



## 模 式 识 别 与 机 器 学 习

# 基于机器学习和深度学习的中文多模态情感分析

**Chinese multimodal sentiment analysis based on machine learning and deep learning**

姓 名： 马永辉，苏升榕，黃艺豪，朱壮

学 号： 34520212201617,34520212201650,

34520212201530,34520212201813

专 业： 自动化

年 级： 2021 级

校内指导教师： 关锦婷老师

二〇二三年十二月二十四日

## 摘要

本文探讨了多模态数据下情绪识别的问题，首先利用一些工具一起提取我们想要的特征，接着再进行机器学习（决策树、随机森林、提升树、Adaboost、bagging 和 stacking），深度学习（bert, bert+textcnn, cam++, timnet, yolov8）对文本，音频以及视频进行识别得出各自单峰模态的结果，再利用多数表决法对总体进行表决，得出最终结果，最后运用了MLMF 和 MTFN 进行多模态特征融合，再进行多任务分类得出结果。比较得出虽然深度学习在单峰模态的分类结果比较优秀，但是对于总体情绪而言，还是多模态学习性能更好。

**关键词：**机器学习；多模态特征；深度学习；多模态学习

## Abstract

This paper discusses the problem of emotion recognition under multi-modal features. First, some tools are used to remove the desired features. Then, machine learning (decision trees, random forests, lifting trees, Adaboost, bagging, and stacking), deep learning (bert, bert+textcnn, cam++, timnet,yolov8, etc.) are implemented. timnet, yolov8) identified text, audio and video to get their own single-modal results, and then voted on the whole by majority voting method to get the final results. Finally, MLMF and MTFN were used for multi-modal feature fusion, and then multi-task classification was carried out to get the results. The comparison shows that although the classification results of deep learning in unimodal are better, the performance of multi-modal learning is better for the overall emotion.

**Key Words:** Machine learning; Multimodal characteristics; Deep learning; Multimodal learning

# 目 录

<b>第一章 绪论 .....</b>	<b>1</b>
1.1 组内分工 .....	1
1.2 目的 .....	1
<b>第二章 数据描述及采集方法 .....</b>	<b>3</b>
2.1 数据采集方法 .....	4
2.1.1 文本 .....	4
2.1.2 音频 .....	5
2.1.3 视频 .....	6
2.2 数据描述 .....	6
2.2.1 数据降维可视化 .....	6
2.2.2 数据聚类可视化 .....	8
<b>第三章 机器学习算法 .....</b>	<b>10</b>
3.1 决策树 .....	10
3.2 Adaboost .....	12
3.3 梯度提升树 .....	14
3.4 Bagging .....	15
3.5 随机森林 .....	16
3.6 Stacking .....	16
<b>第四章 深度学习算法 .....</b>	<b>18</b>
4.1 文本 .....	18
4.1.1 BERT .....	18
4.1.2 BERT+TEXTCNN .....	23
4.2 音频 .....	24
4.2.1 CAM++ .....	24
4.2.2 TIM-NET .....	26
4.3 视频 .....	28
4.3.1 YOLOv8 目标检测 .....	28
4.3.2 YOLOv8 目标分类 .....	30

<b>第五章 多模态学习算法 .....</b>	<b>32</b>
5.1 MTFN .....	32
5.2 MLMF .....	35
<b>第六章 训练过程及结果 .....</b>	<b>38</b>
6.1 机器学习 .....	38
6.1.1 文本 .....	38
6.1.2 音频 .....	39
6.1.3 视频 .....	40
6.1.4 总体 .....	40
6.2 深度学习 .....	41
6.2.1 文本 .....	41
6.2.2 音频 .....	41
6.2.3 视频 .....	42
6.2.4 总体 .....	42
6.3 多模态学习 .....	43
6.3.1 MLMF .....	43
6.3.2 MTFN .....	43
6.4 结果 .....	45
<b>第七章 总结 .....</b>	<b>46</b>
7.1 苏升榕 .....	46
7.2 黄艺豪 .....	46
7.3 朱壮 .....	47
7.4 马永辉 .....	47
<b>参考文献 .....</b>	<b>50</b>
<b>附录 .....</b>	<b>51</b>
A.1 MMSA .....	51
A.2 MMSA FFT .....	52
A.3 YOLOv8 .....	53
A.4 bert .....	54
A.5 投票集成 .....	55
A.6 我的项目 .....	58

致谢 .....	60
----------	----

## Contents

<b>Chapter 1 Introduction .....</b>	<b>1</b>
1.1 Intra-group division of labor .....	1
1.2 Purpose .....	1
<b>Chapter 2 Data description and acquisition method .....</b>	<b>3</b>
2.1 Data acquisition method .....	4
2.2 Data description .....	6
<b>Chapter 3 Machine learning algorithm .....</b>	<b>10</b>
3.1 Decision tree .....	10
3.2 Adaboost .....	12
3.3 Gradient lifting tree .....	14
3.4 Bagging .....	15
3.5 Random forest .....	16
3.6 Stacking .....	16
<b>Chapter 4 Deep learning algorithm .....</b>	<b>18</b>
4.1 Text .....	18
4.2 Audio .....	24
4.3 Video .....	28
<b>Chapter 5 Multimodal learning algorithm .....</b>	<b>32</b>
5.1 MTFN .....	32
5.2 MLMF .....	35
<b>Chapter 6 Training process and results .....</b>	<b>38</b>
6.1 Machine learning .....	38
6.2 Deep learning .....	41
6.3 Multimodal learning .....	43
6.4 Results .....	45
<b>Chapter 7 Conclusion .....</b>	<b>46</b>
7.1 sushengrong .....	46
7.2 huangyihao .....	46

7.3 zhuzhuang .....	47
7.4 mayonghui .....	47
References .....	50
Appendix .....	51
A.1 MMSA .....	51
A.2 MMSA FET .....	52
A.3 YOLOv8 .....	53
A.4 bert .....	54
A.5 Voting integration .....	55
A.6 My project .....	58
Acknowledgements .....	60

# 第一章 绪论

## 1.1 组内分工

表 1-1: 小组分工

姓名	学号	职务	分工
马永辉	34520212201617	组长	负责数据采集, 深度学习和多模态学习代码, 汇总并撰写论文
苏升榕	34520212201650	组员	负责机器学习代码, 总结参考论文, 撰写论文
黄艺豪	34520212201530	组员	负责数据描述代码, 总结参考论文, 撰写论文
朱壮	34520212201813	组员	负责机器学习代码, 总结参考论文, 撰写论文

## 1.2 目的

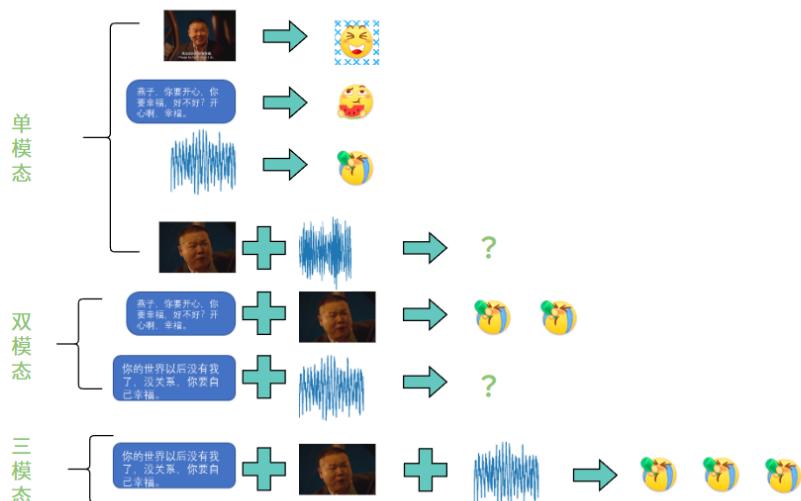


图 1-1: 单模态, 双模态与三模态

随着社交媒体在互联网上的日益活跃，人们大部分工作和生活都离不开社交软件。每个人都可以在社交平台上发表自己对当前热点话题的看法或讨论。因此，多模态情感分析 (Multimodal Sentiment Analysis, MSA) 是一项重要的研究任务，旨在探索人们在社交媒体中所表露的情感，其中包括听觉和视觉等非语言信息和文本信息。多模态数据在其特点上具有相似性和独立性。一方面，视频片段中的多模态信息倾向于表达出采访者相同情绪的概率分布。另一方面，来自不

同模态的信息是独立的，在高维线性空间中服从不同的概率分布。利用多模态数据的这两大特性可以消除语义上的歧义，为多模态情感分许提供额外的线索。然而，多模态情感分析目前面临着两大挑战：在多媒体视频中，音频和视觉等非语言信息很难为该任务提供有用的反馈，而文本信息往往对最终结果具有决定性作用。因此，如何有效利用非言语特征，为文本模态提供额外的提示成为任务的关键；单模态特征的提取和跨模态的异构性为多模态特征融合带来了较大的困难。

## 第二章 数据描述及采集方法

CH-SIMS 是一个中文多模态情感分析数据集。在一段视频中，说话人的脸和声音必须同时出现在画面中，并保持一段特定的时间。为了获得尽可能贴近生活的视频片段，数据集从电影、电视剧中收集目标片段。在获得原始视频后，使用视频编辑工具裁剪目标片段，保证数据足够精确。此外，在数据收集和裁剪的过程中，使用了以下限制：

- 1) 视频中只包含普通话，并谨慎地选择有口音的材料。
- 2) 每个视频片段的长度不低于一秒，不超过十秒。
- 3) 对于每个视频片段，除了说话人的脸，没有其他人的脸出现。

本文使用的数据集是中国多模态情感分析数据集含有 2281 个视频数据，收集自不同的电影、电视连续剧和综艺节目。这些视频片段带有自发表情、各种头部姿势、遮挡物和光照。对于每一个视频样本，其长度不超过 10 秒，不短于 1 秒。每个视频片段中除了说话者的面孔，不会出现其他人的面孔。将其中的 1824 个样本作为训练集，另外的 457 个样本作为测试集。训练集中的各类数量占比如下图：

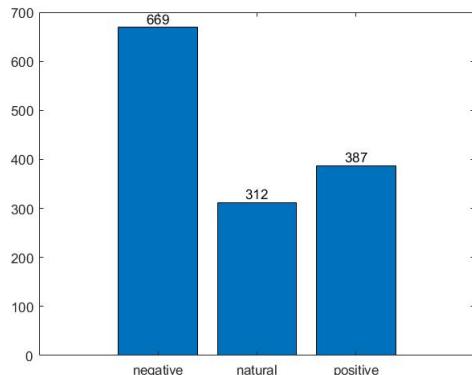


图 2-1: 训练集中的各类数量占比

表 2-1: 数据集描述

Item	#
视频总数	60
片段总数	2281
男性	1500
女性	781
不同发言人总数	474
片段平均长度 (秒)	3.67
每个片段的平均字数	15

```
{
  "train": {
    "raw_text": [],           # 原始文本
    "audio": [],             # 音频特征
    "vision": [],            # 视频特征
    "id": [],                # [视频ID_片段ID, ..., ...]
    "text": [],               # BERT特征
    "text_bert": [],          # BERT中的单词ID
    "audio_lengths": [],      # 音频特征长度 (在每个样本上)
    "vision_lengths": [],     # 与音频长度相同
    "annotations": [],        # 注释
    "classification_labels": [], # 负向(0), 中立(1), 正向(2)。在v_2.0中已被弃用
    "regression_labels": []   # 负向(<0), 中立(0), 正向(>0)
  },
  "valid": {***},           # 与 "train" 相同
  "test": {***},            # 与 "train" 相同
}
```

## 2.1 数据采集方法

### 2.1.1 文本

所有视频都有手动转录，包括中文和英文版本。我们只用中文翻译。我们添加两个唯一的标记来指示每个转录本的开始和结束。然后，使用预训练的中文 BERT 基词嵌入来从转录本中获得词向量 (Devlin 等人, 2018 年)。值得注意的是，由于 BERT 的特点，我们没有使用分词工具。最后，每个单词被表示为 768 维的单词向量。

## 2.1.2 音频

我们使用 LibROSA (McFee 等人, 2015) 语音工具包, 其具有默认参数以在 22050Hz 下提取声学特征。共提取了 33 维帧级声学特征, 包括 1 维对数基频 (log F0)、20 维梅尔倒谱系数 (MFCC) 和 12 维恒 Q 谱 (CQT)。这些特征与情绪和语音语调有关 (Li 等人, 2018 年)。

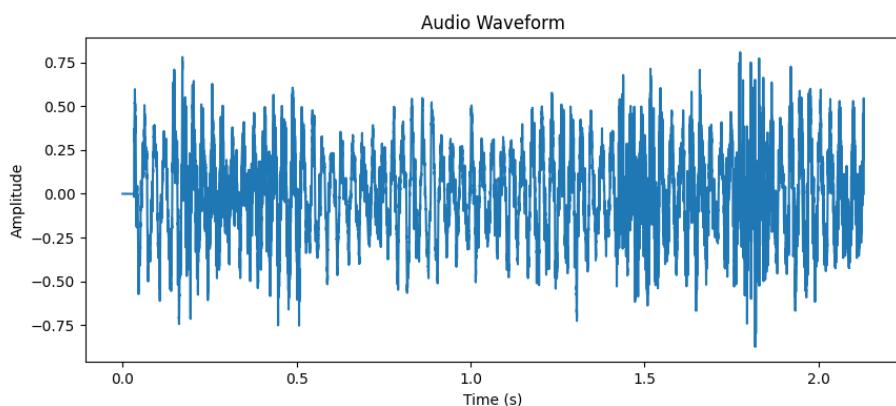


图 2-2: 音调与时间的关系

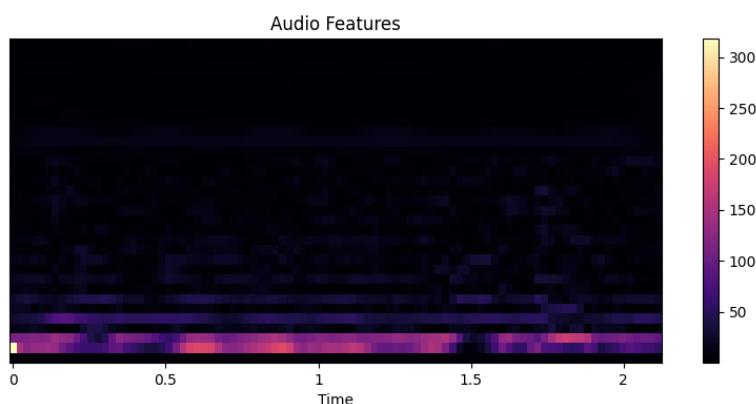


图 2-3: 音频特征

**对数基频 (log F0):** 基频是音频信号中最低的频率, 通常对应于音乐中的音调。对数基频是将基频取对数后得到的值, 这样可以使得频率的感知更接近人耳的感知。

**梅尔倒谱系数 (MFCC):** MFCC 是一种在语音识别中常用的特征。它首先将信号的频谱转换为梅尔频率, 然后取对数, 最后进行离散余弦变换。梅尔频率是一种对频率的非线性变换, 使得频率的感知更接近人耳的感知。

**恒 Q 谱 (CQT):** CQT 是一种频谱分析方法, 它的特点是在低频时有较高的频率分辨率, 而在高频时有较高的时间分辨率。这也更接近人耳的感知。

### 2.1.3 视频

以 30Hz 从视频片段中提取帧。我们使用 MTCNN 人脸检测算法 (Zhang et al., 2016a) 以提取对齐的面部。然后, 根据 Zadeh 等人 (2018 b), 我们使用 MultiComp OpenFace2.0 工具包 (Baltrušaitis 等人, 2018) 来提取 68 个面部标志、17 个面部动作单元、头部姿势、头部取向和眼睛注视的集合。最后, 共提取了 709 维帧级视觉特征。

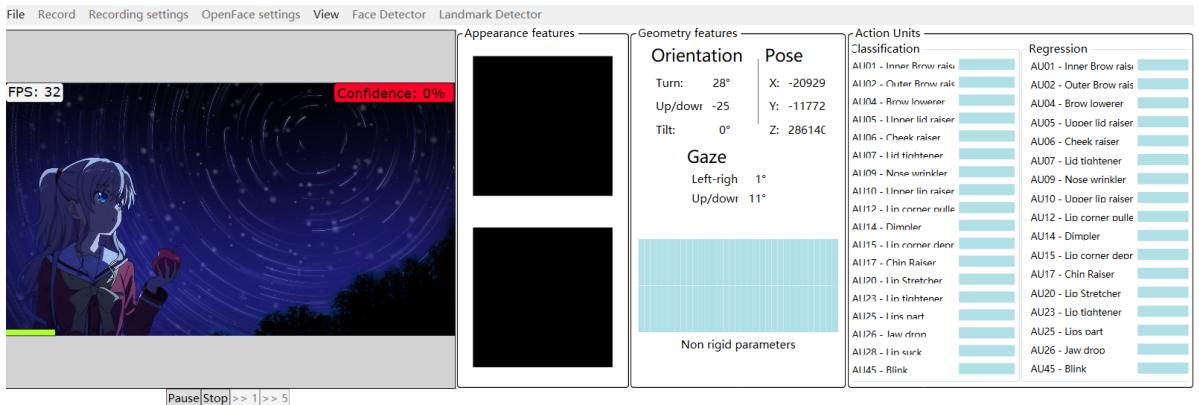


图 2-4: OpenFace2.0

## 2.2 数据描述

### 2.2.1 数据降维可视化

数据降维是通过保留数据的主要信息和结构, 将高维数据转为低维表示的过程。它旨在消除冗余和噪声, 提炼出数据的最重要的方面, 从而简化数据分析任务和可视化。数据可视化是通过图形、图表和交互式界面等方式, 将抽象的数据转化为可视化形式, 以便更直观地理解和分析数据。它能够帮助我们发现数据之间的关系、趋势和异常等。本文采用的数据降维方法为 t-SNE, 其基本思想是在高维空间构建一个概率分布拟合高维样本点间的相对位置, 在低维空间, 也构建一个概率分布拟合高维样本点之间的位置关系, 通过学习, 调整低维数据点, 令两个分布接近。用条件概率表示高维样本点间的位置关系:

$$p_{j|i} = \frac{e^{-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}}} \quad (2-1)$$

当点  $i$  与点  $j$  的距离越近,  $p_{j|i}$  就越大;

用条件概率表示低维样本点间的位置关系:

$$q_{j|i} = \frac{e^{-\|y_i - y_j\|^2}}{\sum_k \neq i e^{-\|y_i - y_k\|^2}}. \quad (2-2)$$

为了让两个分布接近，采用 KL 散度对其进行衡量：

$$C = \sum_i \text{KL}(P_i || Q_i) = \sum \sum p_{j|i} \log \frac{P_{j|i}}{Q_{j|i}} \quad (2-3)$$

通过对  $y$  的调整使  $C$  最小化，实现降维。

途中红色的点为 negative 类，绿色的点为 natural 类，蓝色的点为 positive 类，本文三类数据降维后的数据分布入下图所示：

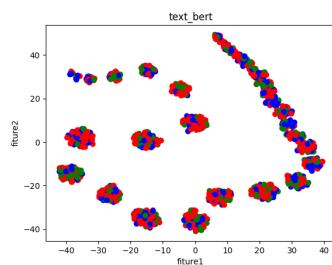


图 2-5: 文本 t-sne

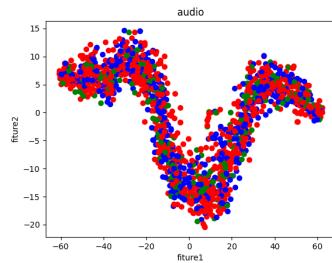


图 2-6: 音频 t-sne

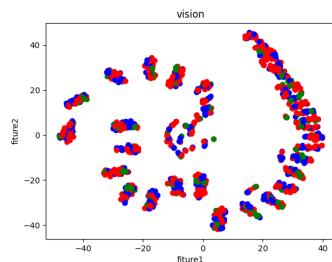


图 2-7: 视频 t-sne

## 2.2.2 数据聚类可视化

K-means 是一种迭代的聚类算法，用于将数据点分为 K 个组或类别。以下是 K-means 算法的基本步骤：

1. 随机选择 K 个数据点作为初始的聚类中心（也称为质心）。
2. 对于数据集中的每个点，计算其到每个质心的距离，并将其分配给最近的质心所在的类别。
3. 对于每个类别，计算所有分配给该类别的点的均值，并将该均值设置为新的质心。

重复步骤 2 和 3，直到质心不再显著变化，或者达到预设的最大迭代次数。

K-means 算法的目标是最小化每个点到其所在类别质心的距离之和，这也被称为群内平方和（Within-Cluster Sum of Squares，WCSS）。

kmeans 聚类如下图所示：

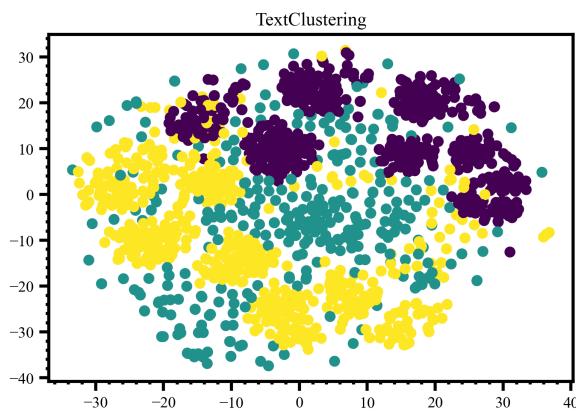


图 2-8: 文本 kmeans 聚类

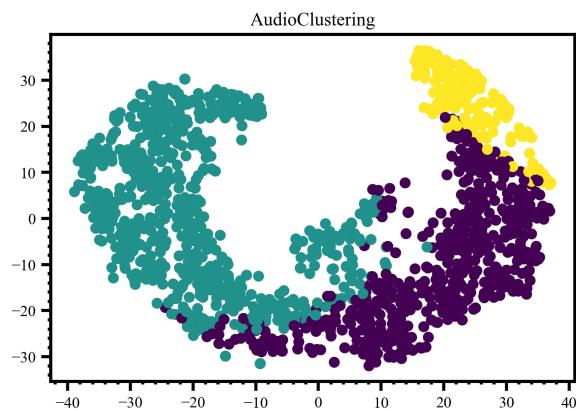


图 2-9: 音频 kmeans 聚类

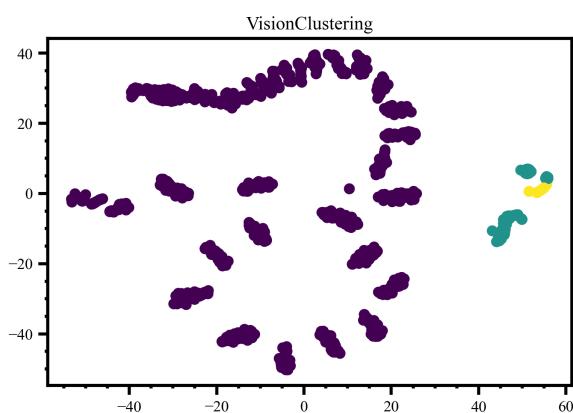


图 2-10: 视频 kmeans 聚类

## 第三章 机器学习算法

本次实验使用了 6 种不同的机器学习方法，包括决策树、随机森林、提升树、Adaboost、bagging 和 stacking 算法，以此来对课上所将方法进行实践和加深其认识。

本次主要对文本、音频和视频运用这六种机器学习方法进行分类，对采集和处理过的数据进行拉长处理后将特征矩阵变为二维矩阵后再输入模型进行分类。以下是对各个方法原理的简要描述：

### 3.1 决策树

决策树（Decision Tree），它是一种以树形数据结构来展示决策规则和分类结果的模型，作为一种归纳学习算法，其重点是将看似无序、杂乱的已知数据，通过某种技术手段将它们转化成可以预测未知数据的树状模型，每一条从根结点（对最终分类结果贡献最大的属性）到叶子结点（最终分类结果）的路径都代表一条决策的规则。

一般，一棵决策树包含一个根节点，若干个内部结点和若干个叶结点。

叶结点对应于决策结果，其他每个结点对应于一个属性测试。每个结点包含的样本集合根据属性测试的结果划分到子结点中，根结点包含样本全集，从根结点到每个叶结点的路径对应了一个判定的测试序列。决策树学习的目的是产生一棵泛化能力强，即处理未见示例强的决策树。

使用决策树进行决策的过程就是从根节点开始，测试待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为决策结果。

#### 1. 步骤

至于如何生成决策树，具体步骤如下图：

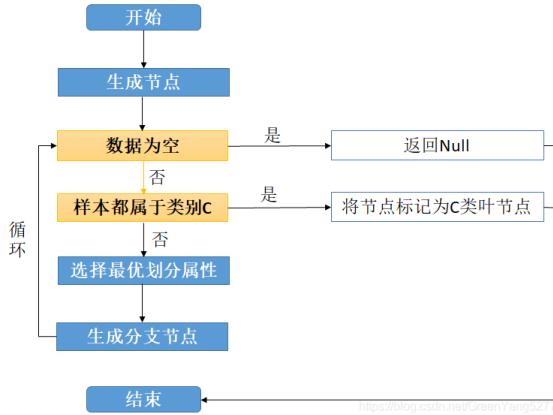


图 3-1: 生成决策树具体步骤

上图就是在生成决策树的过程中经历的步骤。详细步骤如下：

- (1). 首先从开始位置，将所有数据划分到一个节点，即根节点。
- (2). 然后经历橙色的两个步骤，橙色的表示判断条件：  
若数据为空集，跳出循环。如果该节点是根节点，返回 `null`；如果该节点是中间节点，将该节点标记为训练数据中类别最多的类  
若样本都属于同一类，跳出循环，节点标记为该类别；
- (3). 如果经过橙色标记的判断条件都没有跳出循环，则考虑对该节点进行划分。既然是算法，则不能随意的进行划分，要讲究效率和精度，选择当前条件下的最优属性划分（什么是最优属性，这是决策树的重点，后面会详细解释）
- (4). 经历上步骤划分后，生成新的节点，然后循环判断条件，不断生成新的分支节点，直到所有节点都跳出循环。
- (5). 结束。这样便会生成一棵决策树。

## 2. 划分选择

了解了其步骤后，便明白了其关键便是如何寻找最优属性，其选择方法一般有三种。

### (1). 信息增益

通过方程

$$\text{Gain}(D, a) = \text{Ent}(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} \text{Ent}(D^v) \quad (3-1)$$

来计算每个属性的信息增益，最优属性即为信息增益最大的属性。

$$\text{Ent}(D) = - \sum_{k=1}^{|Y|} p_k \log_2 p_k \quad (3-2)$$

， $\text{Ent}(D)$  即为样本  $D$  的信息熵，由该公式计算可得。

### (2). 信息增益率

信息增益准则对可取值数目较多的属性有所偏好，增益率定义如下：

$$\text{Gain\_ratio}(D, a) = \frac{\text{Gain}(D, a)}{\text{IV}(a)} \quad (3-3)$$

其中

$$\text{IV}(a) = - \sum_{v=1}^V \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|} \quad (3-4)$$

称为属性 a 的固有值。该划分算法先从候选划分属性中找出信息增益高于平均水平的属性，再从中选择增益率最高的属性。

### (3). 基尼指数

数据 D 的纯度可用基尼值来度量，即用如下公式来计算：

$$\text{Gini}(D) = \sum_{k=1}^{|\gamma|} \sum_{k' \neq k} p_k p_{k'} = 1 - \sum_{k=1}^{|\gamma|} p_k^2 \quad (3-5)$$

$\text{Gini}(D)$  反映了从数据集 D 中随机抽取两个样本，其类别标记不一致的概率， $\text{Gini}(D)$  越小，表示数据集 D 的纯度越高。而属性 a 的基尼指数则是根据这个公式推广而得，具体形式如下：

$$\text{Gini\_index}(D, a) = \sum_{v=1}^V \frac{|D^v|}{|D|} \text{Gini}(D^v) \quad (3-6)$$

在候选属性集合 A 中，选择那个使得划分后基尼指数最小的属性作为最优划分属性。

### 3. 剪枝

剪枝顾名思义就是给决策树“去掉”一些判断分支，同时在剩下的树结构下仍然能得到不错的结果。之所以进行剪枝，是为了防止或减少“过拟合现象”的发生，是决策树具有更好的泛化能力。

剪枝主要分为两种方法，其一是预剪枝，即在决策树构造时就进行剪枝。在决策树构造过程中，对节点进行评估，如果对其划分并不能再验证集中提高准确性，那么该节点就不要继续往下划分。这时就会把当前节点作为叶节点。其二是后剪枝，即在生成决策树之后再剪枝。通常会从决策树的叶节点开始，逐层向上对每个节点进行评估。如果剪掉该节点，带来的验证集中准确性差别不大或有明显提升，则可以对它进行剪枝，用叶子节点来代替该节点。

## 3.2 Adaboost

AdaBoost 是 Adaptive Boosting（自适应增强）的缩写，它的自适应在于：被前一个基本分类器误分类的样本的权值会增大，而正确分类的样本的权值会减

小，并再次用来训练下一个基本分类器。同时，在每一轮迭代中，加入一个新的弱分类器，直到达到某个预定的足够小的错误率或预先指定的最大迭代次数再确定最后的强分类器。

### 1. 算法步骤

首先，是初始化训练数据的权值分布  $D_1$ 。假设有  $N$  个训练样本数据，则每一个训练样本最开始时，都会被赋予相同的权值： $w_1 = 1/N$ 。训练弱分类器  $C_i$ 。具体训练过程：如果某个训练样本点，被弱分类器  $C_i$  准确地分类，那么再构造下一个训练集中，它对应的权值要减小；相反，如果某个训练样本点被错误分类，那么它的权值就应该增大。权值的更新过的样本被用于训练下一个弱分类器，整个过程如此迭代下去。

最后，将各个训练得到的弱分类器组合成一个强分类器。各个弱分类器的训练过程结束后，加大分类误差率小的弱分类器的权重，使其在最终的分类函数中起着较大的决定作用，而降低分类误差率大的弱分类器的权重，使其在最终的分类函数中起着较小的决定作用。换而言之，误差率低的弱分类器在最终分类器中占的权重较大，否则较小。

### 2. 算法过程

(1). 首先，初始化训练集的权值分布。每个训练样本最开始都被赋予相同的权值：

$$w_i = \frac{1}{N} \quad (3-7)$$

这样样本集的权值初始分布为

$$D_1(i) = (w_1, w_2, \dots, w_N) = \left( \frac{1}{N}, \frac{1}{N}, \dots, \frac{1}{N} \right) \quad (3-8)$$

(2). 进行迭代  $t = 1, 2, \dots, T$

(a). 选取一个当前误差率最低的分类器  $h$  作为第  $t$  个基分类器  $H_t$ ，并计算弱分类器  $h_t$  在训练集上的分类误差率：

$$e_t = \sum_{i=1}^m w_{t,i} I(h_t(x_i) \neq f(x_i)) \quad (3-9)$$

(b). 计算该分类器在最终分类器中所占的权重：

$$\partial_t = \frac{1}{2} \ln \frac{1 - e_t}{e_t} \quad (3-10)$$

(c). 更新样本的权重分布：

$$D_{t+1} = \frac{D_t \exp(-\partial_t f(x) h_t(x))}{Z_t} \quad (3-11)$$

其中:

$$Z_t = \sum_{i=1}^m w_{t,i} \exp(-\partial_t f(x_i) h_t(x_i)) \quad (3-12)$$

(3). 最后按照弱分类器权重  $\partial_t$  组成各个弱分类器:

$$f(x) = \sum_{i=1}^T \partial_i H_i(x) \quad (3-13)$$

通过符号函数 sign 最终得到一个强分类器:

$$H_{final} = \text{sign}(f(x)) = \text{sign}\left(\sum_{i=1}^T \partial_i H_i(x)\right) \quad (3-14)$$

### 3.3 梯度提升树

提升方法实际采用加法模型（即基函数的线性组合）与前向分步算法。以决策树为基函数的提升方法称为提升树（boosting tree）。对分类问题决策树是二叉分类树，对回归问题决策树是二叉回归树。提升树模型可以表示为决策树的加法模型:

$$f_M(x) = \sum_{m=1}^M T(x; \theta_m) \quad (3-15)$$

其中， $T(x; \theta_m)$  表示决策树， $\theta_m$  为决策树参数， $M$  为树的个数。而梯度提升树的具体步骤如下：

1. 初始化  $f_0(x) = 0$ ，并选取损失函数 ( $L(y, f(x))$ );
2. 对于  $m = 0, 1, \dots, M$ 
  - (1). 计算负梯度:

$$-g_m(x_i) = -\frac{\partial (L(y, f(x_i)))}{\partial f(x_i)} \Big|_{f(x)=f_{m-1}(x)} \quad (3-16)$$

- (2). 以负梯度  $-g_m(x_i)$  为预测值，训练一个回归树  $T(x; \theta_m)$ ;
- (3). 更新  $f_m(x) = f_{m-1}(x) + \rho T(x; \theta_m)$ ;
3. 经过  $M$  次迭代后取得的模型即为

$$f_M(x) = \sum_{m=1}^M \rho T(x; \theta_m) \quad (3-17)$$

这里的  $\rho$  为学习率，可用来防止过拟合。

此次实验用梯度提升树来实现多分类任务，在这种情况下输出模型经过 softmax 函数转化为每个类别的置信概率，从而实现分类目标。

### 3.4 Bagging

Bagging [Breiman, 1996a] 是并行式集成学习方法最著名的代表。从名字即可看出，它直接基于自助采样法 (bootstrap sampling)。给定包含  $m$  个样本的数据集，我们先随机取出一个样本放入采样集中，再把该样本放回初始数据集，使得下次采样时该样本仍有可能被选中，这样，经过  $m$  次随机采样操作，我们得到含  $m$  个样本的采样集，初始训练集中有的样本在采样集里多次出现，有的则从未出现，初始训练集中约有 63.2% 的样本出现在采样集中。

照这样，我们可采样出  $T$  个含  $m$  个训练样本的采样集，然后基于每个采样集训练出一个基学习器，再将这些基学习器进行结合。这就是 Bagging 的基本流程。在对预测输出进行结合时，Bagging 通常对分类任务使用简单投票法，对回归任务使用简单平均法。若分类预测时出现两个类收到同样票数的情形，则最简单的做法是随机选择一个，也可进一步考察学习器投票的置信度来确定最终胜者。其步骤如下：

1. 对于给定的训练样本  $S$ ，每轮从训练样本  $S$  中采用有放回抽样 (Bootstraping) 的方式抽取  $M$  个训练样本，共进行  $n$  轮，得到了  $n$  个样本集合，需要注意的是这里的  $n$  个训练集之间是相互独立的。
2. 在获取了样本集合之后，每次使用一个样本集合得到一个预测模型，对于  $n$  个样本集合来说，我们总共可以得到  $n$  个预测模型。
3. 如果我们需要解决的是分类问题，那么我们可以对前面得到的  $n$  个模型采用投票的方式得到分类的结果，对于回归问题来说，我们可以采用计算模型均值的方法来作为最终预测的结果。

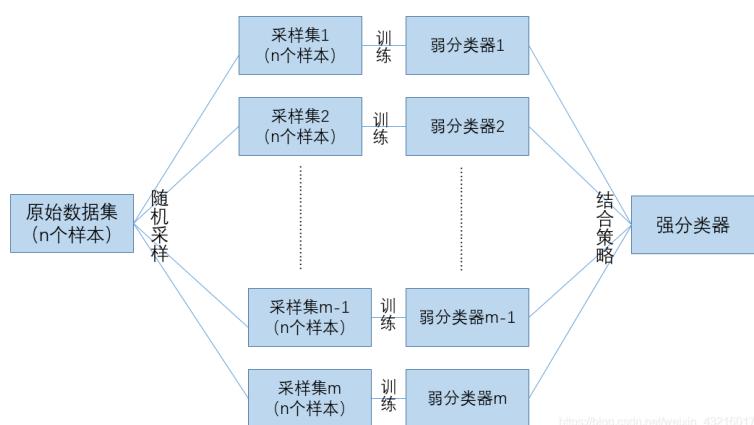


图 3-2: Bagging

### 3.5 随机森林

随机森林 (Random Forest, RF) 是 Bagging 的一个扩展变体。RF 在以决策树为基学习器构建 Bagging 集成的基础上，在决策树的训练过程中引入随机属性选择。训练每颗决策树时随机选出部分特征作为输入，所以该算法被称为随机森林算法。

在 RF 中，对基决策树的每个结点，先从该结点的属性集合中随机选择一个包含  $k$  个属性的子集（假定有  $d$  个属性），然后再从这个子集中选择一个最优属性用于划分。参数  $k$  控制了随机性的引入程度，一般情况下推荐  $k = \log_2 d$ 。随机森林的具体形式如下图：

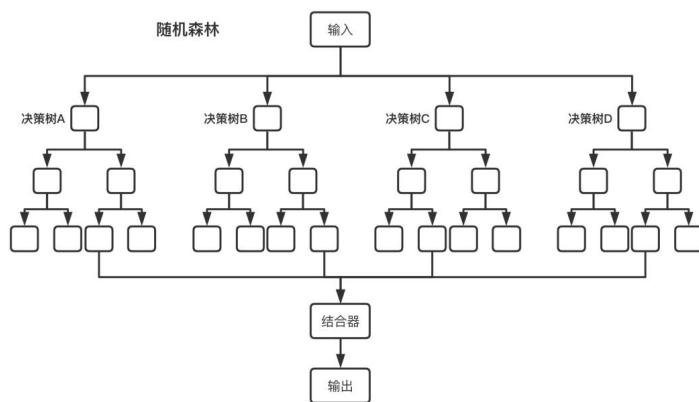


图 3-3: 随机森林的具体形式

假设训练集  $T$  的大小为  $N$ ，特征数目为  $M$ ，随机森林的大小为  $K$ ，随机森林算法的具体步骤如下：

1. 遍历随机森林的大小  $K$  次：
  - (1). 从训练集  $T$  中有放回抽样的方式，取样  $N$  次形成一个新子训练集  $D$
  - (2). 随机选择  $m$  个特征，其中  $m < M$
  - (3). 使用新的训练集  $D$  和  $m$  个特征，学习出一个完整的决策树
2. 得到随机森林

### 3.6 Stacking

在本次实验中以决策树、svm 和随机森林为基学习器，以决策树为元学习器。

Stacking 的做法是首先构建多个不同类型的一级学习器，并使用他们来得到一级预测结果，然后基于这些一级预测结果，构建一个二级学习器，来得到最终的预测结果。Stacking 的动机可以描述为：如果某个一级学习器错误地学习了特

征空间的某个区域，那么二级学习器通过结合其他一级学习器的学习行为，可以适当纠正这种错误。具体步骤如下图所示：

```

输入: 训练集  $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$ ;
初级学习算法  $\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_T$ ;
次级学习算法  $\mathcal{L}$ .
过程:
1: for  $t = 1, 2, \dots, T$  do
2:    $h_t = \mathcal{L}_t(D)$ ;
3: end for
4:  $D' = \emptyset$ ;
5: for  $i = 1, 2, \dots, m$  do
6:   for  $t = 1, 2, \dots, T$  do
7:      $z_{it} = h_t(\mathbf{x}_i)$ ;
8:   end for
9:    $D' = D' \cup ((z_{i1}, z_{i2}, \dots, z_{iT}), y_i)$ ;
10: end for
11:  $h' = \mathcal{L}(D')$ ;
输出:  $H(\mathbf{x}) = h'(h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_T(\mathbf{x}))$ 

```

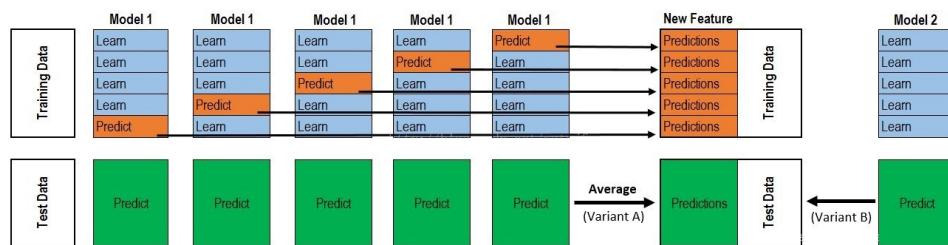
图 3-4: Stacking 具体步骤

过程 1-3 是训练出来个体学习器，也就是初级学习器。

过程 5-9 是使用训练出来的个体学习器来得预测的结果，这个预测的结果当做次级学习器的训练集。

过程 11 是用初级学习器预测的结果训练出次级学习器，得到我们最后训练的模型。

但是这样的实现是有很大的缺陷的。在原始数据集  $D$  上面训练的模型，然后用这些模型再  $D$  上面再进行预测得到的次级训练集肯定是非常好的。会出现过拟合的现象。因此一般通过使用交叉验证法或留一法这样的方式，用训练初级学习器未使用的样本来产生次级学习器的训练样本。以  $K$  折交叉为例，具体情况如下图：



[https://blog.csdn.net/qq\\_19446965](https://blog.csdn.net/qq_19446965)

图 3-5: K 折交叉

## 第四章 深度学习算法

### 4.1 文本

#### 4.1.1 BERT

传统的预训练方法存在一些问题，如单向语言模型的局限性和无法处理双向上下文的限制。为了解决这些问题，一种新的预训练方法随即被提出，即 BERT (Bidirectional Encoder Representations from Transformers)。通过在大规模无标签数据上进行预训练，BERT 可以学习到丰富的语言表示，从而在各种下游任务上取得优秀的性能。

BERT 与之前的语言表示模型不同，BERT 的设计目标是通过在所有层中联合考虑左右上下文，从无标签文本中预训练深度双向表示。因此，预训练的 BERT 模型只需添加一个额外的输出层，就可以用于各种任务，如问答和语言推理，而无需进行大量的任务特定架构修改。BERT 在概念上简单而实证强大，它在包括自然语言处理任务在内的十一个任务上取得了新的最先进结果。

#### 一. 输入形式

为了使得 BERT 模型适应下游的任务（比如说分类任务，以及句子关系 QA 的任务），输入将被改造成 [CLS]+ 句子 A (+[SEP]+ 句子 B+[SEP]）其中

1.[CLS]: 代表的是分类任务的特殊 token，它的输出就是模型的 pooler output。

2.[SEP]: 分隔符。

3. 句子 A 以及句子 B 是模型的输入文本，其中句子 B 可以为空，则输入变为 [CLS]+ 句子 A。

在 BERT 中，输入的向量是由三种不同的 embedding 求和而成，在以下所举的例子中，每个单词都表示为一个 768 维的向量。具体形式如下图：

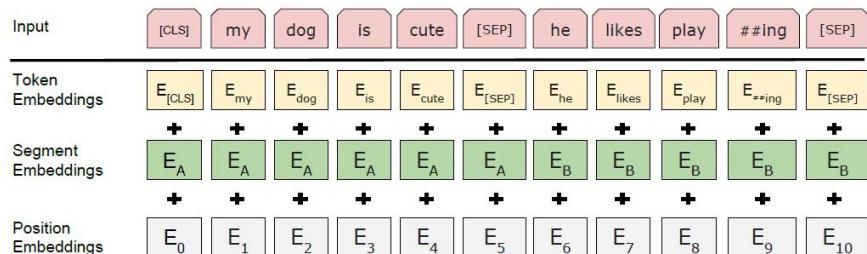


图 4-1: bert 输入部分

分别是 Token 嵌入层，Segment 嵌入层和 Position 嵌入层，以下是对其介绍：

1. token 嵌入层的作用是将单词转换为固定维的向量表示形式。在将输入文本传递到 token 嵌入层之前，首先对其进行 token 化。tokens 化是使用一种叫做 WordPiece token 化的方法来完成的。这是一种数据驱动的 token 化方法，旨在实现词汇量和非词汇量之间的平衡。token 嵌入层将每个 wordpiece token 转换为指定的高维向量表示形式。

2. Segment 嵌入层的作用是标记相同句子的每个词以区分不同的句子，假设我们的输入文本对是 (“I like cats”， “I like dogs”)，则 Segment 的标记过程如下图：

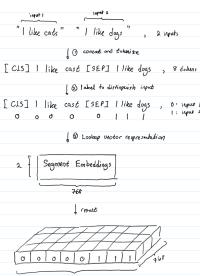


图 4-2: Segment 嵌入层

3. Position 嵌入层的作用为允许 BERT 理解给定的输入文本。例如语句 “I think, therefore I am”，第一个 I 和第二个 I 不应该用同一向量表示。假设 BERT 被设计用来处理长度为 512 的输入序列。作者通过让 BERT 学习每个位置的向量表示来包含输入序列的顺序特征。这意味着 Position 嵌入层是一个大小为 (512,768) 的查找表，其中第一行是第一个位置上的任意单词的向量表示，第二行是第二个位置上的任意单词的向量表示，等等。因此，如果我们输入 “Hello world” 和 “Hi there”，“Hello” 和 “Hi” 将具有相同的 Position 嵌入，因为它们是输入序列中的第一个单词。同样，“world” 和 “there”的 Position 嵌入是相同的。

综上可得出长度为 n 的 token 化输入序列将有三种不同的表示，即：

1. token 嵌入，形状 (1,n, 768)，这只是词的向量表示
2. Segment 嵌入，形状 (1,n, 768)，这是向量表示，以帮助 BERT 区分成对的输入序列。

3. Position 嵌入，形状 (1,n, 768)，让 BERT 知道其输入具有时间属性。

对这些表示进行元素求和，生成一个形状为 (1,n, 768) 的单一表示。这是传递给 BERT 的编码器层的输入表示。

## 二. 网络结构

多个 Transformer Encoder 一层一层地堆叠起来，就组装成了 BERT 了。BERT 是用了 Transformer 的 encoder 侧的网络，如下图的 transformer 的 Encoder 部分。

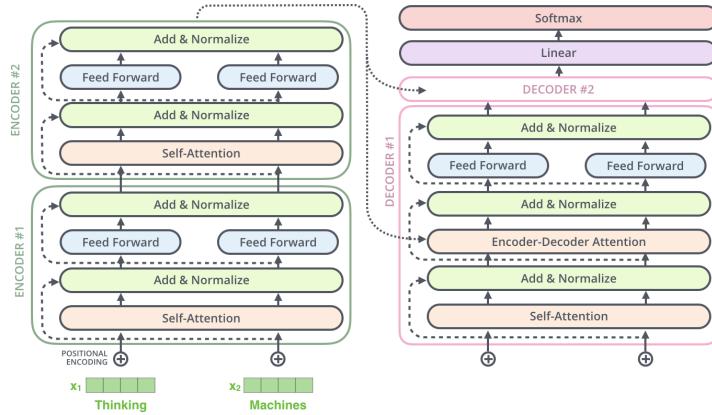


图 4-3: transformer 结构

transformer 的 encoder 先计算查询、键和值矩阵。我们通过将嵌入打包到矩阵 X 中，并将其乘以我们训练过的权重矩阵（WQ、WK、WV）来做到这一点。接着按照如下的公式计算：

$$\text{softmax} \left( \frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V}$$

$$= \mathbf{Z}$$

The diagram illustrates the calculation of attention weights. It shows three matrices: Q (purple), K<sup>T</sup> (orange), and V (blue). The Q matrix is a 3x3 grid. The K<sup>T</sup> matrix is a 3x3 grid. The V matrix is a 3x3 grid. The softmax function is applied to the product of Q and K<sup>T</sup> (divided by the square root of d<sub>k</sub>) to produce the Z matrix, which is a 3x3 grid.

图 4-4: transformer 计算原理

它扩展了模型关注不同位置的能力。是的，在上面的示例中，z1 包含一些其他编码，但它可能由实际单词本身主导。如果我们翻译一个句子，比如 “The Animal did not cross the street because it was tooert”，那么知道 “it” 指的是哪个单词会很有用。

它为注意力层提供了多个“表示子空间”。正如我们接下来将看到的，通过多头注意力，我们不仅拥有一组查询/键/值权重矩阵，而且拥有多组查询/键/值权重矩阵（Transformer 使用八个注意力头，因此我们最终为每个编码器/解码器提供八组）。这些集合中的每一个都是随机初始化的。然后，在训练之后，每个集

合用于将输入嵌入（或来自较低编码器/解码器的向量）投影到不同的表示子空间中。

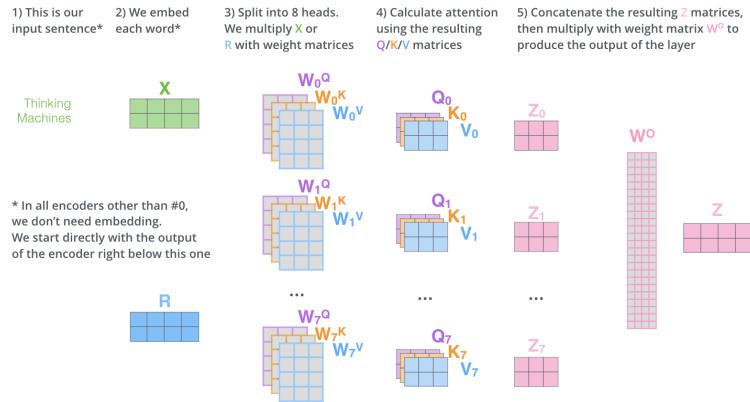


图 4-5: transformer 计算过程可视化

而两者存在着微小的差异，在 Transformer 中，模型的输入会被转换成 512 维的向量，然后分为 8 个 head，每个 head 的维度是 64 维，但是 BERT 的维度是 768 维度，然后分成 12 个 head，每个 head 的维度是 64 维。Transformer 中 position Embedding 是用的三角函数，BERT 中也有一个 Postion Embedding 是随机初始化，然后从数据中学出来的。

BERT 模型分为 24 层和 12 层两种，其差别就是使用 transformer encoder 的层数的差异，BERT-base 使用的是 12 层的 Transformer Encoder 结构。bert 模型大致如下图所示：

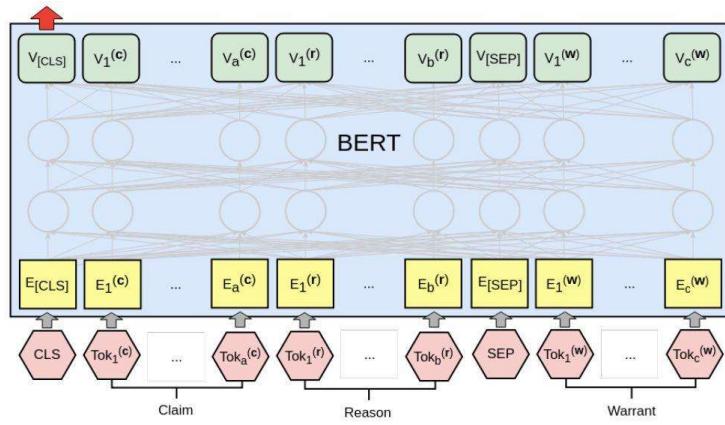


图 4-6: bert 模型结构

下图将注意力可视化为连接正在更新的单词（左）和正在关注的单词（右）的线，遵循上图的设计。颜色强度反映注意力权重；接近 1 的权重显示为非常暗的线条，而接近 0 的权重显示为微弱的线条或根本不可见。



图 4-7: 注意力可视化

下图跟踪从左侧所选单词到右侧完整单词序列的注意力计算。正值显示为蓝色，负值显示为橙色，颜色强度代表大小。

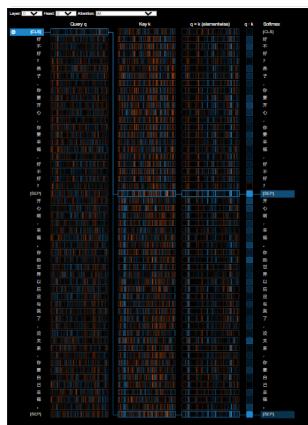


图 4-8: transformer 计算过程可视化

### 三. 模型预训练

#### 任务 1: Masked LM

即随机屏蔽 (masking) 部分输入 token，然后只预测那些被屏蔽的 token。在模型中，随机地屏蔽了每个序列中 15% 的 WordPiece token。训练数据生成器随机选择 15% 的 token。例如在这个句子 “my dog is hairy” 中，它选择的 token 是 “hairy”。然后，执行以下过程：

数据生成器将执行以下操作，而不是始终用 [MASK] 替换所选单词：

80% 的时间：用 [MASK] 标记替换单词，例如， my dog is hairy → my dog is [MASK]

10% 的时间：用一个随机的单词替换该单词，例如， my dog is hairy → my dog is apple

10%的时间：保持单词不变，例如，my dog is hairy → my dog is hairy. 这样做的目的是将表示偏向于实际观察到的单词。

Transformer encoder 不知道它将被要求预测哪些单词或哪些单词已被随机单词替换，因此它被迫保持每个输入 token 的分布式上下文表示。此外，因为随机替换只发生在所有 token 的 1.5% (即 15% 的 10%)，这似乎不会损害模型的语言理解能力。

### 任务 2：下一句预测

语料中 50% 的句子，选择其相应的下一句一起形成上下句，作为正样本；其余 50% 的句子随机选择一句非下一句一起形成上下句，作为负样本。而后进行训练，有利于 sentence-level tasks，例如问答。

总的来说，BERT 本质上是在海量语料的基础上，通过自监督学习的方法为单词学习一个好的特征表示。该模型的优点是可以根据具体的人物进行微调，或者直接使用预训练的模型作为特征提取器。

#### 4.1.2 BERT+TEXTCNN

我们进行模型的微调，将 bert 模型的最后一层的输出的内容作为 TextCNN 模型的输入，送入模型在继续进行学习，得到最终的结果，进行文本分类。

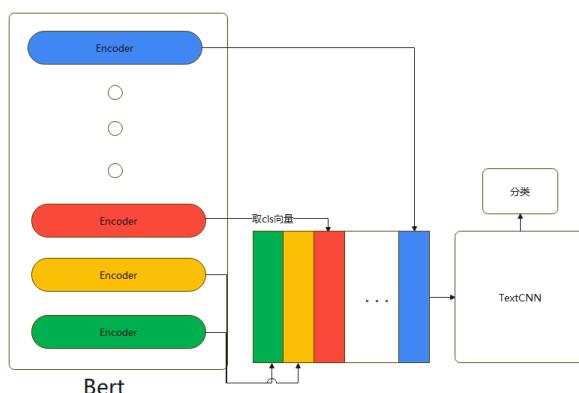


图 4-9: bert+textcnn

我们描述了三个滤波器区域大小：2, 3 和 4，每个都有 2 个滤波器。过滤器对句子矩阵执行卷积并生成（可变长度）特征图；对每个图执行 1-max 池化，即，记录来自每个特征图的最大数字。因此，从所有六个映射生成单变量特征向量，并且这 6 个特征被连接以形成倒数第二层的特征向量。最后的 softmax 层接收这个特征向量作为输入，并使用它来分类句子。

下面是 textcnn 的代码。首先，输入 x 增加一个维度。然后，对每个卷积层，

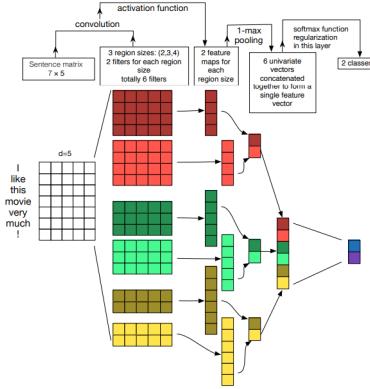


图 4-10: textcnn

执行卷积操作和 ReLU 激活函数，然后执行最大池化操作，最后将结果添加到  $pooled\_outputs$  列表中。接着，将所有池化的输出连接起来，然后将其重塑为一个二维张量。最后，通过线性层和偏置参数计算输出，然后返回结果。

```

class TextCnnModel(nn.Module):
    def __init__(self):
        super(TextCnnModel, self).__init__()
        self.num_filter_total = parsers().num_filters * len(parsers().filter_sizes)
        self.weight = nn.Linear(self.num_filter_total, parsers().class_num, bias=False)
        self.bias = nn.Parameter(torch.ones([parsers().class_num]))
        self.filter_list = nn.ModuleList([
            nn.Conv2d(1, parsers().num_filters, kernel_size=(size, parsers().hidden_size)) for size in parsers().filter_sizes
        ])

    def forward(self, x):
        # x: [batch_size, 12, hidden]
        x = x.unsqueeze(1) # [batch_size, channel=1, 12, hidden]

        pooled_outputs = []
        for i, conv in enumerate(self.filter_list):
            out = F.relu(conv(x)) # [batch_size, channel=2, 12-kernel_size[0]+1, 1]
            maxpool = nn.MaxPool1d(
                kernel_size=(parsers().encode_layer - parsers().filter_sizes[i] + 1, 1)
            )
            # maxpool: [batch_size, channel=2, 1, 1]
            pooled = maxpool(out).permute(0, 3, 2, 1) # [batch_size, h=1, w=1, channel=2]
            pooled_outputs.append(pooled)

        h_pool = torch.cat(pooled_outputs, len(parsers().filter_sizes)) # [batch_size, h=1, w=1, channel=2 * 3]
        h_pool_flat = torch.reshape(h_pool, [-1, self.num_filter_total]) # [batch_size, 6]

        output = self.weight(h_pool_flat) + self.bias # [batch_size, class_num]

        return output

```

图 4-11: bert+textcnn 代码

## 4.2 音频

### 4.2.1 CAM++

TDNN 与 1989 年就已提出 (在我出生之前)，用于音素识别。TDNN 在处理较宽的上下文输入时很有效。TDNN 的权重是共享的，每一帧的特征的权重是一样的，这就比线性层少很多参数了。

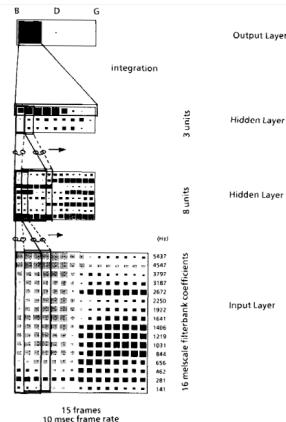


图 4-12: TDNN

正常的 CAM 模块如下图所示。但是 CAM++ 不一样。首先，输入  $x$  通过  $linear_{local}$  卷积层得到  $y$ 。然后，计算  $x$  的均值和段内池化的结果，得到上下文 context。接着，context 通过  $linear1$  卷积层和 ReLU 激活函数，然后通过  $linear2$  卷积层和 Sigmoid 激活函数，得到注意力权重  $m$ 。最后，返回  $y$  和  $m$  的乘积。CAM++ 采用了段内池化，首先，根据  $stype$  参数选择平均池化或最大池化。然后，将池化的结果扩展到原始输入的长度。最后，返回扩展后的结果。

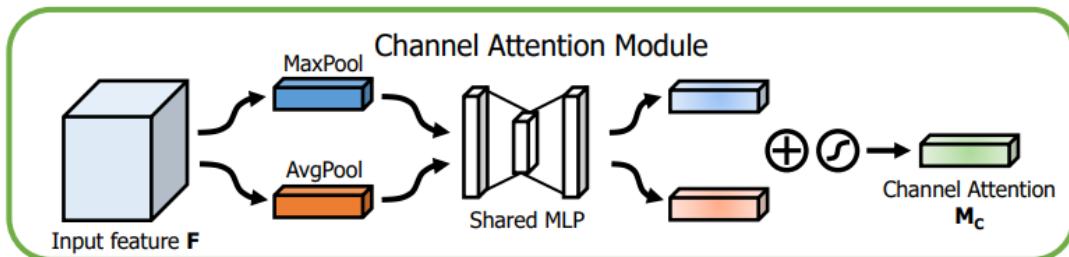


图 4-13: CAM

```
class CAMDenseTDNNBlock(nn.LayerList):
    def __init__(self,
                 num_layers,
                 in_channels,
                 out_channels,
                 bn_channels,
                 kernel_size,
                 stride=1,
                 dilation=1,
                 bias=False,
                 config_str='batchnorm-relu',
                 memory_efficient=False):
        super(CAMDenseTDNNBlock, self).__init__()
        for i in range(num_layers):
            layer = CAMDenseTDNNLayer(in_channels=in_channels + i * out_channels,
                                       out_channels=out_channels,
                                       bn_channels=bn_channels,
                                       kernel_size=kernel_size,
                                       stride=stride,
                                       dilation=dilation,
                                       bias=bias,
                                       config_str=config_str,
                                       memory_efficient=memory_efficient)
            self.add_sublayer('tdnn%d' % (i + 1), layer)

    def forward(self, x):
        for layer in self:
            x = paddle.concat([x, layer(x)], axis=-1)
        return x
```

图 4-14: Dense block

所提出的 CAM++ 体系结构的总体框架如图所示。该架构主要由两个组件组成：前端卷积模块（FCM）和 D-TDNN 骨干。FCM 由具有残差连接的多个二维卷积块组成，其在时频域中编码声学特征以利用高分辨率的时频细节。随后将得到的特征图沿信道和频率维度沿着变平，并用作 D-TDNN 的输入。D-TDNN 主干包括三个块，每个块包含一系列 D-TDNN 层。在每个 D-TDNN 层中，我们构建了一个改进的 CAM 模块，为内部 TDNN 层的输出特征分配不同的注意力权重。多粒度池合并了全局平均池和分段平均池，以有效地聚合不同级别的上下文信息。对于密集连接，掩码输出与所有前面的层连接在一起，并作为下一层的输入。

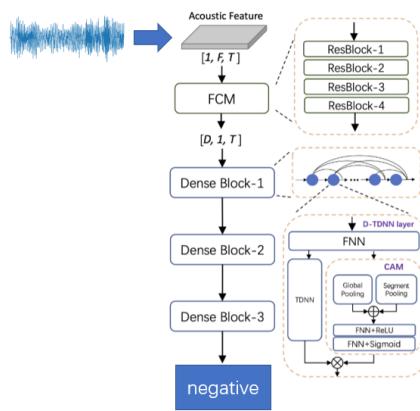
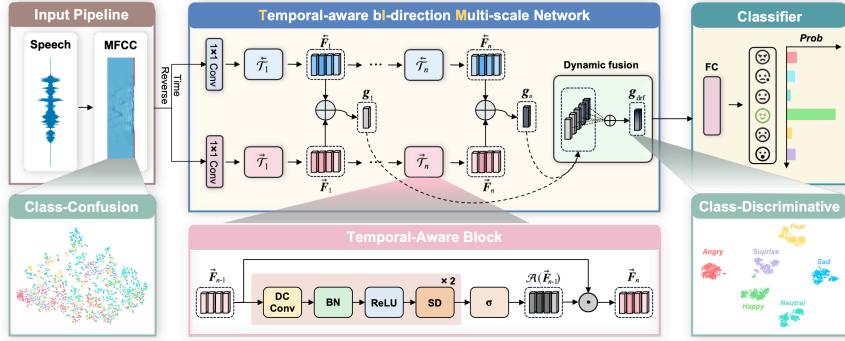


图 4-15: CAM++

#### 4.2.2 TIM-NET

在 SER，我们提出了一个时间感知的双向多尺度网络，称为 TIM-NET，这是一种新的时间情感建模方法，学习多尺度上下文情感表示从不同的时间尺度。贡献有三方面。首先，我们提出了一个基于扩张因果卷积（DC Conv）的时间感知块作为 TIM 网络的核心单元。扩张卷积能扩大和细化时间模式的感受野。与 RNN 相比，因果卷积与扩张卷积相结合可以帮助模型放松一阶马尔可夫性质的假设。通过这种方式，我们可以在网络中加入一个三阶（三阶表示所有先前帧的数量）连接，以聚合来自不同时间位置的信息。其次，我们设计了一种新的双向架构，整合了过去和未来的互补信息，用于建模长距离时间依赖关系。据我们所知，TIM-Net 是第一个专注于 SER 中多尺度融合的双向时间网络，而不是简单地连接前向和后向隐藏状态。第三，我们设计了一个动态融合模块，结合动态感受野学习不同时间尺度上的相互依赖关系，以提高模型的泛化能力。由于发音速度和暂停时间在说话者之间显著变化，语音需要不同的有效感受域（即，反映情

感特征的时间尺度) 的每个低级别特征 (例如, MFCC)。提出了用于情感特征学习的 TIM-NET 框架, 其特征提取部分由双向模块和动态融合模块组成。请注意, 前向和后向是具有不同输入的相同结构。



**Fig. 1.** The framework of the TIM-Net for learning affective features, whose feature extraction part is composed of a bi-direction module and a dynamic fusion module. Note that the forward  $\tilde{F}_j$  and backward  $\tilde{F}_j$  are the same structure with different inputs.

图 4-16: TIM-Net

当 input 的维度和 output 的维度相同时, 这种 padding 叫做 same padding。同时也叫做 half padding。同样, 我们可以根据 PPT 中的图片看出在包含 padding 的 conv 时, 输出维度和输入维度对应的关系。其中,  $p$  表示 padding 的维度, 在此处, 我们同样假定会在长和宽这两个维度进行相同数目的 padding。

当我们进行  $k/2$  的 padding 时, 我们运用刚才的公式可以得到如下结果。同时如果  $k$  是奇数 ( $2n+1$ ), 则通过推导可知刚好 output 维度 = input 维度。

$$\begin{aligned} o &= (i - k) + 2p + 1 \\ o &= i + 2\lfloor k/2 \rfloor - (k - 1) \\ &= i + 2n - 2n = i \end{aligned} \tag{4-1}$$

在实现上, 1D 的 casual 主要是通过 padding 来实现的。在 2D 的 casual 主要是通过 mask filter map 来实现的。

具体的, 对于采用大小为的卷积核  $k$ , 对于输入为  $x$  的空洞卷积 (dilated convolution) 函数的数学表达为:

$$F(s) = (\mathbf{x} *_d f)(s) = \sum_i^{k-1} f(i) \cdot x_{s-d \cdot i} \tag{4-2}$$

其中,  $s$  为输入的时间序列信息;  $d$  为空洞参数 (dilation parameter), 即空洞间隔大小; 代表空洞卷积算子 (dilated convolution operator) 或者称为 d-扩张卷积算子 (d-dilated convolution operator), 普通卷积算子则为空洞卷积算则在  $d=1$  时的一种特殊情况;  $s - d \cdot i$  指代对历史某一信息的定位。通过加入 dilated 机制,

当前的因果卷积网络就可以在网络深度（layer 数量）不变的情况下，通过增大卷积核大小（filter size: k）或增大空洞参数（dilation parameter: d）来扩大因果网络的感受视野。通常，我们设定：( $d = 2^i$ , i 代表网络的第 i 层)，即 d 的大小随网络深度呈指数型增长，从而确保了整体网络结构能够覆盖较长的历史信息。

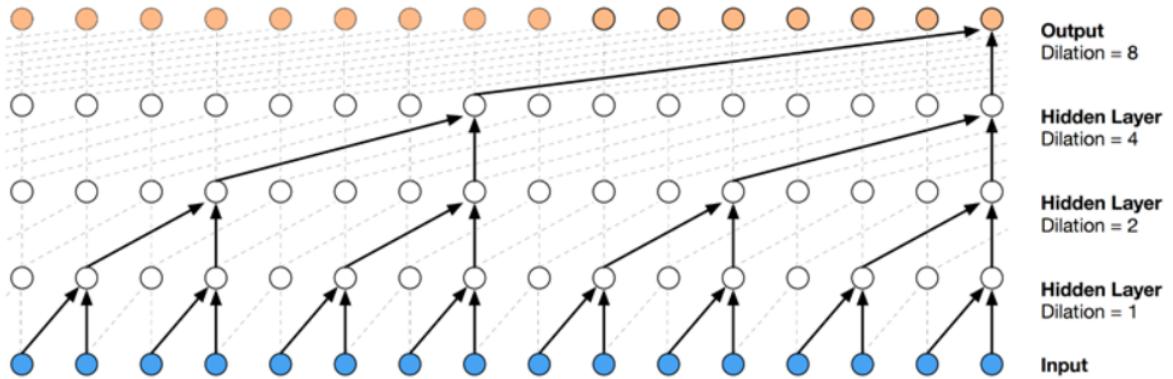


图 4-17: DC Conv

一旦生成了具有很强可分辨性的情感表示 bandrf，我们就可以简单地使用一个带有 softmax 函数的全连接层来进行情感分类。创建多个时序感知块，然后将它们的输出进行相加，池化，维度扩展，最后添加到最终的跳跃连接中。

```

forward_conv = Conv1D(filters=self.nb_filters,kernel_size=1,dilation_rate=1,padding='causal')(forward)
backward_conv = Conv1D(filters=self.nb_filters,kernel_size=1,dilation_rate=1,padding='causal')(backward)

final_skip_connection = []

skip_out_forward = forward_conv
skip_out_backward = backward_conv

for s in range(self.nb_stacks):
    for i in [2 ** i for i in range(self.dilations)]:
        skip_out_forward = Temporal_Aware_Block(skip_out_forward,s,i,self.activation,
                                                self_nb_filters,
                                                self_kernel_size,
                                                self_dropout_rate,
                                                name=self.name)
        skip_out_backward = Temporal_Aware_Block(skip_out_backward,s,i,self.activation,
                                                self_nb_filters,
                                                self_kernel_size,
                                                self_dropout_rate,
                                                name=self.name)

    temp_skip = add([skip_out_forward,skip_out_backward],name = "biadd_"+str(i))
    temp_skip = GlobalAveragePooling1D()(temp_skip)
    temp_skip=tf.expand_dims(temp_skip, axis=-1)
    final_skip_connection.append(temp_skip)

output_2 = final_skip_connection[0]
for i,item in enumerate(final_skip_connection):
    if i>0:
        Continue
    output_2 = K.concatenate([output_2,item],axis=-2)
x = output_2

return x

```

图 4-18: Temporal-aware block

## 4.3 视频

### 4.3.1 YOLOv8 目标检测

Ultralytics 最新发布的 YOLOv8 是一种尖端的、最先进的模型，它建立在以前的 YOLO 版本成功的基础上，并引入了新功能和改进，以进一步提高性能和灵活性。YOLOv8 大致可分为 Backbone —> Neck —> head。YOLOv8 设计为快

速、准确且易于使用，使其成为各种对象检测和跟踪、实例分割、图像分类和姿态估计任务的绝佳选择。

YOLOv8 的改进有如下的几个方面。在骨干网络（Backbone）中，YOLOv8 仍然采用 CSP 的思想，使用了 C2f 模块替换了 YOLOv5 中的 C3 模块，每个阶段的模块个数从 [3, 6, 9, 3] 修改为 [3, 6, 6, 3]，实现了进一步的轻量化，同时继续使用了 YOLOv5 中的 SPPF 模块；在颈部网络（Neck）中，将 YOLOv5 的 PAN-FPN 中的 top-down 上采样阶段中的卷积删除了，并且将 C3 模块替换为了 C2f 模块；在最后的预测头网络（Head），换成了目前主流的解耦头结构，将分类和检测头分离，同时也抛弃了 Anchor-Based，使用了 Anchor-Free 的思想。不难看出，Ultralytics 官方团队在实现模型轻量化与性能平衡的目标上做出了相当大的努力，而在工业中的端到端部署场景中还是因为计算内存开销过大而难以部署，这仍需进一步研究。

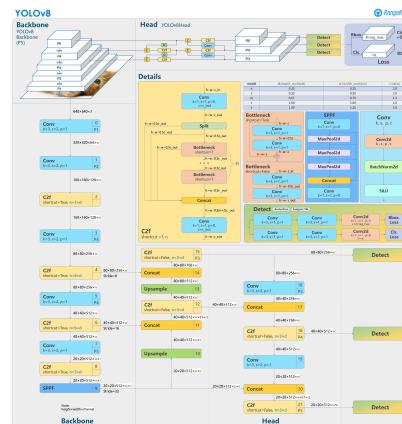


图 4-19: YOLOv8 结构

从下图所看 YOLOv 中的 Backbone 主要有 Conv, C2f, SPPF 三个模块，C2f 模块有 [3, 6, 6, 3],head 主要有下采样模块，C2f 模块，检测头将分类和检测头分离。计算 size 的公式：

$$\text{out\_size} = (\text{in\_size} - k + 2^*p)/s + 1 \quad (4-3)$$

```

backbone:
# [from, repeats, module, args]
- [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
- [-1, 1, C2f, [128, True]] # 1-P2/4
- [-1, 3, C2f, [128, True]] # 3-P3/8
- [-1, 1, Conv, [256, 3, 2]] # 5-P4/16
- [-1, 1, C2f, [256, True]] # 7-P5/32
- [-1, 1, Conv, [512, 3, 2]] # 9-P6/64
- [-1, 1, SPPF, [1024, 5]] # 9-P7/32
# YOLOv8.on head
head:
- [-1, 1, nn.Upsample, [None, 2, 'nearest']] # cat backbone P4
- [[-1, 6], 1, Concat, [1]] # cat backbone P3
- [-1, 3, C2f, [512]] # 12-P3/8-small
- [-1, 1, nn.Upsample, [None, 2, 'nearest']] # cat backbone P3
- [[-1, 4], 1, Concat, [1]] # cat backbone P3
- [-1, 3, C2f, [256]] # 15-(P3/8-medium)
- [-1, 1, Conv, [512, 3, 2]] # cat head P5
- [[-1, 9], 1, Concat, [1]] # cat head P5
- [-1, 3, C2f, [1024]] # 23-(P5/32-large)
- [[15, 18, 21], 1, Detect, [nc]] # Detect(P3, P4, P5)

```

图 4-20: YOLOv8 目标检测结构

我们训练一个网络将人脸检测出来，并截图下来，为我们后面进行情绪分类做好准备。

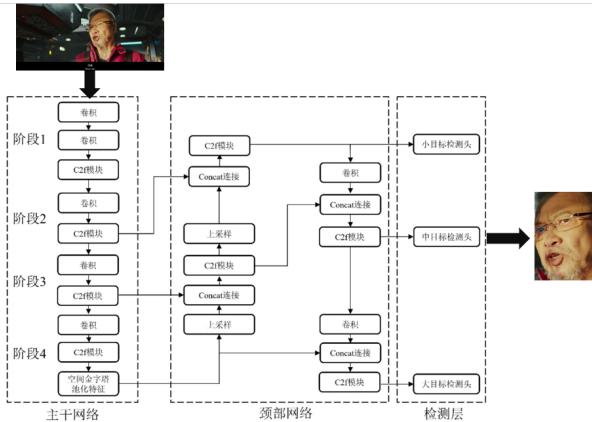


图 4-21: YOLOv8 人脸检测

### 4.3.2 YOLOv8 目标分类

从下图可以看出，backbone 和检测的差不多，只不过少了 SPPF 模块。对比检测网络，可以发现分类网络结构没有 NECK 部分，head 也只有分类头。

```
backbone:  
    # [from, repeats, module, args]  
    - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2  
    - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4  
    - [-1, 3, C2f, [128, True]]  
    - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8  
    - [-1, 6, C2f, [256, True]]  
    - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16  
    - [-1, 6, C2f, [512, True]]  
    - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32  
    - [-1, 3, C2f, [1024, True]]  
  
    # YOLOv8.0n head  
head:  
    - [-1, 1, classify, [nc]] # Classify
```

图 4-22: YOLOv8 目标分类结构

下图展示了使用 YOLOV8 对图片进行分类的流程图。

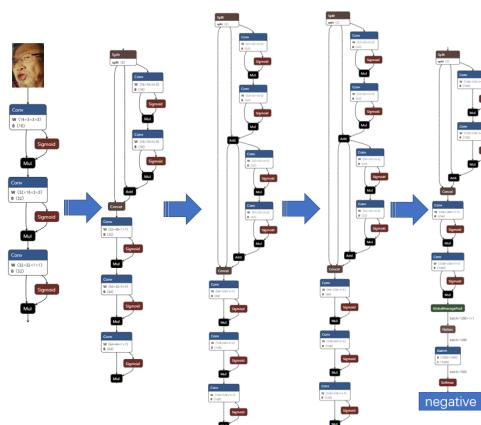


图 4-23: YOLOv8 情绪分类

## 第五章 多模态学习算法

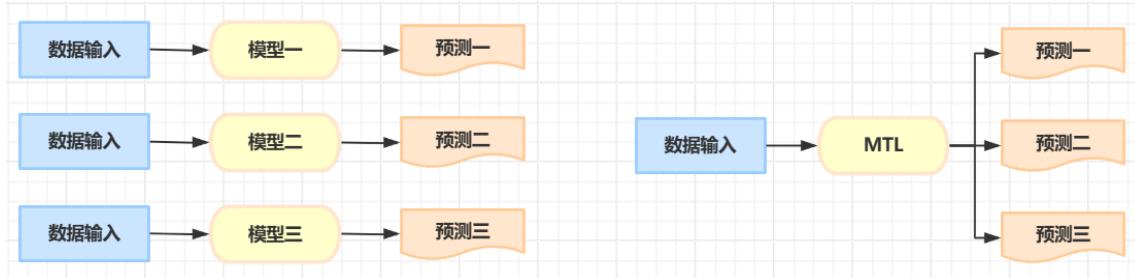


图 5-1: 单任务学习 VS 多任务学习

单任务学习是传统的机器学习方法，它假设不同的任务之间是相互独立的，因此每个模型的参数都是独立训练的。在这种方法中，每个任务都是一个单独的训练问题，模型需要分别学习每个任务的特定特征和规律。相比之下，多任务学习认为不同任务之间实际上是具有一定的关联性。因此，多任务学习采用的方法是将不同任务之间的训练数据集成在一起，然后联合同时训练多个不同任务的模型参数。这种学习方法可以共享多个任务之间的信息和知识，从而提高模型的学习能力，增强模型的泛化性。

### 5.1 MTFN

MTFN 模型（Multi-Task Fusion Network）是一种多任务学习模型，旨在提高情感分析的准确性。该模型通过融合多个相关任务的信息，共享底层网络参数，从而提高每个任务的性能。MTFN 模型的核心思想是利用多任务学习的优势，将多个相关任务同时进行训练。这些任务可以是不同语言、不同模态的情感分析任务，如文本分类、语音识别、图像识别等。多模态数据的融合是将多个单模态表示集成到一个紧凑的多模态表示中的过程。通过共享底层网络参数，MTFN 模型能够有效地利用不同任务之间的信息，从而提高每个任务的性能。我们提出的TFN由三个主要组成部分组成：1) 模态嵌入子网络将单峰特征作为输入，并输出丰富的模态嵌入。2) 张量融合层使用来自模态嵌入的 3 重笛卡尔积显式地对单峰、双峰和三峰相互作用进行建模。3) 情感推理子网络是以张量融合层的输出为条件的网络，并执行情感推理。

#### 1. 模态嵌入子网：

##### (1). Spoken Language Embedding Subnetwork

网络通过使用时间相关嵌入的非线性仿射变换来简单地关注相关部分，编码可由流水线的其余部分使用，所述时间相关嵌入的非线性仿射变换可充当维数减少注意机制。带有遗忘门的 LSTM 网络 (Hochreiter 和 Schmidhuber, 1997) (Gers 等人, 2000) 用于学习时间相关的语言表示，根据以下 LSTM 公式计算单词。

$$\begin{pmatrix} i \\ f \\ o \\ m \end{pmatrix} = \begin{pmatrix} \text{sigmoid} \\ \text{sigmoid} \\ \text{sigmoid} \\ \tanh \end{pmatrix} W_{l_d} \begin{pmatrix} X_t W_{l_e} \\ h_{t-1} \end{pmatrix} \quad (5-1)$$

$$c_t = f \odot c_{t-1} + i \odot m$$

$$h_t = o \otimes \tanh(c_t)$$

$$\mathbf{h}_l = [h_1; h_2; h_3; \dots; h_{T_l}]$$

```
class TextSubNet(nn.Module):
    """
    The LSTM-based subnetwork that is used in TIN for text
    ...
    def __init__(self, in_size, hidden_size, out_size, num_layers=1, dropout=0.2, bidirectional=False):
        ...
        Args:
            in_size: input dimension
            hidden_size: hidden layer dimension
            num_layers: number of layers of LSTM
            dropout: dropout probability
            bidirectional: specify usage of bidirectional LSTM
        ...
        (return value in forward) a tensor of shape (batch_size, out_size)
        super(TextSubNet, self).__init__()
        if num_layers > 1:
            self.rnn = nn.LSTM(in_size, hidden_size, num_layers=num_layers, dropout=dropout, bidirectional=bidirectional, batch_first=True)
            self.rnn = nn.Dropout(dropout)
            self.rnn = nn.Linear(hidden_size, out_size)
        else:
            self.rnn = nn.LSTM(in_size, hidden_size, num_layers=num_layers, dropout=dropout, bidirectional=bidirectional, batch_first=True)
            self.rnn = nn.Dropout(dropout)
            self.rnn = nn.Linear(hidden_size, out_size)
        ...
    def forward(self, x):
        ...
        x: tensor of shape (batch_size, sequence_len, in_size)
        ...
        final_states = self.rnn(x)
        h = self.dropout(final_states[0]).squeeze(0)
        y_1 = self.linear(h)
        return y_1
```

图 5-2: textwork

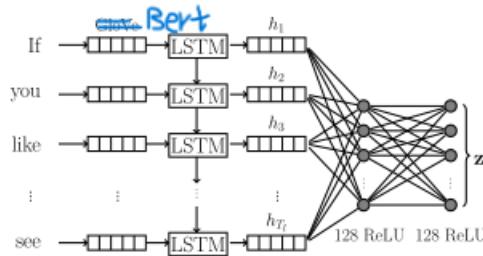


图 5-3: text 结构

## (2).Visual Embedding Subnetwork 和 Acoustic Embedding Subnetwork

我们可以从下图看到，首先，输入  $x$  被传递给批量归一化层。然后，归一化后的数据被传递给 Dropout 层。最后，dropout 后的数据被依次传递给三个线性层，每个线性层后都跟着一个 ReLU 激活函数。

通过整合不同模态的特征来捕捉它们之间的交互信息，并输出丰富的模态嵌入表示。

```

class SubNet(nn.Module):
    The subnetwork that is used in TFM for video and audio in the pre-fusion stage
    ...
    def __init__(self, in_size, hidden_size, dropout):
        ...
        Args:
            in_size: input dimension
            hidden_size: hidden layer dimension
            dropout: dropout probability
        ...
        return value in forward() a tensor of shape (batch_size, hidden_size)
    ...
    super(SubNet, self).__init__()
    ...
    self.linear1 = nn.Linear(in_size, hidden_size)
    self.linear2 = nn.Dropout(p=dropout)
    self.linear3 = nn.Linear(hidden_size, hidden_size)
    self.linear4 = nn.Linear(hidden_size, hidden_size)

    def forward(self, x):
        ...
        Args:
            -x: tensor of shape (batch_size, in_size)
        normed = self.norm(x)
        x = self.linear1(normed)
        x = F.relu(self.linear2(x))
        x = self.linear3(x)
        x = F.relu(self.linear4(x))
        ...
        return x

```

图 5-4: subet

## 2. 张量融合层:

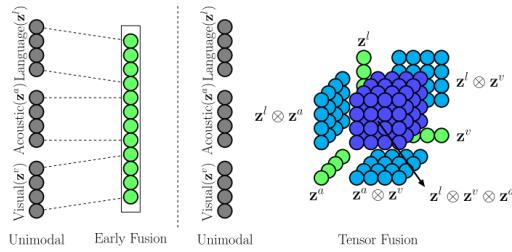


图 5-5: 张量融合三种类型的子张量: 单峰, 双峰和三峰

张量融合层是 MTFN 模型的核心部分，它负责将多个任务的特征进行融合。张量融合层的设计需要考虑不同任务之间的关联性和差异性。在 MTFN 模型中，张量融合层使用 3-fold 笛卡尔积显式地模拟单模态、双模态和三模态相互作用。这种设计可以有效地利用不同模态之间的信息，提高情感分析的准确性。

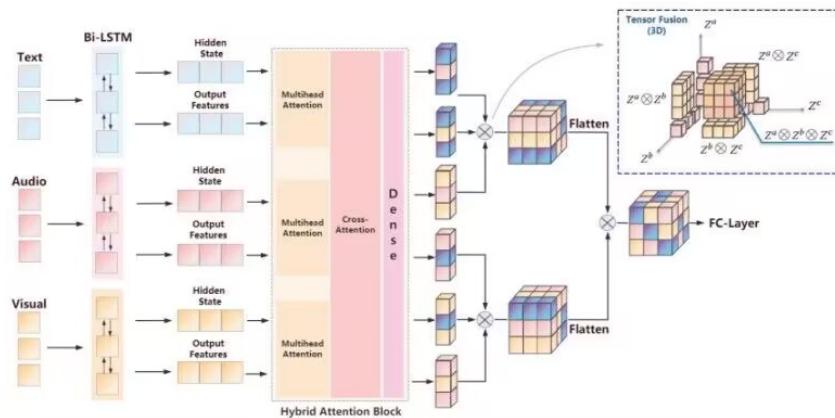


图 5-6: 多张量网络

该定义在数学上等同于  $z^l$ 、视觉表示  $z^v$  和声学表示  $z^a$  之间的可微外积，如下公式所示：

$$\mathbf{z}^m = \begin{bmatrix} \mathbf{z}^l \\ 1 \end{bmatrix} \otimes \begin{bmatrix} \mathbf{z}^v \\ 1 \end{bmatrix} \otimes \begin{bmatrix} \mathbf{z}^a \\ 1 \end{bmatrix} \quad (5-2)$$

### 3. 情感推理子网：

情感推理子网络是 MTFN 模型的另一个重要组成部分，它以张量融合层的输出为条件进行情感推理。情感推理子网络的设计也需要考虑不同任务之间的差异性和关联性。在 MTFN 模型中，情感推理子网络使用多头自注意力机制对张量融合层的输出进行进一步的处理，以生成最终的情感预测结果。情感推理子网络的似然函数定义如下，其中  $\phi$  是情感预测：

$$\arg \max_{\phi} p(\phi | \mathbf{z}^m; W_s) = \arg \max_{\phi} \mathcal{U}_s(\mathbf{z}^m; W_s) \quad (5-3)$$

## 5.2 MLMF

多模态融合 MLMF 模型是一种基于多模态融合技术的算法，它通过将来自不同模态的信息进行整合，利用多模态数据之间的互补性来提高准确性和鲁棒性。在多模态融合 MLMF 模型中，通常会采用低秩进行多模态融合张量来提高效率。通过将不同模态的数据进行线性变换和多维度点积，可以得到多个低秩向量的结果，从而减少了模型中的参数数量，提高了模型的效率和可解释性。

### 1. Multimodal Fusion using Tensor Representations

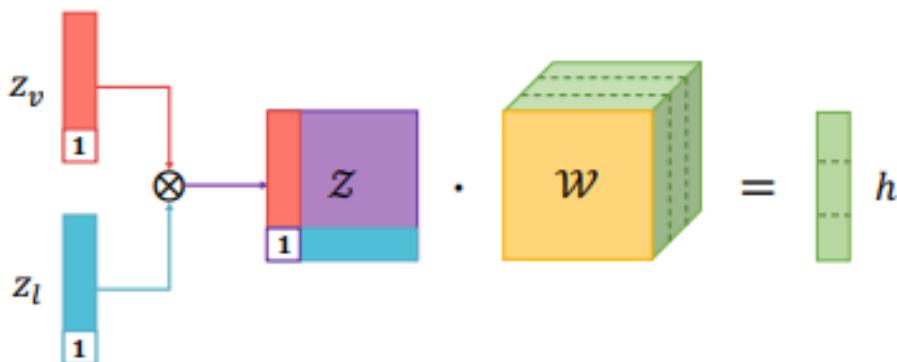


图 5-7: 传统的张量外积的张量融合

具体公式为：

$$\mathcal{Z} = \bigotimes_{m=1}^M z_m, z_m \in \mathbb{R}^{d_m} \quad (5-4)$$

$$h = g(\mathcal{Z}; \mathcal{W}, b) = \mathcal{W} \cdot \mathcal{Z} + b, h, b \in \mathbb{R}^{d_y} \quad (5-5)$$

张量融合的一个主要缺点是我们必须显式地创建高维张量  $Z$ 。 $Z$  的维数将随着模态的数量以指数方式增加，因为  $\prod_{m=1}^M d_m$ 。权重张量  $W$  中要学习的参数的数量也将呈指数级增加。这不仅引入了大量的计算，而且使模型暴露于过拟合的风险。

## 2.Low-rank Multimodal Fusion with Modality-Specific Factors

低秩多模态融合是一种多模态融合方法，它通过将不同模态的数据进行低秩分解，并利用模态特定的低阶因子来执行多模态融合。这种方法能够减少模型中的参数数量，降低内存开销，并将指数级的时间复杂度降低到线性。在多模态融合中，低秩多模态融合模型能够随着模态的数量线性缩放，并且可以有效执行基于张量的融合。该模型避免了计算高维张量，降低了内存开销，将指数级的时间复杂度降低到了线性。张量表示的多模态融合是一种成功的方法，用于将多模态数据融合到一个紧凑的多模态表示中。这种方法首先将多输入转换为高维张量，然后将其映射回一个低维输出向量空间。通过对输入模态取外积可以得到张量表示。

我们把低秩的因子分解成  $M$  阶 3 张量，并交换我们进行元素乘积和求和的顺序：

$$h = \sum_{i=1}^r \left[ \prod_{m=1}^M [\mathbf{w}_m^{(1)}, \mathbf{w}_m^{(2)}, \dots, \mathbf{w}_m^{(r)}] \cdot \hat{\mathbf{z}}_m \right]_{i,:} \quad (5-6)$$

并且现在沿着带括号的矩阵的第一维沿着进行求和。通过这种方式，我们可以用  $M$  阶 3 张量来参数化模型，而不是用向量集来参数化。

多模态融合的目标是将单模态的表示整合为一个紧凑的多模态表示，以便进行下游的工作。通过使用张量表示，可以模拟任意模态子集之间的相互作用。

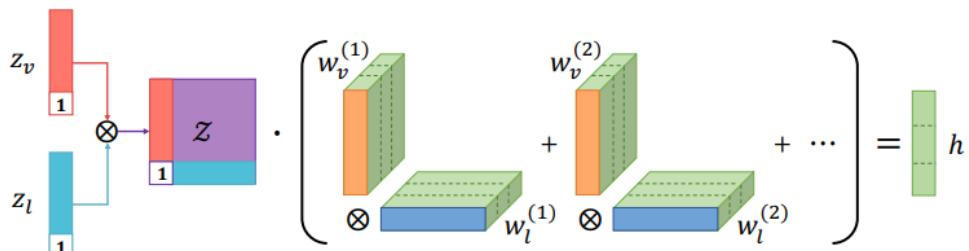


图 5-8: 将权重张量分解为低秩因子

MLMF 模型低秩多模态融合之后，可以进一步进行以下操作：

**特征提取：**利用低秩因子进行特征提取，提取出数据中的有用特征。这些特征可以用于后续的分类、聚类等任务中。

**预测：**将低秩因子作为输入，利用训练好的模型进行预测。这可以帮助我们根据不同的输入数据得到相应的输出结果。

**评估：**对模型的预测结果进行评估，比较模型的表现和预测精度。这可以帮助我们了解模型的性能，并进一步优化模型。

## 第六章 训练过程及结果

### 6.1 机器学习

五轮 10 折交叉验证代码形式如下图：

```

for y in range(5):
    accuracy = []
    precision = []
    recall = []
    F1 = []
    f1a = []
    kf = KFold(n_splits=10, shuffle=True, random_state=y + 10)
    for train, test in kf.split(data, train_labels):
        adaboost_classifier = AdaBoostClassifier(n_estimators=25, random_state=42)
        adaboost_classifier.fit(data[train], train_labels[train])
        predicts = adaboost_classifier.predict(data[test])
        accuracy1 = accuracy_score(train_labels[test], predicts)
        accuracy.append(accuracy1)
        precision1 = precision_score(train_labels[test], predicts, average='macro')
        precision.append(precision1)
        recall1 = recall_score(train_labels[test], predicts, average='macro')
        recall.append(recall1)
        F1 = f1_score(train_labels[test], predicts, average='macro')
        F1a.append(F1)
        F1 = f1_score(train_labels[test], predicts, average='micro')
        F1.append(F1)
    mean_accuracy = np.mean(accuracy)
    mean_accuracy_every.append(mean_accuracy)
    mean_precision = np.mean(precision)
    mean_precision_every.append(mean_precision)
    mean_recall = np.mean(recall)
    mean_recall_every.append(mean_recall)
    mean_f1a = np.mean(F1a)
    mean_f1a_every.append(mean_f1a)
    mean_F1 = np.mean(F1)
    mean_F1_every.append(mean_F1)

```

图 6-1: 五轮 10 折交叉验证

#### 6.1.1 文本

对文本数据用 jieba 库进行分词，而后用 tokenizer 函数构建词表，而后将文本输入得到每个文本对应的句向量。句向量由文本中每个词在词表中的索引。由于数据集中文本次数较少，故设置特征向量维数为 100，不足则用 0 填充。

而后先分别对 6 个模型进行五轮 10 折交叉验证并选择最佳参数，选择其中效果最好的三个模型，将处理后的特征矩阵输入模型进行训练和预测。其中 bagging 模型以决策树为初始分类器;stacking 模型以 svm、梯度提升树和随机森林为基学习器，用决策树为元学习器。

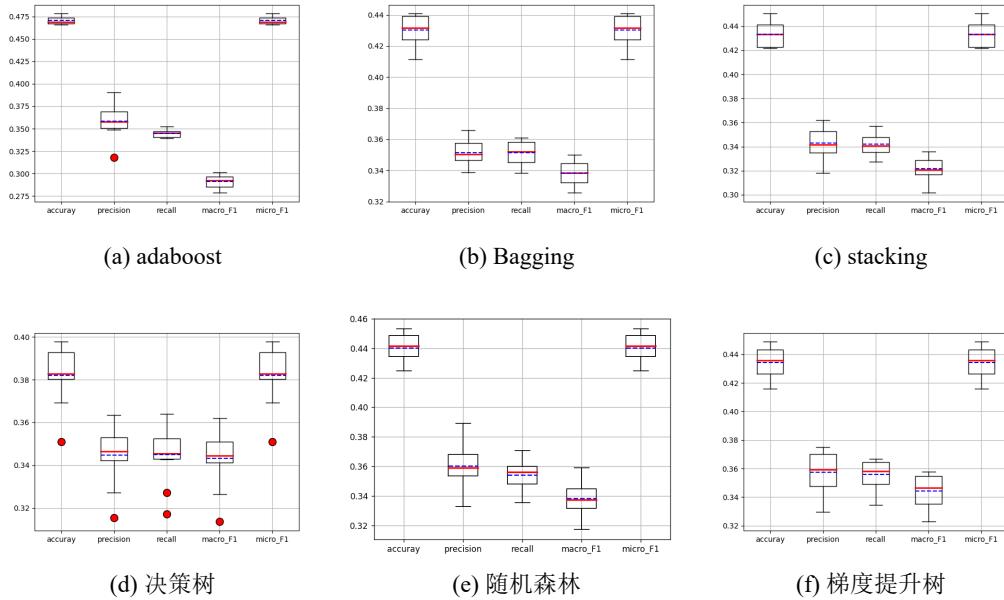


图 6-2: 文本不同模型的结果

### 6.1.2 音频

对采集到的数据进行拉长处理，使输入矩阵变为二维矩阵，先分别对 6 个模型进行五轮 10 折交叉验证并选择最佳参数，选择其中效果最好的三个模型，而后输入模型中进行训练和预测，其中 bagging 模型和 stacking 模型分类器选择不变。

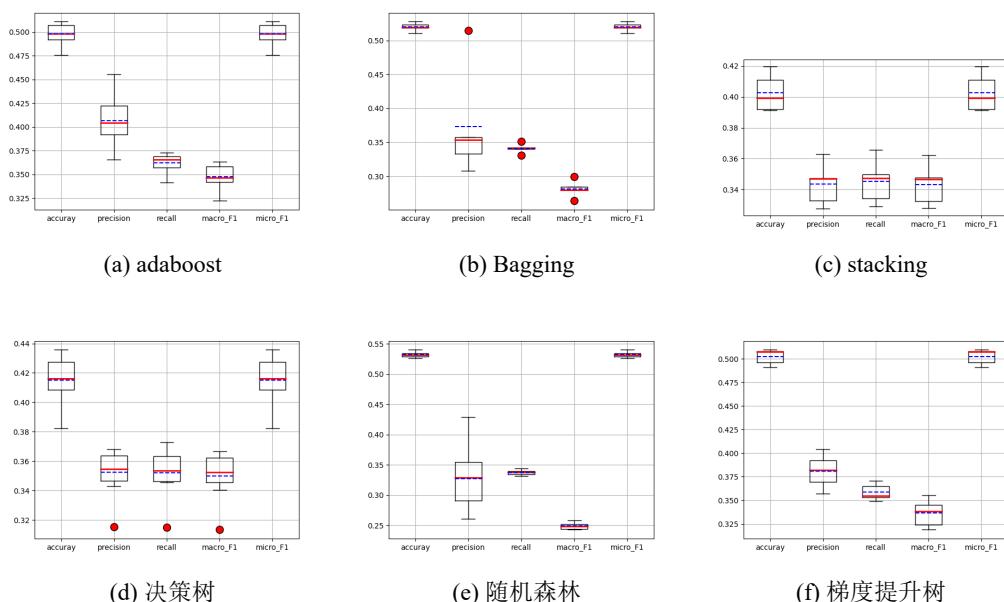


图 6-3: 音频不同模型的结果

### 6.1.3 视频

对采集到的数据进行拉长处理，使输入矩阵变为二维矩阵，而后分别对 6 个模型进行五轮 10 折交叉验证并选择最佳参数，选择其中效果最好的三个模型，输入模型中进行训练和预测，其中 bagging 模型和 stacking 模型分类器选择不变。

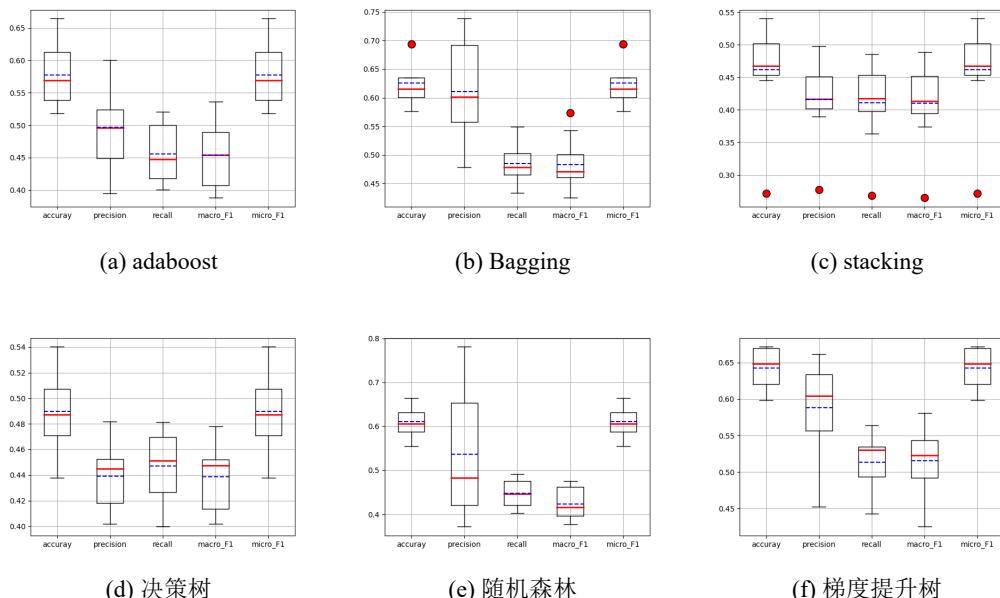


图 6-4: 视频不同模型的结果

#### 6.1.4 总体

对文本，音频和视频的分类结果分别用加权和投票方式得出最终结果，发现加权方法效果不佳，故使用投票方式得出最终结果。选取三个最好的模型，进行投票得到结果。

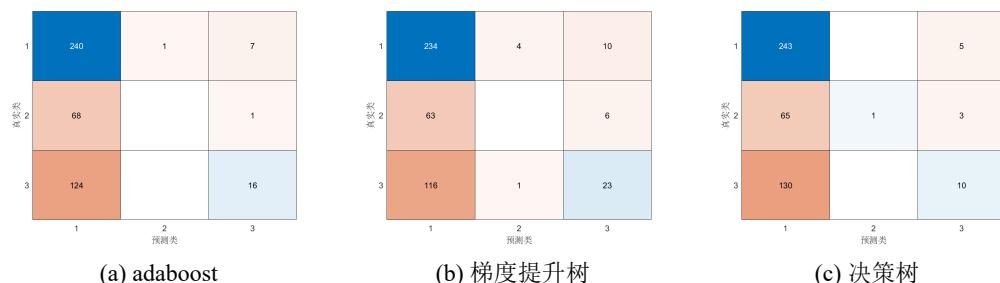


图 6-5: 不同模型的投票结果

## 6.2 深度学习

### 6.2.1 文本

#### 1.bert

输入数组大小为(47,768), 迭代10次, 学习率为 $1e-5$ , 训练策略为Adamw, 损失函数为CrossEntropyLoss(它结合了LogSoftmax和NLLLoss(负对数似然损失)两个操作)。

#### 2.bert+cnn

输入数组大小为(47,768), 迭代10次, 学习率为 $1e-5$ , 训练策略为Adamw, 损失函数为CrossEntropyLoss, dropout为0.5。

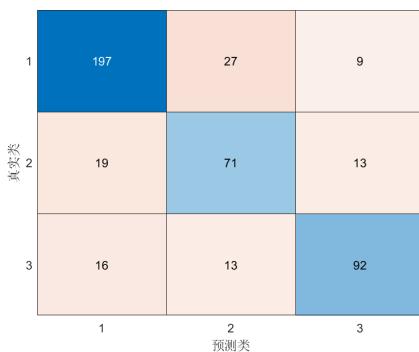


图 6-6: bert

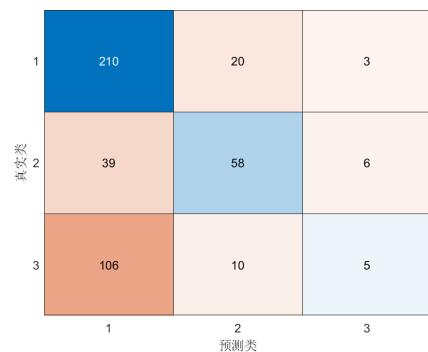


图 6-7: bert+cnn

### 6.2.2 音频

#### 1.cam++

音频预处理方法为Fbank, 采样率为16000, 滤波器为40, optimizer为AdamW, 学习率衰减函数为CosineAnnealingLR, 学习率为0.001, 训练轮数为15轮

#### 2.tim-net

音频预处理方法为librosa, 采样率为16000, nmfcc为39维, 训练轮数为100轮, 进行五折交叉验证, batchsize为64, dropout为0.1, optimizer为Adam。

	1	146	85	
真实类	2		55	24
	3	41		106
		1	2	3
预测类				

图 6-8: cam++

	1	196	35	
真实类	2		63	16
	3	29		118
		1	2	3
预测类				

图 6-9: tim-net

### 6.2.3 视频

#### 1.yolov8

先用检测模型得出人脸具体在的位置，再用分类模型进行分类。训练轮数为 15 轮，图片大小为 224x224,dropout 为 0.5,batch 为 32，使用余弦学习策略。

	1	218	9	7
真实类	2	23	41	2
	3	67	3	87
		1	2	3
预测类				

图 6-10: yolov8

### 6.2.4 总体

#### Overall 1. 使用加权得分法

我们将文本预测结果乘以 0.4，音频和视频结果分别乘以 0.3，选择靠近最近的整数为总体预测结果，但是效果不尽人意。

#### 2. 使用多数表决法

取文本，音频，视频预测结果出现最多相同的类别为总体预测结果，预测效果比加权得分法优秀。

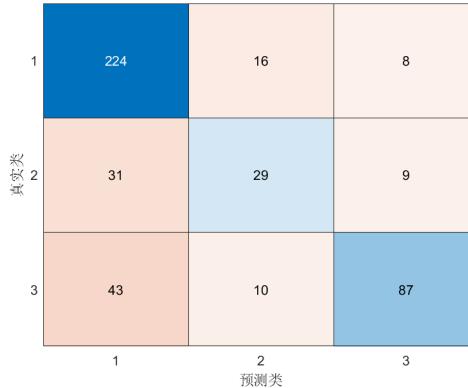


图 6-11: 总体—深度学习

## 6.3 多模态学习

### 6.3.1 MLMF

训练配置如下图所示：

```
MMSA - ===== Program Start =====
MMSA - Running with args:
MMSA - {'model_name': 'mlmf', 'dataset_name': 'sims', 'featurePath': '/home/sharing/disk3/Datasets/MMSA-Standard\LSIMS/Processed/unaligned_39.pkl', 'seq_lens': [39, 400, 55], 'feature_dims': [768, 33, 709], 'train_samples': 1368, 'num_classes': 3, 'language': 'cn', 'KeyEval': 'Loss', 'missing_rate': [0.2, 0.2, 0.2], 'missing_seed': [1111, 1111, 1111], 'need_data_aligned': False, 'need_model_aligned': False, 'need_normalized': True, 'early_stop': 8, 'hidden_dims': [256, 32, 256], 'post_text_dim': 8, 'post_audio_dim': 4, 'post_video_dim': 32, 'post_dropouts': [0.4, 0.4, 0.4, 0.4], 'dropouts': [0.3, 0.3, 0.3], 'rank': 3, 'batch_size': 16, 'learning_rate': 0.001, 'factor_lr': 0.001, 'M': 0.2, 'T': 1.0, 'A': 0.4, 'V': 0.8, 'text_weight_decay': 0.0001, 'audio_weight_decay': 0.0, 'video_weight_decay': 0.0001, 'weight_decay': 0.005, 'model_save_path': 'WindowsPath\model/mlmf-sims.onnx'}, 'device': device(type='cpu'), 'train_mode': 'classification', 'custom_feature': None, 'feature_T': None, 'feature_A': None, 'feature_V': None}
```

图 6-12: MLMF 训练配置

训练结果：

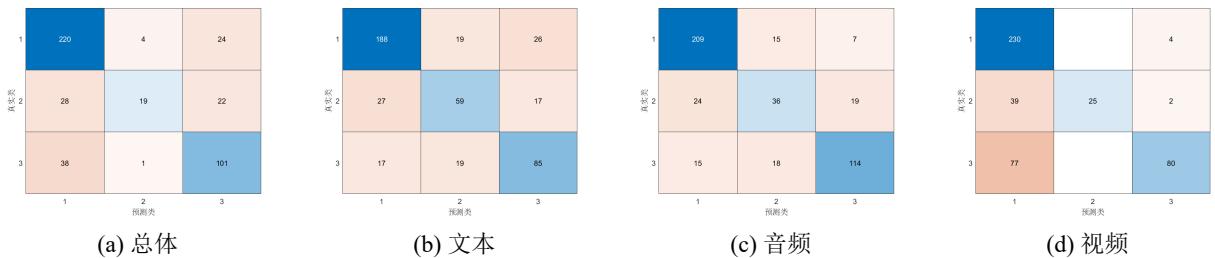


图 6-13: MLMF 多任务分类的结果

### 6.3.2 MTFN

训练配置如下图所示：

```
MMSA - ----- Program Start -----
MMSA - Running with args:
MMSA - {'model_name': 'mtfn', 'dataset_name': 'sims', 'featurePath': '/home/shanping/disk2/Datasets/MMSA Standard\STMS/Processed/unaligned_39.pkl', 'seq_lens': [39, 400, 405], 'feature_dims': [768, 33, 709], 'train_samples': 1368, 'num_classes': 3, 'language': 'cn', 'KeyEval': 'Loss', 'missing_rate': [0.2, 0.2, 0.2], 'missing_seed': [1111, 1111, 1111], 'need_data_aligned': False, 'need_model_aligned': False, 'need_normalized': True, 'early_stop': 8, 'hidden_dims': [256, 32, 256], 'text_out': 256, 'post_fusion_dim': 64, 'post_text_dim': 64, 'post_audio_dim': 4, 'post_video_dim': 8, 'dropouts': [0.2, 0.2, 0.2], 'post_dropouts': [0.2, 0.2, 0.2, 0.2], 'batch_size': 16, 'learning_rate': 0.005, 'M': 0.6, 'T': 0.4, 'A': 0.0, 'V': 1.0, 'text_weight_decay': 0.001, 'audio_weight_decay': 0.0001, 'video.weight_decay': 1e-05, 'weight_decay': 0.005, 'model_save_path': WindowsPath('model/mtfn-sims.onnx')}, 'device': device(type='cpu'), 'train_mode': 'classification', 'custom_feature': None, 'feature_T': None, 'feature_A': None, 'feature_V': None}
MMSA - Seeds: [1111, 1112, 1113, 1114, 1115]
MMSA - ----- Running with seed 1111 [1/5] -----
```

图 6-14: MTFN 训练配置

训练结果:

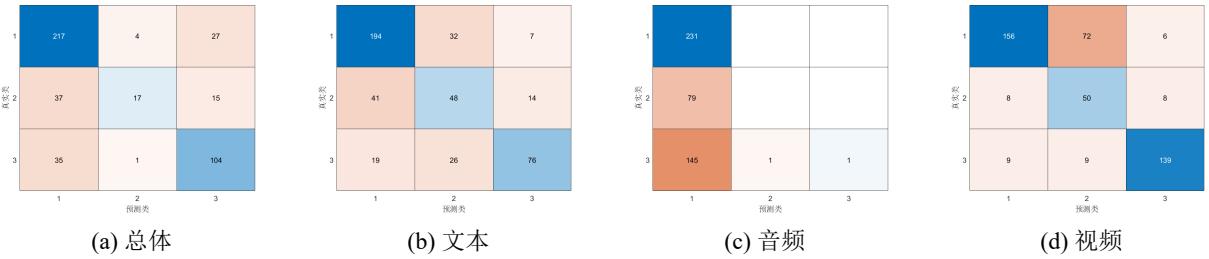


图 6-15: MTFN 多任务分类的结果

我们的模型是先将特征融合在一起，然后再进行多分类，如下图所示，A-6是没融合之前的特征经过 tsne 降维的结果，6-17是特征融合后经过 tsne 降维的结果，可以发现 MLMF 特征融合后效果比 MTFN 好，文本，音频，视频之间的耦合性减少。

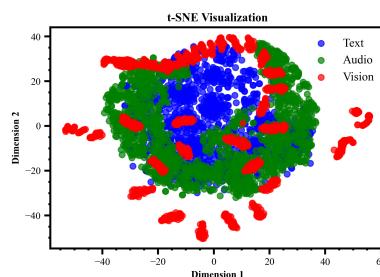


图 6-16: 特征融合前的结果

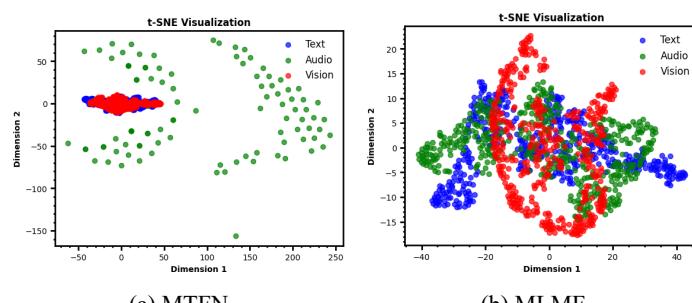


图 6-17: 特征融合后的结果

## 6.4 结果

我们将模型在测试集上测试，得到准确率，平均召回率和平均 F1 分数，如下表所示。

表 6-1: 测试集结果

类别	文本			音频			视频			总体		
	模型\指标	acc	recall	F1	acc	recall	F1	acc	recall	F1	acc	recall
梯度提升树	49.23	37.74	35.97	49.01	37.10	35.47	63.23	49.30	49.40	56.23	36.93	48.25
随机森林	43.76	37.03	36.64	50.77	34.22	38.90	62.14	46.01	44.52	55.58	35.52	28.79
adaboost	47.92	35.23	31.78	49.89	34.54	42.86	58.64	47.75	48.18	56.02	36.07	45.05
bert	78.77	76.50	76.46									
bert+cnn	59.73	50.19	46.52									
yolov8							75.71	70.23	72.71	74.40	64.83	66.71
cam++				67.18	68.31	65.54						
tim-net				82.49	81.62	80.38						
MTFN	69.58	64.22	65.11	50.77	33.56	34.34	75.49	76.99	72.36	73.96	62.14	63.64
MLMF	72.65	69.40	69.38	78.56	71.20	71.79	73.30	62.37	66.70	74.41	62.79	64.55

可知在文本方面 bert 模型的准确率最高，达到了 78.77%，在音频方面中 tim-net 模型的准确率最高，达到了 82.49%，在视频方面中 yolov8 模型准确率最高，达到了 75.71%，但是在总体分类的情况下。MLMF 的准确率最高，达到了 74.41%

## 第七章 总结

### 7.1 苏升榕

从开始到结束，一共两周的时间，在我看来这不仅仅是一个作业，更是一次对未知领域的探索。在此次学习中，我深感自己的不足，但同时也收获颇丰。记得刚开始时，面对海量的数据和复杂的算法，我感到迷茫和困惑。但随着一步步深入研究，我逐渐领略到了数据背后的奥秘和算法的神奇。

在处理数据的过程中，我学到了数据清洗、特征提取等关键技能。每当我以为自己已经掌握了所有知识点时，总会遇到新的问题和挑战，促使我不断地去查阅资料、尝试新的方法。正是这种不断探索的精神，让我在实践中得到了成长。而在算法的选择和调优过程中，我更是体会到了机器学习的魅力。原本我以为，只要选定了算法，就可以轻松完成任务。但实际上，算法的参数、数据的预处理等每一个细节都可能影响到最终的结果。通过反复的实验和调整，我逐渐摸索出了其中的门道，也更加深入地理解了机器学习的核心思想。

这次大作业，不仅仅是对我技能的一次锤炼，更是对我意志的一次考验。在遇到困难和挫折时，我告诉自己不能轻言放弃。正是这种坚持和毅力，让我最终完成了这次任务。

回首这次经历，我深知自己还有许多不足之处。但同时，我也为自己和团队所取得的进步感到骄傲。我相信，在未来的学习和工作中，我会更加努力地探索和实践，不断地突破自我、追求卓越。

感谢这次大作业给我带来的机会和挑战，也感谢老师和同组队友的帮助与支持。

### 7.2 黄艺豪

我们所做的多模态的情感分析，相比于单模态的情感分析，充分的利用了人类所表达的情感信息。我所做的工作是数据的处理，从视频中分离图像，文本，音频，通过专业的工具包进行数据特征的提取，以及通过填充分割的方式来使得每个样本特征数量的一致，通过 t-SNE 降维可视化，在二维空间里观察数据的分布特性，最后再通过聚类的方法来观察数据的特性。通过这一系列的工作使我更加的了解了数据的特性，做好机器学习的前提是掌握好初始数据的特性，所以我觉得我的工作很重要，并且在我做的时候，我也是在边做边学，通过这次的作业

使我学到了很多。

### 7.3 朱壮

多模态机器学习是当前人工智能领域研究的热点，它不再局限于单一的文本或图像数据，而是将多种媒体信息整合起来，以提供更全面、深入的理解。在本次任务中，我主要面临如何有效地融合来自不同模态的数据的挑战。为了解决这个问题，我采用了 MTFN 和 MLMF 两种模型。

MTFN 模型的张量笛卡尔积融合策略能很好地模拟多模态特征之间的关系，它通过张量乘积来融合不同模态的特征，从而更全面地理解任务。而 MLMF 模型的低秩融合策略则大大降低了参数的数量，减少了过拟合的风险。这种策略通过将不同模态的特征矩阵进行低秩分解，然后将其融合在一起，从而实现了更有效的特征表示。

通过这两种模型，我成功地实现了多模态数据的融合，并在本次视频情绪分析任务中取得不错的结果。这让我对多模态机器学习有了更深入的理解，也让我学会了如何处理复杂的多模态数据。这次任务的经验和收获，对于我未来在多模态机器学习领域的学习具有重要意义。

### 7.4 马永辉

多模态学习是一种机器学习策略，它使用来自多种不同源（或“模态”）的信息进行学习。这些源可以包括文本、图像、音频、视频等。多模态学习的目标是利用不同模态之间的互补信息来提高学习性能。

例如，考虑一个视频分类任务，我们可以从视频的图像、音频和文本（如字幕）中获取信息。这些不同的模态可以提供不同的、互补的信息，例如，图像可以提供视觉信息，音频可以提供声音信息，文本可以提供语义信息。通过结合这些信息，我们可以更准确地分类视频。

多模态学习的主要挑战是如何有效地结合不同模态的信息。一种常见的策略是使用融合（fusion）技术，例如早期融合（early fusion）、晚期融合（late fusion）和中间融合（intermediate fusion）。

**早期融合：**在特征级别结合不同模态的信息，然后使用一个共享的模型进行学习。例如，我们可以将图像特征和文本特征拼接在一起，然后输入到一个神经网络中。

**晚期融合：**对每个模态独立地进行学习，然后在决策级别结合这些模态的信息。例如，我们可以独立地训练一个图像分类器和一个文本分类器，然后将这两

个分类器的预测结果结合起来。

**中间融合：**在模型的中间层结合不同模态的信息。例如，我们可以使用一个卷积神经网络处理图像，使用一个循环神经网络处理文本，然后将这两个网络的隐藏状态结合起来。

在本次作业中，我主要负责论文选题的确定，数据集的查找，深度学习和多模态学习代码书写，论文的整合以及书写。在这次的作业中，我们的模型有如下优点：

1. 具有很强的应用性。利用不同模态的信息进行学习，这是近些年来的热点，例如 OpenAI 的 Chatgpt，谷歌的 Gemini 等，学习不同之间的数据表达方式以及相互之间的融合联系是势在必行的。
2. 数据的多样性；我们的模型能够处理文本，视频和音频这样的数据，已经是一种进步了。
3. 利用集成学习的思想；最后的决策利用集成学习的思想对最终结果进行投票，实现了后期的融合。
4. 模型的丰富性；我们采用的模型都是在各自对应的领域中的佼佼者，具有很强的预测能力。

在这次的作业中，我深刻地体会到了数据的多样性以及融合的困难。主要有以下几点：

1. 数据的不足；虽然我们有两千多个视频总共 9GB 的数据，但是还是远远不够，由于计算资源的限制，导致我们的模型虽然很厉害，但是无法发挥全部实力；再者，数据的标注也值得改进，比如可以将视频与音频对齐，使得两者数据在时间上产生联系，进而更好地融合特征，进行分类；最后，数据的特征的选择也值得思考，对于人的情绪我想应该有对应的特征去判别，而不是按照我们常规的方法去选择，这点值得我们思考。

2. 模型的选择；模型的结构的选择最为关键，如果能有一个统一的模型框架，能够同时处理文本音频视频等不同类型的数据，并将其融合在一起进行情绪识别就好了。

在这次的作业中，我学习到了：

1. transformer 对于自然语言处理的运算机制以及注意力机制的优点。
2. 在处理序列数据中不同时间点的特征之间的关系尤为重要。
3. 在计算机视觉中的任务有分割，分类，检测等，yolov8 模型的优点在于计算快！

4. 认识了 bert 模型，知道了预训练与微调的关键
5. 学习了 pytorch 的基本代码，以及深度学习的总体过程。

我认为我们改进的点主要有：

1. 数据集还有 v2.0 版本的，数据量更大。
2. 改进模型结构，能够把 YOLOV8 和 bert 等模型联系在一起，比如在各自模型处理数据时能够交换各自的数据信息，从而更好地为后面的判别做好准备。
3. 项目文件的书写，代码的规范性，整洁性与易读性，可维护性值得我们思考。

## 参考文献

- [1] <https://jalammar.github.io/illustrated-transformer/>
- [2] <https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1>
- [3] <https://github.com/ultralytics/ultralytics>
- [4] <https://github.com/thuiai/MMSA>
- [5] <https://github.com/thuiai/MMSA-FET>
- [6] Wenmeng Yu, Hua Xu.CH-SIMS: A Chinese Multimodal Sentiment Analysis Dataset with Fine-grained Annotations of Modality[J].Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics,2020,(58):3718–3727.
- [7] Amir Zadeh and Minghai Chen and Soujanya Poria and Erik Cambria and Louis-Philippe Morency.Tensor Fusion Network for Multimodal Sentiment Analysis[J].arXiv,2017,(1707.07250).
- [8] Zhun Liu and Ying Shen and Varun Bharadhwaj Lakshminarasimhan and Paul Pu Liang and Amir Zadeh and Louis-Philippe Morency.Efficient Low-rank Multimodal Fusion with Modality-Specific Factors[J].arXiv,2018,(1806.00064).
- [9] Ye, Jiaxin and Wen, Xin-Cheng and Wei, Yujie and Xu, Yong and Liu, Kunhong and Shan, Hongming.Temporal Modeling Matters: A Novel Temporal Emotional Modeling Approach for Speech Emotion Recognition[J].ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),2023,(10.1109).
- [10] Hui Wang and Siqi Zheng and Yafeng Chen and Luyao Cheng and Qian Chen.CAM++: A Fast and Efficient Network for Speaker Verification Using Context-Aware Masking[J].arXiv,2023,(2303.00332).
- [11] Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina.BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[J].Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers),2019,(4171–4186).

# 附录

## A.1 MMSA

---

```
def MMSA_run(
    model_name: str, dataset_name: str, config_file: str = "",
    config: dict = None, seeds: list = [], is_tune: bool = False,
    tune_times: int = 50, feature_T: str = "", feature_A: str = "",
    feature_V: str = "", model_save_dir: str = "", res_save_dir: str = "",
    log_dir: str = "", gpu_ids: list = [0], num_workers: int = 4,
    verbose_level: int = 1
)
```

---

参数如图所示：

参数：

- `model_name` (必填)：MSA 型号的名称，有关详细信息，请参阅[支持的型号](#)。
- `dataset_name` (必填)：MSA 数据集的名称，有关详细信息，请参阅[支持的数据集](#)。
- `config_file`：配置文件的路径。如果未指定，将使用默认配置文件。有关详细信息，请参阅[配置文件](#)。
- `config`：Python dict 格式的配置，用于覆盖 `config_file` 在调谐模式下被忽略。
- `seeds`：种子列表。默认：[1111, 1112, 1113, 1114, 1115]
- `is_tune`：调谐模式开关。详细信息请参见[调谐模式](#)。默认：False
- `tune_times`：# 要调整的超参数集。默认：50
- `feature_T`：文本特征文件的路径。提供一个空字符串以使用默认的 BERT 功能。默认：""
- `feature_A`：音频特征文件的路径。提供一个空字符串以使用数据集创建者提供的默认功能。默认：""
- `feature_V`：视频特征文件的路径。提供一个空字符串以使用数据集创建者提供的默认功能。默认：""
- `model_save_dir`：保存训练模型的路径。默认：~/MMSA/saved\_models
- `res_save_dir`：csv结果保存路径。默认：~/MMSA/results
- `log_dir`：日志文件保存路径。默认：~/MMSA/logs
- `gpu_ids`：要使用的 GPU。如果提供空列表，将分配最空闲的 GPU。默认：[0]。目前仅支持单GPU。
- `num_workers`：用于加载数据的工作人员数量。默认：4
- `verbose_level`：标准输出的详细级别。0 用于错误、1 信息、2 调试。默认：1

图 A-1: MMSA 的 API

用法实例：

---

```
from MMSA import MMSA_run

# run lmf on mosi with default params
MMSA_run('lmf', 'mosi')

# tune mult on mosei with default param ranges
MMSA_run('mult', 'mosei', is_tune=True, seeds=[1111])
```

---

配置使用:

---

```
from MMSA import MMSA_run, get_config_regression

# get default config of mult on sims
config = get_config_regression('mult', 'sims')
# alter the default config
config['nlevels'] = 4
config['conv1d_kernel_size_l'] = 3
config['conv1d_kernel_size_a'] = 3
config['conv1d_kernel_size_v'] = 3
# check config
print(config)
# run with altered config
MMSA_run('mult', 'sims', config=config)
```

---

## A.2 MMSA FFT

该 getdefaultconfig 函数返回 python 字典中的配置。它只需要一个参数，即提取方法的名称。支持的方法有:’bert’,’glove’,’librosa’,’mediapipe’,’openface’,’opensmile’,’roberta’,’wa示例代码:

---

```
from MSA_FET import get_default_config, FeatureExtractionTool

# Get default config for OpenFace & alter
config_v = get_default_config('openface')
# Enable Active Speaker Detection
config_v['video']['multiFace']['enable'] = True
# Get default config for openSMILE & alter
config_a = get_default_config('opensmile')
# Use LLD features
config_a['audio']['args']['feature_level'] = 'LowLevelDescriptors'
# Get default config for bert & alter
config_t = get_default_config('bert')
# Switch to Chinese
config_t['text']['pretrained'] = 'bert-base-chinese'
# Combine the three modalities
config = {**config_a, **config_v, **config_t}
# Initialize main class
fet = FeatureExtractionTool(config)
```

---

支持的工具和功能:

4.1 音频工具		
• 图书馆 ( <a href="#">链接</a> )	支持这里列出的所有librosa功能，包括： <a href="#">mfcc</a> , <a href="#">rms</a> , <a href="#">zero_crossing_rate</a> , <a href="#">spectrum_rolloff</a> , <a href="#">spectrum_centroid</a> 等。 <a href="#">详细配置可以在<a href="#">这里</a>找到。</a>	
• openSMILE ( <a href="#">链接</a> )	<a href="#">支持这里</a> 列出的所有功能集，包括： <a href="#">ComParE_2016</a> , <a href="#">GeMAPS</a> , <a href="#">eGeMAPS</a> , <a href="#">emobase</a> 等。 <a href="#">详细配置可以在<a href="#">这里</a>找到。</a>	
• Wav2vec2 ( <a href="#">链接</a> )	由拥抱面部变音器集成。 <a href="#">详细配置可以在<a href="#">这里</a>找到。</a>	
4.2 视频工具		
• OpenFace ( <a href="#">链接</a> )	支持 OpenFace 的 FeatureExtraction 二进制文件中的所有功能，包括：2D 和 3D 中的面部标志、头部姿势、注视点、面部动作单元、HOG 二进制文件。这些功能的详细信息可以在此处和此处的 OpenFace Wiki 中找到。 <a href="#">详细配置可以在<a href="#">这里</a>找到。</a>	
• 摄像管道 ( <a href="#">链接</a> )	支持面部网格和整体（面部、手部、姿势）解决方案。 <a href="#">详细配置可以在<a href="#">这里</a>找到。</a>	
• 谈话网 ( <a href="#">链接</a> )	TalkNet 用于支持主动说话人检测，以防视频中存在多个人脸。	
4.3 文本工具		
• BERT ( <a href="#">链接</a> )	由拥抱面部变音器集成。 <a href="#">详细配置可以在<a href="#">这里</a>找到。</a>	
• XUNet ( <a href="#">链接</a> )	由拥抱面部变音器集成。 <a href="#">详细配置可以在<a href="#">这里</a>找到。</a>	
4.4 对准器		
• Wav2vec CTC 对准器	使用预训练的 Wav2vec ASR 模型为每个单词生成时间戳，然后将视频和音频与文本对齐，目前仅支持英语。	

图 A-2: 支持的工具和功能

### A.3 YOLOv8

YOLOv8 训练参数如下图所示：

键	值	描述
model	None	模型文件路径，例如 <code>yolov8n.pt</code> , <code>yolov8n.yaml</code>
data	None	数据文件路径，例如 <code>coco128.yaml</code>
epochs	100	训练的轮次数量
patience	50	早停训练的等待轮次
batch	16	每批图像数量 (-1 为自动批大小)
imgsz	640	输入图像的大小，以整数表示
save	True	保存训练检查点和预测结果
save_period	-1	每 1 小时保存检查点 (如果<1 则禁用)
cache	False	True/ram/disk 或 False，使用缓存加载数据
device	None	运行设备，例如 <code>cuda device=0</code> 或 <code>device=0,1,2,3</code> 或 <code>device=cpu</code>
workers	8	数据加载的工作线程数 (如果 DDP 则为每个 RANK)
project	None	项目名称
name	None	实验名称
exist_ok	False	是否要重置现有实验
pretrained	True	(bool 或 str) 是否使用预训练模型 (bool) 或从中加载权重的模型 (str)
optimizer	'auto'	使用的优化器，选项范围：[SGD, Adam, Adamax, AdamW, NAdam, RMSProp, auto]
verbose	False	是否打印详细输出
seed	0	随机种子，用于可重复性
deterministic	True	是否启用确定性模式
single_cls	False	将多类数据转化为单类训练
rect	False	矩形训练，每批为最小填充整合
cos_lr	False	使用余弦学习率调度器
close_mosaic	10	(int) 最后 10 次迭代用马赛克增强 (0 为禁用)
resume	False	从最后检查点恢复训练
amp	True	自动混合精度 (AMP) 训练，选择范围：[True, False]
fraction	1.0	训练的数据集比例 (默认为 1.0，即训练集中的所有图像)
profile	False	在训练期间为记录器分析ONNX和TensorRT速度
freeze	None	(int 或 list, 可选) 在训练期间冻结前 n 层，或冻结层索引列表
lr0	0.01	初始学习率 (例如 SGD=1E-2, Adam=1E-3)
lrf	0.01	最终学习率 (lr0 * lrf)
momentum	0.937	SGD 动量/Adam beta1
weight_decay	0.0005	优化器权重衰减 5e-4
warmup_epochs	3.0	热身轮次 (小数 ok)

图 A-3: YOLOv8 训练参数

示例代码：

---

```

from ultralytics import YOLO

# 从头开始创建一个新的YOLO模型
model = YOLO('yolov8n.yaml')

# 加载预训练的YOLO模型（推荐用于训练）
model = YOLO('yolov8n.pt')

# 使用“coco128.yaml”数据集训练模型3个周期
results = model.train(data='coco128.yaml', epochs=3)

# 评估模型在验证集上的性能
results = model.val()

# 使用模型对图片进行目标检测
results = model('https://ultralytics.com/images/bus.jpg')

# 将模型导出为ONNX格式
success = model.export(format='onnx')

```

---

## A.4 bert

注意力可视化：

---

```

from bertviz import head_view, model_view
from transformers import BertTokenizer, BertModel
import imageio
model_version = 'D:\PycharmProjects\Multimodal_emotion\model\\bert-base-chinese'
model = BertModel.from_pretrained(model_version, output_attentions=True)
tokenizer = BertTokenizer.from_pretrained(model_version)
sentence_a = "好不好？燕子，你要开心，你要幸福，好不好？"
sentence_b = "开心啊，幸福。你的世界以后没有我了，没关系，你要自己幸福。"
inputs = tokenizer.encode_plus(sentence_a, sentence_b, return_tensors='pt')
print(inputs)
input_ids = inputs['input_ids']
print(input_ids)
token_type_ids = inputs['token_type_ids']
print(token_type_ids)
attention = model(input_ids, token_type_ids=token_type_ids)[-1]
print(attention)
sentence_b_start = token_type_ids[0].tolist().index(1)
print(sentence_b_start)

```

---

```
input_id_list = input_ids[0].tolist() # Batch index 0
print(input_id_list)
tokens = tokenizer.convert_ids_to_tokens(input_id_list)
print(tokens)

head_view(attention, tokens, sentence_b_start)
```

---

多头注意力可视化:

---

```
model_view(attention, tokens, sentence_b_start)
```

---

注意力计算过程可视化:

---

```
from bertviz.transformers_neuron_view import BertModel, BertTokenizer
from bertviz.neuron_view import show

model_type = 'bert'
model_version = 'D:\PycharmProjects\Multimodal emotion\model\\bert-base-chinese'
model = BertModel.from_pretrained(model_version, output_attentions=True)
tokenizer = BertTokenizer.from_pretrained(model_version, do_lower_case=True)
show(model, model_type, tokenizer, sentence_a, sentence_b, layer=4, head=3)
```

---

## A.5 投票集成

---

```
clc;clear;
%%多模态学习
A=importdata("predictions.csv");
data=A.data;
bert_cnn=data(:,1);
bert=data(:,2);
yolov8=data(:,3);
cam=data(:,4);
tim=data(:,5);
sd{1}=bert;
sd{2}=tim;
sd{3}=yolov8;
true_M=data(:,6);
true_T=data(:,7);
true_V=data(:,8);
true_A=data(:,9);
%%深度学习
B=importdata("predictions_mutil.csv");
B=B.data;
```

```

t_ml=B(:,1);
v_ml=B(:,2);
m_ml=B(:,3);
a_ml=B(:,4);
m_mf=B(:,5);
a_mf=B(:,6);
t_mf=B(:,7);
v_mf=B(:,8);
sd1=ff(sd);
%%机器学习
A1=importdata('predictions.xlsx');
AA1=A1.data;
t{1}=AA1(:,1);
t{2}=AA1(:,2);
s{1}=AA1(:,3);
s{2}=AA1(:,4);
a{1}=AA1(:,5);
a{2}=AA1(:,6);
t{3}=AA1(:,7);
s{3}=AA1(:,8);
a{3}=AA1(:,9);
t1=ff(t);
a1=ff(a);
s1=ff(s);
%%多模态学习
zb(true_M+1,m_ml+1,'m_ml')
zb(true_T+1,t_ml+1,'t_ml')
zb(true_A+1,a_ml+1,'a_ml')
zb(true_V+1,v_ml+1,'v_ml')
zb(true_M+1,m_mf+1,'m_mf')
zb(true_T+1,t_mf+1,'t_mf')
zb(true_A+1,a_mf+1,'a_mf')
zb(true_V+1,v_mf+1,'v_mf')
%%深度学习
zb(true_M+1,sd1+1,'m_sd')
zb(true_T+1,bert+1,'t_bert')
zb(true_T+1,bert_cnn+1,'t_bert_cnn')
zb(true_V+1,yolov8+1,'yolov8')
zb(true_A+1,cam+1,'cam')
zb(true_A+1,tim+1,'tim')
%%机器学习
zb(true_M+1,t1+1,'m_td')
zb(true_T+1,t{1}+1,'t_td')

```

```

zb(true_A+1,t{2}+1,'a_td')
zb(true_V+1,t{3}+1,'v_td')
zb(true_M+1,s1+1,'m_sj')
zb(true_T+1,s{1}+1,'t_sj')
zb(true_A+1,s{2}+1,'a_sj')
zb(true_V+1,s{3}+1,'v_sj')
zb(true_M+1,a1+1,'m_ad')
zb(true_T+1,a{1}+1,'t_ad')
zb(true_A+1,a{2}+1,'a_ad')
zb(true_V+1,a{3}+1,'v_ad')

function t1=ff(t)
t1 = mode([t{1},t{2},t{3}], 2);
end

function a1=f(t,t1,t2,t3,true_A, true_V ,true_M ,true_T)
acc_t1=sum(true_M==t)/ length(true_T);
acc_tt=sum(true_T==t1)/ length(true_T);
acc_ta=sum(true_A==t2)/ length(true_T);
acc_tv=sum(true_V==t3)/ length(true_T);
a1=[acc_t1,acc_tt,acc_ta,acc_tv];
end

function zb(true_labels,predicted_labels,str)
% 计算混淆矩阵
C = confusionmat(true_labels, predicted_labels);

% 绘制混淆矩阵
confusionchart(C);
str2=['D:\桌面\大三上\模式识别\multi emotion\figures\' str '_cm.png'];
% 保存图片
saveas(gcf, str2);
% 计算总体准确率
accuracy = sum(diag(C)) / sum(C(:));

% 计算每个类别的召回率（也称为真正率或灵敏度）
recall = diag(C) ./ sum(C, 2);

% 计算每个类别的精确率（也称为阳性预测值）
precision = diag(C) ./ sum(C, 1)';

% 计算每个类别的F1分数
F1 = 2 * (precision .* recall) ./ (precision + recall);

% 删除数组中的NaN
F1(isnan(F1)) = [];

```

```
% 计算宏平均
macro_precision = mean(precision);
macro_recall = mean(recall);
macro_F1 = mean(F1);

% 打印结果
fprintf(str);
fprintf('\n');
fprintf('Accuracy: %f\n', accuracy);
fprintf('Macro Precision: %f\n', macro_precision);
fprintf('Macro Recall: %f\n', macro_recall);
fprintf('Macro F1 Score: %f\n', macro_F1);
end
```

## A.6 我的项目

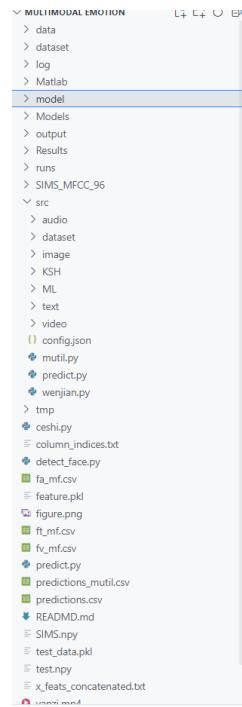


图 A-4: 我的项目

举个例子，如下图所示，岳云鹏说：“好不好？燕子，你要开心，你要幸福，好不好？开心啊，幸福。你的世界以后没有我了，没关系，你要自己幸福。”



图 A-5: 视频例子

对其进行预测得到结果：

```
PS D:\PycharmProjects\Multimodal emotion> & D:/py/python3.11.0/python.exe "d:/PycharmProjects/Multimodal emotion/src/predict.py"
2023-12-24 18:30:01.481994: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
WARNING:tensorflow:From D:\py\python3.11.0\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

{'model_name': 'mlmf', 'dataset_name': 'sims', 'featurePath': '/home/sharing/disk3/Datasets/WMSA-Standard\SIMS/Processed/unaligned_39.pkl', 'seq_lens': [39, 400, 55], 'feature_dims': [768, 33, 709], 'train_samples': 1368, 'num_classes': 3, 'language': 'cn', 'keyEval': 'Loss', 'missing_rate': [0.2, 0.2, 0.2], 'missing_seed': [1111, 1111, 1111], 'need_data_aligned': False, 'need_model_aligned': False, 'need_normalized': True, 'early_stop': 8, 'hidden_dims': [256, 32, 256], 'post_text_dim': 8, 'post_audio_dim': 4, 'post_video_dim': 32, 'post_dropouts': [0.4, 0.4, 0.4, 0.4], 'dropouts': [0.3, 0.3, 0.3], 'rank': 3, 'batch_size': 16, 'learning_rate': 0.001, 'factor_lr': 0.001, 'M': 0.2, 'T': 1.0, 'A': 0.4, 'V': 0.8, 'text_weight_decay': 0.0001, 'audio_weight_decay': 0.0, 'video_weight_decay': 0.0001, 'weight_decay': 0.005, 'train_mode': 'classification'}
[768, 33, 709]
[39, 400, 55]
torch.Size([39, 768]) torch.Size([400, 33]) torch.Size([55, 709])
总体情感倾向: negative
文本情感倾向: positive
音频情感倾向: negative
视频情感倾向: negative
```

图 A-6: 预测结果

可知总体情感倾向是消极的，视频和音频是也是消极，但是文本是积极的。

## 致 谢

金陵十二月，冬意渐浓，天寒地冻，霜气凛冽。我深知岁月不待人，自今年十月在此投身此书，转眼间已是十二月，犹如白驹过隙。我深感时光荏苒，韶华易逝，不禁感叹人生苦短。如今我已近二十之年，却仍未能在事业和家庭上有所建树，心中不禁惶恐不安。在这闭卷之际，我回首过去的岁月，心中既有欢喜也有悲凉；我依窗而坐，怀抱着满腔的思绪。虽然我自知并非天资聪颖，但凭借着一股坚韧不拔的意志和些许微薄的力量，终于小有所成。我深知，身体发肤受之父母，是他们赐予我生命；而道德学问则受之师长，是他们引领我成长。对于他们的养育之恩、教诲之情和切磋之谊，我虽欲结草衔环以报，却发现无以为报。在此，我借着这寥寥数语，向他们表达我最深的感激之情！