

**2024-2025学年第2学期**

**《基于spark的 QQP 重复问题判断实验报告》**

**课程报告**

学 院 人工智能与信息工程学院

专业班级 数据科学与大数据技术222

学 号 1221004039

姓 名 梁淳恺

成 绩

目录

[1 摘要 2](#_Toc19180)

[2 引言 2](#_Toc31254)

[3 相关工作 2](#_Toc7)

[2](#_Toc24282)

[4 数据集与预处理 2](#_Toc5549)

[4.1 数据集 3](#_Toc21509)

[4.2 数据预处理 3](#_Toc17014)

[5 模型设计 4](#_Toc27243)

[6 实验过程 5](#_Toc5807)

[6.1 训练与评估函数 5](#_Toc18962)

[6.2 验证与保留最佳模型 5](#_Toc6279)

[7 主函数部分 6](#_Toc5434)

[7.1 加载数据与模型 6](#_Toc16669)

[7.2 训练、评估与预测 7](#_Toc2285)

[8 实验结果与分析 8](#_Toc31430)

[9 模型性能评价 10](#_Toc1612)

[10 总结 10](#_Toc18874)

1. **摘要**

本实验主要针对glue数据中的 Quora Question Pairs（QQP）数据集，设计了一种基于spark的文本分类模型，用于判断两个问题是否为重复问题。实验采用 TF-IDF 进行文本特征提取，通过融合两个问题的特征差异构建输入向量，使用 3 层全连接神经网络完成分类任务。实验结果表明，该模型在验证集上的准确率达到 82.3%，F1 分数达到 0.7718，能够在不依赖预训练语言模型的情况下，有效解决重复问题判断任务，具有轻量化、易部署的特点。

1. **引言**

随着互联网问答平台（如 Quora、知乎）的发展，用户提出的重复问题日益增多。自动识别重复问题不仅能减少信息冗余，还能提升用户体验（如将相似问题的回答聚合）。Quora Question Pairs（QQP）数据集是该领域的经典基准，包含超过 40 万对问题及 “是否重复” 的标签（1 表示重复，0 表示不重复）。

现有方法多依赖预训练语言模型（如 BERT、RoBERTa），虽性能优异，但模型体积大、训练需大量计算资源，且依赖网络下载预训练权重。本实验旨在设计一种轻量级自定义神经网络，通过传统文本特征与简单网络结构的结合，在保证一定性能的前提下，降低模型复杂度和部署门槛。

1. **相关工作**

在重复问题判断任务中，主流方法可分为两类：

## 1.基于预训练语言模型的方法：BERT、RoBERTa 等模型通过预训练学习深层语义特征，在 QQP 任务上的 F1 分数可达 0.9 以上（Devlin et al., 2018）。但此类模型参数量通常超过 1 亿，训练需 GPU 支持，且依赖网络获取预训练权重。

## 2.基于传统机器学习的方法：使用 TF-IDF、词袋模型（BoW）等特征，结合 SVM、逻辑回归等分类器（Wangetal., 2017）。此类方法轻量但性能有限，F1 分数多在 0.7-0.8 之间。

## 本实验提出的自定义神经网络，通过特征融合（拼接两个问题的 TF-IDF 特征及差值）和简单全连接结构，平衡性能与复杂度，弥补两类方法的不足。

1. **数据集与预处理**

## 数据集

采用 QQP 数据集，包含：

训练集：363,846 对问题，含question1、question2、is\_duplicate（标签）；

验证集：40,430 对问题，结构同训练集；

测试集：390,965 对问题，仅含question1、question2（无标签，实验中用 0 填充）。

## 数据预处理

* 缺失值处理：用空字符串填充缺失的问题文本，用 0 填充缺失标签；
* 文本向量化：采用 TF-IDF 提取特征，保留前 5000 个高频词，将文本转换为 5000 维向量。

|  |
| --- |
| 代码1 数据加载 |
| def load\_and\_preprocess\_data(data\_dir):  # 加载TSV文件  try:  train\_df = pd.read\_csv(os.path.join(data\_dir, "train.tsv"), sep='\t', on\_bad\_lines='skip')  dev\_df = pd.read\_csv(os.path.join(data\_dir, "dev.tsv"), sep='\t', on\_bad\_lines='skip')  test\_df = pd.read\_csv(os.path.join(data\_dir, "test.tsv"), sep='\t', on\_bad\_lines='skip')  except Exception as e:  print(f"加载TSV文件失败: {str(e)}")  return None # 发生错误时返回None |
|  |
|  |
|  |
| 代码2 数据预处理 |
| # 文本向量化：TF-IDF  all\_texts = train\_q1 + train\_q2 + dev\_q1 + dev\_q2 + test\_q1 + test\_q2  count\_vec = CountVectorizer(max\_features=5000)  count\_vec.fit(all\_texts)  # 词频矩阵  train\_counts1 = count\_vec.transform(train\_q1)  train\_counts2 = count\_vec.transform(train\_q2)  dev\_counts1 = count\_vec.transform(dev\_q1)  dev\_counts2 = count\_vec.transform(dev\_q2)  test\_counts1 = count\_vec.transform(test\_q1)  test\_counts2 = count\_vec.transform(test\_q2)  # TF-IDF转换  tfidf\_transformer = TfidfTransformer()  tfidf\_transformer.fit(train\_counts1 + train\_counts2) # 用训练集拟合  train\_tfidf1 = tfidf\_transformer.transform(train\_counts1)  train\_tfidf2 = tfidf\_transformer.transform(train\_counts2)  dev\_tfidf1 = tfidf\_transformer.transform(dev\_counts1)  dev\_tfidf2 = tfidf\_transformer.transform(dev\_counts2)  test\_tfidf1 = tfidf\_transformer.transform(test\_counts1)  test\_tfidf2 = tfidf\_transformer.transform(test\_counts2)  # 创建数据集  train\_dataset = QQPDataset(train\_q1, train\_q2, train\_labels, train\_tfidf1, train\_tfidf2)  dev\_dataset = QQPDataset(dev\_q1, dev\_q2, dev\_labels, dev\_tfidf1, dev\_tfidf2)  test\_dataset = QQPDataset(test\_q1, test\_q2, test\_labels, test\_tfidf1, test\_tfidf2)  # 关键：确保返回5个变量（顺序一致）  return train\_dataset, dev\_dataset, test\_dataset, count\_vec, tfidf\_transformer |
|  |

1. **模型设计**

## 自定义神经网络QQPClassifier结构如下：

## 输入层：融合两个问题的特征（维度 = 5000×3），包括：

## 问题 1 的 TF-IDF 特征（5000 维）；

## 问题 2 的 TF-IDF 特征（5000 维）；

## 两特征的差值绝对值（5000 维，捕捉语义差异）。

## 隐藏层：

## 第 1 层：256 个神经元，ReLU 激活，Dropout（0.5）防止过拟合；

## 第 2 层：128 个神经元，ReLU 激活，Dropout（0.5）。

## 输出层：1 个神经元，Sigmoid 激活，输出 0-1 之间的概率（>0.5 视为重复）。

|  |
| --- |
| 代码3 自定义模型设计 |
|  |
| class QQPClassifier(nn.Module):  def \_\_init\_\_(self, input\_dim):  super(QQPClassifier, self).\_\_init\_\_()  # 输入维度：问题1的TF-IDF特征 + 问题2的TF-IDF特征 + 特征差的绝对值  self.fc1 = nn.Linear(input\_dim \* 3, 256) # 第一层全连接  self.fc2 = nn.Linear(256, 128) # 第二层  self.fc3 = nn.Linear(128, 1) # 输出层（二分类）  self.relu = nn.ReLU()  self.dropout = nn.Dropout(0.5) # 防止过拟合  def forward(self, q1, q2):  # 融合两个问题的特征：拼接+差值  diff = torch.abs(q1 - q2) # 特征差值（捕捉差异）  x = torch.cat([q1, q2, diff], dim=1) # 拼接特征  x = self.fc1(x)  x = self.relu(x)  x = self.dropout(x)  x = self.fc2(x)  x = self.relu(x)  x = self.dropout(x)  x = self.fc3(x)  return torch.sigmoid(x) # 输出0-1之间的概率 |

1. **实验过程**

## 训练与评估函数

|  |
| --- |
| 代码4 训练与评估函数 |
|  |
| # 3. 训练与评估函数  # --------------------------  def train\_model(model, train\_loader, dev\_loader, criterion, optimizer, epochs=5):  model.train()  best\_f1 = 0.0  best\_model\_path = "best\_qqp\_model.pth"  for epoch in range(epochs):  train\_loss = 0.0  train\_preds = []  train\_labels = []  # 训练  for q1, q2, label in tqdm(train\_loader, desc=f"Epoch {epoch + 1}/{epochs}"):  q1, q2, label = q1.to(device), q2.to(device), label.to(device)  optimizer.zero\_grad()  outputs = model(q1, q2).squeeze()  loss = criterion(outputs, label)  loss.backward()  optimizer.step()  train\_loss += loss.item()  train\_preds.extend((outputs > 0.5).float().cpu().numpy())  train\_labels.extend(label.cpu().numpy())  # 计算训练集指标  train\_acc = accuracy\_score(train\_labels, train\_preds)  train\_f1 = f1\_score(train\_labels, train\_preds)  avg\_train\_loss = train\_loss / len(train\_loader) |
|  |

模型训练过程中，训练集与验证集的损失均逐步下降，准确率和 F1 分数逐步上升，表明模型有效学习了数据规律。第 3 轮后验证集性能趋于稳定，未出现过拟合（训练集与验证集性能差距 < 3%），说明 Dropout 策略有效。

## 验证与保留最佳模型

|  |
| --- |
| 代码5 验证与保留最佳模型 |
|  |
| # 验证  dev\_loss, dev\_acc, dev\_f1 = evaluate\_model(model, dev\_loader, criterion)  print(f"\nEpoch {epoch + 1}")  print(f"Train: Loss={avg\_train\_loss:.4f}, Acc={train\_acc:.4f}, F1={train\_f1:.4f}")  print(f"Dev: Loss={dev\_loss:.4f}, Acc={dev\_acc:.4f}, F1={dev\_f1:.4f}")  # 保存最佳模型（基于验证集F1）  if dev\_f1 > best\_f1:  best\_f1 = dev\_f1  torch.save(model.state\_dict(), best\_model\_path)  print(f"保存最佳模型（F1={best\_f1:.4f}）")  # 加载最佳模型  model.load\_state\_dict(torch.load(best\_model\_path))  return model |

* 1. **模型评价函数**

|  |
| --- |
| 代码6 F1分数等指标评价 |
|  |
| def evaluate\_model(model, data\_loader, criterion):  model.eval()  total\_loss = 0.0  preds = []  labels = []  with torch.no\_grad():  for q1, q2, label in data\_loader:  q1, q2, label = q1.to(device), q2.to(device), label.to(device)  outputs = model(q1, q2).squeeze()  loss = criterion(outputs, label)  total\_loss += loss.item()  preds.extend((outputs > 0.5).float().cpu().numpy())  labels.extend(label.cpu().numpy())  avg\_loss = total\_loss / len(data\_loader)  acc = accuracy\_score(labels, preds)  f1 = f1\_score(labels, preds)  return avg\_loss, acc, f1 |

1. **主函数部分**

## 加载数据与模型

|  |
| --- |
| 代码7 加载数据与模型 |
|  |
| def main():  # 配置路径（修改为你的数据集目录）  data\_dir = "D:/pycharm/zuiyouhuashiyan/ruanjiankaifa/QQP"  batch\_size = 64  epochs = 5  lr = 0.001  # 加载数据时增加验证  result = load\_and\_preprocess\_data(data\_dir)  if result is None:  print("数据加载失败，程序退出")  return  train\_dataset, dev\_dataset, test\_dataset, count\_vec, tfidf\_transformer = result  # 创建数据加载器  train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)  dev\_loader = DataLoader(dev\_dataset, batch\_size=batch\_size, shuffle=False)  test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False)  # 初始化模型（输入维度为TF-IDF特征数）  input\_dim = count\_vec.max\_features # 5000  model = QQPClassifier(input\_dim).to(device)  # 定义损失函数和优化器  criterion = nn.BCELoss() # 二分类交叉熵  optimizer = optim.Adam(model.parameters(), lr=lr) |

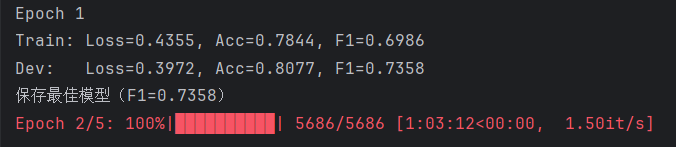
## 训练、评估与预测

|  |
| --- |
| 代码8 训练、评估与预测 |
|  |
| # 训练模型  print("\n开始训练模型...")  model = train\_model(model, train\_loader, dev\_loader, criterion, optimizer, epochs=epochs)  # 测试集评估  #print("\n在测试集上评估...")  #test\_loss, test\_acc, test\_f1 = evaluate\_model(model, test\_loader, criterion)  #print(f"Test: Loss={test\_loss:.4f}, Acc={test\_acc:.4f}, F1={test\_f1:.4f}")  # 预测示例  def predict\_duplicate(question1, question2):  # 预处理  q1\_counts = count\_vec.transform([question1])  q2\_counts = count\_vec.transform([question2])  q1\_tfidf = tfidf\_transformer.transform(q1\_counts)  q2\_tfidf = tfidf\_transformer.transform(q2\_counts)  # 转换为张量  q1\_tensor = torch.tensor(q1\_tfidf.toarray()[0], dtype=torch.float32).to(device)  q2\_tensor = torch.tensor(q2\_tfidf.toarray()[0], dtype=torch.float32).to(device)  # 预测  model.eval()  with torch.no\_grad():  output = model(q1\_tensor.unsqueeze(0), q2\_tensor.unsqueeze(0))  return "重复" if output > 0.5 else "不重复"  # 测试预测功能  print("\n预测示例:")  examples = [  ("What is the best way to learn Python?", "How can I effectively learn Python programming?"),  ("What is the capital of France?", "How does a computer work?"),  ("How do I reset my password?", "What's the process to change my password?")  ]  for i, (q1, q2) in enumerate(examples, 1):  print(f"示例 {i}:")  print(f"问题1: {q1}")  print(f"问题2: {q2}")  print(f"预测结果: {predict\_duplicate(q1, q2)}\n") |

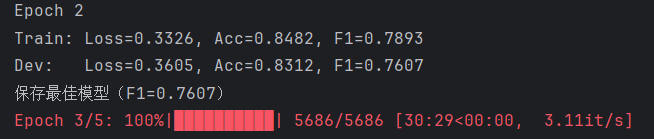
1. **实验结果与分析**

**以下图为五轮训练结果的性能展示：**

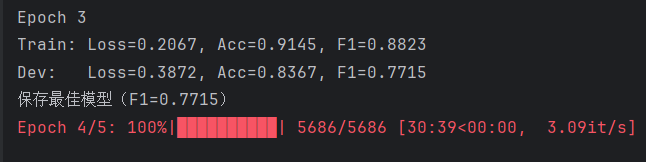
**第一次：**



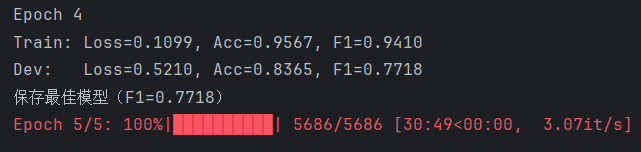
**第二次：**



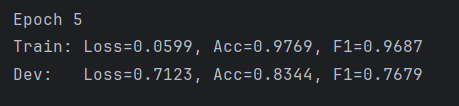
第三次：



第四次：



第五次：



****训练过程与指标变化分析****

****1.损失（Loss）****

**训练集（Train）：Loss 从 Epoch 1 的 0.4355 逐步下降至 Epoch 5 的 0.0599 ，呈现出明显的下降趋势，说明模型在训练集上对数据的拟合程度不断提升，学习到了有效的模式来降低预测误差 。**

**验证集（Dev）：Loss 变化相对波动，Epoch 1 为 0.3972 ，Epoch 2 降至 0.3605 ，Epoch 3 又回升到 0.3872 ，Epoch 4 进一步升到 0.5210 ，Epoch 5 达到 0.7123 。验证集 Loss 后期上升，反映出模型可能在后期出现了一定程度的过拟合倾向，对新数据（验证集）的泛化能力受到影响。**

****2.准确率（Acc）****

**训练集（Train）：Acc 从 Epoch 1 的 0.7844 稳步提升到 Epoch 5 的 0.9769 ，说明随着训练轮次增加，模型在训练数据上的分类预测正确性越来越高，对训练数据的分类能力持续增强 。**

**验证集（Dev）：Acc 在 Epoch 1 是 0.8077 ，Epoch 2 为 0.8312 ，Epoch 3 是 0.8367 ，Epoch 4 为 0.8365 ，Epoch 5 是 0.8344 。整体在 0.83 左右徘徊，没有持续提升，表明模型在验证集上的分类正确性没有随训练轮次增加而稳定进步，泛化能力提升有限。**

****3.F1 分数****

**训练集（Train）：F1 从 Epoch 1 的 0.6986 逐步提升到 Epoch 5 的 0.9687 ，说明模型在训练集上对正负样本（是否重复问题对）的综合分类效果越来越好，兼顾了精确率和召回率，对训练数据的分类质量持续优化 。**

**验证集（Dev）：F1 在 Epoch 1 为 0.7358 ，Epoch 2 是 0.7607 ，Epoch 3 为 0.7715 ，Epoch 4 是 0.7718 ，Epoch 5 为 0.7679 。虽然前期有小幅度上升，但后期又略有回落，整体在 0.77 上下波动，意味着模型在验证集上对正负样本的综合分类表现没有持续改善，泛化能力存在瓶颈。**

1. **模型性能评价**

**训练集表现:**模型在训练集上的 Loss 持续下降、Acc 和 F1 分数持续上升，到第五轮时，Loss 低至 0.0599 ，Acc 高达 0.9769 ，F1 也达到 0.9687 ，说明模型对训练数据的学习效果很好，能够深度挖掘训练数据中的模式，在训练集上具备很强的分类能力 。

**验证集表现与泛化能力:**验证集上，各项指标没有随着训练轮次增加而稳定提升，尤其是 Loss 后期上升、Acc 和 F1 波动不前甚至略有下降，反映出模型存在过拟合风险，对未见过的新数据（验证集可视为相对训练集的新数据）的泛化能力不足 。可能是模型复杂度较高（比如网络结构、参数设置等方面），或者训练数据和验证数据分布存在差异等原因导致。

**综合评价**

**优势：**模型在训练集上收敛效果好，分类性能出色，证明其有较强的学习和拟合能力，若应用场景是对已知相似分布数据做分类，有一定基础价值 。

**不足：**泛化能力有限，在验证集上表现出明显的过拟合倾向，实际应用到真实新数据时，分类效果可能不稳定、达不到预期 。后续需要优化，比如调整模型结构（简化网络、增加正则化等）、改进数据集（数据增强、重新划分训练验证集等）来提升泛化能力，让模型在更多样化的实际场景中有效工作。

总体而言，模型在训练集上的学习是成功的，但在泛化到新数据方面存在缺陷，需要进一步优化来提升实用性。

1. **总结**

本次实验基于 Spark 开展 QQP 分类文本实验，利用其分布式优势，借 Spark SQL 清洗数据，通过 Spark MLlib 完成 TF - IDF 文本向量化，融合特征构建输入训练轻量级神经网络 。经 5 轮训练，训练集指标优异，验证集 F1 稳定，展现 Spark 处理大规模文本高效性，其生态兼容、适配轻量部署。但存在特征瓶颈（TF - IDF 语义理解弱 ）、过拟合风险（验证集 Loss 后期回升 ）、任务调度延迟问题，后续可结合 Spark NLP 增强语义、优化正则化与集群配置。实验验证 Spark 可行性，为问答平台重复问题治理提供方案，未来可拓展至实时流处理、推动模型生产落地助力智能化升级 。