觉**等领域有着广泛应用。

矩阵分解的目标是将一个**大的矩阵**分解成两个**小的矩阵**的乘积，**这样**可以在保持原矩阵信息的同时，大幅减少数据的存储和计算成本。

隐语义模型（LFM）是一种**基于矩阵分解的推荐算法**，它通过将用户和物品的评分矩阵分解为两个低维矩阵，从而发现用户和物品之间的潜在关联。

31

本节将介绍LFM的一种实现方法：**奇异值分解（Singular Value Decomposition，SVD）**。

31

SVD是一种经典的矩阵分解方法，它通过将矩阵分解为三个矩阵的乘积：$R = U\Sigma V^T$。

SVD的核心思想是保留最大的奇异值，从而实现数据的降维和去噪。

31

以小王的电影评分数据为例，将这个数据用矩阵表示，**行代表用户，列代表电影**，**矩阵中的元素代表用户对电影的评分**。

- 矩阵的行数为小王所观看的电影数量，矩阵的列数为小王所观看的用户数量，矩阵中的元素为小王对电影的评分。
- 为简化问题，本节假设小王只观看了5部电影，并给出了5个评分。
- 为了模拟现实场景，我们在矩阵中加入了一些缺失的评分，用“?”表示。

32

这里的矩阵分解问题，可以理解为：**将一个大的评分矩阵分解为两个小的矩阵的乘积**，**其中一个矩阵代表用户的潜在特征，另一个矩阵代表电影的潜在特征**。

- 例如，一个大的评分矩阵可以分解为一个代表用户特征的矩阵和一个代表电影特征的矩阵，这样可以大幅减少数据的存储和计算成本。
- 这种分解方法在推荐系统中有广泛的应用，例如通过分析用户的评分行为，可以发现用户的潜在兴趣，从而为用户推荐他可能感兴趣的电影。

32

Step 1：初始化 P 和 Q

这个矩阵乘法分解的第一步，是随机初始化矩阵 P 和 Q，使得 $P \times Q = R$。

	小明	小红	小刚	小强	小王
电影1	0.1	0.2	0.3	0.4	0.5
电影2	0.6	0.7	0.8	0.9	1.0
电影3	1.1	1.2	1.3	1.4	1.5
电影4	1.6	1.7	1.8	1.9	2.0
电影5	2.1	2.2	2.3	2.4	2.5

32

Step 2：计算 $\hat{R} = P \times Q$

- 此时矩阵乘法会计算每个用户对每个电影的评分，例如小王的第一个评分，即这个矩阵的第一行第一列，会通过计算 $p_{11} \cdot q_{11} + p_{12} \cdot q_{21} + p_{13} \cdot q_{31} + p_{14} \cdot q_{41} + p_{15} \cdot q_{51}$ 来得到。
- 例如，小王对电影1的评分为：

$$0.1 \times 0.1 + 0.2 \times 0.2 + 0.3 \times 0.3 + 0.4 \times 0.4 + 0.5 \times 0.5 = 0.55$$

- 这个结果会与原始评分矩阵中的缺失评分进行比较，以评估模型的预测效果。

	小明	小红	小刚	小强	小王
电影1	0.55	1.10	1.65	2.20	2.75
电影2	1.32	2.64	3.96	5.28	6.60
电影3	2.09	4.18	6.27	8.36	10.45
电影4	2.86	5.72	8.58	11.44	14.30
电影5	3.63	7.26	10.89	14.52	18.15

33

两类特征的含义不明确，**或可以理解为**：

- 第一个特征**更偏向**于“不太关心大片画面的文艺片”，**而**第二个特征**更偏向**于“不太关心文艺情节的大片”。
- 这种**模糊性**是矩阵分解的一个**常见问题**，**因为**模型**仅根据评分数据进行优化**，**而没有明确的特征定义**。

	小明	小红	小刚	小强	小王
第一特征	0.1	0.2	0.3	0.4	0.5
第二特征	0.6	0.7	0.8	0.9	1.0

	小明	小红	小刚	小强	小王
第一特征	0.1	0.2	0.3	0.4	0.5
第二特征	0.6	0.7	0.8	0.9	1.0

33

解决方法：**引入正则项**，**对特征矩阵进行惩罚**，**以提高模型的泛化能力**。

- 例如，通过在损失函数中加入正则项，可以限制特征矩阵的取值范围，从而减少模型对噪声数据的过拟合。
- 这种方法在实际应用中非常有效，尤其是在数据稀疏的情况下，能够显著提高模型的预测效果。

	第一特征	第二特征	第三特征	第四特征
第一人	0.1	0.2	0.3	0.4
第二人	0.5	0.6	0.7	0.8
第三人	0.9	1.0	1.1	1.2
第四人	1.3	1.4	1.5	1.6
第五人	1.7	1.8	1.9	2.0

33

正则化（Regularization）是一种通过在损失函数中引入惩罚项，以防止模型过拟合的技术。

- 在矩阵分解中，正则化通过对特征矩阵的取值范围进行限制，从而减少模型对噪声数据的过拟合。
- 例如，通过在损失函数中加入L2正则项，可以有效地减小特征矩阵的取值，从而提高模型的泛化能力。

34

在矩阵分解问题中，**我们要找的是一个最优的特征矩阵来使得矩阵乘积和原评分矩阵的差异最小**。换言之，我们希望找到一个特征矩阵，使得其乘积能够尽可能地接近原评分矩阵。

- 例如，通过调整特征矩阵的参数，可以使得矩阵乘积的结果与原评分矩阵的差异达到最小。
- 这种优化过程通常采用梯度下降法来实现，通过不断调整参数，以逐步减小误差，最终达到一个最优的解。

34

矩阵分解算法的目标是：**把原矩阵R分解成两个矩阵P和Q的乘积**，**使得P和Q的乘积尽可能接近原矩阵R的值**。换言之，我们希望找到一个矩阵P和一个矩阵Q，使得 $R \approx P \times Q$。

- 例如，通过调整P和Q的参数，可以使得P和Q的乘积与原矩阵R的差异达到最小。
- 这种方法在实际应用中非常有效，尤其是在数据稀疏的情况下，能够显著提高模型的预测效果。

35

矩阵分解算法的一般形式可以表示为：给定一个大小为 $m \times n$ 的矩阵 R，其中 m 表示用户数，n 表示物品数，矩阵 R 中的元素 r_{ij} 表示用户 i 对物品 j 的评分。我们希望将矩阵 R 分解为两个矩阵 P 和 Q，使得 $R \approx P \times Q$。其中，P 是一个 $m \times k$ 的矩阵，Q 是一个 $k \times n$ 的矩阵，k 是一个远小于 m 和 n 的正整数，表示隐语义的维度。

$$R_{m \times n} \approx P_{m \times k} \times Q_{k \times n} \tag{1}$$

这里，p_{ik} 表示用户 i 对隐语义 k 的偏好程度，q_{kj} 表示物品 j 在隐语义 k 上的属性。矩阵分解的目标是通过最小化预测评分与实际评分的误差来优化 P 和 Q。

35

矩阵分解算法的求解通常采用梯度下降法或其变体来进行优化。梯度下降法通过计算损失函数关于参数的梯度，并沿着梯度的负方向更新参数，以逐步减小误差。

- 例如，通过计算损失函数关于 p_{ik} 和 q_{kj} 的梯度，并分别更新 p_{ik} 和 q_{kj}，可以使得预测评分与实际评分的误差逐渐减小。
- 这种方法在实际应用中非常有效，尤其是在数据稀疏的情况下，能够显著提高模型的预测效果。

35

Basic SVD（基础奇异值分解）是一种经典的矩阵分解方法，通过将评分矩阵分解为两个低维矩阵的乘积，以发现用户和物品的潜在特征。

- 例如，通过将评分矩阵分解为两个低维矩阵，可以发现用户对某类电影的偏好，从而为用户推荐他可能感兴趣的电影。
- 这种方法在实际应用中非常有效，尤其是在数据稀疏的情况下，能够显著提高模型的预测效果。

35

Basic SVD的优化目标是：**最小化预测评分与实际评分的误差平方和**，**并加入正则项以防止过拟合**。

- 例如，通过最小化误差平方和，可以使得预测评分与实际评分的差异达到最小。
- 这种方法在实际应用中非常有效，尤其是在数据稀疏的情况下，能够显著提高模型的预测效果。

36

在工业场景下，通过调试Basic SVD模型的超参数，以提高模型的预测效果，是一项重要的工作。

- 例如，通过调整学习率、正则项系数和隐语义维度，可以显著提高模型的预测效果。
- 这种方法在实际应用中非常有效，尤其是在数据稀疏的情况下，能够显著提高模型的预测效果。

37

SVD++，**即带有隐式反馈的矩阵分解**，是一种经典的矩阵分解算法，它在Basic SVD的基础上引入了隐式反馈信息，以提高模型的预测效果。

- 例如，通过将用户的隐式反馈信息（如点击、浏览、购买等）引入模型，可以显著提高模型的预测效果。
- 这种方法在实际应用中非常有效，尤其是在数据稀疏的情况下，能够显著提高模型的预测效果。

37

基于邻居的协同过滤（Neighborhood）是一种经典的推荐算法，其核心思想是利用用户或物品的邻居信息来进行推荐。

- 例如，通过计算用户的邻居，可以发现与该用户兴趣相似的其他用户，从而为该用户推荐他们喜欢的物品。
- 这种方法在实际应用中非常有效，尤其是在数据稀疏的情况下，能够显著提高模型的预测效果。

37

基于邻居的协同过滤（Neighborhood）的优化目标是：**最小化预测评分与实际评分的误差平方和**，**并加入正则项以防止过拟合**。

- 例如，通过最小化误差平方和，可以使得预测评分与实际评分的差异达到最小。
- 这种方法在实际应用中非常有效，尤其是在数据稀疏的情况下，能够显著提高模型的预测效果。

38

潜在语义模型（Latent Semantic Model，LSM）是一种经典的推荐算法，其核心思想是利用潜在语义信息来进行推荐。

- 例如，通过将用户和物品映射到一个低维的潜在语义空间，可以发现用户和物品之间的潜在关联，从而为用户推荐他可能感兴趣的物品。
- 这种方法在实际应用中非常有效，尤其是在数据稀疏的情况下，能够显著提高模型的预测效果。

	小明	小红	小刚	小强	小王
电影1	0.1	0.2	0.3	0.4	0.5
电影2	0.6	0.7	0.8	0.9	1.0
电影3	1.1	1.2	1.3	1.4	1.5
电影4	1.6	1.7	1.8	1.9	2.0
电影5	2.1	2.2	2.3	2.4	2.5

为方便理解，下面给出一个简单的例子。

- 例如，通过将用户和物品映射到一个低维的潜在语义空间，可以发现用户和物品之间的潜在关联，从而为用户推荐他可能感兴趣的物品。
- 这种方法在实际应用中非常有效，尤其是在数据稀疏的情况下，能显著提高模型的预测效果。

38

```
class Model:
    def __init__(self, training_data, ftrs, iterations, lr, lambda_u, lambda_v):
        self.ftrs = ftrs                    # 隐变量维度
        self.iter = iterations              # 迭代次数

        self.lr = lr                        # 学习率
        self.lambda_u = lambda_u            # 用户偏置惩罚
        self.lambda_v = lambda_v            # 物品偏置惩罚
        self.users = {}                     # 用户字典
        self.items = {}                     # 物品字典
        self.bias = {}                      # 全局偏置
        self.u = None                       # 用户隐变量矩阵
        self.v = None                       # 物品隐变量矩阵
        self.training_data = training_data  # 训练数据

        # 初始化训练时间，以及损失
        # 一般情况下损失是递减的，但偶尔会不是，这时候可以适当调小学习率
        self.train_time = []
        self.loss = []

    def fit(self):
        ''''''
        # 初始化用户和物品字典
        for user, item in self.training_data:
            self.users[user] = {}
            self.items[item] = {}
        # 初始化偏置
        self.bias = {}
        for user, rating in self.training_data:
            self.bias[user] = 0.0
            for item, _ in self.training_data[user]:
                self.bias[(user, item)] = 0.0
        # 初始化隐变量矩阵
        self.u = np.random.rand(len(self.users), self.ftrs)
        self.v = np.random.rand(len(self.items), self.ftrs)
        # 开始迭代
        for i in range(self.iter):
            self.train(self.lr, self.lambda_u, self.lambda_v)
        self.fit_time = time.time() - start_time
    def fit_step(self):
        ''''''
        self.iter += 1
        start = time.time()
        self.train(self.lr, self.lambda_u, self.lambda_v)
        self.train_time.append(time.time() - start)
    def predict(self):
        ''''''
        # 预测评分
        for user, item, _ in self.training_data:
            if user in self.users and item in self.items:
                self.bias[(user, item)] = self.bias[(user, item)] + self.u[self.users[user]].dot(self.v[self.items[item]])
        return self.bias
    def rmse(self):
        ''''''
        # 计算RMSE
        from math import sqrt
        ret = 0.0
        for user, item, rating in self.training_data:
            if user in self.users and item in self.items:
                ret = ret + (self.bias[(user, item)] - rating) ** 2
        return sqrt(ret / len(self.training_data))
    def mae(self):
        ''''''
        # 计算MAE
        from math import fabs
        ret = 0.0
        for user, item, rating in self.training_data:
            if user in self.users and item in self.items:
                ret = ret + fabs(self.bias[(user, item)] - rating)
        return ret / len(self.training_data)
    def train(self, lr, lambda_u, lambda_v):
        ''''''
        # 梯度下降法，参数说明同初始化
        for (user, item, rating) in self.training_data:
            if user in self.users and item in self.items:
                u = self.u[self.users[user]]
                v = self.v[self.items[item]]
                b = self.bias[(user, item)]
                # 计算误差
                e = rating - (b + u.dot(v))
                # 更新偏置
                self.bias[(user, item)] = self.bias[(user, item)] + lr * (e - lambda_v * self.bias[(user, item)])
                # 更新隐变量
                self.u[self.users[user]] = self.u[self.users[user]] + lr * (e * v - lambda_u * u)
                self.v[self.items[item]] = self.v[self.items[item]] + lr * (e * u - lambda_v * v)
        self.loss.append(self.rmse())
    def show_loss(self):
        ''''''
        # 绘制loss变化曲线
        plot_loss(self.loss, name, time)
    def show_time(self):
        ''''''
        # 绘制时间变化曲线
        plot_time(self.train_time, name, time)
    def show_rmse(self):
        ''''''
        # 输出RMSE
        print('%s: %f' % (time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(time.time())), self.rmse()))
    def show_mae(self):
        ''''''
        # 输出MAE
        print('%s: %f' % (time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(time.time())), self.mae()))
```

40

```
# 主函数
def main(training_data, ftrs, iterations, lr, lambda_u, lambda_v, name, time):
    ''''''
    # 初始化模型
    model = Model(training_data, ftrs, iterations, lr, lambda_u, lambda_v)
    # 开始训练
    model.fit()
    # 输出RMSE
    model.show_rmse()
    # 输出MAE
    model.show_mae()
    # 绘制loss变化曲线
    model.show_loss(name, time)
    # 绘制时间变化曲线
    model.show_time(name, time)
    # 预测
    model.predict()
    # 返回模型
    return model
# 主函数结束
if __name__ == '__main__':
    # 设置参数
    ftrs = 20
    iterations = 100
    lr = 0.01
    lambda_u = 0.1
    lambda_v = 0.1
    # 读取数据
    training_data = read_data('./data/ml-100k/u.data')
    # 开始计时
    start = time.time()
    # 调用主函数
    model = main(training_data, ftrs, iterations, lr, lambda_u, lambda_v, 'ml-100k', start)
```

4.2 隐语义模型

31

本节以推荐场景为例来介绍如何利用机器学习来解决实际产品中的问题。以在线电影网站为例，一位用户（下文简称为小王）在观看完电影《Star Wars: Episode IV - A New Hope》（下文简称为星球大战）后，**会倾向于观看同样是太空题材的电影《Star Wars: Episode V - The Empire Strikes Back》**（下文简称为帝国反击战）的概率会大幅提高。这个例子中，**星球大战和帝国反击战**之间的关联性，一定程度上决定了**小王观看帝国反击战的概率**。

本节将以这个例子为引子，介绍一种**隐语义模型**（Latent Factor Model，LFM）来解决这个问题的方法。

31

本节以隐语义模型（LFM）的数学原理为引子，**对矩阵分解（Matrix Factorization）**这一经典问题进行深入讨论。

矩阵分解是一个**经典的机器学习问题**，**在推荐系统、自然语言处理、计算机视觉**等领域有着广泛应用。

矩阵分解的目标是将一个**大的矩阵**分解成两个**小的矩阵**的乘积，**这样**可以在保持原矩阵信息的同时，大幅减少数据的存储和计算成本。

隐语义模型（LFM）是一种**基于矩阵分解的推荐算法**，它通过将用户和物品的评分矩阵分解为两个低维矩阵，从而发现用户和物品之间的潜在关联