rbf-tarea

January 29, 2020

1 Funciones Base Radial

1.1 Radial Basis Functions, RBFs

Mariano Rivera

enero 2020

Sea f una función que puede ser expresada como suma de funciones base:

(1)

$$f(x) = \sum_{j=1}^{n} \beta_j \phi_j(x; \Theta_j) + \eta(x) \tag{1}$$

donde

- ϕ_j es la función base.
- Θ_j son los parámetros.
- β_j la contribución de la j-ésima función base (coeficiente).
- η es el residual (error de representación).

Existen muchas posibilidades entre las cuales podemos escoger las funciones base, típicamente se escoge una familia parametrizada de funciones: $\phi_j(x, \theta_j) = \phi(x, \Theta_j)$. Un tipo de familia de funciones base son las Funciones Base Radial. Estas son funciones en que pueden variar en la posición y en la escala. Por ejemplo

- Kernel Gaussiano

(2)

$$\phi(x, \Theta) = \exp\left(-(\theta_2 r)^2\right) \tag{2}$$

donde

(3)

$$r = \|x - \theta_1\|$$

en este caso, θ_1 define la posición y θ_2 la escala.

- Kernel Quadrático Inverso

(4)
$$\phi(x,\Theta) = \frac{1}{1+(\theta_2 r)^2}$$

- Kernel Multicuádrico

(5)
$$\phi(x,\Theta) = \sqrt{1+(\theta_2 r)^2}$$

- Kernel Multicuádrico Inverso

(5)
$$\phi(x,\Theta) = \frac{1}{\sqrt{1+(\theta_2 r)^2}}$$

- Kernel Función Tope (Bump Function).

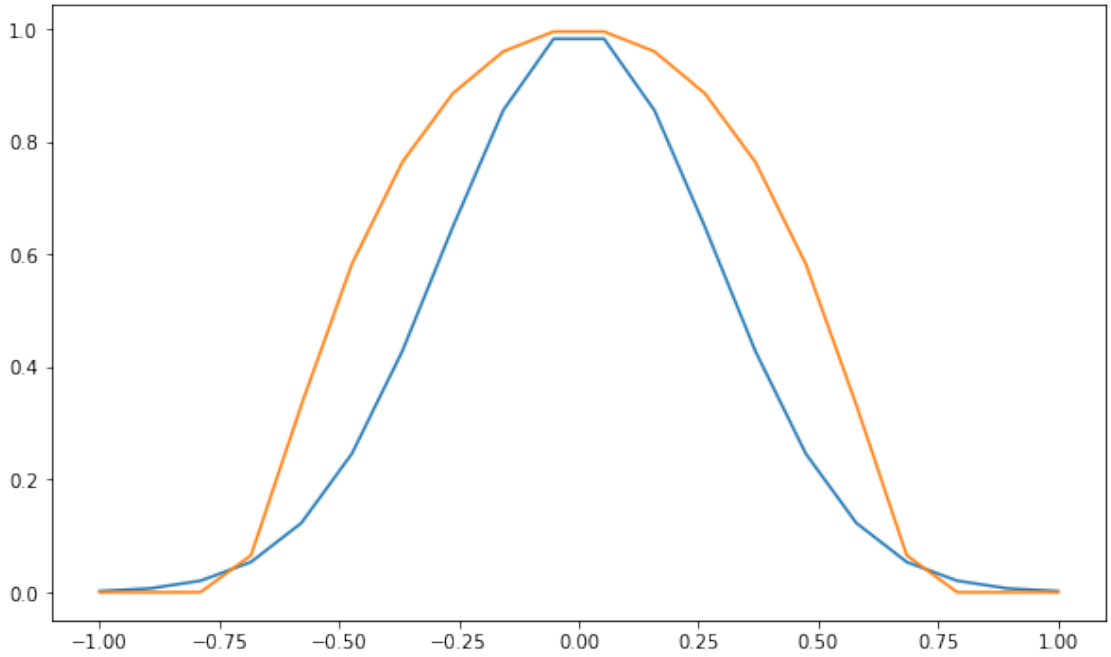
(6)
$$\phi(x,\Theta) = \begin{cases} \exp\left(-\frac{(\theta_2 r)^2}{1-(\theta_2 r)^2}\right) & (\theta_2 r)^2 > 1 \\ 0 & \text{otro} \end{cases}$$

Esta función tienen soporte finito

Las funciones anteriores son multivariadas $\phi : \mathbb{R}^n \to \mathbb{R}$, sin embargo se les denomina rediales porque para efectos de radio r (3) se comportan como univariadas; abusando de la notación tendríamos $\phi(r) : \mathbb{R} \to \mathbb{R}$.

[64]:
```
import numpy as np
import matplotlib.pyplot as plt

N=1
i = np.linspace(-N,N,20)

def rbf_gaussian(i, m=0, sigma=1):
    r, th = (i-m),  1./(2.*sigma)
    return np.exp(-(th*i)**2)

def rbf_bump(i, m=0, sigma=1):
    r, th = (i-m),  1./(2.*sigma)
    val = (th*i)**2
    msk = (val<1)
    print(1./sigma)
    return msk*np.exp(- val/ (1.-val))

plt.figure(figsize=(10,6))
plt.plot(i,rbf_gaussian(i, sigma=.2))
plt.plot(i,rbf_bump(i, sigma=.4))
plt.show()
```

```
2.5
```



1.2 Cálculo de los coeficientes de expansión β

Muy frecuentemente, se fija la escala y posición de los kernels para estimar aproximaciones de funciones. En tal caso, asumiendo que contamos con valores observados

(7)
$$y = [y_1, y_2, \ldots, y_m]^T$$

entonces haciendo

$$\Theta_{j,2} = 1/(2\sigma^2)$$

y **suponiendo equi-espaciadas las funciones base** $\delta = m/n$ hacemos

$$\Theta_{j,1} = \delta\,(j-1)$$

para $j = 1, 2, \ldots, n+1$. **Otra estrategia para colocar las funciones base es distribuirlas aleatoriamente en el dominio.**

Luego calculamos

(8)
$$\phi_{i,j} = \exp\left(-\theta_{2,j}\|y_i - \theta_{1,j}\|^2\right)$$

y ordenandolo como matriz

(9)
$$\Phi = [\phi_{i,j}]_{i=1,2,\ldots,mj=1,2,\ldots,n}$$

es decir, cada función base corresponde a una columna en la matriz Φ. La forma de calcular los coeficientes de la expansión es mediante mínimos cuadrados:

(10)
$$\min_{\beta} \|\Phi\beta - y\|^2$$

cuya solución se calcula con

(11)
$$\beta = (\Phi^\top \Phi)^{-1}\Phi^\top y$$

1.3 Optimización de las posiciones θ_1

Resolver para los parámetros de las funciones base (posición o escala) implica resolver problemas de optimización no lineal. A continuación veremos como resolver simultaneamente (11) para los coeficientes y posiciones; esta estrategia es generalizable a los parámetros de escala.

Ahora el problema de optimización lo podremos escribir como

(12)
$$\min_{\beta,\mu} L(\beta,\mu;y) = \sum_i \left[\sum_j \beta_j \phi(y,\mu_j) - y_i\right]^2 = \sum_i \left[\sum_j \beta_j \exp\left(-\frac{1}{2\sigma^2}(x_i-\mu_j)^2\right) - y_i\right]^2$$

La estrategia que seguiremos es realizar optimizaciones alternadas en las variables β y μ.

1.3.1 Algoritmo 1

Inicialización. Poner μ^0 y $k = 1$

Paso 1. Optimizar (12) con respecto a β asumiendo fija μ:

(13)
$$\beta^k = \arg\min_{\beta} \; L(\beta, \mu^{k-1}; y)$$

Este paso se calcula como en (11).

Paso 2. Optimizar (12) con respecto a μ asumiendo fija β^k:

(14)
$$\mu^k = \arg\min_{\mu} \; L(\beta^k, \mu; y)$$

En este paso es posible usar métodos de del tipo descenso de gradiente.

Iterar Hacer k = k+1 y mientras no se alcance convergencia ir al paso 1.

Nota. La convergencia se alcanza cuando ya no hay avance sustancial en la iteración; es decir cuando $\|\mu^k - \mu^{k-1}\| < \epsilon$.

Respecto al implementar el paso 2 del algoritmo arriba presentado, nos conformaremos en cada iteración con obtener una μ^k que mejore suficientemente el valor actual de la función objetivo; es decir, no implementaremos la optimización completa sino que nos conformaremos con conseguir que

(15)
$$L(\beta^k, \mu^k) < L(\beta^k, \mu^{k-1})$$

Para ello es posible usar un algoritmo tipo descenso, donde el gradiente con resecto a μ esta dado por

(15)
$$\nabla_\mu L(\beta, \mu) = \begin{bmatrix} \frac{\partial}{\partial \mu_1} L(\beta, \mu) \\ \frac{\partial}{\partial \mu_2} L(\beta, \mu) \\ \vdots \\ \frac{\partial}{\partial \mu_n} L(\beta, \mu) \end{bmatrix}$$

y cada parcial es de la forma

(16)
$$\frac{\partial L}{\partial \mu_l} = 2 \sum_i \left[\sum_j \beta_j \exp\left(-\frac{1}{2\sigma^2}(x_i - \mu_j)^2 \right) - y_i \right] \left[\beta_l \exp\left(-\frac{1}{2\sigma^2}(x_i - \mu_l)^2 \right) \right] \left[-\frac{1}{\sigma^2}(x_i - \mu_l) \right] [-1]$$

$$= \frac{2}{\sigma^2} \sum_i \left[\sum_j \beta_j \exp\left(-\frac{1}{2\sigma^2}(x_i - \mu_j)^2 \right) - y_i \right] \left[\beta_l \exp\left(-\frac{1}{2\sigma^2}(x_i - \mu_l)^2 \right) \right] (x_i - \mu_l)$$

1.4 Tarea

Resolver el problema dado por (11) para el caso en que y se obtienen como el renglón de una imagen de al menos 500 pixeles y se usan al menos 10 funciones base. Usar la versión estocástica de los algoritmos SGD, SGD-Momentum, NAG, ADAGRAD, ADADELTA y ADAM. Haga una comparación del desempeño.