Kernel Density Estimation with Mixture of Gaussians

by

Hongming Zhang

Kernel density estimation (KDE) (A.K.A. Parzen-Rosenblatt window) is a non-parametric way to estimate the probability density function of a random variable, where inferences about the population are made according to a finite data sample.

In this work, a Gaussian kernel is used to fit the models from the MNIST and CIFAR-100 datasets. The results of the work contain two parts, as shown below.

A. Data Preprocessing and Visualization

The details of the MNIST and CIFAR-100 datasets are summarized as follows:

MNIST:	28×28 grayscale images
CIFAR-100:	$32 \times 32 \times 3$ RGB images

The visualization code is given in ”main-visu.py”, where,

- the original training set is randomly shuffled;
- the first 10^4 shuffled data is used as the new training set;
- the second 10^4 shuffled data is used as the new validation set;
- the original 10^4 test set remains unchanged.

Finally, the selected visualized datasets of MNIST and CIFAR-100, each of which is shown in a 20×20 grid structure, are given in Fig. 1 and Fig. 2, respectively.

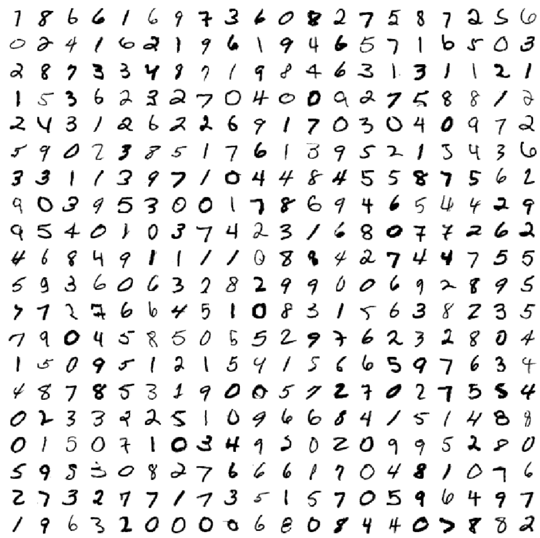

Fig. 1: MNIST

Fig. 2: CIFAR-100

B. KDE

First, Eq.(6) can be simplified as

$$\begin{aligned}
\mathcal{L}_{\mathcal{D}_B} =& \frac{1}{m}\sum_{i=1}^{m}\log p(x_i^B) \\
=& -\log k - \frac{d}{2}\log 2\pi h + \frac{1}{m}\sum_{i=1}^{m}\log\sum_{i=1}^{k}\exp\left\{\sum_{j=1}^{d} -\frac{(x_j-\mu_{i,j})^2}{2h}\right\} \qquad \text{(A)} \\
\approx& -\log k - \frac{d}{2}\log 2\pi h + \frac{1}{m}\sum_{i=1}^{m}\max\left\{\sum_{j=1}^{d} -\frac{(x_j-\mu_{i,j})^2}{2h}\right\}, \qquad \text{(B)}
\end{aligned}$$

where,

$h = \sigma^2$ denotes the smoothing bandwidth;

Eq.(A) is obtained according to the log-sum-exp trick to avoid numerical underflow;

Eq.(B) is based on the approximation of $\log\sum_i \exp x_i \approx \max_i\{x_i\}$ for simplifying the calculation.

The KDE algorithm is shown in KDE-Gauss.py. Specifically, the term $\sum\limits_{j=1}^{d} -\frac{(x_j-\mu_{i,j})^2}{2h}$ in Eq.(B) is calculated with the aid of broadcast in numpy. As a result, a single for-loop is used in our algorithm for speeding up the code. Note that, further improvements can be done by using the GPU based parallel processing.

By running KDE-Gauss.py code, experimental results are obtained for choosing the optimal σ from a grid-search of $\{0.05, 0.08, 0.1, 0.2, 0.5, 1.0, 1.5, 2.0\}$ on both MNIST and CIFAR-100. The corresponding results are shown in Fig. 3 and Table I.

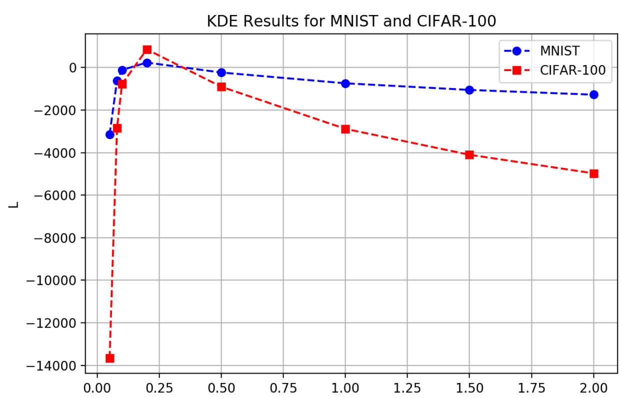

Fig. 3: KDE Results for MNIST and CIFAR-100

TABLE I: KDE Results for MNIST and CIFAR-100

σ	$\mathcal{D}_{\text{valid}}^{\text{MNIST}}$	$\mathcal{D}_{\text{valid}}^{\text{CIFAR}}$
0.05	-3149.5312	-13659.3198
0.08	-612.1928	-2845.7477
0.10	-116.5632	-776.1584
0.20	234.1082	854.1262
0.50	-233.9160	-903.2076
1.00	-741.5795	-2882.2616
1.50	-1052.8412	-4100.1511
2.00	-1276.0659	-4974.0981

As seen in Fig. 3 and the red colored data shown in Table I, $\sigma = 0.2$ is the optimal value in the set of $\{0.05, 0.08, 0.1, 0.2, 0.5, 1.0, 1.5, 2.0\}$ that maximize the mean log-likelihood for both MNIST and CIFAR.

To evaluate the runtime of the codes, the system configuration is shown below

Processor:	2 GHz Intel Core i5
Memory:	8 HB 1867 MHz LPDDR3
Python version:	Python 3.6.5

Finally, the average runtime for KDE of each σ on MNIST and CIFAR-100 is 146.9135s and 2056.9747s, respectively.

C. Pros and Cons

Pros:

As a non-parametric method, it does not require the a priori knowledge of the model.

Cons:

Generalization ability maybe not high enough.