Kernel density estimation with Mixture of Gaussians

1 Background

In unsupervised learning, traditionally known as density modeling, one usually constructs a probabilistic model $p(x)$. The fitting of the model is performed on a training set, and its generalization performance evaluated on a separate test set. The hyper-parameters of the model are typically tuned on a validation set. As opposed to supervised learning, unsupervised learning is arguably more challenging as $p(x)$ is typically much more complicated than $p(y|x)$.

Among many possible choices of $p(x)$, one of the simplest is the well-known good-and-old-fashioned "kernal density estimator". It is non-parametric in the sense that $p(x)$ "memorizes" the entire training set. The scoring function is usually defined by a Gaussian kernel. This work borrows such a basic idea from the standard kernel density estimator and formulates it with a mixture of Gaussian distributions.

2 Model

Given a dataset that contains two splits $\mathcal{D}_A \in \mathcal{R}^{k\times d}$ and $\mathcal{D}_B \in \mathcal{R}^{m\times d}$, compute the log-likelihood of $\mathcal{D}_B$ under $\mathcal{D}_A$ with the following probability density function

$$\log p(x) = \log \sum_{i=1}^{k} p(z_i)p(x|z_i) \tag{1}$$

where $x \in \mathcal{R}^d$ and z_i is discrete.

The above formulation assumes the probability of x in terms of a mixture of conditional distributions. Here we call $p(z_i)$ the probability of its i-th mixing component, and $p(x|z_i)$ the probability of x under the i-th component.

To simplify even more, let us further assume the following

$$p(z_i) = \frac{1}{k} \tag{2}$$

and

$$p(x|z_i) = \prod_{j=1}^{d} p(x_j|z_i) \tag{3}$$

where

$$p(x_j|z_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp(-\frac{(x_j - \mu_{i,j})^2}{2\sigma_i^2}) \tag{4}$$

and $\mu \in \mathcal{R}^{k \times d}$. To simplify further, we also assume that all $p(.|z_i)$ Gaussian components share the same σ. Therefore Equ. (1) can be written as

$$\log p(x) = \log \sum_{i=1}^{k} \exp\{\log \frac{1}{k} + \sum_{j=1}^{d} [-\frac{(x_j - \mu_{i,j})^2}{2\sigma^2} - \frac{1}{2}\log(2\pi\sigma^2)]\} \tag{5}$$

With Equ. (5), one can compute for each example in $\mathcal{D}_B$ its log-probability by considering all k examples in $\mathcal{D}_A$ with $\mu_{i,j} \equiv x_{i,j}^A$ where $x^A \equiv \mathcal{D}_A \in \mathcal{R}^{k \times d}$. Finally the mean of the log-probability on $\mathcal{D}_B$ can be written as

$$\mathcal{L}_{\mathcal{D}_B} = \frac{1}{m} \log \prod_{i=1}^{m} p(x_i^B) = \frac{1}{m} \sum_{i=1}^{m} \log p(x_i^B) \tag{6}$$

3 Datasets

We suggest building such a model on both MNIST[1] and CIFAR100[2]. MNIST contains grayscale images of size 28 by 28 while CIFAR100 contains RGB images of size 32 by 32.

[1] `http://www.iro.umontreal.ca/~lisa/deep/data/mnist/mnist.pkl.gz`
[2] `https://www.cs.toronto.edu/~kriz/cifar.html`

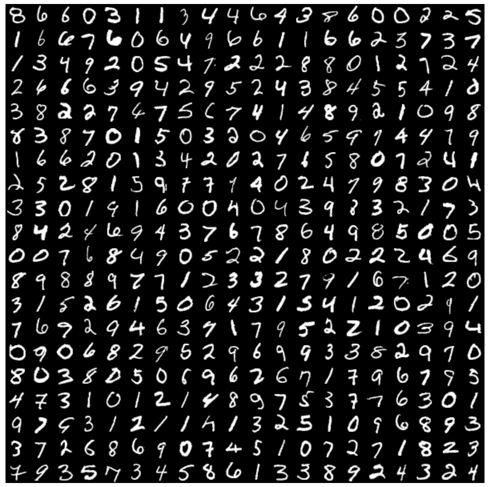

Figure 1: MNIST

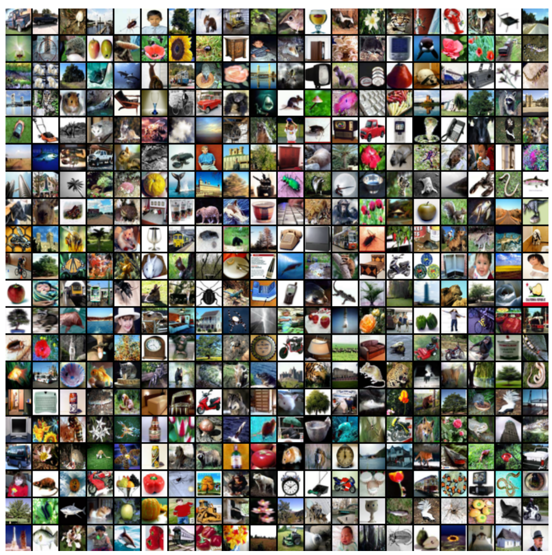

Figure 2: CIFAR100

4 Submission

- You are required to code in Python + Numpy environment. You may only use Python and Numpy standard libraries and you may not use any existing frameworks such as scikit-learn, Theano, Torch, or TensorFlow.

You may use additional tools only for visualization, such as PIL and Matplotlib.

- Preprocessing two datasets. Please split both datasets into training, validation and test sets with the following instructions
  - Scale pixel values between 0 and 1, for example, divided by 255 if the original pixel values are discrete integers between 0 and 255.
  - Shuffle the original training set, use the first 10K as the new training set, the second 10K as the new validation set, discarding the rest.
  - Use the original 10K test set as it is.
  - Verify the correctness of preprocessing by visualizing both datasets similar as those shown in Section 3.
  - Provide the visualization codes.
- Provide the correct CPU-based Python codes that compute the mean of the log-probability of examples provided in Equ. (6).
  - The code should be run on CPUs within a standard Linux/Unix environment. All the results should be reproducible. We do not require codes that run on GPUs.
  - Please take time to optimize the efficiency of your code by minimizing the number of "for loops".
  - Grid-search for the optimal value of σ. Experiment with $\sigma = \{0.05, 0.08, 0.1, 0.2, 0.5, 1., 1.5, 2\}$ on both MNIST and CIFAR100 with ($\mathcal{D}_A = \text{MNIST}_{\text{train}}$, $\mathcal{D}_B = \text{MNIST}_{\text{valid}}$) and ($\mathcal{D}_A = \text{CIFAR}_{\text{train}}$, $\mathcal{D}_B = \text{CIFAR}_{\text{valid}}$), report your results either in table or figure formats.
  - With the optimal σ, compute $\mathcal{L}_{\mathcal{D}_B}$ where ($\mathcal{D}_A = \text{MNIST}_{\text{train}}$, $\mathcal{D}_B = \text{MNIST}_{\text{test}}$). Same for CIFAR100. Benchmark the running time for both datasets.
  - Explain the trend observed while increasing σ
- Include the experimental results in a report in pdf format. Zip both the report and the codes. Double check your results are reproducible as we will be executing your submitted codes.

- Optional: You may provide some insights of the pros and cons of using such a model on those datasets.

5 Evaluation

This exercise evaluates the ability to

- prepare standard machine learning dataset
- understand the math that underlies the proposed model
- code in python + numpy environment
- correctness of your code in terms of free of bugs
- efficiency of your code in terms of execution time on CPU
- the ability to convey your experimental findings in a written format