Chapter 5: Kernel Density Estimation and Probabilistic Neural Networks

1 Density Estimation

Density Estimation is the problem of reconstructing the probability density function using a set of given data points. Suppose we observe $X_1, \cdots, X_n$, and we want to recover the underlying probability density function generating our dataset.

Here, we assume that the data comes from a continuous distribution. In a parametric setting, the density function relies on its parameters. Hence, to estimate the density, it is enough to estimate the parameters involved in the density function. But these estimates of density will be valid only when data actually follows the assumed distribution.

It is very likely that someone who wants to know the distribution does not know ahead of time that the density function belongs to a certain class of parametric distribution functions. In such cases, we need non-parametric estimation. For a detailed reading, see [1].

1.1 Estimating CDF

Assume that X follows a continuous distribution f with CDF F, then for $h > 0$

$$P\left(x - \frac{h}{2} < X < x + \frac{h}{2}\right) = \int_{x-\frac{h}{2}}^{x+\frac{h}{2}} f(y)\mathrm{d}y \tag{1}$$

Now if f is smooth and h is small, then:

$$\int_{x-\frac{h}{2}}^{x+\frac{h}{2}} f(y)\mathrm{d}y \approx h \cdot f(x) \Rightarrow \hat{f}(x) = \frac{F\left(x+\frac{h}{2}\right) - F\left(x-\frac{h}{2}\right)}{h}$$

Hence, we have now moved to the problem of estimating the CDF of an unknown distribution. The following are some of the ways of estimating the cumulative density of an unknown continuous distribution.

Definition 1.1. *(Empirical CDF) If $X_1 \cdots X_n$ are IID random variables with distribution function F, then the empirical CDF is defined as*

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I\left(X_i \leq x\right), \text{ where } I \text{ is the Indicator function.} \tag{2}$$

Note that we approximate the unknown distribution here with a discrete distribution function. It is a step function estimate for F. From Eqs. (1) and (2), we can show that

$$\hat{f}(x) = \frac{\sum_{i=1}^{n} I\left(x - \frac{h}{2} \leq X_i \leq x + \frac{h}{2}\right)}{nh} \tag{3}$$

Some properties of empirical distribution function (EDF) are as follows.

[1] Textbook: Wasserman, Larry. All of nonparametric statistics. Springer Science & Business Media, 2006.

1. The empirical distribution function $(\hat{F}_n)$ is an unbiased estimator of the true underlying distribution (F).

   Proof. $$E(\hat{F}(x)) = E\left(\frac{1}{n}\sum_{i=1}^{n} I\left(X_i \leq x\right)\right) = \frac{1}{n}\sum_{i=1}^{n} E\left(I\left(X_i \leq x\right)\right) = \frac{1}{n}\sum_{i=1}^{n}\int_{-\infty}^{\infty} I(y \leq x) f(y)dy$$
   $$= \frac{1}{n}\sum_{i=1}^{n}\int_{-\infty}^{x} f(y)dy = \frac{1}{n}\sum_{i=1}^{n} F(x) = F(x).$$

2. $MSE = \text{Var}\left(\hat{F}_n\right) = \frac{1}{n}F(x)(1-F(x))$.

   Proof. $$\text{Var}\left(\hat{F}_n(x)\right) = \frac{1}{n^2}\sum_{i=1}^{n}\left[E\left(\{I\left(X_i \leq x\right)\}^2\right) - \{E\left(X_i \leq x\right)\}^2\right],$$

   where $E\left(X_i \leq x\right) = F(x)$ and $E\left(\{I\left(X_i \leq x\right)\}^2\right) = F(x)$ and hence this proves the result.

3. $\hat{F}_n(x) \longrightarrow F(x)$ in probability.

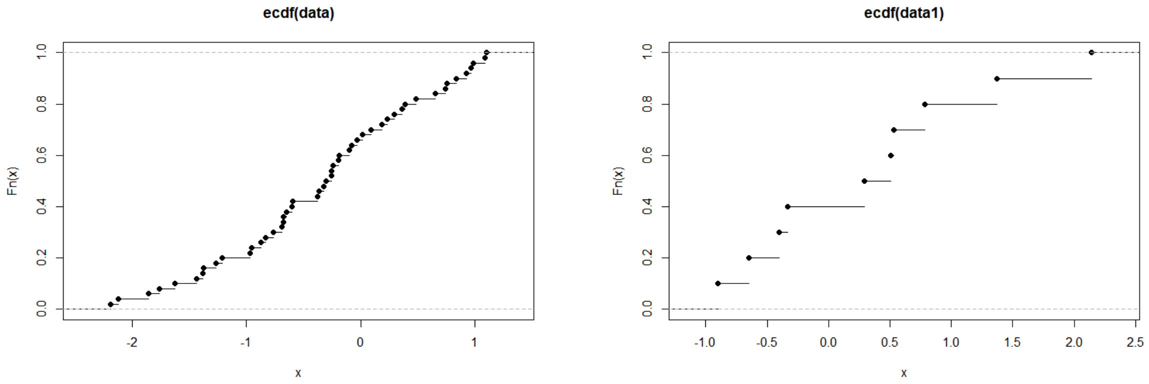
   Proof. This result can be proved using Chebyshev's inequality: $P(|X - \mu| \geq k) \leq \frac{\sigma^2}{k^2}$.
   In fact, it can be shown that $\hat{F}_n(x) \longrightarrow F(x)$ almost surely and uniformly.

Drawbacks: There are two main drawbacks to this estimate:

1. A continuous distribution is being defined by a discrete non-continuous function.
2. If our data set is small, then we cannot expect a good estimate. To get a good estimate in this case, we need a large number of data points.

The following R-code shows how to plot an ecdf in R.

```
data <- rnorm(50 , 0 , 1) ## store 50 normal random variables in variable "data"
F50 <- ecdf(data) # f stores the emperical distribution function
plot.ecdf(F50) # See Figure 1.(a)
data1 <- rnorm(10)
F10 <- ecdf(data1)
plot.ecdf(F10) # See Figure 1.(b)
```



(a) ***ECDF for 50 points from Normal*** *(b)* ***ECDF for 10 points from Normal***

Figure 1 : ***The distribution is better approximated with more points.***

1.2 Histogram and Centred Histogram

A histogram is a well-known and popular method for approximating the density of a continuous distribution. Formally, it can be defined in the following mathematical manner[2]: Let X be the random sample of size n from a continuous population, then the histogram density estimate is given by

$$\hat{h}_n(x) = \frac{\text{no. of } X_i \in B_j}{nh} \text{ when } x \in B_j \tag{4}$$

where B_j are different partitions of $[0, 1]$. Without loss of generality, we can assume that each X_i is in $[0, 1]$, then

$$B_1 = \left[0, \frac{1}{M}\right); B_2 = \left[\frac{1}{M}, \frac{2}{M}\right) \cdots B_M = \left[\frac{M-1}{M}, 1\right) \text{ where } h = \frac{1}{M} \text{ is called bin width.}$$

It can be shown that the expectation of this estimate is:

$$\mathbb{E}\left(\hat{h}_n(x)\right) = \frac{M}{n}\sum_{i=1}^{n} P\left(X_i \in B_j\right) = MP\left(X_i \in B_j\right) = M\left[F\left(\frac{j}{M}\right) - F\left(\frac{j-1}{M}\right)\right] \tag{5}$$

Using the fact that $\frac{1}{M} = \frac{j}{M} - \frac{j-1}{M}$ and then applying mean value theorem to F, we get that there exists x^* such that

$$\mathbb{E}\left(\hat{h}_n(x)\right) = \frac{\left[F\left(\frac{j}{M}\right) - F\left(\frac{j-1}{M}\right)\right]}{\frac{j}{M} - \frac{j-1}{M}} = h\left(x^*\right); \text{ for } x^* \in B_j \tag{6}$$

Applying mean value theorem to h, we get the existance of x^{**} such that

$$\frac{h\left(x^*\right) - h(x)}{x^* - x} = h'\left(x^{**}\right) \tag{7}$$

From Eqs. (6) and (7), we can show that

$$\text{Bias } = E\left(\hat{h}_n(x)\right) - h(x) \leq \frac{\left|h'\left(x^{**}\right)\right|}{M} \tag{8}$$

$$\text{MSE} = \text{Var}\left(\hat{h}_n(x)\right) + \text{Bias}^2 \leq \frac{\left(h'\left(x^{**}\right)\right)^2}{M^2} + \frac{Mh\left(x^*\right)}{n} + \frac{\left(h\left(x^*\right)\right)^2}{n}. \tag{9}$$

Observations: Using similar calculations as shown above, we can show that from the inequalities (8) and (9), we observe that as M increases (number of bins increases), the chance of over-estimating or under-estimating the data reduces. Hence, increasing the bin width increases the bias but reduces the variability. This is referred to as **over-smoothing**. Conversely, if the bin width reduces, our bias reduces, but variability increases. This is referred to as **under-smoothing**.

Drawbacks: The drawbacks of using histogram as density estimate are:

1. The estimate of a continuous distribution is not continuous
2. The density estimator depends on the width and end points of certain fixed intervals which are chosen beforehand.
3. Increasing the number of bins might reduce the bias, but it will also increase the chances of getting regions of 0 probability in the histogram.

[2] https://faculty.washington.edu/yenchic/18W_425/Lec6_hist_KDE.pdf

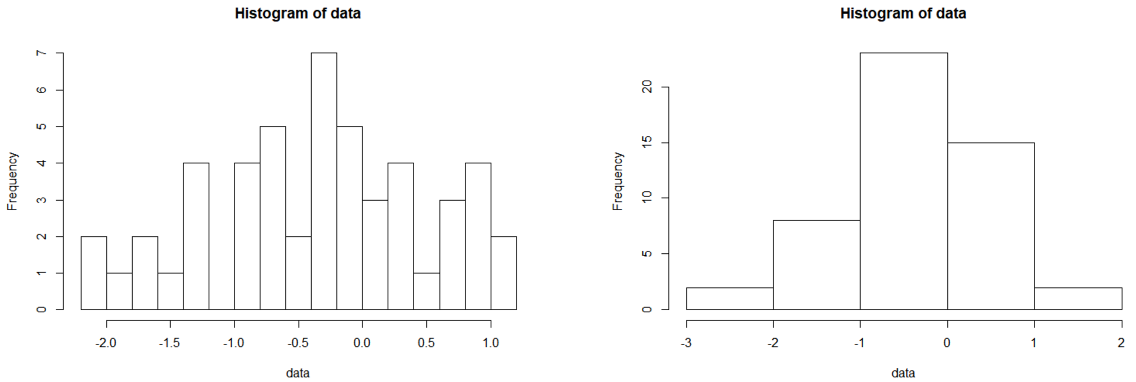
The estimate's dependency on the bin width and its endpoints can be removed by defining a **centered histogram**. In a normal histogram, all B_j are fixed and pre-determined. However, in the case of the centered histogram, the bandwidth is kept fixed at h, but the intervals are not. The pdf estimate at x in this case is

$$\hat{h}_c(x) = \frac{\text{no. of } X_i \in \left(x - \frac{h}{2}, x + \frac{h}{2}\right)}{nh}.$$

1.3 Implementation in R

The R-codes below show how to plot a histogram and a centered histogram in R.

```
hist(data, breaks = 20, col = NULL) ## to get a histogram with 20 bins
hist(data, breaks = 3, col = NULL) #over-smoothed, see Figure 2.
```

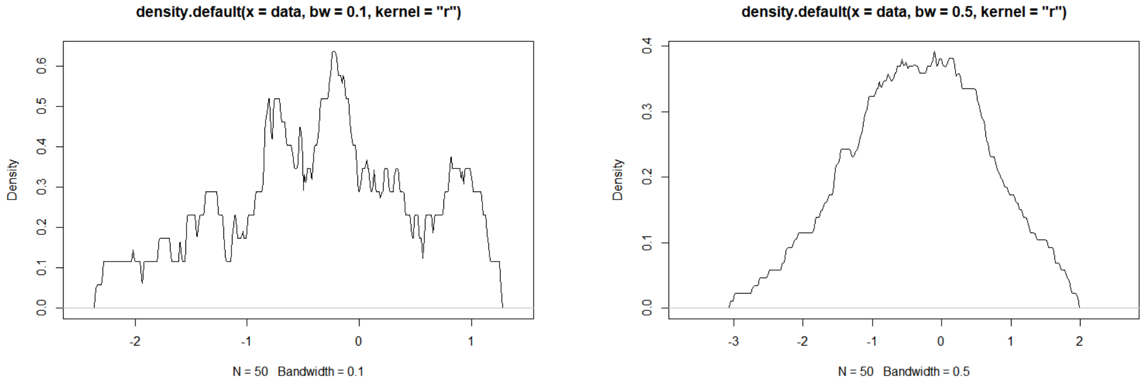


(a) ***High variance due to many bins*** *(b)* ***Oversmoothed estimate***

Figure 2 : ***The distribution is not well approximated with too many or too few points.***

Now, using the R-code below, we plot centered histograms and observe that an increase in bandwidth increases the smoothness of the estimate.

```
x <- density(data, kernel = "r", bw = 0.1)
z <- density(data, kernel = "r", bw = 0.5)
plot(x) # see Figure 3.(a)
plot(z) # see Figure 3.(b)
```



(a) bw = 0.1 ***(high variance)*** *(b)* ***bw*** = 0.5 ***(over-smoothed)***

Figure 3 : ***Plotting a centered histograms with*** bw = 0.1 ***and*** 0.5***.***

2 Kernel Density Estimation

Kernel Density Estimation (KDE) is a non-parametric method to estimate the probability density function of a random variable based on kernels as weights. KDE answers a fundamental data smoothing problem where inferences about the population are made based on a finite data sample. In some fields, such as signal processing and econometrics, it is also termed the Parzen–Rosenblatt window method[3]. One of the famous applications of kernel density estimation is in estimating the class-conditional marginal densities of data when using a naive Bayes classifier, which can improve its prediction accuracy. Another straightforward application of KDE is the development of Probabilistic Neural Networks (to be discussed).

To avoid confusion, in KDE, The "kernel" often includes the normalization constant because it's used as a standalone PDF, whereas in a statistical PDF or PMF, the "kernel" excludes any constant factors not dependent on the variable of interest. We define both the kernel of a PDF and the kernel in the context of KDE below:

Definition 2.1. *The kernel of a probability density function (pdf) or probability mass function (pmf) is the factor of the pdf or pmf in which any factors that are not functions of any of the variables in the domain are omitted.*

For example, the kernel of the Beta distribution is:

$$K(x; a, b) = x^{a-1}(1-x)^{b-1}.$$

The kernel of the Gaussian (Normal) distribution is:

$$K(x; \mu, \sigma) = \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}.$$

Definition 2.2. *A function $K(x)$ is called a kernel function in the context of kernel density estimation if it satisfies the following properties:*

1. $K(x) \geq 0$ *for all* x *(non-negativity),*
2. $K(-x) = K(x)$ *(symmetry),*
3. $\int_{-\infty}^{\infty} K(x)\mathrm{d}x = 1$ *(normalization).*

Example 2.1. *Examples of some of the commonly used kernels are:*

1. $K(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$ *is called the Gaussian kernel function (Smooth, Symmetric, differentiable, and always positive).*
2. $K(x) = \frac{3}{4} \max\left\{1 - x^2, 0\right\}$ *is called the Epanechnikov Kernel (Quadratic shape).*
3. $K(x) = \begin{cases} \frac{1}{2} & \text{if } -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$ *is called the Uniform Kernel (computationally very efficient).*

Disadvantages of histogram have motivated defining the **kernel density estimates** (KDE). Not only does kernel density estimate solve the problem of dependency on the choice of bins, but it also solves the continuity problem, i.e., KDE of a continuous function is continuous, provided we choose a smooth kernel. Moving from

[3] Parzen, Emanuel. "On estimation of a probability density function and mode." The Annals of Mathematical Statistics (1962).

the histogram to KDE, we solved the problem of dependencies on the bin, and then, we generalized the centered histogram to obtain KDE, which also solved the continuity problem.

Definition 2.3. *Given $X_1, \cdots X_n$ as a sample of n observations, the KDE at point x is given by:*

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right), \tag{10}$$

where K is the kernel function (typically a symmetric, non-negative function that integrates to 1) and $h > 0$ is the bandwidth (a smoothing parameter that determines the width of the kernel and controls the trade-off between bias and variance).

If we're explicitly working with observed values $x_1, x_2, \ldots, x_n$, we could also write:

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right).$$

Remark 2.1.

1. *We can view KDE as a generalization of a centered histogram. Changing the kernels gives different estimates for the function.*
2. *The Epanechnikov is a special kernel with the lowest (asymptotic) mean square error.*
3. *Some special types of kernel functions, known as the higher-order kernel functions, will take the negative values at some regions. Though very counterintuitive, These higher-order kernel functions might have a smaller bias than the usual ones.*

In R statistical software, a dataset called Old Faithful Geyser Data is available, which gives the Waiting time between eruptions and the eruption duration for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA. In practice, kernel functions show some differences in the density estimator. In what follows, we consider the three most common kernel functions (given in Eg. 2.1) and apply them to the `faithful dataset`:

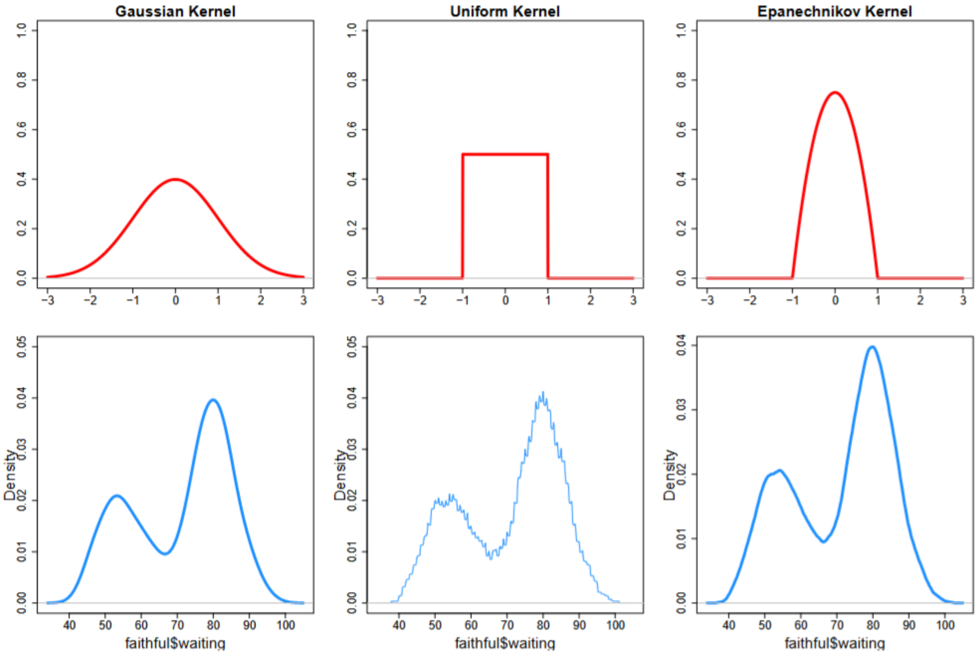


Figure 4 : ***The top row displays the three kernel functions and the bottom row shows the corresponding density estimators.***

2.1 Inferential Properties of KDE

Let $X_1 \cdots X_n$ be an IID sample from an unknown population following a density function "$p(x)$". Let us consider a single point x_0. We analyze the quality of the estimate:

$$\hat{p}_n\left(x_0\right)=\frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x_0-X_i}{h}\right).$$

2.1.1 Bias of the KDE

Using the basic definition of expectation (using integral) of the function of random variables, we analyze the bias of the KDE:

$$\begin{aligned}
\mathbb{E}\left(\widehat{p}_n\left(x_0\right)\right)-p\left(x_0\right) &=\mathbb{E}\left(\frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{X_i-x_0}{h}\right)\right)-p\left(x_0\right) \\
&=\frac{1}{h}\mathbb{E}\left(K\left(\frac{X_i-x_0}{h}\right)\right)-p\left(x_0\right) \\
&=\frac{1}{h}\int K\left(\frac{x-x_0}{h}\right) p(x)dx-p\left(x_0\right).
\end{aligned} \tag{11}$$

Now we do a change of variable $y=\frac{x-x_0}{h}$ so that $dy=dx/h$ and the above becomes (using the fact that $x=x_0+hy$)

$$\begin{aligned}
\mathbb{E}\left(\widehat{p}_n\left(x_0\right)\right)-p\left(x_0\right) &=\int K\left(\frac{x-x_0}{h}\right) p(x)\frac{dx}{h}-p\left(x_0\right) \\
&=\int K(y) p\left(x_0+hy\right) dy-p\left(x_0\right).
\end{aligned}$$

Now, by Taylor expansion, when h is small,

$$p\left(x_0+hy\right)=p\left(x_0\right)-hy\cdot p'\left(x_0\right)+\frac{1}{2}h^2y^2p''\left(x_0\right)+O\left(h^2\right). \tag{12}$$

Note that $O\left(h^2\right)$ means that it is a smaller order term compared to h^2 when $h\to 0$. Plugging this back into the bias, we obtain

$$\begin{aligned}
\mathbb{E}\left(\widehat{p}_n\left(x_0\right)\right)-p\left(x_0\right) &=\int K(y) p\left(x_0-hy\right) dy-p\left(x_0\right) \\
&=\int K(y)\left[p\left(x_0\right)+hy\cdot p'\left(x_0\right)+\frac{1}{2}h^2y^2p''\left(x_0\right)+o\left(h^2\right)\right] dy-p\left(x_0\right) \\
&=\int K(y) p\left(x_0\right) dy+\int K(y) hy\cdot p'\left(x_0\right) dy+\int K(y)\frac{1}{2}h^2y^2p''\left(x_0\right) dy+o\left(h^2\right)-p\left(x_0\right) \\
&=p\left(x_0\right)\underbrace{\int K(y)dy}_{=1}+hp'\left(x_0\right)\underbrace{\int yK(y)dy}_{=0}+\frac{1}{2}h^2p''\left(x_0\right)\int y^2K(y)dy+o\left(h^2\right)-p\left(x_0\right) \\
&=p\left(x_0\right)+\frac{1}{2}h^2p''\left(x_0\right)\int y^2K(y)dy-p\left(x_0\right)+O\left(h^2\right) \\
&=\frac{1}{2}h^2p''\left(x_0\right)\int y^2K(y)dy+O\left(h^2\right) \\
&=\frac{1}{2}h^2p''\left(x_0\right)\mu_K+O\left(h^2\right),
\end{aligned}$$

where $\mu_K=\int y^2K(y)dy$. Namely, the bias of the KDE is

$$\operatorname{bias}\left(\widehat{p}_n\left(x_0\right)\right)=\frac{1}{2}h^2p''\left(x_0\right)\mu_K+O\left(h^2\right). \tag{13}$$

This means that when we allow $h \to 0$, the bias is shrinking at a rate $O\left(h^2\right)$. Equation (13) reveals an interesting fact: the bias of KDE is caused by the curvature (second derivative) of the density function! Namely, the bias will be very large at a point where the density function curves a lot (e.g., a very peaked bump). This makes sense because, for such a structure, KDE tends to smooth it too much, making the density function smoother (less curved) than it used to be.

Note: Big O notation is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity.

Definition 2.4. *Given two functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there exist constants $m > 0$ and $n_0 \geq 0$ such that $f(n) \leq mg(n)$ for all $n \geq n_0$.*

2.1.2 Variance of KDE

For the analysis of variance, we can obtain an upper bound using a straightforward calculation:

$$\begin{aligned}
\operatorname{Var}\left(\widehat{p}_n\left(x_0\right)\right) &= \operatorname{Var}\left(\frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{X_i - x_0}{h}\right)\right) \\
&= \frac{1}{nh^2}\operatorname{Var}\left(K\left(\frac{X_i - x_0}{h}\right)\right) \\
&\leq \frac{1}{nh^2}\mathbb{E}\left(K^2\left(\frac{X_i - x_0}{h}\right)\right) \\
&= \frac{1}{nh^2}\int K^2\left(\frac{x - x_0}{h}\right)p(x)dx \\
&= \frac{1}{nh}\int K^2(y)p\left(x_0 + hy\right)dy \quad \left(\text{using } y = \frac{x - x_0}{h} \text{ and } dy = dx/h \text{ again}\right) \\
&= \frac{1}{nh}\int K^2(y)\left[p\left(x_0\right) + hyp'\left(x_0\right) + O(h)\right]dy \\
&= \frac{1}{nh}\left(p\left(x_0\right)\cdot\int K^2(y)dy + O(h)\right) \\
&= \frac{1}{nh}p\left(x_0\right)\int K^2(y)dy + O\left(\frac{1}{nh}\right) \\
&= \frac{1}{nh}p\left(x_0\right)\sigma_K^2 + O\left(\frac{1}{nh}\right)
\end{aligned}$$

Therefore, we have the following:

$$\operatorname{Var}\left(\hat{p}_n\left(x_0\right)\right) \leq \frac{1}{nh}p\left(x_0\right)\sigma_K^2 + O\left(\frac{1}{nh}\right), \tag{14}$$

where $\sigma_K^2 = \int K^2(y)dy$. Therefore, the variance shrinks at rate $O\left(\frac{1}{nh}\right)$ when $n \to \infty$ and $h \to 0$. An interesting fact from the variance is that: At the point where the density value is large, the variance is also large!

Now, putting both bias and variance together, we obtain the MSE of the KDE:

$$\operatorname{MSE}\left(\hat{p}_n\left(x_0\right)\right) = \left(\frac{h^2}{2}p''\left(x_0\right)\mu_K\right)^2 + \frac{1}{nh}p\left(x_0\right)\sigma_K^2 + O\left(h^4\right) + O\left(\frac{1}{nh}\right) \tag{15}$$

The first two terms on the right-hand side in Eq. (15) are called **Asymptotic Mean Square Error** (AMSE). In the KDE, the smoothing bandwidth h is something we can choose. Thus, the bandwidth h minimizing the AMSE is

$$h_{opt}\left(x_0\right) = \left[\frac{4p\left(x_0\right)\sigma_K^2}{n\left|p''\left(x_0\right)\right|^2\mu_K^2}\right]^{\frac{1}{5}} \tag{16}$$

Remark 2.2. *However, the major problem is that the above optimal h is just a theoretical minimum. We cannot use this in real-life datasets because we do not know the distribution $p(x)$.*

In all the above analysis, we considered only one point x_0, but in general, we want to control the overall MSE of the entire function. For one point x_0,

$$\text{MSE} = \mathbb{E}\left[\left(\hat{p}_n\left(x_0\right) - p\left(x_0\right)\right)^2\right].$$

So for all points x, we have

$$\text{MISE} = \mathbb{E}\left[\int \left(\hat{p}_n\left(x\right) - p\left(x\right)\right)^2 \, \mathrm{d}x\right] \tag{17}$$

MISE is called the mean integrated square error. Now, some calculations will let us show that

$$\text{MISE}\left(\hat{p_n}\right) = \left\{\frac{1}{4}h^4\mu_K^2 \int_{-\infty}^{\infty} \left|p''(x)\right|^2 \mathrm{d}x\right\} + \frac{\sigma_K^2}{nh} + O\left(h^4\right) + O\left(\frac{1}{nh}\right) \tag{18}$$

The dominating term of Eq. (18) is called as the **asymptomatic mean integrated square error**. Then the optimal smoothing bandwidth can be obtained by minimizing AMISE w.r.t h is given by:

$$h_{opt} = \left[\frac{4\sigma_K^2}{n\mu_k^2\left(\int_{-\infty}^{\infty} \left|p''(x)\right|^2 \mathrm{d}x\right)}\right]^{\frac{1}{5}} \tag{19}$$

Remark 2.3. *Again, Eq. (19) can't be used for practical purposes because p is unknown.*

1. *How to choose "h" is still unsolved problem (popularly known as* ***bandwidth selection problem****).*
2. *There are modifications to this procedure where we can use a variable bandwidth (omitted here).*

2.1.3 Confidence Interval using the KDE

In this section, we focus on the CI of the density function at a given point x_0. Recall from Eqn. (10),

$$\widehat{p}_n\left(x_0\right) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{X_i - x_0}{h}\right) = \frac{1}{n}\sum_{i=1}^{n} Y_i,$$

where $Y_i = \frac{1}{h}K\left(\frac{X_i - x_0}{h}\right)$. Thus, the KDE evaluated at x_0 is actually a sample mean of $Y_1, \cdots, Y_n$. By CLT,

$$\sqrt{n}\left(\frac{\widehat{p}_n\left(x_0\right) - \mathbb{E}\left(\widehat{p}_n\left(x_0\right)\right)}{\operatorname{Var}\left(Y_i\right)}\right) \xrightarrow{D} N(0,1),$$

$$\text{where } \operatorname{Var}\left(Y_i\right) = \operatorname{Var}\left(\frac{1}{h}K\left(\frac{X_i - x_0}{h}\right)\right) = \frac{1}{h}p\left(x_0\right)\sigma_K^2 + o\left(\frac{1}{h}\right)$$

diverges when $h \to 0$. Thus, when $h \to 0$, the asymptotic distribution of $\widehat{p}_n\left(x_0\right)$ is

$$\sqrt{nh}\left(\widehat{p}_n\left(x_0\right) - \mathbb{E}\left(\widehat{p}_n\left(x_0\right)\right)\right) \xrightarrow{D} N\left(0, p\left(x_0\right)\sigma_K^2\right).$$

Thus, a $1 - \alpha$ CI can be constructed using

$$\widehat{p}_n\left(x_0\right) \pm z_{1-\alpha/2} \cdot \sqrt{p\left(x_0\right)\sigma_K^2}.$$

This CI cannot be used in practice because $p\left(x_0\right)$ is unknown. One solution to this problem is to replace it with the KDE, leading to

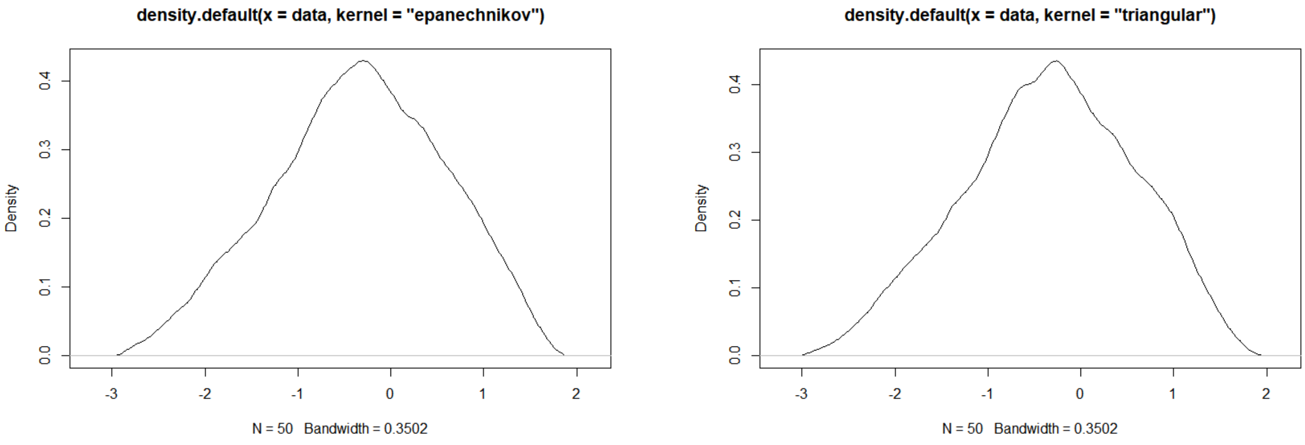
$$\widehat{p}_n\left(x_0\right) \pm z_{1-\alpha/2} \cdot \sqrt{\widehat{p}_n\left(x_0\right)\sigma_K^2}.$$

Remark 2.4. *A problem of these CI is that the theoretical guarantee of coverage is for the expectation of the KDE* $\mathbb{E}\left(\widehat{p}_n\left(x_0\right)\right)$ *rather than the true density value* $p\left(x_0\right)$! *There is another approach for constructing CIs called the bootstrap approach (not discussed here).*

2.2 Implementation in R

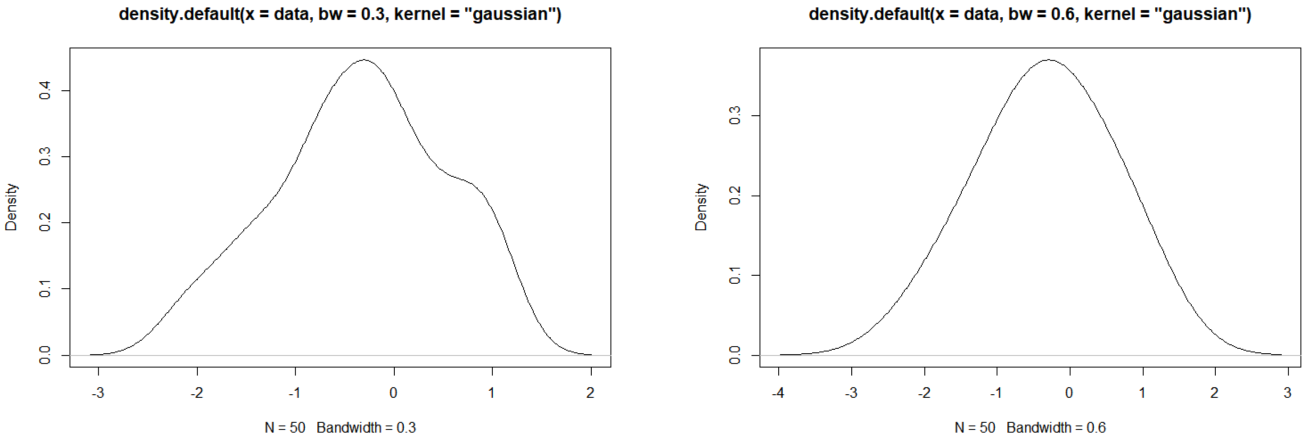
The following R-codes plot KDEs. We observe two things from these plots. Fig. 5 shows that the choice of the kernel slightly affects the shape of the density estimate. Fig. 6 demonstrates that increasing the bandwidth increases the smoothness of the estimate.

```r
plot(density(data, kernel = "gaussian"))
plot(density(data, kernel = "epanechnikov")) #smoother the kernel, smoother the estimate
plot(density(data, kernel = "triangular"))
plot(density(data, kernel = "gaussian", bw= 0.3 ))
plot(density(data, kernel = "gaussian", bw= 0.6 ))
```



(a) ***KDE with kernel*** $=$ ***epanechnikov kernel*** (b) ***KDE with kernel*** $=$ ***triangular kernel***

Figure 5 : ***Change in the kernel does not affect the estimate so much***



(a) ***Kernel*** $=$ ***gaussian , bandwidth*** $= 0.3$ (b) ***Kernel*** $=$ ***gaussian, bandwidth*** $= 0.5$

Figure 6 : ***Increase in bandwidth smoothens the estimate***

3 Smoothing

A smoothing algorithm summarizes the trend in Y as a function of $X_1 \cdots X_n$. Smoother takes the data and returns a function called **smooth**. In the bivariate case, a smoother is a procedure that is applied to the bivariate data $(x_1, y_1) \cdots (x_n, y_n)$ that produces a decomposition $y_i = s(x_i) + \epsilon_i$ where s is called the smooth function, also called as smooth.

In this section, we discuss a couple of smoothing methods (we consider linear ones here). The following are the assumptions for this section:

1. There are n pairs of observations $(x_1, y_1) \cdots (x_n, y_n)$ and without loss of generality, assume that $x_1 \leq x_2 \leq \cdots \leq x_n$.

2. All observations are related through the expression $y_i = f(x_i) + \epsilon_i$ for $i = 1, 2, \cdots n$.

3. ϵ_i's are IID from a continuous distribution centered at 0.

3.1 Local Averaging (Friedman)

This linear smoother is given by the below expression where x_j are such that $f(x_j) = y_j$ and x_j for $j = 1, 2 \cdots s$ are "s" points in the "neighbourhood" of x_i.

$$\hat{f}(x_i) = \frac{\sum_{i=1}^{s} y_j}{s}.$$

Now, the neighbourhood of x_i is the smallest symmetric window about x_i containing s observations. The number of points in the window is called the span. Note that the window size changes for different values of x_i but always includes s data points. A larger span over-smooths the data, and smaller spans provide an under-smoothed estimate. Friedman proposed using a cross-validation method to choose the span. For further reading, see [4].

3.1.1 Cross-Validation

Let $\hat{g}_\lambda(x)$ be an estimate for $g(x)$. Then the **predictive squared error** (pse) of $\hat{g}_\lambda(x)$ is given by

$$\text{pse}(\lambda) = E\left[(y^* - \hat{g}_\lambda(x))^2\right]$$

where y^* is the **new** response value associated with the predictor x i.e. y^* and y_i are independent of each other for all i. Cross-validation is an idea in regression where we estimate pse(λ) by CV(λ) where

$$\text{CV}(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{g}_{\lambda,-i}(x_i)\right)^2 \tag{20}$$

where $\hat{g}_{\lambda,-i}$ is the estimate for the g using the data points $x_1 \cdots x_{i-1}, x_{i+1}, \cdots x_n$. Eliminating the i^{th} point while estimating g makes sure that y_i is independent of all the other response values.

3.1.2 Choosing a Span

Let $\hat{y}_{(i)}$ be the estimate of y_i determined by using all observed data points except (x_i, y_i). If s is the span, $e_i(s)$ is defined as $y_i - \hat{y}_{(i)}$, then s is chosen such that $\frac{\sum_{i=1}^{n} e_i(s)^2}{n}$ is minimized over a set S of possible span values. The span selected this way is called the *global span* because this span is used for every point x_i. There are procedures for obtaining variable spans (not discussed here).

[4] http://users.stat.umn.edu/~helwig/notes/smooth-notes.html

3.2 Implementation in R

Now, we see an R-data example for the above theory. This data is about nitrogen oxide concentrations found in engine exhaust for ethanol engines. There are 88 pairs of data in this data set. We wish to smooth the data using "Friedman's local averaging".

```
##########  Friedman Local Averaging #############
# Use "supsmu" to smooth the data using the local averaging method
# Use span = "cv" to use the cross-validated variable span
# cv span-related smoothing might have a better balance of variance and bias

library(lattice)
etoh <- lattice::ethanol #ethanol data
head(etoh)
plot(x = etoh$E , y = etoh$NOx , xlab = "engine ratio", ylab = "N oxide conc")
plot(supsmu(x = etoh$E, y = etoh$NOx , span = "cv"))
```

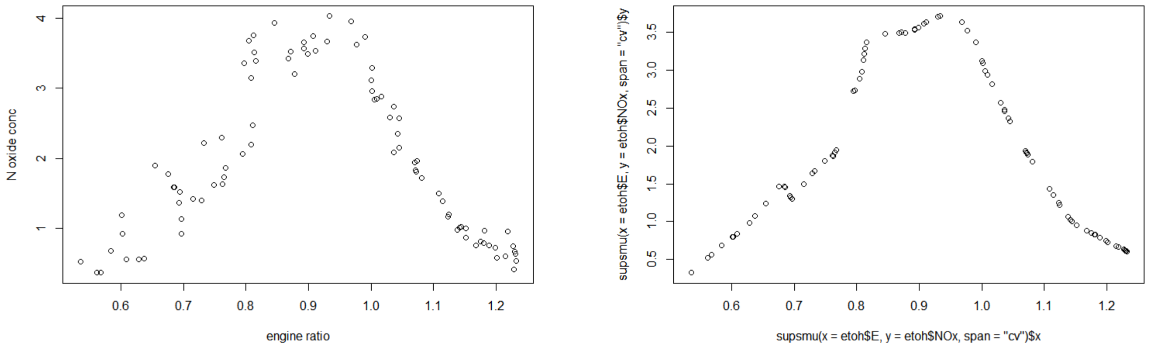


Figure 7 : ***The ethanol data (left) and Ethanol data smoothed using Friedman's averaging and*** cv ***span (right)***

```
# if span = p then it smooths the data using a constant span of size pn
plot(supsmu(x = etoh$E, y = etoh$NOx , span = 0.05))
plot(supsmu(x = etoh$E, y = etoh$NOx , span = 0.30))
# appears to be too smoothed. So biased.
```

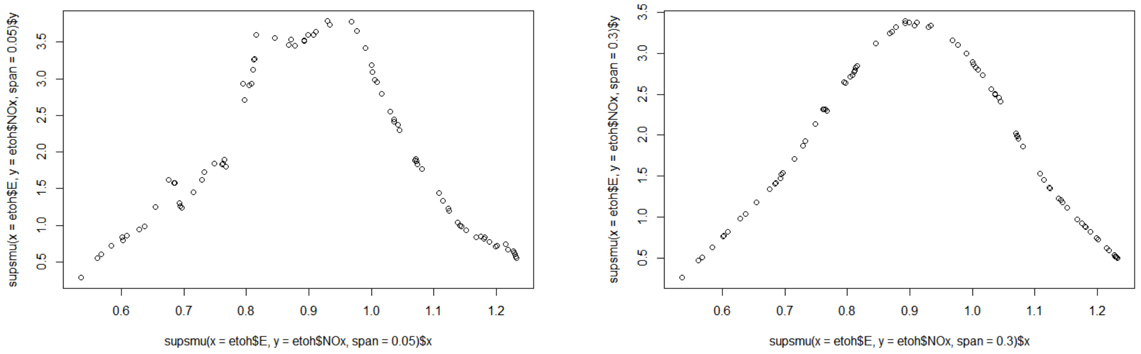


Figure 8 : ***Ethanol data smoothed using Friedman's averaging and span*** $= 0.05$ ***(left) and*** span $= 0.3$ ***(right)***

We observe how a change in bandwidth changes how smoothed the data gets from Fig. 8 . Clearly, increasing the span increases the smoothness, and hence bias increases.

4 Kernel Smoothing (Nadaraya and Watson)

Here, we want to estimate a general function "f" rather than a density function. Nadaraya and Watson independently introduced the following kernel regression estimate:

$$\hat{f}(x)=\frac{\sum_{i=1}^{n} y_i K\left(\frac{x-x_i}{h}\right)}{\sum_{j=1}^{n} K\left(\frac{x-x_j}{h}\right)}=\sum_{i=1}^{n} y_i w_i \text{ where } w_i=\frac{K\left(\frac{x-x_i}{h}\right)}{\sum_{j=1}^{n} K\left(\frac{x-x_j}{h}\right)} \tag{21}$$

Clearly, this estimate is linear in the observed data y_i and the weights w_i depend on kernel (K), bandwidth (h), and distance between x and x_i. Note that this is not the nearest neighbor method, unlike the previous two.

4.1 Deriving Nadaya-Watson Estimator

Let (X_i, Y_i) be independent pairs of random variables such that $Y_i = m(X_i) + \epsilon_i$ given that $E(\epsilon_i \mid X_i = x) = 0$

$$\hat{m}(x)=E(Y \mid X=x)=\int y f_{Y|X}(y \mid x)\mathrm{d}y=\frac{\int y f_{X,Y}(x,y)\mathrm{d}y}{f_X(x)} \tag{22}$$

The marginal (f_X) and joint ($f_{X,Y}$) densities are estimated using the following kernel density estimates.

$$\hat{f}_X(x)=\frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right) \text{ and } \hat{f}_{X,Y}(x,y)=\frac{1}{nh^2}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right) K\left(\frac{y-y_i}{h}\right) \tag{23}$$

Now use Eq. (23) in Eq. (22) and simplify in order to get the following

$$\hat{m}(x)=\frac{\frac{1}{h}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)\int_{\mathbb{R}} yK\left(\frac{y-y_i}{h}\right)\mathrm{d}y}{\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)} \tag{24}$$

Now, to simplify the integral, we use a change of variable $\phi = \frac{y-Y_i}{h}$ to get

$$\int_{\mathbb{R}} yK\left(\frac{y-y_i}{h}\right)\mathrm{d}y=\int_{\mathbb{R}} h\left(h\phi+Y_i\right)K(\phi)\mathrm{d}\phi=\int_{\mathbb{R}}\left(h^2+hY_i\right)K(\phi)\mathrm{d}\phi=0+hY_i \tag{25}$$

The step in Eq. (25) follows from kernel properties. Substituting Eq. (25) in Eq. (24) gives the desired result.

4.2 Implementation in R

Now, we will see how to smooth the ethanol data ((Fig. 8) using an N-W estimator and a Gaussian kernel.

```
etoh$NOx <- etoh$NOx[order(etoh$E)]
etoh$E <- sort(etoh$E) # "npreg" implements N-W Kernel reg. est. and requires x to be sorted
library(np)
etoh.npreg <- npreg(bws = 0.09 , txdat = etoh$E , tydat = etoh$NOx)
plot(etoh.npreg)
```

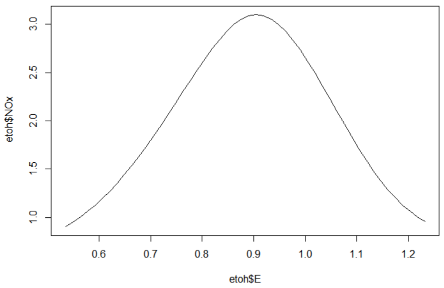


Figure 9 : ***Ethanol data smoothed using N-W estimator and a Gaussian kernel***

5 Probabilistic Neural Networks

Probabilistic Neural Networks (PNN) are a class of artificial neural networks that leverage statistical principles (Bayesian decision theory and kernel density estimation) to perform classification tasks. Introduced by Donald Specht in 1990[5], PNN has gained popularity due to its robustness, simplicity, and ability to handle noisy data.

Key Features:

- Provides probabilistic outputs for classification.
- Can be easily extended to multiclass problems.
- Handles non-linear decision boundaries effectively.

5.1 Architecture of PNN

The architecture of PNN is shown below (also see Fig. 10).

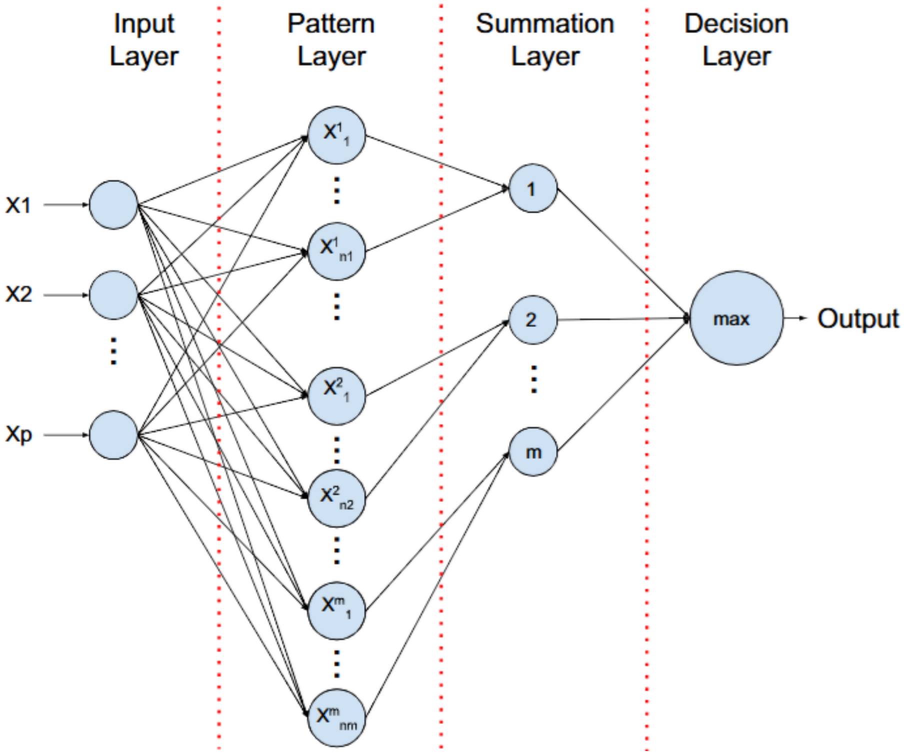


Figure 10 : ***The basic structure of a PNN (m is the number of classes) Link to Fig. Source.***

1. *Input Layer:* This layer receives the input features of the data. Each input feature corresponds to a neuron and passes input features directly to the next layer.
2. *Pattern Layer:* Each neuron in this layer contains one neuron for each training example. This layer computes a similarity score between the input and training examples using a kernel function (e.g., Gaussian).
3. *Summation Layer:* This layer aggregates the outputs from the pattern layer neurons for each class, summing the similarity scores.
4. *Output/Decision Layer:* The final layer provides the probability of the input vector belonging to each class. The class with the highest probability (or score) is selected as the predicted class.

[5] Specht, Donald F. "Probabilistic neural networks" Neural Networks (1990).

5.2 PNN Algorithm

PNN operates by estimating the probability density function (PDF) of each class using the Parzen window technique. The process can be broken down into the following steps:

1. **Training Phase.** The network stores the training samples and their corresponding class labels during training. No weights are learned; the training data directly serves as the model.

2. **Prediction Phase.**

   - Given training data $\{X_i\}_{i=1}^n$ from class C, the KDE for the class's probability density $P(x \mid C)$ at a point x is computed using Kernel function measuring similarity between x and X_i $(K(x, X_i))$. For an input x, we compute the similarity score with every training point using a kernel function:

     $$K(x, X_i) = \exp\left(-\frac{\|x - X_i\|^2}{2\sigma^2}\right),$$

     where σ is the bandwidth of the kernel and the Gaussian kernel function (mentioned above) measures the similarity between two vectors based on their Euclidean distance. The parameter (σ) controls the width of the Gaussian, determining how far away two points can be while still being considered similar.

   - Sum the kernel values for each class C to compute the score for the class:

     $$S_C = \sum_{i \in C} K(x, X_i).$$

   - Use the summation layer's scores to compute class probabilities (if required) and make a decision:

     $$\hat{C} = \arg\max_C S_C.$$

3. **Normalization Methods for Classification.**

   - *Normalization by Training Samples:* Normalize by dividing the kernel sum by the number of training samples in each class (n_C):

     $$P(C \mid x) = \frac{1}{n_C} \sum_{i \in C} K(x, X_i).$$

     This method ensures scores are independent of class sizes (however, the sum of probabilities across all classes may not equal 1). Suitable for balanced datasets or when probabilistic outputs are not required.

   - *Normalization by Total Kernel Values:* Normalize by dividing the kernel sum for a class by the total kernel sum across all classes:

     $$P(C \mid x) = \frac{\sum_{i \in C} K(x, X_i)}{\sum_j \sum_{i \in C_j} K(x, X_i)}.$$

     This procedure produces true posterior probabilities and ensures probabilities sum to 1. This method of normalizing is usually recommended for real-world applications where class proportions matter.

   - The Bayesian decision rule is applied in the Summation Layer and Decision Layer to classify the input:

     $$\text{Predict class } C \text{ if } P(C \mid x) > P(C' \mid x) \quad \forall\, C' \neq C,$$

     where $P(C \mid x)$ is the posterior probability of class C given the input x.

   - The outputs of the two normalization methods will match if classes have equal sizes and the test input lies in a region where all classes contribute equally.

5.3 Advantages and Limitations

Key advantages: Simple, easy to implement, and naturally handles non-linear decision boundaries. Probabilistic outputs enable confidence estimation and, therefore, any classification problem (small-to-medium datasets) with non-linear boundaries can utilize this method. It is very useful for Medical diagnosis applications where probabilistic outputs are valuable.

Major limitations: Computationally expensive for large datasets (storing all training points and computing similarity with each) and sensitive to the choice of kernel bandwidth (σ) and also the choice of kernel functions.

User Guide: (a) Carefully choose the normalization method based on your problem; (b) Experiment with different kernel bandwidths (σ) to achieve the best results (sometimes with different kernels); (c) For large datasets, consider approximations like subsampling or dimensionality reduction to speed up computations; and (d) Use of Number of Neighbors: This parameter determines how many data points from the training set are considered when calculating distances in the pattern layer.

5.4 SkewPNN for Imbalanced Data:

[6] the skew-normal distribution is used as a kernel function in PNN for situations where the data exhibits skewed distributions with a long tail. The skew-normal density, defined by its location parameter ξ, scale parameter σ, and skewness parameter α, is as follows[7]:

$$f(x;\xi,\sigma,\alpha) = \frac{2}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\xi)^2}{2\sigma^2}\right) \Phi\left\{\alpha\left(\frac{x-\xi}{\sigma}\right)\right\}; \quad \xi, x \in (-\infty,\infty),\ \sigma > 0, \tag{26}$$

where ξ determines the central point of the distribution, σ controls the spread of the distribution, α introduces asymmetry (shape parameter), and $\Phi(z)$ denotes the standard normal cumulative distribution function. For $\alpha = 0$, $f(x)$ is simply the normal distribution where $\alpha > 0$ denotes a longer tail to the right and $\alpha < 0$ denotes the longer tail to the left. The skew-normal kernel function is defined as:

$$K(x, X_i) = \exp\left(-\frac{\|x - X_i\|^2}{2\sigma^2}\right) \Phi\left(\frac{\alpha\,\|x - X_i\|}{\sigma}\right), \tag{27}$$

where x is the vector representing the new input sample, X_i is the vector representing the training sample from class c, σ represents the smoothing parameter and Φ denotes the cumulative distribution function (CDF) of the standard normal distribution. The hyperparameters in SkewPNN are σ (smoothing parameter) and α (skewness parameter), and adjusting α helps in handling data imbalance problems.

5.5 An Illustrative Numerical Example

We provide an illustrative example with numerical values demonstrating how the SkewPNN algorithm works for imbalanced datasets. For simplicity and easy computation, we consider the training sample of size 10, and Table 1 displays this example dataset with IR = 4.

- Class 0 (Majority Class): Clustered around [2,2].
- Class 1 (Minority Class): Clustered around [4,4].
- Our objective is to classify a test point [3.5,3.5].

Table 1: ***A toy data example with numerical values having ten observations (eight class 0 examples and two class 1 examples).***

Feature 1	2.0	2.2	2.4	2.6	2.8	1.8	1.9	2.3	4.0	4.1
Feature 2	2.0	2.2	2.4	2.6	2.8	1.8	2.1	2.1	4.0	4.1
Class	0	0	0	0	0	0	0	0	1	1

Given the input data, the pattern layer computes the similarity between the input vector and the training samples using the Gaussian kernel function in PNN

$$K(x, x_i) = \exp\left(-\frac{d_e^2}{2\sigma^2}\right) \tag{28}$$

and skew-normal kernel function in SkewPNN (with an extra adjusting factor or constant)

$$K(x, x_i) = 2\exp\left(-\frac{d_e^2}{2\sigma^2}\right)\Phi\left\{\alpha\left(\frac{d_e}{\sigma}\right)\right\}, \tag{29}$$

where d_e is the Euclidean distance between the test point ([3.5,3.5]) and the training points, Φ is the CDF of standard normal distribution, σ and α are the parameters controlling the scale and skewness of the kernels, respectively. It is important to note that for $\alpha = 0$, SkewPNN is simply PNN with a Gaussian kernel. The summation layer aggregates the outputs of the pattern layer neurons for each class. Finally, the output layer provides the probability of the input vector (test sample here) belonging to each class, and the class with the highest probability is chosen as the predicted class. This can be done in two ways: Normalization by dividing the kernel sum for a class by the number of training samples in each class or normalizing by dividing the kernel sum for a class by the total sum across all classes. When classes are equal in size, the output of the two normalization methods matches. The normalization by total kernel values produces true posterior probabilities and ensures probabilities sum to 1; therefore, we adopt it for this numerical example for its simplicity (c denotes class label):

$$P(c \mid x) = \frac{\sum_{i \in c} K\left(x, x_i\right)}{\sum_j \sum_{i \in c_j} K\left(x, x_i\right)}.$$

We work with this numerical data to provide a detailed calculation for the predicted class using PNN and SkewPNN. The computational results are depicted in Table 2, and we plotted the decision boundaries generated by both the models in Fig. 11 .

This illustrative example with numerical values finds that if we use the Gaussian kernel, the majority of class points dominate the probability distribution, leading to the classification of the test point as class 0 (the minority class points have a much smaller contribution). Now, instead of a Gaussian kernel, if we use a skew-normal kernel with the skewness parameter (here we choose $\alpha = -2$), it amplifies the influence of the minority class points, making their probability contribution more significant. This leads to a different classification result, favoring class 1 in this example, as observed in Table 2 and Fig. 11 . This example demonstrates how the SkewPNN works for imbalanced binary classification datasets.

5.6 Implementation in Python

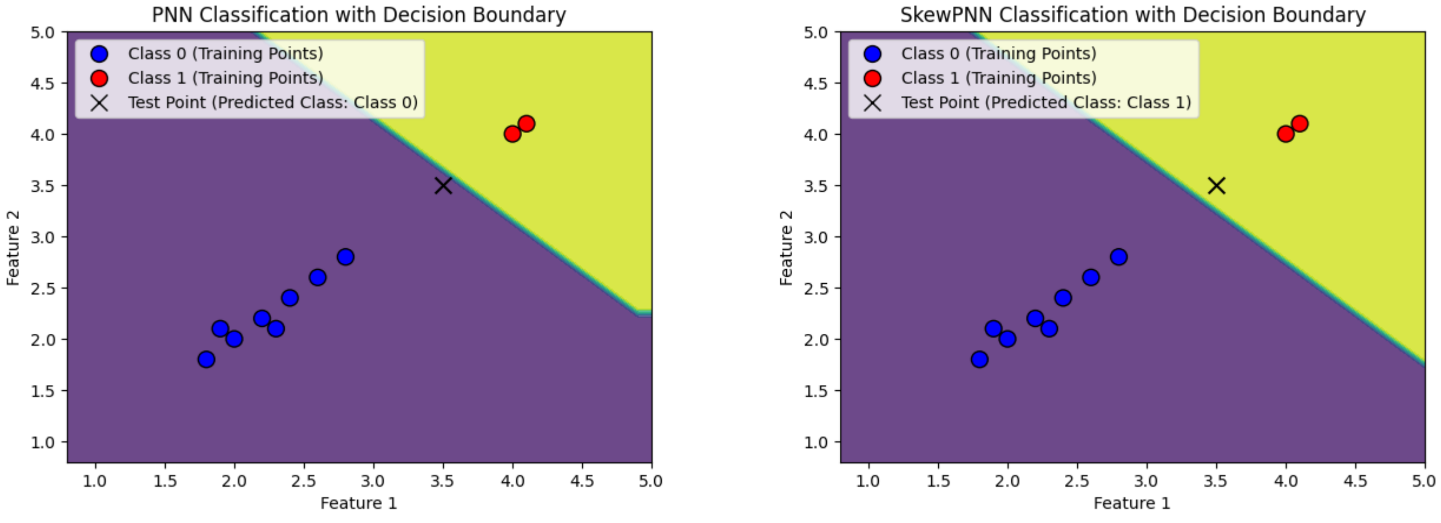
```
# PNN and SkewPNN on Toy Dataset with numerical values

import numpy as np
from scipy.stats import norm, skew
```

[6] Skew-Probabilistic Neural Networks for Learning from Imbalanced Data

[7] Azzalini, A. A class of distributions that includes the normal ones. Scandinavian journal of statistics, 1985.

Table 2: ***Values of different layers of PNN and SkewPNN calculated for the toy example. The networks assign the test input to the class with the highest estimated probability (highlighted in bold).***

x_i	$d = \|\|x_i - x\|\|$	Gaussian Kernel with $\sigma = 1$ (Eqn. 28)	Skew-Normal Kernel with $\sigma = 1,\ \alpha = -2$ (Eqn. 29)	Class Labels
[2.0, 2.0]	2.1213	0.1053	$2.3283E-06$	0
[2.2, 2.2]	1.8384	0.1845	$4.3552E-05$	0
[2.4, 2.4]	1.5556	0.2981	0.0005	0
[2.6, 2.6]	1.2727	0.4448	0.0048	0
[2.8, 2.8]	0.9899	0.6126	0.0292	0
[1.8, 1.8]	2.4041	0.0555	$8.4586E-08$	0
[1.9, 2.1]	2.1260	0.1043	$2.2102E-06$	0
[2.3, 2.1]	1.8439	0.1826	$4.1320E-05$	0
[4.0, 4.0]	0.7071	0.7788	0.1225	1
[4.1, 4.1]	0.8485	0.6976	0.0625	1
Summation layer	sum(kernel values for class 0)	1.9882	0.0347	Predicted class for test point (below)
	sum(kernel values for class 1)	1.4765	0.1850	
Output layer	Prob (class 0 | x)	$\frac{1.9882}{(1.9882+1.4765)}$=**0.5738**	0.1580	PNN predicts class 0
	Prob (class 1 | x)	0.4262	$\frac{0.1850}{(0.0347+0.1850)}$=**0.8420**	SkewPNN predicts class 1



(a)

(b)

Figure 11 : ***Plots of decision boundaries separating two classes (Class 0 and 1) using (a) PNN algorithm and (b) SkewPNN algorithm. The plot also depicts the predicted class label for the test point.***

```python
# Define Gaussian and Skew Normal Kernels
def gaussian_kernel(distance, sigma=1.0):
    return np.exp(-distance**2 / (2 * sigma**2))

def skew_normal_kernel(distance, omega=1.0, alpha=0.0):
    kdf = np.exp(-distance**2 / (2 * sigma**2))
    # pdf = norm.pdf(distance / omega)
    cdf = norm.cdf(alpha * (distance / omega))
    return 2 * kdf * cdf

# Training data: [feature1, feature2, class]
training_data = np.array([[2.0, 2.0, 0], [2.2, 2.2, 0], [2.4, 2.4, 0], [2.6, 2.6, 0],
[2.8, 2.8, 0], [1.8, 1.8, 0], [1.9, 2.1, 0], [2.3, 2.1, 0], [4.0, 4.0, 1], [4.1, 4.1, 1]])

# Query point
query_point = np.array([3.5, 3.5])

# Calculate distances and kernel values
sigma = 1.0
omega = 1.0
alpha = -2.0

results = []
for data_point in training_data:
    distance = np.linalg.norm(query_point - data_point[:2])
    gaussian_value = gaussian_kernel(distance, sigma)
    skew_value = skew_normal_kernel(distance, omega, alpha)
    results.append((data_point[2], distance, gaussian_value, skew_value))

# Aggregate results by class
class_0_gaussian = sum(row[2] for row in results if row[0] == 0)
class_1_gaussian = sum(row[2] for row in results if row[0] == 1)
class_0_skew = sum(row[3] for row in results if row[0] == 0)
class_1_skew = sum(row[3] for row in results if row[0] == 1)

print(class_0_gaussian, class_1_gaussian, class_0_skew, class_1_skew)

# Probability of the input belonging to each class
prob_class_0_gaussian = class_0_gaussian / (class_0_gaussian + class_1_gaussian)
prob_class_1_gaussian = class_1_gaussian / (class_0_gaussian + class_1_gaussian)
prob_class_0_skew = class_0_skew / (class_0_skew + class_1_skew)
prob_class_1_skew = class_1_skew / (class_0_skew + class_1_skew)

print(prob_class_0_gaussian, prob_class_1_gaussian, prob_class_0_skew, prob_class_1_skew)

# Predict the class with the highest probability
predicted_class_gaussian = 0 if prob_class_0_gaussian > prob_class_1_gaussian else 1
predicted_class_skew = 0 if prob_class_0_skew > prob_class_1_skew else 1

print(f"Predicted class: {predicted_class_gaussian}")
print(f"Predicted class: {predicted_class_skew}")

### Outputs ###

1.9882107098110808 1.4764771091424362 0.03472956560401427 0.18507655979594764
0.5738498859650809 0.42615011403491904 0.15800089984216742 0.8419991001578326
Predicted class: 0
Predicted class: 1
```

Python Code of this example is available on `https://github.com/ctanujit/Special-Topics`**.**