Machine Learning Algebra

By Matthieu Lagarde

April 9, 2022

1 Multivariate linear regression

1.1 Notations

Let m be the number of examples or observations in our training set. Let n be the number of features or explanatory variables observed for each example of the training set. Let $x_j^{(i)}$ be the value of feature j for example i. Let $y^{(i)}$ be the output value for example i.

$$y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ ... \\ y^{(m)} \end{pmatrix} \in \mathbb{R}^m \tag{1}$$

y is called the output vector. It contains the output values of the training set.

$$X = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & ... & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & ... & x_n^{(2)} \\ ... & ... & ... & ... & \\ 1 & x_1^{(m)} & x_2^{(m)} & ... & x_n^{(m)} \end{pmatrix} \in \mathbb{R}^{m \times (n+1)} \tag{2}$$

X is called the data matrix. If we ignore the first column of 1, each row of matrix X is one example of the training set and each column of the matrix X is the values observed for one feature in the training set.

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ ... \\ \theta_n \end{pmatrix} \in \mathbb{R}^{(n+1)} \tag{3}$$

θ is the vector of parameters.

1.2 Hypothesis

We assume that there is a linear relationship between the features and the output. Note that the relationship is linear in θ but the features themselves can be non linear transformations of the initial features such as quadratic terms or interaction terms.

The unvectorized form of the hypothesis for a given example i is:

$$h_\theta(x^{(i)}) = \sum_{j=0}^{n} \theta_j * x_j^{(i)} \in \mathbb{R}\ with\ x_0^{(i)} = 1 \tag{4}$$

The vectorized form of the hypothesis can be written as follows:

$$h_\theta(X) = X\theta \in \mathbb{R}^m \tag{5}$$

For a single example, we can also write the vectorized form of the hypothesis:

$$x^{(i)} = \begin{pmatrix} 1 \\ x_1^{(i)} \\ \dots \\ x_n^{(i)} \end{pmatrix} \in \mathbb{R}^{(n+1)}$$

$$h_\theta(x^{(i)}) = {x^{(i)}}^\top \theta \in \mathbb{R}$$

1.3 Cost function

The unvectorized form of the cost function is:

$$J(\theta) = \frac{1}{2m} * \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \in \mathbb{R}$$
$$where\ x^{(i)} \in \mathbb{R}^{(n+1)}\ and$$
$$where\ y^{(i)} \in \mathbb{R}$$

The vectorized form of the cost function is:

$$J(\theta) = \frac{1}{2m} * (X\theta - y)^\top (X\theta - y) \in \mathbb{R} \tag{6}$$

1.4 Normal equation

$J(\theta)$ is convex so any local minimum is a global minimum. We thus know that θ^* obeys the following equation:

$$\nabla J(\theta^*) = 0_{(n+1)}$$

Let recall a property of matrix differentiation. Let α be a scalar equal to $y^\top x$ where y and x be two column vectors of $\mathbb{R}^m$ that are respectively a function of another column vector z of $\mathbb{R}^n$. We have:

$$\alpha = y^\top x$$

$$\frac{\partial \alpha}{\partial z} = \frac{\partial y}{\partial z}^\top x + \frac{\partial x}{\partial z}^\top y \in \mathbb{R}^n$$

Using this property, we have:

$$\begin{aligned} & 1/2m * 2 * \frac{\partial (X\theta^* - y)}{\partial \theta}^\top (X\theta^* - y) = 0_{(n+1)} \\ \iff & 1/m * X^\top (X\theta^* - y) = 0_{(n+1)} \\ \iff & X^\top X\theta^* - X^\top y = 0_{(n+1)} \\ \boxed{\iff} & \boxed{\theta^* = (X^\top X)^{-1} X^\top y \in \mathbb{R}^{(n+1)}} \end{aligned}$$

1.5 Gradient descent

If necessary, here is the vectorized implementation of gradient descent. Denoting α the learning rate:

$$\theta := \theta - \frac{\alpha}{m} * X^\top (X\theta - y) \in \mathbb{R}^{(n+1)} \tag{7}$$

1.6 Regularized cost function

Denoting λ the regularization parameter, the unvectorized form of the cost function can be written as:

$$J(\theta) = \frac{1}{2m} * \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} * \sum_{j=1}^{n} \theta_j^2 \in \mathbb{R}$$
$$where\ x^{(i)} \in \mathbb{R}^{(n+1)}\ and$$
$$where\ y^{(i)} \in \mathbb{R}$$

Be careful, in the regularization part of the expression (the second sum), the j index goes from 1 to n and NOT from 0 to n. Indeed, by convention, we do not regularize θ_0.

The vectorized form of the regularized cost function is thus:

$$J(\theta) = \frac{1}{2m} * (X\theta - y)^\top (X\theta - y) + \frac{\lambda}{2m} * \theta_r^\top \theta_r$$
$$where\ \theta_r = \begin{pmatrix} \theta_1 \\ ... \\ \theta_n \end{pmatrix} \in \mathbb{R}^n$$

1.7 Regularized normal equation

The vectorized form of the regularized normal equation is:

$$\theta^* = (X^\top X + \lambda * M)^{-1} X^\top y \in \mathbb{R}^{(n+1)}$$
$$where\ M = \begin{pmatrix} 0 & 0 & 0 & ... & 0 \\ 0 & 1 & 0 & ... & 0 \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & 1 & 0 \\ 0 & 0 & ... & 0 & 1 \end{pmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}$$

1.8 Regularized gradient descent

Let compute the gradient of $J(\theta)$ when it is regularized. There are two cases, one for the partial derivative with respect to θ_0 and one for the partial derivatives with respect to θ_j:

$$\frac{\partial J(\theta)}{\partial \theta_0} = \left[\frac{1}{m} * X^\top (X\theta - y)\right]_1 \; i.e.\; 1st\; element\; of\; previous\; gradient$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left[\frac{1}{m} * X^\top (X\theta - y) + \frac{\lambda}{m} * \theta\right]_{j+1} \; for\; j \in \{1, 2, ..., n\}$$

2 Logistic regression

2.1 Sigmoid function

Let introduce the sigmoid function:

$$g \; : \; x \in \mathbb{R} \longrightarrow \frac{1}{1 + e^{-x}} \in]0, 1[\tag{8}$$

The interesting properties of the sigmoid function are:

- It is defined over $\mathbb{R}$.
- It is increasing.
- $g(0) = \frac{1}{2}$.
- It is converging to 0 in $-\infty$ and to 1 in $+\infty$.
- It is convex over $]-\infty, 0]$ and concave over $[0, +\infty[$.
- It means that (0, 0.5) is an inflection point of g.
- $g(-4) = 0.02$ and $g(4) = 0.98$

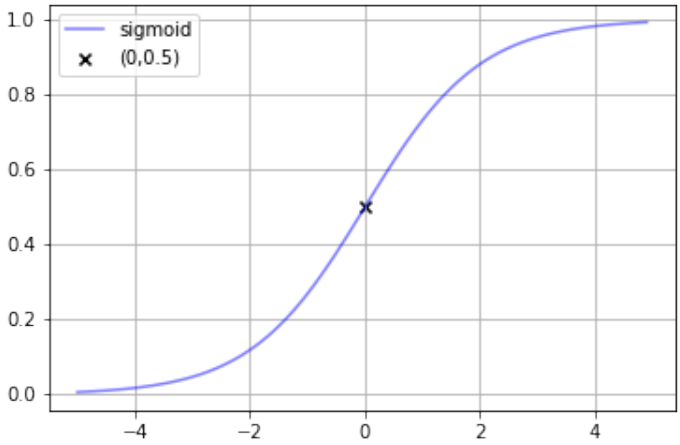

Figure 1: Graph of the sigmoid function

2.2 Hypothesis

The unvectorized form of the hypothesis for a given example i is:

$$h_\theta(x^{(i)}) = g\left(\sum_{j=0}^{n} \theta_j * x_j^{(i)}\right) \in]0,1[$$

$$where\ g\ is\ the\ sigmoid\ function.$$

The hypothesis can be interpreted as the probability that a new observation belongs to the positive class i.e. that $y = 1$ given the value of its features i.e. given x, parameterized by θ. In other words:

$$h_\theta(x) = P\left(y = 1 | x; \theta\right) \tag{9}$$

We then introduce the following decision rule:

$$y = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5 \\ 0 & \text{if } h_\theta(x) < 0.5 \end{cases} \tag{10}$$

Finally, the vectorized form of the hypothesis can be written as follows:

$h_\theta(X) = g\left(X\theta\right) \in]0,1[^m$
where g is the sigmoid function applied element-wise

2.3 Cost function

Before introducing the cost function, let study two functions:

$$v \ : \ x \in]0,1[\longrightarrow -ln(x) \in [0,+\infty[$$

$$w \ : \ x \in]0,1[\longrightarrow -ln(1-x) \in [0,+\infty[$$

Function v has the following properties on the interval $]0,1[$:

- It is decreasing.
- It converges to $+\infty$ in 0.
- $v(1) = 0$.

Function w has the following properties on the interval $]0,1[$:

- It is increasing.
- It converges to $+\infty$ in 1.
- $w(0) = 0$.

Here is the graph of v and w on the interval $]0,1[$:

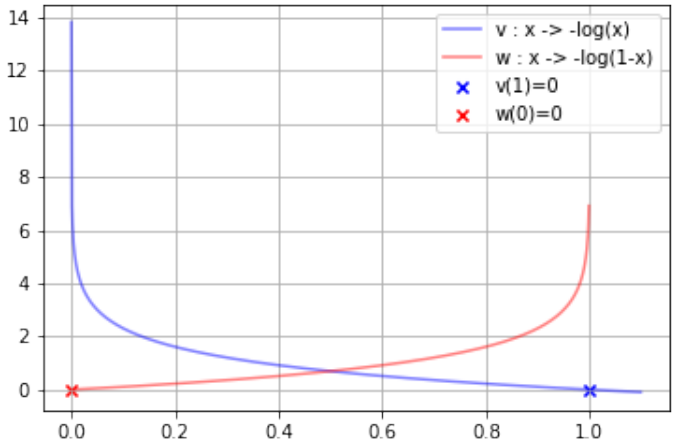

Figure 2: Graph of v and w

Now we introduce the following cost function:

$$J(\theta) = \frac{1}{m} * \sum_{i=1}^{m} Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) \in \mathbb{R}$$

where

$$Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = \begin{cases} -ln\left(h_\theta\left(x^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -ln\left(1 - h_\theta\left(x^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

Given that $y \in \{0, 1\}$, the cost function can be rewritten as:

$$J(\theta) = \frac{1}{m} * \sum_{i=1}^{m} \left[\left(y^{(i)} * -ln\left(h_\theta\left(x^{(i)}\right)\right)\right) + \left((1 - y^{(i)}) * -ln\left(1 - h_\theta\left(x^{(i)}\right)\right)\right)\right] \in \mathbb{R} \tag{11}$$

We can also write the vectorized form of the cost function as follows:

$$J(\theta) = -\frac{1}{m} * \left[y^\top ln(h_\theta(X)) + (1 - y)^\top ln(1 - h_\theta(X))\right] \in \mathbb{R}$$

where ln is applied element-wise

2.4 Gradient descent

The unvectorized form of the gradient of the cost function can be written as:

$$\frac{\partial J(\theta)}{\partial \theta_k} = \frac{1}{m} * \sum_{i=1}^{m} \left(\frac{1}{1 + exp(-\sum_{j=0}^{n} x_j^{(i)} * \theta_j)} - y^{(i)} \right) * x_k^{(i)} \in \mathbb{R} \qquad (12)$$

The vectorized form of the gradient of the cost function can be written as:

$$\nabla J(\theta) = \frac{1}{m} * X^\top \left(h_\theta(X) - y \right) \in \mathbb{R}^{n+1}$$

where $h_\theta(X) \in]0,1[^m$

Denoting α the learning rate, a vectorized implementation of the gradient descent can thus be written as:

$$\theta := \theta - \frac{\alpha}{m} * X^\top \left(h_\theta(X) - y \right) \in \mathbb{R}^{n+1}$$

where $h_\theta(X) = g(X\theta) \in]0,1[^m$

and g is the sigmoid function applied element-wise

2.5 Regularization

The regularized cost function can be written as:

$$J(\theta) = -\frac{1}{m} * \left[y^\top ln(h_\theta(X)) + (1-y)^\top ln(1 - h_\theta(X)) \right] + \frac{\lambda}{2m} * \theta_r^\top \theta_r \in \mathbb{R}$$

where ln is applied element-wise

The regularized gradient can be written as:

$\left[\widetilde{\nabla J(\theta)}\right]_1 = [\nabla J(\theta)]_1$for $\partial\theta_0$ i.e. the first element of the regularized gradient

$\left[\widetilde{\nabla J(\theta)}\right]_j = \left[\nabla J(\theta) + \frac{\lambda}{m} * \theta\right]_j$ where $j \in \{2, 3, ..., n\}$

3 Neural network classifier

3.1 Notations

Let consider the following neural network:

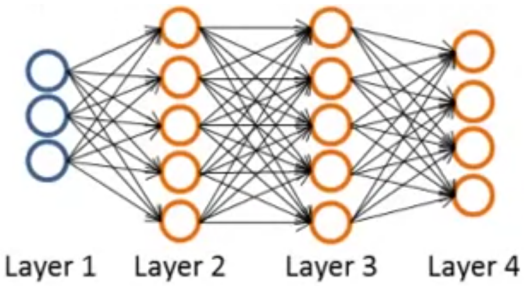

Figure 3: Example of neural network

We assume that we face a multi-class classification where we want to predict whether an observation belongs to one of K possible classes. We denote L the total number of layers in the network.

By convention, the input layer is not counted as a true layer. So in this example, we are facing a 3-layer neural network. The first layer is called the input layer and is not counted in the total, it is indexed as layer 0. The last layer is called the output layer, it is indexed as layer 3 in this example.

Intermediate layers are called hidden layers. Here, there are 2 hidden layers. We denote $n_h^{[l]}$ the number of units in layer l, not counting the bias unit. In the example, we thus have:

- $n_h^{[0]} = 3$ which is n_x, the number of input features.
- $n_h^{[1]} = 5$.
- $n_h^{[2]} = 5$.
- $n_h^{[3]} = 4$ which is K or n_y or $n_h^{[L]}$, the number of classes.

We assume that a bias unit is added as an input to any layer. We denote m the total number of examples in our training set.

We are going to change a bit our notations for solving neural networks as it will be more convenient. The data matrix X is gonna be transposed i.e. we are going to stack horizontally each training example.

$X = \begin{pmatrix} x^{(1)} & x^{(2)} & ... & x^{(m)} \end{pmatrix} \in \mathbb{R}^{n_x \times m}$

We are doing the same for the output label matrix:

$Y = \begin{pmatrix} y^{(1)} & y^{(2)} & ... & y^{(m)} \end{pmatrix} \in \mathbb{R}^{n_y \times m}$

Both for X and Y, the number of rows is equal to the number of features (resp. classes) and the number of columns is equal to the number of training examples. One column corresponds to one training example.

Finally, we denote $W^{[l]}$ the weight matrix required to go from layer $(l-1)$ to layer l. This matrix is in the space $\mathbb{R}^{n_h^{[l]} \times n_h^{[l-1]}}$ since it maps vectors of dimension $n_h^{[l-1]}$ into vectors of dimension $n_h^{[l]}$. The number of rows correspond to the number of units in the next layer while the number of columns correspond to the number of units in the previous layer!

Note that in this example, there are 3 matrices of parameters to estimate : $W^{[1]}$, $W^{[2]}$ and $W^{[3]}$. It requires to estimate $5 * 3 + 5 * 5 + 4 * 5 = 15 + 25 + 20 = 60$ weights, without counting the weights associated to the biais units.

Finally, the i-th row of matrix $W^{[l]}$ corresponds to the weights used to compute the i-th activation unit of layer l.

We also introduce the bias vectors $b^{[l]}$ of layer l:

$b^{[l]} \in \mathbb{R}^{n_h^{[l]}}$

There is one weight for each activation unit of layer l.

3.2 Hypothesis

The hypothesis can be computed recursively as follows:

$A^{[1]} = g^{[1]}\left(W^{[1]} * X + b^{[1]}\right) \in n_h^{[l]} \times m$
where $g^{[l]}(.)$ denotes the activation function for layer l applied element-wise and where $b^{[1]}$ is broadcasted to be $n_h^{[l]} \times m$
$\hat{Y} = A^{[L]} = g^{[L]}\left(W^{[L]} * A^{[L-1]} + b^{[L-1]}\right) \in n_y \times m$

Again, the hypothesis only gives the probability that an observation belongs to each class. As for the one vs. all multi-class classification problem, we then introduce the following decision rule to pick the most probable class given the input features:

$$\text{Class of } y^{(i)} = index_{of}\left(max(\hat{y^{(i)}})\right) \in \mathbb{N} \tag{13}$$

3.3 Choices of the activation functions

Here are the different functions used as activation functions i.e. as functions that are applied to the linear combinations of the activation units of the previous layer and the weights.

Sigmoid function $\frac{1}{1+e^{-x}}$	Hyperbolic tangent function (tanh) $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Rectified Linear Unit (ReLU) $max(0, x)$	Leaky ReLU $max(0.001x, x)$

In practice, here are some tips and notes:

- Never use the sigmoid function except for the output layer if it is a binary classification problem ;
- The standard is to used the RELU function for all units.
- The disadvantage of the RELU function is that it is not differentiable at 0 so if for a given example, $z^{(i)[l]}$ is equal to 0, it can mess up but it is very seldom.
- The disadvantage of the sigmoid function is that its derivative is close to 0 on large range of its domain so it slows down gradient descent.

3.4 Cost function

Very similar to the cost function for the logistic regression, we just add a cost for the estimation of the probability of each of the K class for each example. We can first write it in a non-vectorized way:

$$J(W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]}) = -\frac{1}{m} * \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} * ln(a_k^{[2](i)}) + (1 - y_k^{(i)}) * ln(1 - a_k^{[2](i)}) + \frac{\lambda}{2m} * \sum_{l=1}^{L} \sum_{i=1}^{n_h^{[l]}} \sum_{j=1}^{n_h^{[l-1]}} {W_{i,j}^{[l]}}^2$$

where $a_k^{[2](i)}$ is the predicted probability that example i belongs to class k
and $W_{i,j}^{[l]}$ is the weight associated to the i-th hidden unit of layer [l]
and the j-th activation unit of previous layer

It can be vectorized as follows:

$$J(W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]}) = -\frac{1}{m} * np.sum \left[Y. * ln(A^{[2]}) + (1 - Y). * ln(1 - A^{[2]}) \right] + \frac{\lambda}{2m} * np.sum \left(\sum_{l=1}^{L} W^{[l]}. * W^{[l]} \right) \tag{14}$$

3.5 Backpropagation algorithm

In order to minimize the cost function, we need to compute the partial derivatives of $J(W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]})$ with respect to all the parameters i.e. with respect to each element of the L matrices of weights and to each element of the L vectors of bias:

$$\forall l, \ \frac{\partial J}{\partial W^{[l]}} \text{ and } \frac{\partial J}{\partial b^{[l]}} \tag{15}$$

To do so, we can use the backpropagation algorithm. It basically applies the chain rule to go back the computational graph of $A^{[L]}$.

3.6 Synthesis

Here is a summary of the steps to implement a neural network.

- Choose the number of hidden layers $[L-1]$;
- Choose the number of hidden units $n_h^{[l]}$ in each hidden layer ;
- Choose the activation functions $g^{[l]}$ at each layer: use the sigmoid function for the output layer if it is a binary classification problem. Otherwise, use the RELU function everywhere. If it is a regression problem, you can use an identity activation function at the output layer ;
- Randomly initialize the parameters $W^{[l]}$ between small negative and small positive values such as $0.001 * -2$ and $0.001 * 2$ to break up symmetry. It is useful to have small initial weights if the derivatives of the activation functions are locally small for large values of their input (it is the case for tanh, not for RELU).
- You can initialize the bias vectors $b^{[l]}$ at zeros vectors because symmetry is already broken by the random initialization of the matrices of weights ;
- Choose the learning rate α ;
- Repeat N times the following steps: (1) forward propagation to compute each activate unit $A^{[l]}$ as well as the local value of the cost function J; (2) backward propagation to compute the derivatives of J ; (3) update the parameters subtracting α times the derivative ; (4) restart.
- Plot the evolution of J every N_i iterations for instance every 1000 iterations to check J converged.

4 Support Vector Machines

4.1 Hypothesis

Support Vector Machines or SVM is another classifier. The hypothesis is a discriminant function based on the value of the linear combination:

$$h_\theta(x^{(i)}) = \begin{cases} 1 & \text{if } \theta^\top x^{(i)} >= 0 \\ 0 & \text{otherwise} \end{cases}$$

where $x^{(i)} \in \mathbb{R}^{(n+1)}$ and $\theta \in \mathbb{R}^{(n+1)}$

4.2 Cost function

The cost function is inspired by the cost function of the logistic regression but is slightly different:

$$J(\theta) = C * \left[\sum_{i=1}^{m} y^{(i)} * cost_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) * cost_0(\theta^\top x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2 \tag{16}$$

Compared to the cost function of the logistic regression:

- We have dropped the $\frac{1}{m}$ factors
- We have replaced the parameter λ by a parameter $C = \frac{1}{\lambda}$. The bigger C, the more we want to fit exactly the data. The lower C, the more we want the parameters to be small.
- The individual cost function assigning a cost to the prediction is the Hinge loss function.

It can be written in a vectorized form as follows:

$$J(\theta) = C * \left[(Y^\top cost_1(X\theta)) + ((1 - Y)^\top cost_0(X\theta)) \right] + \frac{1}{2} * \theta_r^\top \theta_r \in \mathbb{R}$$

where θ_r is equal to θ without θ_0 thus $\in \mathbb{R}^n$
$cost_0(.)$ and $cost_1(.)$are applied element-wise.

4.3 Hinge loss functions

For $y^{(i)} = 1$, the hinge loss function is:

$$cost_1(z) = \begin{cases} 0 & \text{if } z >= 1 \\ k(1 - z) & \text{if } z < 1 \end{cases} \text{ with k an arbitrary slope parameter } > 0 \tag{17}$$

For $y^{(i)} = 0$, the hinge loss function is:

$$cost_0(z) = \begin{cases} 0 & \text{if } z < -1 \\ k(1 + z) & \text{if } z >= -1 \end{cases} \text{ with k an arbitrary slope parameter } > 0 \tag{18}$$

Here is a graph of $cost_0$ and $cost_1$ on $[-2, 2]$ with $k = 2$:

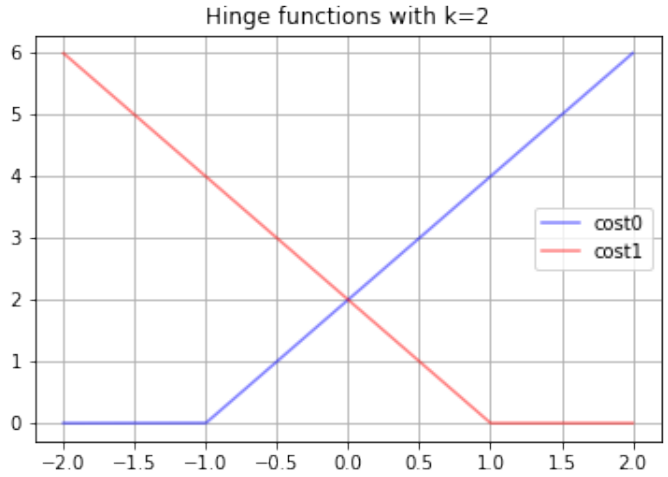

Figure 4: Graph of $cost_0$ and $cost_1$ with $k = 2$

4.4 Projections

Let be E a vector space of dimension n. Let F be a vector subspace and G a supplementary vector subspace to F in E. As a reminder:

$$F \oplus G = E \iff F \cup G = E \text{ and } F \cap G = \{0\} \tag{19}$$

Note that if $dim(F) < n$, then there is an infinite number of supplementary vector subspaces to F in E. To get this point, consider the example of $\mathbb{R}^2$:

$F = Vect((1, 1))$
$G_1 = Vect((2, 1))$ and $G_2 = Vect((0, 2))$
G_1 and G_2 are both supplementary subspaces to F for instance.

Given that G is supplementary to F, we know that:

$$\forall x \in E, \exists! (a, b) \in F \times G \text{ such that } x = a + B \tag{20}$$

We can define p the linear application that gives the part a of the decomposition of x. p is a projection. Let focus on a special kind of projections called orthogonal

projections. To do so, we will focus on the euclidean space $\mathbb{R}^n$ and we will take $F = Vect(y)$ where $y \in \mathbb{R}^n$. Now we will consider the supplementary vector subspace to F that is orthogonal to it and we call it G. It is a vector subspace that verifies:

$$\forall x \in G, < x, y >= 0$$
where $< . , . >$ is the scalar product operator
i.e. $x^\top y = 0$

We can write that:

$$\forall x \in \mathbb{R}^n, x = p(x) + x - p(x)$$

where $p(x) \in Vect(y)$ and $x - p(x) \in G$

thus $\exists! k \in \mathbb{R}, p(x) = ky$ and $< x - p(x), y >= 0$

replacing $p(x)$ in the second equation yields

$$k = \frac{< x, y >}{||y||^2}$$

i.e. $p(x) = \frac{< x, y >}{||y||^2} * y$

where $||x|| = \sqrt[2]{\sum_{i=1}^{m} x_i^2}$

4.5 Large margin intuition

Let go back to the SVM cost function. Assume that C is massively large, so much as we set the first term of the sum in the cost function $J(\theta)$ to be null. The minimization problem becomes:

$\min_{\theta} \theta^\top \theta$ subject to the constraints that

if $y^i = 1, {x^{(i)}}^\top \theta >= 1 \iff ||p(x)|| * ||\theta|| >= 1$ based on previous section

if $y^i = 0, {x^{(i)}}^\top \theta <= -1 \iff -||p(x)|| * ||\theta|| <= -1$ based on previous section

For the constraints to hold true, we need the length of the projections of both the positive and negative examples $x^{(i)}$ to be as large as possible. In other words, with a big C, minimizing the cost function is tantamount to finding the line spanned by $Vect(\theta)$ such that the orthogonal projections of both the positives examples and the negative examples on this line are the largest possible.

5 Appendix

aaa

5.1 F-score

aaa

5.2 Vectorization tips

aaa