Machine Learning Lecture Notes

Predrag Radivojac

February 19, 2015

Suppose we are interested in building a linear classifer $f : \mathbb{R}^k \to \{-1, +1\}$. Linear classifiers try to find the relationship between inputs and outputs by constructing a linear function (a point, a line, a plane or a hyperplane) that splits $\mathbb{R}^k$ into two half-spaces. The two half-spaces act as decision regions for the positive and negative examples, respectively. Given a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ consisting of positive and negative examples, there are many ways in which linear classifiers can be constructed. For example, a training algorithm may explicitly work to position the decision surface in order to separate positive and negative examples according to some problem-relevant criteria; e.g. it may try to minimize the fraction of examples on the incorrect side of the decision surface. Alternatively, the goal of the training algorithm may be to directly estimate the posterior distribution $p(y|\mathbf{x})$, in which case the algorithm is more likely to rely on the formal parameter estimation principles; e.g. it may maximize the likelihood. An example of a classifier with a linear decision surface is shown in Figure 1.

To simplify the formalism in the following sections, we will add a component $x_0 = 1$ to each input $(x_1, \ldots, x_k)$. This extends the input space to $\mathcal{X} = \mathbb{R}^{k+1}$ but, fortunately, it also leads us to a simplified notation in which the decision boundary in $\mathbb{R}^k$ can be written as $\mathbf{w}^T\mathbf{x} = 0$, where $\mathbf{w} = (w_0, w_1, \ldots, w_k)$ is a set of weights and $\mathbf{x} = (x_0 = 1, x_1, \ldots, x_k)$ is any element of the input space. Neverthless, we should remember that the actual inputs are k-dimensional.

Earlier in the introductory remarks, we presented a classifier as a function $f : \mathcal{X} \to \mathcal{Y}$ and have transformed the learning problem into approximating $p(y|\mathbf{x})$. In the case of linear classifiers, our flexibility is restricted because our method must learn the posterior probabilities $p(y|\mathbf{x})$ and at the same time have a linear decision surface in $\mathbb{R}^k$. This, however, can be achieved if $p(y|\mathbf{x})$ is modeled as a monotonic function of $\mathbf{w}^T\mathbf{x}$; e.g. $\tanh(\mathbf{w}^T\mathbf{x})$ or $(1 + e^{-\mathbf{w}^T\mathbf{x}})^{-1}$. Of course, a model trained to learn posterior probabilities $p(y|\mathbf{x})$ can be seen as a function $g : \mathcal{X} \to [0, 1]$. Then, the conversion from g to f is a straightforward application of the maximum a posteriori principle: the predicted output is positive if $g(\mathbf{x}) \geq 0.5$ and negative if $g(\mathbf{x}) < 0.5$.

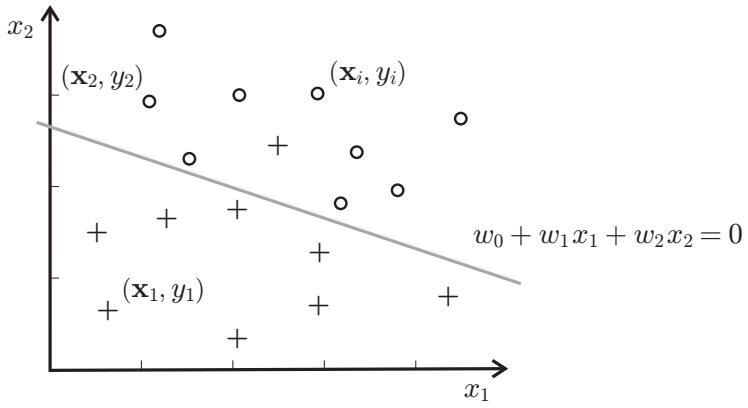

Figure 1: A data set in $\mathbb{R}^2$ consisting of nine positive and nine negative examples. The gray line represents a linear decision surface in $\mathbb{R}^2$. The decision surface does not perfectly separate positives from negatives.

1 Logistic regression

Let us consider binary classification in $\mathbb{R}^k$, where $\mathcal{X} = \mathbb{R}^{k+1}$ and $\mathcal{Y} = \{0, 1\}$. The basic idea for many classification approaches is to hypothesize a closed-form representation for the posterior probability that the class label is positive and learn parameters $\mathbf{w}$ from data. In logistic regression, this relationship can be expressed as

$$P(Y = 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}, \tag{1}$$

which is a monotonic function of $\mathbf{w}^T\mathbf{x}$. Function $f(t) = (1 + e^{-t})^{-1}$ is called the sigmoid function or the logistic function and is plotted in Figure 2.

1.1 Maximum conditional likelihood estimation

To frame the learning problem as parameter estimation, we will assume that the data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ is an i.i.d. sample from a fixed but unknown probability distribution $p(\mathbf{x}, y)$. Even more specifically, we will assume that the data generating process randomly draws a data point $\mathbf{x}$, a realization of the random vector $(X_0 = 1, X_1, \ldots, X_k)$, according to $p(\mathbf{x})$ and then sets its class label Y according to the Bernoulli distribution

$$p(y|\mathbf{x}) = \begin{cases} \left(\frac{1}{1+e^{-\boldsymbol{\omega}^T\mathbf{x}}}\right)^y & \text{for } y = 1 \\ \left(1 - \frac{1}{1+e^{-\boldsymbol{\omega}^T\mathbf{x}}}\right)^{1-y} & \text{for } y = 0 \end{cases} \tag{2}$$

where $\boldsymbol{\omega} = (\omega_0, \omega_1, \ldots, \omega_k)$ is a set of unknown coefficients we want to recover (or learn) from the observed data $\mathcal{D}$. Based on the principles of parameter estimation, we can estimate $\boldsymbol{\omega}$ by maximizing the conditional likelihood of the observed class labels $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ given the inputs $\mathbf{X} = (\mathbf{x}_1^T, \mathbf{x}_2^T, \ldots, \mathbf{x}_n^T)$.

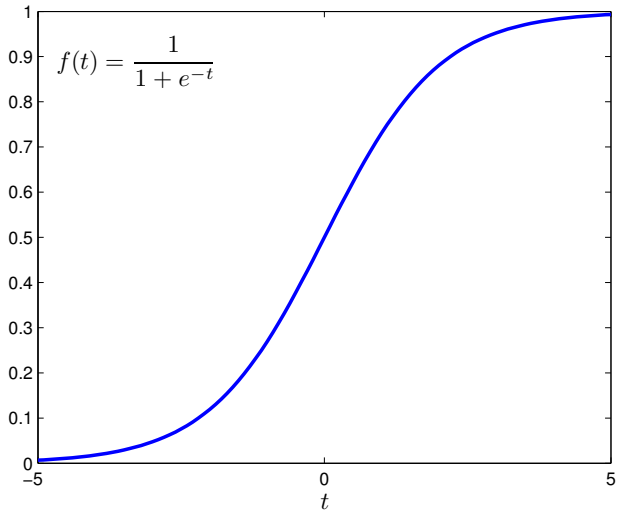

Figure 2: Sigmoid function in $[-5, 5]$ interval.

We shall first write the conditional likelihood function $p(\mathbf{y}|\mathbf{X}, \mathbf{w})$, or simply $l(\mathbf{w})$, as

$$l(\mathbf{w}) = \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}). \tag{3}$$

This function can be thought of as the probability of observing a set of labels $\mathbf{y}$ given the set of data points $\mathbf{X}$ and the particular set of coeficients $\mathbf{w}$. However, compared to Eq. (2) where for a given $\mathbf{x}$ and $\boldsymbol{\omega}$

$$\int p(y|\mathbf{x}, \boldsymbol{\omega})dy = 1,$$

here we have that for a given $\mathbf{x}$ and y

$$\int p(y|\mathbf{x}, \mathbf{w})d\mathbf{w} \neq 1.$$

This means that the likelihood is not a probability distribution over the domain of $\mathbf{w}$. More formally, we define the parameter vector that maximizes the likelihood as

$$\begin{aligned} \mathbf{w}_{\mathrm{ML}} &= \arg\max_{\mathbf{w}} \{l(\mathbf{w})\} \\ &= \arg\max_{\mathbf{w}} \left\{ \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}) \right\}. \end{aligned} \tag{4}$$

By combining Eqs. (2-3) we can now express the likelihood function as

$$l(\mathbf{w}) = \prod_{i=1}^{n} \left(\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}_i}} \right)^{y_i} \cdot \left(1 - \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}_i}} \right)^{1-y_i}. \tag{5}$$

Note that maximizing Eq. (5) is equivalent to maximizing the log-likelihood function $ll(\mathbf{w}) = \log(l(\mathbf{w}))$

$$ll(\mathbf{w}) = \sum_{i=1}^{n} \left(y_i \cdot \log\left(\frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}_i}} \right) + (1 - y_i) \cdot \log\left(1 - \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}_i}} \right) \right). \quad (6)$$

The negative of the log-likelihood from Eq. (6) is sometimes referred to as cross-entropy; thus, cross-entropy minimization is equivalent to the maximum likelihood method. To make everything more suitable for further steps, we will slightly rearrange Eq. (6) as

$$ll(\mathbf{w}) = \sum_{i=1}^{n} \left((y_i - 1)\, \mathbf{w}^T \mathbf{x}_i + \log\left(\frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}_i}} \right) \right). \quad (7)$$

It does not take much effort to realize that there is no closed-form solution to $\nabla ll(\mathbf{w}) = \mathbf{0}$ (we did have this luxury in linear regression, but not here). Thus, we have to proceed with iterative optimization methods. That is, our goal is to calculate $\nabla ll(\mathbf{w})$ and $H_{ll(\mathbf{w})}$ in order to specify the update rule described by Newton-Raphson's method, as a function of inputs $\mathbf{X}$, class labels $\mathbf{y}$, and the current parameter vector. We can calculate the first and second partial derivatives of $ll(\mathbf{w})$ as follows

$$\begin{aligned}
\frac{\partial ll(\mathbf{w})}{\partial w_j} &= \sum_{i=1}^{n} \left((y_i - 1) \cdot x_{ij} - \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}_i}} \cdot e^{-\mathbf{w}^T \mathbf{x}_i} \cdot (-x_{ij}) \right) \\
&= \sum_{i=1}^{n} x_{ij} \cdot \left(y_i - 1 + \frac{e^{-\mathbf{w}^T \mathbf{x}_i}}{1 + e^{\mathbf{w}^T \mathbf{x}_i}} \right) \\
&= \sum_{i=1}^{n} x_{ij} \cdot \left(y_i - \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}_i}} \right) \\
&= \mathbf{f}_j^T \left(\mathbf{y} - \mathbf{p} \right),
\end{aligned}$$

where $\mathbf{f}_j$ is the j-th column (feature) of data matrix $\mathbf{X}$, $\mathbf{y}$ is an n-dimensional column vector of class labels and $\mathbf{p}$ is an n-dimensional column vector of (estimated) posterior probabilities $p_i = P(Y_i = 1 | \mathbf{x}_i, \mathbf{w})$, for $i = 1, ..., n$. Considering partial derivatives for every component of $\mathbf{w}$, we have

$$\nabla ll(\mathbf{w}) = \mathbf{X}^T \left(\mathbf{y} - \mathbf{p} \right). \quad (8)$$

The second partial derivative of the log-likelihood function can be found as

$$\begin{aligned}
\frac{\partial^2 ll(\mathbf{w})}{\partial w_j \partial w_k} &= \sum_{i=1}^{n} x_{ij} \cdot \frac{e^{-\mathbf{w}^T \mathbf{x}_i}}{\left(1 + e^{-\mathbf{w}^T \mathbf{x}_i}\right)^2} \cdot (-x_{ik}) \\
&= -\sum_{i=1}^{n} x_{ij} \cdot \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}_i}} \cdot \left(1 - \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}_i}} \right) \cdot x_{ik} \\
&= -\mathbf{f}_j^T \mathbf{P} \left(\mathbf{I} - \mathbf{P} \right) \mathbf{f}_k,
\end{aligned}$$

where $\mathbf{P}$ is an $n \times n$ diagonal matrix with $P_{ii} = p_i = P(Y_i = 1|\mathbf{x}_i, \mathbf{w})$ and $\mathbf{I}$ is an $n \times n$ identity matrix. The Hessian matrix $H_{ll(\mathbf{w})}$ can now be calculated as

$$H_{ll(\mathbf{w})} = -\mathbf{X}^T \mathbf{P} \left(\mathbf{I} - \mathbf{P}\right) \mathbf{X}. \tag{9}$$

This is a negative semi-definite matrix, which guarantees that the optimum we find will be a global maximum. Negative semi-definite Hessian corresponds to a concave likelihood function and has a global maximum (unique if negative definite). Substituting Eqs. (8-9) into Newton-Raphson's method results in the following weight update rule

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \left(\mathbf{X}^T \mathbf{P}^{(t)} \left(\mathbf{I} - \mathbf{P}^{(t)}\right) \mathbf{X}\right)^{-1} \mathbf{X}^T \left(\mathbf{y} - \mathbf{p}^{(t)}\right), \tag{10}$$

where the initial weights $\mathbf{w}^{(0)}$ can be calculated using the ordinary least squares regression as $\mathbf{w}^{(0)} = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y}$. Note that the second term on the right-hand side of Eq. (10) is calculated in each iteration t; thus we wrote $\mathbf{P}$ and $\mathbf{p}$ as $\mathbf{P}^{(t)}$ and $\mathbf{p}^{(t)}$, respectively, to indicate that $\mathbf{w}^{(t)}$ was used to calculate them. The computational complexity of this procedure is $O(k^3 + k^2 n)$ in each iteration, assuming $O(k^3)$ time for finding matrix inverses.

1.2 Minimization of Euclidean distance

Another approach that is frequently considered is minimization of the Euclidean distance between a vector of class labels $\mathbf{y}$ and a vector of model outputs $\mathbf{p} = (p_1, p_2, ..., p_n)$, where $p_i = P(Y_i = 1|\mathbf{x}_i, \mathbf{w})$. This is equivalent to minimizing the squared error function $E(\mathcal{D}, \mathbf{w})$ or $E(\mathbf{w})$ as

$$\begin{aligned} E(\mathbf{w}) &= \sum_{i=1}^{n} \left(y_i - p_i\right)^2 \\ &= \sum_{i=1}^{n} e_i^2, \end{aligned}$$

where $e_i = y_i - p_i$ is the error term that corresponds to a training data point $\mathbf{x}_i$. The minimization of $E(\mathbf{w})$ is formally expressed as

$$\begin{aligned} \mathbf{w}^* &= \arg\min_{\mathbf{w}} \left\{E(\mathbf{w})\right\} \\ &= \arg\min_{\mathbf{w}} \left\{\sum_{i=1}^{n} \left(y_i - p_i\right)^2\right\}. \end{aligned} \tag{11}$$

Similar to the maximum likelihood process, our goal will be to calculate the gradient vector and the Hessian of the error function. The partial derivatives of the error function can be calculated as follows

$$
\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial w_j} &= \frac{\partial}{\partial w_j}\sum_{i=1}^{n} e_i^2 \\
&= \sum_{i=1}^{n} 2\cdot e_i \cdot \frac{\partial e_i}{\partial w_j} \\
&= 2\cdot\sum_{i=1}^{n}\left(y_i - \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}_i}}\right)\cdot\frac{1}{(1+e^{-\mathbf{w}^T\mathbf{x}_i})^2}\cdot e^{-\mathbf{w}^T\mathbf{x}_i}\cdot(-x_{ij}) \\
&= -2\cdot\sum_{i=1}^{n} x_{ij}\cdot\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}_i}}\cdot\left(1-\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}_i}}\right)\cdot\left(y_i-\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}_i}}\right) \\
&= -2\mathbf{f}_j^T\mathbf{P}\left(\mathbf{I}-\mathbf{P}\right)\left(\mathbf{y}-\mathbf{p}\right).
\end{aligned}
$$

This provides the gradient vector in the following form

$$
\nabla E(\mathbf{w}) = -2\mathbf{X}^T\mathbf{P}\left(\mathbf{I}-\mathbf{P}\right)\left(\mathbf{y}-\mathbf{p}\right).
$$

Matrix $\mathbf{J} = \mathbf{P}\left(\mathbf{I}-\mathbf{P}\right)\mathbf{X}$ is referred to as Jacobian. In general, Jacobian is an $n \times k$ matrix calculated as

$$
J_{E(\mathbf{w})} = \begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \frac{\partial e_1}{\partial w_2} & \cdots & \frac{\partial e_1}{\partial w_k} \\ \vdots & \ddots & & \\ & & & \\ \frac{\partial e_n}{\partial w_1} & & & \frac{\partial e_n}{\partial w_k} \end{bmatrix}.
$$

The second partial derivative of the error function can be found as

$$
\begin{aligned}
\frac{\partial^2 E(\mathbf{w})}{\partial w_j \partial w_k} &= 2\cdot\sum_{i=1}^{n}\frac{\partial e_i}{\partial w_k}\cdot\frac{\partial e_i}{\partial w_j} + e_i\cdot\frac{\partial^2 e_i}{\partial w_j\partial w_k} \\
&= 2\cdot\sum_{i=1}^{n} x_{ij}\cdot\left(p_i^2(1-p_i)^2 + p_i\cdot(1-p_i)\cdot(2p_i-1)\cdot(y_i-p_i)\right)\cdot x_{ik}.
\end{aligned}
$$

Thus, the Hessian can be computed as

$$
\begin{aligned}
H_{E(\mathbf{w})} &= 2\mathbf{X}^T\left(\mathbf{I}-\mathbf{P}\right)^T\mathbf{P}^T\mathbf{P}\left(\mathbf{I}-\mathbf{P}\right)\mathbf{X} + 2\mathbf{X}^T\left(\mathbf{I}-\mathbf{P}\right)^T\mathbf{P}^T\mathbf{E}(2\mathbf{P}-\mathbf{I})\mathbf{X} \\
&= 2\mathbf{J}^T\mathbf{J} + 2\mathbf{J}^T\mathbf{E}(2\mathbf{P}-\mathbf{I})\mathbf{X},
\end{aligned}
$$

where $\mathbf{P} = \operatorname{diag}\{\mathbf{p}\}$, $\mathbf{E} = \operatorname{diag}\{\mathbf{e}\}$ is a diagonal matrix containing elements $E_{ii} = e_i = y_i - p_i$ and $\mathbf{I}$ is an identity matrix. Thus, the process of minimizing the Euclidean distance between $\mathbf{y}$ and $\mathbf{p}$ results in the following update rule

$$
\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \left(\mathbf{J}^{(t)T}\mathbf{J}^{(t)} + \mathbf{J}^{(t)T}\mathbf{E}^{(t)}(2\mathbf{P}^{(t)}-\mathbf{I})\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{P}^{(t)}\left(\mathbf{y}-\mathbf{p}^{(t)}\right),
$$

where $\mathbf{w}^{(0)}$ can be calculated using ordinary least squares regression or assigned randomly.

An interesting problem here is that the Hessian is not guaranteed to be positive semi-definite. This sugests that $E(\mathbf{w})$ is not convex, i.e. it must have multiple minima with different values of the objective function. Finding a global optimum depends on how favorable the initial solution $\mathbf{w}^{(0)}$ is and how well the weight update step can escape local minima to find better ones.

This difficulty can be mitigated when

$$\left| \frac{\partial e_i}{\partial w_k} \cdot \frac{\partial e_i}{\partial w_k} \right| \gg \left| e_i \cdot \frac{\partial^2 e_i}{\partial w_j \partial w_k} \right|$$

because the Hessian can be computed as

$$H_{E(\mathbf{w})} \approx 2\mathbf{J}^T\mathbf{J}.$$

Such an approach is referred to as Gauss-Newton optimization. This Hessian is provably positive semi-definite, but the error function has effectively been changed.

Furthermore, assuming that $H_{E(\mathbf{w})} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix, results in a so-called gradient descent rule that is often used in unconstrained optimization. Gradient descent occasionally suffers from instability and is modified into the following update rule

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta\mathbf{X}^T\mathbf{P}^{(t)}\left(\mathbf{I} - \mathbf{P}^{(t)}\right)\left(\mathbf{y} - \mathbf{p}^{(t)}\right),$$

where η is a positive constant smaller than one. The computational complexity necessary for each step of the gradient descent method is $O(kn^2)$.

1.3 Stochastic mode of optimization

We have derived that the weight update rule depends on the data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ and the predictions on all training examples using the weight vector from the current step. We will first rewrite the update rule of the gradient descent method as

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta\mathbf{w}^{(t)},$$

where

$$\Delta\mathbf{w}^{(t)} = \eta\mathbf{X}^T\mathbf{P}^{(t)}\left(\mathbf{I} - \mathbf{P}^{(t)}\right)\left(\mathbf{y} - \mathbf{p}^{(t)}\right).$$

To simplify the following steps, we will remove the designation of step t and rewrite $\Delta\mathbf{w}$ as

$$\Delta\mathbf{w} = \eta \cdot \begin{bmatrix} p_1(1-p_1)x_{11} & p_2(1-p_2)x_{21} & \cdots & p_n(1-p_n)x_{n1} \\ p_1(1-p_1)x_{12} & p_2(1-p_2)x_{22} & & \\ \vdots & & \ddots & \\ p_1(1-p_1)x_{1k} & & & p_n(1-p_n)x_{nk} \end{bmatrix} \cdot \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$$

From here, we can see that the weight update is simply a linear combination of training data points

$$\Delta \mathbf{w} = \eta \sum_{i=1}^{n} e_i p_i (1 - p_i) \mathbf{x}_i,$$

This leads us to an interesting modification of the learning method. When the training data is large, it may be beneficial to update the weight vector after each data point is presented to the learning algorithm. That is, if data point $\mathbf{x}_i$ with its class label y_i is presented to the learner, we can modify the weight update to be

$$\Delta \mathbf{w} = \eta e_i p_i (1 - p_i) \mathbf{x}_i.$$

A method where weights are updated after seeing each individual data point is referred to as *incremental* or *stochastic gradient descent* method. Alternatively, the training algorithm that utilizes all data points is frequently referred to as *batch* mode of training. The training algorithm can now be revised to randomly draw one data point at a time from $\mathcal{D}$ and then update the current weights using the previous equation. The algorithm stops when the weight vector converges.

Observe that both stochastic and batch modes have the following property: the influence of each data point on the weight update depends on how close the data point is to the separation hyperplane and whether it lies on the correct side of it. The points on the correct side but far away from the decision boundary have negligible influence on $\Delta \mathbf{w}$ (this is because $p_i(1 - p_i) \approx 0$ and $e_i \approx 0$), the points on the incorrect side and far away from the decision boundary have a relatively larger influence (this is because $p_i(1 - p_i) \approx 0$ but and $|e_i| \approx 1$), whereas the points close to the decision boundary have the most individual influence (this is because $p_i(1 - p_i) \approx 0.25$ and $|e_i| \approx 0.5$).

1.4 Predicting class labels

For a previously unseen data point $\mathbf{x}$ and a set of coefficients $\mathbf{w}^*$ found from Eq. (4) or Eq. (11), we simply calculate the posterior probability as

$$P(Y = 1 | \mathbf{x}, \mathbf{w}^*) = \frac{1}{1 + e^{-\mathbf{w}^{*T} \cdot \mathbf{x}}}.$$

If $P(Y = 1|\mathbf{x}, \mathbf{w}^*) \geq 0.5$ we conclude that data point $\mathbf{x}$ should be labeled as positive ($\hat{y} = 1$). Otherwise, if $P(Y = 1|\mathbf{x}, \mathbf{w}^*) < 0.5$, we label the data point as negative ($\hat{y} = 0$). We can see this predictor as a simple mathematical function or as a computer code that outputs $P(Y = 1|\mathbf{x}, \mathbf{w}^*)$. Thus, the predictor maps a $(k + 1)$-dimensional vector $\mathbf{x} = (x_0 = 1, x_1, \ldots, x_k)$ into a zero or one. Note that $P(Y = 1|\mathbf{x}, \mathbf{w}^*) \geq 0.5$ only when $\mathbf{w}^T \mathbf{x} \geq 0$. Expression $\mathbf{w}^T \mathbf{x} = 0$ represents equation of a hyperplane that separates positive and negative examples. Thus, logistic regression model is a linear classifier.

It is also interesting to discuss the relationship between the posterior probability $P(Y = 1|\mathbf{x}, \mathbf{w})$ and distance of a data point $\mathbf{x}$ from the decision surface determined by $\mathbf{w}$. It can be shown that the distance from $\mathbf{w}^T \mathbf{x} = 0$ can be expressed as

$$d = \frac{w_0 + \sum_{j=1}^{k} w_j x_j}{\sqrt{\sum_{j=1}^{k} w_j^2}}.$$

We usually refer to d as signed distance because its sign determines on which side of the hyperplane the data point is found. This distance can also be expressed in a vector form as

$$d = \frac{w_0 + \mathbf{w}_-^T \mathbf{x}_-}{\|\mathbf{w}_-\|}$$

where $\mathbf{x}_- = (x_1, x_2, \ldots, x_k)$ and $\mathbf{w}_- = (w_1, w_2, \ldots, w_k)$. Thus, it follows straightforwardly that

$$P(Y = 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-d\|\mathbf{w}_-\|}}.$$

1.5 The significance of minimizing Euclidean distance

Consider a process of learning a classification model that is trained to minimize the Euclidean distance between the class labels y and soft predictions $f(\mathbf{x}, \mathbf{w})$. We shall assume binary classification on real-numbered inputs, i.e. $\mathcal{X} = \mathbb{R}^k$ and $\mathcal{Y} = \{0, 1\}$, and express the expectation of the squared error $e^2(\mathbf{w})$ as

$$E\left[e^2(\mathbf{w})\right] = \int_{\mathcal{X}} \sum_{\mathcal{Y}} \left(y - f(\mathbf{x}, \mathbf{w})\right)^2 p(\mathbf{x}, y) d\mathbf{x},$$

where $p(\mathbf{x}, y)$ is the true but unknown distribution of the data. Because $f(\mathbf{x}, \mathbf{w}) \in [0, 1]$ and $\mathcal{Y} = \{0, 1\}$ we have

$$y - f(\mathbf{x}, \mathbf{w}) = \begin{cases} -f(\mathbf{x}, \mathbf{w}) & y = 0 \\ \\ 1 - f(\mathbf{x}, \mathbf{w}) & y = 1 \end{cases}$$

Using $p(\mathbf{x}, y) = p(y|\mathbf{x})p(\mathbf{x})$, we can now modify the expected squared error as

$$E\left[e^2(\mathbf{w})\right] = \int_{\mathcal{X}} \left(f^2(\mathbf{x}, \mathbf{w})p(0|\mathbf{x}) + (1 - f(\mathbf{x}, \mathbf{w}))^2 p(1|\mathbf{x})\right) p(\mathbf{x}) d\mathbf{x}.$$

Assuming that $f(\mathbf{x}, \mathbf{w})$ is powerful enough to be independently optimized for each unit volume $d\mathbf{x}$, we observe that minimizing $E\left[e^2(\mathbf{w})\right]$ corresponds to finding a model that minimizes

$$f^2(\mathbf{x}, \mathbf{w})p(0|\mathbf{x}) + (1 - f(\mathbf{x}, \mathbf{w}))^2 p(1|\mathbf{x})$$

for each $d\mathbf{x}$. Minimizing the expression above results in

$$2f(\mathbf{x}, \mathbf{w}^*)p(0|\mathbf{x}) - 2(1 - f(\mathbf{x}, \mathbf{w}^*))p(1|\mathbf{x}) = 0$$

from where we infer that $f(\mathbf{x}, \mathbf{w}^*) = p(1|\mathbf{x})$; i.e. the classifier that minimizes Euclidean distance is learning the posterior probability distribution and makes an optimal decision for any given data point $\mathbf{x}$. The expected squared error for such a predictor can be expressed as

$$E\left[e^2(\mathbf{w}^*)\right] = \int_{\mathcal{X}} p(1|\mathbf{x})\left(1 - p(1|\mathbf{x})\right) p(\mathbf{x}) d\mathbf{x}.$$

Unfortunately, there are caveats to this argument. First, the model $f(\mathbf{x}, \mathbf{w})$ has to have enough flexibility to be able to learn the posterior probability of class 1 in $d\mathbf{x}$ independently of the rest of the input space. In addition, the data must be abundant to allow for such training and the optimization step must be able to find a global minimum (recall that the sum-of-squares objective function is not convex in classification). Linear classifiers are not flexible enough to be considered models that learn class posterior probabilities; they exhibit strong dependencies in predictions between faraway inputs. However, we will later see that classification models that are theoretically capable of learning the posterior distribution are neural networks.