头条投满分项目实训

任

务

书

|  |  |  |
| --- | --- | --- |
| **课题名称：** | **：** | **热点分类系统** |
| **姓 名：** | **：** | **吕雨阳** |
| **学 号：** | **：** | **7220766213** |
| **专 业：** | **：** | **数据科学与大数据技术** |
| **班 级：** | **：** | **22数据2** |
| **指导老师：** | **：** | **李天映** |

**目 录**

[一、 项目背景 3](#_Toc2807)

[二、 项目需求 4](#_Toc12335)

[三、 项目实现 5](#_Toc18656)

[3.1 项目环境搭建 5](#_Toc6513)

[3.2 项目需求分析 11](#_Toc31936)

[四、 成果展示 12](#_Toc9469)

[五、 结束语 14](#_Toc572)

[六、 参考文献 14](#_Toc17227)

# 项目背景

在当今信息爆炸的时代，互联网上每天都会产生海量的新闻、社交媒体帖子、论坛讨论等内容。这些信息中蕴含着丰富的社会热点、公众情绪和趋势变化。然而，面对如此庞大的信息流，如何高效、准确地识别和分类这些热点内容，使其能够被快速理解、追踪和利用，成为了一个巨大的挑战。

传统的信息分类方法，如基于关键词匹配或简单的机器学习模型，在处理自然语言时往往难以捕捉文本的深层语义和上下文信息，导致分类精度不高，尤其是在面对同义词、一词多义、网络新词或复杂语境时表现不佳。

近年来，深度学习技术，特别是预训练语言模型（如BERT），在自然语言处理（NLP）领域取得了突破性进展。BERT能够通过双向Transformer架构深度理解文本的上下文语义，显著提升了文本表示的质量。同时，FastText作为一种高效的词向量和文本分类工具，尤其擅长处理稀有词和形态丰富的语言，并且计算效率高。而随机森林作为一种稳健的集成学习算法，能够有效处理高维特征、减少过拟合，并提供特征重要性分析。

此外，随着Web技术的发展，前后端分离的架构已成为现代应用开发的主流模式，它能够提升开发效率、增强系统可维护性和可扩展性。

因此，本项目旨在构建一个智能化、高效且用户友好的热点分类系统。该系统利用BERT强大的语义理解能力生成高质量的文本表示，结合FastText进行快速文本特征提取或作为模型集成的一部分，再通过随机森林模型进行最终的分类决策，后端提供稳定可靠的分类服务，并通过前端界面实现用户交互和结果展示，以解决海量信息中热点内容自动识别与分类的难题。

# 项目需求

本项目的总体目标是设计并实现一个能够自动、准确地对互联网文本内容进行热点分类的智能系统。具体需求和目标如下：

**核心功能需求：**

文本输入： 系统应能接收用户输入的文本内容（如新闻标题、文章摘要、社交媒体文本等）。

热点分类： 系统需能够对输入的文本进行自动分类，将其归入预定义的热点类别（例如：0: "finance", 1: "realty", 2: "stocks", 3: "education", 4: "science", 5: "society", 6: "politics", 7: "sports", 8: "game", 9: "entertainment"）。

分类结果输出： 后端模型处理后，需向前端返回清晰的分类结果，包括预测的类别标签和可能的置信度分数。

**模型性能需求：**

高准确性： 模型的核心目标是实现高分类准确率。利用BERT捕捉深层语义，FastText提供补充特征或快速表示，随机森林进行鲁棒性决策，旨在超越传统方法的性能。

良好的泛化能力： 模型应能有效处理未见过的文本，适应语言的变化和新出现的热点话题。

可解释性： 利用随机森林的特性，可尝试分析影响分类决策的关键特征（如关键词），为结果提供一定程度的解释。

**非功能性需求：**

可扩展性：系统设计应便于未来增加新的热点类别或集成其他模型（如模型融合）。

可维护性：代码结构清晰，模块化设计，便于后续更新和维护。

效率：在保证准确率的前提下，优化模型推理速度，提升用户体验。

# 项目实现

3.1 项目环境搭建

1. 基础运行环境

操作系统：Windows 11

Python 版本：3.12

包管理工具：pip 25.2

2. 虚拟环境配置

虚拟环境：(.venv) D:\111huiyu

3. 核心依赖库安装

（1）Web 服务依赖

fastapi==0.104.1：搭建 API 接口的核心框架

uvicorn==0.24.0：FastAPI 的 ASGI 服务器，用于启动 Web 服务

命令：uvicorn main:app --reload

fastapi.middleware.cors：已集成在 FastAPI 中

（2）文本预处理依赖

jieba==0.42.1：中文分词工具

numpy==1.24.4：处理数值计算（如fasttext概率结果转换、random forest特征矩阵），版本需与scikit-learn兼容

（3）模型依赖（BERT、随机森林、FastText）

| 模型 | 依赖库及版本 | 说明 |
| --- | --- | --- |
| 随机森林 | scikit-learn=1.2.2 | 实现随机森林分类器 |
| FastText | fasttext=0.9.2 | 安装fasttext-wheel |
| BERT | transformers=4.30.2 | Hugging Face 库，加载预训练 BERT 模型 |

（4）前端与可视化依赖

1. 核心框架 (Core Frameworks)

**Vue.js (v3.5.20):** 主要的渐进式JavaScript框架，用于构建用户界面。利用其组合式API和响应式系统开发组件化的单页应用（SPA）。

**React (v18.3.1)**: 作为备选或并行的核心框架，提供强大的组件模型和虚拟DOM，支持构建复杂的用户界面。

**Alpine.js (v3.14.9)**: 轻量级库，用于在HTML中直接添加简单的交互行为（如显示/隐藏、表单验证），适用于微交互场景，减少对大型框架的依赖。

2. 用户界面 (UI) 框架与样式

**Bootstrap (v5.3.0)**: 成熟的响应式前端框架，提供网格系统、预设组件（按钮、表单、导航栏、模态框）和基础样式，加速UI开发。

**Tailwind CSS:** 功能优先（utility-first）的CSS框架，提供大量原子化CSS类，允许在HTML中直接组合出高度定制化的设计，与Bootstrap结合可实现快速且个性化的界面。

3. 工具与可视化库 (Utility & Visualization Libraries)

**Axios:**基于Promise的HTTP客户端，负责前端与后端FastAPI/Flask API的通信，处理数据请求（如文本分类）和响应。

**GSAP (GreenSock Animation Platform) (v3.12.5):**高性能JavaScript动画库，用于创建复杂、流畅的动画效果（如结果展示动画、页面过渡），显著提升视觉体验。

**Popper (v2.11.6)**: 用于定位元素的库，常被Bootstrap的下拉菜单、提示框（Tooltips）和弹出框（Popovers）等组件所依赖，确保这些元素能正确地相对于触发元素定位。

4. 字体与图标 (Font Scripts)

**Font Awesome (v6.0.0):** 提供丰富的SVG图标和图标字体。通过CDN引入，为按钮、导航、类别标签等UI元素添加直观、可缩放的视觉符号。

5. 开发与文档工具 (Development & Documentation)

**Swagger UI:** 由后端FastAPI框架自动生成的交互式API文档界面。前端开发者可在此直接查看API端点、参数、响应格式，并进行实时测试，极大简化了前后端联调。

6. 内容分发网络 (CDN)

**Cloudflare, Unpkg, cdnjs, jsDelivr:** 用于托管和分发前端静态资源（如Vue.js, React, Axios, GSAP, Font Awesome, Bootstrap CSS/JS等）。通过全球CDN节点，实现资源的快速加载、减轻服务器负担并提高应用的可用性和性能。

7. 后端服务与运行环境 (Backend Services & Runtime)

**Web 框架:**

FastAPI: Python后端框架，提供高性能API和自动生成的Swagger UI。

Flask (v3.1.3): Python后端微框架，用于构建API服务。

Express: Node.js后端框架，可用于搭建开发服务器或处理代理请求。

**Web 服务器:**

Uvicorn: ASGI服务器，用于运行FastAPI应用（生产或开发环境）。

Flask (v3.1.3): 内置开发服务器。

Express: 可作为Node.js服务器。

**编程语言:**

Node.js: 运行JavaScript/前端工具链（npm/yarn, 构建工具）的环境。

Python (v3.12.8): 运行后端模型和API服务的环境。

8. 模型与数据文件准备

模型文件路径：在MODEL\_DIR（如D:\111huiyu\model）下放置 3 类模型文件：

随机森林：rf\_model.pkl

* **随机森林模型代码**

print("📊 正在加载数据...")  
df = pd.read\_csv(DATA\_PATH)  
# 检查必要列  
if 'sentence' not in df.columns or 'label' not in df.columns:  
 raise ValueError("❌ 数据文件必须包含 'sentence' 和 'label' 列")  
  
# 预处理  
print("🧹 正在预处理文本...")  
df['text\_clean'] = df['sentence'].apply(preprocess)  
  
# 过滤空文本  
before\_count = len(df)  
df = df[df['text\_clean'].str.strip() != '']  
after\_count = len(df)  
print(f"📉 过滤空文本: {before\_count} → {after\_count}")  
  
# ======= 划分训练集（可选：小样本调试） ===  
# df\_sample = df.sample(n=50000, random\_state=42) # 调试用  
# X = df\_sample['text\_clean']  
# y = df\_sample['label']  
  
X = df['text\_clean']  
y = df['label']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.2, random\_state=42, stratify=y  
)  
print(f"✅ 训练集: {len(X\_train)}, 测试集: {len(X\_test)}, 类别数: {y.nunique()}")  
# ======= 构建 Pipeline（推荐做法） =========  
pipeline = Pipeline([  
 ('tfidf', TfidfVectorizer(  
 max\_features=50000, # 控制特征维度  
 ngram\_range=(1, 2), # 使用 unigram + bigram  
 sublinear\_tf=True, # 使用 sublinear scaling: 1 + log(tf)  
 max\_df=0.95, # 忽略出现频率过高（>95%）的词  
 min\_df=3 # 忽略出现次数太少（<3）的词  
 )),  
 ('rf', RandomForestClassifier(  
 n\_estimators=200, # 树的数量  
 max\_depth=20, # 控制过拟合  
 min\_samples\_split=5, # 内部节点再划分所需最小样本数  
 min\_samples\_leaf=2, # 叶子节点最少样本数  
 random\_state=42,  
 n\_jobs=-1, # 使用所有CPU核心  
 verbose=1 # 显示训练进度  
 ))  
])

FastText：fasttext\_model.bin（FastAPI 加载的二进制模型文件，需确保是训练时导出的bin格式）

* **Fasttext模型代码**

print("\n🚀 开始训练 fastText 模型...")  
  
try:  
 model = fasttext.train\_supervised(  
 input=TEMP\_FILE,  
 epoch=25, # 训练轮数  
 lr=1.0, # 学习率  
 wordNgrams=2, # 二元语法，提升效果  
 dim=100, # 词向量维度  
 minCount=1, # 最小词频  
 verbose=2, # 显示详细日志（相当于进度）  
 loss="softmax" # 损失函数  
 )  
  
 # 保存模型  
 model.save\_model(MODEL\_PATH)  
 print(f"\n✅ 模型训练完成！已保存到：\n👉 {MODEL\_PATH}")  
  
 # 可选：打印一些模型信息  
 print(f"📊 模型维度: {model.f.dim}")  
 print(f"📚 词汇量: {model.f.words}")  
 print(f"🏷️ 标签数: {model.f.labels}")  
  
except Exception as e:  
 print(f"❌ 训练失败！错误信息：{e}")  
 raise

BERT：bert\_model文件夹（包含config.json、pytorch\_model.bin、vocab.txt等 Hugging Face 标准模型文件）

* **Bert模型代码**

class TextDataset(Dataset):  
 def \_\_init\_\_(self, texts, labels, tokenizer, max\_len=64): # max\_len从 128 → 64  
 self.texts = texts  
 self.labels = labels  
 self.tokenizer = tokenizer  
 self.max\_len = max\_len  
 def \_\_len\_\_(self):  
 return len(self.texts)  
 def \_\_getitem\_\_(self, idx):  
 text = str(self.texts[idx])  
 label = self.labels[idx]  
 encoding = self.tokenizer(  
 text,  
 truncation=True,  
 padding='max\_length',  
 max\_length=self.max\_len,  
 return\_tensors=None  
 )  
 return {  
 'input\_ids': torch.tensor(encoding['input\_ids'], dtype=torch.long),  
 'attention\_mask': torch.tensor(encoding['attention\_mask'], dtype=torch.long),  
 'labels': torch.tensor(label, dtype=torch.long)  
 }  
  
# ==================== 加载分词器和模型 ====================  
print("🔍 正在加载本地 BERT 模型...")  
try:  
 tokenizer = BertTokenizer.from\_pretrained(BERT\_LOCAL\_PATH)  
 print("✅ 分词器加载成功")  
  
 model = BertForSequenceClassification.from\_pretrained(  
 BERT\_LOCAL\_PATH,  
 num\_labels=num\_labels  
 )  
 print(f"✅ BERT 模型加载成功，类别数: {num\_labels}")  
except Exception as e:  
 print(f"❌ 模型加载失败: {e}")  
 raise  
# ==================== 训练参数（4060 优化版）====================  
print("⚙️ 配置训练参数..."  
training\_args = TrainingArguments(  
 output\_dir=SAVE\_PATH,  
 num\_train\_epochs=1, # 🔥 只训练 1 轮，足够！  
 per\_device\_train\_batch\_size=32, # 🔥 4060 支持大 batch（原 16）  
 warmup\_steps=200, # 少量 warmup  
 weight\_decay=0.01,  
 logging\_dir=os.path.join(SAVE\_PATH, "logs"),  
 logging\_steps=50, # 每 50 步输出 loss  
 save\_steps=1000,  
 save\_total\_limit=2,  
 learning\_rate=2e-5,  
 seed=42,  
 disable\_tqdm=False, # 显示进度条  
 report\_to=[], # 不上报  
 fp16=True, # ⚡ 启用半精度（4060 支持）  
 dataloader\_num\_workers=4, # 加快数据加载  
 remove\_unused\_columns=True,  
 # 注意：旧版不支持 evaluation\_strategy 或 device  
)  
  
# ==================== 构建训练器 ====================  
print("🧠 构建训练器...")  
try:  
 train\_dataset = TextDataset(  
 df['sentence'].astype(str).tolist(),  
 df['label'].tolist(),  
 tokenizer,  
 max\_len=64 # 🔥 与上面保持一致  
 )  
 print(f"✅ 数据集构建完成，共 {len(train\_dataset)} 个样本")  
  
 trainer = Trainer(  
 model=model,  
 args=training\_args,  
 train\_dataset=train\_dataset,  
 )  
 print("✅ 训练器构建成功")  
except Exception as e:  
 print(f"❌ 构建训练器失败: {e}")  
 raise

停用词文件：在STOPWORDS\_PATH（如D:\111huiyu\慧与\课上代码\头条满分\data\stopwords.txt）放置中文停用词表，每行 1 个停用词（如 “的”“了”“在” 等）

9. 硬件环境

GPU 环境：4060 laptop

3.2 项目需求分析

技术架构需求：

后端模型构建：

基于 BERT 模型（如 bert-base-chinese）进行微调，或提取其[CLS]向量作为文本的深度语义表示。

利用 FastText 生成文本的词袋向量或n-gram特征，作为补充特征或独立模型。

使用 随机森林 分类器，接收来自BERT和/或FastText的特征向量，完成最终的多类别分类任务。

实现模型的训练、验证、评估和持久化（保存/加载）。

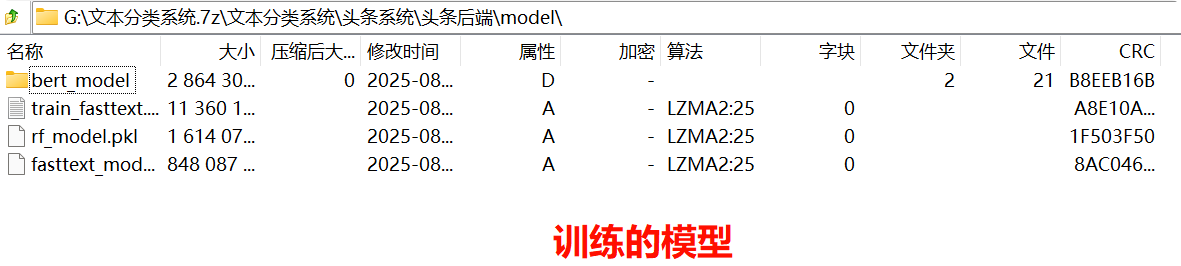
后端服务：

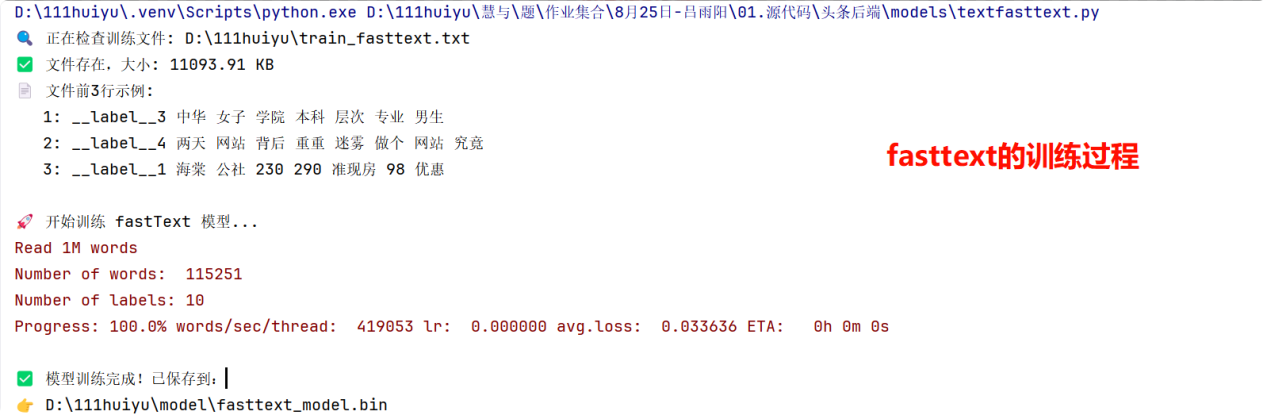
1. 使用Python Web框架（如 Flask 或 FastAPI）搭建RESTful API服务。
2. API提供接收文本、调用模型进行预测、返回JSON格式结果的接口。
3. 确保服务的稳定性、响应速度和一定的并发处理能力。

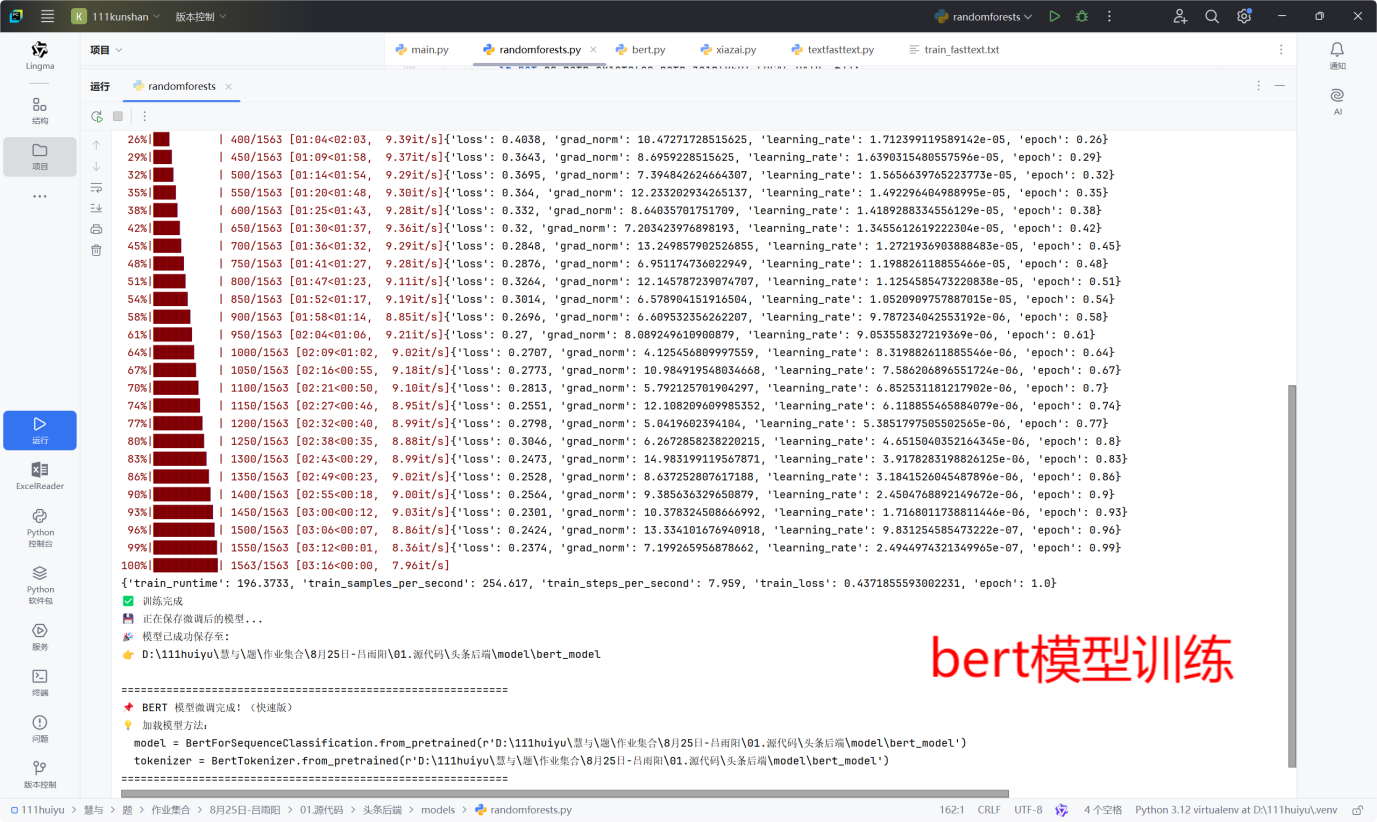
前端接入：

1. 开发一个直观的Web前端界面（可使用 Vue.js, React 或 HTML/CSS/JS）。
2. 前端负责用户输入文本的展示、提交，并调用后端API。
3. 前端清晰地展示分类结果（类别、置信度等），提升用户体验。
4. 实现前后端的跨域通信（CORS）。

# 成果展示









# 结束语

通过此次实训总结与收获通过本次“头条热点分类系统”的实训项目，我不仅完成了一个从概念到可交互原型的完整开发流程，更在实践中深化了对机器学习、前后端协作及用户体验设计的综合理解。

在技术层面，我深入掌握了文本分类的核心流程，从数据预处理、模型选择（随机森林、fastText、BERT）到结果融合（投票共识机制），并成功搭建了前后端交互的桥梁（FastAPI + React）。GSAP动画库的运用，则让静态结果转化为生动的动态可视化，显著提升了用户交互体验。

在能力层面，我锻炼了独立解决问题的能力。无论是调试模型接口的通信问题，还是优化打字机效果与进度条动画的时序配合，都让我学会了如何系统性地分析与攻克技术难点。同时，项目也强化了我的工程化思维，从代码结构设计到UI组件化，都力求清晰与可维护。

更重要的是，这次实训让我深刻体会到理论知识与实际应用的差距，以及持续学习的重要性。面对快速迭代的AI技术，唯有不断实践、不断探索，才能将所学转化为真正的生产力。

# 参考文献

1. 苏红刚. 基于SVM的中文文本分类系统实现[D]. 长春: 吉林大学, 2013.
2. 巩军全. 文本分类算法的研究与改进[D]. 西安: 西安电子科技大学, 2012.
3. 张真. 基于语义相似度的中文文本分类系统的研究与实现[D]. 大连: 大连海事大学, 2008.
4. 通义千问. 通义千问-2技术报告 [EB/OL]. (2023-09-25) [2025-08-31]. https://arxiv.org/abs/2309.17173.