# Classification with Linear Equations

Amir Globerson

We will begin with a nice illustration of material learned previously, showing how a high accuracy MNIST classifier can be obtained very quickly with only few lines of code! This is based on a recent paper in [1].

Consider the standard binary classification problem where input data is $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ and $y_1, \ldots, y_n \in \{0, 1\}$. We can approach this using standard linear or kernel SVMs. But let's try a simpler approach.

We will now treat this as a regression problem. Namely, we will look for a vector $\boldsymbol{w} \in \mathbb{R}^d$ such that $\boldsymbol{w} \cdot \boldsymbol{x} \approx \boldsymbol{y}_i$. If we had such a $\boldsymbol{w}$ we could classify $\boldsymbol{x}$ by checking if $\boldsymbol{w} \cdot \boldsymbol{x} > 0.5$.

Now let us be "greedy" and look for a $\boldsymbol{w}$ such that $\boldsymbol{w} \cdot \boldsymbol{x}_i = y_i$ for all $i$. We can write this in matrix form as follows. Denote $X \in \mathbb{R}^{n,d}$ a matrix whose rows are the $\boldsymbol{x}_i$. And $\boldsymbol{y} \in \mathbb{R}^n$. Then we are looking for a $\boldsymbol{w}$ such that:

$$X\boldsymbol{w} = \boldsymbol{y} \tag{1}$$

This is a set of $n$ equations in $d$ unknowns. Thus we generally do not expect for it to have a solution if $d < n$. On the other hand when $d > n$ we expect it to have infinitely many solutions. So, if we had a large enough dimensionality we could solve Equation 10 and our training error would be zero! But this will come at a cost of bad generalization (you can convince yourself that you can choose a set of $d - n$ test points for which your test error will be arbitrarily bad for some $\boldsymbol{w}$ which correctly labels the training points).

In order to avoid over-fitting lets add regularization. Specifically, let us look for a minimum norm $\boldsymbol{w}$. Namely we want to solve:

$$\begin{array}{ll} \min_{\boldsymbol{w}} & 0.5\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} & X\boldsymbol{w} = \boldsymbol{y} \end{array} \tag{2}$$

Using Lagrange multipliers $\boldsymbol{\alpha} \in \mathbb{R}^n$ (for the $n$ linear constraints in Equation 10) we get:

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}) = 0.5\|\boldsymbol{w}\|_2^2 + \boldsymbol{\alpha}^T (\boldsymbol{y} - X\boldsymbol{w}) \tag{3}$$

Taking gradient with respect to $\boldsymbol{w}$ we get:

$$\boldsymbol{w} = X^T\boldsymbol{\alpha} = \sum_i \alpha_i \boldsymbol{x}_i \tag{4}$$

Where we have used the fact that the gradient of $\boldsymbol{v} \cdot \boldsymbol{w}$ with respect to $\boldsymbol{w}$ is $\boldsymbol{v}$. Equation 4 tells us something nice and intuitive, known as the representer theorem. Namely, that the optimal weight $\boldsymbol{w}$ is a combination of input points $\boldsymbol{x}_i$. This places a significant constraint on what this $\boldsymbol{w}$ can be. Presumably there can be multiple $\boldsymbol{w}$ that have this property and minimize the norm, but in fact there is only one, as we show next (this can also be proved using the strict convexity of the optimization problem).

Given the representer form of $\boldsymbol{w}$, we just need to find the $\boldsymbol{\alpha}$. We can do this by substituting $\boldsymbol{w} = X^T\boldsymbol{\alpha}$ into the constraint $X\boldsymbol{w} = \boldsymbol{y}$, which results in:

$$XX^T\boldsymbol{\alpha} = \boldsymbol{y} \tag{5}$$

Now the matrix $K = XX^T$ is a matrix in $\mathbb{R}^{n,n}$ called the kernel matrix of the data. Since $n < d$, the $K$ can have rank $n$ (if we had $d < n$, then the rank would be less than $n$). So it's not unreasonable to assume $K$ will be invertible. Therefore we can find $\boldsymbol{\alpha}$:

$$\boldsymbol{\alpha} = K^{-1}\boldsymbol{y} \tag{6}$$

We see that there is a single $\boldsymbol{\alpha}$ that satisfies the constraints and therefore we have solved the problem.

To classify a new point $\boldsymbol{x}$ we need to calculate $\boldsymbol{w} \cdot \boldsymbol{x}$, which can be done using the "kernel-trick". Namely:

$$\boldsymbol{w} \cdot \boldsymbol{x} = \sum_i \alpha_i \boldsymbol{x}_i \cdot \boldsymbol{x} \tag{7}$$

# 1 Using Kernels

The kernel-trick allows us to use the above method as long as we can calculate dot products $\boldsymbol{x} \cdot \boldsymbol{x}'$. As we have seen in the previous course, this lets us consider other feature vectors $\boldsymbol{\phi}(\boldsymbol{x})$ that can be high dimensional, as long as we have a function $K$ that calculates $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{x}')$. One well known example of such a kernel is the Radial Basis Function (RBF) defined as:

$$K(\boldsymbol{x}, \boldsymbol{x}') = e^{-\frac{1}{2\sigma^2}\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2} \tag{8}$$

Using this definition, the output of the regression can be calculated via:

$$\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}) = \sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) \tag{9}$$

# 2 Extension to Multi Class

Above we assumed that a binary label was represented as $y \in \{0, 1\}$. What happens if the number of classes is $L > 2$? Instead of the discrete label $y$ lets have a "one-hot" vector $\boldsymbol{y} \in \mathbb{R}^L$ where $\boldsymbol{y}$ is all zeros except the element in the $y$ place is 1. Thus for example if $y_i = 2$ and $L = 4$ then $\boldsymbol{y}_i = [0, 1, 0, 0]$. This is a very useful trick for converting discrete features into continuous ones.

We can now return to the regression approach above. But instead of one regression function we will have $L$ weight vectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_L$. And $\boldsymbol{w}_i \cdot \boldsymbol{x}$ will be used to predict whether $\boldsymbol{x}$ has the label $i$.

Lets write this in matrix form. Define the matrix $W \in \mathbb{R}^{d,L}$, whose columns are the $\boldsymbol{w}_i$ vectors. Denote $X \in \mathbb{R}^{n,d}$ a matrix whose rows are the $\boldsymbol{x}_i$. And $Y \in \mathbb{R}^{n,L}$ a row whose rows are the $\boldsymbol{y}_i$. Then we are looking for a $W$ such that:

$$XW = Y \tag{10}$$

We can use the approach for binary classification to learn each of the $\boldsymbol{w}_i$ separately.

# References


[1] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.