Support vector machine: A quick look on cost function

20. Februar 2020

1 Hinge Loss Function

It is written as

$$\mathcal{H}\left(x, y, f\left(x\right)\right) = \begin{cases} 0, & \text{if } yf\left(x\right) \geq 1 \\ 1 - yf\left(x\right), & \text{otherwise} \end{cases} \tag{1}$$

with $x \in \mathbb{R}^n$ and $y \in \{-1, +1\}$ to be the training samples and their corresponding labels. Here, $f\left(x\right) \coloneqq \langle x \cdot w \rangle \in \{-1, +1\}$ is the predicted label of the training sample x. Here, $w \in \mathbb{R}^n$, with n and m defined in $\mathbb{N}$.

2 Objective Function

Based on (1), support vector machine (SVM) objective's function is written as

$$\min_{w} \left(\lambda \left\| w \right\|^2 + \sum_{i=1}^{n} \mathcal{H} \left(x, y, \overbrace{f\left(x\right)}^{\coloneqq \langle x \cdot w \rangle} \right) \right) \tag{2}$$

with $\lambda \in (0, +\infty)$.

3 Optimization

To optimize (2), we need its derivatives

$$\frac{\partial}{\partial w} \left(\min_{w} \left(\lambda \left\| w \right\|^2 + \sum_{i=1}^{n} \mathcal{H}\left(x, y, f\left(x\right)\right) \right) \right) \tag{3}$$

that can be taken separately as

$$\frac{\partial}{\partial w_k} \left(\lambda \left\| w \right\|^2 \right) = 2\lambda w_k \tag{4}$$

and

$$\frac{\partial}{\partial w_k}\left(\mathcal{H}\left(x, y, f\left(x\right)\right)\right) = \begin{cases} 0, & \text{if } y_i \left\langle x_i \cdot w_k \right\rangle \geq 1 \\ -y_i x_i, & \text{otherwise} \end{cases} \tag{5}$$

with $k \in \{1, \cdots, n\}$ leading us to the below stochastic gradient descent updation rules

$$w_k = w_k - \eta \left(2\lambda w_k - \left\langle x_i \cdot w_k \right\rangle\right) \tag{6}$$

if $y_i \left\langle x_i \cdot w_k \right\rangle < 1$ and otherwise

$$w_k = w_k - \eta \left(2\lambda w_k\right). \tag{7}$$

One writes the snippet python code corresponding to (6) and (7) as

```
def SVM_SGD(X,Y):
  w = np.zeros(len(X[0]))
  eta = 1
  epochs = 100000
  for epoch in range(1,n):
   for i, x in enumerate(X):
    if (Y[i]*np.dot(X[i], w)) < 1:
         w = w + eta * ( (X[i] * Y[i]) + (-2  *(1/epoch)* w) )
        else:
         w = w + eta * (-2  *(1/epoch)* w)
return w
```