CS207 Algorithm Analysis: Kernel Density Estimation

Gioia Dominedò   Nicolas Drizard   Kendrick Lo   Malcolm Mason Rodriguez

May 3, 2016

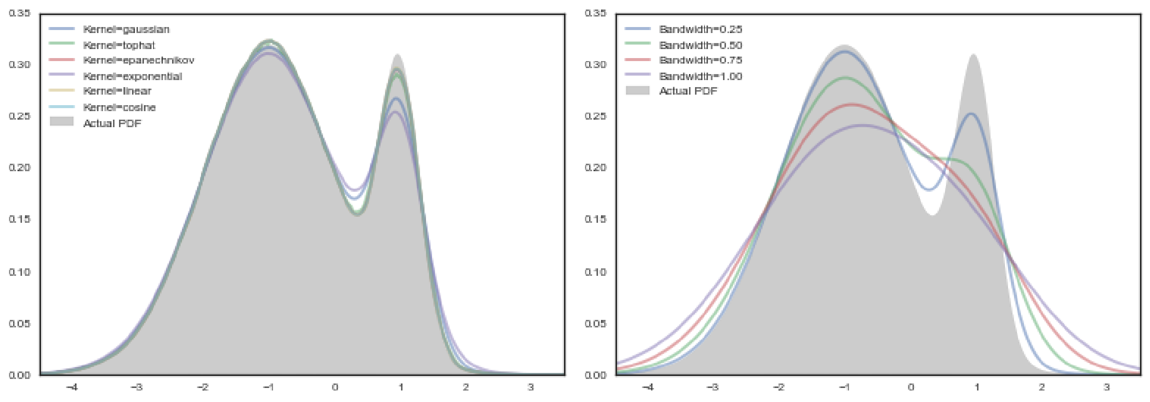
1 Introduction

Kernel Density Estimation (KDE) is a non-parametrical statistical technique that is used to estimate the probability density function (PDF) of a random variable based on a finite data sample[1]. Mathematically, we can denote the KDE of the PDF of independent and identically distributed (*iid*) random variables $X = \{x_1, x_2, ..., x_n\}$ as:

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\Big(\frac{x - x_i}{h}\Big) \qquad (1)$$

where $K(\cdot)$ represents the *kernel* and h represents the *bandwidth* (i.e. a smoothing parameter). We note that $\hat{f}_h(x)$ denotes the estimated PDF at a particular point x, and that the entire data set X is used to calculate it.

Various kernels are used in practice (e.g. Gaussian, Epanechnikov, exponential, etc.), all of which meet the criteria of being non-negative functions with mean zero that integrate to one. Similarly, different (strictly positive) values of h can be chosen depending on the data set in question. Figure 1 illustrates how the choice of these two parameters affects the shape of the estimated PDF.



(a) Changing the kernel (bandwidth= 0.2)   (b) Changing the bandwidth (gaussian kernel)

Figure 1: Impact of KDE parameters

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

2 Algorithm

For ease of comparability, we used a consistent set of parameters for all our experiments: a Gaussian kernel and a bandwidth of 0.2. Figure 2 shows the result of running the algorithm with these parameters on 100K one-dimensional random variables drawn from a bimodal distribution, where the PDF is plotted at 1K evenly-spaced points.

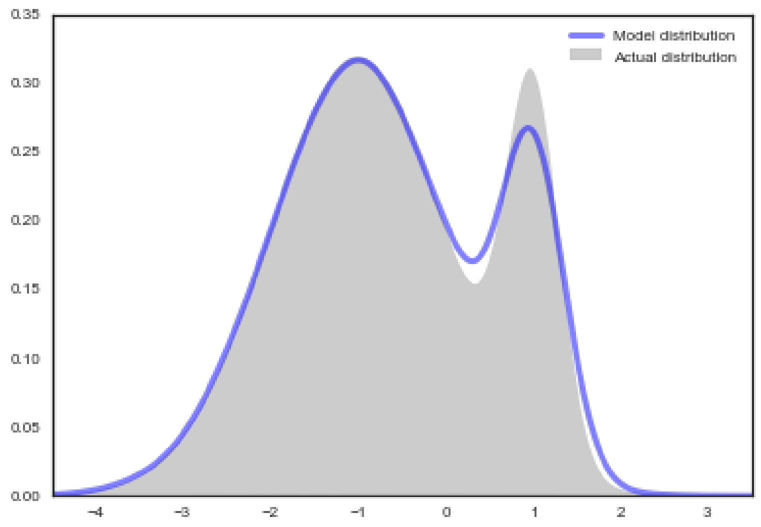


Figure 2: Sample KDE output

We started by implementing a "pure" Python version of KDE; in keeping with that objective, we only used lists. We can see from the code below that the KDE algorithm is a direct application of Equation 1. The double loop, which is necessary due to the use of lists, immediately stands out as a significant source of inefficiencies.

```
def naive_kde(x, x_grid, h):
    '''
    Calcalates the KDE PDF estimate, based on a Gaussian kernel.
    Inputs
    ----------
    x      : A list of iid random variables that are used to calculate the KDE estimate
    x_grid : A list of values along which to calculate the KDE estimate
    h      : Float, the bandwidth parameter for the KDE estimate
    Returns
    ----------
    A list of KDE estimates for all values of x_grid.
    '''
    global N # number of iid random variables
    estimates = [0] * len(x_grid)

    for j, gridpt in enumerate(x_grid):
        for i in range(N):
            val_in_sum = gaussian((x[i] - gridpt) / h)
            estimates[j] += val_in_sum

        estimates[j] = estimates[j] / (N * h)

    return estimates
```

Before moving on to the Scikit Learn implementation, we tested the effect of switching from lists to Numpy arrays and vectorizing the inner loop. This simple change, which is shown in the code below, reduced the runtime from nearly 300 seconds to just over 1 second!

```
def vectorized_kde(x, x_grid, h):
    '''
    Calcalates the KDE PDF estimate, based on a Gaussian kernel.
    Inputs
    ----------
    x      : An array of iid random variables that are used to calculate the KDE estimate
    x_grid : An array of values along which to calculate the KDE estimate
    h      : Float, the bandwidth parameter for the KDE estimate
    Returns
    ----------
    An array of KDE estimates for all values of x_grid.
    '''
    global N # number of iid random variables

    estimates = np.zeros(len(x_grid))

    for j, gridpt in enumerate(x_grid):
        val_in_sum = gaussian((x - gridpt)/h)  # returns vector of dim N
        estimates[j] = np.sum(val_in_sum) / (N * h)

    return estimates
```

3 ScikitLearn Optimizations

Implementation	Runtime (seconds)
Naive Python	298.28
Vectorized Python	1.34
Scikit Learn	5.73

Table 1: Comparing one-dimensional KDE implementations