NLP大作业2：多粒度情感分析 技术报告

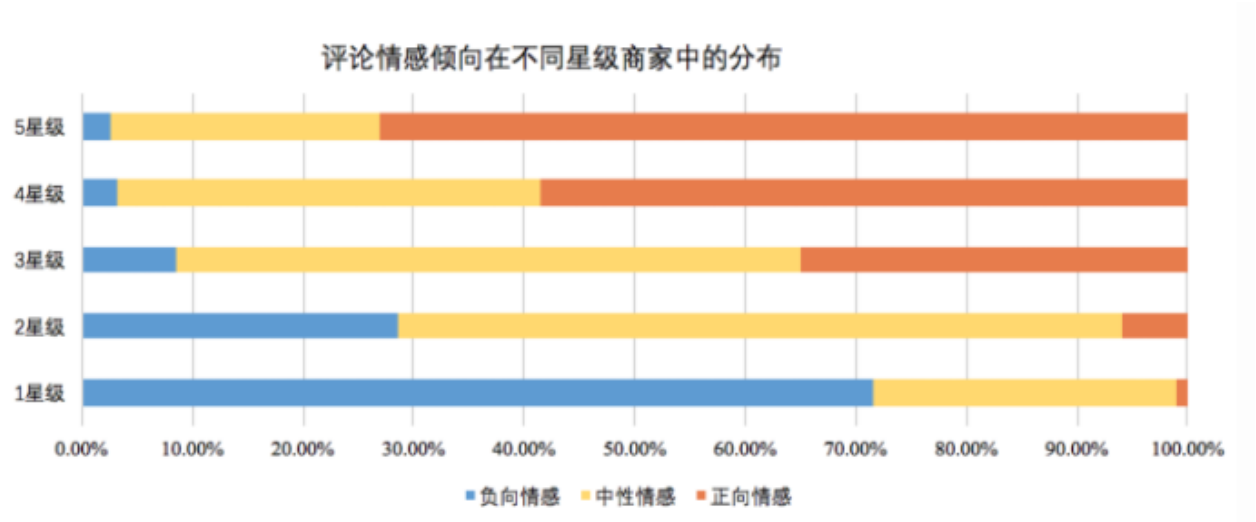
张驰 1120191600 18810575675 1061823234@qq.com
魏慧聪 1120191866 15733906832 870249395@qq.com
徐幸波 1120191232 19967471965 1065658706@qq.com

技术报告目录：

1 问题背景

移动互联网的发展和普及已经达到空前规模，我们在使用美团APP点外卖和寻找美食时，经常会查看用户对商家的点评。在美团APP的点评中，用户的情感倾向在不同星级的商家中分布如图1所示，商家整体星级打分反映出用户对商家的整体情感。然而，用户的高星点评中仍然会出现一些负面评价，用户的低星点评中仍然会出现正向的评价内容，如下图所示。

经过美团APP的统计信息，我们可以看出评论情感倾向在不同星级的商家中的分布：



所以我们希望根据输入的评论内容，可以得到用户的对商家的相关评价。

2 核心思想和算法描述

评论细粒度情感是属性级的情感分析，与篇章级情感分析只需给出整体情感倾向而无法获取对某件事物不同属性的具体评价态度不同，属性级分析主要分成了两个子任务，包括属性识别和属性的情感识别。

本程序的属性主要来源于已经定义好的属性类型，而非直接从文本中提取其中含有的属性，即要分析某一段文本的情感属性需要先匹配出文本包含的预定义属性，再根据每个属性的特征分为积极、中性和负面三类情感。

2.1 属性分类

参考一些外卖平台给出的评论选项和SemEval数据集，初步确定属性分为位置、服务、价格、环境、菜品、其他六大类，具体细分为20类

位置	服务	价格	环境	菜品	其他
交通是否便利	排队等候时间	价格水平	装修情况	分量	本次消费感受
距离商圈远近	服务人员态度	折扣力度	噪杂情况	口感	再次消费意愿
是否容易寻找	是否容易停车	性价比	卫生情况	外观	
	点菜/上菜速度		就餐空间	推荐程度	

为了在算法中区分情感程度，每一个具体属性都有四种标签——{正向，中立，负向，未有提及}，而在最终输出阶段，只会显示提及到的属性，未提及的属性不会显示出来

情感倾向量化	1	0	-1	-2
情感含义	正面情感	中性情感	负面情感	情感倾向未提及

2.2 算法结构

算法共包含5层结构：预处理、Bert模型处理、对抗训练、任务相关层

预处理包括文本的分词以及特征的提取。分词使用Keras框架的分词器Tokenizer处理，Tokenizer是一个将文本转换为序列的类。属性词提取常用的方法有传统的机器学习方法和基于深度学习的方法，我们对属性值划分了两个层次，因此采用基于深度学习的方法，利用CNN提取更高层次的特征。借助embedding模型嵌入，用一个低维度的向量表示属性和观点，距离相近的向量具有相近的情感倾向。

在数据经过预处理后，将会对每组数据转换成Bert模型所需要的数据结构，建立BERT细粒度标签情感分类多任务模型 。Bert模型功能强大，处于共享层也即语义编码层（Encoder），使用双向训练来嵌入单词，目标是学习在不同属性的上下文的深度语义特征，对属性的拓展有良好的兼容性，有利于上下文的理解与比较，而不局限于属性前或者属性后的文本。

$$H = Encoder(R)$$

其中，H向量为评论R经过语义编码层（Encoder）编码后得到的向量

对抗训练需要输入训练数据的Keras模型，之所以要引入对抗模型是想要用正则化的手段提高模型的泛化能力，提高模型的表现。原则上对embedding层的输出也要进行同样的操作，但是拆解，重构模型过于繁琐，由于Embedding的模型最后输出由Embedding参数矩阵决定，因此可以直接对Embedding的参数矩阵进行修改，虽然不同样本的相同属性共用了相同的更改规则，但是仍然有一定的正则化的作用，大大减小了计算量，达到了效率和时间较为平衡的效果。

任务相关层有注意力层和分类层，对句子中不同重要程度的词赋予不同的权重，让模型聚焦到整条评论中指定属性最相关的句子和片段减少其他无关信息的干扰。

$$M_i^a = tanh(W_i^a * H)$$
$$\alpha_i = softmax(\omega_i^T * M_i^a)$$
$$r_i = tanh(W_i^p * H * \alpha_i^T)$$

α_i是注意力权重向量，对不同的词赋予不同程度的权重，r_i是评论对第i个属性的加权表征

最后模型经过Softmax分类器，计算每个属性的情感分析损失

$$y_i = softmax(W_i^q * r_i + b_i^q)$$
$$L_{ACSA} = \frac{1}{N}\sum_{i=1}^{N}\sum_{C} y_i * logy_i$$

3 系统主要模块流程

3.1 系统配置环境

使用python3.6、安装bert4keras、安装Keras==2.1.0、安装h5py==2.10.0、安装sklearn、安装numpy、安装pandas

3.2 训练模块流程

3.2.1 相关配置的路径

```
config_path = 'RoBERTa-tiny3L312-clue/bert_config.json'
checkpoint_path = 'RoBERTa-tiny3L312-clue/bert_model.ckpt'
dict_path = 'RoBERTa-tiny3L312-clue/vocab.txt'

train_data = pd.read_csv('data/sentiment_analysis_trainingset.csv').iloc[:, 1:].dropna()
valid_data = pd.read_csv('data/sentiment_analysis_validationset.csv').iloc[:,
1:].dropna()

train_data.head()
train_data.iloc[:, 1:] = train_data.iloc[:, 1:].apply(lambda x: x + 2)
valid_data.iloc[:, 1:] = valid_data.iloc[:, 1:].apply(lambda x: x + 2)

tags = train_data.columns[1:]
num_classes_topic = len(tags)
```

3.2.2 建立BERT细粒度标签情感分类多任务模型

加载bert的预训练模型，进行多任务学习。

```
# 加载预训练模型
bert = build_transformer_model(
      config_path=config_path,
      checkpoint_path=checkpoint_path,
      model='bert',
      return_keras_model=False,
)

pooler = Lambda(lambda x: x[:, 0], name='CLS-token')(bert.model.output)

# 多任务学习
mutil_layers = []
for i in range(num_classes_topic):
      dropout = Dropout(dropout_rate)(pooler)
      preds = Dense(num_classes_sentiment, activation='softmax',
kernel_initializer=bert.initializer, name='preds_{}'.format(i))(dropout)
      mutil_layers.append(preds)

model = keras.models.Model(bert.model.input, mutil_layers)
```

3.2.3 设置模型中的相关参数

```
set_gelu('tanh')   # 切换gelu版本

maxlen = 256
batch_size = 128
epochs = 25
dropout_rate = 0.1
num_classes_sentiment = 4
```

3.2.4 获取数据转换格式：get_data(data)函数

get_data(data)函数，主要用于，将csv文件中输入的编码，转换成BERT模型中需要的输入模式，以便于对模型进行训练和分析。

```
def get_data(data):
      token_ids = []
      segment_ids = []
      label = []
      # 循环每个句子
      for text in tqdm(data['content'].astype(str)):
            # 分词并把token变成编号
            token_id, segment_id = tokenizer.encode(text, maxlen=maxlen)
            token_ids.append(token_id)
            segment_ids.append(segment_id)
      token_ids = sequence_padding(token_ids)
      segment_ids = sequence_padding(segment_ids)

      # 获取20个维度的标签
      for columns in tags:
            label.append(np.array(data[columns]).astype('uint8'))
      label = np.array(label)
      return [token_ids, segment_ids], label

train_input, train_label = get_data(train_data)
valid_input, valid_label = get_data(valid_data)
```

该函数的目的是从数据中获取数据的输入和数据的标签,循环其中的每个句子，通过分词把token变成编号，加入到token_ids,segment_ids,label的列表中。之后获取20个维度的标签，放入列表中，返回输入的列表以及相应的标签。

数据集中的标签为：

> id,content,location_traffic_convenience,location_distance_from_business_district,location_easy_to_find,service_wait_time,service_waiters_attitude,service_parking_convenience,service_serving_speed,price_level,price_cost_effective,price_discount,environment_decoration,environment_noise,environment_space,environment_cleaness,dish_portion,dish_taste,dish_look,dish_recommendation,others_overall_experience,others_willing_to_consume_again

3.2.5 配置Keras中的优化率、损失函数，设定学习率衰减

```
loss_dict = {}
loss_weights_dict = {}
for i in range(num_classes_topic):
      loss_dict['preds_{}'.format(i)] = 'sparse_categorical_crossentropy'
      loss_weights_dict['preds_{}'.format(i)] = 1.

# 分段线性学习率的优化器。
# loss_weights表示每个任务的权重

# 设置分段线性学习率
AdamLR = extend_with_piecewise_linear_lr(Adam, name='AdamLR')

# 指定所采用的优化器，采用的损失函数以及其权重，以及评价函数用于评估当前训练模型的性能，配置相应的参数
model.compile(      loss=loss_dict,
                    loss_weights=loss_weights_dict,
                    optimizer=AdamLR(learning_rate=1e-4, lr_schedule={
                            int((len(train_input[0]) // batch_size * epochs) * 0.2): 1,
                            int((len(train_input[0]) // batch_size * epochs) * 0.3):
0.1
                    }),
                     metrics=['accuracy'],
                   )
```

optimizer：配置采用loss，用于对分段线性学习进行优化

loss：损失函数。

metrics：评价函数用于评估当前训练模型的性能。当模型编译后，评价函数作为 metrics 的参数来输入。

3.2.6 为模型添加对抗训练模型，函数为：adversarial_training()

该函数的参数包含，model、embedding_name、epsilon，该函数用于为model指定的模型添加对抗训练，Embedding_name为model中隐含层的名字，在模型编译后使用。

首先我们需要先了解一下对抗样本，所谓的对抗样本，是指对于人类来说“看起来”几乎一样、但对于模型来说预测结果却完全不一样的样本。“对抗攻击”，其实就是想办法造出更多的对抗样本，而“对抗防御”，就是想办法让模型能正确识别更多的对抗样本。所谓对抗训练，则是属于对抗防御的一种，它构造了一些对抗样本加入到原数据集中，希望增强模型对对抗样本的鲁棒性。

NLP的整体的对抗模型引入的思路分析如下：（引用自：https://kexue.fm/archives/7234）

> 对于NLP任务来说，原则上也要对Embedding层的输出进行同样的操作，Embedding层的输出shape为(b,n,d)，所以也要在Embedding层的输出加上一个shape为(b,n,d)的`Variable`，然后进行上述步骤。但这样一来，我们需要拆解、重构模型，对使用者不够友好。

> 不过，我们可以退而求其次。Embedding层的输出是直接取自于Embedding参数矩阵的，因此我们可以直接对Embedding参数矩阵进行扰动。这样得到的对抗样本的多样性会少一些（因为不同样本的同一个token共用了相同的扰动），但仍然能起到正则化的作用，而且这样实现起来容易得多。

添加对抗模型的整体思路如下：

1. 如果模型中还没有训练函数，就手动添加训练函数，这里要注意备份旧的训练函数。
2. 查找隐含层，之后找到隐含层求出隐含层的梯度，转为dense tensor，将其封装为函数
3. 在进行对抗训练中，隐含层的输出是直接取自隐含层参数的矩阵，所以我们在进行对抗训练时，可以直接对隐含层的参数矩阵进行扰动，先计算扰动，然后在模型中注入扰动，之后使用梯度下降的方法计算，删除扰动。
4. 最后要覆盖原来的训练函数

基于上述思路函数的代码如下所示：

```
def adversarial_training(model, embedding_name, epsilon=1):
    """给模型添加对抗训练
    其中model是需要添加对抗训练的keras模型，embedding_name
    则是model里边Embedding层的名字。要在模型compile之后使用。
    """
    if model.train_function is None:  # 如果还没有训练函数
        model._make_train_function()  # 手动make
    old_train_function = model.train_function  # 备份旧的训练函数

    # 查找Embedding层
    for output in model.outputs:
        embedding_layer = search_layer(output, embedding_name)
        if embedding_layer is not None:
            break
    if embedding_layer is None:
        raise Exception('Embedding layer not found')

    # 求Embedding梯度
    embeddings = embedding_layer.embeddings  # Embedding矩阵
    gradients = K.gradients(model.total_loss, [embeddings])  # Embedding梯度
    gradients = K.zeros_like(embeddings) + gradients[0]  # 转为dense tensor

    # 封装为函数
    inputs = (
        model._feed_inputs + model._feed_targets + model._feed_sample_weights
    )  # 所有输入层
    embedding_gradients = K.function(
        inputs=inputs,
        outputs=[gradients],
        name='embedding_gradients',
    )  # 封装为函数

    def train_function(inputs):  # 重新定义训练函数
        grads = embedding_gradients(inputs)[0]  # Embedding梯度
```

```
            delta = epsilon * grads / (np.sqrt((grads**2).sum()) + 1e-8)    # 计算扰动
            K.set_value(embeddings, K.eval(embeddings) + delta)    # 注入扰动
            outputs = old_train_function(inputs)    # 梯度下降
            K.set_value(embeddings, K.eval(embeddings) - delta)    # 删除扰动
            return outputs

      model.train_function = train_function    # 覆盖原训练函数
```

之后启用对抗训练：adversarial_training(model, 'Embedding-Token', 0.5)

3.2.7 模型评估以及保存模型

对训练出的模型进行评估，保存训练过程中f1值最好的模型。

函数evaluate(x_trues,y_trues)用于对模型进行评估，计算每轮训练的模型的f1值。

类Evaluator(keras.callbacks.Callback)用于保存模型，对每轮的模型，判断其f1值，寻找f1值最优的模型，进行保存。

```
def evaluate(x_trues, y_trues):
      golds_topic, preds_topic = [], []
      golds_sentiment, preds_sentiment = [], []

      y_preds = np.argmax(model.predict(x_trues, batch_size = 128, verbose=1), axis=-1)
      for y_pred, y_true in zip(y_preds.T, y_trues.T):
            golds_topic.append([1 if y > 0 else 0 for y in y_true])
            preds_topic.append([1 if y > 0 else 0 for y in y_pred])
            golds_sentiment.extend(y_true)
            preds_sentiment.extend(y_pred)
      print(classification_report(y_true=golds_topic, y_pred=preds_topic,
target_names=tags, digits=4))
      print(classification_report(y_true=golds_sentiment, y_pred=preds_sentiment,
target_names=['未提及', '负', '中', '正'], digits=4))
      f1_score_topic = f1_score(y_true=golds_topic, y_pred=preds_topic, average='micro')
      f1_score_sentiment = f1_score(y_true=golds_sentiment, y_pred=preds_sentiment,
average='micro')
      f1_score_average = float(format((f1_score_topic + f1_score_sentiment) / 2, '.4f'))
      print("f1_score_average", f1_score_average, type(f1_score_average))
      return f1_score_average

class Evaluator(keras.callbacks.Callback):
      """评估与保存
     """
      def __init__(self):
            self.best_val_f1 = 0.

      def on_epoch_end(self, epoch, logs=None):
            val_f1 = evaluate(valid_input, valid_label)
            if val_f1 > self.best_val_f1:
```

```
                self.best_val_f1 = val_f1
                model.save_weights('model/best_model_tag_sentiment.weights')
                model_json = model.to_json()
                with open('model/best_model_tag_sentiment.json', 'w') as json_file:
                        json_file.write(model_json)
        print(
                u'val_f1: %.5f, best_val_f1: %.5f\n' %
                (val_f1, self.best_val_f1)
        )
```

3.2.8 训练模型

使用model.fit()方法用于执行训练过程，其中的参数为：训练集的输入特征、训练集的标签，每一个batch的大小，迭代次数，以及回调函数（在训练过程中被调用之后进行改变（用于对抗训练中））

```
model.fit(
        train_input, [train_label[i] for i in range(num_classes_topic)],
        batch_size=batch_size,
        epochs=epochs,
        callbacks=[evaluator]
)
```

3.3 配置网页端模块：

3.3.1 配置相关参数及设置路径

在app.py中，同样需要设置相关的model路径，同时要给出标签的列表，根据模型的返回值取出列表中的相关文字，输出至网页端中。

```
##设置路径
dict_path = 'RoBERTa-tiny3L312-clue/vocab.txt'
model_json_path = 'model/best_model_tag_sentiment.json'
model_weights_path = 'model/best_model_tag_sentiment.weights'

target_names_cn = ["交通是否便利", "距离商圈远近", "是否容易寻找", "排队等候时间", "服务人员态度", "是否容易停车", "点菜/上菜速度", "价格水平", "性价比",
                   "折扣力度", "装修情况", "嘈杂情况", "就餐空间", "卫生情况", "分量", "口感", "外观", "推荐程度", "本次消费感受", "再次消费的意愿"]

sentiment_names_cn = ["情感倾向未提及", "负面情感", "中性情感", "正面情感"]

model = model_from_json(open(model_json_path).read())
model.load_weights(model_weights_path)

# 建立分词器
tokenizer = Tokenizer(dict_path, do_lower_case=True)
```

其中，get_data函数与训练模块中类似，不再赘述。

3.3.2 FLASK框架的配置与搭建：

第一部分，初始化：所有的Flask都必须创建程序实例，web服务器使用wsgi协议，把客户端所有的请求都转发给这个程序实例。程序实例是Flask的对象，一般情况下用如下方法实例化。Flask类只有一个必须指定的参数，即程序主模块或者包的名字，name是系统变量，该变量指的是本py文件的文件名。

```
from flask import Flask, render_template, request
app = Flask(__name__)
```

第二部分，路由和视图函数：
客户端发送url给web服务器，web服务器将url转发给flask程序实例，程序实例
需要知道对于每一个url请求启动那一部分代码，所以保存了一个url和python函数的映射关系。处理url和函数之间关系的程序，称为路由。在flask中，定义路由最简便的方式，是使用程序实例的app.route装饰器，把装饰的函数注册为路由。

其中txt为html中的提交表单，neg，neu，pos为返回的负面评价、中性评价与积极评价标签。

```
@app.route('/', methods=['GET', 'POST'])
def hello_world():
      if request.method == 'GET':
            return render_template('index.html')
      else:
            # 1.接请求数据,存放到变量text中
            text = request.form.get('txt')
            txt = request.form.get('txt')
            # 2.predict
            predict_input, predict_texts = get_data_test([text])
            predict_preds = np.argmax(model.predict(predict_input, verbose=1), axis=-1)
            neg = []
            neu = []
            pos = []
            for pred in predict_preds.T:
                  for i, j in enumerate(pred):
                        if j == 1:
                              neg.append(target_names_cn[i])
                        elif j == 2:
                              neu.append(target_names_cn[i])
                        elif j == 3:
                              pos.append(target_names_cn[i])
            neg_result = "|".join(neg)
            neu_result = "|".join(neu)
            pos_result = "|".join(pos)
            print(neg_result)
            print(neu_result)
            print(pos_result)
            return render_template('index.html', txt=txt, neg=neg_result,
neu=neu_result, pos=pos_result)
```

第三部分，直接调用，调用app.run()函数。

如上述代码所示，app是flask的实例，功能就是接受来自web服务器的请求，浏览器将请求给web服务器，web服务器将请求给app,app收到请求，通过路由找到对应的视图函数，然后将请求处理，得到一个响应response。然后app将响应返回给web服务器，web服务器返回给浏览器，浏览器展示给用户观看，流程完毕。

3.4 HTML网页展示界面

由于展示界面与整体的情感分析关系不大，就不再展示代码，整体界面如下图所示：

4 程序使用说明

按照要求配置好环境，下载好相应的库文件后，由于数据集量较大，所以将数据集放在百度网盘中，提取链接如下所示：

> 链接：https://pan.baidu.com/s/1ZcJgwjAUwjLJVPRWNAQOng
> 提取码：otoc

下载数据集并解压在项目文件夹中，目录结构如下图所示：

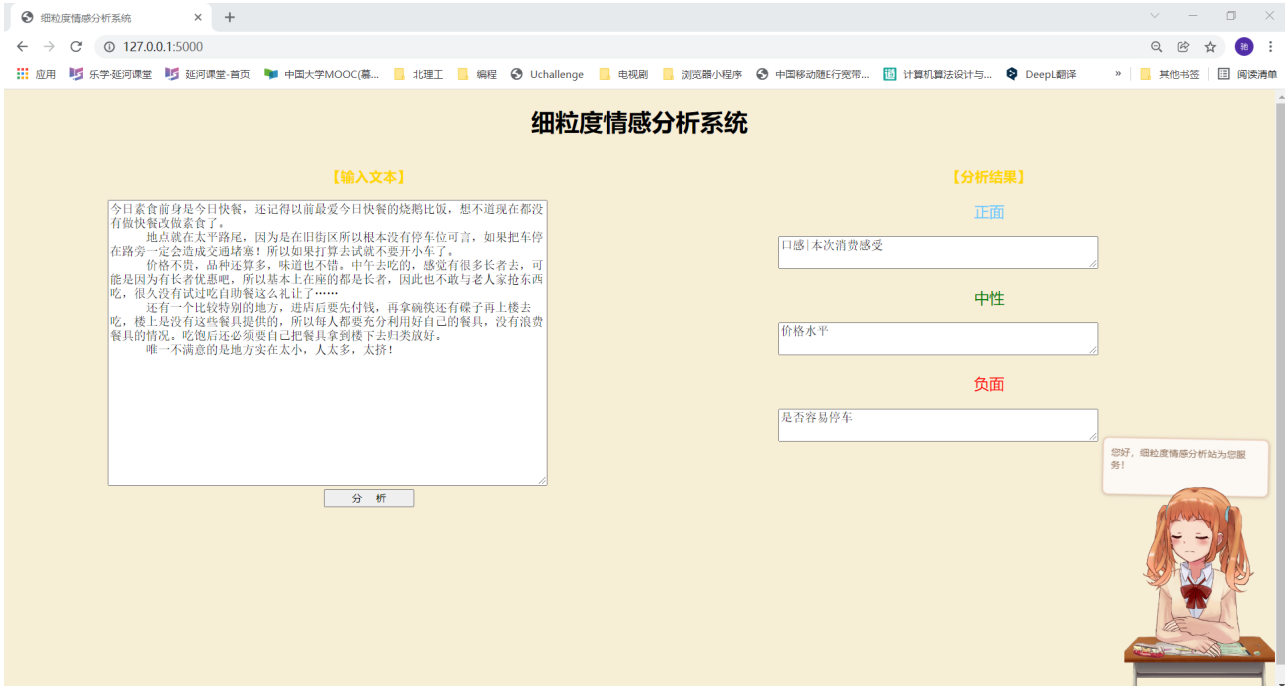
如果不查看数据集，那么可以直接运行app.py，运行后输出如下所示：

将网站：http://127.0.0.1:5000 输入到浏览器中，即可在网页端上使用。

5 实验结果及分析

5.1 HTML展示界面的结果

如果出现字符重叠的现象，请适当调整浏览器界面的大小，缩放浏览器即可。



5.2 模型的评估值（P值、R值、F1值）

评测指标包含："交通是否便利"，"距离商圈远近"，"是否容易寻找"，"排队等候时间"，"服务人员态度"，"是否容易停车"，"点菜/上菜速度"，"价格水平"，"性价比"，"折扣力度"，"装修情况"，"嘈杂情况"，"就餐空间"，"卫生情况"，"分量"，"口感"，"外观"，"推荐程度"，"本次消费感受"，"再次消费的意愿"

模型对于各个评价指标的P值、R值、F1值，如下所示

	precision	recall	f1-score	support
location_traffic_convenience	0.8996	0.8566	0.8776	3243
location_distance_from_business_district	0.7600	0.6584	0.7055	2968
location_easy_to_find	0.8860	0.8031	0.8425	3484
service_wait_time	0.7720	0.6222	0.6891	1763
service_waiters_attitude	0.9194	0.8173	0.8654	9008
service_parking_convenience	0.9359	0.7956	0.8601	954
service_serving_speed	0.8498	0.6870	0.7598	2323
price_level	0.8768	0.7610	0.8148	7503
price_cost_effective	0.8007	0.5882	0.6782	3572
price_discount	0.8751	0.8203	0.8468	5738
environment_decoration	0.9011	0.8551	0.8775	7199
environment_noise	0.8018	0.7254	0.7617	4479
environment_space	0.8194	0.7767	0.7975	5509
environment_cleaness	0.8564	0.7572	0.8038	5404
dish_portion	0.8454	0.6852	0.7570	6888
dish_taste	0.9657	0.9923	0.9788	14244
dish_look	0.7556	0.3885	0.5132	4242
dish_recommendation	0.8137	0.5002	0.6195	2917
others_overall_experience	0.9810	1.0000	0.9904	14715
others_willing_to_consume_again	0.7764	0.4950	0.6046	5646
micro avg	0.8889	0.7881	0.8355	111799
macro avg	0.8546	0.7293	0.7822	111799
weighted avg	0.8804	0.7881	0.8270	111799
samples avg	0.8942	0.8053	0.8316	111799

各个评测指标的评测值对应为：″情感倾向未提及″，″负面情感″，″中性情感″，″正面情感″，其P值、R值、F1值，如下所示

	precision	recall	f1-score	support
未提及	0.8821	0.9415	0.9108	188201
负	0.6819	0.5648	0.6178	12789
中	0.6380	0.4715	0.5422	25929
正	0.7846	0.7448	0.7642	73081
accuracy			0.8369	300000
macro avg	0.7466	0.6806	0.7088	300000
weighted avg	0.8287	0.8369	0.8307	300000

6 团队基本信息及分工情况

本团队共三人，分工情况如下所示：

姓名	学号	联系方式	邮箱	完成的工作
张驰	1120191600	18810575675	1061823234@qq.com	数据集的获取 Bert模型的构建与训练 flask框架的搭建 文档编写工作
魏慧聪	1120191866	15733906832	870249395@qq.com	数据集的处理 对抗训练模块的构建 模型效果的评估 文档编写工作
徐幸波	1120191232	19967471965	1065658706@qq.com	数据集的处理 客户网页端的构建 Bert模型的优化 代码汇总工作