Logistic regression

Despite sharing a few similarities with linear regression, **logistic regression** is actually a **classification** method, where we train a model that will predict the value for a **discrete-valued** output variable, unlike the real-valued output that a linear model is trained to predict.

There are two types of logistic regression: **binomial** and **multinomial**. These terms refer to the nature of the output variable.

Binomial logistic regression

In **binomial logistic regression**, the output can be modeled as a binomial variable. Our training data in this case looks like:

$$\mathcal{D}_{\text{train}} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) \right\}_{i=1}^{N}$$

> **Where**:
>
> - $\mathbf{x}^{(i)} \in \mathbb{R}^D$, a $(D+1)$-dimensional vector of real numbers, $\mathbf{x}^{(i)} = \left(1, x_1^{(i)}, \ldots, 1, x_D^{(i)}\right)^\mathsf{T}$.
>
>   **Note**: In the *Linear Regression* notes, $\mathbf{x}^{(i)}$ represented the original unextended feature vector without the additional 1. In logistic regression, this vector includes the 1 by convention. The same applies for the weight vector $\boldsymbol{\theta}$ (additional bias term θ_0) that we will see later.
>
>   However, it is still possible to apply a vector of basis functions $\boldsymbol{\phi}$ to this vector if you wish.
> - $y^{(i)} \in \{0, 1\}$

As explained in the *Linear Regression* notes, this can be represented as an $N \times D$ matrix of **feature vectors** $\mathbf{X} = \left(\mathbf{x}^{(1)^\mathsf{T}}, \ldots, \mathbf{x}^{(N)^\mathsf{T}}\right)^\mathsf{T}$ and a corresponding **output vector** $\mathbf{y} = \left(y^{(1)}, \ldots, y^{(N)}\right)^\mathsf{T}$.

Each of the columns $\mathbf{x}_j = \left(x_j^{(1)}, \ldots, x_j^{(N)}\right)^\mathsf{T}$ of $\mathbf{X}$ represents a **feature**, which is simply a **random variable** (sometimes called an **explanatory variable**).

$\mathbf{X}$	$\mathbf{x}_0$	$\mathbf{x}_1$	$\mathbf{x}_2$	$\cdots$	$\mathbf{x}_D$	$\mathbf{y}$
$\mathbf{x}^{(1)\mathsf{T}}$	$x_0^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$	$\cdots$	$x_D^{(1)}$	$y^{(1)}$
$\mathbf{x}^{(2)\mathsf{T}}$	$x_0^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$	$\cdots$	$x_D^{(2)}$	$y^{(2)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$\mathbf{x}^{(N)\mathsf{T}}$	$x_0^{(N)}$	$x_1^{(N)}$	$x_2^{(N)}$	$\cdots$	$x_D^{(N)}$	$y^{(N)}$

The accompanying output variable $\mathbf{y}$ contains the output for each feature vector.

> In linear regression, the output $y^{(i)}$ is assumed to be a **linear combination** of its feature values $\mathbf{x}^{(i)}$—that is:
>
> $$\begin{aligned} y^{(i)} &= \theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_D x_D^{(i)} \\ &= \sum_{j=0}^{D} \theta_j x_j^{(i)} \\ &= \boldsymbol{\theta}^\mathsf{T} \mathbf{x}^{(i)} \end{aligned}$$
>
> **Where**:
>
> - $\theta_j \in \mathbb{R}$ is an arbitrary coefficient, referred to as a **weight**.
> - $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_D)^\mathsf{T}$ is a **weight vector**.

However, modeling the output as a linear combination of the feature values won't work for logistic regression, as the output needs to be discrete (0 or 1).

One way to ensure an output fits these constraints is to transform the real-valued $\boldsymbol{\theta}^\mathsf{T} \mathbf{x}^{(i)}$ with some function σ, referred to as a **sigmoid function**.

Sigmoid functions

A **sigmoid function** is a function with the domain of all real numbers, that returns a **monotonically increasing** value ranging from 0 to 1—that is:

$$\sigma : \mathbb{R} \to (0, 1) \qquad \text{s.t.} \quad \forall (x, y) \in \mathbb{R}^2,\ x \leq y \implies \sigma(x) \leq \sigma(y)$$

There are many functions that satisfy these conditions:

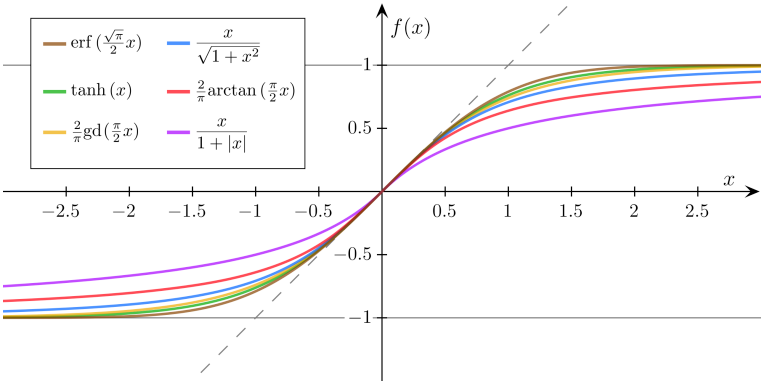

Figure 1: A collection of sigmoid functions.
***Note**: Despite these sigmoids having a range of (-1,1), they can be transformed to (0,1).* (source)

However, the most commonly used sigmoid function for logistic regression is the appropriately named **logistic function**.

Logistic function

The **logistic function** is a sigmoid function that is used in logistic regression to transform the real-valued $\boldsymbol{\theta}^\mathsf{T}\mathbf{x}^{(i)}$ into a value in the $(0, 1)$ interval. This function is denoted σ, and is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

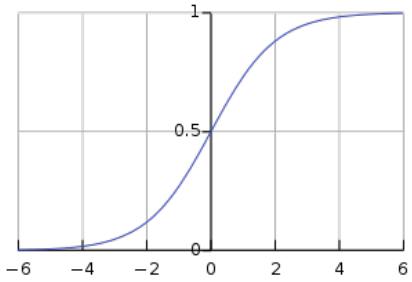

Figure 2: The logistic function. (source)

Probabilistic decision-making

Despite the logistic function mapping the real-valued output $y^{(i)}$ to the range $[0, 1]$, we **cannot** simply model this output as the application of the logistic function to $\boldsymbol{\theta}^\mathsf{T}\mathbf{x}^{(i)}$:

$$\begin{aligned} y^{(i)} &= \sigma\left(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_D x_D^{(i)}\right) \\ &= \sigma\left(\sum_{j=0}^{D} \theta_j x_j^{(i)}\right) \\ &= \sigma\left(\boldsymbol{\theta}^\mathsf{T}\mathbf{x}^{(i)}\right) \\ &= \frac{1}{1 + e^{-\boldsymbol{\theta}^\mathsf{T}\mathbf{x}^{(i)}}} \end{aligned}$$

This is problematic because $y^{(i)}$ may still be any real number in $[0, 1]$—we still need to "decide" on whether to assign this number to the 0 or 1 class. Intuitively, this task is well-suited to probabilistic decision-making since we are already dealing with numbers between 0 and 1.

We can model the decision of $y^{(i)}$ with:

$$y^{(i)} = \begin{cases} 1 & \text{if} \quad \mathbb{P}\left(y^{(i)} = 1 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right) \geq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \qquad \text{Or equivalently:}$$
$$= \begin{cases} 0 & \text{if} \quad \mathbb{P}\left(y^{(i)} = 0 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right) > \frac{1}{2} \\ 1 & \text{otherwise} \end{cases}$$

Where the conditional probability $\mathbb{P}\left(y^{(i)} = 1 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right)$ is the result of applying the logistic function to $\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}$ as seen before:

$$\begin{aligned} \mathbb{P}\left(y^{(i)} = 1 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right) &= \sigma\left(\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}\right) \\ &= \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}} \end{aligned}$$

> **Note**: Observe that $\mathbb{P}\left(y^{(i)} = 1 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right)$ and $\mathbb{P}\left(y^{(i)} = 0 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right)$ must form a probability distribution, meaning that:
>
> $$\mathbb{P}\left(y^{(i)} = 1 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right) + \mathbb{P}\left(y^{(i)} = 0 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right) = 1$$

Decision boundary

In a classification problem with two classes, a **decision boundary** is a **hypersurface** that partitions the underlying feature space into two sets, one for each class.

In the case of binomial logistic regression (**without** a feature transformation with basis functions), this decision boundary is linear—a **hyperplane**. The decision boundary would occur when $\mathbb{P}\left(y^{(i)} = 1 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right) = \frac{1}{2}$, as we are completely uncertain which class to assign when the probability is $\frac{1}{2}$:

$$\begin{aligned} \mathbb{P}\left(y^{(i)} = 1 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right) &= \frac{1}{2} \\ \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}} &= \frac{1}{2} \\ 1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}} &= 2 \\ e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}} &= 1 \\ -\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)} &= 0 \qquad (\text{By taking natural logarithms}) \\ \boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)} &= 0 \end{aligned}$$

> **Note**: Observe that the decision boundary is linear in $\mathbf{x}^{(i)}$. However, feature transformations with basis functions ϕ can result in a non-linear decision boundary.

Learning task

Because logistic regression actually predicts probabilities rather than just classes, we can estimate the weights using **maximum likelihood estimation**. This requires us to define the distribution of the output variable $y^{(i)}$, and the likelihood function for this distribution.

Bernoulli distributed output variable

Recall that the Bernoulli distribution is the discrete probability distribution of a random variable which takes the value 1 with probability p, and the value 0 with probability $q = 1 - p$. In other words, the Bernoulli distribution models a single experiment that flips a (potentially biased) coin, with probability p of the coin landing on heads, and probability $q = 1 - p$ of the coin landing on tails.

As we have a binary output variable with probability $p = \mathbb{P}\left(y^{(i)} = 1 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right)$ of taking value 1, and probability $q = \mathbb{P}\left(y^{(i)} = 0 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right)$ (or $1 - p = 1 - \mathbb{P}\left(y^{(i)} = 1 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right)$) of taking value 0, we can say that the output variable is distributed as a Bernoulli random variable.

$$y^{(i)} \sim \text{Bernoulli}\left(p^{(i)}\right)$$

> **Where**: $p^{(i)} = \mathbb{P}\left(y^{(i)} = 1 \middle| \mathbf{x}^{(i)}; \boldsymbol{\theta}\right)$ is the parameter for this Bernoulli distribution.

Error function and maximum likelihood estimation

In linear regression, we solve ordinary least squares (OLS) problems which use the residual sum of squares (RSS) as an error function. However, we typically don't use this for logistic regression. As we have a discrete probability distribution with parameters, we can instead use **maximum likelihood estimation** to determine the weights for the model.

The **likelihood function** L for a Bernoulli distributed output variable $y^{(i)} \sim \text{Bernoulli}\left(p^{(i)}\right)$ is given by:

$$L\left(\boldsymbol{\theta}; \mathbf{y}\right) = \prod_{i=1}^{N} \left(p^{(i)}\right)^{y^{(i)}} \left(1 - p^{(i)}\right)^{1 - y^{(i)}}$$

A maximum likelihood estimate $\hat{\boldsymbol{\theta}}$ for the weights $\boldsymbol{\theta}$ can be found by taking the derivative of L with respect to $\boldsymbol{\theta}$ and set this equal to 0.

To simplify the process of taking derivatives, we can apply natural logarithms to give us the **log-likelihood** $\mathcal{L}$, which we would still be finding the maximum of, since the natural logarithm function is monotonically increasing:

$$\forall (x_1, x_2) \in \mathbb{R}^2 \land x_1, x_2 > 0, \quad x_1 \geq y_2 \implies \log x_1 \geq \log x_2$$

We can now write the log-likelihood as:

$$\begin{aligned}
\mathcal{L}\left(\boldsymbol{\theta}; \mathbf{y}\right) &= \log L\left(\boldsymbol{\theta}; \mathbf{y}\right) \\
&= \log \prod_{i=1}^{N} \left(p^{(i)}\right)^{y^{(i)}} \left(1 - p^{(i)}\right)^{1 - y^{(i)}} \\
&= \sum_{i=1}^{N} \log \left[\left(p^{(i)}\right)^{y^{(i)}} \left(1 - p^{(i)}\right)^{1 - y^{(i)}}\right] \\
&= \sum_{i=1}^{N} \left[\log \left(p^{(i)}\right)^{y^{(i)}} + \log \left(1 - p^{(i)}\right)^{1 - y^{(i)}}\right] \\
&= \sum_{i=1}^{N} \left[y^{(i)} \log\left(p^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - p^{(i)}\right)\right]
\end{aligned}$$

This allows us to represent the maximization problem for the optimal weights as:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\theta}; \mathbf{y}\right)$$

However, in machine learning it is convention that optimization problems are minimization problems. This can be done by simply negating the function being optimized, if it is currently a maximization problem:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} -\mathcal{L}\left(\boldsymbol{\theta}; \mathbf{y}\right)$$

> **Where**: The **negative log-likelihood** $-\mathcal{L}(\boldsymbol{\theta}; \mathbf{y})$ is the **error function** $C(\boldsymbol{\theta})$ for our model.

> In the *Linear Regression* notes, we saw that the following minimization problem for OLS:
>
> $$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \left(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\right)^{\mathsf{T}} \left(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\right)$$
>
> Had a closed-form (analytical) solution—$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathsf{T}}\mathbf{y}$.

This is not the case for the MLE in logistic regression. We must take the derivative of $C(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. This can be done by taking partial derivatives, since:

$$\nabla C(\boldsymbol{\theta}) = \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}} C(\boldsymbol{\theta}) = \left(\frac{\partial}{\partial\theta_0} C(\boldsymbol{\theta}), \frac{\partial}{\partial\theta_1} C(\boldsymbol{\theta}), \ldots, \frac{\partial}{\partial\theta_N} C(\boldsymbol{\theta})\right)^{\mathsf{T}}$$

> **Where**: $\nabla C(\boldsymbol{\theta})$ is called the **gradient** (in vector calculus).

An individual partial derivative of $C(\boldsymbol{\theta})$ with respect to θ_j would be given by:

Simplifying:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} C(\boldsymbol{\theta}) &= \frac{\partial}{\partial \theta_j} \sum_{i=1}^{N} \left[-y^{(i)} \log\left(p^{(i)}\right) - \left(1 - y^{(i)}\right) \log\left(1 - p^{(i)}\right) \right] \\
&= \frac{\partial}{\partial \theta_j} \sum_{i=1}^{N} \left[-y^{(i)} \log\left(\sigma\left(\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}\right)\right) \right] \\
&= \frac{\partial}{\partial \theta_j} \sum_{i=1}^{N} \left[-y^{(i)} \log\left(\frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}}\right) - \left(1 - y^{(i)}\right) \log\left(1 - \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}}\right) \right] \\
&= \frac{\partial}{\partial \theta_j} \sum_{i=1}^{N} \left[-y^{(i)} \log\left(\frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}}\right) - \left(1 - y^{(i)}\right) \log\left(\frac{e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}}\right) \right] \\
&= \frac{\partial}{\partial \theta_j} \sum_{i=1}^{N} \left[y^{(i)} \log\left(1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}\right) - \left(1 - y^{(i)}\right) \log\left(e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}\right) + \left(1 - y^{(i)}\right) \log\left(1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}\right) \right] \\
&= \frac{\partial}{\partial \theta_j} \sum_{i=1}^{N} \left[\log\left(1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}\right) + (1 - y^{(i)}) \boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)} \right]
\end{aligned}$$

Taking partial derivatives:

$$= \sum_{i=1}^{N} \left[\frac{\left(-x_j^{(i)}\right) e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}} + \left(1 - y^{(i)}\right) x_j^{(i)} \right]$$

Factoring out $x_j^{(i)}$ and simplifying:

$$\begin{aligned}
&= \sum_{i=1}^{N} \left[\left(\frac{-e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}} + 1 - y^{(i)} \right) x_j^{(i)} \right] \\
&= \sum_{i=1}^{N} \left[\left(\frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}}} - y^{(i)} \right) x_j^{(i)} \right] \\
&= \sum_{i=1}^{N} \left[\left(\sigma\left(\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)}\right) - y^{(i)} \right) x_j^{(i)} \right]
\end{aligned}$$

To obtain the θ_j that minimizes this, we would set it equal to zero and solve. However, this is not possible as it is a transcendental equation, and there is no closed-form solution.

Instead, we can utilize numerical optimization methods to approximate a solution—namely, **gradient descent**. However, there are many other optimization methods such as **L-BFGS**, **coordinate descent** or **stochastic gradient descent**.

Gradient descent

In order to minimize our error function, we will use **gradient descent**.

> **Analogy**: A common analogy for gradient descent is one in which we imagine a hiker at the top of a mountain valley, left stranded and blindfolded at the top of a mountain. The objective of the hiker is to reach the bottom of the mountain.
>
> The most intuitive way to approach this would be to feel the slope in different directions around your feet, take one step in the direction with the steepest slope, and repeat.
>
> This action is analogous to the way in which gradient descent works.

The gradient $\nabla C(\boldsymbol{\theta})$ points in the direction of steepest **ascent** (at the current position $\boldsymbol{\theta}$). We are interested in "taking a step" in the direction of steepest descent, $-\nabla C(\boldsymbol{\theta})$.

By "taking a step", we mean moving some distance η in a particular direction in $\boldsymbol{\theta}$-space.

> In the hiker analogy, suppose we have the vertical y-axis in which we are trying to minimize, along with the two axes in the horizontal plane of the hiker, x_1 and x_2. In this case, we "take a step" in the two-dimensional $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$-space, which results in a change in the vertical y position of the hiker.

A step of size η in the direction of steepest descent $-\nabla C(\boldsymbol{\theta})$ (in $\boldsymbol{\theta}$-space) would be represented by the vector $-\eta \nabla C(\boldsymbol{\theta})$. If we take our current position $\boldsymbol{\theta}$ and apply this step by adding the transformation vector $-\eta \nabla C(\boldsymbol{\theta})$, we arrive at our updated position:

$$\boldsymbol{\theta}^{\text{new}} \leftarrow \boldsymbol{\theta}^{\text{old}} - \eta \nabla C(\boldsymbol{\theta})$$

> **Where**: η is referred to as the **learning rate**.

This would be repeated until convergence.

It is also possible to update individual components separately for this step—updating one component **once** then moving on to the next, until we have a completely updated weight vector $\boldsymbol{\theta}^{\text{new}}$. We would repeat this until convergence—when the components of $\boldsymbol{\theta}$ no longer change:

$$\theta_j^{\text{new}} \leftarrow \theta_j^{\text{old}} - \eta \frac{\partial}{\partial \theta_j} C(\boldsymbol{\theta})$$

Coordinate descent

It is also possible to consider an individual direction or component θ_j at a time when taking steps—completely focusing one component, but updating it **multiple** times (until convergence) to minimize this component before returning to minimize the others.

$$\theta_j^{\text{new}} \leftarrow \theta_j^{\text{old}} - \eta \frac{\partial}{\partial \theta_j} C(\boldsymbol{\theta})$$

Once this component is minimized, this would then be repeated for all remaining components. This variation of gradient descent is referred to as **coordinate descent**.

Multinomial logistic regression

Although there may be many tasks involving binary output variables, many classification tasks naturally involve multiple output labels, such as:

- Hand-written digit recognition (labels are the digits 0-9)
- Fruit recognition (labels are digits representing the type of fruit, e.g. $0 = \text{Apple}$, $1 = \text{Banana}$, ...)
- Assigning blog posts to a category (labels are digits representing the categories)
- Blood type classification (labels are digits representing the blood types)

For these kinds of tasks, the binomial logistic regression that was previously introduced won't work. However, it is possible to generalize binomial logistic regression problems to multi-class problems. This generalized classification method is referred to as **multinomial logistic regression**, but is also known under other names such as **softmax regression** or **maximum entropy classifier**.

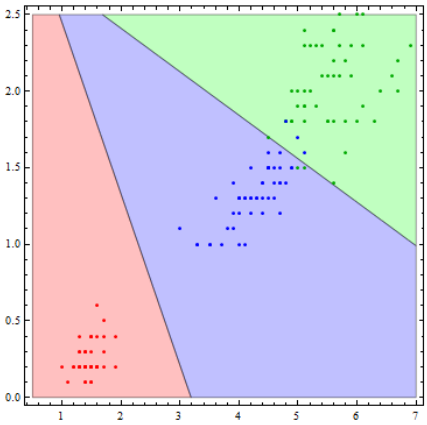

Figure 3: Division of the feature-space of a dataset into three decision regions by a classifier such as multinomial logistic regression that can generate multiple decision boundaries (each being linear in this case). (source)

Given training data $\mathcal{D}_{\text{train}} = \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$ where $y^{(i)} \in \{1, \ldots, K\}$, we can create a separate weight vector $\boldsymbol{\theta}^{(k)} = \left(\theta_0^{(k)}, \ldots, \theta_D^{(k)}\right)^{\mathsf{T}}$ for each class $k \in K$. We can then classify a feature vector $\mathbf{x}^{(i)}$ as k or not-k, through the use of the **softmax** function, which is simply a normalized exponential function:

$$\begin{aligned}\mathbb{P}\left(y^{(i)} = c \middle| \mathbf{x}^{(i)}; \boldsymbol{\Theta}\right) &= \sigma_c\left(\mathbf{x}^{(i)}\right) \\ &= \frac{\exp\left(\boldsymbol{\theta}^{(c)\mathsf{T}}\mathbf{x}^{(i)}\right)}{\sum_{k=1}^{K}\exp\left(\boldsymbol{\theta}^{(k)\mathsf{T}}\mathbf{x}^{(i)}\right)}\end{aligned}$$

> **Where**:
>
> - $\boldsymbol{\Theta} = \left(\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(K)}\right)$, a matrix where each column represents a vector of weights $\boldsymbol{\theta}^{(c)}$ for class c.
> - $\frac{1}{\sum_{k=1}^{K}\exp\left(\boldsymbol{\theta}^{(k)\mathsf{T}}\mathbf{x}^{(i)}\right)}$ is a normalization term, used to ensure that the distribution sums up to one.

To decide on a class for $\mathbf{x}^{(i)}$, we must calculate $\mathbb{P}\left(y^{(i)} = c \middle| \mathbf{x}^{(i)}; \boldsymbol{\Theta}\right)$ for every class c. The chosen class for $\mathbf{x}^{(i)}$ is then the one that results in the maximum conditional probability—that is:

$$\hat{c} = \underset{c \in \{1,\ldots,K\}}{\arg\max}\ \mathbb{P}\left(y^{(i)} = c \middle| \mathbf{x}^{(i)}; \boldsymbol{\Theta}\right)$$

Learning the weights

Due to having a separate weight vector for each class, our negative log-likelihood error function will have to be modified to account for the softmax function—this can be done by summing over all classes in addition to summing over the examples. The modified negative log-likelihood function looks like:

$$
\begin{aligned}
C(\boldsymbol{\Theta}) &= -\mathcal{L}\left(\boldsymbol{\Theta}; \mathbf{y}\right) \\
&= -\sum_{i=1}^{N}\sum_{c=1}^{K}\left[y^{(i)}\log\left(p_c^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-p_c^{(i)}\right)\right]
\end{aligned}
$$

> **Where**:
>
> - $p_c^{(i)} = \mathbb{P}\left(y^{(i)} = c \middle| \mathbf{x}^{(i)}; \boldsymbol{\Theta}\right) = \dfrac{\exp\left(\boldsymbol{\theta}^{(c)\mathsf{T}}\mathbf{x}^{(i)}\right)}{\sum_{k=1}^{K}\exp\left(\boldsymbol{\theta}^{(k)\mathsf{T}}\mathbf{x}^{(i)}\right)}$

Once again, this cannot be minimized with an analytic solution—we must resort to gradient descent (or another numerical optimization method).

Gradient descent with multiple weight vectors

As a result of having to use an individual vector of weights for each class, we will introduce a vector $\boldsymbol{\nabla} C(\boldsymbol{\Theta})$ of the gradients for each class, $\nabla_{\boldsymbol{\theta}^{(c)}} C(\boldsymbol{\Theta})$:

$$
\boldsymbol{\nabla} C(\boldsymbol{\Theta}) = \begin{pmatrix} \nabla_{\boldsymbol{\theta}^{(1)}} C(\boldsymbol{\Theta}) \\ \nabla_{\boldsymbol{\theta}^{(2)}} C(\boldsymbol{\Theta}) \\ \vdots \\ \nabla_{\boldsymbol{\theta}^{(K)}} C(\boldsymbol{\Theta}) \end{pmatrix}^{\mathsf{T}} = \begin{pmatrix} \nabla_{\boldsymbol{\theta}^{(1)}} C(\boldsymbol{\Theta}) & \nabla_{\boldsymbol{\theta}^{(2)}} C(\boldsymbol{\Theta}) & & \nabla_{\boldsymbol{\theta}^{(K)}} C(\boldsymbol{\Theta}) \\ \frac{\partial}{\partial \theta_1^{(1)}} C(\boldsymbol{\Theta}) & \frac{\partial}{\partial \theta_1^{(2)}} C(\boldsymbol{\Theta}) & \cdots & \frac{\partial}{\partial \theta_1^{(K)}} C(\boldsymbol{\Theta}) \\ \frac{\partial}{\partial \theta_2^{(1)}} C(\boldsymbol{\Theta}) & \frac{\partial}{\partial \theta_2^{(2)}} C(\boldsymbol{\Theta}) & \cdots & \frac{\partial}{\partial \theta_2^{(K)}} C(\boldsymbol{\Theta}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial \theta_N^{(1)}} C(\boldsymbol{\Theta}) & \frac{\partial}{\partial \theta_N^{(2)}} C(\boldsymbol{\Theta}) & \cdots & \frac{\partial}{\partial \theta_N^{(K)}} C(\boldsymbol{\Theta}) \end{pmatrix}
$$

> **Where**: $\nabla_{\boldsymbol{\theta}^{(c)}} C(\boldsymbol{\Theta}) = \left(\frac{\partial}{\partial \theta_1^{(c)}} C(\boldsymbol{\Theta}), \frac{\partial}{\partial \theta_2^{(c)}} C(\boldsymbol{\Theta}), \cdots, \frac{\partial}{\partial \theta_N^{(c)}} C(\boldsymbol{\Theta})\right)^{\mathsf{T}}$, the derivative of the error function $\nabla C(\boldsymbol{\Theta})$ with respect to the weight vector $\boldsymbol{\theta}^{(c)}$ for class c. In other words, this represents the gradient of the error function for class c.

Taking derivatives of $C(\boldsymbol{\Theta})$, one can show that the gradient is given by:

$$
\nabla_{\boldsymbol{\theta}^{(c)}} C(\boldsymbol{\Theta}) = -\sum_{i=1}^{N}\left(\mathbb{I}_c^{(i)} - p_c^{(i)}\right)\mathbf{x}^{(i)}
$$

> **Where**:
>
> - $\mathbb{I}_c^{(i)}$ is an **indicator function** such that $\mathbb{I}_c^{(i)} = \begin{cases} 1 & \text{if } y^{(i)} = c \\ 0 & \text{otherwise} \end{cases}$
> - $p_c^{(i)} = \mathbb{P}\left(y^{(i)} = c \middle| \mathbf{x}^{(i)}; \boldsymbol{\Theta}\right)$

The weight vector $\boldsymbol{\theta}^{(c)}$ can then be optimized using gradient descent:

$$
\boldsymbol{\theta}^{(c)} \leftarrow \boldsymbol{\theta}^{(c)} - \eta \nabla_{\boldsymbol{\theta}^{(c)}} C(\boldsymbol{\Theta})
$$

We iteratively update this weight vector until it converges, then we proceed to the next vector of the weight matrix. Alternatively, it is possible to update the entire weight matrix until it converges:

$$\Theta \leftarrow \Theta - \eta \nabla C(\Theta)$$

Regularization

Just as we can add a regularization term to the RSS used as an error function in linear regression (in order to simplify the model and prevent overfitting to the training data), it is also possible to add a regularization term to the negative log-likelihood, and use this sum as a new, regularized error function. For binomial logistic regression:

$$C(\boldsymbol{\theta}) = -\mathcal{L}\left(\boldsymbol{\theta}; \mathbf{y}\right) + \lambda \underbrace{R(\boldsymbol{\theta})}_{\text{reg. term}}$$

> **Where**:
>
> - λ is a **tuning parameter** that controls the importance of the regularization term—higher λ leads to more penalization. This parameter is selected through cross-validation.
> - $R(\boldsymbol{\theta})$ is a regularization term chosen to penalize coefficients by a specific quantity—shrinking them towards zero.

For more general information about regularization and how the L_1 and L_2 norms (along with ElasticNet) can be used as the regularization term, please read the *Linear Regression* notes—these regularization methods apply in the same way to logistic regression.

Resources

- *Nigel Goddard (School of Informatics, University of Edinburgh)*
  Introductory Applied Machine Learning: Logistic Regression - Two-Class Linear Classifier
  Introductory Applied Machine Learning: Logistic Regression - Logistic Regression
  Introductory Applied Machine Learning: Logistic Regression - Learning the Parameters
  Introductory Applied Machine Learning: Logistic Regression - Multiclass Classification
- *Iain Murray (School of Informatics, University of Edinburgh)*
  Logistic Regression
- *Michael F. Brannick (University of South Florida)*
  Logistic Regression
- *Cosma Shalizi (Department of Statistics, Carnegie Mellon University)*
  Logistic Regression
- *Saishruthi Swaminathan (Towards Data Science)*
  Logistic Regression — Detailed Overview
- *Wei Xu (Department of Computer Science and Engineering, Ohio State University)*
  Multi-Class Logistic Regression and Perceptron
- *Wikipedia*
  Decision boundary
  Logistic regression
  Monotonic function
  Sigmoid function
  Bernoulli distribution
  Multinomial logistic regression
  Maximum likelihood estimation
- *Trevor Hastie (Department of Statistics, Stanford University)*
  Fast Regularization Paths via Coordinate Descent

- *Chandler Watson (Stack Overflow)*
  Proof that conditional probabilities sum up to one
- *Dan Nettleton (Iowa State University)*
  A Generalized Linear Model for Bernoulli Response Data
- *Mark (Stack Overflow)*
  Logistic Regression: Bernoulli vs. Binomial Response Variables
- *Neos Guide (University of Wisconsin-Madison)*
  Maximum Likelihood Estimation with Logit Model
- *Dan Nuttle (RPubs)*
  Partial Derivative of Cost Function for Logistic Regression
- *Ayush Pant (Towards Data Science)*
  Introduction to Logistic Regression
- *Ashutosh Singh, Yanxin Li*
  To Study Implementation of Gradient Descent for Multi-class Classification Using a SoftMax Regression and Neural Networks
- *Arthur Juliani (Medium)*
  Simple Softmax Regression in Python — Tutorial
- *Aanish Singla (Analytics Vidhya)*
  An Introductory Guide to Maximum Likelihood Estimation (with a case study in R)
- *Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen, Adam Coates, Andrew Maas, Awni Hannun, Brody Huval, Tao Wang, Sameep Tandon (Stanford University)*
  Supervised Learning and Optimization: Softmax Regression
- *Sebastian Raschka*
  What is Softmax regression and how is it related to Logistic regression?