哈尔滨工业大学深圳校区

软件开发与实践 B

项目报告

题　　目　重大突发公共卫生事件网络舆情态势感知与谣言检测系统

组　　员　刘静婷、贾琳涵、丁怡文、王铭、严心遥、任鹏宇

学　　院　计算机科学与技术学院

专　　业　计算机类

指 导 教 师　徐睿峰

日　　期　2022/10/30

目 录

1 课题背景及研究的目的和意义

1.1 课题背景

舆情，指因变事项发生、发展和变化过程中，民众所持有的社会态度。网络舆情，尤其与公共卫生事件等相关的，往往引发社会各界的广泛关注。随着基于实时性、互动式的社交媒体的迅速发展，各类社交媒体，例如微博、知乎、抖音等，已经逐渐成为传播热点事件、度量网络舆情的媒介。在信息的传播不断加速的同时，越来越多的信息真伪难辨，这些社交媒体也不免沦为了谣言的温床。

以新冠肺炎疫情为代表，自 2020 年初新型冠状病毒疫情发生以来，网络上对于疫情的讨论度一直都占据微博热搜榜前列，甚至出现了不少与事实不符、只为博人眼球的小道消息。这些虚假消息在一定程度上对群众进行了非正确对引导，激起了民众的负面情绪。虽然，捏造和恶意传播虚假消息的人终将受到法律的制裁，但通过正确、真实的信息去引导大众不信谣、不传谣也是至关重要的手段。

1.2 研究的目的和意义

本课题以社交媒体中最大的信息集散地——微博为例，通过感知网络舆情态势并展示正确信息来向公众进行及时有效的情绪疏导和信息传达。

微博评论、帖子等内容是用户对疫情中介性事件的认知、态度、倾向和行为的汇集，为基于用户情感分析的舆情演化研究提供了高现势性和高时序性的文本语料。因此，分析微博内容可以帮助政府及时了解舆论动向和投放正确信息。以新冠肺炎疫情为例，本课题通过大数据来发现重大突发公共卫生事件的线索，例如民众对于热点事件的情感倾向，不同话题的舆论导向等，并分析、筛选和投放正确信息，以此来从根本上减少甚至阻断虚假信息和负面舆论的传播，降低疫情带给民众心理上的负面影响，帮助营造一个积极向上的社会氛围。

综上所述，如何网络舆情中精准、全面地挖掘分析用户关注的话题与情感，对公共卫生事件对谣言进行及时的治理和阻断，是了解社情民意、提升信息公信力、正确引导民众思想、营造积极社会氛围的重要渠道，对突发公共卫生事件对防控具有不可否认的重要意义。

2 软件系统设计与实现

2.1 软件系统设计

本系统通过数据爬虫，从以微博为例的社交媒体中搜索发现重大突发公共卫生事件的线索，分析相关事件的热点话题与用户情感，对相关分析结果进行可视化。此外，对由谣言、虚假信息引发的负面情感进行针对性的正确信息投放，并将相关结果通过前端网页进行可视化展示。本系统的结构如图 2-1 所示。

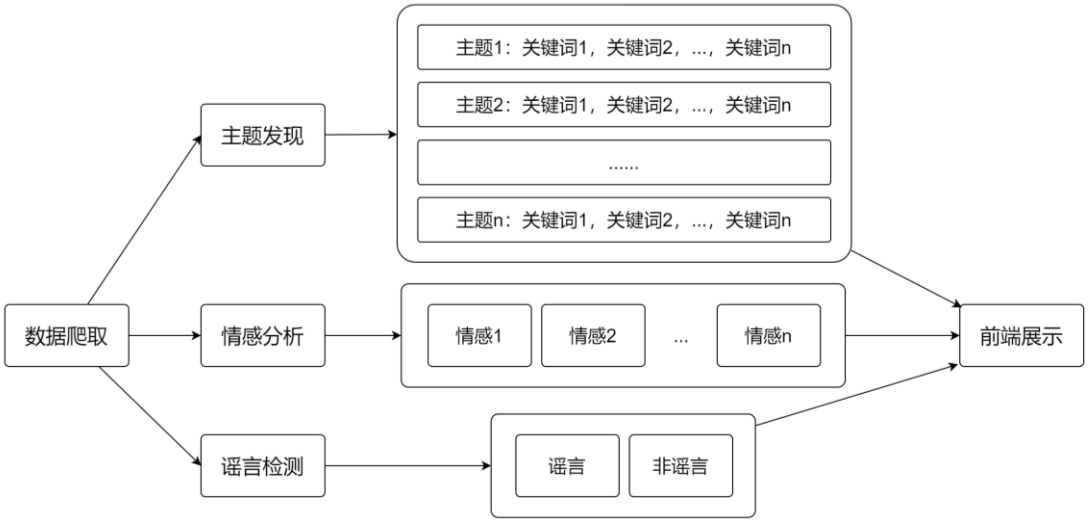

图 2-1 系统结构图

2.1.1 数据爬虫

为了探究社会热点，本系统从微博热搜榜上收集含有“#新冠#”“#疫情#”、“#肺炎#”等关键字的重大突发公共卫生事件话题——通过正则表达式匹配与突发卫生事件相关的热门话题，爬取对应的话题、正文、时间、评论数，并将爬取的字段存储为数据集，如图 2-2 所示。

index	content	topics	reposts_count	comments_count	attitudes_count

图 2-2 数据集格式图

2.1.2 主题分析

本系统利用微博中的帖子、评论等数据，利用主题模型从文本语料中挖掘潜在主题、展示隐藏语义，并深入探索“主题-文档-词汇”三者的关系，可视化绘制每个主题下对应关键字的权重，并展示多个主题间的相关性。

在主题发现任务中，主要尝试了 KMeans、LDA 以及神经主题模型三种方法，

最终选用神经主题模型。

(1) 基于 KMeans 文本聚类的方法

KMeans 文本聚类的基本流程如图 2-3 所示，首先对输入的文本进行预处理并分词，然后使用 TF-IDF 构造词向量，最终使用 KMeans 算法得到聚类的最终结果。其中 KMeans 簇的个数通过轮廓系数决定。

(2) 基于 LDA 主题分析模型的方法

基本流程如图 2-4 所示，与 KMeans 文本聚类方法类似，首先对输入的文本进行预处理并分词，然后使用 TF-IDF 构造词向量，最终使用 LDA 模型得到最终的主题分析结果。

(3) 基于 AVITM 神经网络模型的方法

本系统选择使用 Github 上开源的 Autoencoding Variational Inference for Topic Models(AVITM)神经主题模型[1]。该模型的基本流程及框架如图 2-5、2-6 所示。首先对输入的文本进行预处理并分词，将处理后的文本转换成向量表示并使用 AVITM 模型得到每个主题对应的关键词以及概率矩阵。利用主题概率矩阵计算每个文本所属主题，完成划分。

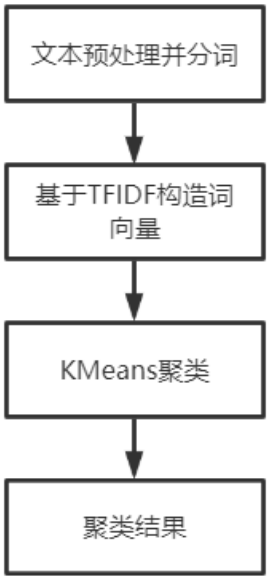

图 2-3 Keans 模型流程图

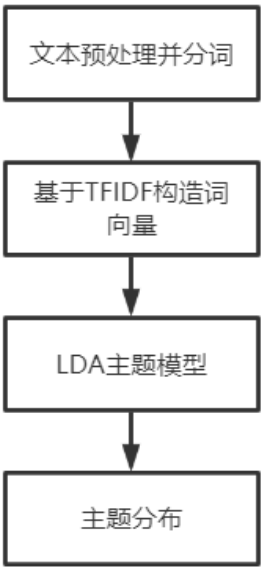

图 2-4 LDA 模型流程图

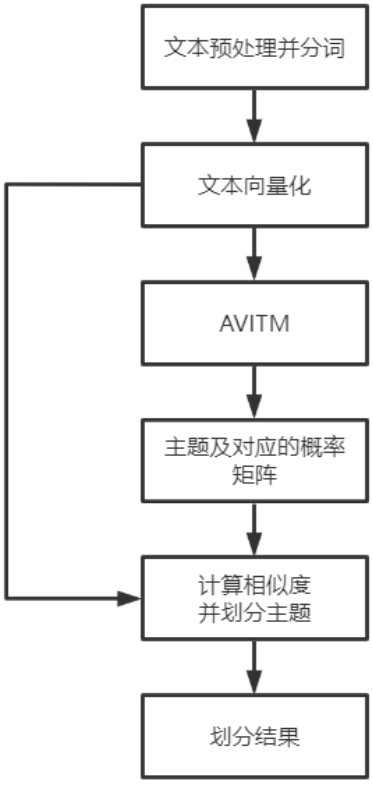

图 2-5 AVITM 模型流程图

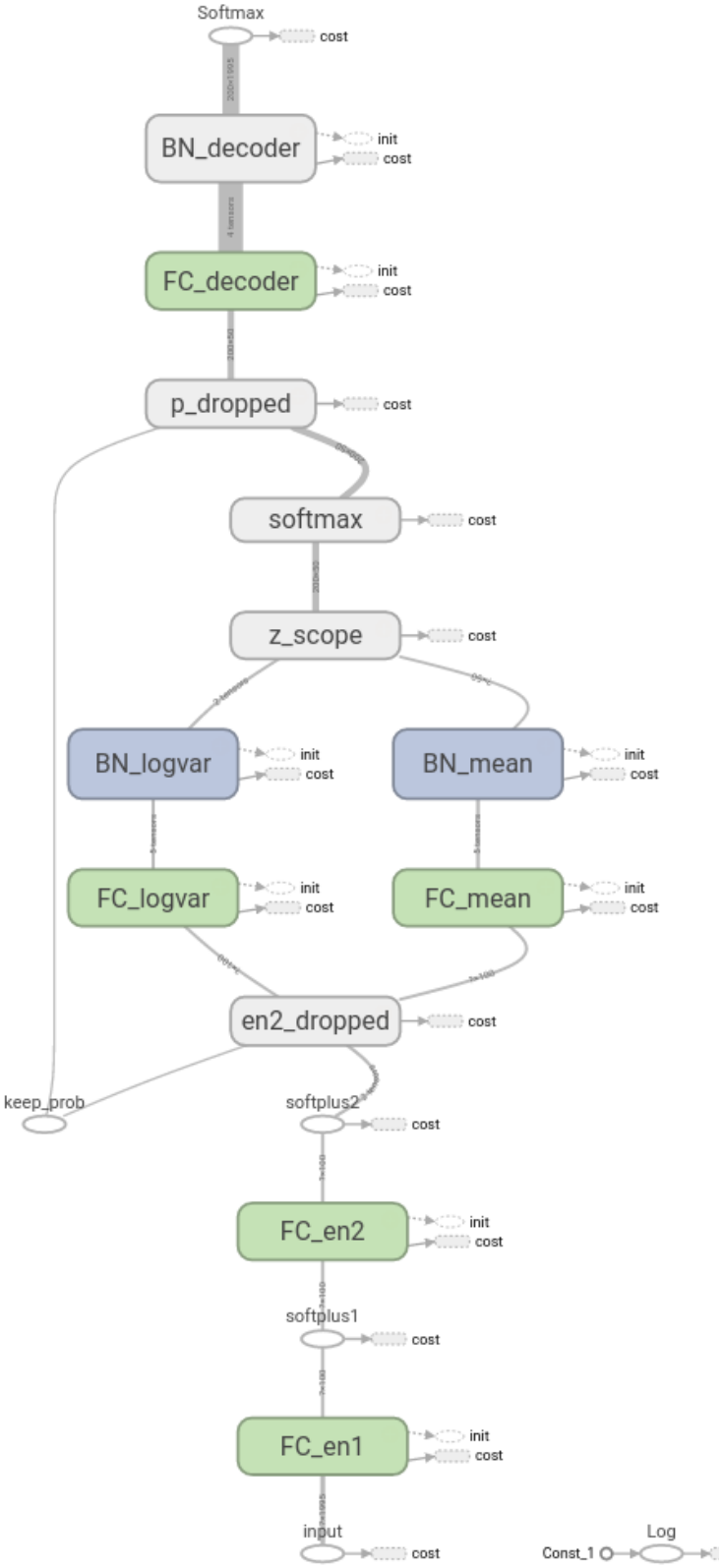

图 2-6 AVITM 模型架构图

2.1.3 情感分析

本系统通过分析微博文本内容，探索文本中蕴含的情感。该模块的输入是一条微博，输出是该微博所蕴含的情感类别。将微博按照其蕴含的情绪分为以下 6 个类别之一： positive、neutral、surprise、sad、fear 及 angry。

在情感分析任务中，使用 Roberta-BiLSTM-FC 的架构，其基本流程如图 2-7 所示。首先将文本预处理并转换为 Roberta 的输入格式，将处理后的结果送入 Roberta 中得到其特征向量表示，并用 LSTM 模型分析其中信息，最后用 LSTM 最后一层输出的隐状态作为输入通过全连接层进行情感分类。

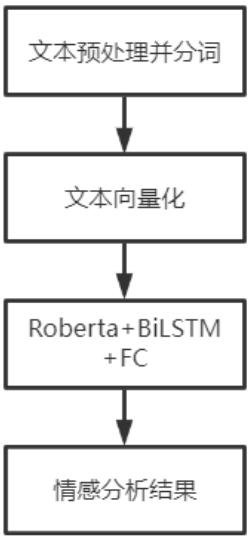

图 2-7 情感分析流程图

2.1.4 谣言检测

本系统使用循环神经网络等方式对微博文本进行深层特征挖掘，并完成二分类任务，即检测是否为谣言/虚假信息。

在谣言检测任务中，主要尝试了 LSTM 模型以及 BERT+BiLSTM 以及 Chinese-bert-wwm+BiLSTM 三种方法。

(1) 基于 LSTM 模型的方法

LSTM 模型[3]本质上是一种特定形式的循环神经网络(RNN)，该模型在 RNN 模型的基础上通过增加门限来解决 RNN 短期记忆的问题，使得循环神经网络能够真正有效地利用长距离的时序信息。LSTM 在 RNN 的基础结构上增加了输入门限、输出门限、遗忘门限 3 个逻辑控制单元，且各自连接到了一个乘法元件上，通过设定神经网络的记忆单元与其他部分连接的边缘处的权值控制信息流的输入、输出以及细胞单元的状态。其具体结构如图 2-8 所示。

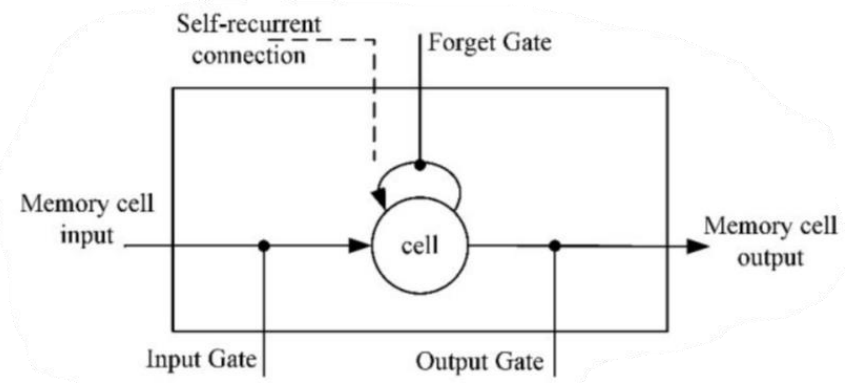

图 2-8 LSTM 模型结构图

(2) 基于 BiLSTM 模型的方法

使用当下主流的预训练模型，包括：BERT[4]、Chinese-bert-wwm[5]。BERT 等预训练模型分为两个阶段：

a. 预训练阶段：在海量语料库上以无监督学习的方学习语言特征。

b. 微调阶段：把 BERT 作为神经网络的特征提取器，在此基础上再添加一个网络层便可以完成对特定任务的微调。

将 BERT/Chinese-bert-wwm 做为嵌入层提取特征，然后传入 BiLSTM，最后使用全连接层输出分类。基本流程如图 2-9 所示。以谣言检测的二分类任务为例，在 pooler output 层的后面加一个全连接层，神经元个数为类别数 2，再经过 softmax 即可得到情感分类概率，或者将 Sequence output 视为字嵌入结合 LSTM、CNN 等模型得到分类输出。

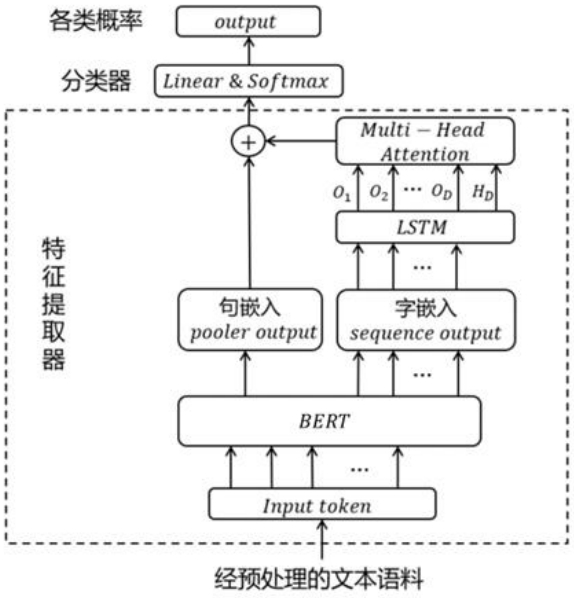

图 2-9 BERT+BiLSTM 模型架构图

2.2 核心功能实现

2.2.1 主题分析

(1) 基于 KMeans 文本聚类的方法

a. 预处理、分词模块：通过正则表达式去除一些无关的词，利用 jieba 库对清洗过的文本分词并去除停用词。

```
def clean(text):
    text = re.sub(r"(回复)?(//)?\s*@\S*?\s*(:| |$)", " ",
                  text)  # 去除正文中的@和回复/转发中的用户名
    text = re.sub(r"\[\S+\]", "", text)  # 去除表情符号
    # text = re.sub(r"#\S+#", "", text)  # 保留话题内容
    URL_REGEX = re.compile(
               r'(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-
z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^
\s()<>]+|(\([^\s()<>]+\)))*\)|[^\s`!()\[\]{};:\'".,<>?«»“”‘’]))',
        re.IGNORECASE)
    text = re.sub(URL_REGEX, "", text)  # 去除网址
    text = text.replace("转发微博", "")  # 去除无意义的词语
    text = text.replace("网页链接", "")
    text = text.replace("微博视频", "")
    text = re.sub(r"\s+", " ", text)  # 合并正文中过多的空格
    return text.strip()
```

图 2-10 预处理模块

```
comments = []
with open(args.data_path, 'r', encoding='UTF-8') as f:
    data = json.loads(f.read())
    lens = 100000
    index = list(range(len(data)))
    random.shuffle(index)
    for i in range(lens):
        if args.option == 0:
            text = jieba.cut(clean(data[index[i]]['text']))
        else:
            text = jieba.cut(clean(data[index[i]]['topics']))
        text_new = []
        for word in text:
            if word not in stopwords:
                text_new.append(word)
        comments.append(' '.join(text_new))
    del data, index
```

图 2-11 分词模块

b. 文本向量化：调用 sklearn 库中的方法将文本向量化，构造文本向量矩阵。

```
def TFIDF(corpus):
    vectorizer = CountVectorizer(max_df=0.7)
    word_vec = vectorizer.fit_transform(corpus)
    transformer = TfidfTransformer()
    tfidf = transformer.fit_transform(word_vec)
    tfidf_matrix = tfidf.toarray()
    return tfidf_matrix
```

图 2-12 文本向量化

c. KMeans 计算轮廓系数值：轮廓系数值可以用来衡量聚类的结果，轮廓系数值越高效果越好，用此方法决定聚类个数。

```
def eval_kMeans(X):
    scores = []
    values = np.arange(8, 25)
    # 迭代计算不同的轮廓系数值
    for num_clusters in values:
        km_cluster = KMeans(n_clusters=num_clusters, max_iter=100,
                            n_init=40, init='k-means++')
        km_cluster.fit(tfidf_matrix)
        score = metrics.silhouette_score(X, km_cluster.labels_,
                                         metric='euclidean',
                                         sample_size=len(X))
```

图 2-13 计算轮廓系数值

d. KMeans 聚类并制作每个类的词云。

```
def kMeans_cluster(tfidf_matrix, path, num_clusters):
    km_cluster = KMeans(n_clusters=num_clusters, max_iter=100,
                        n_init=40, init='k-means++')
    #返回各自文本的所被分配到的类索引
    labels = km_cluster.fit_predict(tfidf_matrix)
    text_classes = {0: '', 1: '', 2: '', 3: '', 4: '', 5: ''}
    for k in range(num_clusters):
        text_cls = []
        for index, res in enumerate(labels):
            if res == k:
                text_cls.append(''.join(comments[index]))
        text_join = ''.join(line for line in text_cls)
        text_classes[k] = text_join
        # 绘制词云图
        generate_wordclouds(text_join, path + str(k) + '.png')
    return labels
```

图 2-14 KMeans 聚类并绘制词云

(2) 基于 LDA 主题分析模型的方法

a. 文本预处理部分与 KMeans 实现相同，不再赘述。

b. 计算困惑度：利用困惑度决定 LDA 的主题个数，困惑度越低效果越好。

```
def run(corpus_1,id2word_1,num,texts):
    lda_model = LdaModel(corpus=corpus_1, id2word=id2word_1,
                         num_topics=num, passes=10, alpha=(50/num),
                         eta=0.01, random_state=42)
    perplex=lda_model.log_perplexity(corpus_1)     # 困惑度
    return lda_model,perplex
```

图 2-15 计算困惑度模块

c. LDA 及可视化实现模块。

```
data = [comment.split(' ') for comment in comments]
dictionary = corpora.Dictionary(data)
corpus = [dictionary.doc2bow(text) for text in data]
lda = models.LdaModel(corpus=corpus, id2word=dictionary,
                      num_topics=6)
d=pyLDAvis.gensim_models.prepare(lda, corpus, dictionary)
pyLDAvis.save_html(d, './ldavisual.html') # 可视化
```

图 2-16 LDA 及可视化模块

(3) 基于 AVITM 神经网络模型的方法

a. 文本预处理部分与 KMeans 实现相同，不再赘述。

b. 构造词表并将文本向量化表示：Vocab 类对输入的预料建立词表，提供将文本转换成向量的功能。

```
def count_corpus(tokens):
    if len(tokens) == 0 or isinstance(tokens[0], list):
        tokens = [token for line in tokens for token in line]
    return collections.Counter(tokens)
class Vocab:
    def __init__(self, tokens=None, min_freq=0):
        if tokens is None:    tokens = []
        counter = count_corpus(tokens)
        print(counter['新冠'])
        self._token_freqs = sorted(counter.items(),
                                   key=lambda x: x[1], reverse=True)
        self.token_to_idx = {'<unk>': 0}
        i = 1
        for token, freq in self._token_freqs:
            if freq < min_freq:  break
            self.token_to_idx[token] = i
            i = i + 1
```

图 2-17.1 构造词表

```
    def __len__(self):
        return len(self.token_to_idx)

    def __getitem__(self, tokens):
        if not isinstance(tokens, (list, tuple)):
            return self.token_to_idx.get(tokens, self.unk)
        return [self.token_to_idx.get(token, self.unk)
                for token in tokens]

    @property
    def unk(self):           return 0
    @property
    def token_freqs(self):   return self._token_freqs
```

图 2-17.2 构造词表(cont)

c. 训练模型：

```
def train():
    for epoch in range(args.num_epoch):
        all_indices =
            torch.randperm(tensor_tr.size(0)).split(args.batch_size)
        loss_epoch = 0.0
        model.train()                   # switch to training mode
        for batch_indices in all_indices:
            if not args.nogpu: batch_indices = batch_indices.cuda()
            input = Variable(tensor_tr[batch_indices])
            recon, loss = model(input, compute_loss=True)
            optimizer.zero_grad()       # clear previous gradients
            loss.backward()             # backprop
            optimizer.step()            # update parameters
            loss_epoch += loss.data    # add loss to loss_epoch
```

图 2-18 训练 NTM 模型

d. 计算相似度：计算文本词向量与主题概率分布矩阵的余弦值，取其中最大的作为该文本对应的主题。

```
def classify(pro,a):
    x = np.dot(pro,a) / (np.sqrt(np.sum(pro * pro,axis=1)) *
        np.sqrt(np.sum(a*a)) + 1e-9)
    topic = np.argmax(x)
    return topic
```

图 2-19 计算相似度

2.2.2 情感分析

a. 预处理并构建数据迭代器。与主题发现部分类似，同样进行数据清洗。将清洗后的文本分词向量化并进行截断。

b. 模型实现：主要使用 Roberta+Bi-LSTM+FC 的结构，用最后的隐状态变量作为输入在全连接层进行情感分类。

```
class Model(nn.Module):
    def __init__(self, config):
        super(Model, self).__init__()
        self.bert = BertModel.from_pretrained(config.bert_path)
        for param in self.bert.parameters():
            param.requires_grad = True
        self.lstm = nn.LSTM(config.hidden_size, config.rnn_hidden,
                            config.num_layers, bidirectional=True,
                            batch_first=True,dropout=config.dropout)
        self.dropout = nn.Dropout(config.dropout)
        self.fc_rnn = nn.Linear(config.rnn_hidden * 2,
                                config.num_classes)
def forward(self, x):
        context = x[0]  # 输入的句子
        mask = x[2]  # 对 padding 部分进行 mask，和句子一个 size
        encoder_out, text_cls = self.bert(context,
               attention_mask=mask, output_all_encoded_layers=False)
        out, _ = self.lstm(encoder_out)
        out = self.dropout(out)
        out = self.fc_rnn(out[:, -1, :])
        return out
```

图 2-20 情感分析模型

c. 训练模型。

```
for epoch in range(config.num_epochs):
    total_loss, total_len, total_r = 0
    for i, (trains, labels) in enumerate(train_iter):
        outputs = model(trains)
        model.zero_grad()
        loss = F.cross_entropy(outputs, labels)
        total_loss += loss
        total_r += (torch.argmax(outputs, dim=1) == labels).sum()
        total_len += outputs.shape[0]
        loss = loss / config.acc_grad
        loss.backward()
        if (i + 1) % config.acc_grad == 0:   optimizer.step()
        total_batch += 1
```

图 2-21.1 训练情感分析模型

```
    dev_f1,dev_loss,dev_precision = evaluate(config,model,dev_iter)
    dev_f1_score.append(dev_f1)
    time_dif = get_time_dif(start_time)
```

图 2-21.2 训练情感分析模型(cont)

d. 情感分析：加载训练好的模型后进行预测。

```
def final_predict(config,model,data_iter):
    model.load_state_dict(torch.load(config.save_path))
    model.eval()
    predict_all = np.array([], dtype=int)
    with torch.no_grad():
        for texts, labels in tqdm(data_iter):
            outputs = model(texts)
            pred = torch.max(outputs.data, 1)[1].cpu().numpy()
            predict_all = np.append(predict_all, pred)
    return predict_all
```

图 2-22 情感分析预测

2.2.3 谣言检测

(1) 基于 LSTM 模型的方法

a. 从 all_data_train.txt 中读取文本，将各标签下的文本分别转换为 list 存储，其中，content 为原始文本字符串；label 为标签，0 为谣言，1 为非谣言。

b. 语料预处理：对每一条微博文本 text，

   a) 去掉每个样本的标点符号；

   b) 用 pkuseg 分词，得到存放分词结果的 cut_list；

   c) 去掉 cut_list 中的停用词得到 cut_list_clean；

   d) 将预处理后的文本存入 text_pre.txt 文件。

c. 使用 word2vec 训练词向量，将训练好的词向量存放在 text.weibo.bigram 中。

```
sentences = word2vec.LineSentence("text_pre.txt")
model = word2vec.Word2Vec(sentences, vector_size=12,
                          window=25, min_count=2,
                          workers=5, sg=1, hs=1)
model.wv.save_word2vec_format('text.weibo.bigram', binary=False)
cn_model = KeyedVectors.load_word2vec_format('text.weibo.bigram',
                                             binary=False,
                                             unicode_errors="ignore")
```

图 2-23 词向量训练

d. 将索引的长度标准化为最大长度：

   a) 使用词向量进行索引化，

b) 获得最大的索引长度，

c) 进行 padding 和 truncating。

f. 准备 Embedding Matrix。

```
embedding_matrix = np.zeros((num_words, embedding_dim))
for i in range(num_words):
    embedding_matrix[i,:] = cn_model[cn_model.index_to_key[i]]
embedding_matrix = embedding_matrix.astype('float32')
```

图 2-24 计算 embedding_matrix

g. 训练模型：

a) 划分训练集和测试集，并搭建网络结构，如图 2-25：

- 用 keras 搭建 LSTM 和 GRU 模型，模型的第一层是 Embedding 层，只有当把 tokens 索引转换为词向量矩阵之后，才可以用神经网络对文本进行处理。 keras 提供了 Embedding 接口，避免了繁琐的稀疏矩阵操作；
- 在 Embedding 层输入的矩阵为：$(batchsize, maxtokens)$，输出矩阵为：$(batchsize, maxtokens, embeddingdim)$；
- 使用预训练的词向量，将 trainable 设为 False，即不可训练；

```
model = Sequential()
model.add(Embedding(num_words, embedding_dim,
                    weights = [embedding_matrix],
                    input_length = max_tokens,
                    trainable = False))
model.add(Bidirectional(LSTM(units=64, return_sequences=True)))
model.add(Bidirectional(LSTM(units=32, return_sequences=False)))
model.add(Dense(64, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))
optimizer = Adam(lr = 1e-3)
```

图 2-25 搭建 LSTM 网络结构

b) 配置模型的 earlystopping、lr_reduction 和 callbacks 等；

```
earlystopping = EarlyStopping(monitor='val_loss', patience=5,
                              verbose=1)
lr_reduction  =  ReduceLROnPlateau(monitor='val_loss',  factor=0.1,
                              min_lr=1e-8, patience=0, verbose=1)
callbacks = [earlystopping, lr_reduction]
model.compile(optimizer = optimizer, loss = 'binary_crossentropy',
              metrics = ['accuracy'])
```

图 2-26 模型参数配置

c) 训练模型并应用于测试集。

(2) 基于 BiLSTM 模型的方法

a. 从 prev_data.txt 读取文本并进行预处理，与 LSTM 实现类似，不再赘述。
b. 使用预训练好的 Bert//Chinese-bert-wwm 模型进行 tokenize。

```
tokenizer = BertTokenizer.from_pretrained(model_config.bert_path)
result_comments_id = tokenizer(result_comments, padding=True,
                               truncation=True, max_length=200,
                               return_tensors='pt')
```

图 2-27 使用 Bert 模型进行 tokenize

c. 定义并训练模型，其中，模型的定义如图 2-32 所示
   a) 划分训练集、验证集和测试集；

```
X_train,X_test, y_train, y_test = train_test_split( X, y,
                                  test_size=0.3, shuffle=True,
                                  stratify=y, random_state=0)
X_valid,X_test,y_valid,y_test = train_test_split(X_test, y_test,
                                  test_size=0.5, shuffle=True,
                                  stratify=y_test, random_state=0)
```

图 2-28 划分训练集、验证集和测试集

   b) 训练模型

```
net = bert_lstm(config.bert_path, config.hidden_dim,
                config.output_size, config.n_layers,
                config.bidirectional)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters(), lr=config.lr)
if(config.use_cuda):
    net.cuda()
net.train()
```

图 2-29 模型训练

   c) 定义评价参数宏平均 macro_f1 和微平均 micro_f1。

```
micro_f1 = f1_score(pred, true, average='micro')    # 计算 F1-score
macro_f1 = f1_score(pred, true, average='macro')
```

图 2-30 评价参数

d. 在测试集上进行测试。

```
net = bert_lstm(config.bert_path, config.hidden_dim,
                config.output_size, config.n_layers,
                config.bidirectional)
net.load_state_dict(torch.load(config.save_path))
net.cuda()
criterion = nn.CrossEntropyLoss()
h = net.init_hidden(config.batch_size)  # init hidden state
net.eval()
```

图 2-31 评价参数

```
class bert_lstm(nn.Module):
def __init__(self, bertpath, hidden_dim, output_size,
             n_layers,bidirectional=True, drop_prob=0.5):
        super(bert_lstm, self).__init__()
        self.output_size = output_size
        self.n_layers = n_layers
        self.hidden_dim = hidden_dim
        self.bidirectional = bidirectional
        # Bert ----  重点，bert 模型需要嵌入到自定义模型里面
        self.bert=BertModel.from_pretrained(bertpath)
        for param in self.bert.parameters():
            param.requires_grad = True
        # LSTM layers
        self.lstm = nn.LSTM(768, hidden_dim, n_layers,
                            batch_first=True, bidirectional=bidirectional)
       # dropout layer
        self.dropout = nn.Dropout(drop_prob)
       # linear and sigmoid layers
        if bidirectional:
            self.fc = nn.Linear(hidden_dim*2, output_size)
        else:
            self.fc = nn.Linear(hidden_dim, output_size)
        self.sig = nn.Sigmoid()

    def forward(self, x, hidden):
        batch_size = x.size(0)
        x=self.bert(x)[0]     # bert 字向量
        lstm_out, (hidden_last,cn_last) = self.lstm(x, hidden)
        if self.bidirectional:
            hidden_last_L=hidden_last[-2]
            hidden_last_R=hidden_last[-1]
            hidden_last_out=torch.cat([hidden_last_L,hidden_last_R],
                                      dim=-1)
        else:
            hidden_last_out=hidden_last[-1]   #[32, 384]
        # dropout and fully-connected layer
        out = self.dropout(hidden_last_out)
        out = self.fc(out)
        out = self.sig(out)
        return out
```

图 2-32 模型训练

3　软件系统测试与分析

3.1　核心功能测试与分析

3.1.1 主题分析

从与公共卫生事件相关的微博博文中随机抽取六万条进行主题发现。

(1) 基于 KMeans 文本聚类的方法

如图 3-1 所示，使用轮廓系数值可以决定聚类的个数为 6。

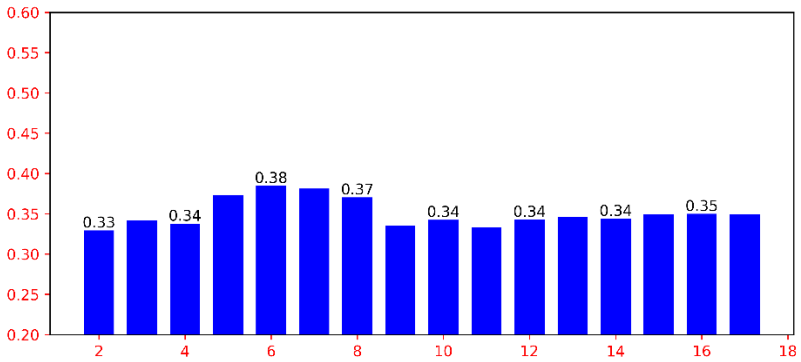

图 3-1 KMeans 轮廓系数

将聚类后的文本用词云进行展示，部分结果图 3-2 所示，由结果可以看出，各类别之间的区分并不明显，聚类的效果较为一般。

图 3-2 KMeans 词云效果展示

(2) 基于 LDA 主题分析模型的方法

使用 LDA 的部分结果图 3-3 所示，由结果可以看出各个主题下关键词区分度不高，模型效果一般。

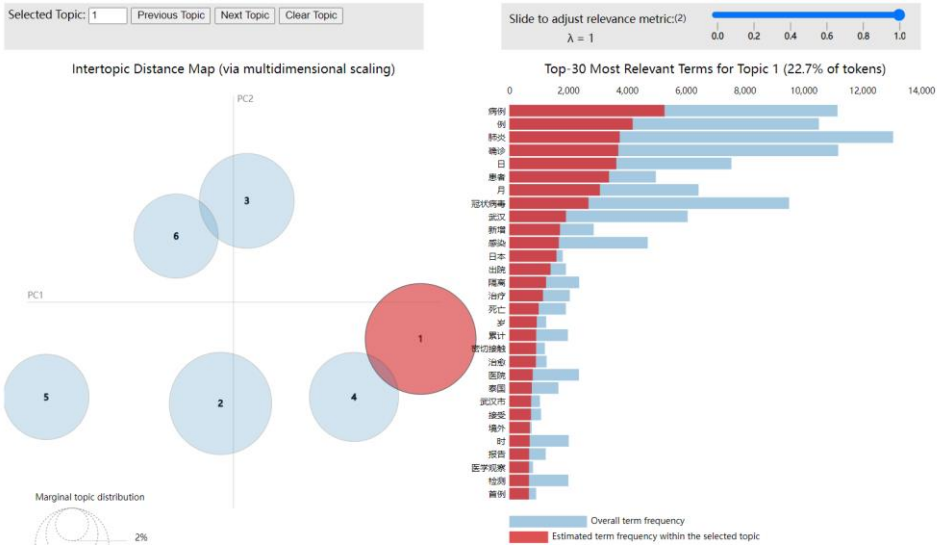

图 3-3 LDA 结果

(3) 基于 AVITM 神经网络模型的方法

如图 3-4 所示，训练模型 75 个 epoch 后，loss 值趋于稳定。模型训练后一共发现了 9 个主题，如图 3-5 所示。其中，topic2 和 topic4 下的关键词较为接，所以可以进行过滤，最终得到 8 个主题。

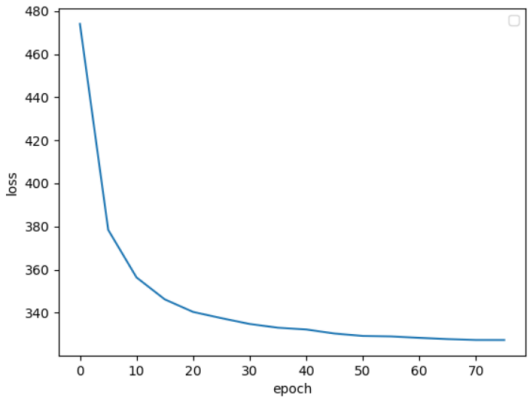

图 3-4 训练损失值

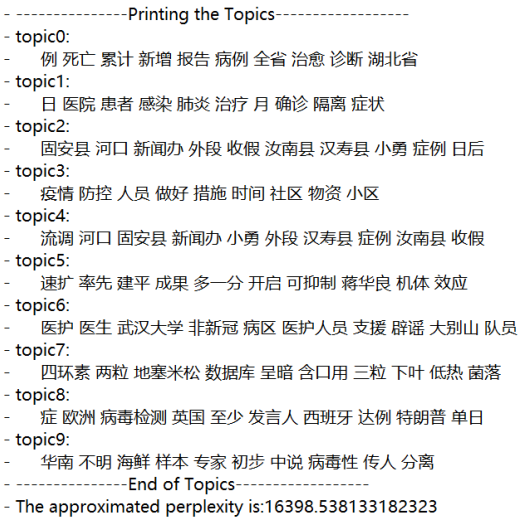

图 3-5 输出关键词

上述三种模型效果均表现一般，原因可能是由于所采集的数据集是仅局限于新冠肺炎话题，故每个主题下的关键短语趋于相同，且数据预处理部分仅简单进行了基于匹配的清洗。

3.1.2 情感分析

情感分析部分采用 SMP2020 微博情绪分类技术评测（SMP2020-EWECT）卫生事件相关的数据集进行 6 分类，共训练 5 个 epoch，训练好的模型在测试集上的准确率为 72.8%。使用训练后的模型在主题发现的结果上进行预测，部分结果如图 3-6 所示，模型效果良好。

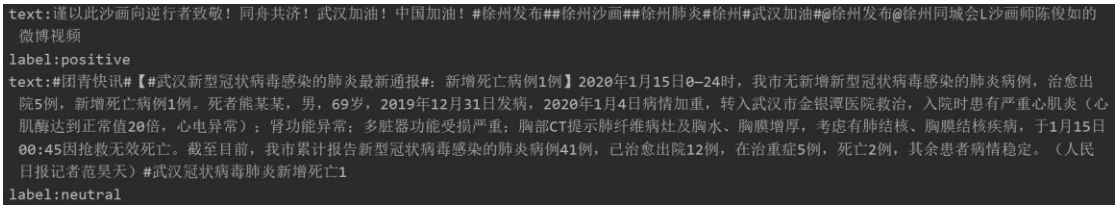

图 3-6 情感分析结果

3.1.3 谣言检测

谣言检测部分采用清华大学开源的疫情谣言数据集 CSDC-Rumor 进行训练和测试，在测试集上的准确率如图 3-7 所示：

Model	LSTM	**Model**	bert+BiLSTM	Chinese-BERT-wwm + BiLSTM
Precision	97.05%	**Macro_F1**	0.919	0.890
		Micro_F1	0.918	0.891

图 3-7 谣言检测结果

3.2 前端系统展示

图 3-8.1 前端(目录)

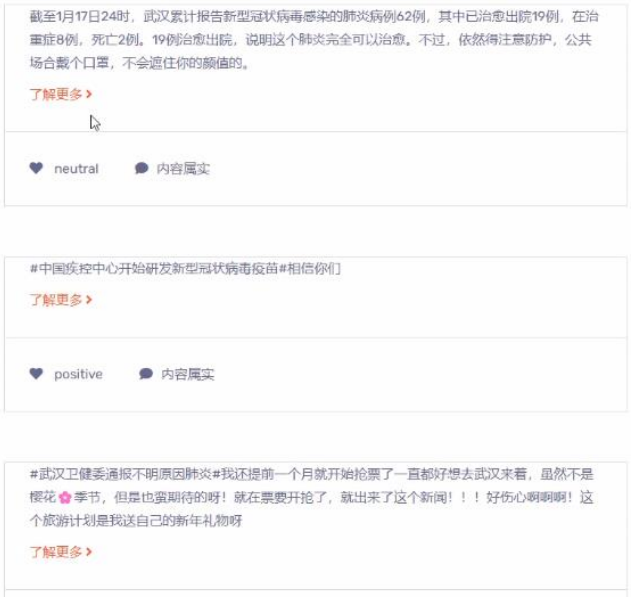

图 3-8.2 前端(部分词条)

参考文献

[1] Xanthopoulos P, Pardalos P M, Trafalis T B. Linear discriminant analysis[M]//Robust data mining. Springer, New York, NY, 2013: 27-33.

[2] Srivastava A, Sutton C. Autoencoding variational inference for topic models[J]. arXiv preprint arXiv:1703.01488, 2017.

[3] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[4] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[5] Cui Y, Che W, Liu T, et al. Pre-training with whole word masking for chinese bert[J]. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2021, 29: 3504-3514..