MNIST Solution

> 机器学习概论lab3
>
> **Author:**@Rosykunai
>
> **Date:**2024 年 11 月

MNIST Solution

1. GMM聚类

1.1 E-step

```python
def _e_step(self, X: np.ndarray)-> np.ndarray:
    """
    E-step: Compute the responsibilities.

    Args:
        - X: np.ndarray, shape (N, D), Data.

    Returns:
        - gamma: np.ndarray, shape (N, K), Responsibilities.
    """
    N, D = X.shape
    gamma = np.zeros((N, self.n_components))

    # Precompute determinants and inverses for each covariance matrix
    dets = np.array([np.linalg.det(cov) for cov in self.covs])
    inv_covs = np.array([np.linalg.inv(cov) for cov in self.covs])

    for k in range(self.n_components):
        gamma[:, k] = self.pi[k] * self._gaussian(X, self.means[k], inv_covs[k], dets[k])
```

```
    # Normalize responsibilities
    gamma_sum = np.sum(gamma, axis=1, keepdims=True)
    gamma /= gamma_sum
    return gamma
```

1.2 M-step

```
def _m_step(self, X: np.ndarray, gamma: np.ndarray):
    """
    M-step: Update the parameters.

    Args:
        - X: np.ndarray, shape (N, D), Data.
        - gamma: np.ndarray, shape (N, K), Responsibilities.
    """
    N, D = X.shape
    n_soft = np.sum(gamma, axis=0)  # [K,]

    # Update mixing coefficients
    self.pi = n_soft / N

    # Update means
    self.means = (gamma.T @ X) / n_soft[:, None]

    # Update covariance matrices
    for k in range(self.n_components):
        X_centered = X - self.means[k]
        gamma_diag = np.expand_dims(gamma[:, k], axis=1)
        self.covs[k] = (X_centered.T @ (gamma_diag * X_centered)) /
n_soft[k] + 1e-6 * np.eye(D)
```

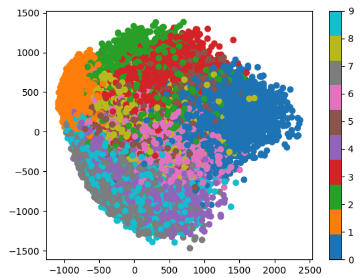
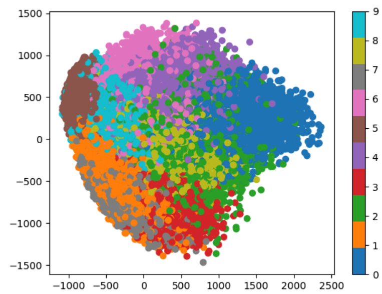
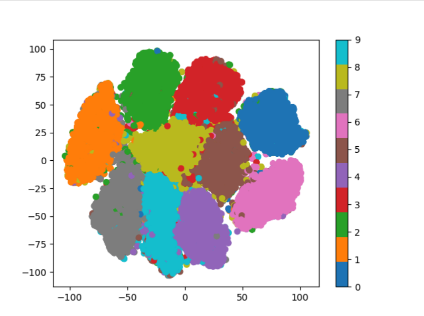
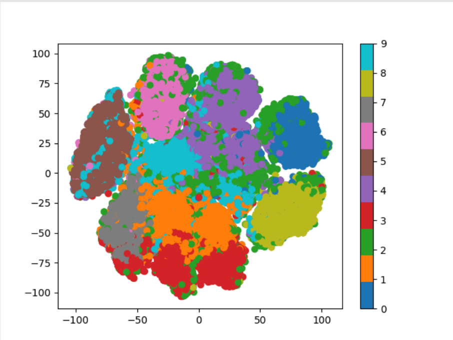
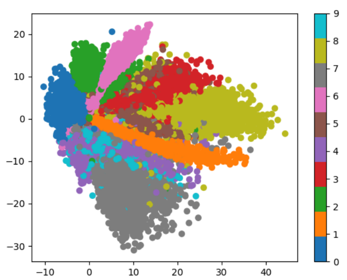
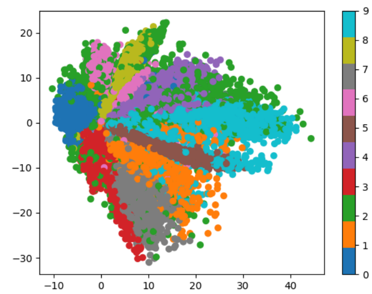
2. PCA降维

2.1 计算主成分

```
def fit(self, X: np.ndarray):
    """
    Fit the PCA model to the data.

    Args:
        - X: np.ndarray, shape (N, D), Data.
    """
    self.mean = np.mean(X, axis=0)
```

```
        X = X - self.mean
        cov = np.cov(X.T)
        eigenvalues, eigenvectors = np.linalg.eigh(cov)
        eigenvectors = eigenvectors.T
        idxs = np.argsort(eigenvalues)[::-1]
        eigenvectors = eigenvectors[idxs]
        self.components = eigenvectors[0:self.dim]
```

4. 比较不同的降维方法

	True	Cluster
PCA		
tSNE		
AutoEncoder		

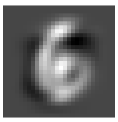
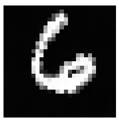
5. 作为生成模型的GMM

5.1 从GMM中采样

```python
def sample_from_gmm(gmm: GMM, pca: PCA, label: int, path: Union[str , Path]):
    """
    Sample images from a Gaussian Mixture Model.

    Args:
        - gmm: GMM, Gaussian Mixture Model.
        - pca: PCA, Principal Component Analysis.
        - label: int, Cluster label.
    """
    # Sample from the Gaussian component
    mean = gmm.means[label]
    cov = gmm.covs[label]
    sample = np.random.multivariate_normal(mean, cov, 1)

    # Project the samples back to the original space
    sample = pca.inverse_transform(sample)

    # Rescale the samples to [0, 255]
    sample = (sample - np.min(sample)) / (np.max(sample) - np.min(sample)) * 255

    # Save an example image(you can change this part of the code if you want)
    sample = Image.fromarray(sample[0], mode='L') # if sample shape is (1, 28, 28)
    path = Path(path)
    sample.save(path / 'gmm_sample.png')
```

*此处直接裁剪到[0, 255]效果更好

6. 测试

GMM	DDPM
	

7. 回答问题

7.1 比较降维方法

	PCA	tSNE	AutoEncoder
训练速度	快($O(nd^2)$)	慢	慢
降维效率	高	低	低
灵活性	低	中	高
对数据分布的保持程度	低	中	高
可视化效果	差	好	好

*基于QR分解的PCA有$O(nd^2)$的时间复杂度. 在本次实验中你已经发现PCA比tSNE快许多,但是当维度很大时PCA并不是一个很好的降维方法, 感兴趣的同学可以在下学习选修《大数据算法》这门课进一步学习降维(Johnson-Lindenstrauss引理).

7.2 比较生成模型

	GMM	DDPM
生成效率	高	低
生成质量	低	高
灵活性	低	高
是否可控	是,但不如DDPM	是

*本次实验中我们在DDPM的输入张量上拼接了一块表示condition的张量, 这并不是标准的做法. 感兴趣的同学可以自行搜索Classifier Guidance和Clssifier-free Guidance.