COMPSCI 589
Lecture 10: Support Vector and Neural Network Regression

Benjamin M. Marlin

College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).
Created with support from National Science Foundation Award# IIS-1350522.

Outline

1. SVR
2. Neural Network Regression

Support Vector Regression

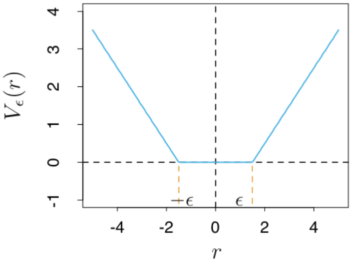
- Support Vector Regression (SVR) is the generalization of SVMs to the case of regression.

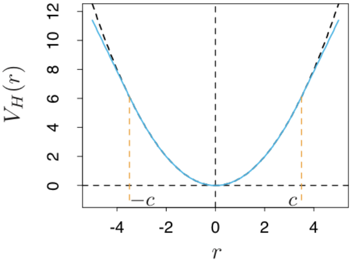
Support Vector Regression

- Support Vector Regression (SVR) is the generalization of SVMs to the case of regression.
- As with SVMs, SVR is a linear regression model trained using a different objective function. In this case, the *epsilon insensitive loss*.

Support Vector Regression

- Support Vector Regression (SVR) is the generalization of SVMs to the case of regression.
- As with SVMs, SVR is a linear regression model trained using a different objective function. In this case, the *epsilon insensitive loss*.





Support Vector Regression

$$f_{SVR}(\mathbf{x}) = \left(\sum_{d=1}^{D} w_d x_d\right) + b = \mathbf{x}\mathbf{w} + b$$

Support Vector Regression

$$f_{SVR}(\mathbf{x}) = \left(\sum_{d=1}^{D} w_d x_d\right) + b = \mathbf{x}\mathbf{w} + b$$

$$\mathbf{w}^*, b^* = \underset{\mathbf{w},b}{\arg\min}\, C \sum_{i=1}^{N} V_\epsilon(y_i - \mathbf{x}_i\mathbf{w} - b) + ||\mathbf{w}||_2^2$$

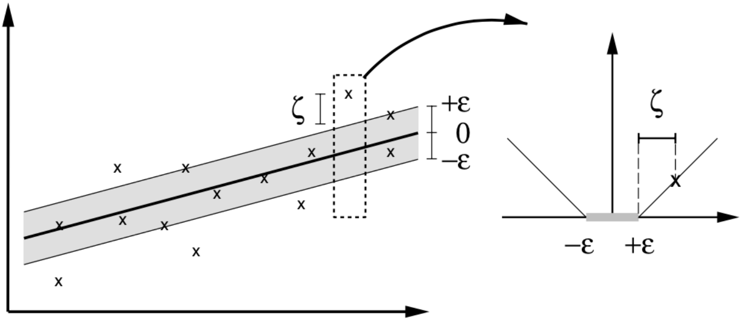
Support Vector Regression

$$f_{SVR}(\mathbf{x}) = \left(\sum_{d=1}^{D} w_d x_d\right) + b = \mathbf{x}\mathbf{w} + b$$

$$\mathbf{w}^*, b^* = \arg\min_{\mathbf{w},b} C \sum_{i=1}^{N} V_\epsilon(y_i - \mathbf{x}_i\mathbf{w} - b) + ||\mathbf{w}||_2^2$$

$$V_\epsilon(r) = \begin{cases} 0 & \text{... if } |r| < \epsilon \\ |r| - \epsilon & \text{... otherwise} \end{cases}$$

Support Vector Regression



Kernelization

Using the same representer theorem used in classification, it can be shown that

$$f_{SVR}(\mathbf{x}) = \mathbf{x}\mathbf{w}^* + b* = \sum_{i=1}^{N} \alpha_i < \mathbf{x}, \mathbf{x}_i > + \sum_{i=1}^{N} \alpha_i < 1, \mathbf{x}_i >$$

Kernelization

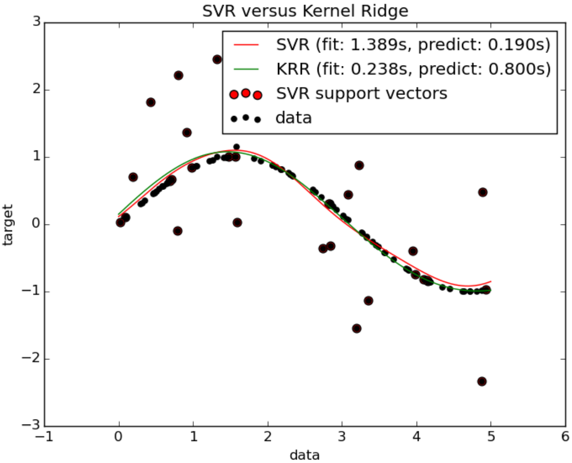
Using the same representer theorem used in classification, it can be shown that

$$f_{SVR}(\mathbf{x}) = \mathbf{x}\mathbf{w}^* + b* = \sum_{i=1}^{N} \alpha_i < \mathbf{x}, \mathbf{x}_i > + \sum_{i=1}^{N} \alpha_i < 1, \mathbf{x}_i >$$

This can again be generalized using kernels to allow for non-linear models:

$$f_{SVR}(\mathbf{x}) = \mathbf{x}\mathbf{w}^* + b* = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$$

SVR vs KRR


SVR versus Kernel Ridge
SVR (fit: 1.389s, predict: 0.190s)
KRR (fit: 0.238s, predict: 0.800s)
SVR support vectors
data
target
data
3
2
1
0
−1
−2
−3
−1
0
1
2
3
4
5
6

Trade-Offs

- SVR is more robust to outliers than OLS (minimizing the MSE) due to the loss being linear in the tails instead of quadratic. This is related to a long line of work on robust regression.

Trade-Offs

- SVR is more robust to outliers than OLS (minimizing the MSE) due to the loss being linear in the tails instead of quadratic. This is related to a long line of work on robust regression.
- Kernel SVR is low bias and has good capacity control, but use of cross validation to select regularization hyperparameters is critical.

Trade-Offs

- SVR is more robust to outliers than OLS (minimizing the MSE) due to the loss being linear in the tails instead of quadratic. This is related to a long line of work on robust regression.
- Kernel SVR is low bias and has good capacity control, but use of cross validation to select regularization hyperparameters is critical.
- The learning problem is convex for any choice of reglarization parameters and thus has a unique global optimum.

Trade-Offs

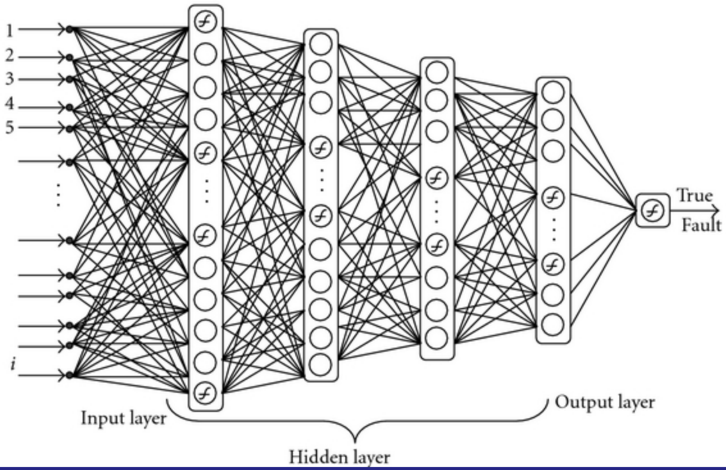
- SVR is more robust to outliers than OLS (minimizing the MSE) due to the loss being linear in the tails instead of quadratic. This is related to a long line of work on robust regression.
- Kernel SVR is low bias and has good capacity control, but use of cross validation to select regularization hyperparameters is critical.
- The learning problem is convex for any choice of reglarization parameters and thus has a unique global optimum.
- The kernel matrix computation is quadratic in the data dimension, but the model has a support vector property.

Trade-Offs

- SVR is more robust to outliers than OLS (minimizing the MSE) due to the loss being linear in the tails instead of quadratic. This is related to a long line of work on robust regression.
- Kernel SVR is low bias and has good capacity control, but use of cross validation to select regularization hyperparameters is critical.
- The learning problem is convex for any choice of reglarization parameters and thus has a unique global optimum.
- The kernel matrix computation is quadratic in the data dimension, but the model has a support vector property.
- You need to know what kernel to use or you need to use some form of validation to select from among several alternatives.

Outline

Multi-Layer Perceptron



Neural Network Regression

To convert an MLP from classification to regression, we only need to change the output activation function from logistic to linear.

- The hidden layer non-linearities are smooth functions:

$$h_k^1 = \frac{1}{1 + \exp(-(\sum_d w_{dk}^1 x_d + b_{dk}^1))}$$

$$h_k^i = \frac{1}{1 + \exp(-(\sum_l w_{lk}^i h_l^{(i-1)} + b_{lk}^i))} \text{ for } i = 2, ..., L$$

Neural Network Regression

To convert an MLP from classification to regression, we only need to change the output activation function from logistic to linear.

- The hidden layer non-linearities are smooth functions:

$$h_k^1 = \frac{1}{1+\exp(-(\sum_d w_{dk}^1 x_d + b_{dk}^1))}$$

$$h_k^i = \frac{1}{1+\exp(-(\sum_l w_{lk}^i h_l^{(i-1)} + b_{lk}^i))} \text{ for } i = 2, ..., L$$

- The output layer activation function is a linear function:

$$\hat{y} = \sum_l w_l^o h_l^L + b^o$$

Learning

Let θ be the complete collection of parameters defining a neural network model. Our goal is to find the value of θ that minimizes the MSE on the training data set $\mathcal{D} = \{y_i, \mathbf{x}_i\}_{i=1:N}$

$$\mathcal{L}_{MSE}(\mathcal{D}|\theta) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

Gradients

We need the gradient with respect to each of the parameters. Let's begin with w^o_l:

$$\frac{\partial \mathcal{L}_{MSE}(\mathcal{D}|\theta)}{\partial w^o_l} = \frac{\partial}{\partial w^o_l}\frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2 = 0 \tag{1}$$

$$= \frac{2}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)\frac{\partial \hat{y}_n}{\partial w^o_l} \tag{2}$$

$$= \frac{2}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)h^L_l \tag{3}$$

Gradients

It's also useful to define the derivatives wrt the hidden units for a single data case:

$$\epsilon_k^L = \frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial h_k^L} = \frac{\partial}{\partial h_k^L}(y - \hat{y})^2 \tag{4}$$

$$= 2(y - \hat{y})\frac{\partial \hat{y}}{\partial h_k^L} \tag{5}$$

$$= 2(y - \hat{y})w_k^o \tag{6}$$

Gradients

It's also useful to define the derivatives wrt the hidden units for a single data case:

$$\epsilon_k^L = \frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial h_k^L} = \frac{\partial}{\partial h_k^L}(y - \hat{y})^2 \tag{4}$$

$$= 2(y - \hat{y})\frac{\partial \hat{y}}{\partial h_k^L} \tag{5}$$

$$= 2(y - \hat{y})w_k^o \tag{6}$$

In general, we can define: $\epsilon_k^j = \dfrac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial h_k^j}$

Gradients

Suppose we're trying to compute the derivative with respect to the weight w^j_{kl} for some layer j and assume we have ϵ^j_l computed for all hidden units l in layer j.

$$\frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial w^j_{kl}} = \frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial h^j_l} \frac{\partial h^j_l}{\partial w^j_{kl}} \tag{7}$$

$$= \epsilon^j_l h^j_l (1 - h^j_l) h^{j-1}_k \tag{8}$$

Gradients

Suppose we're trying to compute the derivative with respect to the weight w^j_{kl} for some layer j and assume we have ϵ^j_l computed for all hidden units l in layer j.

$$\frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial w^j_{kl}} = \frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial h^j_l} \frac{\partial h^j_l}{\partial w^j_{kl}} \tag{7}$$

$$= \epsilon^j_l h^j_l (1 - h^j_l) h^{j-1}_k \tag{8}$$

The total derivative is then given by:

$$\frac{\partial \mathcal{L}_{MSE}(\mathcal{D}|\theta)}{\partial w^j_{kl}} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \mathcal{L}_{MSE}(y_n, \mathbf{x}_n|\theta)}{\partial w^j_{kl}}$$

Gradients

Suppose we're trying to compute the error with respect to hidden unit k in layer $j-1$ and assume we have ϵ_l^j computed for all hidden units l in layer j.

$$\frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial h_k^{j-1}} = \sum_l \frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial h_l^j} \frac{\partial h_l^j}{\partial h_k^{j-1}} \tag{9}$$

$$= \sum_l \epsilon_l^j h_l^j (1 - h_l^j) w_{kl}^{j-1} \tag{10}$$

Backpropagation

- The Backpropagation algorithm works by making a forward pass through the network for each data case and storing all the hidden unit values.

Backpropagation

- The Backpropagation algorithm works by making a forward pass through the network for each data case and storing all the hidden unit values.
- The algorithm then computes the error at the output and makes a backward pass through the network computing the derivatives with respect to the parameters as well as the contribution of each hidden unit to the error. These are the ϵ_k^j values.

Backpropagation

- The Backpropagation algorithm works by making a forward pass through the network for each data case and storing all the hidden unit values.
- The algorithm then computes the error at the output and makes a backward pass through the network computing the derivatives with respect to the parameters as well as the contribution of each hidden unit to the error. These are the ϵ_k^j values.
- The complete computation is just an application of the chain rule with caching of intermediate terms in the neural network graph structure.

Trade-Offs

- Neural network regression has low bias, but high variance.

Trade-Offs

- Neural network regression has low bias, but high variance.
- The objective function has local optima and requires iterative numerical optimization using backpropagation to compute the gradients, which can be slow.

Trade-Offs

- Neural network regression has low bias, but high variance.
- The objective function has local optima and requires iterative numerical optimization using backpropagation to compute the gradients, which can be slow.
- Making predictions with trained models can be very fast.

Trade-Offs

- Neural network regression has low bias, but high variance.
- The objective function has local optima and requires iterative numerical optimization using backpropagation to compute the gradients, which can be slow.
- Making predictions with trained models can be very fast.
- Capacity control in these models can be crucial. The capacity parameters are the depth of the network and the size of each layer.

Trade-Offs

- Neural network regression has low bias, but high variance.
- The objective function has local optima and requires iterative numerical optimization using backpropagation to compute the gradients, which can be slow.
- Making predictions with trained models can be very fast.
- Capacity control in these models can be crucial. The capacity parameters are the depth of the network and the size of each layer.
- These models can also be trained using ℓ_2 or ℓ_1 regularization or the more recent dropout scheme as an alternative to controlling network structure.