Optimizing of Kernel Density Estimator in Python

Reynaldo Gil

July 26, 2019

Abstract

The work shows 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 takes 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 much related to histograms because using it, smoothing histograms can be obtained. In general, the technique allows to approximating a function at 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 Mnist ([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 on the data vectors is small and separated from the computation that only depends on parameters and sizes, that need to be computed only one time. Therefore, 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 $\log \sum_{i=1}^{k} e^{-\frac{|x-z_i|^2}{2\sigma^2}}$. When σ is small and the vector have high dimensionality the exponent value is so much large. This and the limited resolution of the fixed point arithmetic cause the exponential return zero. One solution can be using a more resolution arithmetic, but they are slower. Fortunately, 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 consists of one module named kde and the corresponding test module. There are 3 main programs: preprocessing.py that creates the requested data from the original, fit.py that finds the best σ and apply.py that runs the best σ found over the testing collection. Besides, there are two versions of the kde implementation using both formulas. Using it we can check that they coincide. In the root directory it is a Readme.md file that explains the steps to run the programs.

The interpreted and dynamic computation of python allows easy programming. However, the execution is very inefficient because every object uses a lot of additional memory and it is needed to check its type in execution time, among other problems. These problems increase with the data size. It is needed to avoid computation in Python to achieve better performance. Due to these problems, a lot of libraries implemented in C or other languages have been created. These libraries do the hard work and python works as the glue between them. 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 implementing 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. For example the computation of $\log \sum_{i=1}^{k} e^{-\frac{|x-z_i|^2}{2\sigma^2}}$ was done using differences, maximum,

	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

Table 1: Quality again σ

norm, squares and division by scalar from numpy. One operation that is a lost of efficiency is compute the square of the norm because we needed to compute the norm and the squares but we did not found a way to avoid it. So, if this work need to be improved this is one point to a better implementation. Another improve that be done is the computation of the mapping, for example for map from the validation vector to logp(x). That was implemented with a normal python call and reduce. This can be done in numpy but it is needed to create an ufunc, so with more time can be done too.

The second was parallelizing the code at a higher functional level. This algorithm has several levels of parallelism. In particular the terms of the sum in the quality function can be computed independently. For computing the validation or testing, the matrix is divided among the available processors using the multiprocessing library. Using these parallelism level was easy and achieve a high reduction of times.

4 Results

First, the best value of σ for each data was found. The result are shown in Table 1:

As can be seen, the better fit for both data is obtained when $\sigma = 0.2$ and the quality improve while are more near the best value. There is one problem, 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 think

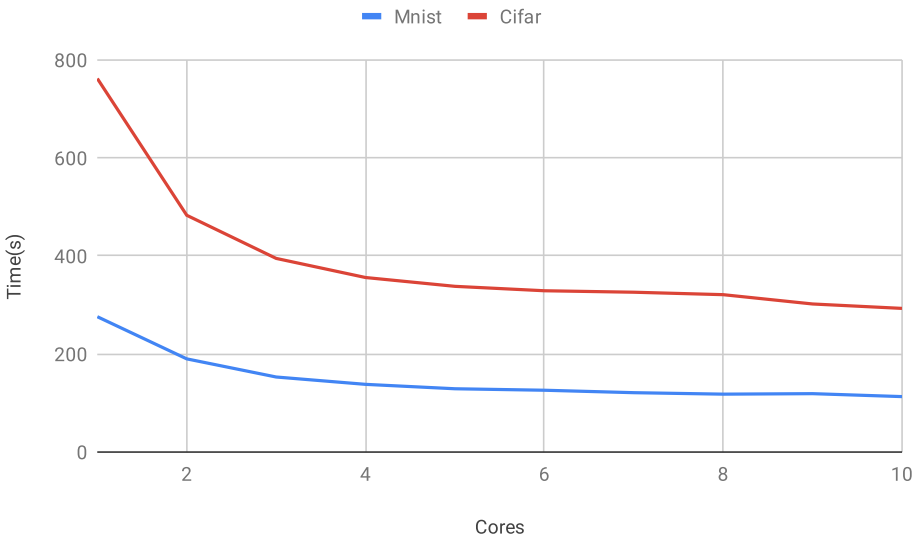

Figure 1: Time again cores

that the trend is clear. The optimized value of σ is obtained when the kernel density estimation better adjusts with the validation data.

The experiments were running in an i7-8750 CPU with 6 cores, hyper-threading and 32 Gb and Linux. Numpy was using openblas library. The times for Minist were around 2 minutes for each σ value with 8 cores. In Cifar were about 5 minutes. These is an expected behavior because Mnist images are black and white with a lot of ceros while Cifar is created by averaging color images and has a few zeros. Figure 1 shows the times for testing on each data with the best $\sigma = 0.2$. Can be seen that the times decrease with the number of processor but not linearly. This behavior can be explained because unless the problem is embarrassing parallel several resources of the computer are shared like the caches of the CPU and the memory. Besides, the numpy fights for cores with the multiprocessing approach. Also the real cores are only 6, when are used more they share computations units too, therefore the performance can not improve the same.

5 KDE on images

The kernel density estimation is a way to learn and reproduce some function, in particular in grey 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 using images of one 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.