## 2024.11.1

1. 在Aviv的数据集上跑一下bagging，跑起来比较慢，已经一天了。

2. 师兄提供了算力，找jiaqi师兄要L40S的算力，可以联网，但不清楚是否只能在fit楼的网上使用。

3. 算法需要修改一下，计算overlap时只在validation上计算overlap，保证每一次实验都在相同的数据上计算overlap。

4. 不要把DKL和bagging加在一起，轮子套轮子没有好结果，对算法进行优化要比对模型进行优化来的好。

5. 科学问题：在序列数据上个使用主动学习的方法进行表达量预测。另外，之前没有工作总结过在序列数据上做主动学习的benchmark。

6. Enamine的数据本身不是序列数据，而是fingerprint，不具有说明性。

## 2024.11.2

1. 把summary/bagging中增添了Aviv数据集的接口，同时将在all\_data上做重合度计算改成了在valid\_data上做重合度计算，删去训练过程中在validation上计算pcc的部分简化训练过程。更新：在summary的三个文件夹中main函数均做了这项修改。

2. problem：关键点不在于找到一个合适的不确定度量化方法，而在于找一个合适的指标来指导主动学习，这个指标可能是不确定度，也可能不是。即便能完全量化出不确定度，这也不一定能提高主动学习的性能。**已有的实验内容不能说明ensemble得到的不确定度是否有用，也不能说明不确定度对于主动学习是否有性能上的提升。**

3. EI方法：按照道理来说，使用EI筛选出来的数据应该有较大的预测均值和较小的预测不确定性。

4. 重大bug：筛选数据时没有把valid\_data从数据池中分离出去，已经对summary中的代码进行了修改。除此之外的没有修改。

## 2024.11.5

1. 在Aviv上使用GP跑出现了问题，原因是cpu内存太小，导致无法顺利完成训练。具体出问题的点在于训练后对train pool预测的地方，或许可以将这一过程进行分布计算。

## 2024.11.6

1. 把之前在所有数据上计算overlap的图像更名为summary\_overall，在validation上计算overlap的图像命名为summary\_valid。

2. 为了便于同时训练，在summary文件夹下新建了bagging\_no\_aviv，主要区别在于去掉了aviv数据集用到的seq2onehot接口，防止对原有三个数据集训练的影响。

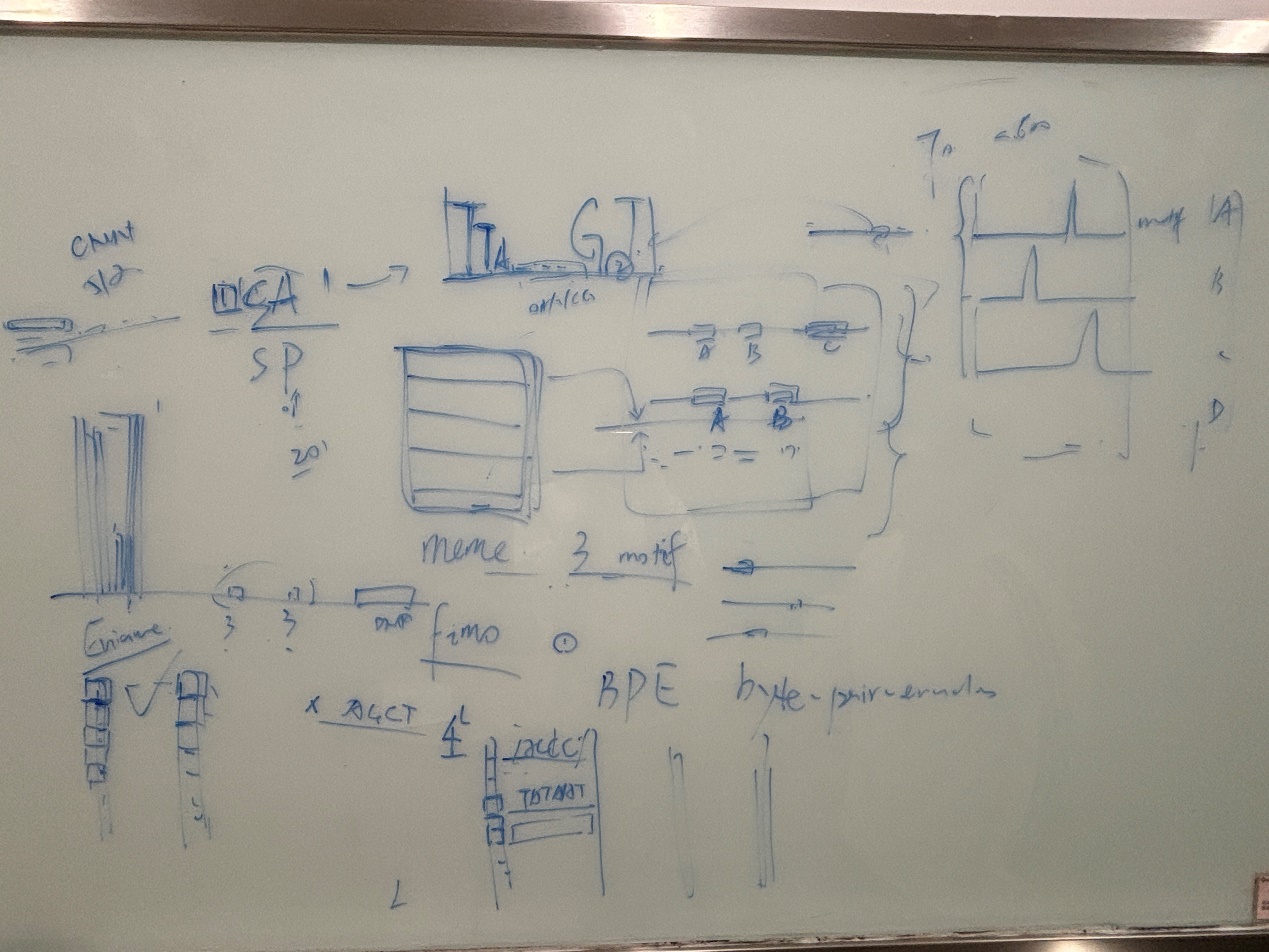
3. 更改了Aviv数据集的命名，改为Yeast\_aviv\_2022，在所有txt结果文件中也进行了修改。

## 2024.11.8

1. 和汪老师交流：

* 确定问题是解决根上的问题（设计一种适合所有数据集的主动学习算法，对数学要求高），还是解决枝上的问题（设计只在生物数据集上有价值的算法）。生物数据由进化决定是一簇一簇的，这不是随机采样能采出来的。生物主动学习的实现形式（人机闭环）决定了其迭代次数小，迭代数据少。
* 多读文章，每天抽出两小时读文章。边看边试方法，有目的地试方法，形成对主动学习的认识。
* 把毕设当做解决问题中的一环，而不是简简单单一个设计。

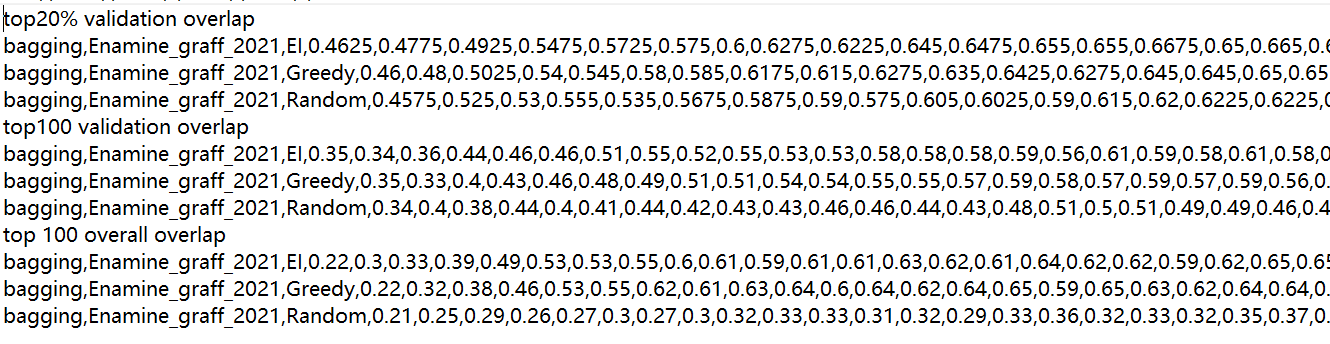
2. 和杜师兄交流：



* Byte pair encodings，在NLP中找一些可能的对序列空间表征的方法。
* Meme和fimo，两个搜索序列中重复度高motif的模型，或许可以用于提取语义。
* Enamine在DKL方法下很好，可能是因为fingerprint对序列空间进行了降维表征（GP和DKL对于高维数据的学习性能不好）

## 2024.11.10

1. 今天师兄发现validation上计算top20% overlap时，EI/Greedy和Random之间没有明显分离。可能原因有：用于计算overlap的数据量过大（数据量不能太大是因为只有最前面的一些高表达数据有明显特征，后面的特征就不明显了，即便是random也能很好的学习） or 不应该做validation分割。下面有一些证明：



微信原文“100条就会有EI or Greedy和Random分离了，如果不做分割的话分离会更明显”

2. 于是决定按照top100 validation的条件把之前的结果再跑一遍。

## 2024.11.11

1. 对checkpoints的储存结构进行了修改，结构为“dataset/algo/sample\_method”储藏每种算法和采样下最后一个iter的十个模型结构（GP和DKL是一个）。

## 2024.11.16

1. 之前已经完成了BPE和使用onehot编码进行embedding的部分，但是由于每条序列含有的分词个数不同导致没有办法直接转成tensor和输入到NN中进行训练，因此还需要修改。具体的修改方式有使用填充或者截断or使用对序列长度无要求的RNN等，目前使用的是填充的方法，保证使用token表示后序列长度一致。

2. BPE/data\_process.py中三个函数的接口有点问题，将其处理成序列调用关系。

3. 将list整理成array，可以提高矩阵运算的速度，这是一个大工程可以放在后面来进行。

## 2024.11.17

1. 测试了50w和10w下，BPE-Onehot和Onehot两种方法，使用的模型为NN和RNN：

100000 BPE-Onehot NN:

Spearman Correlation Coefficient: 0.054747982371091075

p-value: 1.661355615139342e-34

500000 BPE-Onehot NN:

Spearman Correlation Coefficient: 0.10786130670903465

p-value: 2.938982666782882e-129

500000 Onehot NN:

Spearman Correlation Coefficient: 0.18340995896114878

p-value: 0.0

100000 BPE-Onehot RNN:

Speaman Correlation Coefficient: 0.01893758464057371

p-value: 0.05826596564718465

100000 Onehot RNN:

Spearman Correlation Coefficient: -0.07046036580860365

p-value: 1.7394184673973626e-12

跑完之后发现词汇表大小出现了一些问题，设定词汇表最大值为200在10w条的情况下就偏小，这种情况下最后的词汇表大小为192，分词后序列长度为31，显然不符合**motif长度6到15个碱基对(bp)**这一先验知识。

设定为500，词汇表大小为498，分词后序列长度为28。

设定为1000，词汇表大小为998，分词后序列长度为27。

设定为2500，词汇表大小为2498，分词后序列长度为24。

设定为5000，词汇表大小为4998，分词后序列长度为22。

设定为10000，词汇表大小为9998，分词后序列长度为21。

设定为100000，词汇表大小为99983，分词后序列长度为17。

## 2024.11.19

1. 今天首先跑了一下10w条数据下，1000词汇量的BPE-COOmtx/SVD方法，结果如下：

100000data 1000vocab BPE-COOmtx/SVD NN:

Spearman Correlation Coefficient: 0.02324813168848132

p-value: 0.02008069206332053

跑完之后意识到了一个问题——**词向量平均**：按理来说NLP中的做法是每个token的词向量做加和平均，得到的这个和单个token词向量等长的向量作为序列向量。我在这里并没有用词向量平均，而是把每个token的词向量顺次排列在一起，这保留了次序信息，在RNN或者CNN中可能有效果。

2. 另外跑了一下BPE-CBOW的方法，方法中用的是词向量平均，结果不是很好：

100000data 1000vocab BPE-CBOW NN:

Spearman Correlation Coefficient: 0.002009116880091169

p-value: 0.8407871086833066

Pearson Correlation Coefficient: 0.5444921027165721

p-value: 0.0

Top Residula Values: 3.0077977776202016

Internal Coverage: 0.6291788647917529

## 2024.11.20

1. 师兄给了四个指标，scc / pcc / residual of top values / internal coverage pcc：

**SCC：**

衡量预测值的排名顺序与真实值排名顺序之间的相关性，对数值不关注，仅关注相对排名。

**PCC：**

衡量两个变量（预测值和真实值）之间的线性相关性。

**Residual of Top Values：**

评估预测不确定性最高的一部分数据对应的真实误差。

**Internal Coverage PCC：**

分析预测值的置信区间是否可以合理覆盖真实值，这里的PCC衡量的是预测区间和真实区间覆盖率的相似性。

2. 使用了BPE-skipgram，结果比BPE-CBOW还要不好：

Spearman Correlation Coefficient: -0.00014254084942540848

p-value: 0.988628692332661

Pearson Correlation Coefficient: 0.5582712309099054

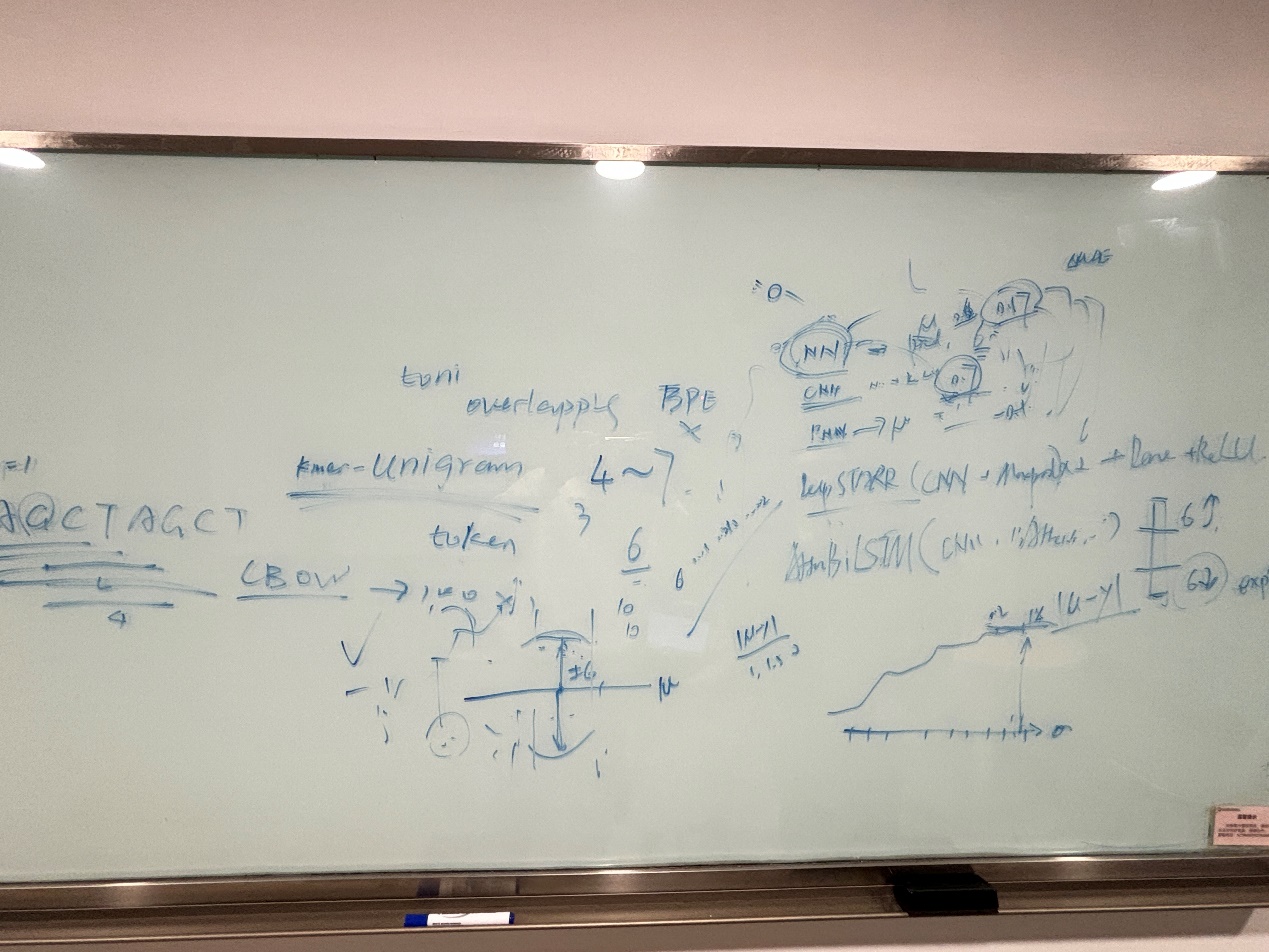
p-value: 0.0

Top Residula Values: 2.8150341645618058

Internal Coverage: 0.5975297733536843

3. 测试了yau的k-mer方法，结果和直接用onehot编码没有什么太大区别。突然意识到k-mer是一种有重叠的读取方法，或许DNA在表达信息时确实是存在重叠读取的，那是否可以找到一种**有重叠的BPE方法（或者有重叠的其他tokenizer方式）**？另外yau的k-mer是使用了词向量平均的，因此在词向量平均的情况下会丢失序列化信息，是否**可以把k-mer依次排开**？

## 2024.11.24

1. 

2. 目标是实现有overlapping的tokenizer。具体的实现方向有：(1) 基于可变长k-mer和统计方法的最优k-mer选择（应该有很多已知的k-mer选择方法，偏生物而非nlp，读这方面文章时注意是否使用overlapping是最优的的问题）；(2) 在BPE等传统tokenizer方法基础上实现overlapping功能（目前没有看到有工作做这样的内容，需要多查阅tokenizer的原理）。

3. 做一组non-overlapping的实验（NO），以及一组多次编码non-overlapping的实验（MultiNO），以此来探究overlapping是否重要。

实验结果是上述提到的这两种都不行）说明overlap大势所趋，因为overlap的存在才有更好地训练结果。

## 2024.11.26

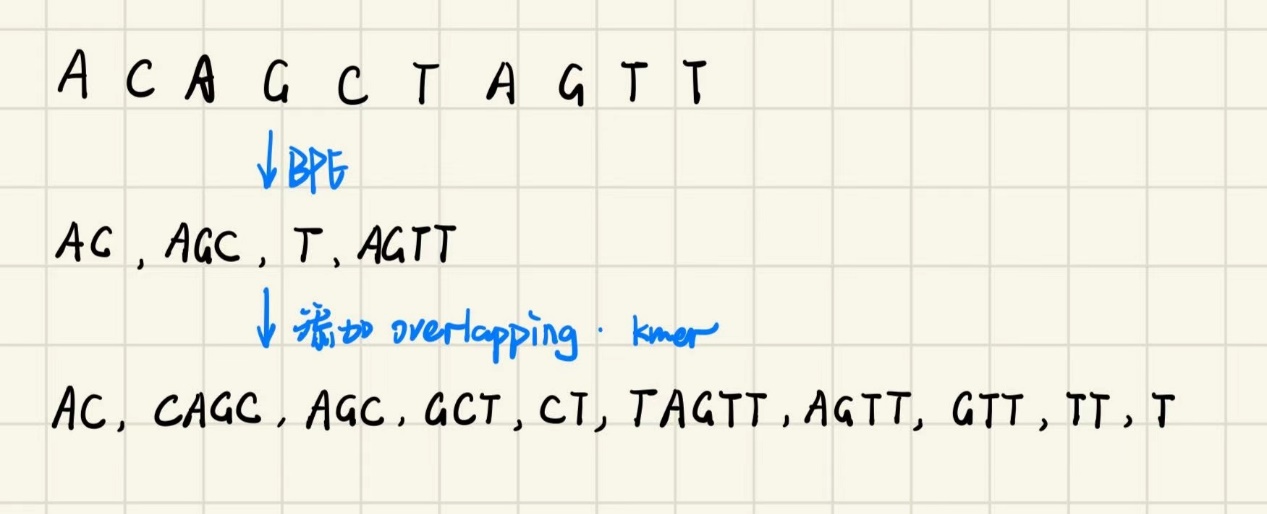
受不了了，写的基于BPE的Overlapping算法完全没法用，比如在分词后前后都加上原有的bigram如“A C G T T T C G A” --> “A C G T T TC” + “TC G A”会引入大量“TC”，导致模型偏向成如“TTTTTTC”这样的bigram，这不是我们想要的。

有两个解决方案可以尝试：

1. 在分词的时候进行正常的BPE，并记录merge的过程，在使用merge过程复现seqs的tokenize过程时再加上bigram。
2. 分词前后都加上原有bigram的模糊版本，比如对于“TC”这个bigram前面的序列加“T”，后面的序列加“C”，类似添加一个边缘遗忘机制。

## 2024.11.27

1. 想了一种新的解决方案，将其暂且称之为OverlappingBPE，实现如下：



在BPE词汇表250/500/1000三种情况下做了一下，结果都是略优于onehot，还需要再改进。

2. 想了一种新的算法，OverlappingBPE2：在BPE1中，相当于把tokenB和前面的部分tokenA结合起来，BPE2中，则是取后面tokenB的部分，保证拼接后长度和tokenA相同。（避免出现两个6mer加起来变成12mer的情况，拼接后应仍为6mer）

这种方法得到的结果较好，但是还没有直接4-mer或者6-mer的结果好，而且这种良好的结果纯粹是因为取了4~6mer的片段，和random是一样的。

3. 另外重新做了一组unigram-CBOW的实验，因为之前的训练集/验证集划分出现了问题。在result.csv中进行了直接的更新（PCC 的pvalue不等于0了，有点奇怪，1.2528297049192895e-70）

4. 待重测BPE-CBOW实验，两次实验结果不一样。

5. 做了一下富集情况，因为使用BPE2的条件在于，某个token长度为n，则他附近的token长度也更可能为n。但事实上，富集情况并不明显：

