KERNEL DENSITY ESTIMATION WITH GAUSSIANS

Christina Kouridi
christinakouride@gmail.com

March 23, 2021

ABSTRACT

Estimating the probability density of a random variable is challenging. Many existing approaches are computationally intractable for complex datasets, or require assumptions about the parametric form of the underlying density function. In this report, we study Kernel Density Estimation with a Gaussian kernel, a classic non-parametric approach that infers the population density based on a finite data sample. We evaluate it on two famous image datasets – MNIST and CIFAR-100.

1 Introduction

Kernel density estimation (KDE), also referred to as the Parzen–Rosenblatt window [1, 2]), is a classic technique for estimating the underlying probability density function of a dataset. It is non-parametric, meaning that it does not assume that the underlying density function is from a parametric family, giving it flexibility to estimate a range of complicated distributions. It constructs an estimate of $p(x)$ for a point x, by placing a kernel at each point in the dataset and averaging the kernel function across all points.

This report constitutes an empirical evaluation of KDE with a Gaussian Kernel, based on our own implementing in Python and Numpy. It is structured as follows. Section 2 provides an overview of the Gaussian KDE model, and Section 3 presents our experiments and results. Finally, Section 4 includes a brief discussion on the experimental results, and the advantages and disadvantages of our approach to density estimation.

2 Model

To construct a probabilistic model *p(x)* of the data *x*, the kernel estimation model is fit on a training set $\mathcal{D}_A$ while its generalization performance is evaluated on a separate set $\mathcal{D}_B$.

Given these data splits $\mathcal{D}_A \in \mathbb{R}^{k\times d}$ and $\mathcal{D}_B \in \mathbb{R}^{m\times d}$, the log-likelihood of a point x in $\mathcal{D}_B$ under $\mathcal{D}_A$ is given by:

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

where $x \in \mathbb{R}^d$ and $p\left(z_i\right)$ is discrete. This formulation assumes that the probability of x can be written in terms of a mixture of conditional distributions over the dataset $\mathcal{D}_A$. Here $p\left(z_i\right)$ is the probability of the i^{th} component and $p\left(x|z_i\right)$ is the probability of x under the i^{th} component. To simplify the expression, let us assume that:

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

and

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

where

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

and $\mu \in \mathbb{R}^{k\times d}$. Finally, assume that all Gaussian components $p\left(\cdot|z_i\right)$ share the same σ. This allows us to re-write the log-likelihood as follows:

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

Using equation 5 we can compute for each instance in $\mathcal{D}_B$ its log-probability under the model trained using $\mathcal{D}_A$ by considering all k instances in $\mathcal{D}_A$ with $\mu_{i,j} = x_{i,j}^A$ where $x^A = \mathcal{D}_A \in \mathbb{R}^{k\times d}$ Finally, the mean log-probability of $\mathcal{D}_B$ can be written as:

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

The formulation of equation 5 is susceptible to numerical issues for moderately small and large values of σ. This is because the exponent of the exponential results in overflow or underflow of the exponential term. To prevent such numerical issues, we use the LogSumExp (LSE) trick[1] which provides a tractable formulation for the logarithm of a sum of exponential terms:

$$\log\sum_{i=1}^{k} e^{y_i} = \log\sum_{i=1}^{k} \mathrm{e}^{c} e^{y_i - c} = \log\left(e^{c}\sum_{i=1}^{k} e^{y_i - c}\right) = c + \log\sum_{i=1}^{k} e^{y_i - c} \tag{6}$$

where c is typically chosen to be:

$$c = \max_i y_i \tag{7}$$

which shifts values in the exponential exponent such that its greatest value is forced to zero.

[1] https://en.wikipedia.org/wiki/LogSumExp

3 Numerical studies

3.1 Data

The numerical studies that follow are conducted on the *MNIST*[2] and *CIFAR-100*[3] datasets. To ensure that we have loaded and processed the data correctly, we visualise a random sample of 64 images from the final datasets.

Figure 1: sample *MNIST* and *CIFAR-100* images

3.2 Hyperparameter optimisation

The standard deviation σ is treated as a hyperparameter, tuned on the validation set ($\mathcal{D}_{\text{valid}}$). We use grid-search to find the optimal $\sigma \in \{0.05, 0.08, 0.1, 0.2, 0.5, 1.0, 1.5, 2.0\}$.

To address deviations in performance introduced by the chosen train-validation data split, we select the optimal σ based on the average mean log-probability across 5 seeds $\in \{0, 10, 20, 30, 40\}$. The averaged results for *MNIST* and *CIFAR100* are summarised in Table 1. For both datasets, the maximum average mean log-probability of $\mathcal{D}_{\text{valid}}$ occurs for $\sigma = 0.2$.

σ	$\mathcal{L}_{\mathcal{D}_{\text{valid}}}^{\text{MNIST}}$			$\mathcal{L}_{\mathcal{D}_{\text{valid}}}^{\text{CIFAR100}}$		
0.05	-3130.8	±	13.7	-13612	±	48.8
0.08	-604.86	±	5.37	-2878.7	±	19.0
0.10	-111.87	±	3.44	-754.20	±	12.2
0.20	**235.30**	±	**0.86**	**863.13**	±	**3.05**
0.50	-233.57	±	0.14	-902.60	±	0.49
1.00	-740.82	±	0.04	-2881.9	±	0.12
1.50	-1051.1	±	0.02	-4099.2	±	0.06
2.00	-1272.9	±	0.02	-4972.6	±	0.03

Table 1: Hyperparameter optimisation results for MNIST and CIFAR100: average and standard deviation of the mean log-probability of the validation set across 5 random seeds

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

3.3 Generalisation performance

With the optimal σ=0.2 on the validation set, we compute the mean log-probability of the test set for MNIST and CIFAR100. The results, along with the running time, are summarised in Table 2.

$\mathcal{L}_{\mathcal{D}_{\text{test}}}^{\text{MNIST}}$	$t_{\text{test}}^{\text{MNIST}}$	$\mathcal{L}_{\mathcal{D}_{\text{test}}}^{\text{CIFAR100}}$	$t_{\text{test}}^{\text{CIFAR100}}$
234.19 $\pm$ 0.53	132.94s $\pm$ 2.27s	846.17 $\pm$ 2.52	1118.04s $\pm$ 2.24s

Table 2: Generalisation performance on MNIST and CIFAR100.
Results are averaged across 5 seeds.

Due to the reproducibility requirement of this exercise, we also report the results for seed=0 on Table 3 in Appendix A.

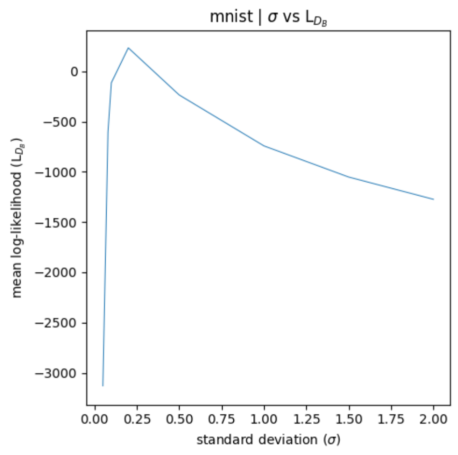
The model is run using Python 3.8 and Numpy 1.19 on an AMD Ryzen 7 3700X 8-core processor. Running time is improved by utilising *Numba*, a just-in-time compiler for Python which compiles a subset of the language into efficient machine code that is comparable in performance to a traditional compiled language [3]. It also enables execution of loops in parallel on separate threads (using *Numba's* prange() instead of range()). *Numba* is compatible with *NumPy*[4], and requires minimal code changes.

4 Discussion

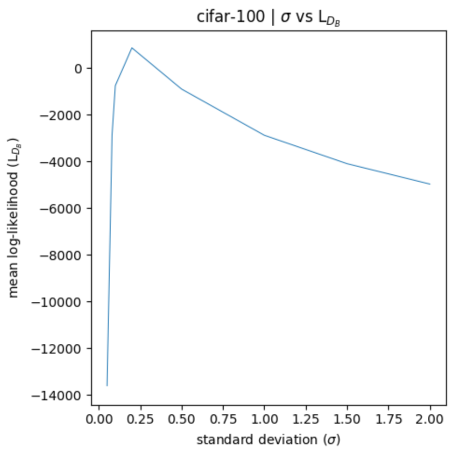
4.1 Standard deviation

To clearly observe the dependency of the mean log-probability on σ, we plot the mean log-probability on the validation set (results of Table 1) on Figure 2. As σ increases, the mean log-probability increases steeply, peaking around $\sigma = 0.2$. Beyond the peak, it decreases with a lower rate due to saturation of the smoothing effect of spreading out the Gaussian kernel.

Intuitively, σ controls the shape of the Gaussian kernel at each data point x_{ij}. It therefore has a smoothing effect; as it increases, the density at that point is more spread out, thereby contributing to the density estimate of surrounding regions. Thus if σ get large, the resulting probability density would be oversmoothed. In the other extreme, as $\sigma \to 0$, the estimate would be non-smooth, as it would be a sum of delta functions.



(a)



(b)

Figure 2: Hyperparameter optimisation results for MNIST and CIFAR100:
average of mean log-probability of the validation set across five random seeds

The same trend is observed for both datasets, perhaps due to the similar support of the image samples.

[4]https://numba.pydata.org/numba-doc/dev/reference/numpysupported.html

4.2 Gaussian KDE

Gaussian KDE is an effective method for computing smooth and continuous probability density estimates. Despite its strong representational power, it is not suitable for large-scale problems due to asymptotic time complexity scaling with the size and dimensionality of the data. The significantly higher running time when switching from gray-scale MNIST images (d=784) to RGB *CIFAR100* images (d=3072) is therefore unsurprising. Existing libraries (e.g. Scipy's KDTree[5] and SKLearn's KDTree[6]) speed up KDE for large datasets by introducing efficient neighbor searching methods.

Another limitation of Gaussian KDE is its sensitivity on the scale parameter σ. Selecting an appropriate value should be done carefully, as σ in the vicinity of the optimal value can still produce unreasonable estimates. This issue would be exacerbated if we did not assume that σ is constant for every Gaussian component $p(x_j|z_i)$. Variable σ would allow the KDE to fit local density better, providing a high resolution estimate in parts of the distribution where data density is sparse and very smooth.

A Appendix

$\mathcal{L}_{\mathcal{D}_{\text{test}}^{\text{MNIST}}}$	$t_{\text{test}}^{\text{MNIST}}$	$\mathcal{L}_{\mathcal{D}_{\text{test}}^{\text{CIFAR100}}}$	$t_{\text{test}}^{\text{CIFAR100}}$
234.78	136.77s	844.88	1123.23s

Table 3: Generalisation performance on MNIST and CIFAR100:
mean log-probability of the test set for seed=0, σ=2

References

[1] Emanuel Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065 – 1076, 1962.

[2] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, 27:832–837, 1956.

[3] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, New York, NY, USA, 2015. Association for Computing Machinery.

[5] https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.spatial.KDTree.html

[6] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html