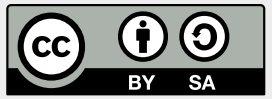
Perceptron and Logistic Regression

Milan Straka

October 21, 2019

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics



Binary Classification

Binary classification is a classification in two classes.

To extend linear regression to binary classification, we might seek a *threshold* and the classify an input as negative/positive depending whether $y(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{w} + b$ is smaller/larger than a given threshold.

Zero value is usually used as the threshold, both because it is symmetric and also because the *bias* parameter acts as a trainable threshold anyway.

Binary Classification

- Consider two points on the decision boundary. Because $y(\boldsymbol{x}_1) = y(\boldsymbol{x}_2)$, we have $(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \boldsymbol{w} = 0$, and so $\boldsymbol{w}$ is orthogonal to every vector on the decision surface – $\boldsymbol{w}$ is a *normal* of the boundary.
- Consider $\boldsymbol{x}$ and let $\boldsymbol{x}_\perp$ be orthogonal projection of x to the bounary, so we can write $\boldsymbol{x} = \boldsymbol{x}_\perp + r\frac{\boldsymbol{w}}{||\boldsymbol{w}||}$. Multiplying both sides by $\boldsymbol{w}^T$ and adding b, we get that the distance of $\boldsymbol{x}$ to the boundary is $r = \frac{y(\boldsymbol{x})}{||\boldsymbol{w}||}$.
- The distance of the decision boundary from origin is therefore $\frac{|b|}{||\boldsymbol{w}||}$.

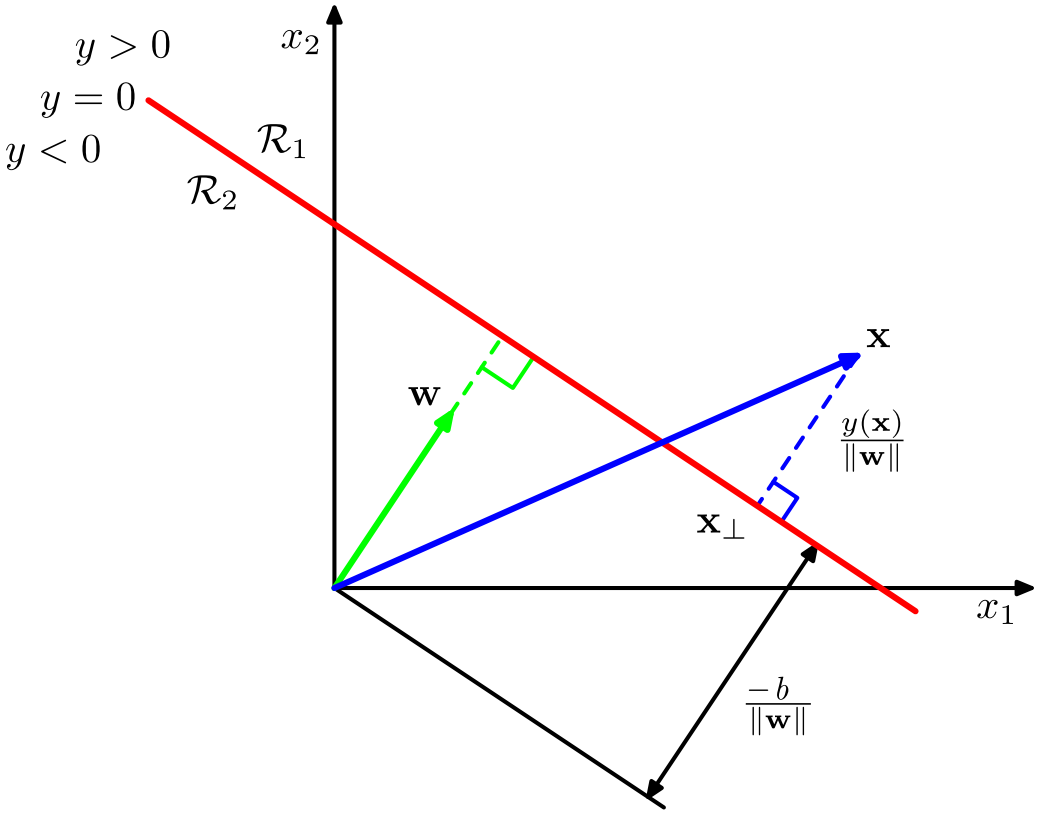


Figure 4.1 of Pattern Recognition and Machine Learning.

Perceptron

The perceptron algorithm is probably the oldest one for training weights of a binary classification. Assuming the target value $t \in \{-1, +1\}$, the goal is to find weights $\boldsymbol{w}$ such that for all train data

$$\operatorname{sign}(\boldsymbol{w}^T \boldsymbol{x}_i) = t_i,$$

or equivalently

$$t_i \boldsymbol{w}^T \boldsymbol{x}_i > 0.$$

Note that a set is called **linearly separable**, if there exist a weight vector $\boldsymbol{w}$ such that the above equation holds.

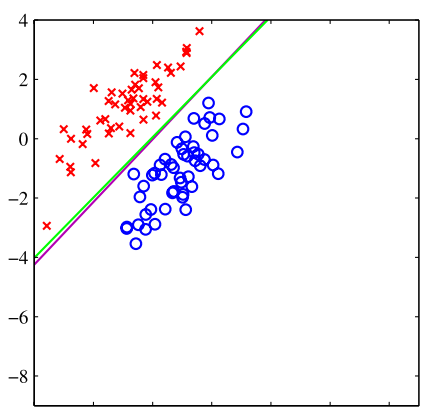


Figure 4.4 of Pattern Recognition and Machine Learning.

Perceptron

The perceptron algorithm was invented by Rosenblat in 1958.

Input: Linearly separable dataset ($\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \{-1, +1\}$).
Output: Weights $\boldsymbol{w} \in \mathbb{R}^D$ such that $t_i \boldsymbol{x}_i^T \boldsymbol{w} > 0$ for all i.

- $\boldsymbol{w} \leftarrow 0$
- until all examples are classified correctly, process example i:
  - $y \leftarrow \boldsymbol{w}^T \boldsymbol{x}_i$
  - if $t_i y \leq 0$ (incorrectly classified example):
    - $\boldsymbol{w} \leftarrow \boldsymbol{w} + t_i \boldsymbol{x}_i$

We will prove that the algorithm always arrives at some correct set of weights $\boldsymbol{w}$ if the training set is linearly separable.

Perceptron as SGD

Consider the main part of the perceptron algorithm:

- $y \leftarrow \boldsymbol{w}^T \boldsymbol{x}_i$
- if $t_i y \leq 0$ (incorrectly classified example):
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + t_i \boldsymbol{x}_i$

We can derive the algorithm using on-line gradient descent, using the following loss function

$$L(f(\boldsymbol{x}; \boldsymbol{w}), t) \stackrel{\text{def}}{=} \begin{cases} -t\boldsymbol{x}^T \boldsymbol{w} & \text{if } t\boldsymbol{x}^T \boldsymbol{w} \leq 0 \\ 0 & \text{otherwise} \end{cases} = \max(0, -t\boldsymbol{x}^T \boldsymbol{w}) = \mathrm{ReLU}(-t\boldsymbol{x}^T \boldsymbol{w}).$$

In this specific case, the value of the learning rate does not actually matter, because multiplying $\boldsymbol{w}$ by a constant does not change a prediction.

Perceptron Example

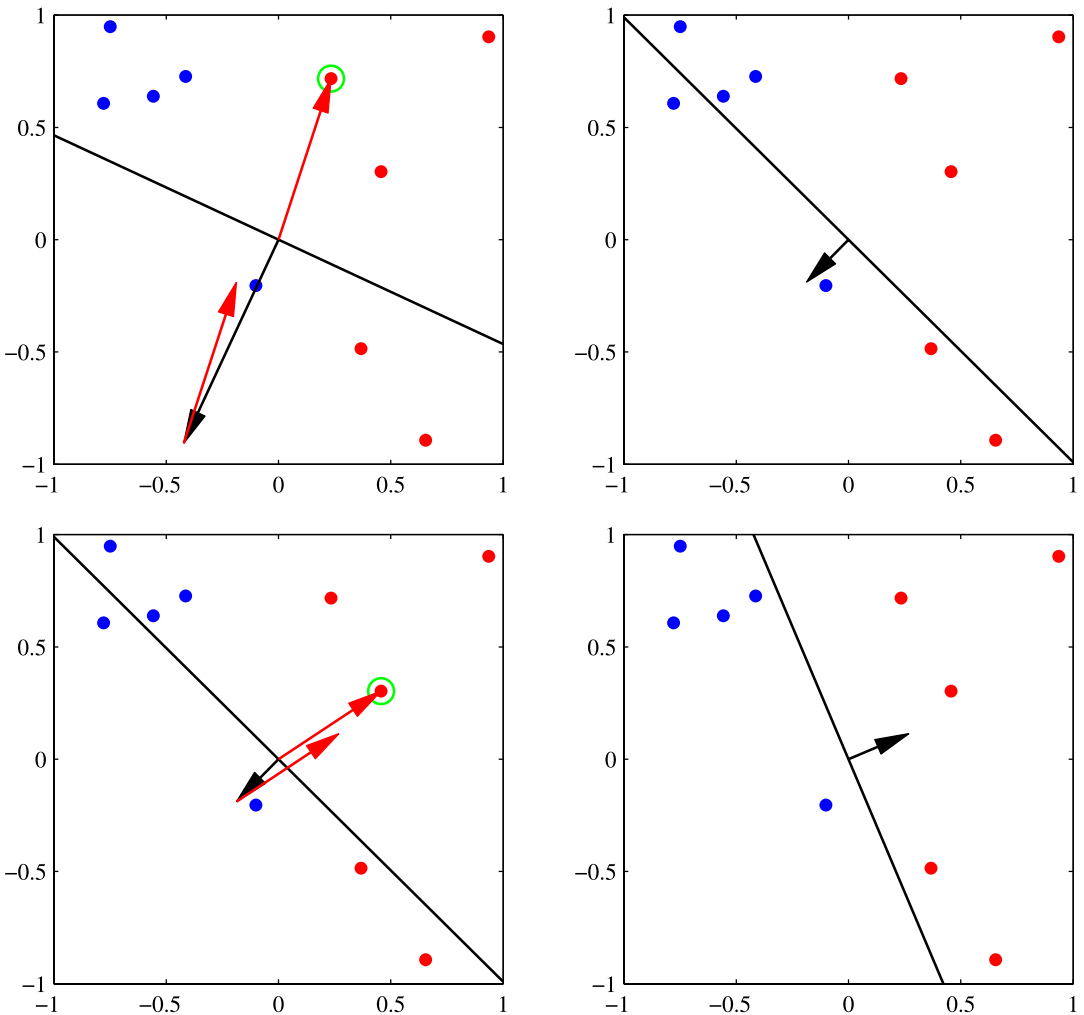
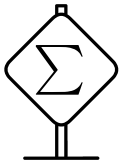


Figure 4.7 of Pattern Recognition and Machine Learning.

Proof of Perceptron Convergence

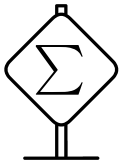


Let $\boldsymbol{w}_*$ be some weights separating the training data and let $\boldsymbol{w}_k$ be the weights after k non-trivial updates of the perceptron algorithm, with $\boldsymbol{w}_0$ being 0.

We will prove that the angle α between $\boldsymbol{w}_*$ and $\boldsymbol{w}_k$ decreases at each step. Note that

$$\cos(\alpha) = \frac{\boldsymbol{w}_*^T \boldsymbol{w}_k}{||\boldsymbol{w}_*|| \cdot ||\boldsymbol{w}_k||}.$$

Proof of Perceptron Convergence



Assume that the maximum norm of any training example $||\boldsymbol{x}||$ is bounded by R, and that γ is the minimum margin of $\boldsymbol{w}_*$, so $t\boldsymbol{w}_*^T\boldsymbol{x} \geq \gamma$.

First consider the dot product of $\boldsymbol{w}_*$ and $\boldsymbol{w}_k$:

$$\boldsymbol{w}_*^T\boldsymbol{w}_k = \boldsymbol{w}_*^T(\boldsymbol{w}_{k-1} + t_k\boldsymbol{x}_k) \geq \boldsymbol{w}_*^T\boldsymbol{w}_{k-1} + \gamma.$$

By iteratively applying this equation, we get

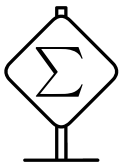
$$\boldsymbol{w}_*^T\boldsymbol{w}_k \geq k\gamma.$$

Now consider the length of $\boldsymbol{w}_k$:

$$||\boldsymbol{w}_k||^2 = ||\boldsymbol{w}_{k-1} + t_k\boldsymbol{x}_k||^2 = ||\boldsymbol{w}_{k-1}||^2 + 2t\boldsymbol{w}_{k-1}^T\boldsymbol{x}_k + ||\boldsymbol{x}_k||^2$$

Because $\boldsymbol{x}_k$ was misclassified, we know that $t\boldsymbol{w}_{k-1}^T\boldsymbol{x}_k < 0$, so $||\boldsymbol{w}_k||^2 \leq ||\boldsymbol{w}_{k-1}||^2 + R^2$.

Proof of Perceptron Convergence



Putting everything together, we get

$$\cos(\alpha) = \frac{\boldsymbol{w}_*^T \boldsymbol{w}_k}{||\boldsymbol{w}_*|| \cdot ||\boldsymbol{w}_k||} \geq \frac{k\gamma}{\sqrt{kR^2}||\boldsymbol{w}_*||}.$$

Therefore, the $\cos(\alpha)$ increases during every update. Because the value of $\cos(\alpha)$ is at most one, we can compute the upper bound on the number of steps when the algorithm converges as

$$1 \leq \frac{k\gamma}{\sqrt{kR^2}||\boldsymbol{w}_*||} \text{ or } k \geq \frac{R^2||\boldsymbol{w}_*||^2}{\gamma^2}.$$

Perceptron Issues

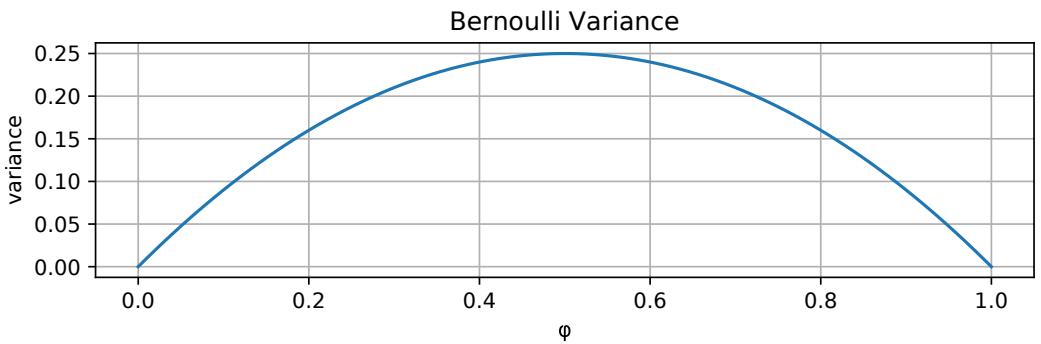
Perceptron has several drawbacks:

- If the input set is not linearly separable, the algorithm never finishes.
- The algorithm cannot be easily extended to classification into more than two classes.
- The algorithm performs only prediction, it is not able to return the probabilities of predictions.

Bernoulli Distribution

The Bernoulli distribution is a distribution over a binary random variable. It has a single parameter $\varphi \in [0, 1]$, which specifies the probability of the random variable being equal to 1.

$$P(x) = \varphi^x (1 - \varphi)^{1-x}$$
$$\mathbb{E}[x] = \varphi$$
$$\mathrm{Var}(x) = \varphi(1 - \varphi)$$



Categorical Distribution

Extension of the Bernoulli distribution to random variables taking one of k different discrete outcomes. It is parametrized by $\boldsymbol{p} \in [0, 1]^k$ such that $\sum_{i=1}^{k} p_i = 1$.

$$P(\boldsymbol{x}) = \prod_i^k p_i^{x_i}$$
$$\mathbb{E}[x_i] = p_i, \mathrm{Var}(x_i) = p_i(1 - p_i)$$

Self Information

Amount of *surprise* when a random variable is sampled.

- Should be zero for events with probability 1.
- Less likely events are more surprising.
- Independent events should have *additive* information.

$$I(x) \stackrel{\text{def}}{=} -\log P(x) = \log \frac{1}{P(x)}$$

Entropy

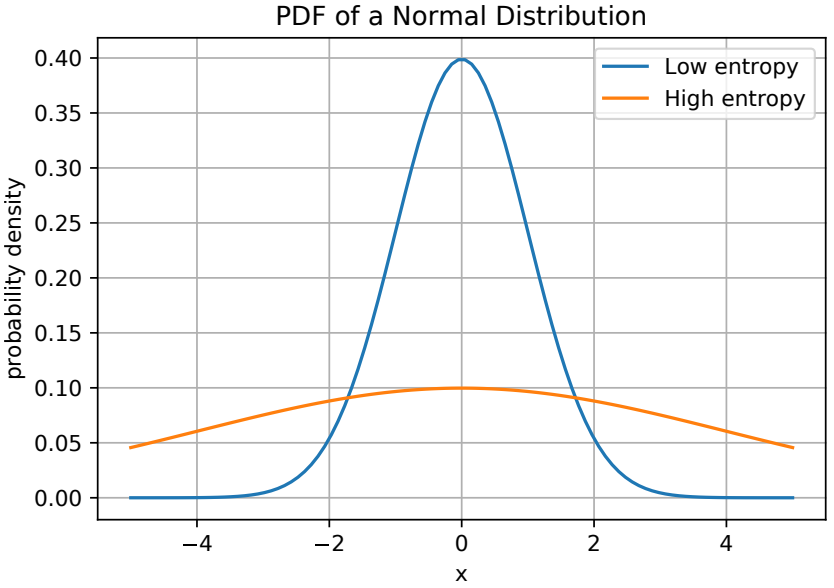
Amount of *surprise* in the whole distribution.

$$H(P) \stackrel{\text{def}}{=} \mathbb{E}_{\mathbf{x}\sim P}[I(x)] = -\mathbb{E}_{\mathbf{x}\sim P}[\log P(x)]$$

- for discrete P: $H(P) = -\sum_x P(x) \log P(x)$
- for continuous P: $H(P) = -\int P(x) \log P(x)\, \mathrm{d}x$

Note that in the continuous case, the continuous entropy (also called *differential entropy*) as a bit different semantics, for example, it can be negative.

From now on, all logarithms are *natural logarithms* with base e.



Cross-Entropy

$$H(P,Q) \stackrel{\text{def}}{=} -\mathbb{E}_{\mathbf{x}\sim P}[\log Q(x)]$$

- Gibbs inequality
  - $H(P,Q) \geq H(P)$
  - $H(P) = H(P,Q) \Leftrightarrow P = Q$
  - Proof: We use that $\log x \leq (x-1)$, with equality only for $x = 1$.

$$\sum_x P(x) \log \frac{Q(x)}{P(x)} \leq \sum_x P(x) \left(\frac{Q(x)}{P(x)} - 1 \right) = \sum_x Q(x) - \sum_x P(x) = 0.$$

  - Alternative proof: Using Jensen's inequality, we get

$$\sum_x P(x) \log \frac{Q(x)}{P(x)} \leq \log \sum_x P(x) \frac{Q(x)}{P(x)} = \log \sum_x Q(x) = 0.$$

Corollary of the Gibbs inequality

For a categorical distribution with n outcomes, $H(P) \leq \log n$, because for $Q(x) = 1/n$ we get $H(P) \leq H(P, Q) = -\sum_x P(x) \log Q(x) = \log n$.

Nonsymmetry

Note that generally $H(P, Q) \neq H(Q, P)$.

Kullback-Leibler Divergence (KL Divergence)

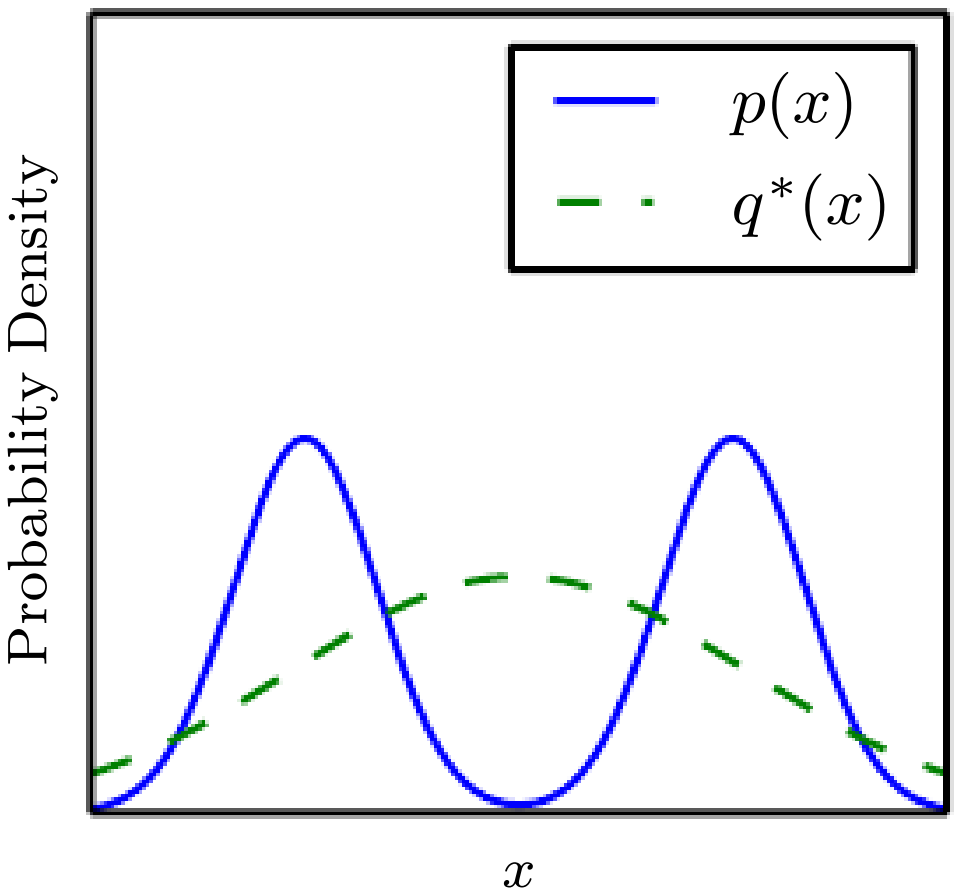
Sometimes also called *relative entropy*.

$$D_{\mathrm{KL}}(P||Q) \stackrel{\mathrm{def}}{=} H(P,Q) - H(P) = \mathbb{E}_{\mathbf{x}\sim P}[\log P(x) - \log Q(x)]$$

- consequence of Gibbs inequality: $D_{\mathrm{KL}}(P||Q) \geq 0$
- generally $D_{\mathrm{KL}}(P||Q) \neq D_{\mathrm{KL}}(Q||P)$

Nonsymmetry of KL Divergence

$$q^* = \text{argmin}_q D_{\text{KL}}(p\|q)$$



$$q^* = \text{argmin}_q D_{\text{KL}}(q\|p)$$

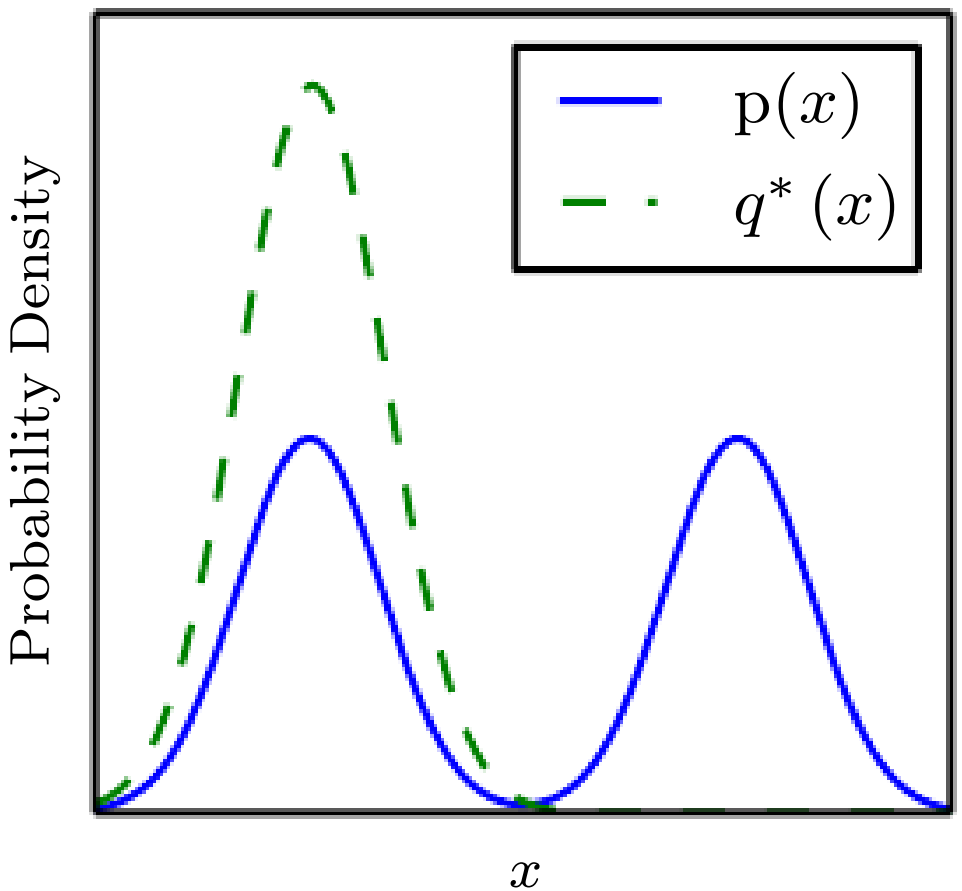


Figure 3.6, page 76 of Deep Learning Book, http://deeplearningbook.org

Normal (or Gaussian) Distribution

Distribution over real numbers, parametrized by a mean μ and variance σ^2:

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

For standard values $\mu = 0$ and $\sigma^2 = 1$ we get $\mathcal{N}(x; 0, 1) = \sqrt{\frac{1}{2\pi}} e^{-\frac{x^2}{2}}$.

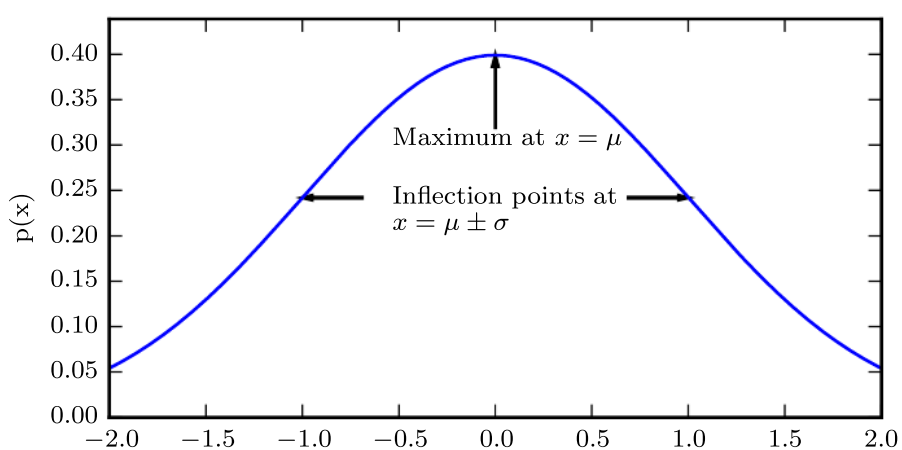


Figure 3.1, page 64 of Deep Learning Book, http://deeplearningbook.org.

Central Limit Theorem

The sum of independent identically distributed random variables with finite variance converges to normal distribution.

Principle of Maximum Entropy

Given a set of constraints, a distribution with maximal entropy fulfilling the constraints can be considered the most general one, containing as little additional assumptions as possible.

Considering distributions with a given mean and variance, it can be proven (using variational inference) that such a distribution with *maximal entropy* is exactly the normal distribution.

Maximum Likelihood Estimation

Let $\mathbb{X} = \{(\boldsymbol{x}_1, t_1), (\boldsymbol{x}_2, t_2), \ldots, (\boldsymbol{x}_N, t_N)\}$ be training data drawn independently from the data-generating distribution p_{data}. We denote the empirical data distribution as $\hat{p}_{\text{data}}$.

Let $p_{\text{model}}(t|\boldsymbol{x}; \boldsymbol{w})$ be a family of distributions.

The *maximum likelihood estimation* of $\boldsymbol{w}$ is:

$$
\begin{aligned}
\boldsymbol{w}_{\text{ML}} &= \arg\max_{\boldsymbol{w}} p_{\text{model}}(\mathbb{X}; \boldsymbol{w}) \\
&= \arg\max_{\boldsymbol{w}} \prod_{i=1}^{N} p_{\text{model}}(t_i|\boldsymbol{x}_i; \boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} -\log p_{\text{model}}(t_i|\boldsymbol{x}_i; \boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}}[-\log p_{\text{model}}(t|\boldsymbol{x}; \boldsymbol{w})] \\
&= \arg\min_{\boldsymbol{w}} H(\hat{p}_{\text{data}}, p_{\text{model}}(\boldsymbol{x}; \boldsymbol{w})) \\
&= \arg\min D_{\text{KL}}(\hat{p}_{\text{data}}||p_{\text{model}}(\boldsymbol{x}; \boldsymbol{w})) + H(\hat{p}_{\text{data}})
\end{aligned}
$$

Properties of Maximum Likelihood Estimation

Assume that the true data generating distribution p_{data} lies within the model family $p_{\text{model}}(\cdot; \boldsymbol{w})$, and assume there exists a unique $\boldsymbol{w}_{p_{\text{data}}}$ such that $p_{\text{data}} = p_{\text{model}}(\cdot; \boldsymbol{w}_{p_{\text{data}}})$.

- MLE is a *consistent* estimator. If we denote $\boldsymbol{w}_m$ to be the parameters found by MLE for a training set with m examples generated by the data generating distribution, then $\boldsymbol{w}_m$ converges in probability to $\boldsymbol{w}_{p_{\text{data}}}$.

  Formally, for any $\varepsilon > 0$, $P(||\boldsymbol{w}_m - \boldsymbol{w}_{p_{\text{data}}}|| > \varepsilon) \to 0$ as $m \to \infty$.

- MLE is in a sense most *statistic efficient*. For any consistent estimator, we might consider the average distance of $\boldsymbol{w}_m$ and $\boldsymbol{w}_{p_{\text{data}}}$, formally $\mathbb{E}_{\mathbf{x}_1,\ldots,\mathbf{x}_m \sim p_{\text{data}}}[||\boldsymbol{w}_m - \boldsymbol{w}_{p_{\text{data}}}||^2]$. It can be shown (Rao 1945, Cramér 1946) that no consistent estimator has lower mean squared error than the maximum likelihood estimator.

Therefore, for reasons of consistency and efficiency, maximum likelihood is often considered the preferred estimator for machine learning.

Logistic Regression

An extension of perceptron, which models the conditional probabilities of $p(C_0|\boldsymbol{x})$ and of $p(C_1|\boldsymbol{x})$. Logistic regression can in fact handle also more than two classes, which we will see shortly.

Logistic regression employs the following parametrization of the conditional class probabilities:

$$P(C_1|\boldsymbol{x}) = \sigma(\boldsymbol{x}^t\boldsymbol{w} + \boldsymbol{b})$$
$$P(C_0|\boldsymbol{x}) = 1 - P(C_1|\boldsymbol{x}),$$

where σ is a *sigmoid function*

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

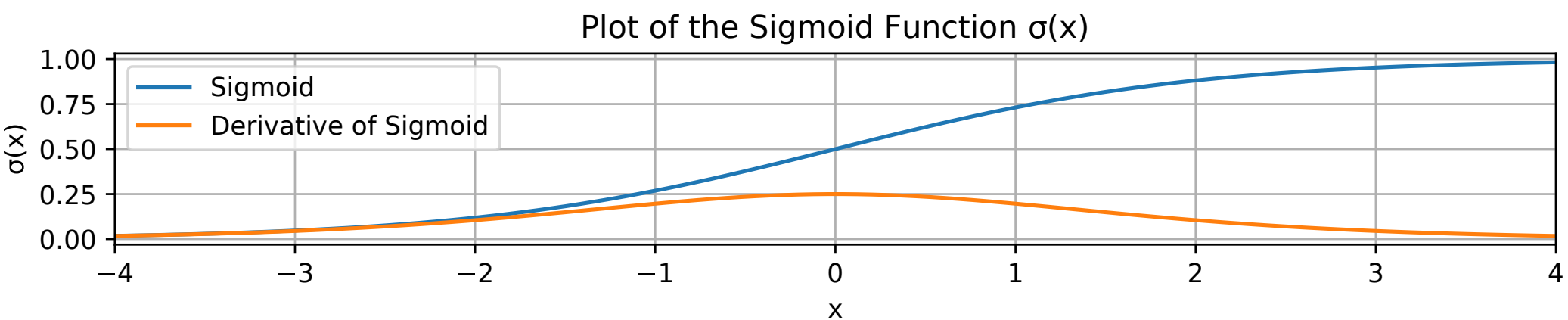
Can be trained using an SGD algorithm.

Sigmoid Function

The sigmoid function has values in range $(0, 1)$, is monotonically increasing and it has a derivative of $\frac{1}{4}$ at $x = 0$.

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

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$$



To give some meaning to the sigmoid function, starting with

$$P(C_1|\boldsymbol{x}) = \sigma(f(\boldsymbol{x};\boldsymbol{w})) = \frac{1}{1+e^{-f(\boldsymbol{x};\boldsymbol{w})}}$$

we can arrive at

$$f(\boldsymbol{x};\boldsymbol{w}) = \log\left(\frac{P(C_1|\boldsymbol{x})}{P(C_0|\boldsymbol{x})}\right),$$

where the prediction of the model $f(\boldsymbol{x};\boldsymbol{w})$ is called a *logit* and it is a logarithm of odds of the two classes probabilities.

Logistic Regression

To train the logistic regression $y(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}^T \boldsymbol{w}$, we use MLE (the maximum likelihood estimation). Note that $P(C_1|\boldsymbol{x}; \boldsymbol{w}) = \sigma(y(\boldsymbol{x}; \boldsymbol{w}))$.

Therefore, the loss for a batch $\mathbb{X} = \{(\boldsymbol{x}_1, t_1), (\boldsymbol{x}_2, t_2), \ldots, (\boldsymbol{x}_N, t_N)\}$ is

$$\mathcal{L}(\mathbb{X}) = \frac{1}{N} \sum_i -\log(P(C_{t_i}|\boldsymbol{x}_i; \boldsymbol{w})).$$

Input: Input dataset ($\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \{0, +1\}$), learning rate $\alpha \in \mathbb{R}^+$.

- $\boldsymbol{w} \leftarrow 0$
- until convergence (or until patience is over), process batch of N examples:
  - $g \leftarrow -\frac{1}{N} \sum_i \nabla_{\boldsymbol{w}} \log(P(C_{t_i}|\boldsymbol{x}_i; \boldsymbol{w})$
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \boldsymbol{g}$

Multiclass Logistic Regression

To extend the binary logistic regression to a multiclass case with K classes, we:

- Generate multiple outputs, notably K outputs, each with its own set of weights, so that

$$y(\boldsymbol{x}; \boldsymbol{W})_i = \boldsymbol{W}_i \boldsymbol{x}.$$

- Generalize the sigmoid function to a $\mathrm{softmax}$ function, such that

$$\mathrm{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}.$$

  Note that the original sigmoid function can be written as

$$\sigma(x) = \mathrm{softmax}\left([x \;\; 0]\right)_0 = \frac{e^x}{e^x + e^0} = \frac{1}{1 + e^{-x}}.$$

The resulting classifier is also known as *multinomial logistic regression*, *maximum entropy classifier* or *softmax regression*.

Multiclass Logistic Regression

Note that as defined, the multiclass logistic regression is overparametrized. It is possible to generate only $K - 1$ outputs and define $z_K = 0$, which is the approach used in binary logistic regression.

In this settings, analogously to binary logistic regression, we can recover the interpretation of the model outputs $\boldsymbol{y}(\boldsymbol{x}; \boldsymbol{W})$ (i.e., the $\mathrm{softmax}$ inputs) as *logits*:

$$y(\boldsymbol{x}; \boldsymbol{W})_i = \log\left(\frac{P(C_i|\boldsymbol{x}; \boldsymbol{w})}{P(C_K|\boldsymbol{x}; \boldsymbol{w})}\right).$$

Multiclass Logistic Regression

Using the $\mathrm{softmax}$ function, we naturally define that

$$P(C_i|\boldsymbol{x};\boldsymbol{W}) = \mathrm{softmax}(\boldsymbol{W}_i\boldsymbol{x})i = \frac{e^{\boldsymbol{W}_i\boldsymbol{x}}}{\sum_j e^{\boldsymbol{W}_j\boldsymbol{x}}}.$$

We can then use MLE and train the model using stochastic gradient descent.

Input: Input dataset ($\boldsymbol{X} \in \mathbb{R}^{N\times D}$, $\boldsymbol{t} \in \{0, 1, \ldots, K-1\}$), learning rate $\alpha \in \mathbb{R}^+$.

- $\boldsymbol{w} \leftarrow 0$
- until convergence (or until patience is over), process batch of N examples:
  - $g \leftarrow -\frac{1}{N}\sum_i \nabla_{\boldsymbol{w}} \log(P(C_{t_i}|\boldsymbol{x}_i;\boldsymbol{w})$
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha\boldsymbol{g}$

Multiclass Logistic Regression

Note that the decision regions of the multiclass logistic regression are singly connected and convex.

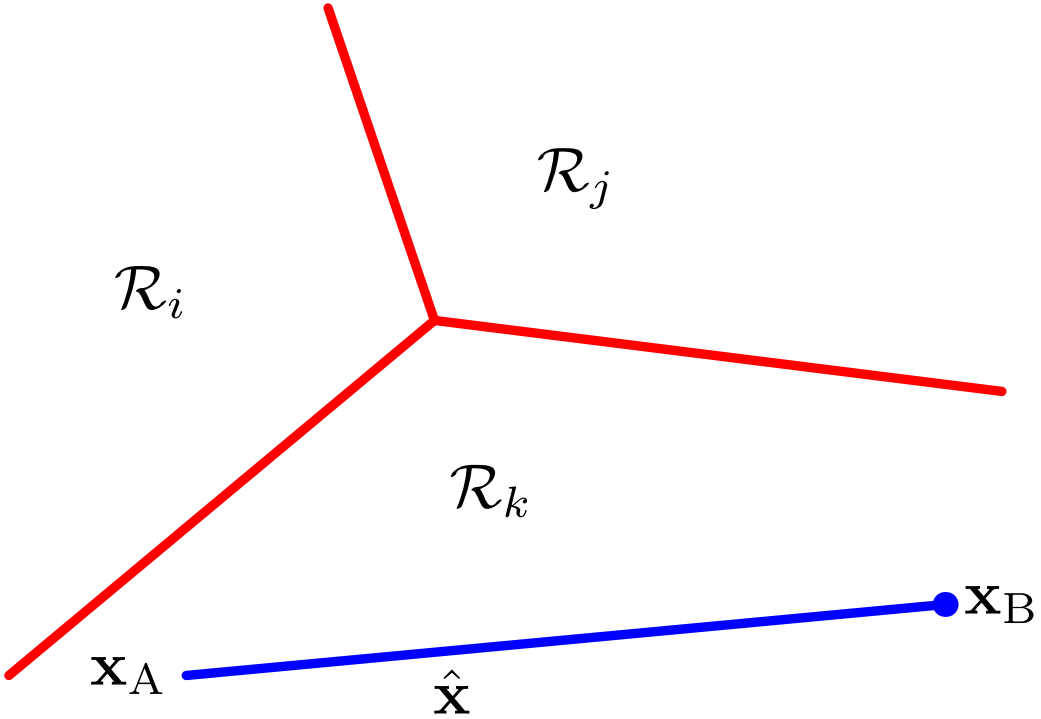


Figure 4.3 of Pattern Recognition and Machine Learning.