# 神经网络与深度学习课程大作业

**题目：NLP 中情感分析问题研究（以 IMDB 数据集为例）**

学号：6243112053　专业：人工智能专硕　姓名：王玺

# 一、引言

情感分析（Sentiment Analysis, SA）是自然语言处理（NLP）领域的一个重要任务，旨在识别和提取文本中的情感信息，通常涉及到分析文本中的情感极性（如积极、消极或中立）或情感强度（如高兴、愤怒、悲伤等）。情感分析的目标是通过计算机自动化理解和分类人类情感，并为用户提供智能化的服务。在本次大作业中我使用了 IMDB 评论这一经典情感分析数据集，我分别使用了传统的贝叶斯分类和集成学习模型，LSTM 模型以及预训练模型 AlBert 分别进行实验，这些是实现情感分析的典型方法，能够锻炼自己的实际代码能力。

## 1.1 情感分析的定义

情感分析，亦称情感分类或意见挖掘，通常分为两大类任务：

- **情感极性分析**：即判断文本表达的是正面、负面还是中立的情感。常见的任务包括电影评论的正负面情感分类、社交媒体评论的情感倾向等。
- **情感类别分析**：将情感细分为多个类别，如愉悦、愤怒、悲伤等，或者更细化的情感（例如“开心”、“生气”等）。
- **情感强度分析**：对于情感的强度进行量化分析，评估情感的强烈程度，如从“轻微愤怒”到“极度愤怒”。

## 1.2 情感分析的背景和发展历程

情感分析的起源可以追溯到 20 世纪 90 年代，当时主要应用于商业领域，用来分析消费者对于产品或品牌的情感倾向。随着社交媒体和互联网的兴起，情感分析逐渐应用于更多的领域，包括舆情监控、政治分析、社会媒体分析等。近年来，深度学习和预训练语言模型（如 BERT、ALBERT 等）为情感分析带来了革命性的进展，使得情感分析从基于规则和机器学习的方法，发展到基于深度神经网络和大规模预训练模型的复杂任务。

## 1.3 情感分析的重要性

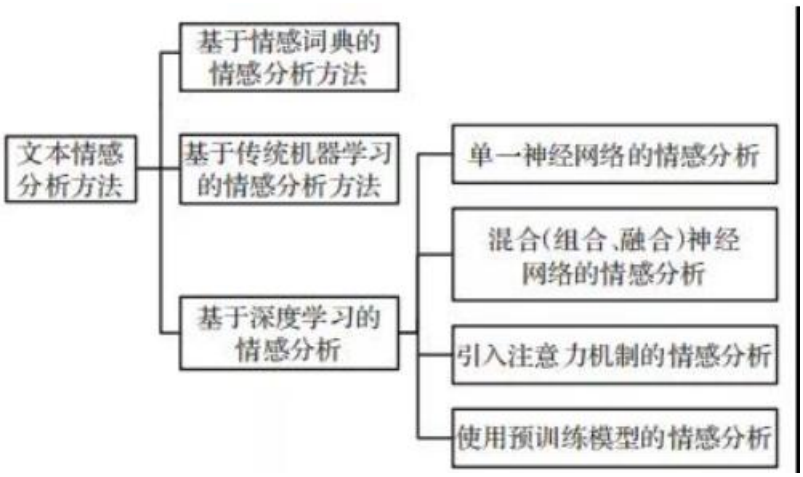
情感分析在多个领域具有广泛的应用意义，具体包括：

- **社交媒体分析**：分析社交平台（如 Twitter、Facebook 等）的用户评论、推文和帖子，帮助公司、政府和媒体了解公众情绪。例如，在政治选举中，通过分析社交媒体的情感倾向，可以预测选民的情绪和投票行为。
- **客户反馈分析**：通过分析用户对产品、服务或品牌的评论，了解用户满意度和潜在的改进空间。例如，企业可以利用情感分析工具来实时监控客户的情感动态，及时回应客户需求。
- **舆情监测**：情感分析可以帮助政府和媒体机构监控网络舆情，了解公众对事件、政策或新闻的情感态度，及时调整舆论导向。
- **市场调研与广告策略**：情感分析可以帮助品牌分析广告效果，了解消费者对广告、产品的情感反应，从而优化市场营销策略。

随着情感分析技术的进步，越来越多的行业依赖情感分析来优化决策、提升服务质量和增强客户体验。

## 1.4 情感分析的技术演进

情感分析的技术路径可以大致分为以下几个阶段：




- **基于词典的方法**：早期的情感分析方法主要依赖情感词典（如 SentiWordNet），通过词汇匹配来判断文本的情感倾向。这类方法简单高效，但容易受到词汇覆盖的限制。
- **基于机器学习的方法**：随着机器学习技术的发展，情感分析逐渐转向基于统计模型的解决方案。常见的机器学习方法包括支持向量机（SVM）、朴素贝叶斯（Naive Bayes）、决策树（Decision Tree）等。这些方法通过训练模型学习文本特征与情感标签之间的关系。

- **基于深度学习的方法**：近年来，深度学习（特别是卷积神经网络 CNN 和长短时记忆网络 LSTM）在情感分析任务中取得了显著的成果，能够自动学习文本中的复杂特征关系。特别是 Transformer 架构（如 BERT、GPT）为情感分析提供了更高效和更精确的建模能力。预训练语言模型能够通过对大规模文本数据的预训练，获得强大的上下文理解能力，极大提升了情感分析的准确度和泛化能力。

# 二、IMDB 数据集

IMDB 数据集（Internet Movie Database）是自然语言处理（NLP）中常用于情感分析任务的标准数据集之一。它包含了大量的电影评论数据，标签为正面（positive）和负面（negative）。这个数据集广泛用于文本分类、情感分析、情感分类以及其他与文本情感相关的研究任务。其数据内容和结构使得它成为情感分析领域的基准数据集。
IMDB 数据集官网

## 2.1 数据集来源

IMDB 数据集来源于 Internet Movie Database（IMDB），这是一个包含电影、电视节目、演员、导演、电影评分等相关信息的大型数据库。IMDB 数据集的电影评论部分包含了数以万计的用户评价和电影评论，用户可以为电影打分并留下评论。IMDB 数据集特别适用于情感分析任务，因为其评论的情感是明确的（正面或负面），便于分类模型训练和评估。

## 2.2 数据集内容与结构

IMDB 数据集主要包括以下几个部分：

电影评论文本（Text）：每个评论为一条文本记录，通常为用户对某一电影的评价。评论内容一般比较自然、口语化，涵盖了对电影情节、表演、导演等方面的评价。
标签（Label）：每条评论都有一个标签，表示评论的情感极性。通常，标签是二分类的：
0：正面评论（positive）
1：负面评论（negative）
对于情感分析任务而言，这些标签作为训练和评估模型的标准答案。

IMDB 数据集包含了大量的电影评论数据。标准的 IMDB 数据集通常分为训练集和测试集，具体包括：
训练集：大约 25,000 条评论，其中约一半为正面评论，另一半为负面评论。
测试集：大约 25,000 条评论，测试集同样包含平衡的正面和负面评论。
数据集的规模适中，能够有效用于训练机器学习或深度学习模型，尤其是在文本分类和情感分析领域。该数据集的文本长度和复杂度也使得它成为检验各种 NLP 模型表现的理想基准。

## 2.3 数据预处理相关操作代码

数据预处理的主要目标是从原始文本数据中提取有效信息，并将其转换为适合模型处理的格式。在该代码中，预处理的主要步骤是读取文件并根据文件夹名称标注情感标签。
该数据预处理代码假设评论文件存储在 pos 和 neg 子文件夹中，并且文件夹结构与 IMDB 数据集的标准结构一致。

```python
import os
import glob

# 数据预处理函数，读取数据并分词
def preprocess_data(data_dir):
    """
    该函数用于加载IMDB数据集中的电影评论数据，并对文本进行分词及标签化。

    参数：
    - data_dir: 数据集文件夹路径

    返回：
    - texts: 评论文本的列表
    - labels: 评论标签的列表，正面评论为0，负面评论为1
    """
    texts = []  # 存储评论文本
    labels = []  # 存储评论标签（0: 正面，1: 负面）

    # 遍历"pos"（正面评论）和"neg"（负面评论）文件夹
    for label in ["pos", "neg"]:
        # 获取当前标签下的所有文本文件路径
        file_paths = glob.glob(os.path.join(data_dir, label, "*.txt"))

        # 遍历每一个评论文件
        for file_path in file_paths:
            # 打开并读取评论文件内容
            with open(file_path, 'r', encoding='utf-8') as f:
                text = f.read().strip()  # 去掉文本两端的空白字符
                texts.append(text)  # 将评论文本添加到 texts 列表
                # 根据标签添加相应的标签（0: 正面，1: 负面）
                labels.append(0 if label == "pos" else 1)

    return texts, labels  # 返回文本和标签列表

# 示例：加载训练集和测试集数据
train_data_dir = "./aclImdb/train"  # 训练集路径
test_data_dir = "./aclImdb/test"    # 测试集路径

# 调用预处理函数加载数据
train_texts, train_labels = preprocess_data(train_data_dir)  # 训练集文本和标签
test_texts, test_labels = preprocess_data(test_data_dir)     # 测试集文本和标签
```

# 三、贝叶斯网络和集成学习方法

## 3.1 方法介绍

### 朴素贝叶斯模型

朴素贝叶斯模型（Naive Bayes Model）是一类基于贝叶斯定理（Bayes' Theorem）的分类模型，广泛用于分类任务，尤其是在文本分类（如垃圾邮件检测、情感分析）等应用中。它的核心思想是通过计算每个类别的条件概率来决定某个数据点属于哪个类别。

贝叶斯定理是基于概率理论的一种推理方法，它描述了如何通过已知信息（观测数据）来更新未知信息的概率。贝叶斯定理公式如下：

在训练过程中，朴素贝叶斯模型通过计算每个类别的先验概率 P(C) 和每个特征在每个类别下的条件概率 P(Xi | C) 来构建模型。训练过程的核心是通过已有的标注数据来估计这些概率。

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

- $P(C|X)$：给定输入特征 $X$ 后，属于某类别 $C$ 的后验概率。
- $P(X|C)$：在类别 $C$ 下，输入特征 $X$ 的似然度。
- $P(C)$：类别 $C$ 的先验概率，即在没有观察到 $X$ 时，类别 $C$ 出现的概率。
- $P(X)$：特征 $X$ 的边际概率。

在本次任务中，我使用了三种贝叶斯分类器：
MultinomialNB 适用于特征为多项式分布的情况（如词频），
BernoulliNB 适用于二元特征的情况（如单词是否出现），
ComplementNB 是对 MultinomialNB 的改进，适用于类别不平衡的情况。
通过将这些分类器与 VotingClassifier 集成在一起，你能够充分利用每种模型的优势，构建一个强大的文本分类器，提升预测性能和稳定性。

### VotingClassifier

VotingClassifier 是一种集成学习方法，它通过将多个不同的分类器（基学习器）结合在一起进行预测，旨在提高分类性能。它属于集成方法（Ensemble Methods），常用于通过组合多个模型的预测结果，减少单一模型的偏差和方差，从而提高整体的预测准确性。

在代码中，我使用了 VotingClassifier，其中将多个朴素贝叶斯模型（MultinomialNB, BernoulliNB, ComplementNB）和逻辑回归（Logistic Regression）作为基学习器进行组合。集成器采用软投票的方式，使用每个模型对每个样本的预测概率进行加权平均，从而得到最终的预测结果。

## 3.2 具体实现

使用 TfidfVectorizer 提取特征，并移除停用词，同时设置 n-grams 范围
数据平衡：使用 RandomUnderSampler 进行下采样

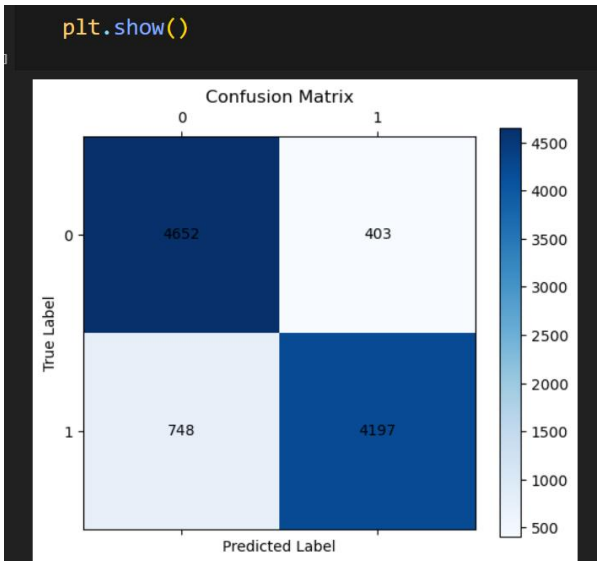
```
# 使用 TfidfVectorizer 提取特征，并移除停用词，同时设置 n-grams 范围
vectorizer = TfidfVectorizer(stop_words='english', ngram_range=(1, 3))
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# 数据平衡：使用 RandomUnderSampler 进行下采样
rus = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = rus.fit_resample(X_train_tfidf, y_train)
```

加载四种分类器，其中额外添加一个线性回归模型。VotingClassifier 将这四个不同类型的分类器进行集成，并使用软投票策略。它结合了每个模型的优点，使得最终的分类效果比任何一个单独模型的表现要好

```
# 定义多个分类模型并加入集成器
models = [
    ('MultinomialNB', MultinomialNB(alpha=0.1)),
    ('BernoulliNB', BernoulliNB(alpha=0.1)),
    ('ComplementNB', ComplementNB(alpha=0.1)),
    ('Logistic Regression', LogisticRegression(max_iter=1000)),
]

# 创建 VotingClassifier，使用软投票
ensemble_model = VotingClassifier(estimators=models, voting='soft')

# 训练集成模型
ensemble_model.fit(X_train_resampled, y_train_resampled)
```

训练代码后，测试绘制混淆矩阵

```
# 计算混淆矩阵
conf_matrix = confusion_matrix(y_test, y_pred)

# 绘制带数字的混淆矩阵
plt.figure(figsize=(6, 5))
plt.matshow(conf_matrix, cmap='Blues', fignum=1)
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')

# 在每个方格中显示对应的数值
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        plt.text(j, i, f'{conf_matrix[i, j]}', ha='center', va='center', color='black')

plt.show()
```




# 四、RNN 和 LSTM 模型

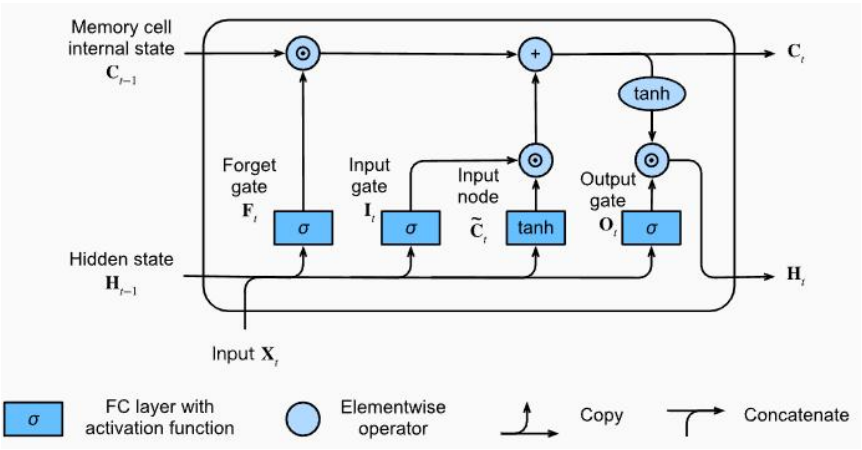
## 4.1 方法介绍

循环神经网络（Recurrent Neural Network, RNN）是一种能够处理序列数据的神经网络架构，擅长对时间序列数据或语言数据的上下文关系建模。其核心特点是具有循环连接，允许网络的隐藏状态在时间步之间传递信息，使得当前输出可以依赖于之前的输入。LSTM（Long Short-Term Memory）是一种特殊的 RNN 变体，专门为解决 RNN 的长时间依赖问题设计。LSTM 引入了 记忆单元（Cell） 和 门控机制（Gates），能够控制信息的流动，从而缓解梯度消失问题。LSTM 的主要组成部分：

遗忘门（Forget Gate）：决定当前时间步要丢弃多少过去的信息；

输入门（Input Gate）：决定当前时间步要保留哪些新的输入信息；

输出门（Output Gate）：决定从记忆单元中输出哪些信息作为隐藏状态。



LSTM 是处理情感分析任务的主流方法之一，其应用如下：

捕获长距离依赖：LSTM 能够记住句子中的关键情感词，即使它们距离句子的结尾较远；
处理复杂语义关系：LSTM 可以理解句子中前后语境对情感的影响（例如，"Although I was disappointed at first, the movie turned out to be amazing"）；
支持序列分类：LSTM 的输出隐藏状态可以被用于分类情感标签。

## 4.2 具体实现

设置 LSTM 模型的超参数（层数，嵌入层、隐藏层维度，总轮数，学习率，正则化系数，Dropout 系数等）

```python
# 超参数设置
embed_dim = 128  # 嵌入层维度
hidden_dim = 256  # 隐藏层维度
num_layers = 2  # LSTM 层数
num_classes = 2  # 输出类别数（正面或负面）
batch_size = 64  # 每个批次的样本数
num_epochs = 20  # 训练的总轮数
learning_rate = 0.0005  # 学习率
dropout_prob = 0.1  # Dropout 概率
weight_decay = 1e-5  # L2 正则化系数
```

定义 LSTM 模型

```python
# 定义 LSTM 模型
class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, num_layers, num_classes, dropout_prob):
        """
        :param vocab_size: 词汇表大小
        :param embed_dim: 嵌入层维度
        :param hidden_dim: LSTM 隐藏层维度
        :param num_layers: LSTM 层数
        :param num_classes: 输出类别数
        :param dropout_prob: Dropout 概率
        """
        super(LSTMClassifier, self).__init__()
        # 嵌入层，将词索引映射为嵌入向量
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        # LSTM 层，用于处理序列数据
        self.lstm = nn.LSTM(embed_dim, hidden_dim, num_layers, batch_first=True, dropout=dropout_prob)
        # Dropout 层，用于防止过拟合
        self.dropout = nn.Dropout(dropout_prob)
        # 全连接层，将 LSTM 的输出映射为类别分数
        self.fc = nn.Linear(hidden_dim, num_classes)

    def forward(self, text, lengths):
        """
        前向传播函数
        :param text: 文本的索引序列
        :param lengths: 文本序列的原始长度
        """
        # 嵌入层，将单词索引映射为嵌入向量
        embedded = self.embedding(text)
        # 使用 pack_padded_sequence 优化 LSTM 的计算效率
        packed_embedded = nn.utils.rnn.pack_padded_sequence(embedded, lengths, batch_first=True, enforce_sorted=False)
        # LSTM 前向传播
        packed_output, (hidden, cell) = self.lstm(packed_embedded)
        # 获取最后一层的隐藏状态
        hidden = self.dropout(hidden[-1])
        # 全连接层输出类别分数
        out = self.fc(hidden)
        return out
```

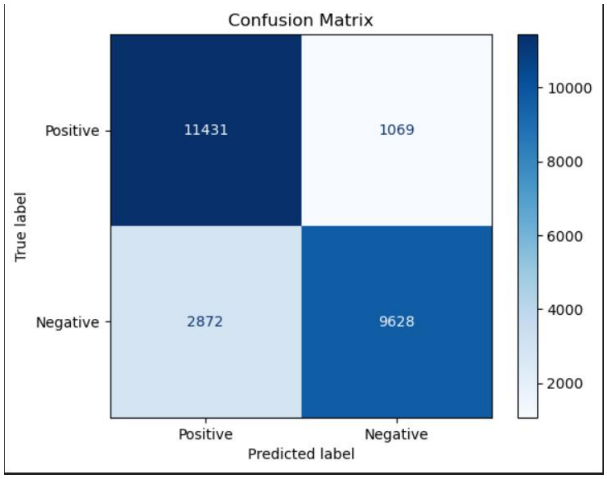
初始化模型定义，使用交叉熵损失和 Adam 优化器

```python
# 初始化模型
vocab_size = len(vocab) + 1  # 词汇表大小（包含填充索引 0）
model = LSTMClassifier(vocab_size, embed_dim, hidden_dim, num_layers, num_classes, dropout_prob).to(device)

# 定义损失函数和优化器
criterion = nn.CrossEntropyLoss()  # 使用交叉熵损失
optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=weight_decay)
```

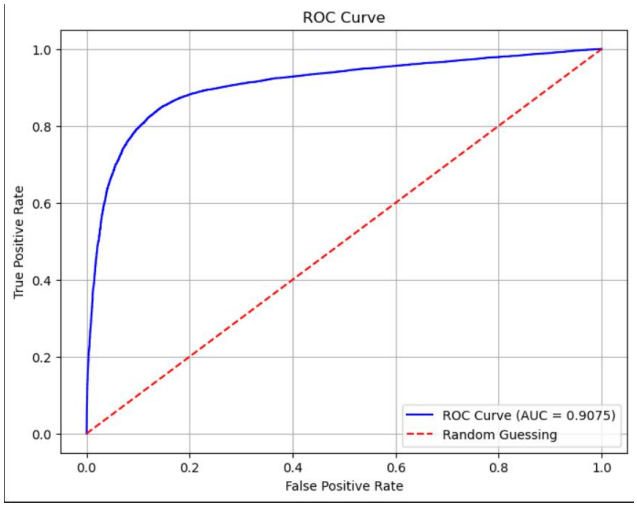
训练过程

```
Using device: cuda
Epoch 1/20, Train Loss: 0.6216, Train Accuracy: 0.6517,
Epoch 2/20, Train Loss: 0.5564, Train Accuracy: 0.7158,
Epoch 3/20, Train Loss: 0.5081, Train Accuracy: 0.7525,
Epoch 4/20, Train Loss: 0.4046, Train Accuracy: 0.8269,
Epoch 5/20, Train Loss: 0.3691, Train Accuracy: 0.8417,
Epoch 6/20, Train Loss: 0.5100, Train Accuracy: 0.7508,
Epoch 7/20, Train Loss: 0.3708, Train Accuracy: 0.8415,
Epoch 8/20, Train Loss: 0.2966, Train Accuracy: 0.8808,
Epoch 9/20, Train Loss: 0.2543, Train Accuracy: 0.9007,
Epoch 10/20, Train Loss: 0.2240, Train Accuracy: 0.9140,
Epoch 11/20, Train Loss: 0.2040, Train Accuracy: 0.9233,
Epoch 12/20, Train Loss: 0.1756, Train Accuracy: 0.9372,
Epoch 13/20, Train Loss: 0.1498, Train Accuracy: 0.9483,
Epoch 14/20, Train Loss: 0.1277, Train Accuracy: 0.9579,
Epoch 15/20, Train Loss: 0.1128, Train Accuracy: 0.9631,
Epoch 16/20, Train Loss: 0.0931, Train Accuracy: 0.9707,
```

测试结果的混淆矩阵




ROC 曲线



# 五、使用预训练模型（AlBert）

## 4.1 预训练模型

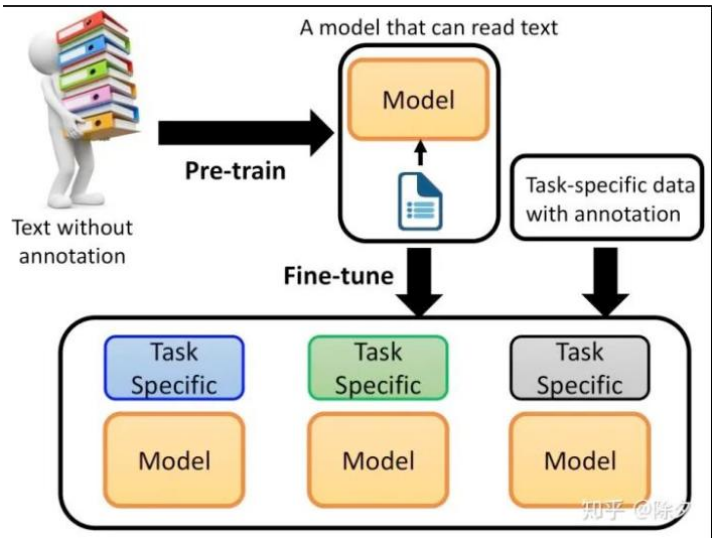
预训练模型（Pre-trained Model）是指在大规模数据集上经过初步训练的机器学习模型，这种训练通常使用通用任务，以学习广泛适用的特征或表示。在特定任务中，这些模型可以作为基础，通过微调（fine-tuning）或特征提取（feature extraction）来适应特定的任务需求。

预训练模型的核心理念是 迁移学习（Transfer Learning），即利用在一个领域中学到的知识（如语义、模式）来提高另一个领域的学习效率和效果。

## 什么是预训练模型？

预训练模型（Pre-trained Model）是指在大规模数据集上经过初步训练的机器学习模型，这种训练通常使用通用任务，以学习广泛适用的特征或表示。在特定任务中，这些模型可以作为基础，通过微调（fine-tuning）或特征提取（feature extraction）来适应特定的任务需求。

预训练模型的核心理念是 **迁移学习**（Transfer Learning），即利用在一个领域中学到的知识（如语义、模式）来提高另一个领域的学习效率和效果。




## 典型的预训练模型

## 自然语言处理（NLP）模型

BERT：双向语言模型，用于分类、命名实体识别、问答等任务。

GPT 系列：生成式语言模型，用于文本生成、对话、代码生成等。
ALBERT：轻量化 BERT，用于文本分类、情感分析等任务。
T5：统一文本生成模型，用于翻译、摘要、问答等。
XLNet：结合自回归与自编码，改进语言理解能力。

## 计算机视觉（CV）模型

ResNet：深层网络架构，用于图像分类、目标检测等。
VGG：深度卷积模型，用于分类和特征提取。
EfficientNet：高效图像分类模型，用于目标检测等任务。
ViT：基于 Transformer 的图像分类模型。
YOLO：实时目标检测模型，用于目标定位和分类。

## 多模态（图像+文本）模型

CLIP：联合图像与文本表示，用于多模态搜索。
DALL·E：文本生成图像，用于图像生成任务。
BLIP：多模态生成与理解，用于描述生成、图像搜索等。

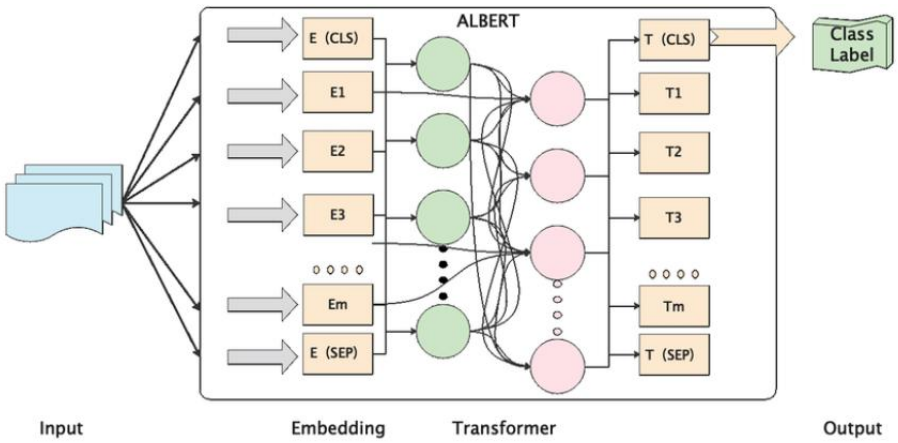
## 4.2 ALBERT

本次实验采取的预训练模型为 ALBERT，其在文本分类和情感分析中有着很好的效果。ALBERT 是由 Google 研究团队提出的一种轻量化、优化版本的 BERT，旨在降低预训练语言模型的参数规模和内存占用，同时保持或提升模型性能。
关键特性：

所有 Transformer 层共享相同的权重，从而大幅减少参数数量。例如，BERT-base 约 110M 参数，而 ALBERT-base 仅约 12M。

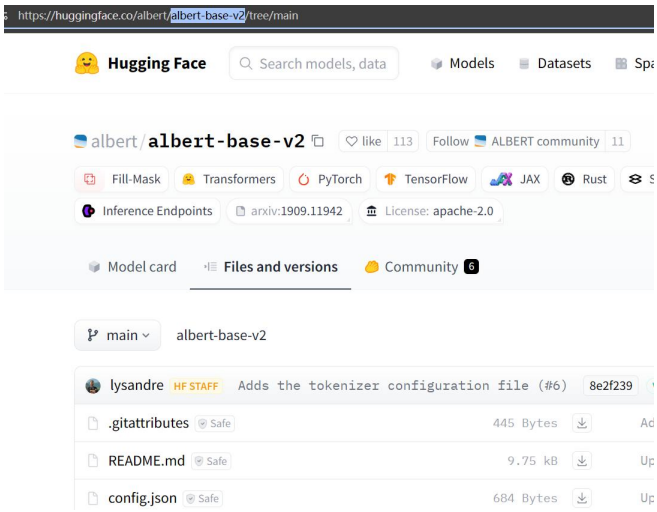
使用因式分解的方式，将大型词嵌入矩阵分解为两个小矩阵，从而降低嵌入层的参数规模。

引入 Sentence Order Prediction (SOP) 任务，改善模型在理解句子间逻辑关系上的能力。

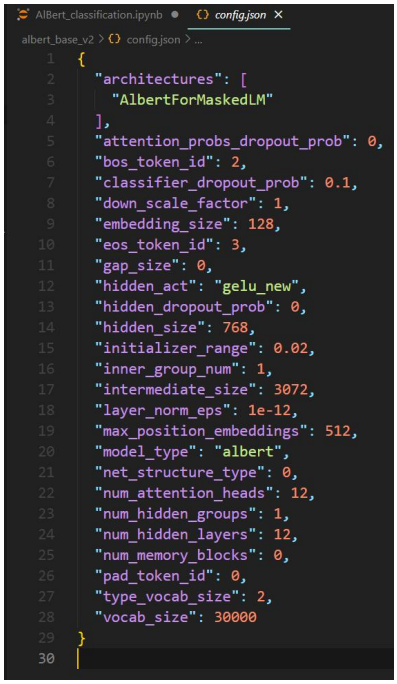


## 4.3 具体实现

在 HuggingFace 中下载 ALBERT 模型，本次实验使用 albert-base-v2。




下载的模型参数如下




加载预训练模型

```
# 加载本地 ALBERT tokenizer 和模型
model_dir = "./albert_base_v2/"
tokenizer = AlbertTokenizer.from_pretrained(model_dir)
model = AlbertForSequenceClassification.from_pretrained(model_dir, num_labels=2).to(device)
```

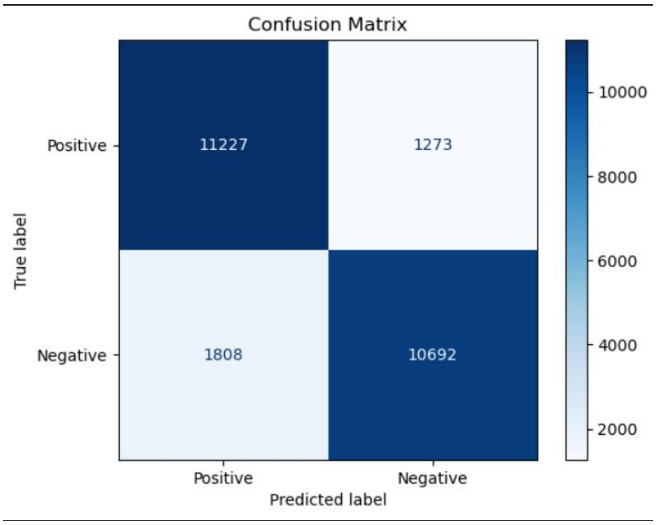
训练模型，记录过程

```python
# 训练模型
for epoch in range(num_epochs):
    model.train()
    total_loss, total_correct = 0, 0

    for batch in train_dataloader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].to(device)

        optimizer.zero_grad()
        outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        logits = outputs.logits

        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        total_correct += (logits.argmax(1) == labels).sum().item()

    train_losses.append(total_loss / len(train_dataloader))
    train_accuracies.append(total_correct / len(train_dataset))
```

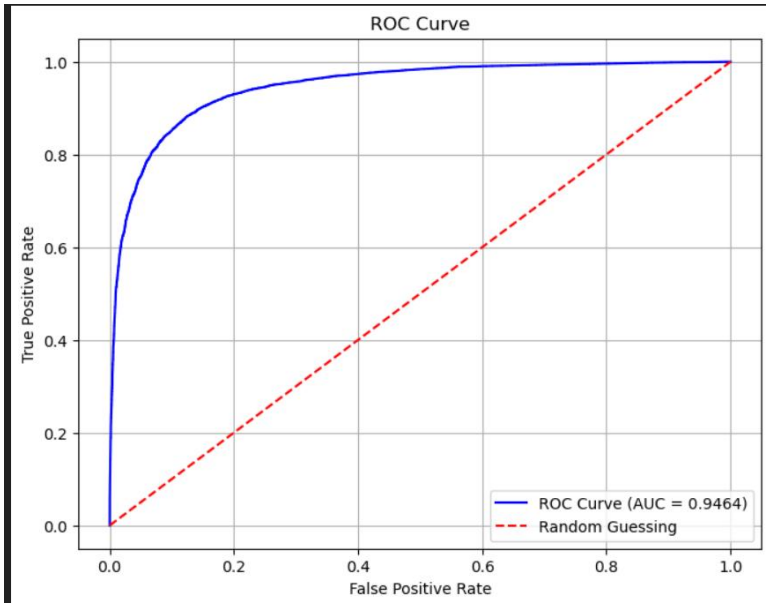
训练过程

```
✓ 98m 52.8s
c:\Users\89556\.conda\envs\DL\lib\site-packages\tq
  from .autonotebook import tqdm as notebook_tqdm
Using device: cuda
Some weights of AlbertForSequenceClassification we
You should probably TRAIN this model on a down-str
c:\Users\89556\.conda\envs\DL\lib\site-packages\tr
  attention_output = torch.nn.functional.scaled_dc
Epoch 1/10, Loss: 0.3476, Train Accuracy: 0.8427,
Epoch 2/10, Loss: 0.2369, Train Accuracy: 0.9024,
Epoch 3/10, Loss: 0.1650, Train Accuracy: 0.9348,
Epoch 4/10, Loss: 0.1106, Train Accuracy: 0.9611,
Epoch 5/10, Loss: 0.0753, Train Accuracy: 0.9746,
Epoch 6/10, Loss: 0.0565, Train Accuracy: 0.9806,
Epoch 7/10, Loss: 0.0425, Train Accuracy: 0.9860,
Epoch 8/10, Loss: 0.0337, Train Accuracy: 0.9886,
Epoch 9/10, Loss: 0.0328, Train Accuracy: 0.9882,
```

进行测试，得出混淆矩阵



ROC 曲线




控制台输出测试结果

```
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.90      0.88     12500
           1       0.89      0.86      0.87     12500

    accuracy                           0.88     25000
   macro avg       0.88      0.88      0.88     25000
weighted avg       0.88      0.88      0.88     25000
```

# 六、总结

情感分析（Sentiment Analysis）是自然语言处理（NLP）中的一种经典任务，旨在识别和提取文本中的情感信息，通常分为正面情感、负面情感和中性情感。在电影评论等文本中，情感分析可帮助识别观众对电影的评价，常用于情感分类、产品推荐、舆情分析等领域。在本实验中，我们使用了三种不同的方法（朴素贝叶斯、LSTM 和 ALBERT）来解决 IMDB 数据集上的情感分析问题，并对这些方法的表现进行了比较。

| 方法 | 优点 | 缺点 | 精度和性能 |
| --- | --- | --- | --- |
| 朴素贝叶斯 | 简单、高效，适合大规模数据集，计算速度快 | 假设特征独立，不适合长文本和复杂语义关系的捕捉 | 准确率较高，但在长文本分类和复杂情感表达中表现较弱 |
| LSTM | 能够捕捉文本中的时序关系，适合长文本情感分析 | 需要大量数据，计算资源消耗较大 | 精度较好，尤其在处理长文本时较为准确 |
| ALBERT | 强大的上下文理解能力，适合复杂语义分析，轻量高效 | 需要较高的计算资源，训练时间较长，模型较大 | 精度和性能优异，尤其在复杂语义分析中表现突出 |

通过本次实验，我对情感分析的基本概念和常用方法有了更深的理解。情感分析不仅仅是一个分类任务，还涉及到文本的预处理、特征提取、模型选择和评估等多个方面。在实验过程中，我学习了如何使用不同的机器学习模型（如朴素贝叶斯、LSTM 和 ALBERT）来解决情感分析任务，以及这些模型的优缺点和适用场景。本次实验让我更加熟悉了如何在 PyTorch 中实现 LSTM 模型，包括数据处理、模型搭建、训练过程中的优化器和损失函数的使用。通过使用 ALBERT 进行情感分析，我学习了如何加载预训练模型，进行微调，并将其应用于实际任务。虽然预训练模型相对复杂，但通过这次实验，我提升了对深度学习框架的掌握，特别是在处理 NLP 任务时的技巧和流程。