Optimizing Kernel Density Estimator in Python

Reynaldo Gil

July 26, 2019

Abstract

The work show the results of creating and optimizing python code for computing a Kernel Density Estimator over objects of high dimensionality. The main result is an optimized library that take advantages of the vector operation and multiprocessing capabilities of modern computers. Besides the application of this method to images was studied.

1 Introduction

Kernel Density Estimation (KDE) is a non-parametric way to estimate the probability density function of a random variable ([3]). It is very related to histograms because using it smoothing histograms can be obtained. The technique allows in general to approximate a function in any point as the sum of kernel functions collocated in the points of known data. The kernel functions have parameters that can be optimized using a learning process.

In this work the kernel selected is a Gaussian and the data are images from the standard databases Minist ([5]) and Cifar ([2]).

2 Mathematical model

Given a collection of k points $z_i \in R^d$ the log-likelihood of a point $x \in R^d$ can be expressed :

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

Assuming that $p(z_i) = \frac{1}{k}$ and $p(x|z_i)$ is described by the product of Gaussian components with the same σ for each dimension:

$$p(x_j|z_i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_j - z_{ij})^2}{2\sigma^2}}$$

Then:

$$\log p(x) = \log \sum_{i=1}^{k} \frac{1}{k(\sigma\sqrt{2\pi})^d} \prod_{j=1}^{d} e^{-\frac{(x_j - z_{ij})^2}{2\sigma^2}}$$

Applying the exponential and logarithms rules this can be expressed as:

$$\log p(x) = -\log k - d\log\sigma - \frac{d}{2}\log 2\pi + \log \sum_{i=1}^{k} e^{-\sum_{j=1}^{d} \frac{(x - z_{ij})^2}{2\sigma^2}}$$

$$\log p(x) = -\log k - d\log\sigma - \frac{d}{2}\log 2\pi + \log \sum_{i=1}^{k} e^{-\frac{|x - z_i|^2}{2\sigma^2}}$$

This is equivalent to:

$$\log p(x) = \log \sum_{i=1}^{k} e^{\log\frac{1}{k} - \sum_{j=1}^{d} \left\{ \frac{(x_j - z_{ij})^2}{2\sigma^2} + \frac{1}{2}\log 2\pi\sigma^2 \right\}}$$

Unless both formulas produce the same results, in the work is used the first one. On it the computation that depends of the data vectors is small and separated of the computation that only depends on parameters and sizes, that need to be computed only one time. So, it is a bit easier to optimize the code in this way.

Finally, for validate the result the average over another data is computed:

$$Q = \frac{1}{m} \sum_{i=1}^{m} \log p(x_i)$$

2.1 Special formula

One problem we found was the computation of $\log\sum_{i=1}^{k} e^{-\frac{|x-z_i|^2}{2\sigma^2}}$ because when σ is small and the vector have high dimensionality the exponential return zero due the value be so large and the limited resolution of the fixed point arithmetic. One solution can be use other arithmetic, but they are so slow. The problem can be solved more efficiently using LogSumExp ([4]) This function set:

$$\log\sum e^{x_i} = \max(x_i) + \log\sum e^{x_i - \max(x_i)}$$

Using it, the exponents are not big and the resolution problem doesn't happens.

3 Implementation

The implementation consist of one module named kde and the corresponding test module. There are 3 main programs: preprocessing.py that create the requested data from the original, fit.py that find the best σ and apply.py that run the best σ found over the testing collection. Besides there are two version of the kde implementation using both formulas. In the root directory is a Readme.md file that explain the steps for run the programs.

The interpreted and dynamic computation of python allow an easy programming but the execution is very inefficient because every object use a lot of additional memory and need to check its type in execution time among another problems. These problem increase with the data size. It is needed to avoid computation in Python to achieve a better performance. Due to this problem, a lot of libraries implemented in C or other languages have been created. This libraries do the hard work and python works as the glue between then. In particular numpy is the basic library for scientific computing. It implements vector operations that can be optimized for some architectures.

The first optimization was implement all the operations that where possible using numpy functions over arrays. This avoid the overload python execution and can take advantage of the vector operation.

The second was paralyzing the code at a higher functional level. This algorithm offer several levels of parallelism, in particular the terms of the final evaluating function can be computed independently. For compute the validation or testing, the matrix is divided among the available processors using the multiprocessing library.

4 Results

First, the best value of σ for each data was found. The result are:

	Minist	Cifar
σ	Q	Q
0.05	-3208	-2489
0.08	-635	-554
0.1	-131	5
0.2	230	347
0.5	-234	-296
1.0	-741	-962
1.5	-1051	-1368
2.0	-1272	-1658

As can be see the better fit for both data is obtained when $\sigma = 0.2$ and the fits improve gradually while are more near the best value. One thing I do not understand is why positive values are obtained. If we are computing log of probabilities the results must be negatives. I implemented both formula versions and the results were the same. Unless I have doubt about the absolute values I thing that the trend is clear. When we get the optimized value of σ is when the kernel density estimation better adjust with the validation data.

The experiments were running in an i7-8750 CPU with 32 Gb and Linux. Numpy was using openblas library. The times for Minist were around 2 minutes for each σ value. In Cifar were 5 minutes. These is an expected behavior because Mnist images are blanck and white with a lot of ceros while Cifar is created by averaging color images and have a few zeros.

5 KDE on images

The kernel density estimation is a way to learn and reproduce some function, in particular in gray images can be used to learn the intensity on each point. For me have more sense if the points are considered in R^2 so the Gaussian in a point has more influence in its R^2 neighbors. For example [1] uses KDE in the spatial domain for image segmentation.

Also it must be used to learn one class, for example in Mnist learn each number, because if not I don't understand what it is learned. With the corresponding estimation we can for example create an smoothing image of the average number.

References

[1] O. Pereira. Edge detection based on kernel density estimation. 2015.

[2] Wikipedia. Cifar-10.

[3] Wikipedia. Kernel density estimation.

[4] Wikipedia. Logsumexp.

[5] Wikipedia. Mnist database.