基于神经网络和决策树的电影预测与决策分析

摘　要

本文为寻找影响电影票房的因素和最赚钱的拍摄方案，并对电影评分和评分数做出预测，建立基于神经网络的评分预测模型和基于决策树的电影赚钱决策模型，运用 BP 算法、CRT 生长法等对模型进行求解，使用 Python 对数据进行清洗、分析和预测，寻找到影响电影票房的主要原因，成功预测电影评分及评分数，并提出电影赚钱的方案。

对于问题一：为研究影响电影票房的因素，需要对附件中的数据进行分析。首先，按照删除缺失、重复和与票房几乎没有相关关系的数据、3σ 原则剔除异常数据、字符串型数据采用特征编码更新的原则，对原始数据进行清洗；不同类型的数据采取不同的分析方法：对于有具体数据的变量而言，以散点图的形式展示，并计算变量与票房的相关系数，分析其对电影票房的影响；对于字符串类型的数据，则将其解析后，按照高票房电影（top25%）中出现的频次进行排序，并以直方图的形式展示。

本文分析获得的结论是：影响电影票房的因素主要有十个，分别为：演员、导演、电影关键词、影片种类、影片发布时间，电影预算、电影流行程度，影片时长、电影制片厂、评分数。当电影发布时间集中于偶数年的 5-7 月份、影片时长为 100-150 分钟、投资越大、热度越高、评分人数越多且以具有英雄主义和理想主义等美国元素的动作冒险片或戏剧性喜剧片时，电影的商业价值越高，电影票房越好。

对于问题二：为预测电影评分和评分数，选择预算、流行度、票房和电影类型作为变量。将预处理后的变量数据按照 K 折交叉验证的方法分为十份，一份作为验证集，九份作为训练集，并建立**基于 BP 神经网络的评分预测模型**，将数据输入模型进行训练。由图 19 和 21 可知，模型训练效果较好，且泛化能力好，可以对测试集进行预测。将 1000 部电影的测试集数据输入模型后，获得图 20 和 22 的预测结果。由散点图可知，对于评分数而言，预测数据均匀分布在真实有效值两侧，预测效果好；对于评分而言，预测数据与真实有效值有一定的差据，但可以一定程度上预测电影评分，预测效果较好。

对于问题三：为确定最优赚钱拍摄方案，选择预算、流行度和电影类型作为自变量，票房为因变量进行分析。首先对原始数据进行预处理，并将电影按票房分为 5 类。建立**基于 CRT 生长法的决策树模型，**按照信息增益率划分属性生成决策树，并根据决策树中自变量重要性提出最优方案。SPSS 仿真分析结果如下：按照自变量的重要性排序做出决策。首先考虑预算的投入，在资金充裕的情况下，投入电影制作的预算越多，票房越高；其次，考虑电影的热度，结合时代背景，迎合大众口味进行拍摄；最后考虑电影的类型，按照优先级顺序依次考虑拍摄外国片、西方片、记录片、爱情片、恐怖片和音乐片类型的电影。

关键词：商业电影 预测与决策 神经网络 决策树 Python

一 问题重述

1.1 背景与研究意义

随着国民经济发展和生活水平的提高，人们对于娱乐文化产品的需求量越来越高，日益增长的娱乐需求极大地推动了各国电影产业的发展，创作活力持续迸发，市场规模不断扩大。面对迎合市场需要和满足消费者多元化审美的挑战，以资本为核心的商业电影需要开拓创新，创作出适应当代审美的优质电影。为了让电影在众多作品中脱颖而出，赚取更多的票房，电影商需要寻找属于商业电影的“成功公式”。

实际上，影响电影票房和评分的因素众多，例如不同的影视演员带来不同的粉丝效益，影片的投入大小，导演和制片组的技能高低等。为了一定程度上减小“大投入小收益”的风险，本文将建立数学模型，分析影响电影票房的因素，以便于创作出满足大众口味的高盈利性商业电影，推动未来电影产业的高质量发展。

1.2 需要解决的问题

基于上述背景，本文将建立数学模型，根据附件中的数据解决以下问题：

（1）对 4803 组电影基本信息和演职员信息的数据集进行数据清洗、数据挖掘、数据分析和数据可视化等操作，从不同维度分析影响电影票房的因素，并讨论结果。

（2）选择合适的指标，对附件 tmdb_1000_predict.csv 中 1000 部电影的基本信息进行特征提取，建立机器学习的预测模型，预测电影的 vote_average 和 vote_count。

（3）制定拍摄电影的最佳赚钱模式或决策方法，并说明原因；否则，说明无法寻找到最佳决策的理由和依据。

二 问题分析

2.1 对问题一的分析

电影票房受多维因素的影响，如制作成本、电影宣传、拍摄技巧、影片情节等等。为研究影响电影票房的主要因素，首先对原始数据进行清洗：删除有特征缺失或重复的数据；将非高斯分布的数据转化为标准正态分布，删除不满足 3σ 原则的数据；将与电影本身无关的因素，例如电影主页、简介、编号等变量删除；将电影类型、关键词等字符串类型的数据类型进行转换，以便于进行后期的数据处理。

根据数据类型的不同，采取不同的数据分析方式，求得相关系数矩阵，寻找相关系数大的因素作为主要因素，选择最具有代表性的变量，作为影响电影票房的关键因素。

2.2 对问题二的分析

问题二要求根据电影的基本信息对 1000 部电影的评分和评分数进行预测，该预测基于问题一的分析结果，问题一中确定的影响电影票房的因子与评分和评分数变量之间存在潜在机制。

基于以上分析，本文将建立 BP 神经网络预测模型，使用 K 折交叉验证，将预处理后的数值影响因子分为相等的十部分，一部分作为验证集，其余九部分作为训练集，新的 1000 部电影作为测试集，完成预测评分和评分数的任务目标。

2.3 对问题三的分析

为确定拍摄电影的最佳赚钱方案，需要对样本的多个变量属性进行分析，寻找最优解。方案的选择可以通过对选择考虑决策树是一种将具有 m 维特征的 n 个样本分到

p 个类别的分类方法，因此本文将建立基于决策树的赚钱决策模型，将电影按照票房分为五类，使用 CRT 生长法对电影样本进行分析，按照重要性排序依次做出决策。

三 基本假设

1. 假设附件中的全部数据科学、真实、有效。
2. 假设高票房的电影为票房排名前 25%的电影。
3.假设缺失和重复的数据均为无效数据，在数据清洗时删除不会对结果产生影响。

四 名词解释与符号说明

4.1 名词解释

- vote_average：电影评分，可简称为 va。
- vote_count：对该电影评分的数，可简称为 vc。
- revenue：电影票房。

4.2 符号说明

符号	意义	单位
X	原数据矩阵	/
I_i^s	输入层阈值状态	/
H_j^s	隐藏层单元状态	/
O^s	输出层单元状态	/
T^s	理想输出单元状态	/
w_{ij}	输入层到隐藏层的权值	/
$\bar{w}_j$	隐藏层到输出层的权值	/
$\varphi(x)$	激活函数	
e	输出误差	/
η	学习效率	/
D	决策树中的样本集合	/
a	样本属性	/
$Ent(D)$	信息熵	/
$Gain(D)$	信息增益	/
$Gain\_ratio(D)$	信息增益率	/
$IV(D)$	固有值	/

五 模型的建立和求解

5.1 影响电影票房因素分析

根据附件中采集到的数据可知，原始变量数目较多，且信息琐碎，难以从中直观分析得到商业电影的成功法则。为获取高质量的分析因子，本文将进行数据清洗、数据挖掘、数据分析和数据可视化，挖掘数据价值，寻找电影票房的影响因素。

5.1.1 数据清洗

由于数据采集获取的数据常出现异常、遗漏或者缺损等情况，因此需要对数据进行处理以便于后期的分析。通过对原始数据的观察和分析，本文对附件数据的清洗分为四个主要内容：

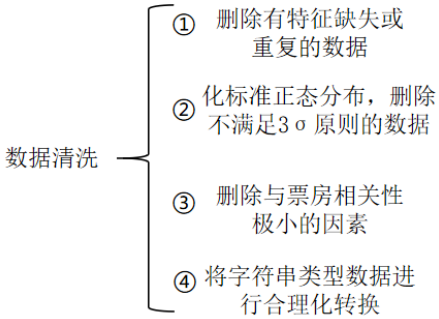


图 1 数据清洗内容

①删除有特征缺失或重复的数据。经过对数据库的分析可知，部分数据缺失信息较多且数量少，样本为无效性，则需剔除含有缺失值的样本；某两个数据有 100%的相关关系，即变量与票房相关性一致，则此类变量为重复数据，需剔除。

②化标准正态分布，删除不满足 3σ 原则的数据。对数据清洗时，会出现错误数据，例如在对电影预算（budget）数据处理时发现，部分数据为 1，与绝大部分数据的数量级相差高达 10^9，且与其本身意义不符。基于以上分析，将非高斯分布的连续数据化为标准化，使用 3σ 原则对异常数据进行剔除。

STEP1：对数变换。将原始数据取对数值作为新的分布，使数据服从对数正态分布的数据正态化，便于进行标准化处理：

$$X^{'}=\log(X)，\quad X^{'}\sim N(\mu,\sigma^2)$$

若存在数据为 0，则删除异常值 0。

STEP2：根据 3σ 原则，数值分布在（μ-3σ，μ+3σ）中的概率为 0.9974，因此数据落在（μ-3σ，μ+3σ）以外的概率小于 0.3%，认为是小概率事件，剔除该部分的数据，完成对错误数据的处理。

STEP3: 将对数正态分布转化为对数的标准正态分布，完成无量纲处理。由于数据的单位不同，可能对后续处理产生较大的影响，因此需要对数据进行无量纲处理。

$$\frac{X^{'}-\mu}{\sigma}\sim N(0,1)$$

③删除与票房相关性极小的因素。根据分析可知，电影的票房高低与电影是否有主页、简介几乎无关，因此忽略该部分变量对票房的影响。

④将字符串类型数据进行合理化转换。由附件可知，除少部分数据外，大部分数据是字符串格式，因此本文采用编码的形式，将其用十进制的数字表示，便于后续处理；据初步分析，电影共 20 个类型，数量较少且与电影票房相关性较高，使用 onehot 编码表示；对于时间而言，则将数据进行标准化处理。

基于上述分析，对数据的清洗结果如下：

（1） 删除特征缺失数据：

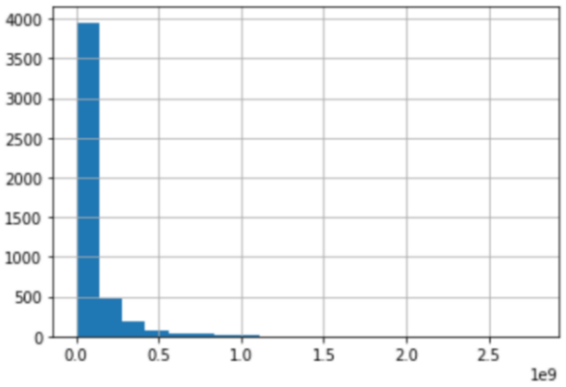
由于 runtimes 或 release_date 为空的数据有 3 个，数量较少且数据缺失，则剔除该 3 个样本；

（2） 删除重复性数据：

original_title 与 title 均为题目，且部分影片名为原影片名的翻译，因此两者称为重复性数据，与票房相关性相同，则保留 title 内容，删除 original_title 变量；

（3） 删除不满足 3σ 的数据：

对 revenue、budget、popularity 数据进行处理。以 revenue 为例，原数据分布和删除异常值 0 后的对数正态分布分别为:



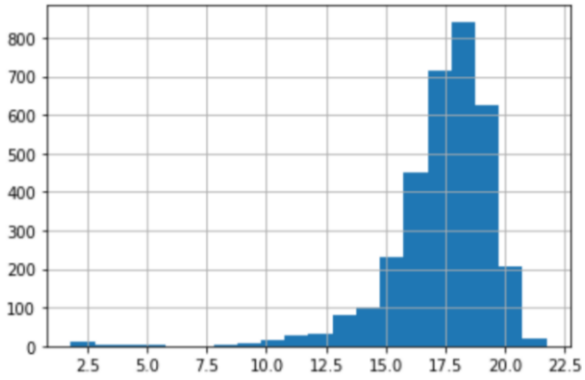


图 2 revenue 原数据分布　　删除异常值 0 后的 revenue 对数正态分布

利用 3σ 原则，删除异常值后获取的数据如下图所示：

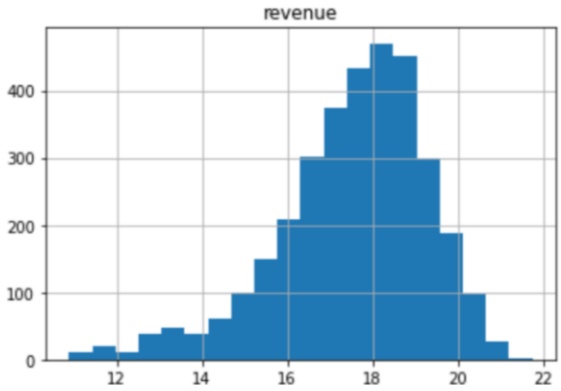


图 3　3 σ 原则剔除异常数据后 revenue 数据分布

同理，剔除异常数据后的 budget 和 popularity 数据分布如下所示：

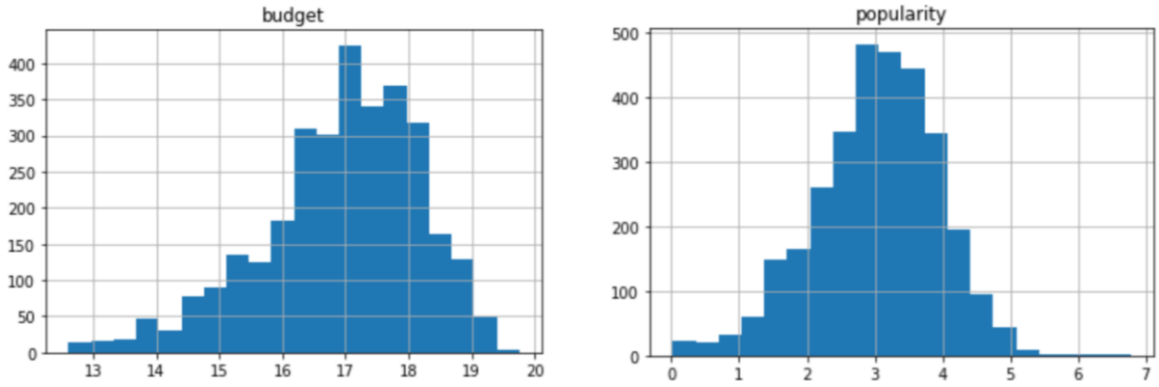


图 4　3σ 原则剔除异常数据后 budget（左）和 popularity（右）数据分布

（4） 删除与票房相关性极小的数据：

经分析，已发布状态的电影为 4795 个，占总影片数的 99.83%，无法呈现与票房的相关关系，则删除 status 变量；同理，spoken_languages 与 original_language 中，English 占总影片数的 93.80%，删除以上 2 个变量。homepage 和 production_countries 与票房相关性极低，删除以上变量。

（5） 字符串类型数据用编码表示：

genres 共 20 种，将该特征转换为 onehot 编码形式；

keyword 共 9813 种，特征过多，数据挖掘和分析时将以高分电影或高票房电影出现的频次进行降序排序分析，但在后续训练中不考虑该变量的影响，cast、crew、product_companies 数据处理方式相同；

release_date 表示日期，将其从字符串转化为 datetime_format 的标准化形式，以便于后续的数据处理；

根据上述结果，最终经清洗处理后获取的数据如下：

表 1 清洗处理后的数据及类型

数据名称	数据类型
budget	float64
popularity	float64
revenue	float64
runtime	float64
vote_average	float64
title	object
keywords	object
production_companies	object
cast	object
crew	object
vote_count	int64
genres0—genres19	int64
release_date	datetime64[ns]

5.1.2 数据分析与可视化

若需寻找对电影票房影响较大的因素，则需分析某变量与票房之间的相关关系。由于数据类型的不同，本文对数据的分析分为以下两种方式：

方法一：对高票房的数据进行降序排序。以前 25%作为高票房的样本，按照变量出现的频次对电影进行降序排序，分析该变量对电影票房是否有显著性影响。

方法二：求变量与票房的相关关系。使用 Python 计算某变量与票房的相关系数，根据相关系数的大小进行排序，选择相关关系较大的变量作为影响电影票房的因素。

方法一主要应用于对字符串进行处理，从排名中分析观众的口味，而方法二着重分析含有具体数据的变量与电影票房的相关关系。

5.1.2.1 对高票房数据的排序分析

keyword:电影关键词（keyword）的不同，观众对电影的吸引力也不同，因此电影关键词对电影票房具有较强的影响。将票房前 25%的电影按照关键词出现的频次排序如下：

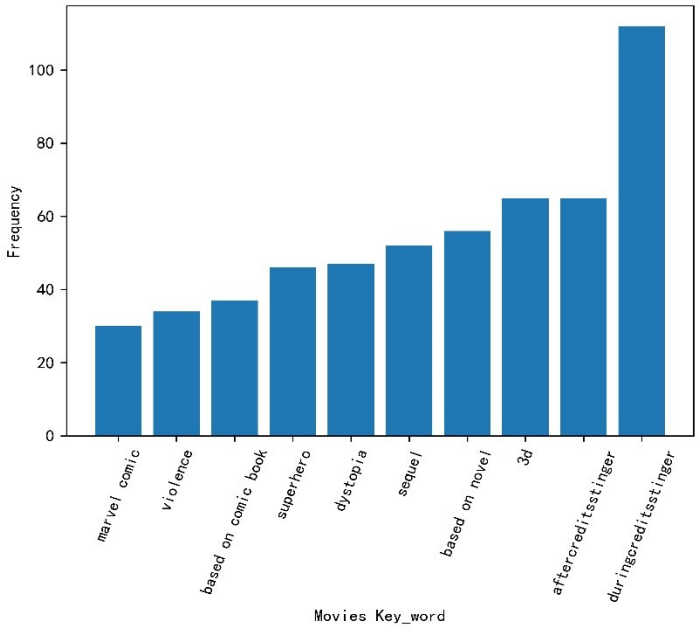


图 5 电影关键词在高票房电影中的出现的频次排序

根据查阅资料可知，duringcreditsstinger、aftercreditsstinger 变量没有实际意义，可以忽略。因此，关键词主要集中于 3d、小说改编、续集、反乌托邦、超级英雄、暴力（violence）等关键词上，一定程度上反应了美国观众的观影口味主要集中于有英雄主义和理想主义等的连续小说大片。

genres: 影片种类（genres）的不同，也会一定程度上反应观众的观影口味。如下图所示：

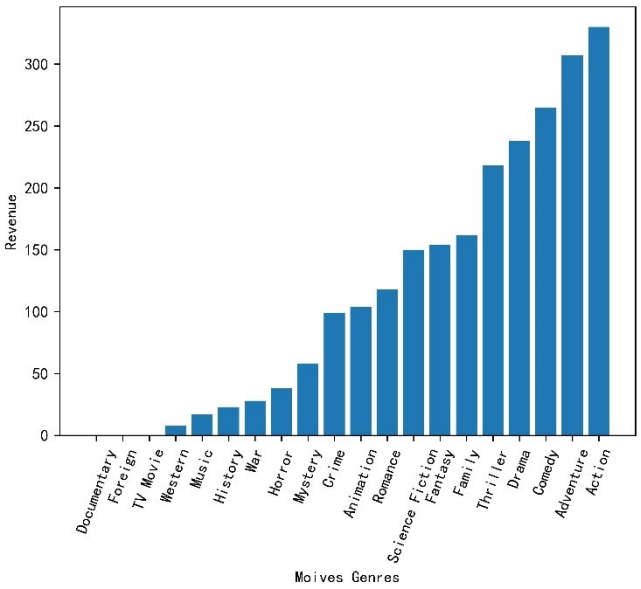


图 6 电影种类在高票房电影中的出现的频次排序

在 20 个电影种类的排序中，可以看到动作片（Action）、冒险片（Adventure）、喜剧片（Comedy）、戏剧片（Drama）、恐怖片（Thriller）等类型的电影出现的频次较高，反应了观众对附加恐怖元素的动作冒险类和带有戏剧性元素的喜剧片等电影的关注度较高。

release_date:影片的发布时间（release_date）会出现一定的规律，如下图所示：

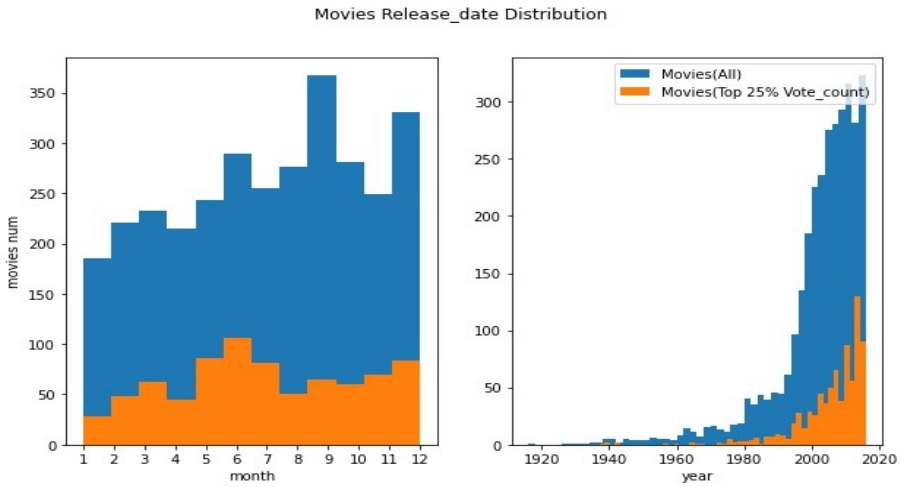


图 7 不同年月发布电影的次数

由图 7 可知橙色的直方图为高评分电影的发布数量随时间（年或月）的分布，蓝色的直方图为在全部电影中电影发布的数量随时间（年或月）的分布。可以看到，每年的 5-7 月份，高票房电影占总电影的比例较高，尤其是 6 月；从年份来看，当电影发布总数较少时，高评分的电影数量较多，且从 2000 年以来呈现逐年波动的趋势，偶数年份高评分电影的占比较高。因此，基于以上分析，电影发布时间会影响电影的票房，且电影适合在偶数年的 5-7 月发布。

production_companies: 不同的电影制片公司（production_companies）产出的电影质量不同，因此经排序获得以下条形图：

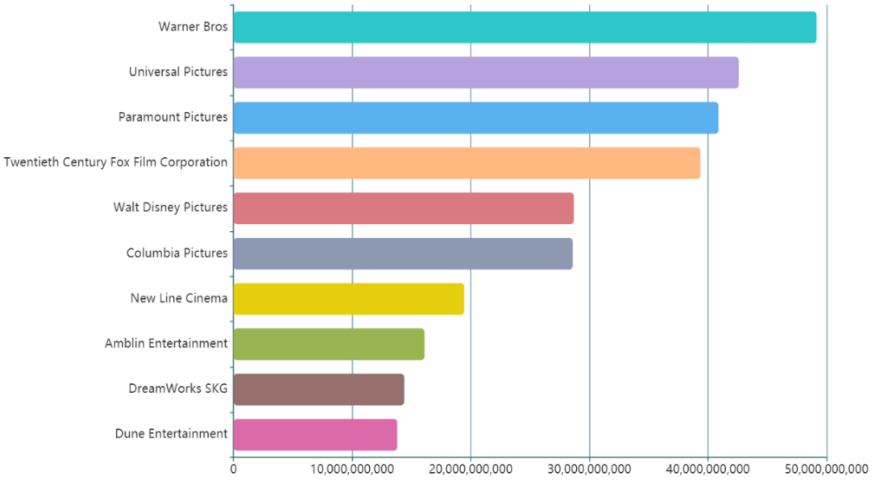


图 8 排名前 10 的电影制片公司

因此，根据排行可以分析，建议选择华纳兄弟娱乐公司、环球影业、派拉蒙影业公司等大牌电影制片厂，可以获取更高的票房。

cast&crew:演职人员和导演在电影中起到非常关键的作用，经验丰富的导演和演技一流、流量颇高的演员为电影票房带来的效益是有很大差别的。

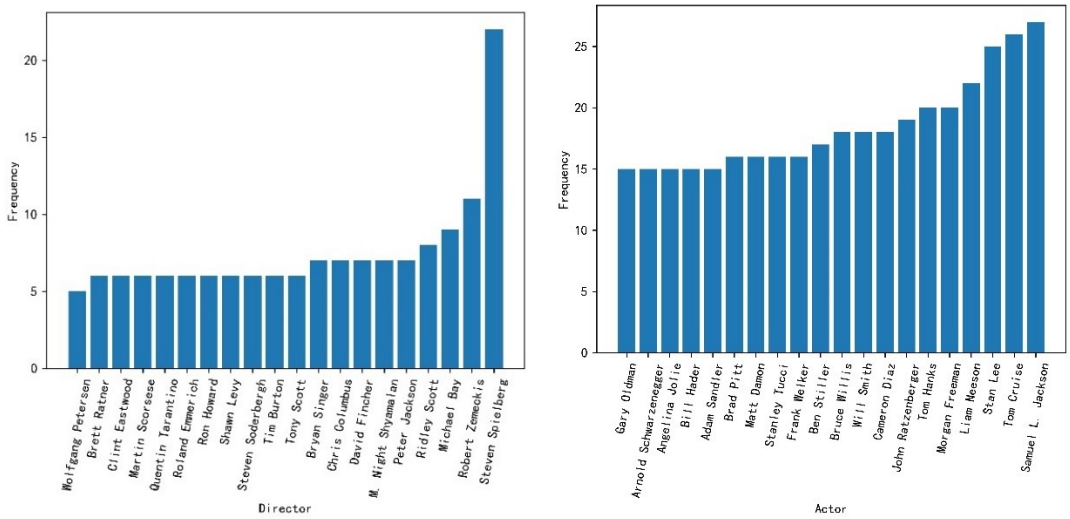


图 9 高票房电影中排名前 20 的导演（左）和演员（右）

可以看到，导演 Steven Spielberg 拍摄的高票房电影最多，演员 Samuel L. Jackson、Tom Cruise、Stan Lee 出演的高票房影片数量较多，因此在商业电影拍摄时，在资金允许的范围内，可以尽可能选择高票房的导演和演员。

5. 1. 2. 2 对变量的相关性分析

budget:电影投资（budget）越多，影片中的妆造、道具、场景、特效等硬性技术能力提高，从而会提高影片本身的艺术价值和商业价值。由投资和票房的散点图可知：

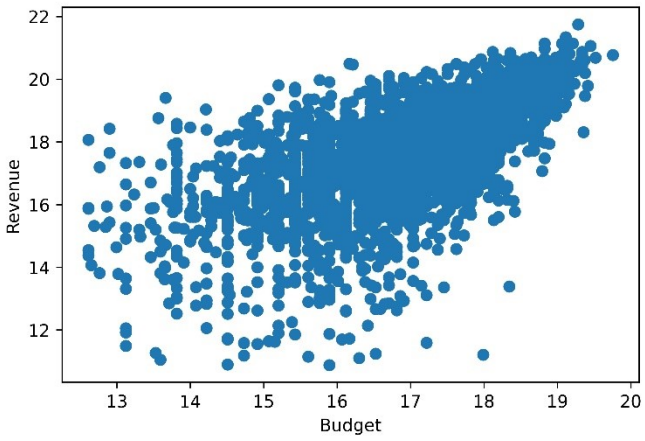


图 10 电影投资与票房的散点图

两个变量呈现较明显的正相关关系,相关系数为 0.640328，由此可见高投资的商业影片质量高，往往收获较高的票房。

popularity：影片的流行程度（popularity）很大程度上决定了影片的热度，进而影响了影片的票房。

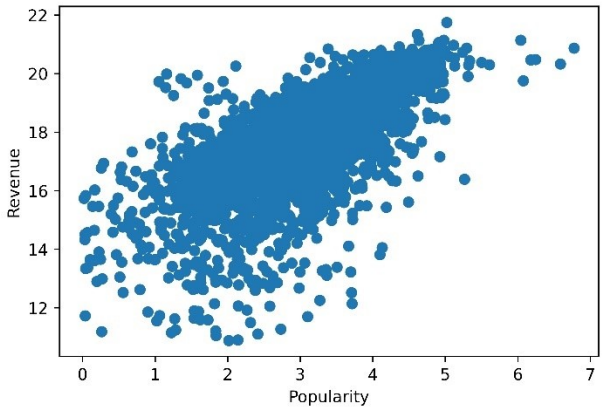


图 11 影片欢迎程度与票房的散点图

由图可以看到，影片的流行程度与电影票房的相关性较大，且相关系数为 0.672502。

runtime：影片时长（runtime）会影响观众的观影感受，由下图可知：

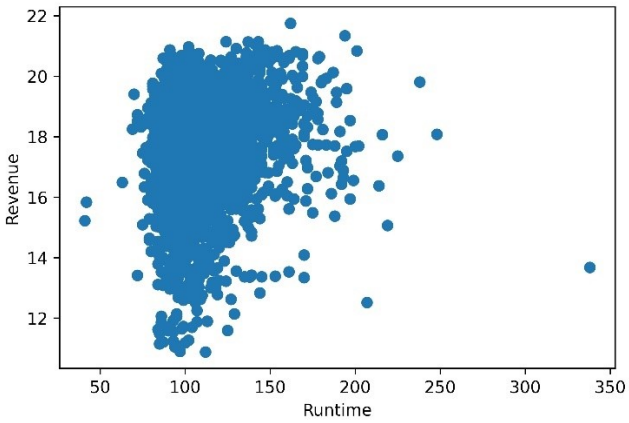


图 12 影片时长与票房的散点图

可以看到，影片时长与票房之间的相关关系较弱，相关系数仅为 0.184652，但是

数据的集中性非常强，可以看到，高票房的影片主要集中在 100-150 分钟之内。

vote_average: 电影评分（va）是指已看电影的观众对电影的主观打分情况，根据这个情况可以反馈出观众对电影的喜好程度。由下图分析：

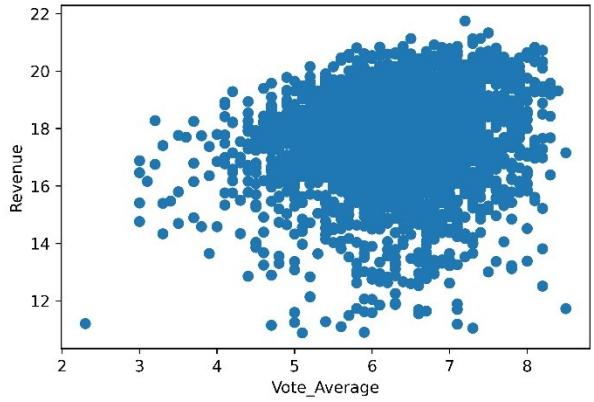


图 13 电影票房与评分的散点图

由图可以看到，大多数的评分集中在 6-7 分，两个变量的相关系数为 0.144759，因此电影的票房与评分之间没有明显的线性相关关系，因此电影评分对电影票房的影响较小，不能作为影响电影票房的因素之一。

vote_count：电影评分人数（vc）是指参与评分的人数，参与评分人数越多，评分的可靠性越高，越能体现电影在观众内的热度。由下图分析：

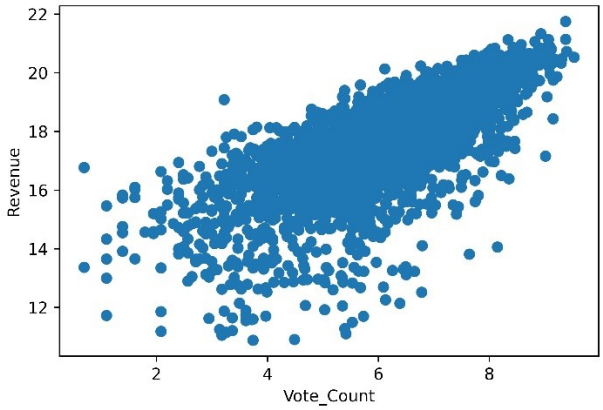


图 14 电影票房与电影评分人数的散点图

电影票房与电影评分人数之间的相关关系非常明显，相关系数为 0.719289，即评分数越多，票房越高，可以将评分人数作为影响电影票房的影响因子。

根据以上的散点图、直方图分析，获得结论如下：

影响电影票房的因素主要有十个，分别为：演员、导演、电影关键词、影片种类、影片发布时间，电影预算、电影流行程度，影片时长、电影制片厂、评分数。

当电影发布时间集中于偶数年的 5-7 月份、影片时长为 100-150 分钟、投资越大、热度越高、评分人数越多且以具有英雄主义和理想主义等美国元素的动作冒险片或戏剧性喜剧片时，电影的商业价值越高，电影票房越好。

5.2 基于神经网络的电影评分预测模型

5.2.1 神经网络基本原理

BP 神经网络的学习规则是使用最速下降法，通过反向传播来不断调整网络的权值和阈值，利用前向传播最后输出的结果来计算误差的偏导数，再后传递误差，用偏导

数和前面的隐藏层进行加权求和，利用每个节点求出的偏导数更新权重，使网络的误差平方和最小。基本的神经网络的拓扑结构主要有三个层：输入层、隐藏层、输出层。

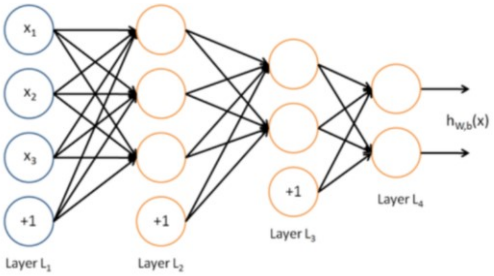


图 15 神经网络的拓扑结构

由问题一分析可知，影响电影评分或评分数的数值类数据有：电影预算（budget）、电影流行程度（popularity）、电影票房（revenue）和 20 种 0-1 数据型的电影类型。由于评分和评分数是两个独立的变量，因此本文将分别建立两个 BP 预测模型，且输出无论是评分还是评分数，神经网络均为 23 输入 1 输出模型，隐藏层为 1 层，隐藏层中设置 512 个神经元，以获取更好的训练效果。

BP 神经网络通常采用 Sigmoid 可微函数和线性函数作为网络的激励函数。本文选择 Sigmoid 作激励函数，数值范围在 0~1 之间，因此在训练之前需要将数据进行标准化处理，激活函数 φ (x)：

$$\varphi(x)=\frac{1}{1+e^{(-\alpha x)}}$$

其中，α 表示一个常数，用来控制输出的斜率。

为更加直观地对本文创建的神经网络模型进行分析，本文将创建神经网络模型如下：

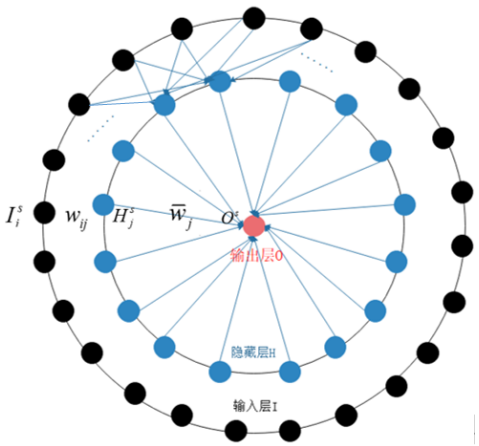


图 16 评分或评分数的神经网络模型

由图所知，I 表示输入层，H 表示隐藏层，O 表示输出层。假设输入 s 个学习样本，s=1,2, …。当将第 s 个样本的数据输入网络时，相应的输出值记为 O^s，隐藏层单元为 $H_j^s(j=1,2,\ldots,512)$，输入层单元记为 $I_i^s(i=1,2,\ldots,23)$。在这一约定下，从输入层到隐藏层的权记为 w_{ij}，从隐藏层到输出层的权记为 $\overline{w}_j$。

则对样本 s 而言，隐单元 j 的输入为：

$$h_j^s = \sum_{i=1}^{23} w_{ij} I_i^s$$

相应的隐含层输出为：

$$H_j^s = \varphi(h_j^s) = \varphi(\sum_{i=1}^{23} w_{ij} I_i^s)$$

因此，输出单元 i 的输入为：

$$\bar{h}^s = \sum_{j=1}^{512} w_j H_j^s = \sum_{j=1}^{512} w_j \varphi(\sum_{i=1}^{23} w_{ij} I_i^s)$$

则网络的输出指为：

$$O^s = \varphi(\bar{h}^s) = \varphi(\sum_{j=1}^{512} w_j H_j^s) = \varphi(\sum_{j=1}^{512} w_j \varphi_1(\sum_{i=1}^{23} w_{ij} I_j^s))$$

假设对于样本的理想输出为 T^s，误差 e 为：

$$e = \frac{1}{2}\sum_s (T^s - O^s)^2 = \frac{1}{2}\sum_s [T^s - \varphi(\sum_{j=1}^{512} w_j \varphi(\sum_{i=1}^{23} w_{ij} I_i^s))]^2$$

误差 e 表示在一组给定的权下，实际输出与理想输出的差异。由上式可知，e 是一个连续可微的非线性函数，为求得极小值与极小值点，采用向后传播算法，对权值迭代更新，可以获得在一定精度内满足要求的权重值序列，并获得最优值。设学习效率为 η，则对应更新权值的公式推导如下：

$$w_{ij} = w_{ij} + \eta H_j^s (1 - H_j^s) \bar{w}_j e$$

$$\bar{w}_j = \bar{w}_j + \eta H_j^s e$$

根据上述推导和分析，则神经网络的算法流程如下：

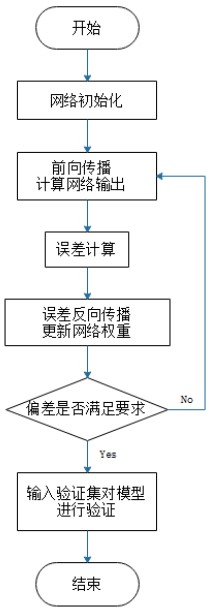


图 17 神经网络的算法流程图

5.2.2 模型求解

5.2.2.1 对电影评分数（vc）的预测

训练数据前，首先要对数据进行预处理操作。与问题一类型，对于预算、流行度和票房三个变量，首先使用对数对其进行缩放，再根据 3 σ 原则剔除异常数据；对于电影类型而言，将 20 种电影类型以 onehot 编码的形式体现，完成对数据的预处理操作。

由于缺少测试集数据，本文将采用 K 折交叉验证用于模型调优，找到使得模型泛化性能最优的超参值，同时也确保每次迭代过程中每个样本点只有一次被划入训练集或验证集的机会[1]。因此，本文采用十折交叉验证，将预处理后的数据分为十等份，一份作为验证集，九份作为测试集，建立 23 输入 1 输出的 BP 神经网络模型（如图 18），训练获取的均方误差曲线如下图所示：

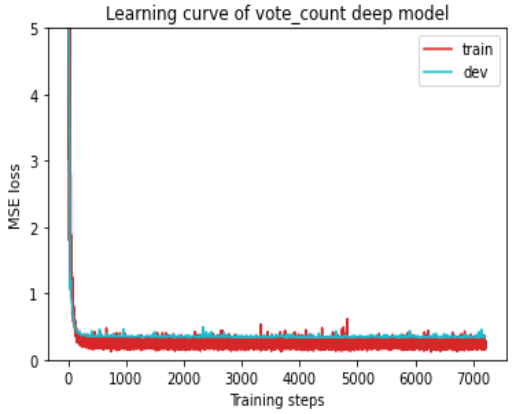


图 18 评分数的均方误差曲线图

红色曲线代表训练集数据的均方误差曲线，蓝色曲线代表验证集的均方误差曲线，两者呈 9:1 的关系，且随着训练次数的增加，均方误差逐渐减小直至收敛，则训练效果，好；蓝色曲线可以反映机器学习的泛化能力，由图可知，模型的泛化性能较高，可以用于预测其他电影的评分数。

因此，将测试集数据输入模型，对 1000 部电影的评分数进行预测，如下所示：

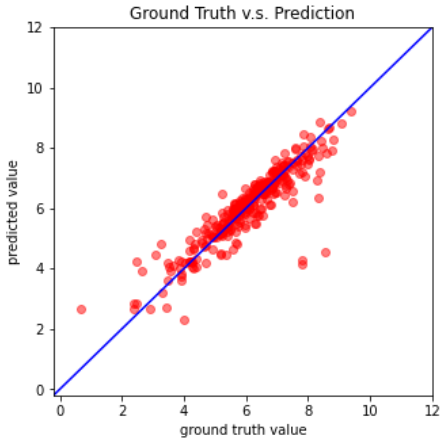


图 19 评分数的预测图

红色的散点为测试集数据在模型中的预测结果，可以看到预测结果均匀地分布在真实有效值的两侧，预测效果好，且预测数据已保存于附件 tmdb_1000_predicted.csv 中。

5.2.2.1 对电影评分（va）的预测

与对评分数的预测步骤相同，对数据进行预处理后，将数据采用 K 折交叉验证的方法分为测试集和验证集，并输入模型中进行训练，获得如下结果：

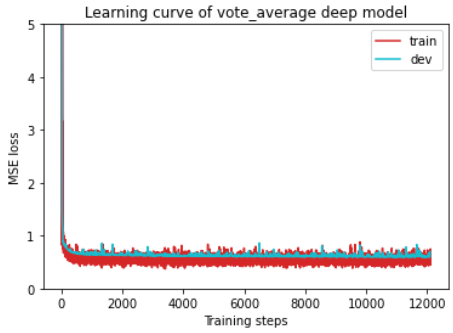


图 20 评分的均方误差曲线图

由图可以看到，蓝色曲线的尾部有微小波动，但趋向于 0，拟合效果较好，可以一定程度上对电影的评分进行预测。

将测试集数据输入模型，对 1000 部电影的评分进行预测，如下所示：

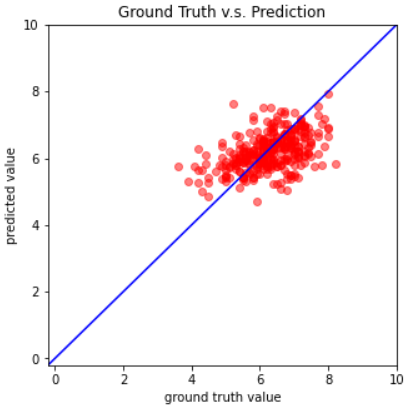


图 21 评分的预测图

红色的散点为测试集数据在模型中的预测结果，由于模型最终没有实现完全收敛，因此预测的结果与真实的有效值之间有一定的差距，但是在现有的数据下，可以一定程度上对电影的评分进行预测，且预测数据已保存于附件 tmdb_1000_predicted.csv 中。

5.3 基于决策树的电影赚钱决策模型

5.3.1 决策树的模型建立

为获得拍摄电影最赚钱的方案，我们将电影按照票房大小分为 1,2,3,4,5 五类。设样本集合为 D，第 m 类样本所占比例为 p_m，m=1,2,3,4,5，离散的特征属性为 a，每一个离散的特征属性中有 v 个可能的取值。基于以上假设，以信息增益率来划分属性的决策树模型建立过程[2]如下：

（1） 计算类别信息熵。

$$Ent(D) = -\sum_{m=1}^{n} p_m \log_2 p_m$$

（2）信息增益

$$Gain(D,a)=Ent(D)-\sum_{v=1}V\frac{|D_v|}{|D|}Ent(D)$$

（3）信息固有值

$$IV(a)=-\sum_{v=1}\frac{D_v}{D}\log_2\frac{D_v}{D}$$

（4）信息增益率

$$Gain\_ratio(D,a)=\frac{Gain(D,a)}{IV(a)}$$

根据上述公式，以信息增益率来划分属性的 CRT 算法流程[3]如下：

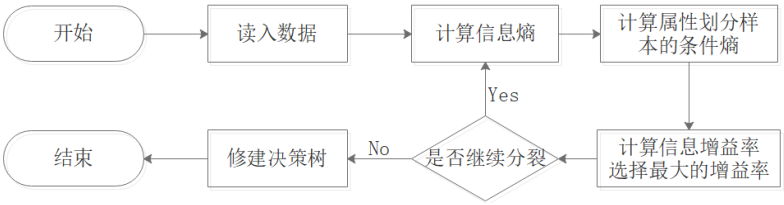


图 22 CRT 算法流程图

5.3.2 模型求解

首先将数据进行预处理，将有效数据集 D 按照电影票房的大小分为 5 类，如下:

电影类别	样本数
4	1278
3	964
5	453
2	330
1	131

表 2 电影分类

根据问题一的分析，对电影票房的数据影响主要为电影预算、流行度和电影类型（onehot 编码），共 22 维度的特征属性。将表 2 中的数据使用 CRT 算法进行求解，获取的决策树如图 24 所示，分析结果如下：

表 3 决策树的自变量重要性

自变量	重要性	正态化重要性
budget	.135	100.0%
popularity	.112	82.5%
genres18	.003	2.4%
genres8	.002	1.7%
genres11	.002	1.6%
genres15	.001	0.8%
genres4	.001	0.5%
genres14	.000	0.4%

由表 3 可知，结合问题一分析得到的结果，对于电影商而言，可以根据自变量的重要性依次进行决策。

为赚取更多的票房，电影拍摄时首先需要考虑预算的投入，在资金充裕的情况下，

投入电影制作的预算越多，票房越高；其次，考虑电影的热度，应结合时代背景，迎合大众口味进行拍摄；再考虑电影的类型，优先考虑是否拍摄 genres18（Foreign）和 genres8（Western）编码的电影，其次考虑拍摄 genres11（Documentary）、genres15（Romance）、genres4（Horror）、genres14（Music）类型。

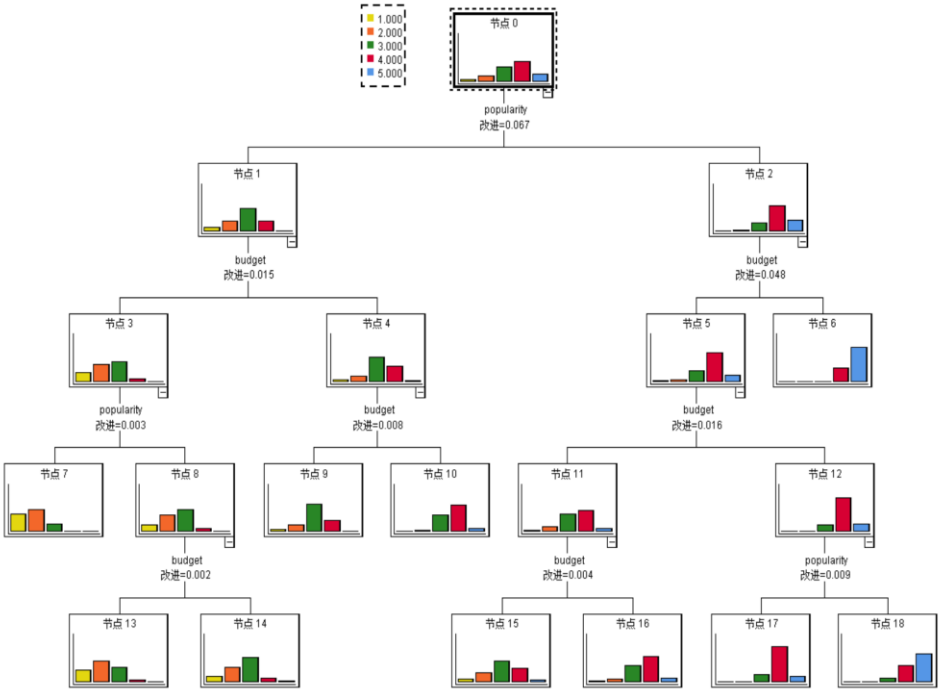


图 23 “电影赚钱”决策树

5.3.3 模型验证

为验证决策树模型是否可以提供合理的赚钱决策方案，使用 K 折交叉验证，将数据集 *D* 分为十份，一份用于验证，九份用于测试。验证结果如下：

表 4 对电影票房的预测结果

实测	1.0	2.0	3.0	4.0	5.0	正确百分比
1.0	0	51	73	6	1	0.0%
2.0	0	80	220	30	0	24.2%
3.0	0	43	606	310	5	62.9%
4.0	0	3	233	919	123	71.9%
5.0	0	0	13	138	302	66.7%
总体百分比	0.0%	5.6%	36.6%	44.5%	13.7%	60.4%

由表可知，票房最高的类别 5 的预测正确率为 60.4%，票房次高的类别 4 的预测正确率为 71.9%。据以上数据分析可知，该模型可以对电影票房进行合理预测，因而可以为电影商创作电影提供有力的理论支撑。

六 模型的评价

6.1 模型的优点

（1）使用 K 折交叉验证，保证每个样本都参与训练或验证，降低泛化误差，一定程度上减小过拟合，从有限的数据中获得更多的有效信息。

（2）神经网络训练时，隐藏层选择 512 个神经元进行训练，神经元越多，训练效

果越好，因而预测结果与真实有效值更加贴近。

（3）神经网络具有普适性函数趋近的特点，它可以以任意精度逼近任何函数，在处理问题时显得灵活有效，可以解决传统数据模型中对复杂数据估计不当的问题。

（4）模型的预测结果与真实有效值差别较小，准确性较高，对电影评分和评分数的预测有较强的参考价值，可以将该模型用于其他变量或者其他领域，对某个指标进行合理化预测，应用价值较大。

6.2 模型的缺点

（1）对于字符串类型的数据处理相对简单，无法直接求得该类数据与电影票房的关系，寻找影响电影票房的因素时较为主观。

（2）对电影评分进行预测时，蓝色的验证集均方误差曲线仍有微小波动，因此与电影评分数相比，对评分的预测效果较差。

6.3 模型的推广

针对缺点一，可以参考郑等[4]文献中提出的“影响因子归一量化”，将字符串类型的数据进行深度挖掘与分析，从数量、时间、空间等维度对数据进行再分析，化为更多可以量化或用数学符号表示的影响因子，建立可量化指标：影响力指数。这个修改方法可以解决对字符串的分析问题，求得全部变量与票房的相关关系后，选择相关性较大的变量，使用主成分分析对其降维处理，最终选择出的影响因素更具有说服力。但是上述方法十分复杂，数据处理过程较为困难，需要大量时间对数据进行处理分析，可操作性较低。

除对商业电影产业的分析外，分析与预测模型可以在多个领域都有应用，如房地产行业、旅游行业、航空公司航班预测、银行贷款预测等，应用范围广，具有较高的参考价值和意义。

七 参考文献

[1] K 折交叉验证，https://blog.csdn.net/tianguiyuyu/article/details/80697223,2018

[2] https://blog.csdn.net/qq_30031221/article/details/108512350,2020

[3] 李振兴. 机器学习在电影票房预测中的应用研究[D]. 陕西:西安石油学院,2020.

[4] 郑坚,周尚波. 基于神经网络的电影票房预测建模[J]. 计算机应用,2014

附录

附录一 数据预处理

```
"""# 数据预处理(movies)
"""
from pandas import DataFrame
movies_genre = DataFrame()
movies_keyword = pd.DataFrame()
movies_company = DataFrame()
movies_country = DataFrame()
movies_language = DataFrame()

for row in data_tr.itertuples():
    df_genre = pd.read_json(row.genres, orient='records')
    df_keyword = pd.read_json(row.keywords, orient='records')
    df_company = pd.read_json(row.production_companies, orient='records')
    df_country = pd.read_json(row.production_countries, orient='records')
    df_language = pd.read_json(row.spoken_languages, orient='records')
    df_genre['movie_id'] = row.Index
    df_keyword['movie_id'] = row.Index
    df_company['movie_id'] = row.Index
    df_country['movie_id'] = row.Index
    df_language['movie_id'] = row.Index
    movies_genre = pd.concat([movies_genre, df_genre],  ignore_index=True, sort=False)
    movies_keyword = pd.concat([movies_keyword, df_keyword],  ignore_index=True, sort=False)
    movies_company = pd.concat([movies_company, df_company],  ignore_index=True, sort=False)
    movies_country = pd.concat([movies_country, df_country],  ignore_index=True, sort=False)
    movies_language = pd.concat([movies_language, df_language],  ignore_index=True, sort=False)

movies_keyword['id'] = movies_keyword['id'].astype(np.int64)
movies_genre['id'] = movies_genre['id'].astype(np.int64)
movies_company['id'] = movies_company['id'].astype(np.int64)

# 得到对应的唯一值表
genres = movies_genre.drop('movie_id', axis=1).drop_duplicates().set_index('id')
keywords = movies_keyword.drop('movie_id', axis=1).drop_duplicates().set_index('id')
companies = movies_company.drop('movie_id', axis=1).drop_duplicates().set_index('id')
countries = movies_country.drop('movie_id', axis=1).drop_duplicates().set_index('iso_3166_1')
languages = movies_language.drop('movie_id', axis=1).drop_duplicates().set_index('iso_639_1')

movies_copy1 = data_tr.copy()

#data_tr=data_tr.drop(['genres', 'keywords', 'production_companies', 'production_countries',
'spoken_languages'], axis=1)

data_tr.head()

movies = data_tr
movies_copy2 = movies.copy()

movies = movies.drop(['original_title', 'overview', 'tagline', 'status'], axis=1)

movies.head()

# 将列重排
movies = movies.reindex(columns=['title', 'runtime', 'original_language', 'release_date', 'popularity',
```

```
                        'vote_average', 'vote_count', 'budget', 'revenue'])

"""数据描述

"""

movies.describe()
movies_genre = pd.merge(movies_genre, movies, left_on='movie_id', right_index=True, how='left')
movies_keyword

"""# 删除无效数据"""

# 删除无效数据
dcol = ['homepage', 'id', 'overview', 'original_title', 'tagline', 'production_companies', 'production_countries',
'spoken_languages', 'status']
data_tr = data_tr.drop(dcol, axis=1)
data_tt = data_tt.drop(dcol, axis=1)

# 删除有空值的行
data_tr.dropna(subset=['runtime'], inplace = True)
data_tr.dropna(subset=['release_date'], inplace = True)

# A glimpse at what's missing
for column in data_tt.columns:
    null_count = data_tt[column].isnull().sum()
    print(f'{column} — {round((null_count/data_tt.shape[0]) * 100)}% MISSING and total missing is:
{null_count}')

pp.pprint(data_tr.info())

"""## genres、keyword 特征处理"""

# pandas 格式转 list
list = data_tr.values.tolist()
# list[0]
# list = np.array(list) # 转 np array

def str_to_idlist(col):
    # 提取第 i 列 的 id_list
    temp = []
    for row in range(len(list)):
        # print("-----"+str(row)+"-------")
        # print(list[row][col])
        # print(list[row][col])
        if(list[row][col] != '[]'):
            json_str = list[row][col][1:-1]
            json_str = json_str.split("}, ");
            for i in range(len(json_str)-1):
                json_str[i] = json_str[i]+'}'
                json_str[i] = eval(json_str[i])
            json_str[-1] = eval(json_str[-1])
            dic_list = json_str

            id_list = []
            for i in range(len(json_str)):
                id_list.append(json_str[i]['id'])
```

```
        else:
            id_list = []

        temp.append(id_list)
    return temp

def get_namelist(col):
    # 提取 name
    temp_name = {}
    for row in range(len(list)):
        # print("-----"+str(row)+"-------")
        # print(list[row][col])
        # print(list[row][col])
        if(list[row][col] != '[]'):
            json_str = list[row][col][1:-1]  # 将 dic_str 转换为 dic_list
            json_str = json_str.split("}, ");
            for i in range(len(json_str)-1):
                json_str[i] = json_str[i]+'}'
                json_str[i] = eval(json_str[i])
            json_str[-1] = eval(json_str[-1])
            for i in range(len(json_str)):
                # print(json_str[i]['id'])
                # print(json_str[i]['name'])
                temp_name[json_str[i]['id']] = json_str[i]['name']
    return temp_name

# 提取 key
genres_list = str_to_idlist(1)
# keyword_list = str_to_idlist(2)
# prodc_list = str_to_idlist(6)

# keyword_list

# 分析 特征种类的数量
def get_num(list):
    list_flat = [i for item in list for i in item]
    list_flat.sort()
    num = list_flat[-1]
    return num
# [get_num(genres_list),get_num(keyword_list), get_num(prodc_list)]
# [get_num(genres_list),get_num(keyword_list), get_num(prodc_list)] = [10770, 238222, 95063]

# 提取电影类型
genres_name_list = get_namelist(1)
# keyword_name_list = get_namelist(2)
# prodc_name_list = get_namelist(6)

genres_name_list_k = sorted(genres_name_list.items(), key=lambda x:x[0])
# 按 key 排序，lambda x:x[0]表示取要排序的第一个元素排序
len(genres_name_list_k)
# 共 20 种 genres_name
# keyword_name_list_k = sorted(keyword_name_list.items(), key=lambda x:x[0])
# 按 key 排序，lambda x:x[0]表示取要排序的第一个元素排序
# len(keyword_name_list_k)
# 共 9813 种 keyword_name
```

```
# prodc_name_list_k= sorted(prodc_name_list.items(), key=lambda x:x[0])
 # 按 key 排序，lambda x:x[0]表示取要排序的第一个元素排序
# len(prodc_name_list_k)
# 共 5047 种 prodc_name
# 源 id -> 0~len()
def id_num_tran(genres_list, genres_name_list_k):
    map = []
    for i in genres_name_list_k:
        map.append(i[0])
    # print(map)
    map_1 = {}
    for i in range(len(map)):
        map_1[map[i]] = i
    # print(map_1)
    # 改变电影特征编号
    genres_list_temp = genres_list[:]
    for i in range(len(genres_list)):
        for j in range(len(genres_list[i])):
            # print(genres_list[i][j])
            if genres_list_temp[i][j] == 10770: # test 集
                continue
            genres_list_temp[i][j] = map_1[genres_list[i][j]]
    return genres_list_temp

# 类型列表
ge_list = []
for i in genres_name_list_k:
    ge_list.append(i[1])

# keyword_name_list_k[-1]
# keyword_list_tran

# 归一化后的 key 特征
# prodc_list_tran = []
genres_list_tran = id_num_tran(genres_list, genres_name_list_k)
# keyword_list_tran = id_num_tran(keyword_list, keyword_name_list_k)
# prodc_list_tran = id_num_tran(prodc_list, prodc_name_list_k)

genres_name_list_k

# keyword_name_list_k
# keyword_list_tran

# 转换为 onehot 格式
# genres_list_tran
genres_onehot = []
for i in range(len(genres_list_tran)):
    temp = [0]*20
    # print(len(genres_list_tran[i]))
    for j in genres_list_tran[i]:
        # print(j)
        if j == 10770:
            continue
        temp[j] = 1;
    genres_onehot.append(temp)
```

```
# 插入到 pandas 数据中
genres_onehot_np = np.array(genres_onehot)

for i in range(20):
    # print('genres' + str(i))
    data_tr.insert(data_tr.shape[1], 'genres' + str(i), genres_onehot_np[:,i])

# keyword id 插入
# keyword_list_np = np.array(keyword_list)
# data_tr.insert(data_tr.shape[1], 'keywords_id', keyword_list_np)

pp.pprint(data_tr.info()) #看每列数据类型和大小

# 删除 genres 和 keywords
dcol = ['genres','keywords']
data_tr = data_tr.drop(dcol, axis=1)

data_tr.head(3)

"""## 处理 title 特征
- list[4] orginal_title
- list[14] title
"""

# orginal_title = [x[4] for x in list]
# title = [x[14] for x in list]

# cnt = 0
# st = [0]*len(title)
# for i in range(len(title)):
#     if title[i] != orginal_title[i]:
#         st[i] = 1
#         cnt = cnt+1
# cnt = 261

# for i in range(len(title)):
#     if(st[i] == 1):
#         print(title[i]+" "+orginal_title[i])

# 删除 orginal title 特征

"""## 处理 language 特征"""

data_tr['original_language'].value_counts()

"""# budget 特征缩放"""

data_tr.hist('budget',bins=20)

data_tr["budget"] = data_tr["budget"].apply(np.log1p)
data_tt["budget"] = data_tt["budget"].apply(np.log1p)

for x in data_tr.index:
# #   if data_tr.loc[x, "revenue"] == 0 && data_tr.loc[x, "vote_count"] < threshold:
    if data_tr.loc[x, "budget"] == 0:
        data_tr.drop(x, inplace = True)
```

```
data_tr.hist('budget',bins=20)

sum(data_tr["budget"] > 0)

# 3 sigma
data_tr_mean = data_tr.mean()
data_tr_std = data_tr.std()
threshold_budget_1 = data_tr_mean['budget'] - 3 *data_tr_std['budget']
threshold_budget_2 = data_tr_mean['budget'] + 3 *data_tr_std['budget']
print(threshold_budget_1)
print(threshold_budget_2)


for x in data_tr.index:
    if data_tr.loc[x, "budget"] < threshold_budget_1 or data_tr.loc[x, "budget"] >threshold_budget_2:
        data_tr.drop(x, inplace = True)

data_tr['budget'] = (data_tr['budget'] - data_tr['budget'].mean()) / (data_tr['budget'].std())
data_tt['budget'] = (data_tt['budget'] - data_tr['budget'].mean()) / (data_tr['budget'].std())

data_tr.hist('budget',bins=20)

plt.figure(dpi=600)
plt.scatter(data_tr.loc[:,  'budget'], data_tr.loc[:, 'revenue'])
plt.ylabel("Vote_count")
plt.xlabel("Budget (processed)")

plt.savefig('./budget.jpg',  bbox_inches = 'tight')
# pearson 相关系数为 0.640328

pea = data_tr.corr(method='pearson')[0:6]

pea[5:6]

"""# popularity 特征缩放"""

data_tr.hist('popularity',bins=20)

data_tr['popularity'] = data_tr['popularity'].apply(np.log1p)
data_tt['popularity'] = data_tt['popularity'].apply(np.log1p)
data_tr['popularity'] = (data_tr['popularity'] - data_tr['popularity'].mean()) / (data_tr['budget'].std())
data_tt['popularity'] = (data_tt['popularity'] - data_tr['popularity'].mean()) / (data_tr['popularity'].std())

data_tr.hist('popularity',bins=20)

plt.figure(dpi=600)
plt.scatter(data_tr.loc[:,  'popularity'], data_tr.loc[:, 'vote_count'])
plt.ylabel("Vote_count")
plt.xlabel("Popularity (processed)")
plt.savefig('./Popularity.jpg',  bbox_inches = 'tight')

"""# revenue 特征缩放"""

data_tr.hist('revenue',bins=20)

data_tr["revenue"] = data_tr["revenue"].apply(np.log1p)
```

```
data_tt["revenue"] = data_tt["revenue"].apply(np.log1p)

for x in data_tr.index:
# #   if data_tr.loc[x, "revenue"] == 0 && data_tr.loc[x, "vote_count"] < threshold:
    if data_tr.loc[x, "revenue"] == 0:
        data_tr.drop(x, inplace = True)

data_tr.hist('revenue',bins=20)

sum(data_tr["revenue"] > 0)

# 计算 threshold_vote, threshold_revenue
data_tr_mean = data_tr.mean()
data_tr_std = data_tr.std()

# threshold_vote_1 = data_tr_mean['vote_count'] - 3 *data_tr_std['vote_count']
# threshold_vote_2 = data_tr_mean['v'] + 3 *data_tr_std['vote_count']
# print(threshold_vote_1)
# print(threshold_vote_2)

threshold_revenue_1 = data_tr_mean['revenue'] - 3 *data_tr_std['revenue']
threshold_revenue_2 = data_tr_mean['revenue'] + 3 *data_tr_std['revenue']
print(threshold_revenue_1)
print(threshold_revenue_2)

for x in data_tr.index:
# #   if data_tr.loc[x, "revenue"] == 0 && data_tr.loc[x, "vote_count"] < threshold:
    if data_tr.loc[x, "revenue"] < threshold_revenue_1 or data_tr.loc[x, "revenue"] >threshold_revenue_2:
        data_tr.drop(x, inplace = True)
data_tr['revenue'] = (data_tr['revenue'] - data_tr['revenue'].mean()) / (data_tr['revenue'].std())
data_tt['revenue'] = (data_tt['revenue'] - data_tr['revenue'].mean()) / (data_tr['revenue'].std())

data_tr.hist('revenue',bins=5)

plt.scatter(data_tr.loc[:,  'revenue'], data_tr.loc[:, 'vote_count'])

"""# 处理 crew cast 数据"""

pd.set_option('display.max_colwidth',500)
print(data_tr['cast'][0:1])
print(data_tr['crew'][0:1])

# crew cast 转换
from pandas import DataFrame
movies_crew = DataFrame()
movies_cast = pd.DataFrame()

cos_list = [] # 电影 对应演员
cre_list = [] # 电影
for row in data_tr.itertuples():
    df_cast = pd.read_json(row.cast, orient='records')
    df_crew = pd.read_json(row.crew, orient='records')
    # pd.set_option('display.max_colwidth',500)

    cast_list = df_cast.values.tolist()
    crew_list = df_crew.values.tolist()
```

```
    # "gender": 2, "id": 3129, "name": "Tim Roth"
    cast_list = [cast_list[i][-4:-1] for i in range(len(cast_list))]
    # "gender": 1, "id": 3110, "job": "Director", "name": "Allison Anders"
    crew_list = [crew_list[i][-4:] for i in range(len(crew_list))]

    cos_list.append(cast_list)
    cre_list.append(crew_list)

cos_list

# 插入 data set 中
cos_list_np = np.array(cos_list)
data_tr.insert(data_tr.shape[1], 'cos_list', cos_list_np)

cre_list_np = np.array(cre_list)
data_tr.insert(data_tr.shape[1], 'cre_list', cre_list_np)

# 删除 'cast','crew'
dcol = ['cast','crew']
data_tr = data_tr.drop(dcol, axis=1)

# 所有演员名单
cos_name_list = [j  for i in cos_list_np for j in i]
# del list
list2=list(set([tuple(t) for t in cos_name_list])) # 去重
cos_name_list = sorted(list2,key=(lambda x:x[1])) # 按 ID 排序

# 所有职工名单
cre_name_list = [j  for i in cre_list_np for j in i]
# del list
list2=list(set([tuple(t) for t in cre_name_list])) # 去重
cre_name_list = sorted(list2,key=(lambda x:x[2])) # 按 ID 排序

cre_name_list

# movies_crew

# 删除 revenue 和 vote_count < threshold 的数据
# threshold = 20
# for x in data_tr.index:
# #   if data_tr.loc[x, "revenue"] == 0 && data_tr.loc[x, "vote_count"] < threshold:
#     if data_tr.loc[x, "revenue"] == 0 or data_tr.loc[x, "vote_count"] == 0:
#         data_tr.drop(x, inplace = True)

# cnt = 0
# for x in data_tr.index:
#   if data_tr.loc[x, "vote_count"] < 20:
#       cnt = cnt+1
# print(cnt)

"""# 电影时长"""

data_tr.hist('runtime',bins=20)

plt.figure(dpi=600)
plt.scatter(data_tr.loc[:, 'runtime'], data_tr.loc[:, 'vote_count'])
```

```
plt.ylabel("Vote_count")
plt.xlabel("Runtime")
plt.savefig('./runtime.jpg',  bbox_inches = 'tight')

"""# vote_average 特征"""

data_tr.hist('vote_average',bins=20)

plt.scatter(data_tr.loc[:,  'vote_average'], data_tr.loc[:, 'revenue'])

data_tr.iloc[:, :].corr()[2:3]

"""#vote_count
"""

data_tr.hist('vote_count',bins=20)

data_tr['vote_count'] = data_tr['vote_count'].apply(np.log1p)

data_tr.hist('vote_count',bins=20)

data_tr['vote_count'].value_counts()

"""# release_date 特征分析"""

data_tr['release_date'] = pd.to_datetime(data_tr['release_date'], infer_datetime_format=True)

# data_tr.dtypes
data_tr.head(3)

# 全部电影 时间的分布
# plt.figure(figsize=(10, 6))
plt.figure(dpi=600)
plt.subplot(1, 2, 1)
df =  data_tr[['release_date']]
plt.hist([t.month for t in df['release_date']], bins = 12)
my_x_ticks = np.arange(0, 13)
plt.xticks(my_x_ticks)
plt.xlabel("month")
plt.ylabel("movies num")

plt.subplot(1, 2, 2)
plt.hist([t.year for t in df['release_date']], bins = 50, label="Movies(All)")
plt.xlabel("year")
# plt.ylabel("movies num")
# data_tr.hist('release_date')
# df.groupby(df["release_date"].dt.year).count().plot(kind="bar")
# df.groupby([df["release_date"].dt.year, df["release_date"].dt.month]).count().plot(kind="bar")
plt.suptitle('Movies Release_date Distribution')
# plt.savefig('./time1.jpg',  bbox_inches = 'tight')

plt.subplot(1, 2, 1)
df =  data_front[['release_date']]

my_x_ticks = np.arange(0, 13)
plt.xticks(my_x_ticks)
plt.hist([t.month for t in df['release_date']], bins = 12)
```

```
plt.xlabel("month")
plt.ylabel("movies num")

plt.subplot(1, 2, 2)
plt.hist([t.year for t in df['release_date']], bins = 50, label="Movies(Top 25% Revenue)")
plt.xlabel("year")
plt.legend(loc='upper right')
# plt.ylabel("movies num")
plt.savefig('./time3.jpg',  bbox_inches = 'tight')

# 高票房电影时间分布
fig = plt.figure(dpi=600)
plt.subplot(1, 2, 1)
df =  data_front[['release_date']]

my_x_ticks = np.arange(0, 13)
plt.xticks(my_x_ticks)
plt.hist([t.month for t in df['release_date']], bins = 12)
plt.xlabel("month")
plt.ylabel("movies num")

plt.subplot(1, 2, 2)
plt.hist([t.year for t in df['release_date']], bins = 50)
plt.xlabel("year")
# plt.ylabel("movies num")
plt.suptitle('Movies(Top 25% Vote_count) Release_date Distribution')

plt.savefig('./time2.jpg',  bbox_inches = 'tight')

plt.scatter(data_tr.loc[:, 'release_date'], data_tr.loc[:, 'vote_count'])

[t.year  for t in df['release  date']]
```

附录二 数据可视化

```
"""# 可视化"""
names = ['budget', 'popularity',  'runtime',  'vote_average',  'vote_count']
for i in range(5):
    fig = plt.figure()
    plt.figure(dpi=600)
    plt.scatter(data_tr.loc[:,  names[i]], data_tr.loc[:, 'revenue'])
    # plt.scatter(data_tr.loc[:,  'popularity'], data_tr.loc[:, 'revenue'])
    # plt.scatter(data_tr.loc[:,  'runtime'], data_tr.loc[:, 'revenue'])
    # plt.scatter(data_tr.loc[:,  'vote_average'], data_tr.loc[:, 'revenue'])
    # plt.scatter(data_tr.loc[:,  'vote_count'], data_tr.loc[:, 'revenue'])
    plt.ylabel("Revenue")
    plt.xlabel(names[i].title())
    # plt.xlabel("Popularity")
    # plt.xlabel("Runtime")
    # plt.xlabel("Vote_average")
    # plt.xlabel("Vote_count")
    plt.savefig('./'+names[i].title()+'.jpg',  bbox_inches = 'tight')

names = ['budget', 'popularity',  'runtime',  'vote_average',  'vote_count']
i = 0
```

```
names[i].title()

# 特征值分布
import seaborn as sns; sns.set()
data_map = data_tr[['budget', 'popularity',  'runtime',  'vote_average',  'vote_count', 'revenue']]
x = sns.PairGrid(data_map)
x = x.map_offdiag(plt.scatter)
x = x.map_diag(plt.hist)

# 相关性矩阵
corrmat = data_map.corr()
fig, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True)
```

附录三 神经网络

```
# PyTorch
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

# For data preprocess
import numpy as np
import csv
import os

# For plotting
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

myseed = 42069  # set a random seed for reproducibility
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(myseed)
torch.manual_seed(myseed)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(myseed)

def get_device():
    ''' Get device (if GPU is available, use GPU) '''
    return 'cuda' if torch.cuda.is_available() else 'cpu'

def plot_learning_curve(loss_record, title=''):
    ''' Plot learning curve of your DNN (train & dev loss) '''
    total_steps = len(loss_record['train'])
    x_1 = range(total_steps)
    x_2 = x_1[::len(loss_record['train']) // len(loss_record['dev'])]
    figure(figsize=(6, 4))
    plt.plot(x_1, loss_record['train'], c='tab:red', label='train')
    plt.plot(x_2, loss_record['dev'], c='tab:cyan', label='dev')
    # plt.ylim(0.0,  0.1007)
    plt.ylim(0.0,  5.)
    plt.xlabel('Training steps')
    plt.ylabel('MSE loss')
    plt.title('Learning curve of {}'.format(title))
    plt.legend()
```

```
    plt.show()


def plot_pred(dv_set, model, device, lim=35., preds=None, targets=None):
    ''' Plot prediction of your DNN '''
    if preds is None or targets is None:
        model.eval()
        preds, targets = [], []
        for x, y in dv_set:
            x, y = x.to(device), y.to(device)
            with torch.no_grad():
                pred = model(x)
                preds.append(pred.detach().cpu())
                targets.append(y.detach().cpu())
        preds = torch.cat(preds, dim=0).numpy()
        targets = torch.cat(targets, dim=0).numpy()

    figure(figsize=(5, 5))
    plt.scatter(targets, preds, c='r', alpha=0.5)
    plt.plot([-0.2, lim], [-0.2, lim], c='b')
    plt.xlim(-0.2, lim)
    plt.ylim(-0.2, lim)
    plt.xlabel('ground truth value')
    plt.ylabel('predicted value')
    plt.title('Ground Truth v.s. Prediction')
    plt.show()

data_tr.info()
for x in data_tr.index:
# if data_tr.loc[x, "revenue"] == 0 && data_tr.loc[x, "vote_count"] < threshold:
    if data_tr.loc[x, "revenue"] < -2:
        data_tr.loc[x, "revenue"] = 1
    elif data_tr.loc[x, "revenue"] < -1:
        data_tr.loc[x, "revenue"] = 2
    elif data_tr.loc[x, "revenue"] < 0:
        data_tr.loc[x, "revenue"] = 3
    elif data_tr.loc[x, "revenue"] < 1:
        data_tr.loc[x, "revenue"] = 4
    else:
        data_tr.loc[x, "revenue"] = 5

data_tr = data_tr.drop(['original_language', 'release_date', 'runtime', 'title'], axis=1)
data_tt = data_tt.drop(['original_language', 'release_date', 'runtime', 'title'], axis=1)
data_tr.to_csv("processed_data.csv")
data_tt.columns

# 将列重排
data_tr = data_tr.reindex(columns=['budget', 'popularity',
       'genres0', 'genres1', 'genres2', 'genres3', 'genres4', 'genres5',
       'genres6', 'genres7', 'genres8', 'genres9', 'genres10', 'genres11',
       'genres12', 'genres13', 'genres14', 'genres15', 'genres16', 'genres17',
       'genres18', 'genres19', 'vote_average', 'vote_count', 'revenue'])
# data_tt = data_tt.reindex(columns=['budget', 'popularity', 'revenue',
#       'genres0', 'genres1', 'genres2', 'genres3', 'genres4', 'genres5',
#       'genres6', 'genres7', 'genres8', 'genres9', 'genres10', 'genres11',
#       'genres12', 'genres13', 'genres14', 'genres15', 'genres16', 'genres17',
#       'genres18', 'genres19'])
```

```
data_tr.columns
data_tt.head(5)
data_tr.head(1)

class MovieDataset(Dataset):
    ''' Dataset for loading and preprocessing the Movie dataset '''
    def __init__(self,
                 path,
                 mode='train',
                 target_only=False):
        self.mode = mode
# Read data into numpy arrays
        data = data_tr.values.tolist()
        # data = data_tt.values.tolist()
        data = np.array(data).astype(float)
        feats = list(range(23))

        if mode == 'test':
            # Testing data
            data = data_tt.values.tolist()
            data = np.array(data).astype(float)
            data = data[:, feats]
            self.data = torch.FloatTensor(data)
        else:
            # Training data (train/dev sets)
            # -1 表示 votecount -2 表示 vote_average
            target = data[:, -1]
            data = data[:, feats]

            # Splitting training data into train & dev sets
            if mode == 'train':
                indices = [i for i in range(len(data)) if i % 10 != 0]
            elif mode == 'dev':
                indices = [i for i in range(len(data)) if i % 10 == 0]
# Convert data into PyTorch tensors
            self.data = torch.FloatTensor(data[indices])
            self.target = torch.FloatTensor(target[indices])
# Normalize features (you may remove this part to see what will happen)
# self.data[:, 40:] = \
#     (self.data[:, 40:] - self.data[:, 40:].mean(dim=0, keepdim=True)) \
#     / self.data[:, 40:].std(dim=0, keepdim=True)

        self.dim = self.data.shape[1]
        print('Finished reading the {} set of Movies Dataset ({} samples found, each dim = {})'
              .format(mode, len(self.data), self.dim))

    def __getitem__(self, index):
        # Returns one sample at a time
        if self.mode in ['train', 'dev']:
            # For training
            return self.data[index], self.target[index]
        else:
            # For testing (no target)
            return self.data[index]

    def __len__(self):
```

```
        # Returns the size of the dataset
        return len(self.data)

# list(range(23))

def prep_dataloader(mode, batch_size, n_jobs=0, target_only=False):
    ''' Generates a dataset, then is put into a dataloader. '''
    dataset = MovieDataset('.',mode=mode, target_only=target_only)  # Construct dataset
    dataloader = DataLoader(
        dataset, batch_size,
        shuffle=(mode == 'train'), drop_last=False,
        num_workers=n_jobs, pin_memory=True)                            # Construct dataloader
    return dataloader

class NeuralNet(nn.Module):
    ''' A simple fully-connected deep neural network '''
    def __init__(self, input_dim):
        super(NeuralNet, self).__init__()

        # Define your neural network here
        self.net = nn.Sequential(
            nn.Linear(input_dim, 512),
            nn.ReLU(),
            # nn.Linear(64, 64),  # change the network structure
            # nn.ReLU(),
            nn.Linear(512, 1)
        )

        # Mean squared error loss
        self.criterion = nn.MSELoss(reduction='mean')

    def forward(self, x):
        ''' Given input of size (batch_size x input_dim), compute output of the network '''
        return self.net(x).squeeze(1)

    def cal_loss(self, pred, target):
        ''' Calculate loss '''
        # TODO: you may implement L1/L2 regularization here
        # L1 regularization
        # regularization_loss = 0
        # for para in self.parameters():
        #     regularization_loss += torch.sum(torch.abs(para))
        # return self.criterion(pred, target) + 0.01 * regularization_loss
        return self.criterion(pred, target)

def train(tr_set, dv_set, model, config, device):
    ''' DNN training '''

    n_epochs = config['n_epochs']  # Maximum number of epochs

    # Setup optimizer
    optimizer = getattr(torch.optim, config['optimizer'])(
        model.parameters(), **config['optim_hparas'])

    min_mse = 1000.
    loss_record = {'train': [], 'dev': []}      # for recording training loss
    early_stop_cnt = 0
```

```
    epoch = 0
    while epoch < n_epochs:
        model.train()                       # set model to training mode
        for x, y in tr_set:                 # iterate through the dataloader
            optimizer.zero_grad()           # set gradient to zero
            x, y = x.to(device), y.to(device)   # move data to device (cpu/cuda)
            pred = model(x)                 # forward pass (compute output)
            mse_loss = model.cal_loss(pred, y)  # compute loss
            mse_loss.backward()             # compute gradient (backpropagation)
            optimizer.step()                # update model with optimizer
            loss_record['train'].append(mse_loss.detach().cpu().item())

        # After each epoch, test your model on the validation (development) set.
        dev_mse = dev(dv_set, model, device)
        if dev_mse < min_mse:
            # Save model if your model improved
            min_mse = dev_mse
            print('Saving model (epoch = {:4d}, loss = {:.4f})'
                .format(epoch + 1, min_mse))
            torch.save(model.state_dict(), config['save_path'])  # Save model to specified path
            early_stop_cnt = 0
        else:
            early_stop_cnt += 1

        epoch += 1
        loss_record['dev'].append(dev_mse)
        if early_stop_cnt > config['early_stop']:
            # Stop training if your model stops improving for "config['early_stop']" epochs.
            break

    print('Finished training after {} epochs'.format(epoch))
    return min_mse, loss_record

def dev(dv_set, model, device):
    model.eval()                            # set model to evalutation mode
    total_loss = 0
    for x, y in dv_set:                     # iterate through the dataloader
        x, y = x.to(device), y.to(device)       # move data to device (cpu/cuda)
        with torch.no_grad():               # disable gradient calculation
            pred = model(x)                 # forward pass (compute output)
            # print(pred)
            # print(y)
            mse_loss = model.cal_loss(pred, y)  # compute loss
        total_loss += mse_loss.detach().cpu().item() * len(x)  # accumulate loss
    total_loss = total_loss / len(dv_set.dataset)              # compute averaged loss

    return total_loss

def test(tt_set, model, device):
    model.eval()                            # set model to evalutation mode
    preds = []
    for x in tt_set:                        # iterate through the dataloader
        x = x.to(device)                    # move data to device (cpu/cuda)
        with torch.no_grad():               # disable gradient calculation
            pred = model(x)                 # forward pass (compute output)
            preds.append(pred.detach().cpu())   # collect prediction
    preds = torch.cat(preds, dim=0).numpy()     # concatenate all predictions and convert to a numpy array
```

```
    return preds

device = get_device()                 # get the current available device ('cpu' or 'cuda')
os.makedirs('models', exist_ok=True)  # The trained model will be saved to ./models/
target_only = False                   # TODO: Using 40 states & 2 tested_positive features

# TODO: How to tune these hyper-parameters to improve your model's performance?
config = {
    'n_epochs': 3000,                # maximum number of epochs
    'batch_size': 270,               # mini-batch size for dataloader
    # 'batch_size': 135,
    'optimizer': 'SGD',              # optimization algorithm (optimizer in torch.optim)
    # 'optimizer': 'Adam',
    'optim_hparas': {                # hyper-parameters for the optimizer (depends on which optimizer you are using)
        'lr': 0.001,                 # learning rate of SGD
        'weight_decay': 0.05,        # regularization L2
        'momentum': 0.9              # momentum for SGD
    },
    'early_stop': 100,                    # early stopping epochs (the number epochs since your model's last improvement)
    'save_path': 'models/model.pth'  # your model will be saved here
}

del list

tr_set = prep_dataloader('train', config['batch_size'] )
dv_set = prep_dataloader('dev', config['batch_size'] )
tt_set = prep_dataloader('test', config['batch_size'] )

model = NeuralNet(tr_set.dataset.dim).to(device)

model_loss, model_loss_record = train(tr_set, dv_set, model, config, device)

plot_learning_curve(model_loss_record, title='vote_count deep model')

model_loss_record['train']

del model
model = NeuralNet(tr_set.dataset.dim).to(device)
ckpt = torch.load(config['save_path'], map_location='cpu')  # Load your best model
model.load_state_dict(ckpt)
plot_pred(dv_set, model, device,lim=10.)  # Show prediction on the validation set

def save_pred(preds, file):
    ''' Save predictions to specified file '''
    print('Saving results to {}'.format(file))
    with open(file, 'w') as fp:
        writer = csv.writer(fp)
        writer.writerow(['id', 'vote_average(be)'])
        for i, p in enumerate(preds):
            writer.writerow([i, p])

preds = test(tt_set, model, device)  # predict cases with your model
save_pred(preds, 'pred.csv')         # save prediction file to pred.csv
```

附录四 问题三决策树的 SPSS 结果分析

警告

执行修剪时，不会计算交叉验证风险估计
由于未定义利润，因此未显示增益摘要表。
由于未定义目标类别，因此未显示目标类别增益表。

模型摘要

指定项	生长法	CRT
	因变量	revenue
	自变量	budget, popularity, genres0, genres1, genres2, genres3, genres4, genres5, genres6, genres7, genres8, genres9, genres10, genres11, genres12, genres13, genres14, genres15, genres16, genres17, genres18, genres19
	验证	无
	最大树深度	5
	父节点中的最小个案数	100
	子节点中的最小个案数	50
结果	包括的自变量	popularity, budget, genres11, genres14, genres15, genres8, genres18, genres4
	节点数	19
	终端节点数	10
	深度	4

风险

估算	标准误差
.396	.009

生长法：CRT
因变量：revenue

分类

	预测					
实测	1.0	2.0	3.0	4.0	5.0	正确百分比
1.0	0	51	73	6	1	0.0%
2.0	0	80	220	30	0	24.2%
3.0	0	43	606	310	5	62.9%
4.0	0	3	233	919	123	71.9%
5.0	0	0	13	138	302	66.7%
总体百分比	0.0%	5.6%	36.3%	44.5%	13.7%	60.4%

生长法：CRT
因变量：revenue

自变量重要性

自变量	重要性	正态化重要性
budget	.135	100.0%
popularity	.112	82.5%
genres18	.003	2.4%
genres8	.002	1.7%
genres11	.002	1.6%
genres15	.001	0.8%
genres4	.001	0.5%
genres14	.000	0.4%

生长法：CRT
因变量：revenue