机器学习与数据挖掘期末课程报告

题目	经典聚类方法探索
姓名	毕泽同
学号	21307129
专业	计算机科学与技术(人工智能方向)

报告概要

在第三次作业中，我对于两种聚类算法(k-means、GMM)进行了探究，在 MNIST 数据集上对两种算法进行了测试与优化。但是在进行了更多的实验探究后，我认为hw3中的模型还有很多可优化的空间，所以便有了本次课程报告的主题，对经典聚类方法的探索。

在本次课程报告中，我想在无监督聚类方法的领域继续探索，我的主要工作有两部分：

1. 在传统聚类算法上提出改进方法，对其在准确率/迭代次数上进行优化。
2. 在MNIST数据集上实现并对比现有聚类方法。

> 需要注意的是，对于聚类问题而言，一个更重要的问题应该是对数据的抽象与表示，但是我认为那样就变成了一个类似于表征学习领域的问题，所以在本次课程项目中，将不会对涉及到对数据的优化处理。所有模型及实验都将部署在一个相同的基准数据上。(问题的关键当然是数据的表示问题，但是我们在本次报告中只讨论如何使用算法分析现有数据，而不是更改数据。我们希望，即使最终数据表达能力有限，我们的聚类算法仍然可以有很好的效果。)

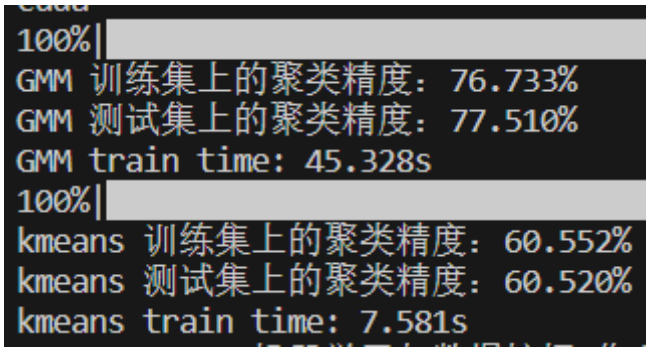
一、背景与动机

聚类是一种无监督学习方法，旨在将相似的数据点分组到同一簇中，从而揭示数据的内在结构。这个问题在数据挖掘、模式识别和机器学习等领域中具有重要意义。随着数据不断增长，聚类成为处理大规模数据集和发现隐藏模式的关键技术。通过聚类分析，我们能够识别数据中的潜在群组，为数据集的进一步分析和理解提供基础。聚类应用广泛，包括市场分割、社交网络分析、图像分割等，因此对聚类方法的研究和改进具有实际应用的价值。

在hw3中，我们使用GMM在测试集上实现了 `77.510%` 的准确率,在训练集上实现了 `76.733%` 的准确率，用30轮迭代，45.328s.

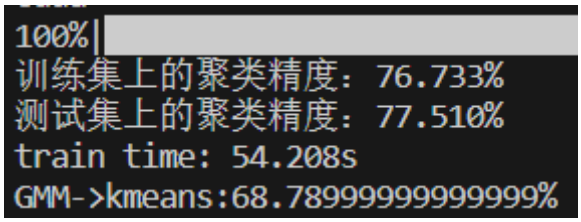
使用kmeans在测试集上实现了 `60.520%` 的准确率，在训练集上实现了 `60.552%` 的准确率，同样的30轮迭代，7.581s.

> 具体程序见 main.py



比较两个模型，可以发现二者都存储或利用了聚类中心坐标，那么一个很自然的想法就是将聚类效果更好的GMM模型的中心坐标，直接赋给kmeans模型,最终实现了 `68.79%` 的准确率。

> 具体程序见 main2.py



结合以上两个实验，我们可以得出以下两个结论：

1. kmeans方法的聚类中心还有很大的优化空间，我们原本的kmeans方法所训练出来的聚类中心可能只是一种 局部最优 而已。
2. kmeans的泛化能力一般，我们可以看到kmeans方法在测试集和训练集上的准确率是相当的，甚至训练集的精度要高于测试集。对比之下GMM的泛化能力似乎要更强一些。

结合以上两点结论，在本次实验中，我的主要动机或目标是让kmeans的泛化性能更强一些，尽量避免kmeans的 过拟合 / 局部最优 问题。而为了达成这一目的，我在kmeans最重要的更新步骤中做了一些改动，通过让其他类别的数据参与中心点的更新，来 "破坏" kmeans的收敛，为其**增加一些噪声**，使其泛化性能更好。

最终经过测试，我们的方法的准确率达到了66.2%左右，相比于最初的kmeans算法提高了约10%，而运行时间的代价只有6秒左右。这从实际层面证明了优化方法的有效性。

> 也许你会说聚类是一个非常随机的过程；并且增加噪声，这一非常冒险的举动，一定需要大量的测试以及调参，是不稳定的。但在实验过程中，只要参数在合理的范围内(这个范围足够人来定位)，模型都能够有不同程度的提升，方法的有效性是不需要质疑的。

> kmeans作为一种经典的聚类算法目前仍在广泛使用着，相比于其他算法，其简单高效、可扩展性可解释性更强，适合于各种数据类型。而这也为本文的优化方法提供了某些现实意义。

二、现有方法

当前解决聚类问题的主要方法包括：

1. K均值聚类： 基于数据点与簇中心的距离进行分组，通过迭代更新簇中心来最小化总体平方误差。
2. GMM(高斯混合模型, Gaussian Mixture Model):假设数据是由多个高斯分布组成的，每个分布称为一个混合成分。GMM通过学习每个高斯分布的均值、协方差矩阵和混合系数，可以灵活地拟合各种形状的数据分布，因此在聚类和密度估计等任务中广泛应用。
3. 层次聚类： 基于数据点之间的相似性构建层次结构，通过逐步合并或分裂簇来得到聚类结果。
4. 谱聚类： 利用数据的图表示，通过图的特征向量进行聚类，适用于非凸形状的簇。

这些方法在不同情境下表现出色，但也存在一些挑战，例如对初始参数敏感、对噪声敏感等。因此，寻找改进和新的聚类方法是当前研究的焦点。

三、模型、算法或方法

> 模型具体实现请见myKMeans.py

3.1 算法流程

算法流程与传统kmeans基本一致

1. 初始化：随机选择 k 个初始簇中心 $C^{(1)}, C^{(2)}, ..., C^{(k)}$.使用kmeans++方法
2. 簇分配：对于每个样本点 $x^{(i)}$，计算其与每个簇中心的距离，并将其分配到距离最近的簇中心：

$$c^{(i)} = \arg\min_j ||x^{(i)} - C^{(j)}||^2$$

3. 更新簇中心：对每个簇 j，更新其中心为该簇内所有样本点的均值：

$$C^{(j)} = \frac{1}{|S_j|} \sum_{i \in S_j} x^{(i)}$$

4. 重复：重复步骤 2 和 3，直到簇中心不再发生显著变化或达到预定的迭代次数。

但与传统方式kmeans不同的是

- 传统kmeans只是对该类别的样本点进行平均，并不涉及到其他类别

- 在我的方法中，我将其他类别的数据也考虑了进来，通过加入其他类别的数据作为 "噪声"，来增大模型的泛化性能。

```
def upd(self, X, labels):
        # 更新聚类中心
        for i in range(self.n_clusters):
            if np.sum(labels == i) > 0:
                noise_data = X[labels == i]
                for j in range(self.n_clusters):
                    noise = X[labels == j]
                    random_noise_index = np.random.choice(
                        noise.shape[0],
                        size=int(self.p * noise.shape[0]),
                        replace=True,
                    )
                    noise = noise[random_noise_index, :]
                    noise_data = np.concatenate((noise_data, noise), axis=0)
                self.centroids[i] = np.mean(noise_data, axis=0)
        self.p *= self.lr
```

具体实现可参考上面的代码，对于每一个类别

1. 首先提取这个类别的数据——> `noise_data`
2. 然后，考虑对于所有的类别，提取其中样本点，并随机选择一部分数据加入到 `noise_data` 中。提取个数由概率 `self.p` 决定。
3. 遍历完所有类后， `noise_data` 将由2部分组成：自身类别的所有样本点，其他类别的小部分样本点。然后对这个加入了噪声的数据进行平均，得到这个类的样本中心。
4. 在为所有的类都更新了中心点以后，将概率 `self.p` 乘以一衰减因子 `self.lr` ，通过一种退火的方式逐步减小噪声数据的比例，从而确保模型能够顺利收敛。

3.2 可行性分析

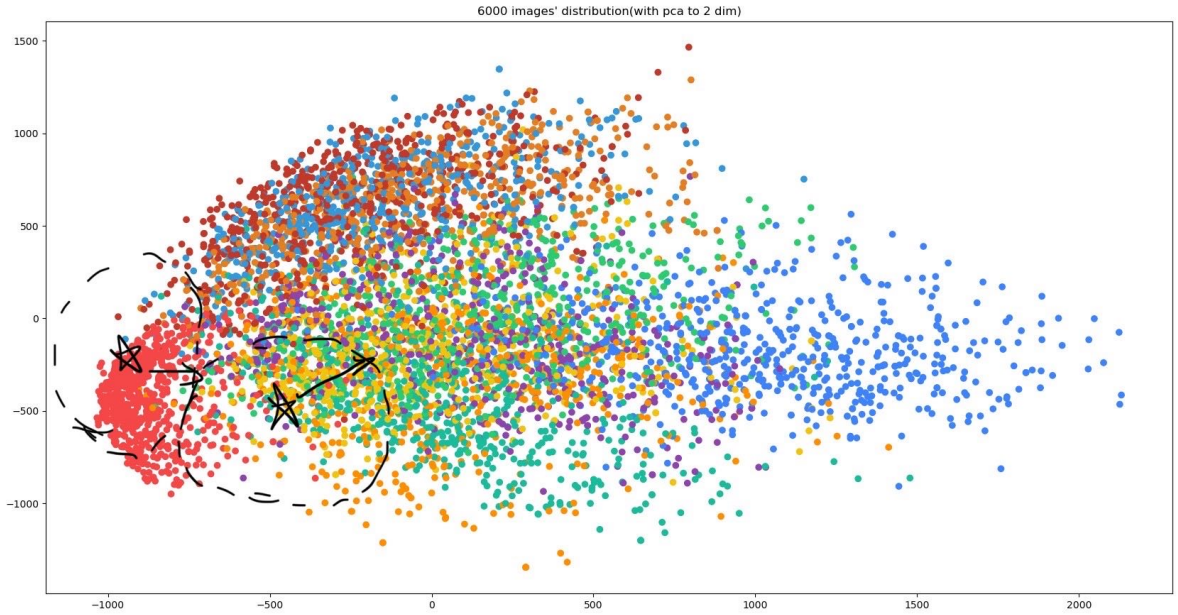
> 为什么这个方法是可行的，让我们从一个数据探索的角度进行分析。

$$C^{(j)} = \frac{1}{|S_j|} \sum_{i \in S_j} x^{(i)}$$

对于某个类的所有样本点，将均值作为样本中心点确实是最优策略，此时均方误差是最小的。没有必要，也不应该考虑其他类别的数据。
但需要注意的是，kmeans是一个迭代更新的方法，每一步更新时计算的样本点不一定包含其所有的样本点，还有可能有其他类别的样本点。

而如果在更新时考虑其他类别的样本点，那么就相当于将该类的范围扩大了，因为此时中心是噪声和样本点的均值，而噪声是其他类别的数据，是和样本点有一定距离的。所以此时，模型的精度会略微下降，但是每个样本点覆盖的区域更大，因此更不容易让中心点陷入到某个局部的小区域，造成过拟合，而是可以让聚类中心走进其他的类别，将其他类别中的数据标记成自己的数据，从而找到一个更加好的解(当然，这个解也有可能是局部最优，但因为其增大了搜索范围，所以最后的结果往往会好一些。)

> 绘图模块见data_show.py



上图是随机采样6000个样本后，pca到2维的散点图。从图中可以看出不同种类的数据重叠是非常严重的，这也是影响kmeans性能的主要因素。

考虑左下角的样本点，每个五角星代表当前的聚类中心，左边的五角星定义为1号中心，右边的五角星定义为2号中心。由于每次更新考虑其他类的数据，所以两个中心都会有一个向右的趋势(已在图中标出)，这使得1号中心能够覆盖更多的红色数据，2号中心能够覆盖更多的黄色数据。但是如果使用传统的kmeans方法，那么聚类中心在此时收敛也是有可能的，又或者说样本点运动的剧烈程度不会这么大。
所以，在这个例子中，我们通过加入噪声数据，使得聚类中心“覆盖的范围”变大了，从而带动聚类中心的移动，使得其更不容易收敛于某个局部区域，能够向其他类的区域靠拢，吸收更多可能的数据点，进而增强了模型的泛化能力。

3.3 算法改进

在上面的实现方法中，我们增加噪声的方式是从每个类中选取一定的样本点，参与聚类中心的更新。但这样需要遍历每一个类，从每一个类中获取 `self.p*类中数据数量` 个数据，算法复杂度为$O(n^2)$.

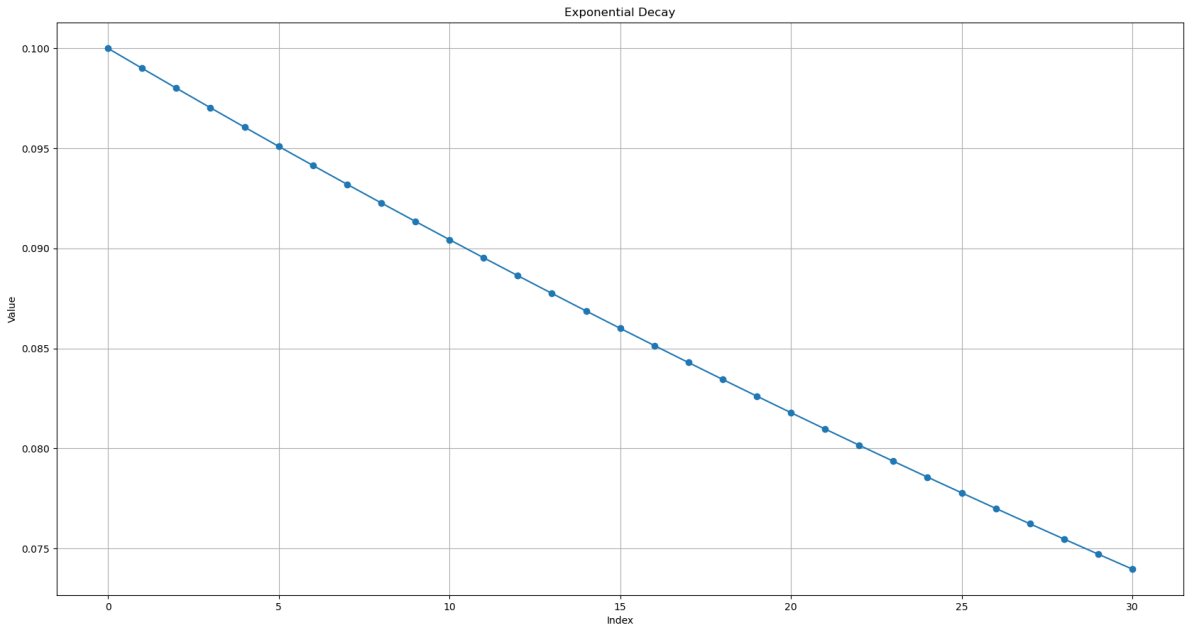
一种更直接、更简单的方式是直接从总样本中进行随机采样，将噪声数据的数目设置为 `self.p*总样本数据数量` ，如果不同类的数据在总样本中的分布是比较均匀的，二者可以达到近似的效果，而此时算法复杂度为$O(n)$.

具体代码实现如下：

```
def upd(self, X, labels):
        # 更新聚类中心
        for i in range(self.n_clusters):
            if np.sum(labels == i) > 0:
                noise_data = X[labels == i]
                random_noise_index = np.random.choice(
                    X.shape[0],
                    size=int(self.p * X.shape[0]),
                    replace=True,
                )
                noise_data = np.concatenate(
                    (noise_data, X[random_noise_index, :]), axis=0
                )
                self.centroids[i] = np.mean(noise_data, axis=0)
        self.p *= self.lr
```

四、实验结果及分析

随着模型的迭代， `self.p` 应该越来越小，这样模型在找到较好的聚类中心时才能够逐步收敛，所以在这里我使用了一个简单的乘子，确保p能够逐步下降，当p初始值为0.1， `self.lr` 为0.99时，下降曲线如下所示。



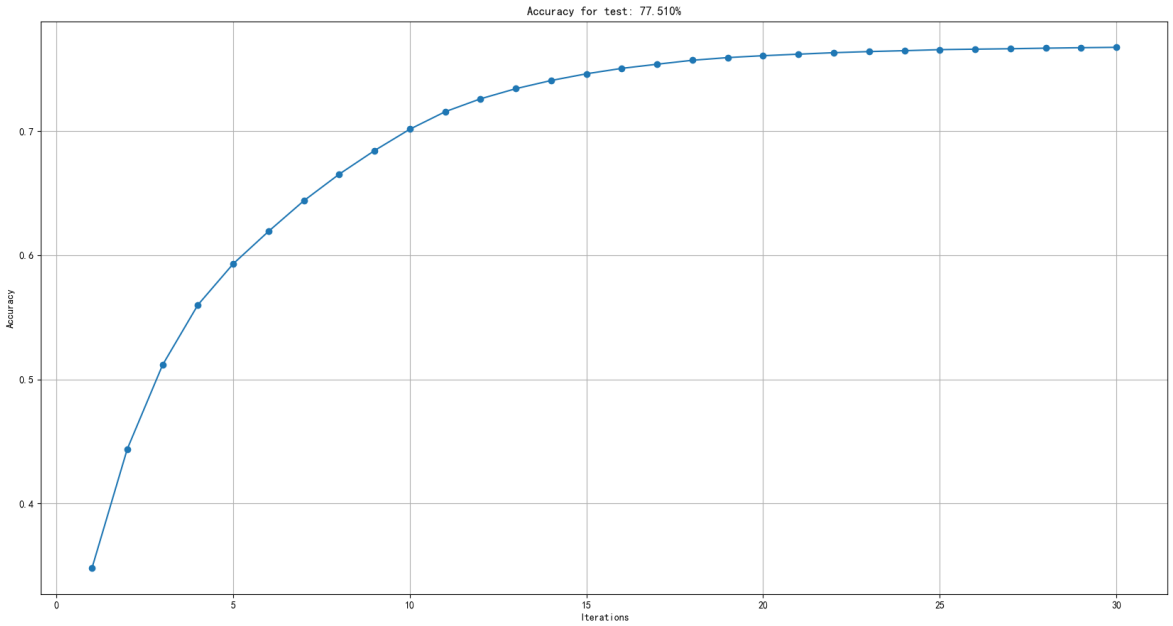
> 分别运行main.py和main5.py

GMM:

GMM 训练集上的聚类精度：76.733%

GMM 测试集上的聚类精度：77.510%

GMM train time: 55.023s

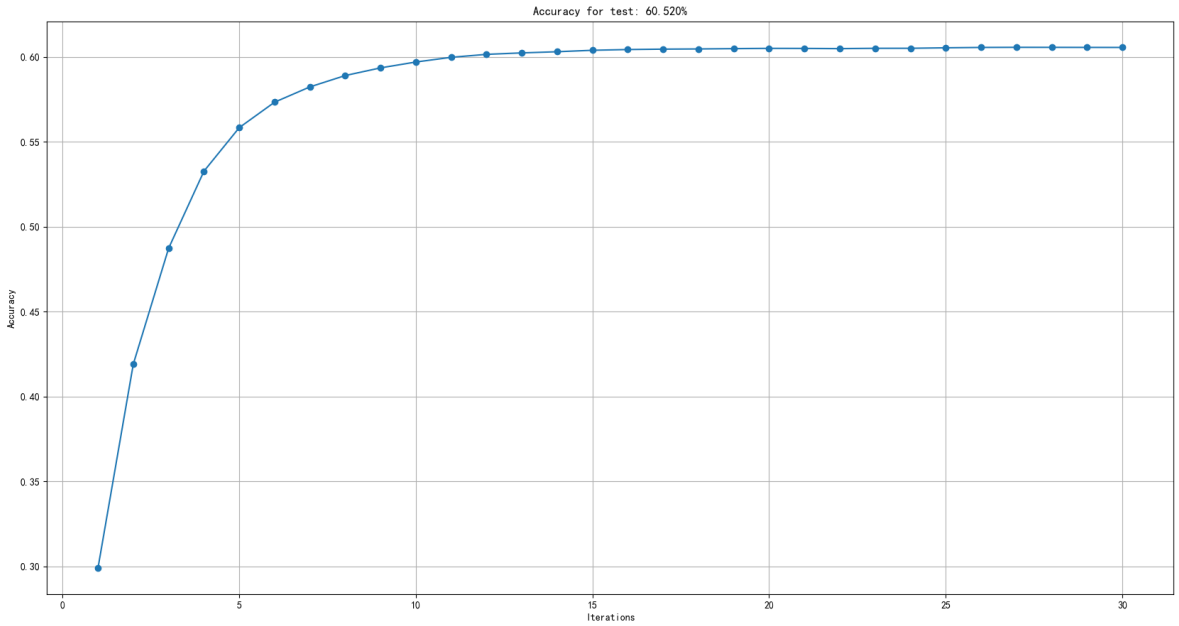


kmeans:

kmeans 训练集上的聚类精度：60.552%

kmeans 测试集上的聚类精度：60.520%

kmeans train time: 8.991s

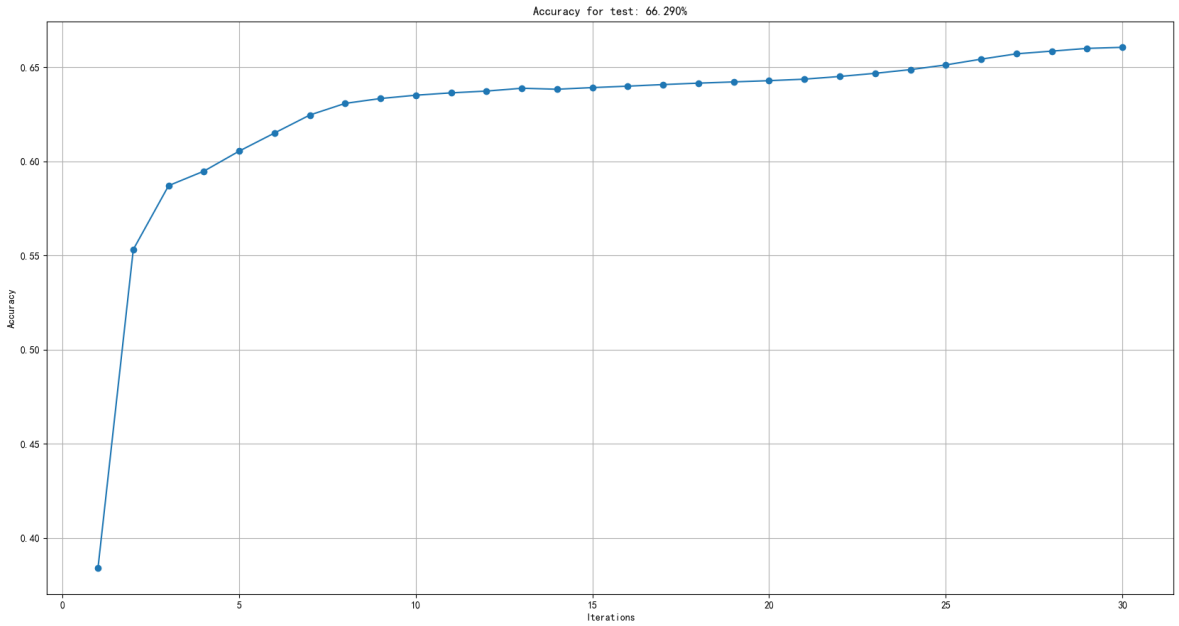


mykmeans:

mykmeans 训练集上的聚类精度：66.055%

mykmeans 测试集上的聚类精度：66.290%

train time: 14.633s

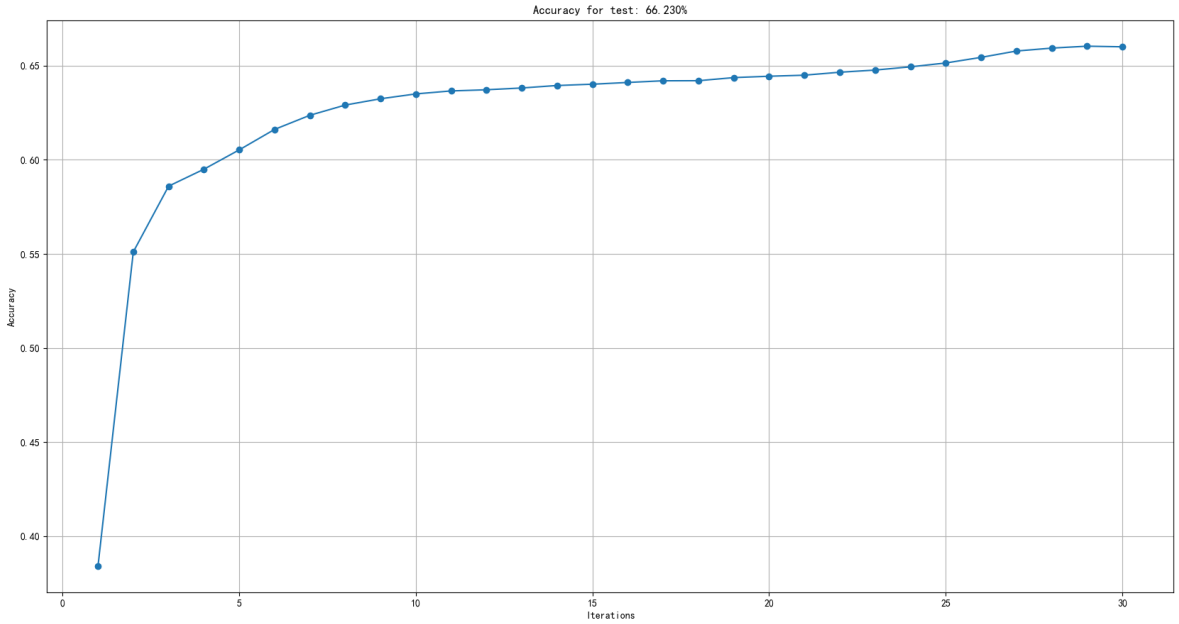


根据以上实验结果，可以得出以下结论：

1. GMM模型仍然是当前最优的模型，77%的准确率是聚类模型、数据、降维方法共同作用的结果。
2. mykmeans方法在测试集预训练集上都有比较好的效果。在测试集上的结果还要优于在训练集上的结果，说明算法具有一定的泛化性能。相比于kmeans，算法在kmeans基础上有接近10%的提升，而时间开销也仅增加了6秒左右。

考虑mykmeans的优化方法：直接在总样本上采样，结果如下

mykmeans 训练集上的聚类精度：66.035%
mykmeans 测试集上的聚类精度：66.230%
train time: 9.137s



可以看到，因为更新时的算法复杂度从$O(n^2)$变到了$O(n)$，所以模型运行时间也大大减小，相比于kmeans算法只增加了零点几秒的开销，但是准确率却仍然能够保持在66.23%.
进而，从这份实验结果可以看到，mykmeans在kmeans的基础上，通过增加噪声数据，实现了模型性能10%左右的提高，而这种优化方法所带来的时延开销相比于训练过程是微乎其微的。这从实际层面证明了优化算法的有效性。

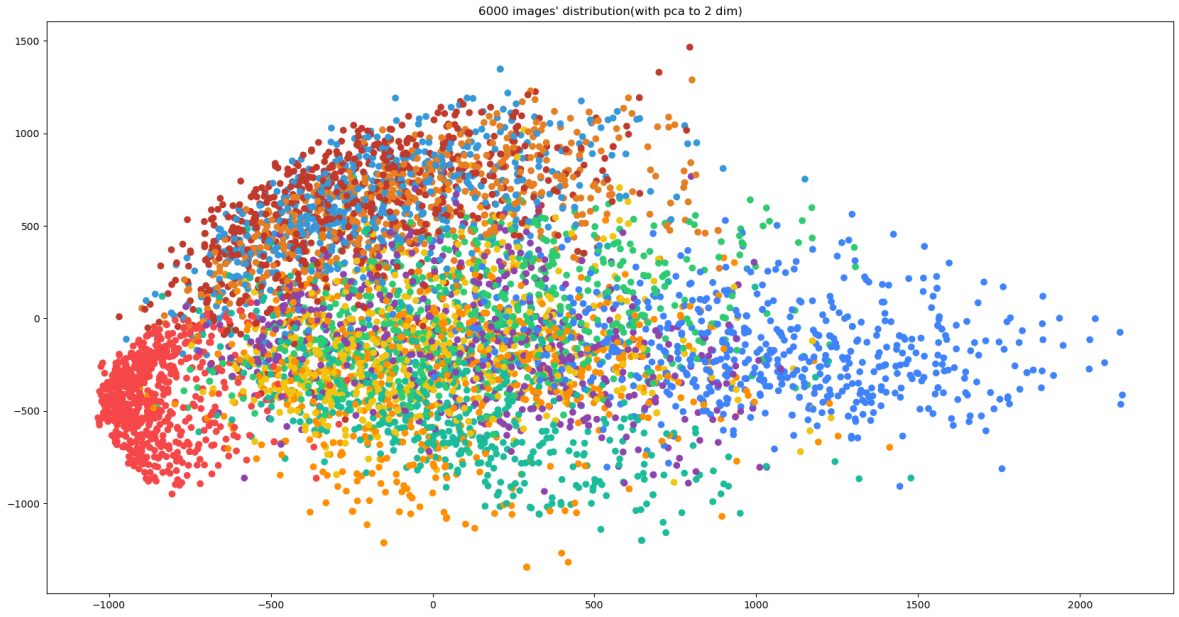
五、结论

在本次实验过程中，我针对hw3的模型继续优化，通过关注kmeans算法的 过拟合 问题，提出了在更新kmeans聚类中心点时加入部分噪声数据的优化方法，以提高模型的泛化性能。最终实验结果表明，这种方法能够较好提升聚类性能，并且所需的额外时间开销非常少，是一种比较成功的优化算法。
当然，这种方法也存在一定缺陷，那便是对 `self.p` 以及 `self.lr` 的设定问题。在本次实验中，我通过不

断尝试找到了较好的一组参数。虽然不能证明参数的泛化性，但是在运行实验时我发现，只要将参数控制在合理范围内，都能对模型有一定的提升，基本不会有负优化的问题。

回到报告最开始的问题，也就是GMM与kmeans的性能差距问题。如果说我们现在用优化方法缓解了二者中心点的差距，那么还有哪些因素影响着二者的性能呢?
我想，还是对数据的分析能力的差异。GMM可以对不同维度的数据进行分析(通过协方差矩阵等方式)，而kmeans只能对每一维度的数据单独分析(作差，求平方)。
并且GMM还是一种Soft K-Means，或者说模糊 K-Means，在更新样本点时，其会考虑每个样本点属于这一类别的概率，根据其进行加权平均，这一点也与我们的kmeans方法不同。

如果继续追问，如何让聚类的准确率达到100%呢，又或者说影响聚类性能的核心要素是什么呢?



我想这一问题还是要回到对数据的表达上，无论是使用核kmeans还是预训练样本数据，本质上都是为了改变数据的分布。如果训练数据在高维空间上也如上图般混杂的话，那么再好的聚类算法，也无法发挥作用。如何让模型认识数据、理解数据，我想这应该成为我未来学习过程中更加关注的问题。