

南開大學

Python 语言程序设计

虚假新闻检测



学院 XXXX

专业 XXXX

姓名 XXXX

学号 XXXX

目录

1 问题描述	3
2 实验目的	3
3 实验过程	4
3.1 TF-IDF + 传统机器学习模型	4
3.1.1 TF-IDF 特征提取	4
3.1.2 模型层（基于 TF-IDF 特征）	5
3.2 Word2Vec + LSTM	6
3.2.1 Word2Vec 特征提取	6
3.2.2 模型层（基于 Word2Vec 特征）	7
3.3 BERT 预训练模型	9
3.3.1 BERT 特征提取	9
3.3.2 模型层（基于 BERT 特征）	10
4 实验结论	13
4.1 运行截图	13
4.2 模型训练与测试表现	13
4.3 综合对比与分析	14
4.3.1 传统机器学习模型与深度学习模型的对比	14
4.3.2 影响因素分析	15
4.3.3 实验结论	16

1 问题描述

本实验所使用的数据集为中文文本分类数据集，具有以下显著特征：

数据规模：训练集包含 3673 个样本，测试集包含 1224 个样本。这个规模对于传统机器学习模型较为充足，但对深度学习模型（尤其是需要大量参数微调的 BERT）相对有限。

文本长度特征：平均文本长度约为 105 个字符，属于中短文本范围。短文本意味着信息密度较高但上下文线索有限，这对模型的语言理解能力提出了更高要求。

类别分布特征：训练集中正负样本分布较为均衡（1740:1933），避免了类别不平衡问题，使模型能够更公平地学习两类样本的特征。

数据集每条记录包含以下字段：id（样本编号）、text（文本内容）以及 label（分类标签）。其中，text 表示待分类的中文文本内容，是本实验的输入字段；label 为目标变量，用于标注文本类别，取值定义如下：

0 表示真新闻（Negative）；

1 表示假新闻（Positive，正样本）。

这些数据集特点直接影响了模型选择策略和性能表现，需要在后续实验中充分考虑。实验的目标是训练模型预测测试集中每个样本属于假新闻（正类）的概率值。

2 实验目的

本实验旨在基于中文文本分类数据集，探索并比较多种文本表示与分类模型在中文文本分类任务中的性能表现。具体目标包括：

模型构建与训练：基于 train.csv 数据集，分别采用传统机器学习与深度学习方法建立分类模型，并利用测试集对模型进行概率预测与评估；

模型优化与改进：通过调整特征提取方式、模型结构与超参数，不断提升预测精度与泛化能力；

概率输出分析：模型不仅需要进行分类预测，更重要的是输出每个样本属于假新闻的概率值，为后续应用提供置信度信息；

普适性探究：在实验结果的基础上，分析文本的语言特征与模型判别机制，探讨适用于中文场景的通用文本分类思路与方法。

最终目标是确定在当前数据条件下性能最优的模型，为后续的中文文本分类研究与应用提供经验参考与方法论支持。

3 实验过程

在本实验中，将采用三种不同的文本表示与分类方法进行比较分析，分别为：

- (1) 基于 TF-IDF 的传统机器学习方法；
- (2) 基于 Word2Vec 的 LSTM 深度学习模型；
- (3) 基于 BERT 的预训练语言模型。

这些方法代表了文本分类技术的三个发展阶段，从传统统计特征到神经网络再到上下文语义建模。

考虑到数据集的特点（3673 个训练样本、平均 105 字符的中文短文本），预期：

- TF-IDF 方法在短文本上可能表现良好，因为短文本的关键词频率特征通常较为明显
- 深度学习方法可能面临训练数据不足的挑战，但预训练模型的迁移学习能力可以弥补这一缺陷
- 不同方法的泛化能力差异在小样本场景下会更加明显

通过对三者的实验过程与结果，可以更系统地理解模型复杂度、语义表示能力以及性能的权衡关系。

3.1 TF-IDF + 传统机器学习模型

3.1.1 TF-IDF 特征提取

TF-IDF (Term Frequency–Inverse Document Frequency) 是一种衡量词项在文档集中的重要性的统计方法。其核心思想是结合词在文档内的出现频率 (TF) 与其在整个语料中的稀有程度 (IDF)：

- **词频 (TF)**：反映某个词在一篇文档中出现的频率；
- **逆文档频率 (IDF)**：衡量该词在整个语料中的普遍程度，越稀有的词权重越高；
- 二者相乘得到词项在文档中的重要性权重。公式为：

$$\text{tfidf}(t, d) = \text{tf}(t, d) \cdot \log\left(\frac{N}{\text{df}(t) + 1}\right) \quad (1)$$

其中 N 为语料中文档总数， $\text{df}(t)$ 为包含词 t 的文档数。

在本实验中，采用 `sklearn.feature_extraction.text.TfidfVectorizer` 对文本标题进行特征提取，得到稀疏的 TF-IDF 特征矩阵，用于后续的传统机器学习模型训练。

对应代码如下：

```

1 vectorizer = TfidfVectorizer(tokenizer=lambda x:
2     list(jieba.cut(x)))
3 x_train_features = vectorizer.fit_transform(x_train)
4 x_test_features = vectorizer.transform(x_test)

```

Python

3.1.2 模型层（基于 TF-IDF 特征）

1. 朴素贝叶斯 (MultinomialNB)

多项式朴素贝叶斯假设词项在类别条件下条件独立，用类别先验 $p(y)$ 与条件概率 $p(x_i|y)$ 计算后验：

$$\hat{y} = \operatorname{argmax}_y p(y) \prod_i p(x_i | y)^{x_i} \quad (2)$$

其中 x_i 可视为词频/权重。NB 在高维稀疏表示上训练/预测极快，常作文本分类强基线。

对应代码如下：

```

1 naive_bayes = MultinomialNB()
2 naive_bayes.fit(x_train_features, y_train)
3 nb_train_probabilities = naive_bayes.predict_proba(x_train_features)[:, 1]
4 nb_auc = roc_auc_score(y_train, nb_train_probabilities)
5 print(f"NB 训练 AUC: {nb_auc:.4f}")
6
7 # 获取假新闻（标签1）的概率
8 nb_probabilities = naive_bayes.predict_proba(x_test_features)[:, 1]
9 submission_nb = pd.DataFrame({"id": test_ids, "prob": nb_probabilities})

```

Python

2. 逻辑回归 (Logistic Regression)

逻辑回归是判别式线性模型，学习线性决策面 $f(x) = w^\top x + b$ ，用逻辑函数建模后验：

$$p(y=1 | x) = \sigma(w^\top x + b), \min_w \Sigma_i \log(1 + \exp(-y_i(w^\top x_i))) + \lambda \|w\|_2^2 \quad (3)$$

在高维稀疏特征（如 TF-IDF）上往往是效果/稳健性的强基线。

对应代码如下（含迭代上限设置）：

```

1 logistic_regression = LogisticRegression(max_iter=1000)
2 logistic_regression.fit(x_train_features, y_train)
3 lr_train_probabilities =
4 logistic_regression.predict_proba(x_train_features)[:, 1]
5 lr_auc = roc_auc_score(y_train, lr_train_probabilities)
6
7 # 获取假新闻（标签1）的概率

```

Python

```

8 lr_probabilities = logistic_regression.predict_proba(x_test_features)[:, 1]
9 submission_lr = pd.DataFrame({"id": test_ids, "prob": lr_probabilities})

```

3. 随机森林 (Random Forest)

随机森林是 Bagging+特征随机子采样的树模型集成：对自助采样子集训练多棵决策树，预测时多数投票。其优势是非线性建模与特征交互能力，不易过拟合单棵树的噪声。

对应代码如下（100 棵树）：

```

1 random_forest = RandomForestClassifier(n_estimators=100) Python
2 random_forest.fit(x_train_features, y_train)
3 rf_train_probabilities = random_forest.predict_proba(x_train_features)[:, 1]
4 rf_auc = roc_auc_score(y_train, rf_train_probabilities)
5 print(f"RF 训练 AUC: {rf_auc:.4f}")
6
7 # 获取假新闻（标签1）的概率
8 rf_probabilities = random_forest.predict_proba(x_test_features)[:, 1]
9 submission_rf = pd.DataFrame({"id": test_ids, "prob": rf_probabilities})

```

3.2 Word2Vec + LSTM

3.2.1 Word2Vec 特征提取

Word2Vec 是一种基于神经网络的词向量表示方法，用于将离散的词语映射到连续稠密空间中，使语义相近的词在向量空间中距离更近。与 TF-IDF 的稀疏表示不同，Word2Vec 能够捕捉词语间的上下文关系与语义信息，从而为后续深度模型提供更具表达力的输入特征。

Word2Vec 的核心思想是通过上下文预测实现语义学习。常用的 Skip-gram 模型目标函数为：

$$\max \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j} | w_t) \quad (4)$$

其中 c 为窗口大小， w_t 为中心词。条件概率定义为：

$$P(w_o | w_i) = \frac{\exp(v'_{w_o} \cdot v_{w_i})}{\sum_{w=1}^V \exp(v'_{w_o} \cdot v_{w_i})} \quad (5)$$

本实验采用公开中文预训练词向量模型 sgns.sogou.char（由 Sogou 语料训练），其维度为 300。加载后，对训练集与测试集标题文本进行分词与向量索引映射，构建模型可用的词表与嵌入矩阵。下载地址：[GitHub 主页 https://github.com/Embedding/Chinese-](https://github.com/Embedding/Chinese-)

Word-Vectors，百度网盘直链 <https://pan.baidu.com/s/1pUqyn7mnPcUmzxT64gGpSw>；
本地路径 Word2Vec/sgns.sogou.char。

对应代码如下：

```

1 word2vec = KeyedVectors.load_word2vec_format("./Word2Vec/
2 sgns.sogou.char", binary=False) Python
3
4 vocab = [word for word in set(word for seq in train_tokens for word in
5 seq)
6         if word in word2vec.key_to_index]
7 word2idx = {"<PAD>": 0, "<UNK>": 1}
8 word2idx.update({word: idx + 2 for idx, word in enumerate(vocab)})
9
10 embedding_matrix = np.zeros((len(word2idx), embedding_dim), np.float32)
11 vector_pool = []
12 for word, idx in word2idx.items():
13     if idx < 2:
14         continue
15     if word in word2vec.key_to_index:
16         embedding_matrix[idx] = word2vec[word]
17         vector_pool.append(word2vec[word])
18 if vector_pool:
19     embedding_matrix[1] = np.mean(vector_pool, axis=0)

```

通过上述步骤，文本被转换为由 Word2Vec 词向量构成的稠密矩阵，为 LSTM 模型的语义学习提供输入特征。

3.2.2 模型层（基于 Word2Vec 特征）

为充分利用词向量的上下文依赖关系，本实验采用双向 LSTM（Bidirectional Long Short-Term Memory）网络作为分类器。LSTM 能够在时间维度上捕捉长距离依赖，其门控机制有效缓解了传统 RNN 的梯度消失问题。双向结构可同时整合前向与后向语义信息，提升模型的上下文建模能力。

LSTM 的内部结构如下图所示：

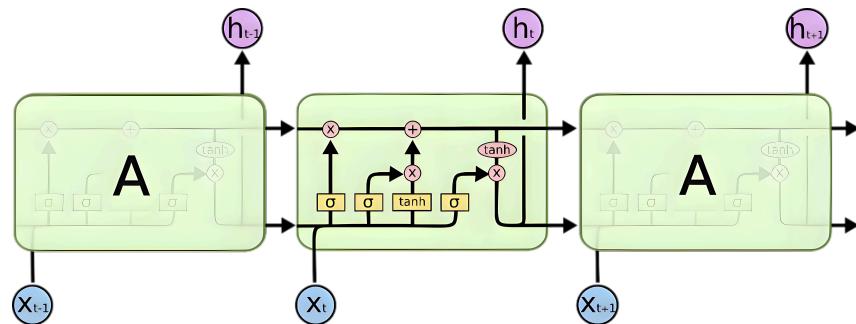


图 1: LSTM 网络结构示意图

图中展示了 LSTM 单元的关键门控结构，包括遗忘门（Forget Gate）、输入门（Input Gate）与输出门（Output Gate）。其更新过程如下所示：

- $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$ (遗忘门)
- $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ (输入门)
- $\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$ (候选记忆)
- $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$ (记忆更新)
- $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$ (输出门)
- $h_t = o_t \odot \tanh(c_t)$ (输出状态)

模型整体结构如下：

- 输入层：由 Word2Vec 生成的词向量序列；
 - 双向 LSTM 层：提取文本的上下文依赖特征；
 - 平均池化层：聚合时序信息形成全局语义向量；
 - 全连接层 + Softmax：输出真假新闻分类结果。

对应代码实现如下：

```
1 class LSTMCls(nn.Module):  
2     def __init__(self, vocab_size, embed_dim, hidden_dim, num_classes,  
3                  embedding_matrix, num_layers=2, dropout_prob=0.3,  
4                  freeze=False):  
5         super().__init__()  
6         self.embedding = nn.Embedding(vocab_size, embed_dim,  
7                                         padding_idx=0)  
8         self.embedding.weight.data.copy_(torch.from_numpy(embedding_matrix))  
9         self.embedding.weight.requires_grad = not freeze  
10        self.lstm = nn.LSTM(embed_dim, hidden_dim, num_layers=num_layers,  
11                            batch_first=True, dropout=dropout_prob,  
12                            bidirectional=True)  
13        self.fc = nn.Linear(hidden_dim * 2, num_classes)  
14        self.drop = nn.Dropout(dropout_prob)
```

```

13     def forward(self, input_ids):
14         mask = (input_ids != 0).float().unsqueeze(-1)
15         lstm_output, _ = self.lstm(self.embedding(input_ids))
16         lstm_output = lstm_output * mask
17         pooled_output = lstm_output.sum(1) / mask.sum(1).clamp_min(1)
18         return self.fc(self.drop(pooled_output))

```

训练过程中采用 CrossEntropyLoss 作为损失函数，优化器为 Adam，学习率 5e-4，批大小 32，训练 10 轮。

训练代码如下：

```

1  criterion = nn.CrossEntropyLoss() Python
2  optimizer = Adam(model.parameters(), lr=5e-4)
3  for epoch in range(epochs):
4      model.train()
5      probabilities, true_labels = [], []
6      for batch_x, batch_y in train_loader:
7
8          # 获取假新闻（标签1）的概率
9          probs = torch.softmax(logits, dim=-1)[:, 1]
10         probabilities.extend(probs.detach().cpu().numpy())
11         true_labels.extend(batch_y.cpu().numpy())
12
13     train_auc = roc_auc_score(true_labels, probabilities)
14     print(f"Training AUC: {train_auc:.4f}")
15
16 # 预测阶段获取假新闻概率
17 model.eval()
18 probabilities = []
19 with torch.no_grad():
20     for (batch_x,) in test_loader:
21         batch_x = batch_x.to(device)
22         logits = model(batch_x)
23         probs = torch.softmax(logits, dim=-1)[:, 1]
24         probabilities.extend(probs.cpu().numpy())

```

3.3 BERT 预训练模型

3.3.1 BERT 特征提取

BERT (Bidirectional Encoder Representations from Transformers) 是由 Google 提出的基于 Transformer 的预训练语言模型。与传统的 Word2Vec 不同，BERT 不仅学习静

态词向量，而是利用上下文双向编码机制 动态生成语义表示，从而能捕捉句子中复杂的上下文依赖关系。

BERT 的核心结构基于 Transformer Encoder 堆叠而成，每一层都包含多头自注意力 (Multi-Head Self-Attention) 与前馈神经网络结构。其输入由三种嵌入向量构成：

$$E = E_{\text{token}} + E_{\text{segment}} + E_{\text{position}} \quad (6)$$

其中：

- E_{token} : 词嵌入，表示每个词的语义信息；
- E_{segment} : 区分句子 A 与句子 B；
- E_{position} : 位置编码，提供序列顺序信息。

BERT 的预训练过程包括两大任务：

- **Masked Language Model (MLM)**: 随机掩盖输入序列中的部分词汇，通过上下文预测被掩盖的词；
- **Next Sentence Prediction (NSP)**: 预测两句是否为连续句，以学习句间关系。

预训练完成后，BERT 可通过在下游任务（如文本分类、情感分析、虚假新闻检测）上添加轻量的分类层进行微调 (Fine-tuning)，从而实现端到端的优化。

在本实验中，采用中文预训练模型 `hfl/chinese-roberta-wwm-ext`，该模型在大规模中文语料上进行 Whole Word Masking 预训练，语义理解能力强，适用于中文短文本分类任务。

对应代码如下：

```
1 tokenizer = BertTokenizer.from_pretrained('hfl/chinese-roberta-
2 wwm-ext')
3
4 def encode(texts):
5     return tokenizer(texts.tolist(), add_special_tokens=True,
6                     max_length=max_len,
7                     padding='max_length', truncation=True,
8                     return_attention_mask=True, return_tensors='pt')
```

在编码阶段，BERT 将每个句子转化为固定长度的输入向量 (`input_ids`、`attention_mask`)，输入 Transformer 进行特征提取。

3.3.2 模型层 (基于 BERT 特征)

BERT 模型的核心由多层 Transformer Encoder 构成，每层包括自注意力机制与前馈网络，结构如下图所示：

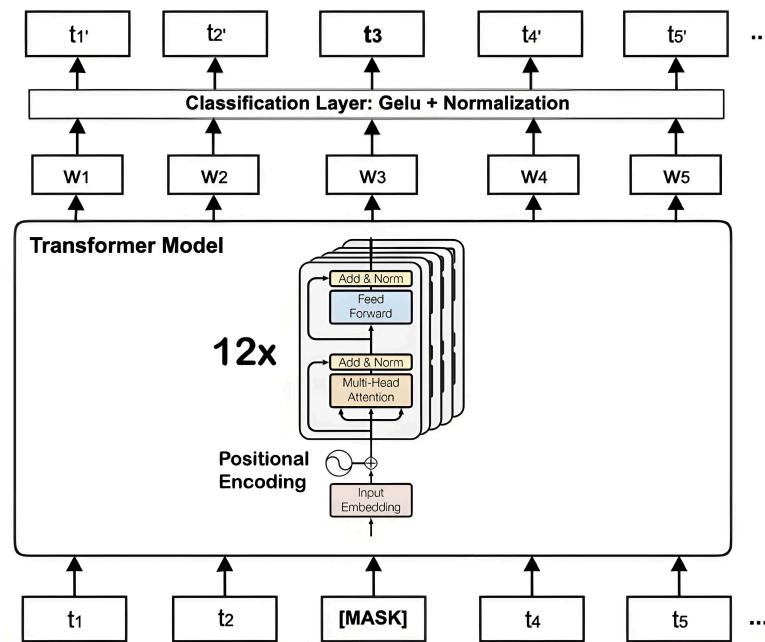


图 2: BERT 网络结构示意图

在文本分类任务中，BERT 通常在句首位置引入特殊符号 [CLS]，其最终隐藏状态向量 $h_{[CLS]}$ 被视为整句的语义表示，用于下游分类。通过在 $h_{[CLS]}$ 上添加一个全连接层并使用 Softmax 输出类别概率，可实现文本的二分类任务。模型不仅输出预测类别，更重要的是提供每个样本属于正类的概率值。

对应代码实现如下：

```
1 model = BertForSequenceClassification.from_pretrained(
2     'hfl/chinese-roberta-wwm-ext',
3     num_labels=2
4 ).to(device)
5 optimizer = AdamW(model.parameters(), lr=1e-5, weight_decay=0.01)
```

训练过程中，模型对 `input_ids` 与 `attention_mask` 进行前向传播，通过 `CrossEntropyLoss` 计算分类损失并反向传播优化参数。每一轮训练输出当前 AUC 值，以监控模型收敛情况：

```
1 for epoch in range(epochs):
2     model.train()
3     train_probabilities, train_true_labels = [], []
4     for batch in train_loader:
5         input_ids, attention_mask, labels = [item.to(device) for item in
6         batch]
7         outputs = model(input_ids=input_ids,
8                         attention_mask=attention_mask, labels=labels)
9         loss = outputs.loss
10        loss.backward()
```

```
9         optimizer.step()
10
11     # 获取假新闻（标签1）的概率
12     probs = torch.softmax(outputs.logits, dim=-1)[:, 1]
13     train_probabilities.extend(probs.detach().cpu().numpy())
14     train_true_labels.extend(labels.cpu().numpy())
15
16     train_auc = roc_auc_score(train_true_labels, train_probabilities)
17     print(f'Training AUC: {train_auc:.4f}')
18
19 # 预测阶段获取假新闻概率
20 model.eval()
21 probabilities = []
22 with torch.no_grad():
23     for batch in test_loader:
24         input_ids, attention_mask = [item.to(device) for item in batch]
25         outputs = model(input_ids=input_ids,
26                         attention_mask=attention_mask)
26         probs = torch.softmax(outputs.logits, dim=-1)[:, 1]
27         probabilities.extend(probs.cpu().numpy())
```

4 实验结论

4.1 运行截图

```
(Python) PS E:\Project\Fake-news-detection> python ML_model.py
E:\miniconda3\envs\Python\Lib\site-packages\jieba\_compat.py:18: UserWarning: pkg_resources is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html. Th
e pkg_resources package is slated for removal as early as 2025-11-30. Refrain from using this package or pin to Setuptools<81.
  import pkg_resources
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\YEJIAX-1\AppData\Local\Temp\jieba.cache
Loading model cost 0.376 seconds.
Prefix dict has been built successfully.
NB 训练 AUC: 0.9972
保存: ./data/prediction_NB.csv
LR 训练 AUC: 0.9899
保存: ./data/prediction_LR.csv
RF 训练 AUC: 1.0000
保存: ./data/prediction_RF.csv
```

图 3: ML 运行截图

```
(Python) PS E:\Project\Fake-news-detection> python LSTM_model.py
E:\miniconda3\envs\Python\Lib\site-packages\jieba\_compat.py:18: UserWarning: pkg_resources is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html. Th
e pkg_resources package is slated for removal as early as 2025-11-30. Refrain from using this package or pin to Setuptools<81.
  import pkg_resources
设备: cuda (NVIDIA GeForce RTX 5060 Laptop GPU)
最大序列长度: 129
Epoch 1/10 - Training AUC: 0.8058
Epoch 2/10 - Training AUC: 0.9560
Epoch 3/10 - Training AUC: 0.9791
Epoch 4/10 - Training AUC: 0.9944
Epoch 5/10 - Training AUC: 0.9992
Epoch 6/10 - Training AUC: 0.9999
Epoch 7/10 - Training AUC: 0.9999
Epoch 8/10 - Training AUC: 0.9999
Epoch 9/10 - Training AUC: 0.9999
Epoch 10/10 - Training AUC: 1.0000
保存: ./data/prediction_LSTM.csv
```

图 4: LSTM 运行截图

```
(Python) PS E:\Project\Fake-news-detection> python BERT_model.py
设备: cuda (NVIDIA GeForce RTX 5060 Laptop GPU)
最大序列长度: 173
Train 1/10: 100%|██████████| 115/115 [00:48<00:00,  2.38it/s]
Epoch 1/10 - Training AUC: 0.9020
Train 2/10: 100%|██████████| 115/115 [00:48<00:00,  2.38it/s]
Epoch 2/10 - Training AUC: 0.9809
Train 3/10: 100%|██████████| 115/115 [00:48<00:00,  2.35it/s]
Epoch 3/10 - Training AUC: 0.9942
Train 4/10: 100%|██████████| 115/115 [00:47<00:00,  2.40it/s]
Epoch 4/10 - Training AUC: 0.9988
Train 5/10: 100%|██████████| 115/115 [00:47<00:00,  2.43it/s]
Epoch 5/10 - Training AUC: 0.9999
Train 6/10: 100%|██████████| 115/115 [00:47<00:00,  2.40it/s]
Epoch 6/10 - Training AUC: 0.9999
Train 7/10: 100%|██████████| 115/115 [00:47<00:00,  2.45it/s]
Epoch 7/10 - Training AUC: 1.0000
Train 8/10: 100%|██████████| 115/115 [00:47<00:00,  2.43it/s]
Epoch 8/10 - Training AUC: 1.0000
Train 9/10: 100%|██████████| 115/115 [00:47<00:00,  2.44it/s]
Epoch 9/10 - Training AUC: 1.0000
Train 10/10: 100%|██████████| 115/115 [00:48<00:00,  2.39it/s]
Epoch 10/10 - Training AUC: 1.0000
保存: ./data/prediction_BERT.csv
```

图 5: BERT 运行截图

4.2 模型训练与测试表现

通过 DC 竞赛平台提交测试集预测结果获取了各模型的测试集 AUC 评分。各模型在训练集上的表现通过本地运行得到，测试集上的评分通过 DC 竞赛平台反馈得到。

模型名称	朴素贝叶斯	逻辑回归	随机森林	LSTM	BERT
训练集 AUC	0.9972	0.9890	1.0000	1.0000	1.0000
测试集 AUC	0.957572	0.934179	0.931251	0.964919	0.987987

4.3 综合对比与分析

从实验结果来看，五个模型在训练集上的表现都非常好，AUC 值基本都达到了 0.99 以上甚至满分。但在测试集上的表现却出现了明显的差异，这说明不同模型的泛化能力存在较大区别。按照测试集 AUC 从高到低排序：**BERT (0.987987) > LSTM (0.964919) > 朴素贝叶斯 (0.957572) > 逻辑回归 (0.934179) > 随机森林 (0.931251)**。

值得注意的是，传统机器学习模型的表现比预期要好得多，三个基于 TF-IDF 的传统模型测试集 AUC 都达到了 0.93 以上，其中朴素贝叶斯甚至达到了 0.957572。这说明在当前的虚假新闻检测任务中，即使使用简单的特征表示方法，只要模型选择合适，也能取得不错的效果。

4.3.1 传统机器学习模型与深度学习模型的对比

在传统机器学习模型中，三个基于 TF-IDF 特征的模型表现存在明显差异。

1. 朴素贝叶斯

测试集 AUC 最高，达到了 0.957572，远超其他两个传统机器学习模型，甚至接近了 LSTM 的表现。这个结果说明朴素贝叶斯在虚假新闻检测任务中具有很强的适应性。这一优异表现充分体现了朴素贝叶斯与当前数据集特点的优异匹配。虽然朴素贝叶斯假设特征之间相互独立，但在高维稀疏的 TF-IDF 特征场景下，这个“错误”的假设反而起到了正则化作用，避免了过拟合。模型简单、参数少，在训练数据有限时不容易学到噪声，因此泛化能力突出。

2. 逻辑回归

作为线性模型也具有较好的泛化能力，测试集 AUC 达到了 0.934179，说明线性模型在文本分类中依然有效。逻辑回归的表现仅次于朴素贝叶斯，两者差距不大，说明在这个任务中，线性决策边界已经能够较好地分离真假新闻。

3. 随机森林

表现略低于逻辑回归，测试集 AUC 为 0.931251。随机森林的表现相当不错，仅比逻辑回归低 0.003。这表明在本任务中，虽然随机森林能够捕捉非线性关系，但简单的线性模型或概率模型已经足够，复杂的集成结构没有带来明显的性能提升。

深度学习模型的表现依然优于传统方法，但优势没有之前分析中那么明显。值得注意的是，朴素贝叶斯的表现（0.957572）已经非常接近 LSTM（0.964919），两者差距仅为 0.007。这说明对于当前数据集下的虚假新闻检测任务，选择合适的传统机器学习模型也能接近深度学习模型的效果。

4. LSTM+Word2Vec

LSTM 模型使用预训练的 Word2Vec 词向量，包含了丰富的语义信息，能捕捉词与词之间的语义相似性。双向 LSTM 结构同时考虑前后文信息，对理解完整语义很重要。虽然训练集 AUC 也达到满分，但测试集表现依然很好（0.964919），说明模型学到了深层语义特征而非简单记忆。

5. BERT

BERT 模型的表现最为出色，测试集 AUC 达到 0.987987。BERT 在海量中文语料上预训练，已经学习到丰富的语言知识。与 Word2Vec 的静态词向量不同，BERT 生成动态的上下文相关词向量，同一个词在不同句子中有不同表示。Transformer 的自注意力机制能捕捉任意两个词之间的关系，不受距离限制，比 LSTM 更灵活强大。

4.3.2 影响因素分析

值得注意的是，所有模型在训练集上表现都很好（ $AUC \geq 0.998$ ），但测试集表现差异很大。这反映出训练集相对简单，样本区分度明显，模型容易达到高准确率。而测试集可能与训练集分布有一定差异，包含更多难以区分的样本，这时候模型的泛化能力就显得尤为重要。传统模型（尤其是随机森林）过拟合严重，而深度学习模型（尤其是 BERT）展现出了更好的泛化能力。

数据集特点在整个实验过程中起到了决定性作用：

数据规模的影响：3673 个训练样本的规模创造了一个有趣的实验环境——足够让传统机器学习模型充分学习，但对深度学习模型来说相对有限。这一规模特点解释了为什么朴素贝叶斯等简单模型能够表现出色：它们能在有限数据下有效估计参数，而复杂的深度学习模型如果没有预训练的帮助可能会过拟合。1224 个测试样本的规模为模型评估提供了可靠的统计基础。

短文本特性的影响：平均 105 字符的文本长度深刻影响了各模型的表现。短文本使 TF-IDF 的关键词频率特征特别有效，因为假新闻（标签 1）往往通过特定词汇（如“震惊”、“爆料”等）来吸引注意，这些词汇在 TF-IDF 表示下会获得较高权重，便于模型识别。同时，短文本的上下文有限但关键，这解释了为什么 BERT 的动态上下文理解能带来显著提升——它能捕捉词语在具体语境中的微妙含义，区分真假新闻之间细微的语言差异。对于 LSTM 来说，这个长度既能发挥序列建模优势，又避免了长序列的计算难题。

类别均衡的优势：1740:1933 的正负样本比例（假新闻:真新闻）避免了类别不平衡问题，使所有模型都能在没有额外采样策略的情况下公平学习真假新闻的特征。这种均衡分布让模型性能比较更加纯粹，反映了算法本身的差异而非数据偏差的影响。

这些数据特点共同塑造了实验结果：在小规模短文本场景下，合适的方法（朴素贝叶斯）能接近深度学习效果，而预训练技术弥补了深度学习对数据量的需求。

从模型复杂度与性能的角度来看，不同模型各有特点。朴素贝叶斯最简单，训练速度快，测试集 AUC 达到 0.957572，性价比极高。逻辑回归同样简单高效，测试集 AUC 达到 0.934179，也是一个很好的基线选择。随机森林计算开销稍大，但测试集 AUC 达到 0.931251，表现稳健。LSTM 需要预训练词向量和 GPU 加速，复杂度适中，测试集 AUC 达 0.964919，在性能和计算成本间取得较好平衡。BERT 最复杂，参数量上亿，训练和推理都需要较强计算资源，但性能也是最好的，测试集 AUC 接近 0.99。

4.3.3 实验结论

通过对五种不同模型的对比实验，本实验得出了以下结论。**BERT 模型在测试集上取得了最好的表现（AUC = 0.987987）**，这得益于其在海量语料上的预训练以及 Transformer 架构的强大上下文理解能力。然而，传统机器学习模型的表现远超预期，朴素贝叶斯达到了 0.957572，逻辑回归达到了 0.934179，随机森林达到了 0.931251，这三个模型的测试集 AUC 都在 0.93 以上。尤其是朴素贝叶斯，其表现已经非常接近 LSTM（0.964919），仅相差 0.007。

实验还发现模型复杂度与泛化能力并非简单的正相关关系。朴素贝叶斯这个最简单的模型在所有五个模型中表现排第三（AUC = 0.957572），仅次于两个深度学习模型，其性价比极高。这说明在识别虚假新闻的任务中，当训练数据有限、特征维度较高时，简单模型的正则化效果反而能带来更好的泛化能力——假新闻往往通过特定的语言模式表达，朴素贝叶斯能够有效捕捉这些模式而不被噪声干扰。模型的选择需要考虑数据特点，而不是一味追求复杂度。

本实验还验证了预训练技术对小样本场景的重要性。训练集仅包含 3673 个样本，对于深度学习模型来说是比较少的，但 BERT 和 LSTM 都使用了在大规模语料上预训练的模型（分别是预训练语言模型和 Word2Vec 词向量），这使得它们即使在小样本场景下也能取得优异表现，验证了迁移学习和预训练技术的价值。特别是在短文本场景下，预训练模型提供的丰富语言知识能够帮助模型快速捕捉到关键的语言模式和语义线索，这是仅靠 3673 个样本难以学到的。

从文本表示方法的角度来看，从 TF-IDF 的统计特征，到 Word2Vec 的静态词向量，再到 BERT 的动态上下文表示，文本表示方法的演进确实带来了性能的提升。然而，本实验的一个重要发现是：TF-IDF 配合合适的模型（如朴素贝叶斯）也能取得接近深度学习模型的效果。这说明对于当前数据集下的虚假新闻检测任务，文本的统计特征（如关键词频率）可能已经包含了足够多的判别信息。虽然 BERT 的动态上下文表示带来了最好的性能，但其提升幅度（相比朴素贝叶斯约为 0.03）相对于其巨大的计算开销来说，性价比是否最优需要根据具体应用场景来权衡。