Tim O'Connor

6038881588

oconnort@usc.edu

EE559 – Rajati – Summer 2024

Homework 7

1. For SVR, prove that $w_0 = y_j - \epsilon - \sum_{i \in S}(\lambda_i^+ - \lambda_i^-)\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$, where S is the set of indices of support vectors and $\boldsymbol{x}_j$ is a support vector at the *upper edge* of the ϵ-tube. (10 pts)

For $\boldsymbol{x}_j$ at the *upper edge*:

$$y_i - \hat{y}_i = \xi_i^+ + \epsilon = \epsilon \ (\xi_i^+ = 0 \ at\ edge)$$
$$y_j - \boldsymbol{w}^T\phi(\boldsymbol{x}_j) - w_0 = \epsilon$$
$$w_0 = y_j - \epsilon - \boldsymbol{w}^T\varphi(\boldsymbol{x}_j)$$
$$L(\boldsymbol{w}, w_0, \boldsymbol{\xi}^+, \boldsymbol{\xi}^-, \boldsymbol{\mu}^+, \boldsymbol{\mu}^-, \boldsymbol{\lambda}^+, \boldsymbol{\lambda}^-) = C\sum_{i=1}^{N}(\xi_i^+ + \xi_i^-) + \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|_2^2 - \sum_{i=1}^{N}(\mu_i^+\xi_i^+ - \mu_i^-\xi_i^-)$$
$$- \sum_{i \in S}\lambda_i^+(\epsilon + \hat{y}_i - y_i) - \sum_{i \in S}\lambda_i^-(\epsilon - \hat{y}_i + y_i)$$
$$\hat{y} = \boldsymbol{w}^T\varphi(\boldsymbol{x}) + w_0$$
$$\left.\frac{\partial L}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\boldsymbol{w}^*} = \boldsymbol{w} - \sum_{i=1}^{N}\lambda_i^+\frac{\partial}{\partial \boldsymbol{w}}\hat{y}_i + \sum_{i=1}^{N}\lambda_i^-\frac{\partial}{\partial \boldsymbol{w}}\hat{y}_i = \boldsymbol{0}$$
$$\boldsymbol{w} = \sum_{i=1}^{N}(\lambda_i^+ - \lambda_i^-)\varphi(\boldsymbol{x}_i) = \sum_{i \in S}(\lambda_i^+ - \lambda_i^-)\varphi(\boldsymbol{x}_i)$$
$$w_0 = y_j - \epsilon - \sum_{i \in S}(\lambda_i^+ - \lambda_i^-)\varphi(\boldsymbol{x}_i)\,\varphi(\boldsymbol{x}_j)$$

2. For the following classification problem, design a single-layer perceptron, by using the multiclass Perceptron update rule. (20 pts)

$$\mathcal{D}_{\omega 1} = \left\{\boldsymbol{x_1} = \begin{bmatrix}1\\1\end{bmatrix}\right\}$$
$$\mathcal{D}_{\omega 2} = \left\{\boldsymbol{x_2} = \begin{bmatrix}1\\-1\end{bmatrix}\right\}$$
$$\mathcal{D}_{\omega 3} = \left\{\boldsymbol{x_3} = \begin{bmatrix}-1\\-1\end{bmatrix}\right\}$$

Use one-hot encoding for classes, for example, ω_3 should be represented using the following vector

$$\boldsymbol{y}_3 = \begin{bmatrix}-1\\-1\\1\end{bmatrix}$$

(a) Start with $\boldsymbol{W}(0) = \boldsymbol{0}_{2\times 3}$ and choose $\eta(i) = 0.5$ in $\boldsymbol{W}(i+1) = \boldsymbol{W}(i) + \eta(i)\boldsymbol{x}(i)\boldsymbol{e}^T$. Do not use the augmented space and assume that the biases are always zero (no update for biases).

Show multiple steps of your algorithm. Does it converge? Why? It is alright if you use a computer or calculator to perform the matrix calculation, but you should write down all the steps, and should not write a computer program to yield the final results.

Epoch 1:

$$\boldsymbol{W}(0) = \underset{2\times3}{\boldsymbol{0}}$$

Step 1: $\boldsymbol{x}(1) = \boldsymbol{x}_1 = [1 \quad 1]^T, \boldsymbol{y}(1) = \boldsymbol{y}_1 = [1 \quad -1 \quad -1]^T, \boldsymbol{W}(1) = \boldsymbol{W}(0) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

$$\boldsymbol{e}(1) = \boldsymbol{y}(1) - f\left(\boldsymbol{W}^T(1)\boldsymbol{x}(1)\right)$$

$$\rightarrow \boldsymbol{W}^T(1)\boldsymbol{x}(1) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} < \boldsymbol{w}_1^T(1), \boldsymbol{x}(1) > \\ < \boldsymbol{w}_2^T(1), \boldsymbol{x}(1) > \\ < \boldsymbol{w}_3^T(1), \boldsymbol{x}(1) > \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$f(x) = 2H(x) - 1, f(\boldsymbol{W}^T\boldsymbol{x}) = \boldsymbol{y}_1 \; if \; \boldsymbol{w}_1^T\boldsymbol{x} = \cdots = \boldsymbol{w}_S^T\boldsymbol{x}$$

$$\rightarrow f\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right) = \boldsymbol{y}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\boldsymbol{e}(1) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \\ -2 \end{bmatrix}$$

$$\boldsymbol{W}(2) = \underset{2\times3}{\boldsymbol{0}} + 0.5 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot [0 \quad -2 \quad -2] = \begin{bmatrix} 0 & -1 & -1 \\ 0 & -1 & -1 \end{bmatrix}$$

Step 2: $\boldsymbol{x}(2) = \boldsymbol{x}_2 = [1 \quad -1]^T, \boldsymbol{y}(2) = \boldsymbol{y}_2 = [-1 \quad 1 \quad -1]^T$

$$\boldsymbol{e}(2) = \boldsymbol{y}(2) - f\left(\boldsymbol{W}^T(2)\boldsymbol{x}(2)\right)$$

$$\rightarrow \boldsymbol{W}^T(2)\boldsymbol{x}(2) = \begin{bmatrix} 0 & -1 & -1 \\ 0 & -1 & -1 \end{bmatrix}^T \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} < \boldsymbol{w}_1^T(2), \boldsymbol{x}(2) > \\ < \boldsymbol{w}_2^T(2), \boldsymbol{x}(2) > \\ < \boldsymbol{w}_3^T(2), \boldsymbol{x}(2) > \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$f(x) = 2H(x) - 1, f(\boldsymbol{W}^T\boldsymbol{x}) = \boldsymbol{y}_1 \; if \; \boldsymbol{w}_1^T\boldsymbol{x} = \cdots = \boldsymbol{w}_S^T\boldsymbol{x}$$

$$\rightarrow f\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right) = \boldsymbol{y}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\boldsymbol{e}(2) = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ -2 \end{bmatrix}$$

$$\boldsymbol{W}(2) = \begin{bmatrix} 0 & -1 & -1 \\ 0 & -1 & 01 \end{bmatrix} + 0.5 \begin{bmatrix} 1 \\ -1 \end{bmatrix} \cdot [-2 \quad 0 \quad -2]$$

$$= \begin{bmatrix} 0 & -1 & -1 \\ 0 & -1 & -1 \end{bmatrix} + \begin{bmatrix} -1 & 0 & -1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & -1 & -2 \\ 1 & -1 & 0 \end{bmatrix}$$

Step 3: $\boldsymbol{x}(3) = \boldsymbol{x}_2 = [-1 \quad -1]^T, \boldsymbol{y}(2) = \boldsymbol{y}_2 = [-1 \quad -1 \quad 1]^T$

$$\boldsymbol{e}(3) = \boldsymbol{y}(3) - f\left(\boldsymbol{W}^T(3)\boldsymbol{x}(3)\right)$$

$$\rightarrow \boldsymbol{W}^T(3)\boldsymbol{x}(3) = \begin{bmatrix} -1 & -1 & -2 \\ 1 & -1 & 0 \end{bmatrix}^T \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} < \boldsymbol{w}_1^T(3), \boldsymbol{x}(3) > \\ < \boldsymbol{w}_2^T(3), \boldsymbol{x}(3) > \\ < \boldsymbol{w}_3^T(3), \boldsymbol{x}(3) > \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix}$$

$$f(x) = 2H(x) - 1, f(\boldsymbol{W}^T\boldsymbol{x}) = \boldsymbol{y}_1 \ if \ \boldsymbol{w}_1^T\boldsymbol{x} = \cdots = \boldsymbol{w}_S^T\boldsymbol{x}$$

$$\rightarrow f\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right) = \boldsymbol{y}_1 = \begin{bmatrix} 1 \\ 1 \\ 11 \end{bmatrix}$$

$$\boldsymbol{e}(3) = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \\ 0 \end{bmatrix}$$

$$\boldsymbol{W}(3) = \begin{bmatrix} -1 & -1 & -2 \\ 1 & -1 & 0 \end{bmatrix} + 0.5 \begin{bmatrix} -1 \\ -1 \end{bmatrix} \cdot [-2 \quad -2 \quad 0]$$

$$= \begin{bmatrix} -1 & -1 & -2 \\ 1 & -1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & -2 \\ 2 & 0 & 0 \end{bmatrix}$$

Epoch 2:

$$\boldsymbol{W}(0) = \begin{bmatrix} 0 & 0 & -2 \\ 2 & 0 & 0 \end{bmatrix}$$

Step 1: $\boldsymbol{x}(1) = \boldsymbol{x}_1 = [1 \quad 1]^T, \boldsymbol{y}(1) = \boldsymbol{y}_1 = [1 \quad -1 \quad -1]^T, \boldsymbol{W}(1) = \boldsymbol{W}(0) = \begin{bmatrix} 0 & 0 & -2 \\ 2 & 0 & 0 \end{bmatrix}$

$$\boldsymbol{e}(1) = \boldsymbol{y}(1) - f\big(\boldsymbol{W}^T(1)\boldsymbol{x}(1)\big)$$

$$\rightarrow \boldsymbol{W}^T(1)\boldsymbol{x}(1) = \begin{bmatrix} 0 & 0 & -2 \\ 2 & 0 & 0 \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} < \boldsymbol{w}_1^T(1), \boldsymbol{x}(1) > \\ < \boldsymbol{w}_2^T(1), \boldsymbol{x}(1) > \\ < \boldsymbol{w}_3^T(1), \boldsymbol{x}(1) > \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ -2 \end{bmatrix}$$

$$f(x) = 2H(x) - 1, f(\boldsymbol{W}^T\boldsymbol{x}) = \boldsymbol{y}_1 \ if \ \boldsymbol{w}_1^T\boldsymbol{x} = \cdots = \boldsymbol{w}_S^T\boldsymbol{x}$$

$$\rightarrow f\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

$$\boldsymbol{e}(1) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \\ 0 \end{bmatrix}$$

$$\boldsymbol{W}(2) = \begin{bmatrix} 0 & 0 & -2 \\ 2 & 0 & 0 \end{bmatrix} + 0.5 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot [0 \quad -2 \quad 0]$$

$$= \begin{bmatrix} 0 & 0 & -2 \\ 2 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 & -2 \\ 2 & -1 & 0 \end{bmatrix}$$

Step 2: $\boldsymbol{x}(2) = \boldsymbol{x}_2 = [1 \quad -1]^T, \boldsymbol{y}(2) = \boldsymbol{y}_2 = [-1 \quad 1 \quad -1]^T$

$$\boldsymbol{e}(2) = \boldsymbol{y}(2) - f\big(\boldsymbol{W}^T(2)\boldsymbol{x}(2)\big)$$

$$\rightarrow \boldsymbol{W}^T(2)\boldsymbol{x}(2) = \begin{bmatrix} 0 & -1 & -2 \\ 2 & -1 & 0 \end{bmatrix}^T \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} < \boldsymbol{w}_1^T(2), \boldsymbol{x}(2) > \\ < \boldsymbol{w}_2^T(2), \boldsymbol{x}(2) > \\ < \boldsymbol{w}_3^T(2), \boldsymbol{x}(2) > \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ -2 \end{bmatrix}$$

$$f(x) = 2H(x) - 1, f(\boldsymbol{W}^T\boldsymbol{x}) = \boldsymbol{y}_1 \ if \ \boldsymbol{w}_1^T\boldsymbol{x} = \cdots = \boldsymbol{w}_S^T\boldsymbol{x}$$

$$\rightarrow f\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right) = \boldsymbol{y}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

$$\boldsymbol{e}(2)=\begin{bmatrix}-1\\1\\-1\end{bmatrix}-\begin{bmatrix}-1\\1\\-1\end{bmatrix}=\begin{bmatrix}0\\0\\0\end{bmatrix}$$
$$\boldsymbol{W}(3)=\begin{bmatrix}0&-1&-2\\2&-1&0\end{bmatrix}+0.5\begin{bmatrix}1\\-1\end{bmatrix}\cdot\begin{bmatrix}0&0&0\end{bmatrix}$$
$$=\begin{bmatrix}0&-1&-2\\2&-1&0\end{bmatrix}$$

Step 3: $\boldsymbol{x}(3)=\boldsymbol{x}_2=[-1\quad -1]^T, \boldsymbol{y}(2)=\boldsymbol{y}_2=[-1\quad -1\quad 1]^T$

$$\boldsymbol{e}(3)=\boldsymbol{y}(3)-f\left(\boldsymbol{W}^T(3)\boldsymbol{x}(3)\right)$$
$$\rightarrow \boldsymbol{W}^T(3)\boldsymbol{x}(3)=\begin{bmatrix}0&-1&-2\\2&-1&0\end{bmatrix}^T\begin{bmatrix}-1\\-1\end{bmatrix}=\begin{bmatrix}<\boldsymbol{w}_1^T(3),\boldsymbol{x}(3)>\\<\boldsymbol{w}_2^T(3),\boldsymbol{x}(3)>\\<\boldsymbol{w}_3^T(3),\boldsymbol{x}(3)>\end{bmatrix}=\begin{bmatrix}-2\\2\\2\end{bmatrix}$$
$$f(x)=2H(x)-1, f(\boldsymbol{W}^T\boldsymbol{x})=\boldsymbol{y}_1\ if\ \boldsymbol{w}_1^T\boldsymbol{x}=\cdots=\boldsymbol{w}_S^T\boldsymbol{x}$$
$$\rightarrow f\left(\begin{bmatrix}0\\0\\0\end{bmatrix}\right)=\boldsymbol{y}_1=\begin{bmatrix}-1\\1\\1\end{bmatrix}$$
$$\boldsymbol{e}(3)=\begin{bmatrix}-1\\-1\\1\end{bmatrix}-\begin{bmatrix}-1\\1\\1\end{bmatrix}=\begin{bmatrix}0\\-2\\0\end{bmatrix}$$
$$\boldsymbol{W}(4)=\begin{bmatrix}0&-1&-2\\2&-1&0\end{bmatrix}+0.5\begin{bmatrix}-1\\-1\end{bmatrix}\cdot\begin{bmatrix}0&-2&0\end{bmatrix}$$
$$=\begin{bmatrix}0&-1&-2\\2&-1&0\end{bmatrix}+\begin{bmatrix}0&1&0\\0&1&0\end{bmatrix}$$
$$=\begin{bmatrix}0&0&-2\\2&0&0\end{bmatrix}$$

Final weight at end of epoch 2 equals final weight at end of epoch 1.

$$(1): \eta(i)=\frac{1}{2}>0$$
$$(2): \sum_{i=1}^{\infty}\eta(i)=\frac{\infty}{2}=\infty$$
$$(3): \lim_{m\to\infty}\frac{\sum_{i=1}^{m}\eta^2(i)}{\left(\sum_{i=1}^{m}\eta(i)\right)^2}=\lim_{m\to\infty}\frac{\frac{m}{4}}{\frac{m^2}{4}}=\lim_{m\to\infty}\frac{1}{m}=0$$
$$Algorithm\ converges$$

(b) Now redo the previous procedure, but this time deal with each of the columns of $\boldsymbol{W}$ as one perceptron, i.e. update each column (the weight associated with a linear discriminant) separately, for example the first iteration becomes:

$$
\begin{aligned}
\boldsymbol{W}(0) &= [\boldsymbol{w}_1(0) \quad \boldsymbol{w}_2(0) \quad \boldsymbol{w}_3(0)] \\
\boldsymbol{w}_1(1) &= \boldsymbol{w}_1(0) + \eta e_1 \boldsymbol{x}(1) \\
\boldsymbol{w}_2(1) &= \boldsymbol{w}_2(0) + \eta e_2 \boldsymbol{x}(1) \\
\boldsymbol{w}_3(1) &= \boldsymbol{w}_3(0) + \eta e_3 \boldsymbol{x}(1)
\end{aligned}
$$

Where $e_i = y_{i1} - sign(\boldsymbol{w}_i(0)^T \boldsymbol{x}(1))$ is the difference between the i^{th} element of $\boldsymbol{y}_1$ (the targer vector for $\boldsymbol{x}_1$) and the output of the i^{th} neuron/linear discriminant $\boldsymbol{w}_i(1)$

Perform two epochs only. This is essentially to make you observe that a multicategory Perceptron algorithm is based on multiple binary problems.

Epoch 1:

Step 1: $\boldsymbol{x} = \boldsymbol{x}_1, \boldsymbol{y} = \boldsymbol{y}_1$

$$
\begin{aligned}
e_1 &= 1 - sign(0) = 0 \\
e_2 &= -1 - sign(0) = -2 \\
e_3 &= -1 - sign(0) = -2 \\
\boldsymbol{w}_1(1) &= \boldsymbol{w}_1(0) + \eta e_1 \boldsymbol{x}(1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.5(0) \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\boldsymbol{w}_2(1) &= \boldsymbol{w}_2(0) + \eta e_2 \boldsymbol{x}(1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.5(-2) \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \\
\boldsymbol{w}_3(1) &= \boldsymbol{w}_3(0) + \eta e_3 \boldsymbol{x}(1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.5(-2) \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}
\end{aligned}
$$

Step 2: $\boldsymbol{x} = \boldsymbol{x}_2, \boldsymbol{y} = \boldsymbol{y}_2$

$$
\begin{aligned}
e_1 &= -1 - sign\left([0 \quad 0] \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = -2 \\
e_2 &= 1 - sign\left([-1 \quad -1] \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = 0 \\
e_3 &= -1 - sign\left([-1 \quad -1] \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = -2 \\
\boldsymbol{w}_1(2) &= \boldsymbol{w}_1(1) + \eta e_1 \boldsymbol{x}(1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.5(-2) \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \\
\boldsymbol{w}_2(2) &= \boldsymbol{w}_2(1) + \eta e_2 \boldsymbol{x}(1) = \begin{bmatrix} -1 \\ -1 \end{bmatrix} + 0.5(0) \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \\
\boldsymbol{w}_3(2) &= \boldsymbol{w}_3(1) + \eta e_3 \boldsymbol{x}(1) = \begin{bmatrix} -1 \\ -1 \end{bmatrix} + 0.5(-2) \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}
\end{aligned}
$$

Step 3: $\boldsymbol{x} = \boldsymbol{x}_3, \boldsymbol{y} = \boldsymbol{y}_3$

$$
\begin{aligned}
e_1 &= -1 - sign\left([-1 \quad 1] \cdot \begin{bmatrix} -1 \\ -1 \end{bmatrix}\right) = -1 - sign(0) = -2 \\
e_2 &= -1 - sign\left([-1 \quad -1] \cdot \begin{bmatrix} -1 \\ -1 \end{bmatrix}\right) = -1 - sign(2) = -2 \\
e_3 &= 1 - sign\left([-2 \quad 0] \cdot \begin{bmatrix} -1 \\ -1 \end{bmatrix}\right) = 1 - sign(2) = 0 \\
\boldsymbol{w}_1(3) &= \boldsymbol{w}_1(2) + \eta e_1 \boldsymbol{x}(2) = \begin{bmatrix} -1 \\ 1 \end{bmatrix} + 0.5(-2) \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}
\end{aligned}
$$

$$\boldsymbol{w}_2(3) = \boldsymbol{w}_2(2) + \eta e_2 \boldsymbol{x}(2) = \begin{bmatrix} -1 \\ -1 \end{bmatrix} + 0.5(-2)\begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\boldsymbol{w}_3(3) = \boldsymbol{w}_3(2) + \eta e_3 \boldsymbol{x}(2) = \begin{bmatrix} -2 \\ 0 \end{bmatrix} + 0.5(0)\begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

Epoch 1: $\boldsymbol{w}_1(0) = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \boldsymbol{w}_2(0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \boldsymbol{w}_3(0) = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$

Step 1: $\boldsymbol{x} = \boldsymbol{x}_1, \boldsymbol{y} = \boldsymbol{y}_1$

$$e_1 = 1 - sign\left([0 \quad 2] \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) = 1 - sign(2) = 0$$
$$e_2 = -1 - sign\left([0 \quad 0] \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) = -1 - sign(0) = -2$$
$$e_3 = -1 - sign\left([-2 \quad 0] \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) = -1 - sign(-2) = 0$$
$$\boldsymbol{w}_1(1) = \boldsymbol{w}_1(0) + \eta e_1 \boldsymbol{x}(1) = \begin{bmatrix} 0 \\ 2 \end{bmatrix} + 0.5(0)\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$
$$\boldsymbol{w}_2(1) = \boldsymbol{w}_2(0) + \eta e_2 \boldsymbol{x}(1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.5(-2)\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$
$$\boldsymbol{w}_3(1) = \boldsymbol{w}_3(0) + \eta e_3 \boldsymbol{x}(1) = \begin{bmatrix} -2 \\ 0 \end{bmatrix} + 0.5(0)\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

Step 2: $\boldsymbol{x} = \boldsymbol{x}_2, \boldsymbol{y} = \boldsymbol{y}_2$

$$e_1 = -1 - sign\left([0 \quad 2] \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = -1 - sign(-2) = 0$$
$$e_2 = 1 - sign\left([-1 \quad -1] \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = 1 - sign(0) = 0$$
$$e_3 = -1 - sign\left([-2 \quad 0] \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = -1 - sign(-2) = 0$$
$$\boldsymbol{w}_1(2) = \boldsymbol{w}_1(1) + \eta e_1 \boldsymbol{x}(1) = \begin{bmatrix} 0 \\ 2 \end{bmatrix} + 0.5(0)\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$
$$\boldsymbol{w}_2(2) = \boldsymbol{w}_2(1) + \eta e_2 \boldsymbol{x}(1) = \begin{bmatrix} -1 \\ -1 \end{bmatrix} + 0.5(0)\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$
$$\boldsymbol{w}_3(2) = \boldsymbol{w}_3(1) + \eta e_3 \boldsymbol{x}(1) = \begin{bmatrix} -2 \\ 0 \end{bmatrix} + 0.5(-2)\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

Step 3: $\boldsymbol{x} = \boldsymbol{x}_3, \boldsymbol{y} = \boldsymbol{y}_3$

$$e_1 = -1 - sign\left([0 \quad 2] \cdot \begin{bmatrix} -1 \\ -1 \end{bmatrix}\right) = -1 - sign(-2) = 0$$
$$e_2 = -1 - sign\left([-1 \quad -1] \cdot \begin{bmatrix} -1 \\ -1 \end{bmatrix}\right) = -1 - sign(2) = -2$$
$$e_3 = 1 - sign\left([-2 \quad 0] \cdot \begin{bmatrix} -1 \\ -1 \end{bmatrix}\right) = 1 - sign(2) = 0$$
$$\boldsymbol{w}_1(3) = \boldsymbol{w}_1(2) + \eta e_1 \boldsymbol{x}(2) = \begin{bmatrix} 0 \\ 2 \end{bmatrix} + 0.5(0)\begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$
$$\boldsymbol{w}_2(3) = \boldsymbol{w}_2(2) + \eta e_2 \boldsymbol{x}(2) = \begin{bmatrix} -1 \\ -1 \end{bmatrix} + 0.5(-2)\begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\boldsymbol{w}_3(3) = \boldsymbol{w}_3(2) + \eta e_3 \boldsymbol{x}(2) = \begin{bmatrix} -2 \\ 0 \end{bmatrix} + 0.5(0)\begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$
$$Final\ \boldsymbol{w}_1 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \boldsymbol{w}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \boldsymbol{w}_3 = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

3. Consider the two classes of patterns that are shown in the figure below. Design a multilayer neural network with the following architecture to distinguish these categories (30 pts)

Assuming the patterns are:

$$\begin{matrix} x_1 & x_2 \\ x_3 & x_4 \end{matrix} \rightarrow \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} : x_i \in \{0,1\}$$

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \ in\ \omega_1, \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \ in\ \omega_2$$

For the first hidden layer

$$\begin{aligned} \boldsymbol{w}_1^{(1)} &= [1 \quad 1 \quad 0 \quad 0]^T \\ \boldsymbol{w}_2^{(1)} &= [0 \quad 0 \quad 1 \quad 1]^T \end{aligned}$$

For each neuron in the hidden layer:

$$\begin{aligned} \boldsymbol{w}^{T}{}_1^{(1)} \boldsymbol{x} + w_{10}^{(1)} &= x_1 + x_2 + w_{10}^{(1)} = x_1 + x_2 - 1 \rightarrow x_1 \wedge x_2 \\ \boldsymbol{w}^{T}{}_2^{(1)} \boldsymbol{x} + w_{20}^{(1)} &= x_1 + x_2 + w_{20}^{(1)} = x_3 + x_4 - 1 \rightarrow x_3 \wedge x_4 \end{aligned}$$

Because inputs are binary and we desire an output to be binary, use step function as activation function, producing these inputs to the final hidden layer.

$$\begin{bmatrix} H(x_1 + x_2 - 1) \\ H(x_3 + x_4 - 1) \end{bmatrix}$$

With weight:

$$\boldsymbol{w}_1^{(2)} = [1 \quad 1]^T$$

Producing the output:

$$H(x_1 + x_2 - 1) + H(x_3 + x_4 - 1) =$$

Again, using the Heaviside step function as the activation function, we have the final output.

$$\begin{aligned} &y = H\big(H(x_1 + x_2 - 1) + H(x_3 + x_4 - 1)\big) \\ &y = 0 \rightarrow \boldsymbol{x} \in \omega_1 \\ &y = 1 \rightarrow \boldsymbol{x} \in \omega_2 \end{aligned}$$

4. Programming Assignment: Parkinsons Telemonitoring

(a) Download the Parkinsons Telemonitoring Data Set from: http://archive.ics.scu.edu/ml/datasets/Parkinsons+Telemonitoring .Choose 70% of the data randomly as the training set.

(b) Use metric learning with Gaussian kernels to estimate each of the outputs motor UPDRS and total UPDRS from the features. As metric leaning uses a low dimensional transformation of the features except the non-predictive feature subject#, use 5-fold cross-validation to decide the number of components form $M = 5,10,15, p$, where p is the number of all predictive features you can use. Initialize the linear transformation with PCA features for $M = 5, 10, 15$ and with original features for $M = p$. This corresponds to setting intit as (default='auta'). Remember to standardize your features. Report the R^2 on training and test sets for each of the outputs. (30 pts)

(c) Use sklearn's neural network implementation to train a neural network with two outputs that predicts motor UPDRS and total UPDRS. Use a single layer. You are responsible to determine other architectural parameters of the network, including the number of neurons in the hidden and output layers, method of optimization, type of activation functions, and the L2 "regularization" parameter etc. You should determine the design parameters via trial and error, by testing your trained network on the test set and choosing the architecture that yields the smallest test error. For this part, set early-stopping=False. Remember to standardize your features. Report your R^2 on both training and test sets. (20 pts)

(d) Use the design parameters that you chose in the first part and train a neural network, but this time set early-stopping=True. Research what early stopping is, and compare the performance of you network on the test set with the previous network. You can leave the validation-fraction as the default (0.1) or change it to see whether you can obtain a better model. Remember to standardize your features. Report your R^2 on both training and test sets. (10 pts)

Note: there are a lot of design parameters in a neural network. If you are not sure how they work, just set them as the default of sklearn, but if you use them masterfully, you can have better models.

EE559-HW7-code-oconnort-6038881588

July 22, 2024

Tim O'Connor

603-888-1588

oconnort@usc.edu

EE559 - Rajati - Summer 2024

HW 7 - Programming Part

4. Programming Assignment: Parkinsons Telemonitoring

(a) Download the Parkinsons Telemonitoring Data Set from: http://archive.ics.scu.edu/ml/datasets/Parkinsons+Telemonitoring .Choose 70% of the data randomly as the training set.

(b) Use metric learning with Gaussian kernels to estimate each of the outputs motor UPDRS and total UPDRS from the features. As metric leaning uses a low dimensional transformation of the features except the non-predictive feature subject#, use 5-fold cross-validation to decide the number of components form M=5,10,15,p, where p is the number of all predictive features you can use. Initialize the linear transformation with PCA features for M = 5,10,15 and with original features for M=p. This corresponds to setting intit as (default='auto'). Remember to standardize your features. Report the R^2 on training and test sets for each of the outputs. (30 pts)

[92]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.kernel_ridge import KernelRidge
from sklearn.metrics import r2_score
from metric_learn import MLKR
from joblib import Parallel, delayed

path = './parkinsons_updrs.data'

data = pd.read_csv(path)

X = data.drop(columns=['subject#', 'motor_UPDRS', 'total_UPDRS'])
y_motor = data['motor_UPDRS']
y_total = data['total_UPDRS']
```

```
X_train, X_test, y_motor_train, y_motor_test, y_total_train, y_total_test =␣
↪train_test_split(X, y_motor, y_total, test_size=0.3, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

kf = KFold(n_splits=5, shuffle=True, random_state=42)

n = 5 #originally tried to run all of the possible components in a loop, but it␣
↪was too computationally expensive, 5 was able to finish.

pca_transformations = PCA(n_components=n).fit(X_train_scaled)

def process_fold(train_i, value_i, n, X_train_scaled, y_motor_train,␣
↪y_total_train):
    X_train_kf, X_value_kf = X_train_scaled[train_i], X_train_scaled[value_i]
    y_motor_train_kf, y_motor_value_kf = y_motor_train.iloc[train_i],␣
↪y_motor_train.iloc[value_i]
    y_total_train_kf, y_total_value_kf = y_total_train.iloc[train_i],␣
↪y_total_train.iloc[value_i]

    X_motor_train_PCA = pca_transformations.transform(X_train_kf)
    X_motor_test_PCA = pca_transformations.transform(X_value_kf)

    mlkr_motor = MLKR(n_components=n, init='pca')
    mlkr_motor.fit(X_motor_train_PCA, y_motor_train_kf)
    X_motor_train_MLKR = mlkr_motor.transform(X_motor_train_PCA)
    X_motor_test_MLKR = mlkr_motor.transform(X_motor_test_PCA)

    gauss_kernel_motor = KernelRidge(kernel='rbf')
    gauss_kernel_motor.fit(X_motor_train_MLKR, y_motor_train_kf)

    y_train_pred_motor = gauss_kernel_motor.predict(X_motor_train_MLKR)
    y_test_pred_motor = gauss_kernel_motor.predict(X_motor_test_MLKR)

    r2_train_motor = r2_score(y_motor_train_kf, y_train_pred_motor)
    r2_test_motor = r2_score(y_motor_value_kf, y_test_pred_motor)

    X_total_train_PCA = pca_transformations.transform(X_train_kf)
    X_total_test_PCA = pca_transformations.transform(X_value_kf)

    mlkr_total = MLKR(n_components=n, init='pca')
    mlkr_total.fit(X_total_train_PCA, y_total_train_kf)
    X_total_train_MLKR = mlkr_total.transform(X_total_train_PCA)
    X_total_test_MLKR = mlkr_total.transform(X_total_test_PCA)
```

```
    gauss_kernel_total = KernelRidge(kernel='rbf')
    gauss_kernel_total.fit(X_total_train_MLKR, y_total_train_kf)

    y_train_pred_total = gauss_kernel_total.predict(X_total_train_MLKR)
    y_test_pred_total = gauss_kernel_total.predict(X_total_test_MLKR)

    r2_train_total = r2_score(y_total_train_kf, y_train_pred_total)
    r2_test_total = r2_score(y_total_value_kf, y_test_pred_total)

    return (n, r2_train_motor, r2_test_motor, r2_train_total, r2_test_total)

def process_m(n, kf, X_train_scaled, y_motor_train, y_total_train):
    results = Parallel(n_jobs=-1)(
        delayed(process_fold)(train_i, value_i, n, X_train_scaled,␣
 ↪y_motor_train, y_total_train)
        for train_i, value_i in kf.split(X_train_scaled)
    )

    return results

parallel_results = Parallel(n_jobs=-1)(
    delayed(process_m)(n, kf, X_train_scaled, y_motor_train, y_total_train) for␣
 ↪n in [n]
)

for result in parallel_results:
    for fold_result in result:
        print(f"Components: {fold_result[0]}, R^2 Train Motor:␣
 ↪{fold_result[1]}, R^2 Test Motor: {fold_result[2]}, R^2 Train Total:␣
 ↪{fold_result[3]}, R^2 Test Total: {fold_result[4]}")
```

[70]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.kernel_ridge import KernelRidge
from sklearn.metrics import r2_score
from metric_learn import MLKR
from joblib import Parallel, delayed

path = './parkinsons_updrs.data'

data = pd.read_csv(path)

X = data.drop(columns=['subject#', 'motor_UPDRS', 'total_UPDRS'])
y_motor = data['motor_UPDRS']
```

```
y_total = data['total_UPDRS']

X_train, X_test, y_motor_train, y_motor_test, y_total_train, y_total_test = train_test_split(X, y_motor, y_total, test_size=0.3, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

kf = KFold(n_splits=5, shuffle=True, random_state=42)

n = 10

pca_transformations = PCA(n_components=n).fit(X_train_scaled)

def process_fold(train_i, value_i, n, X_train_scaled, y_motor_train, y_total_train):
    X_train_kf, X_value_kf = X_train_scaled[train_i], X_train_scaled[value_i]
    y_motor_train_kf, y_motor_value_kf = y_motor_train.iloc[train_i], y_motor_train.iloc[value_i]
    y_total_train_kf, y_total_value_kf = y_total_train.iloc[train_i], y_total_train.iloc[value_i]

    X_motor_train_PCA = pca_transformations.transform(X_train_kf)
    X_motor_test_PCA = pca_transformations.transform(X_value_kf)

    mlkr_motor = MLKR(n_components=n, init='pca')
    mlkr_motor.fit(X_motor_train_PCA, y_motor_train_kf)
    X_motor_train_MLKR = mlkr_motor.transform(X_motor_train_PCA)
    X_motor_test_MLKR = mlkr_motor.transform(X_motor_test_PCA)

    gauss_kernel_motor = KernelRidge(kernel='rbf')
    gauss_kernel_motor.fit(X_motor_train_MLKR, y_motor_train_kf)

    y_train_pred_motor = gauss_kernel_motor.predict(X_motor_train_MLKR)
    y_test_pred_motor = gauss_kernel_motor.predict(X_motor_test_MLKR)

    r2_train_motor = r2_score(y_motor_train_kf, y_train_pred_motor)
    r2_test_motor = r2_score(y_motor_value_kf, y_test_pred_motor)

    X_total_train_PCA = pca_transformations.transform(X_train_kf)
    X_total_test_PCA = pca_transformations.transform(X_value_kf)

    mlkr_total = MLKR(n_components=n, init='pca')
    mlkr_total.fit(X_total_train_PCA, y_total_train_kf)
    X_total_train_MLKR = mlkr_total.transform(X_total_train_PCA)
    X_total_test_MLKR = mlkr_total.transform(X_total_test_PCA)
```

```
    gauss_kernel_total = KernelRidge(kernel='rbf')
    gauss_kernel_total.fit(X_total_train_MLKR, y_total_train_kf)

    y_train_pred_total = gauss_kernel_total.predict(X_total_train_MLKR)
    y_test_pred_total = gauss_kernel_total.predict(X_total_test_MLKR)

    r2_train_total = r2_score(y_total_train_kf, y_train_pred_total)
    r2_test_total = r2_score(y_total_value_kf, y_test_pred_total)

    return (n, r2_train_motor, r2_test_motor, r2_train_total, r2_test_total)

def process_m(n, kf, X_train_scaled, y_motor_train, y_total_train):
    results = Parallel(n_jobs=-1)(
        delayed(process_fold)(train_i, value_i, n, X_train_scaled,␣
 ↪y_motor_train, y_total_train)
        for train_i, value_i in kf.split(X_train_scaled)
    )

    return results

parallel_results = Parallel(n_jobs=-1)(
    delayed(process_m)(n, kf, X_train_scaled, y_motor_train, y_total_train) for␣
 ↪n in [n]
)

for result in parallel_results:
    for fold_result in result:
        print(f"Components: {fold_result[0]}, R^2 Train Motor:␣
 ↪{fold_result[1]}, R^2 Test Motor: {fold_result[2]}, R^2 Train Total:␣
 ↪{fold_result[3]}, R^2 Test Total: {fold_result[4]}")
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[70], line 83
     76     results = Parallel(n_jobs=-1)(
     77         delayed(process_fold)(train_i, value_i, n, X_train_scaled,␣
 ↪y_motor_train, y_total_train)
     78         for train_i, value_i in kf.split(X_train_scaled)
     79     )
     81     return results
---> 83 parallel_results = Parallel(n_jobs=-1)(
     84     delayed(process_m)(n, kf, X_train_scaled, y_motor_train,␣
 ↪y_total_train) for n in [n]
     85 )
     87 # Print results for each fold
     88 for result in parallel_results:
```

```
File ~/anaconda3/envs/EE559/lib/python3.11/site-packages/joblib/parallel.py:
↪2007, in Parallel.__call__(self, iterable)
   2001 # The first item from the output is blank, but it makes the interpreter
   2002 # progress until it enters the Try/Except block of the generator and
   2003 # reaches the first `yield` statement. This starts the asynchronous
   2004 # dispatch of the tasks to the workers.
   2005 next(output)
-> 2007 return output if self.return_generator else list(output)

File ~/anaconda3/envs/EE559/lib/python3.11/site-packages/joblib/parallel.py:
↪1650, in Parallel._get_outputs(self, iterator, pre_dispatch)
   1647     yield
   1649     with self._backend.retrieval_context():
-> 1650         yield from self._retrieve()
   1652 except GeneratorExit:
   1653     # The generator has been garbage collected before being fully
   1654     # consumed. This aborts the remaining tasks if possible and warn
   1655     # the user if necessary.
   1656     self._exception = True

File ~/anaconda3/envs/EE559/lib/python3.11/site-packages/joblib/parallel.py:
↪1762, in Parallel._retrieve(self)
   1757 # If the next job is not ready for retrieval yet, we just wait for
   1758 # async callbacks to progress.
   1759 if ((len(self._jobs) == 0) or
   1760     (self._jobs[0].get_status(
   1761         timeout=self.timeout) == TASK_PENDING)):
-> 1762     time.sleep(0.01)
   1763     continue
   1765 # We need to be careful: the job list can be filling up as
   1766 # we empty it and Python list are not thread-safe by
   1767 # default hence the use of the lock

KeyboardInterrupt:
```

[ ]:

```python
import pandas as pd
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.kernel_ridge import KernelRidge
from sklearn.metrics import r2_score
from metric_learn import MLKR
from joblib import Parallel, delayed

path = './parkinsons_updrs.data'
```

```python
data = pd.read_csv(path)

X = data.drop(columns=['subject#', 'motor_UPDRS', 'total_UPDRS'])
y_motor = data['motor_UPDRS']
y_total = data['total_UPDRS']

X_train, X_test, y_motor_train, y_motor_test, y_total_train, y_total_test = train_test_split(X, y_motor, y_total, test_size=0.3, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

kf = KFold(n_splits=5, shuffle=True, random_state=42)

n = 15

pca_transformations = PCA(n_components=n).fit(X_train_scaled)

def process_fold(train_i, value_i, n, X_train_scaled, y_motor_train, y_total_train):
    X_train_kf, X_value_kf = X_train_scaled[train_i], X_train_scaled[value_i]
    y_motor_train_kf, y_motor_value_kf = y_motor_train.iloc[train_i], y_motor_train.iloc[value_i]
    y_total_train_kf, y_total_value_kf = y_total_train.iloc[train_i], y_total_train.iloc[value_i]

    X_motor_train_PCA = pca_transformations.transform(X_train_kf)
    X_motor_test_PCA = pca_transformations.transform(X_value_kf)

    mlkr_motor = MLKR(n_components=n, init='pca')
    mlkr_motor.fit(X_motor_train_PCA, y_motor_train_kf)
    X_motor_train_MLKR = mlkr_motor.transform(X_motor_train_PCA)
    X_motor_test_MLKR = mlkr_motor.transform(X_motor_test_PCA)

    gauss_kernel_motor = KernelRidge(kernel='rbf')
    gauss_kernel_motor.fit(X_motor_train_MLKR, y_motor_train_kf)

    y_train_pred_motor = gauss_kernel_motor.predict(X_motor_train_MLKR)
    y_test_pred_motor = gauss_kernel_motor.predict(X_motor_test_MLKR)

    r2_train_motor = r2_score(y_motor_train_kf, y_train_pred_motor)
    r2_test_motor = r2_score(y_motor_value_kf, y_test_pred_motor)

    X_total_train_PCA = pca_transformations.transform(X_train_kf)
    X_total_test_PCA = pca_transformations.transform(X_value_kf)
```

```
    mlkr_total = MLKR(n_components=n, init='pca')
    mlkr_total.fit(X_total_train_PCA, y_total_train_kf)
    X_total_train_MLKR = mlkr_total.transform(X_total_train_PCA)
    X_total_test_MLKR = mlkr_total.transform(X_total_test_PCA)

    gauss_kernel_total = KernelRidge(kernel='rbf')
    gauss_kernel_total.fit(X_total_train_MLKR, y_total_train_kf)

    y_train_pred_total = gauss_kernel_total.predict(X_total_train_MLKR)
    y_test_pred_total = gauss_kernel_total.predict(X_total_test_MLKR)

    r2_train_total = r2_score(y_total_train_kf, y_train_pred_total)
    r2_test_total = r2_score(y_total_value_kf, y_test_pred_total)

    return (n, r2_train_motor, r2_test_motor, r2_train_total, r2_test_total)

def process_m(n, kf, X_train_scaled, y_motor_train, y_total_train):
    results = Parallel(n_jobs=-1)(
        delayed(process_fold)(train_i, value_i, n, X_train_scaled,␣
 ↪y_motor_train, y_total_train)
        for train_i, value_i in kf.split(X_train_scaled)
    )

    return results

parallel_results = Parallel(n_jobs=-1)(
    delayed(process_m)(n, kf, X_train_scaled, y_motor_train, y_total_train) for␣
 ↪n in [n]
)

for result in parallel_results:
    for fold_result in result:
        print(f"Components: {fold_result[0]}, R^2 Train Motor:␣
 ↪{fold_result[1]}, R^2 Test Motor: {fold_result[2]}, R^2 Train Total:␣
 ↪{fold_result[3]}, R^2 Test Total: {fold_result[4]}")
```

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.kernel_ridge import KernelRidge
from sklearn.metrics import r2_score
from metric_learn import MLKR
from joblib import Parallel, delayed

path = './parkinsons_updrs.data'
```

```
data = pd.read_csv(path)

X = data.drop(columns=['subject#', 'motor_UPDRS', 'total_UPDRS'])
y_motor = data['motor_UPDRS']
y_total = data['total_UPDRS']

X_train, X_test, y_motor_train, y_motor_test, y_total_train, y_total_test =␣
 ↪train_test_split(X, y_motor, y_total, test_size=0.3, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

kf = KFold(n_splits=5, shuffle=True, random_state=42)

n = 19

pca_transformations = PCA(n_components=n).fit(X_train_scaled)

def process_fold(train_i, value_i, n, X_train_scaled, y_motor_train,␣
 ↪y_total_train):
    X_train_kf, X_value_kf = X_train_scaled[train_i], X_train_scaled[value_i]
    y_motor_train_kf, y_motor_value_kf = y_motor_train.iloc[train_i],␣
 ↪y_motor_train.iloc[value_i]
    y_total_train_kf, y_total_value_kf = y_total_train.iloc[train_i],␣
 ↪y_total_train.iloc[value_i]

    mlkr_motor = MLKR(n_components=n, init='auto')
    mlkr_motor.fit(X_train_kf, y_motor_train_kf)
    X_motor_train_MLKR = mlkr_motor.transform(X_train_kf)
    X_motor_test_MLKR = mlkr_motor.transform(X_value_kf)

    gauss_kernel_motor = KernelRidge(kernel='rbf')
    gauss_kernel_motor.fit(X_motor_train_MLKR, y_motor_train_kf)

    y_train_pred_motor = gauss_kernel_motor.predict(X_motor_train_MLKR)
    y_test_pred_motor = gauss_kernel_motor.predict(X_motor_test_MLKR)

    r2_train_motor = r2_score(y_motor_train_kf, y_train_pred_motor)
    r2_test_motor = r2_score(y_motor_value_kf, y_test_pred_motor)


    mlkr_total = MLKR(n_components=n, init='auto')
    mlkr_total.fit(X_train_kf, y_total_train_kf)
    X_total_train_MLKR = mlkr_total.transform(X_train_kf)
    X_total_test_MLKR = mlkr_total.transform(X_value_kf)
```

```
    gauss_kernel_total = KernelRidge(kernel='rbf')
    gauss_kernel_total.fit(X_total_train_MLKR, y_total_train_kf)

    y_train_pred_total = gauss_kernel_total.predict(X_total_train_MLKR)
    y_test_pred_total = gauss_kernel_total.predict(X_total_test_MLKR)

    r2_train_total = r2_score(y_total_train_kf, y_train_pred_total)
    r2_test_total = r2_score(y_total_value_kf, y_test_pred_total)

    return (n, r2_train_motor, r2_test_motor, r2_train_total, r2_test_total)

def process_m(n, kf, X_train_scaled, y_motor_train, y_total_train):
    results = Parallel(n_jobs=-1)(
        delayed(process_fold)(train_i, value_i, n, X_train_scaled,␣
 ↪y_motor_train, y_total_train)
        for train_i, value_i in kf.split(X_train_scaled)
    )

    return results

parallel_results = Parallel(n_jobs=-1)(
    delayed(process_m)(n, kf, X_train_scaled, y_motor_train, y_total_train) for␣
 ↪n in [n]
)

for result in parallel_results:
    for fold_result in result:
        print(f"Components: {fold_result[0]}, R^2 Train Motor:␣
 ↪{fold_result[1]}, R^2 Test Motor: {fold_result[2]}, R^2 Train Total:␣
 ↪{fold_result[3]}, R^2 Test Total: {fold_result[4]}")
```

(c) Use sklearn's neural network implementation to train a neural network with two outputs that predicts motor UPDRS and total UPDRS. Use a single layer. You are responsible to determine other architectural parameters of the network, including the number of neurons in the hidden and output layers, method of optimization, type of activation functions, and the L2 "regularization" parameter etc. You should determine the design parameters via trial and error, by testing your trained network on the test set and choosing the architecture that yields the smallest test error. For this part, set early-stopping=False. Remember to standardize your features. Report your R^2 on both training and test sets. (20 pts)

[91]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.neural_network import MLPRegressor
```

```
path = './parkinsons_updrs.data'
data = pd.read_csv(path)

X = data.drop(columns=['subject#', 'motor_UPDRS', 'total_UPDRS'])
y = data[['motor_UPDRS', 'total_UPDRS']]


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

nn = MLPRegressor(hidden_layer_sizes=(200,), activation='tanh', solver='sgd',␣
 ↪alpha=0.001, max_iter=5000, learning_rate_init=0.001,␣
 ↪learning_rate='adaptive', random_state=42,early_stopping=False,␣
 ↪validation_fraction=0.1)

nn.fit(X_train_scaled, y_train)

y_train_pred = nn.predict(X_train_scaled)
r2_train_motor = r2_score(y_train['motor_UPDRS'], y_train_pred[:, 0])
r2_train_total = r2_score(y_train['total_UPDRS'], y_train_pred[:, 1])

y_test_pred = nn.predict(X_test_scaled)
r2_test_motor = r2_score(y_test['motor_UPDRS'], y_test_pred[:, 0])
r2_test_total = r2_score(y_test['total_UPDRS'], y_test_pred[:, 1])

print(f'R^2 on training set for motor UPDRS: {r2_train_motor}')
print(f'R^2 on training set for total UPDRS: {r2_train_total}')
print(f'R^2 on test set for motor UPDRS: {r2_test_motor}')
print(f'R^2 on test set for total UPDRS: {r2_test_total}')
```

```
R^2 on training set for motor UPDRS: 0.9591315893397608
R^2 on training set for total UPDRS: 0.9641449844842482
R^2 on test set for motor UPDRS: 0.8292075924024247
R^2 on test set for total UPDRS: 0.8348681095343674
```

(d)Use the design parameters that you chose in the first part and train a neural network, but this time set early-stopping=True. Research what early stopping is, and compare the performance of your network on the test set with the previous network. You can leave the validation-fraction as the default (0.1) or change it to see whether you can obtain a better model. Remember to standardize your features. Report your R2 on both training and test sets. (10 pts)

[86]:
```
import numpy as np
import pandas as pd
```

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.neural_network import MLPRegressor

path = './parkinsons_updrs.data'
data = pd.read_csv(path)

X = data.drop(columns=['subject#', 'motor_UPDRS', 'total_UPDRS'])
y = data[['motor_UPDRS', 'total_UPDRS']]


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
 ↪random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

nn = MLPRegressor(hidden_layer_sizes=(200,), activation='tanh', solver='sgd',
 ↪alpha=0.0001, max_iter=5000, learning_rate_init=0.01,
 ↪learning_rate='adaptive', random_state=42,early_stopping=True,
 ↪validation_fraction=0.1)

nn.fit(X_train_scaled, y_train)

y_train_pred = nn.predict(X_train_scaled)
r2_train_motor = r2_score(y_train['motor_UPDRS'], y_train_pred[:, 0])
r2_train_total = r2_score(y_train['total_UPDRS'], y_train_pred[:, 1])

y_test_pred = nn.predict(X_test_scaled)
r2_test_motor = r2_score(y_test['motor_UPDRS'], y_test_pred[:, 0])
r2_test_total = r2_score(y_test['total_UPDRS'], y_test_pred[:, 1])

print(f'R^2 on training set for motor UPDRS: {r2_train_motor}')
print(f'R^2 on training set for total UPDRS: {r2_train_total}')
print(f'R^2 on test set for motor UPDRS: {r2_test_motor}')
print(f'R^2 on test set for total UPDRS: {r2_test_total}')
```

```
R^2 on training set for motor UPDRS: 0.9321351544702516
R^2 on training set for total UPDRS: 0.9338009724721683
R^2 on test set for motor UPDRS: 0.8119978743947998
R^2 on test set for total UPDRS: 0.8058333276066811
```

With tuning I was able to get R^2 values in the 0.80s range, but still worse than without early stopping