Using the Kernel Trick With Inner Products

The kernel trick is a way to get around this dilemma by learning a function in the much higher dimensional space, without ever computing a single vector $\phi(\mathbf{x})$ or ever computing the full vector $\mathbf{w}$.

Gradient Descent with Squared Loss

It is based on the following observation: if we use gradient descent with any one of our standard loss functions, the gradient is a linear combination of the input samples. For example, let us take a look at the squared loss:

$$\ell(\mathbf{w}) = \sum_{i=1}^{n} \left(\mathbf{w}^\top \mathbf{x}_i - y_i\right)^2$$

The gradient descent rule, with learning rate $s > 0$ updates $\mathbf{w}$ over time:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - s\left(\frac{\partial \ell}{\partial \mathbf{w}}\right)$$

where:

$$\frac{\partial \ell}{\partial \mathbf{w}} = \sum_{i=1}^{n} \underbrace{2\left(\mathbf{w}^\top \mathbf{x}_i - y_i\right)}_{\gamma_i\text{: function of } \mathbf{x}_i, y_i} \mathbf{x}_i = \sum_{i=1}^{n} \gamma_i \mathbf{x}_i$$

Computing "w" with Input Samples

We will now show that we can express $\mathbf{w}$ as a linear combination of all input vectors, $\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$.

Since the squared loss is convex, gradient descent will reach the only minima of the loss function. Consequently, the final solution is independent of the initialization. Let us assume that we initialize $\mathbf{w}_0$ to be $\mathbf{w}_0 = [0, \ldots, 0]^\top$.

For this initial choice of $\mathbf{w}_0$, the coefficients α_i in the linear combination formulation, $\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$, are trivially $\alpha_1 = \cdots = \alpha_n = 0$. We now show that throughout multiple gradient descent steps, the coefficients must always exist.

$$\mathbf{w}_1 = \mathbf{w}_0 - s\sum_{i=1}^{n} 2\left(\mathbf{w}_0^\top \mathbf{x}_i - y_i\right)\mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^0 \mathbf{x}_i - s\sum_{i=1}^{n} \gamma_i^0 \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^1 \mathbf{x}_i \qquad (\text{with } \alpha$$

$$\mathbf{w}_2 = \mathbf{w}_1 - s\sum_{i=1}^{n} 2\left(\mathbf{w}_1^\top \mathbf{x}_i - y_i\right)\mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^1 \mathbf{x}_i - s\sum_{i=1}^{n} \gamma_i^1 \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^2 \mathbf{x}_i \qquad (\text{with } \alpha$$

$$\cdots \qquad \cdots$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - s\sum_{i=1}^{n} 2\left(\mathbf{w}_{t-1}^\top \mathbf{x}_i - y_i\right)\mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^{t-1} \mathbf{x}_i - s\sum_{i=1}^{n} \gamma_i^{t-1} \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^t \mathbf{x}_i \qquad (\text{with } \alpha_i^t =$$

Formally, the argument is by induction. $\mathbf{w}$ is trivially a linear combination of the training vectors for $\mathbf{w_0}$ (base case). If we assume that $\mathbf{w}_t$ is a linear combination of the training vectors, $\mathbf{w}_{t+1}$ also becomes a linear combination of the training vectors as the coefficients $\alpha_i^{t+1} = \alpha_i^t - s\gamma_i^t = \alpha_i^t - s \cdot 2\left(\mathbf{w}_t^\top \mathbf{x}_i - y_i\right)$ only depend on the training vectors as well (inductive case).

Representing Loss Without "w"

The update-rule for α_i^t is thus $\alpha_i^t = \alpha_i^{t-1} - s\gamma_i^{t-1}$. If we repeatedly substitute $\alpha_i^{t-1} = \alpha_i^{t-2} - s\gamma_i^{t-2}$, then for α_i^{t-2} and so on until $\alpha_i^0 = 0$, we get $\alpha_i^t = -s\sum_{r=0}^{t-1}\gamma_i^r$.

In other words, we can perform the entire gradient descent update rule without ever expressing $\mathbf{w}$ explicitly. We just keep track of the n coefficients $\alpha_1, \ldots, \alpha_n$.

Now that $\mathbf{w}$ can be written as a linear combination of the training set, we can also express the inner-product of $\mathbf{w}$ with any vector $\mathbf{x}$ purely in terms of inner-products between training vectors and $\mathbf{x}$:

$$\mathbf{w}^\top \mathbf{x} = \sum_{i=1}^n \alpha_i \mathbf{x}_i^\top \mathbf{x}$$

Consequently, we can also re-write the squared-loss from $\ell(\mathbf{w}) = \sum_{i=1}^n \left(\mathbf{w}^\top \mathbf{x}_i - y_i\right)^2$ entirely in terms of inner-product between training inputs:

$$\ell(\alpha) = \sum_{i=1}^n \left(\sum_{j=1}^n \alpha_j \mathbf{x}_j^\top \mathbf{x}_i - y_i\right)^2$$

During test-time we only need the coefficients $\alpha_1, \ldots, \alpha_n$ to make a prediction on a test-input $\mathbf{x}$.

$$h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} = \sum_{i=1}^n \alpha_i \mathbf{x}_i^\top \mathbf{x}$$

The only information we ever need in order to learn a hyperplane classifier with the squared-loss is inner-products between all pairs of data vectors. Isn't it similar to Kernel SVMs?