Lecture 5

1. Event models
2. Neural networks
3. Support vector machines

1 Event models

In the previous section we assumed that all x_i are binary, $x_i \in \{0, 1\}$. To generalize the previous model we assume that $x_i \in \{1, \ldots, k\}$, then the generative learning algorithm implies that we model

$$p(x \mid y) = \prod_{i=1}^{n} p(x_i \mid y),$$

where on the right-hand side we have multinomial probabilities (rather than Bernoulli as before). In case of continuous features this model can be also implemented if you apply the procedure of discretization:

$x_1 < 1$	$1 \leqslant x_1 < 10$	$10 \leqslant x_1 < 50$	$50 \leqslant x_1 < 100$	$100 \leqslant x_1$
0	1	2	3	4

The model with more than two possible values for each feature is called **multinomial event model** (compared to the multivariate Bernoulli event model we had earlier).

We consider the spam classification problem from different point of view. One email will be described by the vector $(x_1^{(i)}, x_2^{(i)}, \ldots, x_{d_i}^{(i)})$, where d_i is a number of words in the email i and $x_j \in \{1, 2, \ldots, n\}$, where n is a size of the bug of words and could be a huge number, for example, $n = 50\,000$. In other words, we assign some number to each word and takes these numbers to the feature vector. The main difference in such formulation is that feature vectors have different lengths for different training examples (because email lengths are different).

The joint distribution for x and y can be written as

$$p(x, y) = \left(\prod_{j=1}^{d} p(x_j \mid y) \right) p(y),$$

where x_j is j-th word in the email, d is a number of words in the email. We identify the parameters for this model and apply the maximum likelihood estimation to find the values for them. Parameters are defined by the equations:

$$\begin{aligned}
&\varphi_{k\,|\,y=1} = p(x_j = k\,|\,y = 1) \text{ (for any } j),\\
&\varphi_{k\,|\,y=0} = p(x_j = k\,|\,y = 0) \text{ (for any } j),\\
&\varphi_y = p(y = 1).
\end{aligned}$$

For example, the parameter $\varphi_{k\,|\,y=1}$ defines the probability of having word k in the spam email on any position j and so on.

The likelihood is defined as

$$\begin{aligned}
&l(\varphi_{k\,|\,y=1}, \varphi_{k\,|\,y=0}, \varphi_y) = \ln \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}, \varphi_{k\,|\,y=1}, \varphi_{k\,|\,y=0}, \varphi_y) =\\
&= \ln \prod_{i=1}^{m} p(x^{(i)}\,|\,y^{(i)}, \varphi_{k\,|\,y=1}, \varphi_{k\,|\,y=0}) \cdot p(y^{(i)}; \varphi_y)
\end{aligned}$$

and the maximization gives the following list of parameters

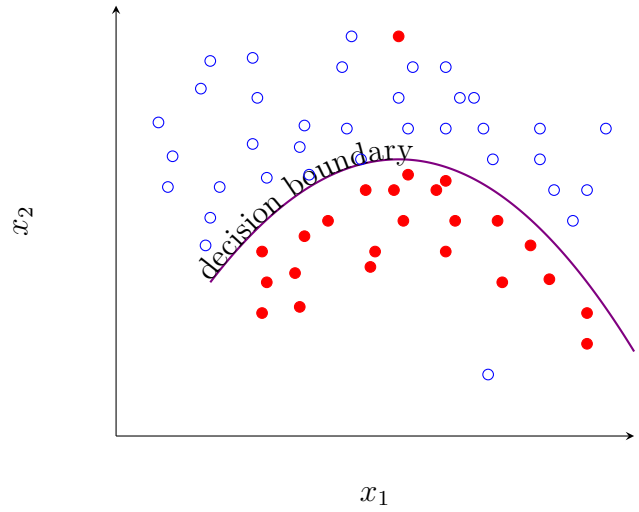
$$\varphi_{k\,|\,y=1} = \frac{\sum_{i=1}^{m} \left(\mathbb{1}\{y^{(i)} = 1\} \sum_{j=1}^{d_i} \mathbb{1}\{x_j^{(i)} = k\} \right) + 1}{\sum_{i=1}^{m} \left(\mathbb{1}\{y^{(i)} = 1\} \cdot d_i \right) + n},$$

$$\varphi_{k\,|\,y=0} = \frac{\sum_{i=1}^{m} \left(\mathbb{1}\{y^{(i)} = 0\} \sum_{j=1}^{d_i} \mathbb{1}\{x_j^{(i)} = k\} \right) + 1}{\sum_{i=1}^{m} \left(\mathbb{1}\{y^{(i)} = 0\} \cdot d_i \right) + n}.$$

The meaning of the first parameter is the number of appearances of the word k in all spam emails divided by the total number of words in all spam emails. Notice that we use the general Laplace smoothing described in the previous lecture.

2 Neural networks

The following example shows that in some cases we need non linear decision boundary which means that we should build nonlinear classifiers.



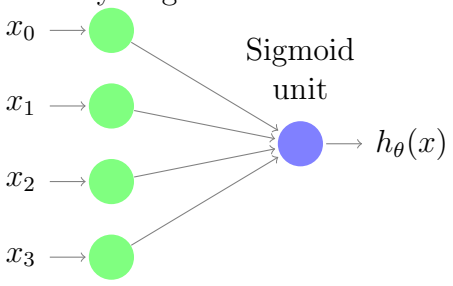
One example of nonlinear classifiers we took before was the logistic regression with hypothesis $h_\theta(x) = \dfrac{1}{1+e^{-\theta^T x}}$. Another example is generative learning algorithm with the assumptions

$$x \,|\, y = 1 \sim \text{ExpFamily}(\eta_1),$$
$$x \,|\, y = 0 \sim \text{ExpFamily}(\eta_0).$$

One way to build nonlinear classifier is to fit the hypothesis with higher degree function instead of $\theta^T x$. For example, we can consider the hypothesis

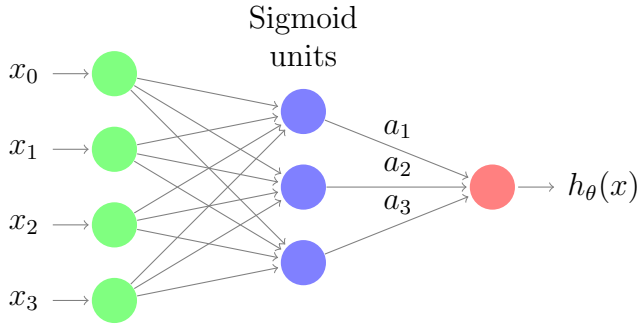
$$h_\theta(x) = \frac{1}{1+e^{-\theta_0-\theta_1 x_1-\theta_2 x_2-\theta_3 x_1 x_2-\theta_4 x_1^2 x_2-\ldots}}.$$

Another way is to visualize the logistic regression algorithm in some fancy way and try to generalize the idea:



For this diagram we are going to use the biological terminology: the circles are **neurons** or **units**, the set of green circles (representing the features) is an **input layer**, the blues circle with sigmoid function is an **output layer** with

sigmoid activation function. The natural way to generalize this diagram is to add more neurons:



On this diagram we have 3 layers: input layer (green), hidden layer (blue) with sigmoid activation functions and output layer (red) with sigmoid activation function. Let $g(z) = \dfrac{1}{1+e^{-z}}$ be a sigmoid function then

$$a_1 = g(x^T\theta_1^{(1)}),\ a_2 = g(x^T\theta_2^{(1)}),\ a_3 = g(x^T\theta_3^{(1)}),\ h_\theta(x) = g(a^T\theta^{(2)}),$$

where

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}.$$

The number of parameters (**weights** in the neural network terminology) for this network equals to the number of edges for this diagram: $12 + 3 = 15$. In other words, we should learn 4 vectors of parameters $\theta_1^{(1)}$, $\theta_2^{(1)}$, $\theta_3^{(1)}$ and $\theta^{(2)}$. Obviously we could add more hidden layers and our hypothesis becomes more complicated.

To fit the parameters we recall that the log-likelihood for logistic regression has a form

$$l(\theta) = \sum_{i=1}^{m} y^{(i)} \ln h_\theta(x^{(i)}) + (1 - y^{(i)}) \ln(1 - h_\theta(x^{(i)}))$$

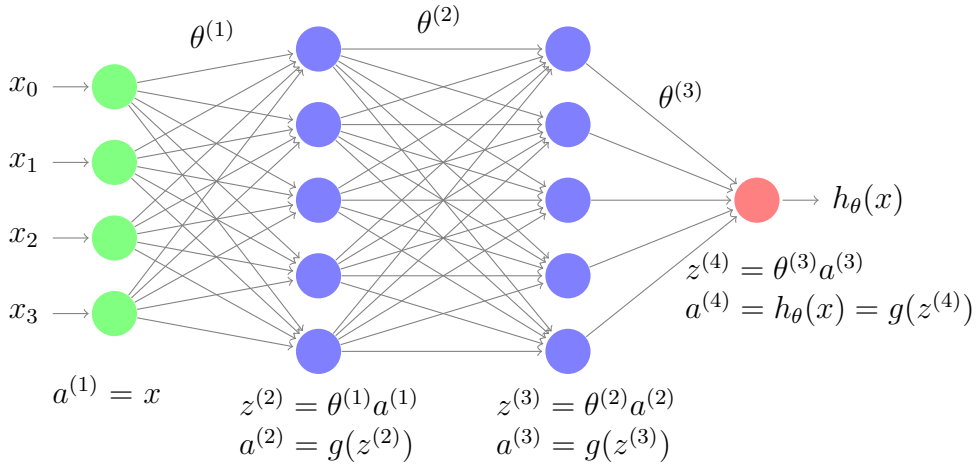
and we obtain the neural network as a generalization of logistic regression. This is the reason that for the neural network we are going to use the following loss function

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \ln h_\theta(x^{(i)}) + (1 - y^{(i)}) \ln(1 - h_\theta(x^{(i)})),$$

and find the minimum of this function using the gradient descent. For the neural networks the gradient descent algorithm has a special name: **back-propagation**. The procedure of predictions in the neural network is called **forward propagation**.

To have the convenient notations we denote by $\theta^{(k)}$ the matrix of parameters from the layer k to the layer $k+1$. The element $\theta_{ij}^{(k)}$ defines the weight from the i-th neuron of layer $k+1$ to the j-th neuron of layer k. Let $a_i^{(k)}$ be the activation function of the neuron i in layer k. To use the gradient descent algorithm we should compute all derivatives $\dfrac{\partial J(\theta)}{\partial \theta_{ij}^{(k)}}$.

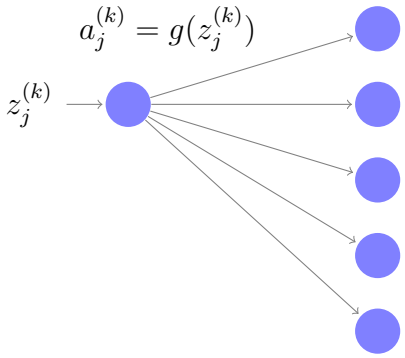
Given one training example (x, y) consider one forward propagation step for the following network configuration:



The backpropagation procedure includes the calculation of partial derivatives

$$\delta_j^{(k)} = \frac{\partial J(\theta)}{\partial z_j^{(k)}}.$$

Consider the unit j in the layer k and the next layer $k+1$:



$$z^{(k+1)} = \theta^{(k)} a^{(k)}$$

Assuming that we know derivatives $\delta_i^{(k+1)} = \dfrac{\partial J(\theta)}{\partial z_i^{(k+1)}}$ and

$$z_i^{(k+1)} = \ldots + \theta_{ij}^{(k)} a_j^{(k)} + \ldots = \ldots + \theta_{ij}^{(k)} g(z_j^{(k)}) + \ldots, \tag{1}$$

where terms from other units of layer k are denoted by dots. Then the derivative $\delta_j^{(k)}$ can be calculated using Chain Rule:

$$\delta_j^{(k)} = \frac{\partial J(\theta)}{\partial z_j^{(k)}} = \sum_i \frac{\partial J(\theta)}{\partial z_i^{(k+1)}} \cdot \frac{\partial z_i^{(k+1)}}{\partial z_j^{(k)}} =$$
$$= \sum_i \frac{\partial J(\theta)}{\partial z_i^{(k+1)}} \cdot \theta_{ij}^{(k)} \cdot g'(z_j^{(k)}) = \sum_i \delta_i^{(k+1)} \theta_{ij}^{(k)} \cdot g'(z_j^{(k)}).$$

More general, for all units of the layer k the formula can be written as

$$\delta^{(k)} = (\theta^{(k)})^T \delta^{(k+1)} g'(z^{(k)}) = (\theta^{(k)})^T \delta^{(k+1)} a^{(k)}(1 - a^{(k)})$$

(remember that $g(z)$ is a sigmoid function and its derivatives can be calculated in a very simple way).

When all "deltas" are calculated it is easy to calculate our target derivatives (again using Chain Rule and the equation (1)):

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(k)}} = \frac{\partial J(\theta)}{\partial z_i^{(k+1)}} \cdot \frac{\partial z_i^{(k+1)}}{\partial \theta_{ij}^{(k)}} = \delta_i^{(k+1)} \cdot g(z_j^{(k)}) = \delta_i^{(k+1)} \cdot a_j^{(k)}.$$

Now we can summarize the backpropagation algorithm (we are using stochastic gradient descent to update weights):

Algorithm 1 Backpropagation algorithm for neural networks

Set number of hidden layers K and number of neurons m_k in the layer k
Initialize matrices $\theta^{(k)}, k = 0, \ldots, K$ of random weights
repeat
    for all training examples (x, y) **do**
        Set $a^{(0)} = x$
        Run the forward step to compute $a^{(k)}, k = 1, \ldots, K + 1$
        Set $\delta^{(K+1)} = a^{(K+1)} - y$ (here $a^{(K+1)}$ is a prediction)
        for $k = K$ **downto** 1 **do**
            Set $\delta^{(k)} = (\theta^{(k)})^T \delta^{(k+1)} a^{(k)}(1 - a^{(k)})$
            for all $i = 1, \ldots, m_{k+1}$ **and** $j = 1, \ldots, m_k$ **do**
                Perform the stochastic gradient descent step

$$\theta_{ij}^{(k)} := \theta_{ij}^{(k)} - \alpha \cdot \delta_i^{(k+1)} \cdot a_j^{(k)}$$

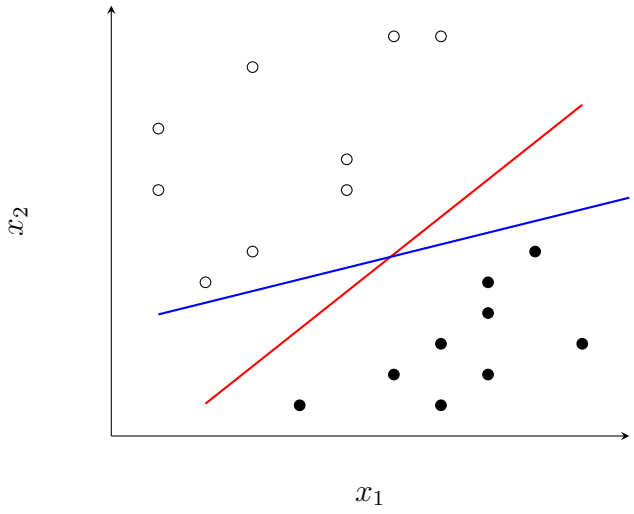
until convergence

3 Support vector machines

In this section we develop another nonlinear algorithm. There are two intuitions behind it:

- Logistic regression computes $\theta^T x$ and predict 1 if $\theta^T x > 0$, 0 - otherwise. If $\theta^T x >> 0$, then the algorithm is very "confident" that $y = 1$. If $\theta^T x << 0$, the algorithm is very "confident" that $y = 0$. Our aim is to obtain the algorithm that gives $\theta^T x^{(i)} >> 0$ for any i such that $y^{(i)} = 1$, and $\theta^T x^{(i)} << 0$ for any i such that $y^{(i)} = 0$.

- If we assume that classes are linearly separable, then we prefer the red line as our decision boundary (the intuition is that blue line is not good decision boundary):



For this algorithm we should use slightly different notations. First of all, we assume that $y \in \{-1, 1\}$ which means that output values could be 1 or -1. The function $g(z)$ is defined as

$$g(z) = \begin{cases} 1 & \text{if } z \geqslant 0, \\ -1 & \text{otherwise} \end{cases}$$

Second of all, we are using the hypothesis

$$h_{w,b}(x) = g(w^T x + b),$$

instead of $h_\theta(x) = g(\theta^T x)$. We removed the convention $x_0 = 1$ and θ_0 is replaced by b, θ is replaced by vector w.

Definition. Functional margin of a hyperplane $w^T x + b = 0$ with respect to $(x^{(i)}, y^{(i)})$ is:

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b).$$

Notice that if $y^{(i)} = 1$ then the algorithm should give $w^T x^{(i)} + b >> 0$, and if $y^{(i)} = -1$ the algorithm should give $w^T x^{(i)} + b << 0$. In both cases $\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b) > 0$. Our aim is to build the algorithm that gives the functional margin positive for all training examples.

Definition. Minimal functional margin is

$$\hat{\gamma} = \min_i \hat{\gamma}^{(i)}.$$

Based on our intuition we should claim from the algorithm that the worst functional margin should be large.

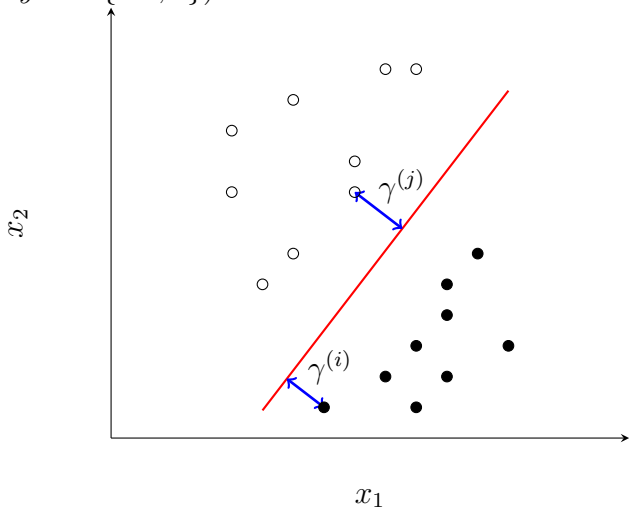
Definition. A **geometric margin** $\gamma^{(i)}$ is the distance between training example $x^{(i)}$ to the decision boundary $w^T x + b = 0$:

$$\gamma^{(i)} = y^{(i)} \left[\left(\frac{w}{||w||} \right)^T x^{(i)} + \frac{b}{||w||} \right].$$

Notice that the formula for the distance between point and hyperplane from Calculus is:

$$d = \left(\frac{w}{||w||} \right)^T x^{(i)} + \frac{b}{||w||},$$

and d has different signs for points on different sides of the plane. We multiply this distance by $y^{(i)}$ to have positive value for all training examples (remember that $y^{(i)} \in \{-1, 1\}$).



Definition. Minimal geometric margin is

$$\gamma = \min_i \gamma^{(i)}.$$

It is easy to see that functional and geometric margins are connected by $\gamma^{(i)} = \dfrac{\hat{\gamma}^{(i)}}{||w||}$. If $||w|| = 1$, then $\hat{\gamma}^{(i)} = \gamma^{(i)}$. The disadvantage of the functional margin is that it is not normalized: for example, if we double w and b, then we double functional margin, but the hyperplane $w^T x + b = 0$ does not

change. To simplify our following calculations we will assume that $||w|| = 1$ and $|w_1| = 1$.

We can formulate two equivalent optimization problems (**optimal margin classifier**):

1. $\max\limits_{\gamma,w,b} \gamma$
$$\text{subject to } y^{(i)}(w^T x^{(i)} + b) \geqslant \gamma,\ i = 1 \dots m,$$
$$||w|| = 1.$$

2. $\max\limits_{\hat{\gamma},w,b} \dfrac{\hat{\gamma}}{||w||}$
$$\text{subject to } y^{(i)}(w^T x^{(i)} + b) \geqslant \hat{\gamma},\ i = 1 \dots m.$$

Notice that for the first formulation the functional and geometric margins are the same. Also this is an example of non-convex optimization.

Consider the second optimization problem, as we mentioned before functional margin can be increasing, but the decision boundary stays the same. To avoid this situation we impose additional constraint on $\hat{\gamma}$:

$$\hat{\gamma} = 1 \text{ (scaling constraint)}.$$

With this constraint the second optimization problem transforms to the problem (**Support Vector Machine (SVM) classifier**)

$$\min_{w,b} ||w||^2 \text{ (the same as } \max_{w,b} \frac{1}{||w||})$$
$$\text{subject to } y^{(i)}(w^T x^{(i)} + b) \geqslant 1,\ i = 1 \dots m.$$

This is the quadratic programming problem, because our objective function is a quadratic function and all our constraints are linear.