Loss function

We'll use the Hinge loss. This is a loss function used for training classifiers. The hinge loss is used for "maximum-margin" classification, most notably for support vector machines (SVMs).

$$c(x, y, f(x)) = (1 - y * f(x))_+$$

c is the loss function, x the sample, y is the true label, f(x) the predicted label.

This means the following: $c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$

Objective Function

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

As you can see, our objective of a SVM consists of two terms. The first term is a regularizer, the heart of the SVM, the second term the loss. The regularizer balances between margin maximization and loss. We want to find the decision surface that is maximally far away from any data points.

How do we minimize our loss/optimize for our objective (i.e learn)?

We have to derive our objective function to get the gradients! Gradient descent ftw. As we have two terms, we will derive them seperately using the sum rule in differentiation.

$$\frac{\delta}{\delta w_k} \lambda \parallel w \parallel^2 = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k} \left(1 - y_i \langle x_i, w \rangle\right)_+ = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

This means, if we have a misclassified sample, we update the weight vector w using the gradients of both terms, else if classified correctly,we just update w by the gradient of the regularizer.

Misclassification condition

$$y_i \langle x_i, w \rangle < 1$$

Update rule for our weights (misclassified)

$$w = w + \eta(y_i x_i - 2\lambda w)$$

including the learning rate η and the regularizer λ The learning rate is the length of the steps the algorithm makes down the gradient on the error curve.

- Learning rate too high? The algorithm might overshoot the optimal point.
- Learning rate too low? Could take too long to converge. Or never converge.

The regularizer controls the trade off between the achieving a low training error and a low testing error that is the ability to generalize your classifier to unseen data. As a regulizing parameter we choose 1/epochs, so this parameter will decrease, as the number of epochs increases.

- Regularizer too high? overfit (large testing error)
- Regularizer too low? underfit (large training error)

Update rule for our weights (correctly classified)

$$w = w + \eta(-2\lambda w)$$