# 随机森林

## 1.1决策树的构建

从B站上学习决策树，随机森林的结构，包括ID3/C4.5/CART决策树，并且跟着教程搭建决策树：通过递归的方式搭建决策树，使用信息增益判断最佳特征为当前点。通过判断当前节点类别是否一致和还没进行分类的特征数量来构造终止条件。

## 1.2随机森林的构建

在B站上学习随机森林思想后通过编写两个类来搭建森林。

先编写DecisionTree搭建出决策树，并且通过遍历每个节点来编写回归模型

然后编写Random Tree来构建通过循环使用DecisionTree搭建随机森林，并在其中通过遍历节点计算每个特征的重要性。同时使用DecisionTree中的predict函数进行回归预测。

最后定义训练模型，使用循环来减少搭建决策树时随机取样和遍历多个num\_trees来找到森林最佳的决策树数量。

计算正确率，通过混淆矩阵表示预测值和真实值的差别和对模型效果进行评估。

计算特征重要性，画柱状图表示归一化后的重要性可视化展现

# Bangumi评论分数预测器的训练

## 2.1数据清洗

给爬虫爬取的文本增加长度限制，去除语义不清，含义太少的不良数据。

通过计算当前动漫评分众数，设置区间，去除评分差距过大的恶意评分和过高评分，让文本与评分关联更大。

## 2.2模型微调

在B站上学习bert-chinese中分词器的用法后使用其进行数据预处理和引入

通过AI找到适合nlp的optimizer：Adam W和可以自动调节学习速率，是的前期收敛迅速，后期学习效果好的get\_linear\_schedule\_with\_warmup。

定义模型，进行训练集训练，然后固定参数进行验证集验证获得loss和accuracy评估模型性能。

由于我的模型过拟合较快，3到4轮训练就过拟合了，于是使用增加Dropout比例进行数据增强和加快学习率。并且设置终止函数在过拟合前保存最佳模型的参数。

多轮调试发现调节学习速率和Dropout比例对模型accuracy的影响不大，个人猜测是跟transformer内部query,key,value参数设置与当前任务不够匹配和训练数据质量不佳引起正确率始终在44%左右

# 3注意力机制及其及其变体的实现与理解

## 学习记录：

本人之前学习了李宏毅老师的transformer课程，所以对注意力机制的结构，原理和代码编写已经有了部分了解，于是在第一题直接自主编写代码。在第二题通过在AI上进一步确认GQA和MQS与MHA的差别后，直接在MHA代码的基础上进行修改，所以可以发现，前三个代码只有在query，key,value的处理上利用不同机制而代码不同，其余地方十分相似。

## 3.1多头注意⼒机制的实现

用linear构建query,key,value的变换矩阵。通过多头分类建立多个子空间来处理文本。

使用kv\_cache方法联系上下文，先判断是否有上文，如果有，将上文的key,value与本文的进行拼接，虽然这样会改变两者序列长度，不过在计算注意力分数时会因为矩阵乘法被消去，所以不用担心

接着计算注意力分数，注意力权重和最终输出并返回权重和输出

## 3.2GQA和MQA

不过是在MHA的基础上多头间共享query,key罢了，直接在view进行子空间建立时调整维度。

MHA:(batch,sequence\_len,num\_heads,head\_dim)

GQA:(batch,sequence\_len,num\_groups,head\_dim)

MQA:(batch,sequence\_len,1(全部共享）,head\_dim0

然后对GQA,MQA用repeat将query,key共享到子空间中每个头即可。

## 3.3MHA，GQA，MQA的比较

将三个模块调用到同一个文件，对同一个随机矩阵进行处理，使用repeat的方法将权重调整到同一维度大小，使用相减的方法观察比较输出和权重，以下是个人发现和结论：

GQA与MQA权重十分接近且因为GQA分组共享key,value和MQA全部共享key,value,权重差值具有一定规律

MQA与MHA权重差距较大，因为两者每个头考虑维度差距较大

MHA与GQA权重差距也有点大，但比MQA与MHA小，因为GQA是对MHA的多个head进行分组，且本代码中单个组内只要2个head，所以差距小

# 4.附加题：多头潜在注意⼒机制的理解与实现

## 学习记录：

看了题目下面的论文，重点看了其中MLA结构解释和附录中的数学表示。在自己用纸笔确认了query,key,value的压缩处理和缓存方式，上下投影矩阵等各种变换的维度之后，开始编写代码。

## 代码编写：

由于水平有限，没能做到将W\_UK,W\_UV分别吸收到W\_Q和W\_O中，按照论文内的数学表示和个人理解设置了一系列变换矩阵：

self.batch=batch

self.embedd\_dim=embedd\_dim

self.sequence\_len=sequence\_len

self.num\_heads=num\_heads

self.head\_dim=head\_dim

self.compress\_dim=compress\_dim#query压缩后的维度

self.head\_latent\_dim=int(latent\_dim/num\_heads)

self.W\_Q=nn.Linear(embedd\_dim,embedd\_dim)

self.W\_K=nn.Linear(embedd\_dim,embedd\_dim)

self.W\_V=nn.Linear(embedd\_dim,embedd\_dim)

self.W\_O=nn.Linear(embedd\_dim,embedd\_dim)

#上投影矩阵和下投影矩阵

self.W\_DQ=nn.Linear(embedd\_dim,compress\_dim)

self.W\_UQ=nn.Linear(compress\_dim,head\_dim)

self.W\_UK=nn.Linear(latent\_dim,embedd\_dim)

self.W\_UV=nn.Linear(latent\_dim,embedd\_dim)

self.W\_DKV=nn.Linear(embedd\_dim,latent\_dim)

#定义将解耦后的key压缩来缓存的矩阵

self.W\_DK=nn.Linear(2\*embedd\_dim,latent\_dim)

#定义ROPE的上投影矩阵

self.W\_QR=nn.Linear(compress\_dim,embedd\_dim)

self.W\_KR=nn.Linear(latent\_dim,embedd\_dim)

分别对应query的上下投影和压缩，key,value的缓存压缩和上投影，实现ROPE的上投影矩阵。

先将上文缓存的key,value解压缩后与当前的拼接，联系上下文。然后将query,key,value压缩来减少数据量，对应论文中节约算了，消除键值缓存瓶颈的创新。

接着使用ROPE方法，对压缩过的query，key解压缩后先按照两者序列分别建立旋转矩阵。按照论文方式先设置三角函数的频率，再用时间序列模拟位置序列，然后使用欧拉公式建立复数矩阵得到旋转矩阵。之后将query,key旋转，得到两者位置信息。

位置信息与文本信息的解耦:在使用ROPE方法得到位置信息后，将其与之前的文本信息拼接。

将新的key,value缓存。

分头建立子空间

计算注意力分数和权重，并得到最终输出

## 以上就是我的学习记录和代码思路

我分别在周六一下午和晚上完成随机森林，并在周三进行改进。

在周日完成bert-chinese,并且进行相应调试，并想办法提高正确率，最终失败并得出上述结论。

在周一晚上完成MHA，GQA，MQA和其对比

在周二，周三晚上看了论文并完成MLA。

## 虽然我还有很多不足，并且还是处于起步阶段，不过毕竟还是大一，我相信给我时间学习，我可以成为实力型选手。