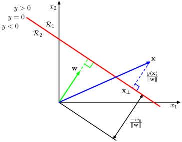
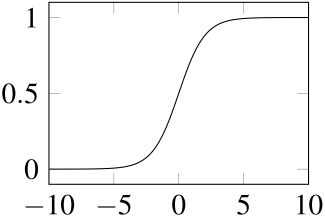
Machine Learning

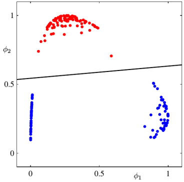
Linear Models for Classification

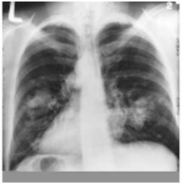


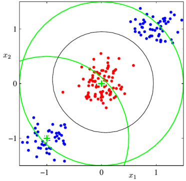
Marcello Restelli



March 15, 2017







Outline

Classification Problems

- The **goal** of classification is to assign an input $\mathbf{x}$ into one of K discrete classes C_k, where $k = 1, \ldots, K$
- Typically, each input is assigned **only to one** class

- **Example**: The input vector $\mathbf{x}$ is the set of pixel intensities, and the output variable t will represent the presence of cancer (class C_1) or absence of cancer (class C_2)

Linear Classification

- In linear classification, the input space is divided into decision regions whose boundaries are called **decision boundaries** or **decision surfaces**
- We will consider **linear models** for classification
- In the linear regression case, the model is **linear in parameters**:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{D-1} w_j x_j = \mathbf{x}^T \mathbf{w} + w_0$$

- For classification, we need to predict discrete class labels, or posterior probabilities that lie in the range of $(0, 1)$, so we use a **nonlinear function**

$$y(\mathbf{x}, \mathbf{w}) = f(\mathbf{x}^T \mathbf{w} + w_0)$$

Generalized Linear Models

$$y(\mathbf{x}, \mathbf{w}) = f(\mathbf{x}^T\mathbf{w} + w_0)$$

- The **decision surfaces** correspond to $y(\mathbf{x}, \mathbf{w}) = \text{const}$
- It follows that $\mathbf{x}^T\mathbf{w} + w_0$ and hence the **decision surfaces are linear** functions of $\mathbf{x}$, even if the activation function is nonlinear
- These class of models are called **generalized linear models**
- They are **no longer linear** in parameters
- **More complex** analytical and computational properties than regression
- As we did for regression, we can consider **fixed nonlinear basis functions**

Notation

- In two-class problems, we have a binary target value $t \in \{0, 1\}$, such that $t = 1$ is the **positive** class and $t = 0$ is the **negative** class
  - We can interpret the value of t as the **probability** of the positive class
  - The output of the model can be represented as the probability that the model assigns to the positive class
- If there are K classes, we can use a 1-of-K encoding scheme
  - $\mathbf{t}$ is a vector of length K and contains **a single 1** for the correct class and 0 elsewhere
  - **Example**: if $K = 5$, then an input that belongs to class 2 corresponds to target vector:

$$\mathbf{t} = (0, 1, 0, 0, 0)^T$$

  - We can interpret $\mathbf{t}$ as a vector of class probabilities

Three Approaches to Classification

- **Discriminant function**: build a function that **directly maps** each input to a specific class
- **Probabilistic approach**: model the conditional probability distribution $p(C_k|\mathbf{x})$ and use it to make optimal decisions. Two alternatives:
  - **Probabilistic discriminative approach**: Model $p(C_k|\mathbf{x})$ directly, for instance using parametric models (e.g., **logistic regression**)
  - **Probabilistic generative approach**: Model class conditional densities $p(\mathbf{x}|C_k)$ together with prior probabilities $p(C_k)$ for the classes. Infer the posterior using **Bayes' rule**:

$$P(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})}$$

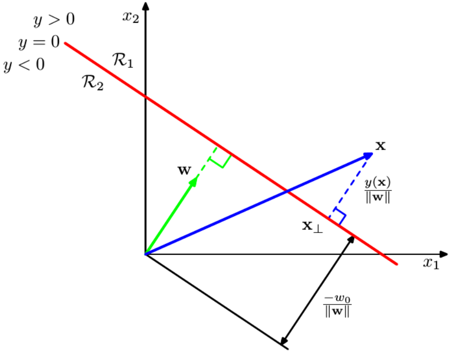
  - For **example**, we could fit multivariate Gaussians to the input vector of each class. Given a test vector, we see under which Gaussian the vector is **most probable**

Two classes

- $y(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + w_0$
- Assign $\mathbf{x}$ to C_1 is $y(\mathbf{x}) \geq 0$ and class C_2 otherwise
- Decision boundary: $y(\mathbf{x}) = 0$
- Given two points on the decision surface:

$$
\begin{aligned}
y(\mathbf{x}_A) = y(\mathbf{x}_B) &= 0 \\
\mathbf{w}^T(\mathbf{x}_A - \mathbf{x}_B) &= 0
\end{aligned}
$$

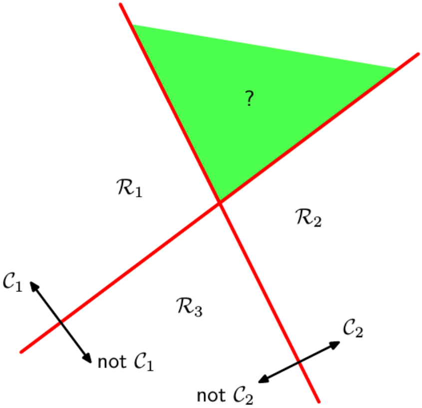
- $\mathbf{w}$ is orthogonal to the decision surface



- if $\mathbf{x}$ is on the decision surface, then $\dfrac{\mathbf{w}^T\mathbf{x}}{\|\mathbf{w}\|_2} = -\dfrac{w_0}{\|\mathbf{w}\|_2}$
- Hence w_0 determines the location of the decision surface

Multiple classes

- Consider the **extension** to $K > 2$ classes
- **One-versus-the-rest**: $K - 1$ classifiers, each of which solves a two class problem:
  - Separate points in class C_k from points not in that class
  - There are regions in input space that are **ambiguously** classified
- **One-versus-one**: $K(K-1)/2$ binary discriminant functions
  - Each function discriminates between two particular classes
  - Similar problems of **ambiguity**



Multiple classes: Simple Solution

- Use K **linear discriminant functions** of the form

$$y_k(\mathbf{x}) = \mathbf{x}^T \mathbf{w}_k + w_{k0}, \text{ where } k = 1, \dots, K$$

- Assign $\mathbf{x}$ to class C_k, if $y_k(\mathbf{x}) > y_j(\mathbf{x}) \quad \forall j \neq k$
- The resulting decision boundaries are **singly connected** and **convex**

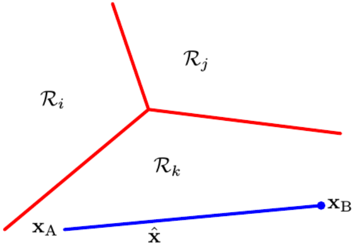
- For any two points that lie inside the region $\mathcal{R}_k$:

$$y_k(\mathbf{x}_A) > y_j(\mathbf{x}_A) \text{ and } y_k(\mathbf{x}_B) > y_j(\mathbf{x}_B)$$

implies that for positive α

$$y_k(\alpha \mathbf{x}_A + (1-\alpha)\mathbf{x}_B) > y_j(\alpha \mathbf{x}_A + (1-\alpha)\mathbf{x}_B)$$

due to linearity of the discriminant functions



Least Squares for Classification

- Consider a general classification problem with K classes using 1-of-K encoding scheme for the target vector $\mathbf{t}$
- Recall: Least squares approximates **conditional expectation** $\mathbb{E}[\mathbf{t}|\mathbf{x}]$
- Each class is described by its own linear model

$$y_k(\mathbf{x}) = \mathbf{w}_k{}^T\mathbf{x} + w_{k0}, \text{ where } k = 1, \ldots, K$$

- Using vector notation:

$$\mathbf{y}(\mathbf{x}) = \tilde{\mathbf{W}}^T\tilde{\mathbf{x}}$$

  - $\tilde{\mathbf{W}}$ is a $(D+1) \times K$ matrix whose k-th column is $\tilde{\mathbf{w}}_k = \left(w_{k0}, \mathbf{w}_k{}^T\right)^T$
  - $\tilde{\mathbf{x}} = \left(1, \mathbf{x}^T\right)^T$

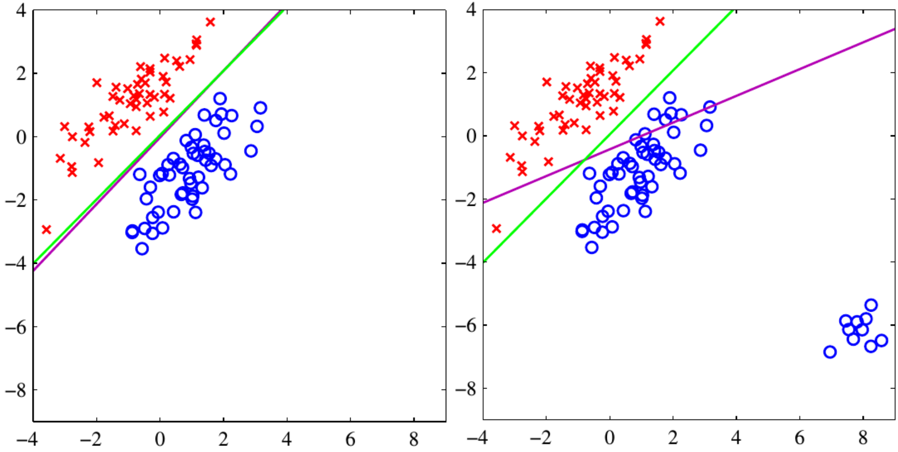
Least Squares for Classification

- Given a dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$, where $i = 1, \ldots, N$
- We have already seen how to minimize least squares

$$\tilde{\mathbf{W}} = \left(\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}\right)^{-1} \tilde{\mathbf{X}}^T \mathbf{T}$$

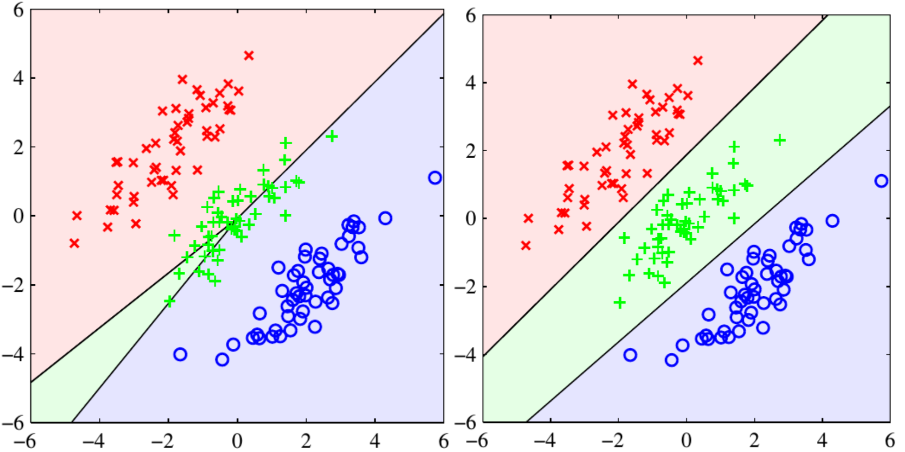
  - $\tilde{\mathbf{X}}$ is an $N \times (D + 1)$ matrix whose i-th row is $\tilde{\mathbf{x}}_i^T$
  - $\mathbf{T}$ is an $N \times K$ matrix whose i-th row is $\mathbf{t}_i{}^T$
- A new input is assigned to a class for which $t_k = \tilde{\mathbf{x}}^T \tilde{\mathbf{w}}_k$ is **largest**
- There are **problems** in using least squares for classification

Problems using Least Squares: Outliers



Least squares is highly sensitive to **outliers**, unlike logistic regression

Problems using Least Squares: Non-Gaussian Distributions

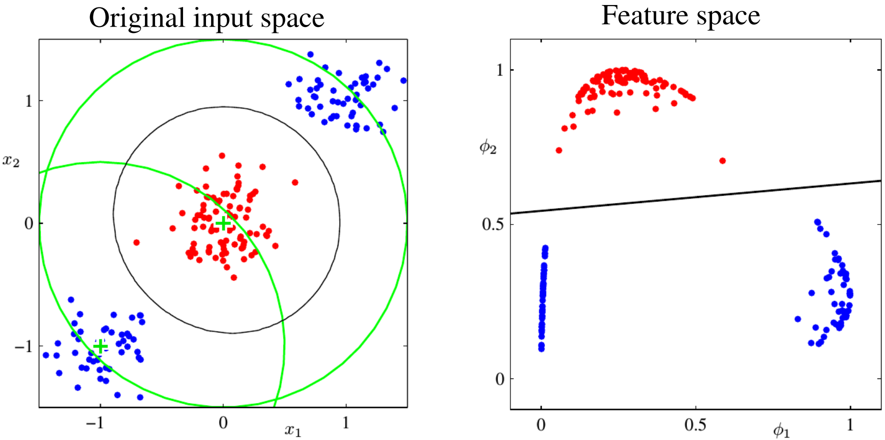


The reason for the failure is due to the assumption of a **Gaussian conditional distribution** that is not satisfied by binary target vectors

Fixed Basis Functions

- So far, we have considered classification models that work directly in the **input space**
- All considered algorithms are equally applicable if we first make a **fixed nonlinear transformation** of the input space using vector of basis functions $\phi(\mathbf{x})$
- Decision boundaries will be **linear** in the **feature space**, but would correspond to **nonlinear** boundaries in the original **input space**
- Classes that are linearly separable in the feature space **need not** be linearly separable in the original input space

Linear Basis Function Models



- We consider **two Gaussian basis functions** with centers shown by green crosses and with contours shown by green circles
- **Linear** decision boundary (right) is obtained using logistic regression and corresponds to **nonlinear** decision boundary in the input space (left)

Perceptron

- The perceptron (Rosenblatt, 1958) is another example of **linear discriminant models**
- It is an **online** linear classification algorithm
- It corresponds to a two-class model:

$$y(\mathbf{x}) = f(\mathbf{w}^T \phi(\mathbf{x})), \text{ where } f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

- Target values are +1 for $\mathcal{C}_1$ and -1 for $\mathcal{C}_2$
- The algorithm finds the **separating hyperplane** by minimizing the distance of **misclassified points** to the **decision boundary**
- Using the number of misclassified points as loss function is not effective since it is a **piecewise constant function**

Perceptron Criterion

- We are seeking a vector $\mathbf{w}$ such that $\mathbf{w}^T\phi(\mathbf{x}_n) > 0$ when $\mathbf{x}_n \in \mathcal{C}_1$ and $\mathbf{w}^T\phi(\mathbf{x}_n) < 0$ otherwise
- The **perceptron criterion** assigns
  - zero error to correct classification
  - $\mathbf{w}^T\phi(\mathbf{x}_n)t_n$ to misclassified patterns $\mathbf{x}_n$ (it is proportional to the distance to the decision boundary)
- The **loss** function to be minimized is

$$L_P(\mathbf{w}) = -\sum_{n\in\mathcal{M}} \mathbf{w}^T\phi(\mathbf{x}_n)t_n$$

- Minimization is performed using **stochastic gradient descent**:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha\nabla L_P(\mathbf{w}) = \mathbf{w}^{(k)} + \alpha\phi(\mathbf{x}_n)t_n$$

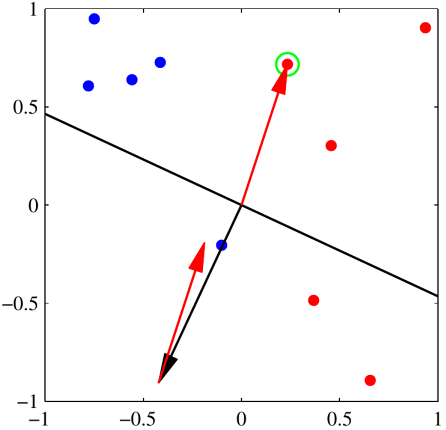
- Since the perceptron function does not change if $\mathbf{w}$ is multiplied by a constant, **the learning rate α can be set to 1**

Algorithm

Input:data set $\mathbf{x}_n \in \mathbb{R}^D$,
$t_n \in \{-1, +1\}$, for $n = 1 : N$
Initialize $\mathbf{w}_0$
$k \leftarrow 0$
repeat
    $k \leftarrow k + 1$
    $n \leftarrow k \bmod N$
    if $\hat{t}_n \neq t_n$ **then**
        $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \phi(\mathbf{x}_n) t_n$
    end if
until convergence

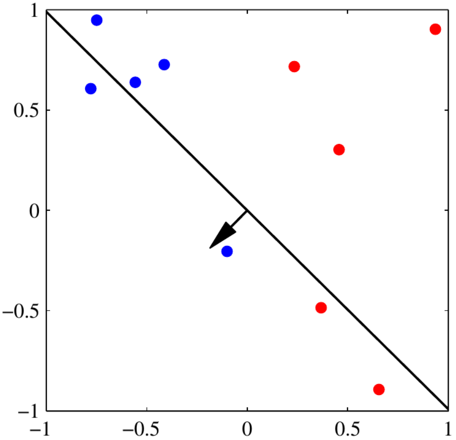
Algorithm

Input:data set $\mathbf{x}_n \in \mathbb{R}^D$,
$t_n \in \{-1, +1\}$, for $n = 1 : N$
Initialize $\mathbf{w}_0$
$k \leftarrow 0$
repeat
    $k \leftarrow k + 1$
    $n \leftarrow k \bmod N$
    if $\hat{t}_n \neq t_n$ **then**
        $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \phi(\mathbf{x}_n) t_n$
    end if
until convergence



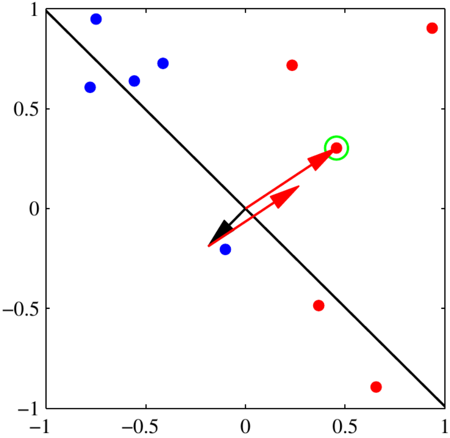
Algorithm

Input:data set $\mathbf{x}_n \in \mathbb{R}^D$,
$t_n \in \{-1, +1\}$, for $n = 1 : N$
Initialize $\mathbf{w}_0$
$k \leftarrow 0$
repeat
    $k \leftarrow k + 1$
    $n \leftarrow k \bmod N$
    if $\hat{t}_n \neq t_n$ **then**
        $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \phi(\mathbf{x}_n) t_n$
    end if
until convergence



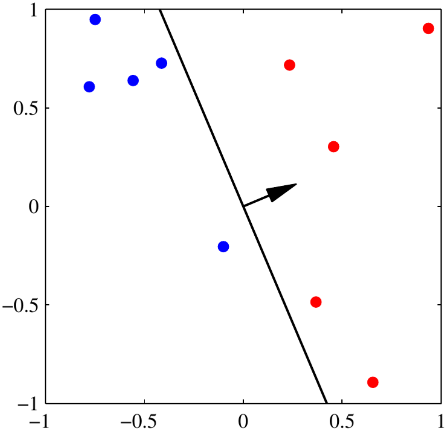
Algorithm

Input:data set $\mathbf{x}_n \in \mathbb{R}^D$,
$t_n \in \{-1, +1\}$, for $n = 1 : N$
Initialize $\mathbf{w}_0$
$k \leftarrow 0$
repeat
    $k \leftarrow k + 1$
    $n \leftarrow k \bmod N$
    if $\hat{t}_n \neq t_n$ **then**
        $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \phi(\mathbf{x}_n)t_n$
    end if
until convergence



Algorithm

Input:data set $\mathbf{x}_n \in \mathbb{R}^D$,
$t_n \in \{-1, +1\}$, for $n = 1 : N$
Initialize $\mathbf{w}_0$
$k \leftarrow 0$
repeat
    $k \leftarrow k + 1$
    $n \leftarrow k \bmod N$
    if $\hat{t}_n \neq t_n$ **then**
        $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \phi(\mathbf{x}_n) t_n$
    end if
until convergence



Preceptron Convergence Theorem

- The effect of a single update is to **reduce the error** due to the **misclassified pattern**:

$$-\mathbf{w}^{(k+1)^T}\phi(\mathbf{x}_n)t_n = -\mathbf{w}^{(k)^T}\phi(\mathbf{x}_n)t_n - (\phi(\mathbf{x}_n)t_n)^T\phi(\mathbf{x}_n)t_n < -\mathbf{w}^{(k)^T}\phi(\mathbf{x}_n)t_n$$

- This **does not imply** that the loss is reduced at each stage

Theorem (Perceptron Convergence Theorem)

If the training data set is ***linearly separable*** *in the feature space* Φ*, then the perceptron learning algorithm is guaranteed to find an* ***exact solution*** *in a* ***finite number of steps***

- The number of steps before convergence may be **substantial**
- We are not able to distinguish between **nonseparable** problems and **slowly converging** ones
- If **multiple solutions** exist, the one found depends by the **initialization** of the parameters and the **order of presentation** of the data points

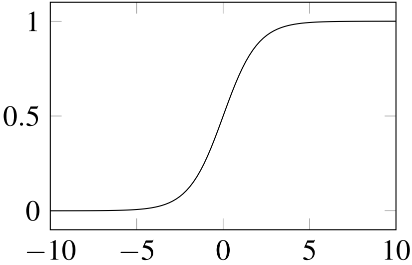
Logistic Regression

- Consider the problem of **two-class classification**
- The posterior probability of class C_1 can be written as a **logistic sigmoid function**:

$$p(C_1|\phi) = \frac{1}{1 + \exp(-\mathbf{w}^T\phi)} = \sigma(\mathbf{w}^T\phi)$$

where $p(C_2|\phi) = 1 - p(C_1|\phi)$ and the bias term is omitted for clarity

- This model is known as **logistic regression** (even if it is for classification!)
- Differently from generative models, here we model $p(C_k|\phi)$ directly



Maximum Likelihood for Logistic Regression

- Given a dataset $\mathcal{D} = \{\mathbf{x}_n, t_n\}, n = 1, \dots, N; t_n \in \{0, 1\}$
- **Maximize the probability** of getting the right label:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{1-t_n}, \quad y_n = \sigma(\mathbf{w}^T \phi_n)$$

- Taking the negative log of the likelihood, we can define **cross-entropy error function** (to be minimized):

$$L(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = -\sum_{n=1}^{N} (t_n \ln y_n + (1 - t_n) \ln(1 - y_n)) = \sum_{n=1}^{N} L_n$$

- Differentiating and using the chain rule:

$$\frac{\partial L_n}{\partial y_n} = \frac{y_n - t_n}{y_n(1 - y_n)}, \quad \frac{\partial y_n}{\partial \mathbf{w}} = y_n(1-y_n)\phi_i, \quad \frac{\partial L_n}{\partial \mathbf{w}} = \frac{\partial L_n}{\partial y_n}\frac{\partial y_n}{\partial \mathbf{w}} = (y_n - t_n)\phi_n$$

Maximum Likelihood for Logistic Regression

- The gradient of the loss function is

$$\nabla L(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\phi_n$$

- It has the same form as the gradient of the **sum-of-squares error function** for linear regression
- There is **no closed form solution**, due to nonlinearity of the logistic sigmoid function
- The error function is **convex** and can be optimized by standard **gradient-based optimization** techniques
- Easy to adapt to the **online learning setting**

Multiclass Logistic Regression

- For the multiclass case, we represent posterior probabilities by a **softmax** transformation of linear functions of feature variables:

$$p(C_k|\phi) = y_k(\phi) = \frac{\exp(\mathbf{w}_k{}^T\phi)}{\sum_j \exp(\mathbf{w}_j{}^T\phi)}$$

- Differently from generative models, here we will use **maximum likelihood** to determine parameters of this discriminative model directly

$$p(\mathbf{T}|\mathbf{\Phi}, \mathbf{w}_1, \dots, \mathbf{w}_K) = \prod_{n=1}^{N} \underbrace{\left(\prod_{k=1}^{K} p(C_k|\phi_n)^{t_{nk}}\right)}_{\substack{\text{Only one term corresponding}\\ \text{to correct class}}} = \prod_{n=1}^{N}\left(\prod_{k=1}^{K} y_{nk}^{t_{nk}}\right)$$

where $y_{nk} = p(C_k|\phi_n) = \dfrac{\exp(\mathbf{w}_k{}^T\phi_n)}{\sum_j \exp(\mathbf{w}_j{}^T\phi_n)}$

Multiclass Logistic Regression

- Taking the negative logarithm gives the **cross-entropy function** for multi-class classification problem

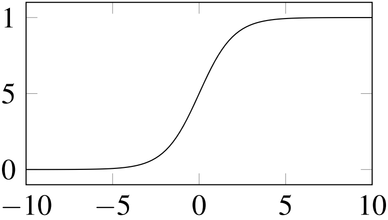
$$L(\mathbf{w}_1, \dots, \mathbf{w}_K) = -\ln p(\mathbf{T}|\mathbf{\Phi}, \mathbf{w}_1, \dots, \mathbf{w}_K) = -\sum_{n=1}^{N} \left(\sum_{k=1}^{K} t_{nk} \ln y_{nk} \right)$$

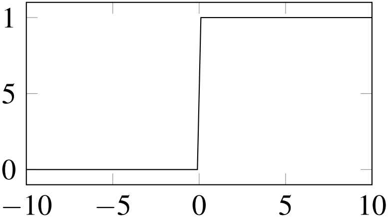
- Taking the gradient

$$\nabla L_{\mathbf{w}_j}(\mathbf{w}_1, \dots, \mathbf{w}_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj}) \phi_n$$

Connection Between Logistic Regression and Perceptron Algorithm

If we replace the **logistic function** with a **step function**





$$y(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}\cdot\phi(\mathbf{x})}}$$

$$y(\mathbf{x}, \mathbf{w}) = \begin{cases} 1 & \text{if } \mathbf{w}\cdot\phi(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Both algorithms use **the same updating rule**:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(y(\mathbf{x}_n, \mathbf{w}) - t_n\right)\phi_n$$