Introduction to logistic regression

Given: dataset $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i$ is a k-dimensional vector of real-valued features (or attributes) and y_i is a binary class label (or target). Thus, we can say that $\mathbf{x}_i \in \mathbf{R}^k$ and $y_i \in \{0, 1\}$.

For example, if $k = 4$, a dataset of $n = 3$ data points (or examples) would be:

$\mathbf{x}_1 = (2.1\ 4.7\ -11\ 0.31)$ $\quad y_1 = 0$
$\mathbf{x}_2 = (4.0\ -1.3\ -1\ 0.001)$ $\quad y_2 = 1$
$\mathbf{x}_3 = (-2.1\ 2.7\ 1.4\ 2.1)$ $\quad y_3 = 0$

In this dataset, there is one positive data point ($\mathbf{x}_2$) and two negative data points ($\mathbf{x}_1$, $\mathbf{x}_3$). All vectors are 4-dimensional. Each of the four dimensions is called a feature (say, blood pressure, sugar level, bone density etc.). Each class label denotes one of the two groups, e.g. $y_i = 1$ may mean that patient with features $\mathbf{x}_i$ has a particular disease (or trait), while $y_i = 0$ would then mean that patient described with features $\mathbf{x}_i$ does not have that disease. In summary, the dataset D in this example consists of the 3 vectors of features, each vector being associated with a corresponding class label.

The *task of our data mining system* is to construct a predictor that would, for an unseen data point $\mathbf{x}$, infer its class label. For example, if a new patient comes and we perform four cheap tests, we might be able to infer a disease without running very expensive tests (which would effectively label that data point). Every inference is associated with a quality of inference. We will measure quality of inference by the number of mistakes the predictor makes (in percent) only for previously unseen data points. Formally, the quality of inference will be expressed through predictor accuracy.

You can see a *predictor* as a simple mathematical function. Thus, it virtually maps a k-dimensional vector ($\mathbf{x}$) into a zero or one (y). This type of a predictor is called a binary classifier; classifier because y can take discrete values and binary because there are only two such values that y can take. Of course, you can also imagine generalizations to this case. If y was taking values from the set of real numbers, we would call this process regression, instead of classification.

A core of any statistical (or machine learning) approach is to *assume* that vectors $\mathbf{x}$ and their labels y observed in D were generated by some source that outputs vectors $\mathbf{x}$ and labels y according to some probability distribution $p(\mathbf{x}, y)$. In such a case, we can express class membership probabilistically.

The basic idea for logistic regression approach is to try to establish a simple (possibly linear) closed-form dependency (meaning there is a formula) between the probability of a

class membership (called posterior probability) and the set of features. One such form could be

$$P(y_i = 1 | \mathbf{x}_i, \boldsymbol{\beta}) = \boldsymbol{\beta} \cdot \mathbf{x}_i^T \quad (1)$$

where $\boldsymbol{\beta} \in \mathbf{R}^k$ is a vector of k real-valued numbers (in general) and $\mathbf{x}^T$ is a transpose of the vector $\mathbf{x}$. Therefore, a dot product $\boldsymbol{\beta} \cdot \mathbf{x}^T$ results in a single number. For example, if $\boldsymbol{\beta}$ = (2 1 0.5 -1.3) and $\mathbf{x} = \mathbf{x}_1$, then

$$\begin{aligned}
\boldsymbol{\beta} \cdot \mathbf{x}_1^T &= (2\,1\,0.5\,-1.3) \cdot (2.1\,4.7\,-11\,0.31)^T = \\
&= (2\,1\,0.5\,-1.3) \cdot \begin{pmatrix} 2.1 \\ 4.7 \\ -11 \\ 0.31 \end{pmatrix} = \\
&= 2 \cdot 2.1 + 1 \cdot 4.7 + 0.5 \cdot (-11) + (-1.3) \cdot 0.31 = \\
&= 2.997
\end{aligned}$$

The problem with equation (1) is that $\boldsymbol{\beta} \cdot \mathbf{x}^T \in \mathbf{R}$, while the probability needs to be limited to the interval [0, 1]. Thus, we cannot use closed-form from equation (1).

Another approach to model the posterior probability is to try to express the odds function as a linear combination of the parameter vector $\boldsymbol{\beta}$ and feature vector $\mathbf{x}$. That is

$$\underbrace{\frac{P(y_i = 1 | \mathbf{x}_i, \boldsymbol{\beta})}{1 - P(y_i = 1 | \mathbf{x}_i, \boldsymbol{\beta})}}_{odds\ function} = \boldsymbol{\beta} \cdot \mathbf{x}_i^T \quad (2)$$

Closer look at this function (please verify this!) reveals that this function's co-domain is interval [0, ∞), while $\boldsymbol{\beta} \cdot \mathbf{x}^T \in \mathbf{R}$. Therefore, this is not an appropriate parametric dependency either.

Our third try will be to take a logarithm of the odds function and represent it as a linear combination of $\boldsymbol{\beta}$ and $\mathbf{x}$. That is

$$\log \frac{P(y_i = 1 | \mathbf{x}_i, \boldsymbol{\beta})}{1 - P(y_i = 1 | \mathbf{x}_i, \boldsymbol{\beta})} = \boldsymbol{\beta} \cdot \mathbf{x}_i^T \quad (3)$$

In this case both $\boldsymbol{\beta} \cdot \mathbf{x}^T$ and log(*odds*) belong to the interval (-∞, ∞). The logarithm in the expression above is to the base *e*, where *e* = 2.718281828...

A reorganization of the expression (3) gives the following

$$\frac{P(y=1|\mathbf{x},\boldsymbol{\beta})}{1-P(y=1|\mathbf{x},\boldsymbol{\beta})} = e^{\boldsymbol{\beta}\cdot\mathbf{x}^T}$$

$$P(y=1|\mathbf{x},\boldsymbol{\beta}) = e^{\boldsymbol{\beta}\cdot\mathbf{x}^T}\cdot(1-P(y=1|\mathbf{x},\boldsymbol{\beta}))$$

$$P(y=1|\mathbf{x},\boldsymbol{\beta}) = e^{\boldsymbol{\beta}\cdot\mathbf{x}^T} - e^{\boldsymbol{\beta}\cdot\mathbf{x}^T}\cdot P(y=1|\mathbf{x},\boldsymbol{\beta}))$$

$$P(y=1|\mathbf{x},\boldsymbol{\beta}) + e^{\boldsymbol{\beta}\cdot\mathbf{x}^T}\cdot P(y=1|\mathbf{x},\boldsymbol{\beta})) = e^{\boldsymbol{\beta}\cdot\mathbf{x}^T}$$

$$P(y=1|\mathbf{x},\boldsymbol{\beta})\cdot(1+e^{\boldsymbol{\beta}\cdot\mathbf{x}^T}) = e^{\boldsymbol{\beta}\cdot\mathbf{x}^T}$$

$$P(y=1|\mathbf{x},\boldsymbol{\beta}) = \frac{e^{\boldsymbol{\beta}\cdot\mathbf{x}^T}}{1+e^{\boldsymbol{\beta}\cdot\mathbf{x}^T}} = \frac{1}{1+e^{-\boldsymbol{\beta}\cdot\mathbf{x}^T}} \tag{4}$$

Therefore, a probability that the class of the vector **x** is 1 can in this case be modeled according to the expression (4).

The function

$$f(t) = \frac{1}{1+e^{-t}}$$

is called the sigmoid function or the logistic function (see the plot all the way at the bottom).

Determining optimal coefficients β

For a given dataset *D* and assumed dependency from expression (4) the optimal set of coefficients **β** is determined by maximizing the following expression (called likelihood function)

$$\prod_{i=1}^{n} P(y_i|\mathbf{x}_i,\boldsymbol{\beta}) \tag{5}$$

or in a formal mathematical notation

$$\boldsymbol{\beta}^* = \arg\max_{\boldsymbol{\beta}}\left\{\prod_{i=1}^{n} P(y_i|\mathbf{x}_i,\boldsymbol{\beta})\right\}$$

which says that vector $\boldsymbol{\beta}^*$ is the one for which expression (5) is maximal.

Now, let's use the following representation (you can easily verify it is true by substituting y_i)

$$P(y_i \mid \mathbf{x}_i, \boldsymbol{\beta}) = \begin{cases} \left(\frac{1}{1+e^{-\boldsymbol{\beta}\cdot\mathbf{x}_i^T}} \right)^{y_i} & \text{for } y_i = 1 \\ \left(1 - \frac{1}{1+e^{-\boldsymbol{\beta}\cdot\mathbf{x}_i^T}} \right)^{1-y_i} & \text{for } y_i = 0 \end{cases}$$

Now, we can maximize the following expression (note that if y_i = 1 the first part disappears, while if y_i = 1 the second part disappears), which follows from expression (4)

$$\prod_{i=1}^{n} \left(\frac{1}{1+e^{-\boldsymbol{\beta}\cdot\mathbf{x}_i^T}} \right)^{y_i} \cdot \left(1 - \frac{1}{1+e^{-\boldsymbol{\beta}\cdot\mathbf{x}_i^T}} \right)^{1-y_i}$$

or after taking a logarithm (to the base *e*)

$$\sum_{i=1}^{n} y_i \cdot \log\left(\frac{1}{1+e^{-\boldsymbol{\beta}\cdot\mathbf{x}_i^T}} \right) + (1-y_i) \log\left(1 - \frac{1}{1+e^{-\boldsymbol{\beta}\cdot\mathbf{x}_i^T}} \right)$$

I assumed here you are familiar with some basic logarithm manipulations. This equation is maximized using iterative optimization process. Determining optimal coefficients $\boldsymbol{\beta}^*$ is called training or learning (hence the name "machine learning"). Formally, this whole optimization process is called maximum likelihood optimization and there are various toolboxes on the Internet that can do this for you (may be pretty complex).

The training process is now over. We just calculated $\boldsymbol{\beta}^*$.

How do we predict or infer class for a new data point?

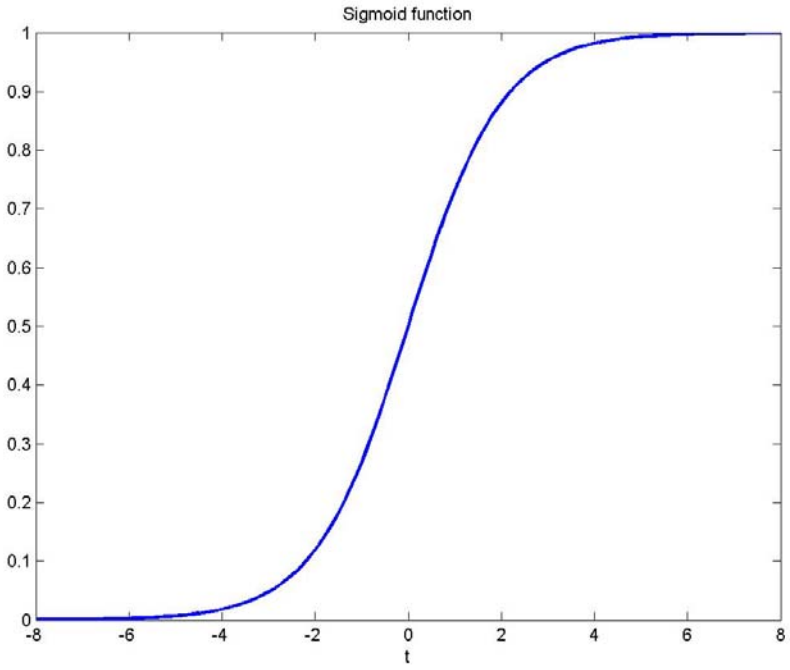
For an unseen data point **x**, and optimal (or, sometimes, nearly optimal) set of coefficients $\boldsymbol{\beta}^*$ (determined from a training dataset *D*) we simply calculate the following expression

$$P(y = 1 \mid \mathbf{x}, \boldsymbol{\beta}^*) = \frac{1}{1+e^{-\boldsymbol{\beta}^*\cdot\mathbf{x}^T}}$$

If $P(y = 1 \mid \mathbf{x}, \boldsymbol{\beta}^*) \geq 0.5$ we simply conclude that the data point **x** should be labeled as positive. On the other hand, if $P(y = 1 \mid \mathbf{x}, \boldsymbol{\beta}^*) < 0.5$ we label the unseen data point **x** as negative.

The prediction can be made even without calculating the logistic expression: if $\boldsymbol{\beta} \cdot \mathbf{x}^T \geq 0$, we predict positive, and otherwise we predict negative (you can easily verify this by substituting $\boldsymbol{\beta} \cdot \mathbf{x}^T$ in the expression above)

What is the shape of the sigmoid function?



Final point

Although the assumption that

$$P(y=1|\mathbf{x},\boldsymbol{\beta}) = \frac{1}{1+e^{-\boldsymbol{\beta}\cdot\mathbf{x}^T}}$$

may not look intuitive, logistic regression has been shown to work surprisingly well in practice!!!

K-nearest neighbor algorithm

K-NEAREST NEIGHBOR is a simple algorithm that stores all available data points (examples) and classifies new data points based on a similarity measure. A variant of this algorithm addresses the task of function approximation.

In detail:

- Examples are described by **numerical attribute-values**. That is

  $\mathbf{x} = (x_1, x_2, x_3, ..., x_k)$ where k is the dimensionality of the instance space

- The complete example (or data) set D is simply stored in the "training phase". The data set D is defined as

  $D = \{(\mathbf{x}_i, y_i) \mid i = 1..n\}, \mathbf{x}_i \in \mathbf{R}^k, y_i \in \{0, 1\}$

- Calculations are delayed until queries occur, **no trained model in the usual sense** is returned! Trained models are implicitly defined by the stored example set and the rules new instances are classified by.
- If there are k attributes all vectors can be interpreted as instances of $\mathbf{R}^k$. The distance $d(\mathbf{x_1}, \mathbf{x_2})$ of two example vectors $\mathbf{x_1}$ and $\mathbf{x_2}$ is defined as their usual vector distance (Euclidean distance). That is:

  $$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{(x_{11} - x_{21})^2 + (x_{12} - x_{22})^2 + (x_{13} - x_{23})^2 + \ldots + (x_{1k} - x_{2k})^2}$$

- The **distance** between two example vectors is regarded as a **measure for their similarity**. The smaller the distance the more similar the vectors.
- To classify a new instance **x** from the set of stored examples, the K examples most similar to **x** are determined. The new instance is assigned the class label most of these K examples belong to.

This approach is suited for **function approximation** as well. Instead of assigning the most frequent classification among the K examples most similar to an instance **x**, an average of the function values of the K examples is calculated as the prediction for the function value **x**.

A variant of this approach calculates a **weighted average** of the nearest neighbors. Given a specific instance **x** that shall be classified, the weight of an example increases with increasing similarity to **x**.

A major problem of the simple approach of K-NEAREST NEIGHBOR is that the vector distance will not necessarily be suited for finding intuitively similar examples, especially if irrelevant attributes are present.

Source (modified): http://kiew.cs.uni-dortmund.de:8001/mlnet/instances/81d91e93-df4da3c279