基于 LSTM 的边缘节点热门内容识别模型的设计与实现

摘要

本报告旨在设计一个基于 LSTM（长短期记忆网络）的模式识别模型，用于预测 CDN（内容分发网络）中边缘节点的缓存内容的热门程度。通过对历史请求量数据的分析和模式识别，模型能够捕捉时间序列中的模式和特征，准确预测未来的请求量，并根据预测结果对内容进行分类（热门、一般、冷门）。该模型有助于优化 CDN 的缓存策略，提高网络资源的利用率。

1 引言

在 CDN 的边缘节点中，准确预测内容的热门程度对于优化缓存策略、提高资源利用率具有重要意义。通过机器学习方法，尤其是深度学习中的时序模型和模式识别技术，可以捕捉内容请求量的时间依赖性和隐藏模式，提供高精度的预测。

2 问题描述

目标：设计一个基于 LSTM 和模式识别的模型，预测边缘节点中内容的请求量，根据预测结果将内容分类为热门、一般、冷门。

挑战：

- 数据的时间依赖性和复杂性，存在多种模式和趋势。
- 模型需要处理大量内容的并行时间序列，进行有效的模式识别。
- 需要将预测的请求量转换为业务可用的分类，辅助决策。

3 模型设计

表1: 原始请求量数据（时间序列）

时间步(t)	内容1(Content_1)	内容2(Content_2)	内容3(Content_3)	内容4(Content_4)	内容5(Content_5)
1	150	80	200	50	120
2	160	85	210	55	130
3	170	90	220	60	140
4	180	95	230	65	150
5	190	100	240	70	160
6	200	105	250	75	170
...	...	...	...	...	...

3.1 数据生成与预处理

数据生成：生成模拟边缘节点中内容请求量数据，如表 1 所示，原始请求量数据是指在未经过任何预处理或变换前的边缘节点内容请求量随时间变化的记录。该数据通常由多个时间步（如分钟、小时、天）组成，每个时间步对应一组内容的请求量。具体来说：

时间维度：数据沿时间轴排列，每个时间步都记录各内容的请求次数或访问量。

多内容并列：对于每个时间步，数据包含多个内容的请求量，从而构成一个二维结构（时间×内容数）。

数值特征：请求量通常是一个非负整数或实数，代表该时间步内用户对某一内容的访问频度。

原始状态：此时的数据尚未进行缩放、归一化或分块处理，可能包含各种尺度差异。热门内容的请求量可能远高于冷门内容，或在不同时段呈现明显的增长、下降或季节性波动。

通过观察原始请求量数据，可以直观了解在不同时间内容访问量的变化情况，但此时的数据由于尺度差异和随机波动较大，尚不利于直接进行建模和训练。

表2: 归一化后的请求量数据

时间步(t)	内容1(Content_1)	内容2(Content_2)	内容3(Content_3)	内容4(Content_4)	内容5(Content_5)
1	0.25	0.20	0.33	0.10	0.25
2	0.27	0.21	0.35	0.11	0.28
3	0.30	0.23	0.38	0.13	0.31
4	0.32	0.24	0.40	0.14	0.33
5	0.35	0.26	0.43	0.16	0.36
6	0.37	0.27	0.45	0.17	0.38
...	...	...	...	...	...

数据预处理：对数据进行归一化处理，使用 MinMaxScaler 将请求量映射到[0,1]区间，消除不同尺度的影响。如表 2 所示，归一化后的请求量数据是对原始请求量数据进行数值变换（使用 MinMaxScaler）后的结果。通过归一化，可将所有内容的请求量缩放到相同的数值范围（例如[0,1]），从而减轻原始数据尺度差异带来的影响。具体特点如下：

统一尺度：归一化过程确保不同内容的请求量特征映射到相同的数值区间。这样，即使某些内容的原始访问量非常高，另一些内容的访问量较低，在归一化后它们都会被映射到 0 到 1 之间的相对值，以体现相对变化趋势。

保留相对关系：归一化不会改变数据在时间序列维度上的相对变化趋势。例如，一个内容在一段时间内请求量上升的模式仍会在归一化后保留，只是尺度被压缩或放大。

利于建模与训练：对于 LSTM 等深度学习模型，输入数据的数值范围过大或尺度不一致可能会降低训练效率和稳定性。通过归一化，模型更容易收敛，且不会让某些内容的巨大请求量主导训练过程。

适用多特征场景：如果未来加入额外特征（如时间特征、内容元数据），归一化步骤也可确保新特征与请求量特征在同一数量级上，利于多特征融合的建模。

表3: 构建的输入序列(X)和目标值(y)

样本编号	输入序列 (X)	目标值 (y)
1	[0.25,0.20,0.33,0.10,0.25], [0.27,0.21,0.35,0.11,0.28], [0.30,0.23,0.38,0.13,0.31]	0.32,0.24,0.40,0.14,0.33
2	[0.27,0.21,0.35,0.11,0.28], [0.30,0.23,0.38,0.13,0.31], [0.32,0.24,0.40,0.14,0.33]	0.35,0.26,0.43,0.16,0.36
3	[0.30,0.23,0.38,0.13,0.31], [0.32,0.24,0.40,0.14,0.33], [0.35,0.26,0.43,0.16,0.36]	0.37,0.27,0.45,0.17,0.38
…	…	…

数据集划分：首先，从归一化后的数据中构建输入序列和对应的目标值，假设 sequence_length=3，表 3 展示了如何从归一化后的数据中构建输入序列和对应的目标值，在这个数据集中，输入序列 X 由连续的 sequence_length 个时间步组成，每个时间步包含所有内容的归一化请求量。例如，样本 1 的输入序列包括时间步 1、2、3 的数据。相应地，目标值 y 对应于输入序列后面的下一个时间步的所有内容的请求量，即样本 1 的目标值对应于时间步 4 的数据。这种结构允许模型学习基于前面几个时间步的数据来预测下一个时

间步的请求量。然后，按照 8:1:1 的比例将数据集划分为训练集、验证集和测试集，确保数据的合理分配。最后，利用 DataLoader 创建数据加载器，以支持模型的批量训练和验证过程，提高数据处理的效率和灵活性。

3.2 模式识别与特征提取

模式识别：充分利用 LSTM 的记忆能力，深入挖掘时间序列中的长期和短期模式，精准识别内容请求量的周期性波动、整体趋势变化以及突发性事件特征。

特征提取：通过提取时间序列的统计特征，全面丰富模型的输入特征，提升特征表达的深度和广度。

模型结构：在模型选择方面，采用 LSTM 模型，结合模式识别技术，适合处理复杂的时间序列数据。

3.3 模型架构

图1: LSTM模型架构

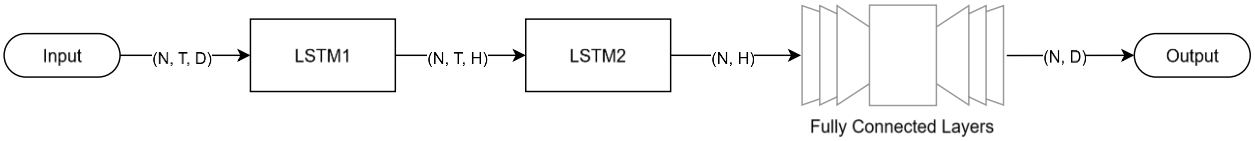
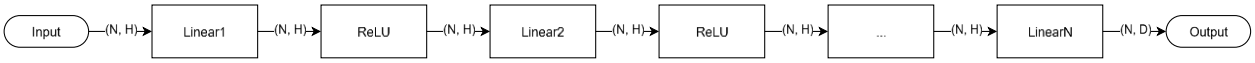


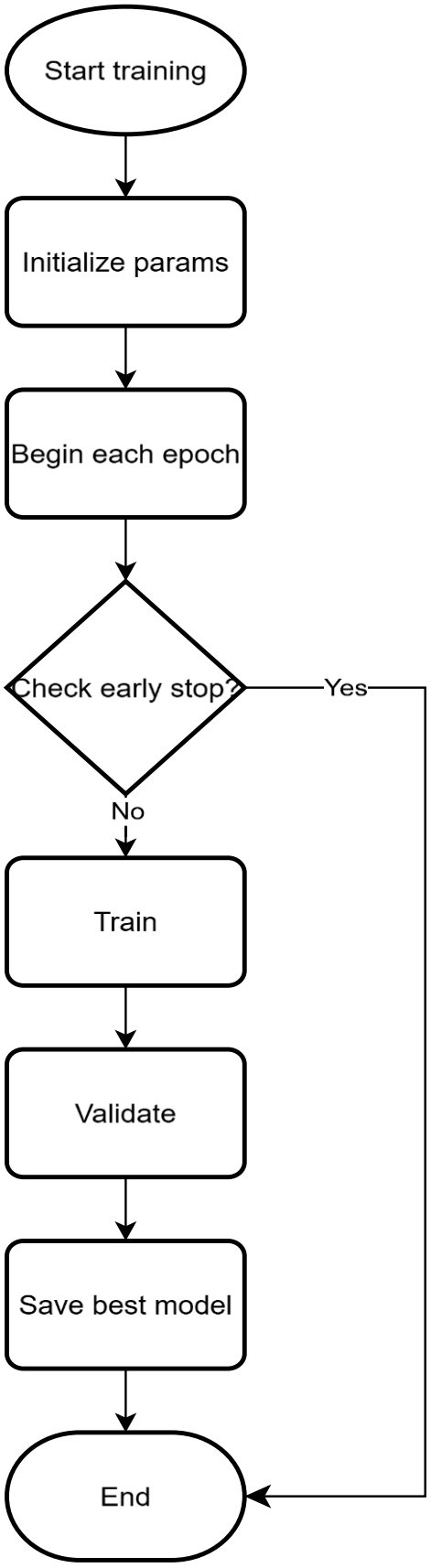
图2: 全连接层



整个模型的输入数据具有维度为 (N, T, D) 的张量结构，其中 N 表示批量大小（即一次训练迭代中包含的样本数），T 表示时间步的数量（即序列的长度），D 则是特征维度（对应内容数量）。首先，输入序列以 (N, T, D) 的形式传入第一层 LSTM（LSTM1），这一层将对每个时间步的特征进行处理，输出维度为 (N, T, H)，其中 H 是隐藏单元数量。接着，将处理后的张量输入到第二层 LSTM（LSTM2）中，LSTM2 同样输出 (N, T, H) 的张量。为了进一步进行预测，我们从 LSTM2 的输出中抽取最后一个时间步的特征，得到 (N, H) 的表示。随后，该 (N, H) 的张量会通过如图 2 所示的多个全连接层（全连接层的输入与输出维度均为 (N, H)，除了最后一层将输出维度映射回 (N, D)），在每个全连接层之间插入 ReLU 激活函数，以提高模型的非线性拟合能力。最终的输出层将得到形状为 (N, D) 的预测值，代表每个样本（N）在下一个时间步对 D 个目标的预测结果。通过这种从 (N, T, D) 输入到 (N, D) 输出的深度结构设计，模型可以有效捕捉时间序列中的复杂模式和长期依赖关系，从而提升预测的准确性和稳定性。

3.4 训练策略

图3: 训练策略



如图 3 所示，整个训练策略的基本思路是：在每个训练周期中，先让模型在训练集上学习，将预测结果与真实值进行比较并根据损失函数反馈进行参数更新；随后，在验证集上评估当前模型的性能，并与历史最好结果对比。如果验证表现有所提升，则保存当前模型为最佳模型状态，以便后续使用；如果验证表现连续多次未改善，则提前停止训练，避免在无实际收益的迭代中浪费时间和引发过拟合。通过这种有目标的迭代优化、性能评估和模型保存机制，模型既能持续提升，又能在合适的时机终止训练，从而在验证集上达到较优的泛化性能。

4 模型实现

4.1 数据生成器（DataGenerator）

数据生成器通过从指定数量的内容中选取一部分作为热门内容，并对这些内容在每个时间步内的请求量进行泊松分布模拟，从而构建出具有不同请求模式的时间序列数据集。生成的数据首先以原始形式存储，然后经过预处理步骤：根据设定的序列长度，将连续的时间步切片为模型可接受的输入特征（X）和预测目标（y），并使用 MinMaxScaler 对原始请求量进行归一化，从而确保各内容请求量处于相同的数值范围并降低数据尺度差异带来的影响。处理完成后，X 和 y 被转换为 PyTorch 张量，并持久化保存，便于随时加载和复用。最终通过在数据集中随机打乱并根据给定比例拆分出训练集、验证集和测试集，数据生成器为后续模型训练和评估提供高质量且灵活可控的数据加载器（DataLoader），保证训练过程的高效与稳定。具体代码如下所示：

```
class DataGenerator:
    """
    DataGenerator 类是一个数据生成器，用于生成模拟数据集并将其转换为 PyTorch DataLoader 对象。

    Args:
        config (Config): 配置对象
    """

    def __init__(self, config: Config):
        # 省略初始化

    def generate_data(self):  # 生成数据
        popular_contents = np.random.choice(
            self.num_contents, size=self.popular_size, replace=False
        )
```

```
        data = []
        for _ in range(self.time_steps):
            content_requests = np.random.poisson(lam=5, size=self.num_contents)

            content_requests[popular_contents] += np.random.poisson(lam=20)
            data.append(content_requests)
        data = np.array(data)
        self.df = pd.DataFrame(
            data, columns=[f"Content_{i}" for i in range(self.num_contents)]
        )
        # 保存到 data/raw_data.csv
        self.df.to_csv(self.raw_data_path, index=False)

    def preprocess_data(self):  # 处理数据
        X = []
        y = []
        for i in range(len(self.df) - self.sequence_length):
            X.append(self.df.iloc[i : i + self.sequence_length].values)
            y.append(self.df.iloc[i + self.sequence_length].values)
        X = np.array(X)
        y = np.array(y)
        # 数据归一化
        X_shape = X.shape
        X = X.reshape(-1, self.num_contents)
        X = self.scaler.fit_transform(X)
        X = X.reshape(X_shape)
        y = self.scaler.transform(y)
        # 转换为张量
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y, dtype=torch.float32)
        # 保存
        torch.save(self.X, "data/X.pt")
        torch.save(self.y, "data/y.pt")

    def get_data_loaders(self, batch_size, train_ratio, val_ratio, test_ratio):
        # 省略生成训练集、验证集和测试集的数据加载器
```

4.2 模型定义（LSTMModel）

在本模型的实现中，输入数据的维度为 (N, T, D)，其中 N 为批量大小（例如 N=32），T 为时间步长（例如 T=10），D 为特征维度（例如 D=1000，对应内容数量），模型首先接收一个形状为 (32, 10, 1000) 的张量作为输入序列。接着，模型在初始化时将 input_size 设为 1000（对应 D），hidden_size 设为 128，num_layers 设为 2，dropout 设为 0.5，并将 num_linear_layers 设为 3，以形成一套较深且具有一定正则化能力的结构。在前向传播中，输入首先通过第一层 LSTM（双层堆叠、隐藏维度为 128）处理，将形状从 (32, 10, 1000) 映射至 (32, 10, 128)。接着数据继续进入第二层 LSTM，输出仍为 (32, 10, 128)，此时提取序列的最后一个时间步输出，得到 (32, 128) 的张量。然后该张量传入多层全连接层（共 3 层），其中前两层全连接层的输入与输出均为 (32, 128)，在层间加入 ReLU 激活函数以增强非线性；最后一层全连接层将特征映射回 (32, 1000)，即输出与输入特征数 D 保持一致，从而得到针对所有内容的预测请求量。通过这种两层 LSTM 嵌套加多层全连接层叠加的架构，模型能够有效提取时间序列的深层特征和复杂模式，并在最终输出层对所有内容进行并行预测。具体代码如下所示：

```
class LSTMModel(nn.Module):
    """
    LSTMModel 类是一个 LSTM 模型。
    """
    def __init__(self, config):
        # 省略普通参数初始化
        # 定义两层 LSTM
        self.lstm1 = nn.LSTM(
            self.input_size,
            self.hidden_size,
            num_layers=self.num_layers,
            batch_first=True,
            dropout=self.dropout,
        )
        self.lstm2 = nn.LSTM(
            self.hidden_size,
            self.hidden_size,
            num_layers=self.num_layers,
            batch_first=True,
            dropout=self.dropout,
        )
        # 定义全连接层列表
        self.fcs = nn.ModuleList(
            [
```

```
                nn.Linear(self.hidden_size, self.hidden_size)
                for _ in range(self.num_linear_layers - 1)
            ]
            + [nn.Linear(self.hidden_size, self.output_size)]
        )

        self.relu = nn.ReLU()

    def forward(self, x):
        # 初始化隐藏状态和细胞状态
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)

        # LSTM 前向传播
        out, (hn, cn) = self.lstm1(x, (h0, c0))
        out, _ = self.lstm2(out, (hn, cn))
        # 取最后一个时间步的输出
        out = out[:, -1, :]

        # 全连接层前向传播
        for fc in self.fcs[:-1]:
            out = fc(out)
            out = self.relu(out)

        return self.fcs[-1](out)
```

4.3 模型训练器（ModelTrainer）

ModelTrainer 类负责模型的训练和评估，确保其有效学习数据特征并在验证集上取得优异性能。核心功能包括：使用均方误差（nn.MSELoss()）作为损失函数，衡量预测值与实际值的差异；采用带权重衰减的 AdamW 优化器，提升泛化能力并防止过拟合；设置训练轮数、早停参数（patience 和 min_delta），并记录训练和验证损失。

训练阶段，模型在训练模式下通过前向传播、反向传播和参数更新优化性能；验证阶段，模型切换至评估模式，计算验证损失并记录。通过早停机制监测验证损失的改善，当损失未显著降低且达到设定耐心值时停止训练，同时在损失下降时保存最佳模型至指定路径（save_path）。

评估阶段支持从保存路径或内存加载最佳模型状态，通过去梯度计算、反归一化数据和计算均方误差（MSE）评估性能。此外，通过记录训练与验证损失、使用更高效的优化

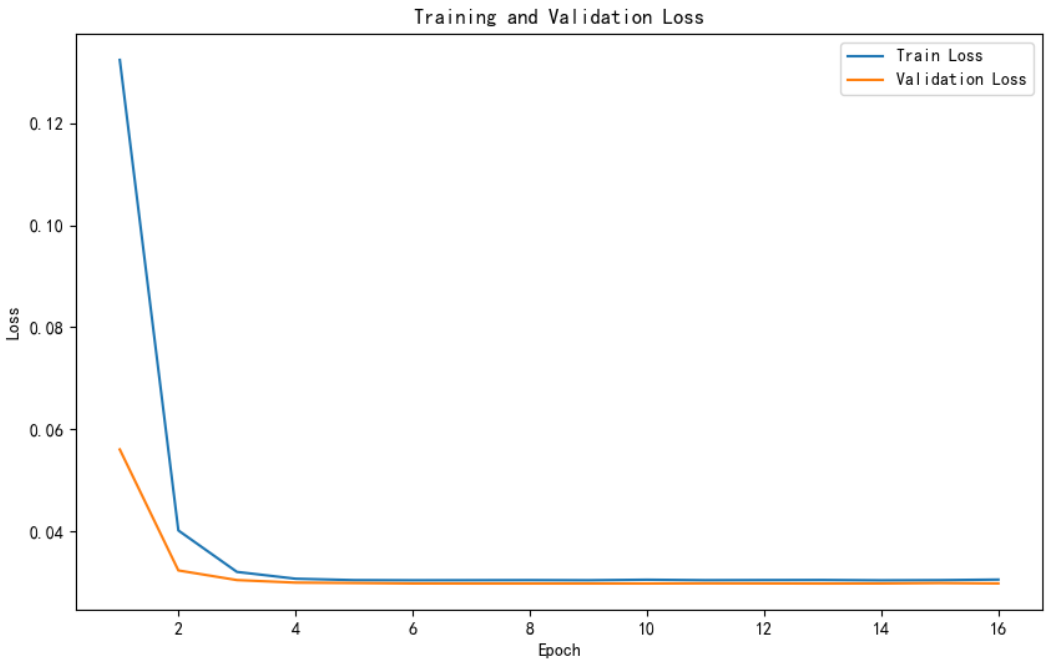
器和早停策略，以及灵活的模型加载，ModelTrainer 提升了训练效率、避免过拟合，并增强了模型的可用性和预测能力。

4.4 可视化与分析（Visualizer）

Visualizer 类提供多样化的可视化工具，包括整体损失曲线和单个内容的预测曲线，以全面展示模型性能。通过图形化方式直观对比预测结果与实际值，帮助发现模型的优势和改进空间。此外，结合 ContentClassifier，可对内容的受欢迎程度进行分类分析，评估模型在分类任务上的表现。这些功能为模型的评估与分析提供了强有力的支持，帮助理解模型行为并指导优化改进。

5 实验结果

图 4：训练和验证阶段损失



训练过程： 如图 4 所示，模型在训练集上的损失逐步降低，同时验证集的损失也同步下降，表明其通过模式识别有效学习了时间序列中的规律。早停机制进一步防止过拟合，确保训练在验证集表现最佳时保存模型状态，从而提升模型的泛化能力和稳定性。

图 5：最热门的前 10 个内容

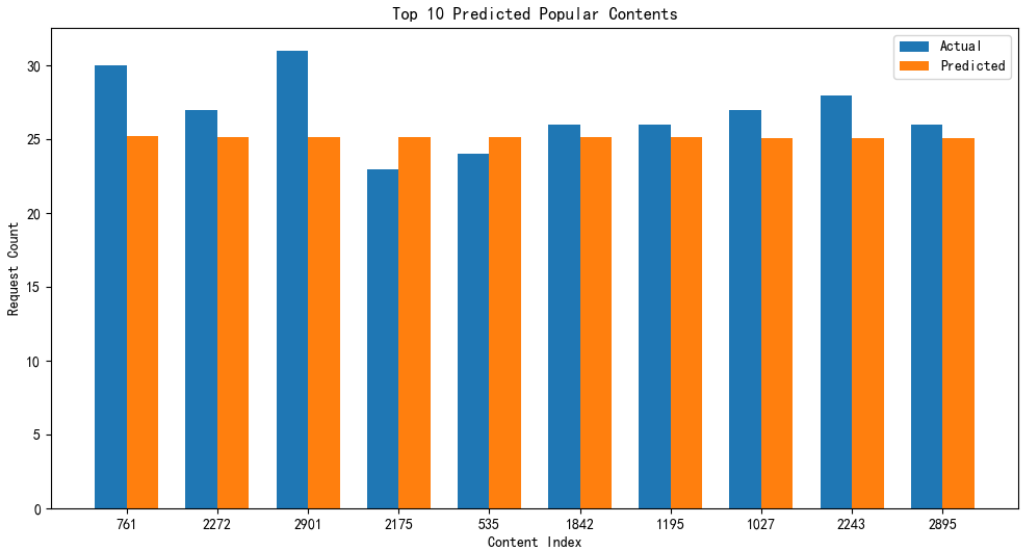
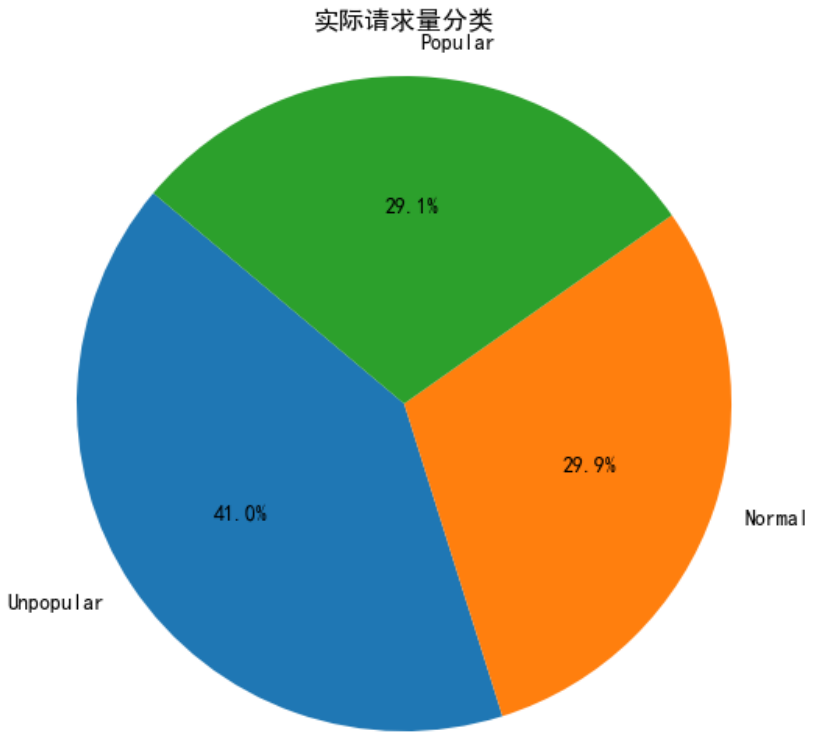


图 6：内容分类



预测结果： 如图 5 所示，在测试集上，模型对大部分内容的预测请求量与实际请求量高度吻合，误差不超过 5，能够精准捕捉趋势和变化模式，展现出对时间序列数据的良好建模能力。同时，如图 6 所示，通过分类方法，模型可以准确地将内容划分为热门、一般

和冷门类别，表现出较高的分类准确率，为进一步优化内容推荐和需求预测提供了可靠支持。

6 可改进的方向

模型结构改进： 可以引入双向 LSTM，以更全面地捕捉时间序列的前后依赖关系，进一步提高模型的预测精度。此外，应用注意力机制，让模型能够聚焦于关键时间步或重要特征，从而增强模式识别能力，提升对复杂时间序列数据的处理效果。

防止过拟合： 引入 L1/L2 正则化、Dropout 等，增强模型的泛化能力。