**提出问题：**

本文是基于向量分解+编码压缩的方式来对测试集进行压缩并得出最终的压缩率。主要的思想是将测试集，使用主分量+残分量的形式来表示，残分量拥有较少的有效信息（较多的无关位含有较少的1），相对于完全混乱的测试向量而言，更好压缩。

验室之前使用哈达玛矩阵以及最大相容类的方式来选取主分量集合。

但是两者均有其自身的局限性，哈达玛矩阵虽然压缩效果较好，但是产生的基向量集合于元测试向量无关，因此压缩率应该还能提升。

若使用最大相容类的方式，首先寻找最大相容类的时间复杂度就是O立方级别。其次涉及的测试向量众多，要选取出有代表性的主分量，然后两两异或产生主分量集，从而达到全局最优解并没有一个很好的办法。若使用贪心法，那么选取的主分量有两个问题，

1. 不能涵盖所有向量的特征，并且可能相似，那么异或之后产生的向量特征较少，无法覆盖更多的测试向量。
2. 一开始的最大相容类拥有众多的测试向量，后面越来越少，即使如此，选出的基向量能涵盖的测试向量也是少数，效果不佳。

**解决思路：**

实质上要提高压缩率需要解决的问题有两个

一个是选取的向量要和原测试集有联系。

第二选取的基向量有足够的代表性，即基向量异或之后能产生更多与原测试集相似的向量，即生成的主分量集合可以与原测试集更相似。

基于此，我们提出使用聚类的方式来解决这个问题。就是利用特定的聚类算法，将原测试集分成K类，找出K个集合的聚类中心，对于这K个聚类中心那么均和这个K个聚类中的测试向量很相似。

为了保证这K个测试向量异或出来的结果更加具有代表性，更能与原测试集相似，选取的这个聚类中心之间的距离必须足够远。

综上所述k-means++算法符合我们的需求，并且经过实验证明此算法能产生很好的结果。

**具体步骤：**

步骤一：对无关位进行预填充。 对于不同的编码方式，无关位的填充方式对数据压缩率有决定性的影响。一旦编码方式得以确定，无关位具体是填充0或者1也随之确定，填充的目的是为了使测试集在当前编码规则下的压缩率更高，一旦我们填充完毕，当前的测试集中就不存在无关位，对于一个完全确定的测试集，其中所包含的等价列向量是相当少的，因此我们需要使用合适的聚类算法，选取此测试集中我们所需要的k个聚类中心。

步骤二：由于原测试集中的无关位已经得到了填充，我们可以使用机器学习中的K-means++聚类算法找到我们所需要的“基向量”,每次在选取“基向量”时，预先确定我们所需要聚类的数目，之后找到聚类中心对应的列作为我们所确定的基向量，再根据确定好的基向量生成主分量集。最后的我们将生成的主分量集与未填充前的测试集进行异或产生残差集。

基向量产生方法如下所示：

1. 从我们生成的“填充测试集”随机选择一个列向量作为第一个聚类中心。
2. 对于数据集中的每一个列向量x，计算它与最近聚类中心(指已选择的聚类中心)

的距离D(x)。

1. 在原测试集中选择一个新的列向量作为下一个聚类中心，选取为下一个聚类中心的概率为，显然D(x)较大的点，被选取为聚类中心的概率也较大。
2. 重复2和3直到k个聚类中心被选择出来，其中k的数量由我们原测试集的数据规模决定。
3. 选取出来k个初始的聚类中心之后，计算每个列向量到聚类中心的距离，这里采用欧式距离计算。
4. 每个列向量均能计算出k个距离，我们将其划分在最小距离所对应的聚类中心的类中。
5. 对应类簇中所有数据对象的均值，即为更新后该类簇中心。
6. 判断中心点是否满足迭代条件，如果不满足则返回第二步重复计算代条件可以设定为迭代之后类族的中心点不发生变化或者直接初始化迭代的次数。

此时，我们需要找的k列基向量已经确定，那么主分量集也随之确定，对于k列的“基向量”，它有种进行异或的组合方式，也就可以生成个列向量，与此同时我们将测试集中具体的每一列与这个列向量中的每一列进行异或操作，最后取异或之后值最小的那一种组合对应的列向量作为主分量集中的一列。

**实验效果评论**

如下是实验效果相关的数据，选取的基向量个数从7-11不等，我们一般按照log2为底n的对数来取具体的基向量，其中n代表原测集的行数。

以下的结果是当前测试集再log2为底n的对数个基向量的情况下，所达到的压缩率。

本人也通过将基向量的个数一个个增加，描述出的压缩率变化的大致曲线。

**FDR编码压缩率（kmeans++）**



**EFDR编码压缩率（kmeans++）**



**VIHC编码压缩率（kmeans++）**



**golomb编码压缩率（kmeans++）**



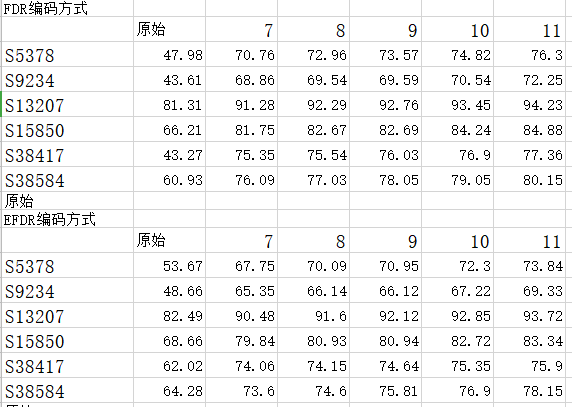
**ALT-FDR编码压缩率（kmeans++）**



**RL-Huff编码压缩率（kmeans++）**



这个图是当选取不同的基向量个数之后产生的压缩效果：



**结论：**

使用kmeans++确实达到了一定的压缩效果，但是kmeans++算法也有一定的局限性，比如不能自主地确定聚类中心的个数，因此我们选取基向量时可以在选取个数和提高压缩率的两个维度做权衡。

之前还尝试过使用别的聚类算法来压缩测试集，例如kemans算法和db\_scanner

算法，但是最终的效果不佳，kmeans算法选取的聚类中心并不具代表性，因为初始聚类中心的聚类不够大导致，最终选取的基向量可能存在相似度很高，异或之后产生的向量于原测试集存在一定的差距。

Db\_scanner算法实质上是一种基于密度的聚类算法，可以自动确定聚类中心的个数，但是我们必须先确定聚类的半径r，并且在以这个半径位圆的区域内出现的向量大于k个才被定义为是一个聚类，半径和区域内向量的个数非常难以确定，因此也没有达到预期的效果。