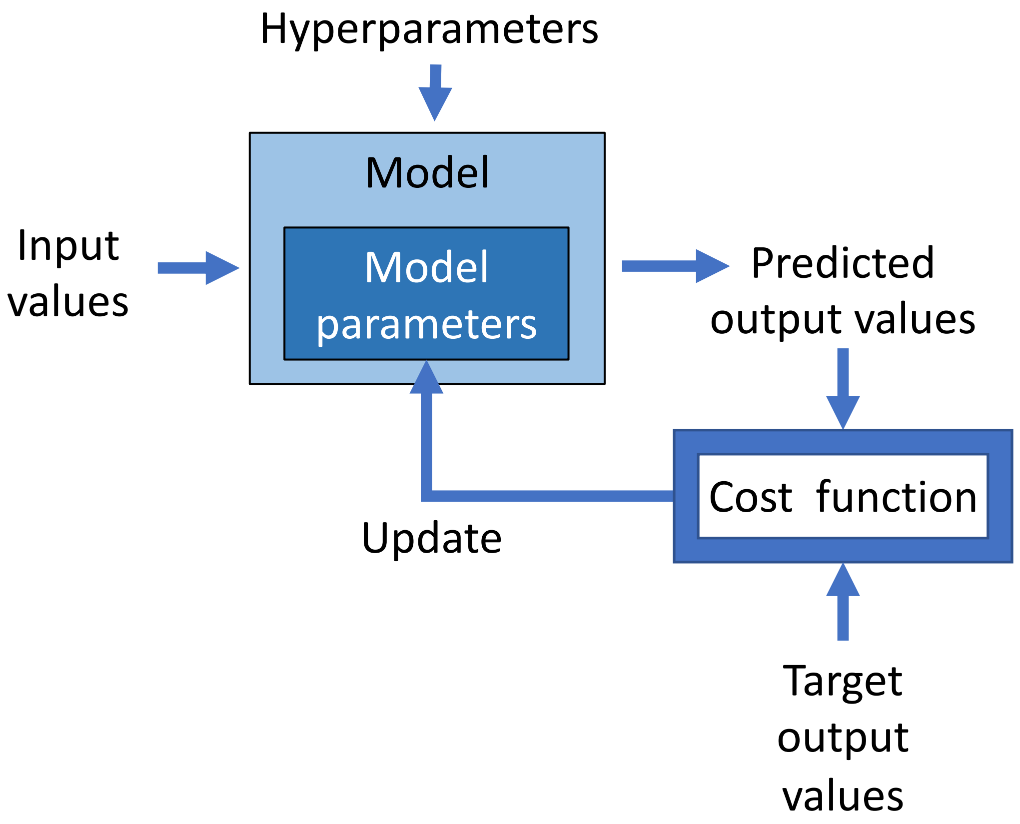
Logistic Regression and Support Vector Machines

Using Machine Learning Tools

Geron Chapter 4 and 5

Last Time …

Training: Minimise cost function by adjusting model parameters



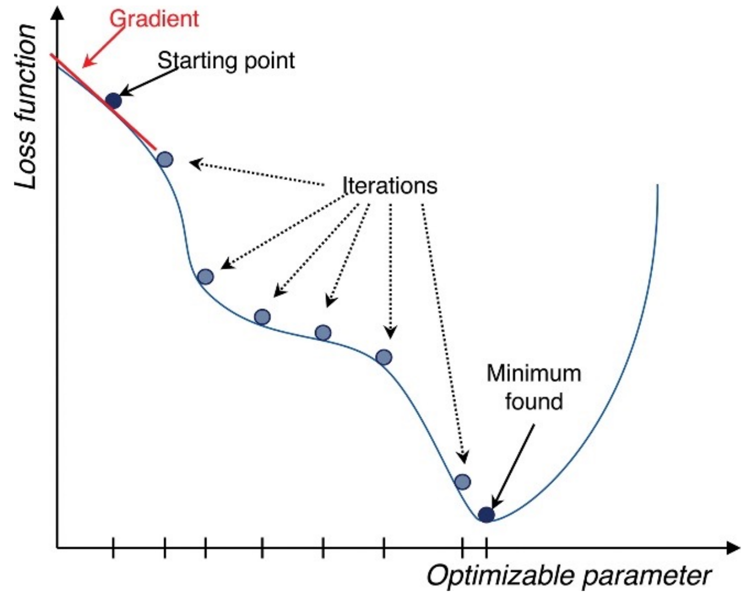


Image: Chartrand et al. RadioGraphics 2017

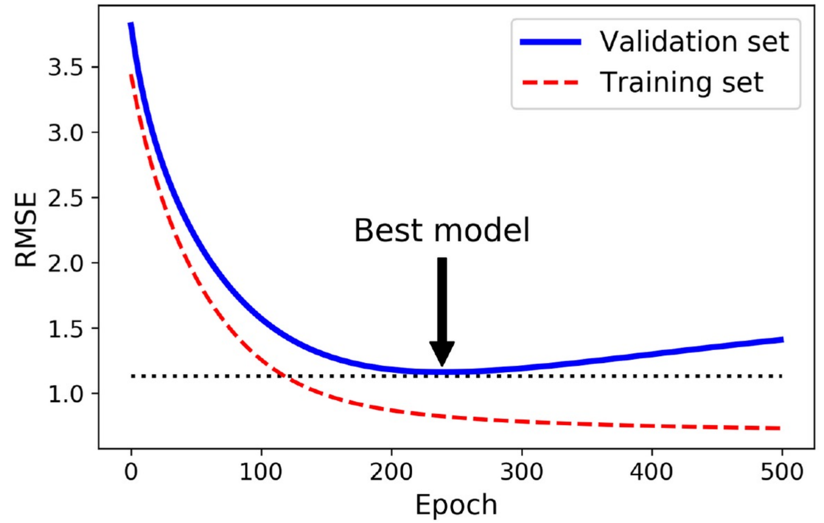


Image: Geron, Hands On ML

Logistic Regression = Logit Regression

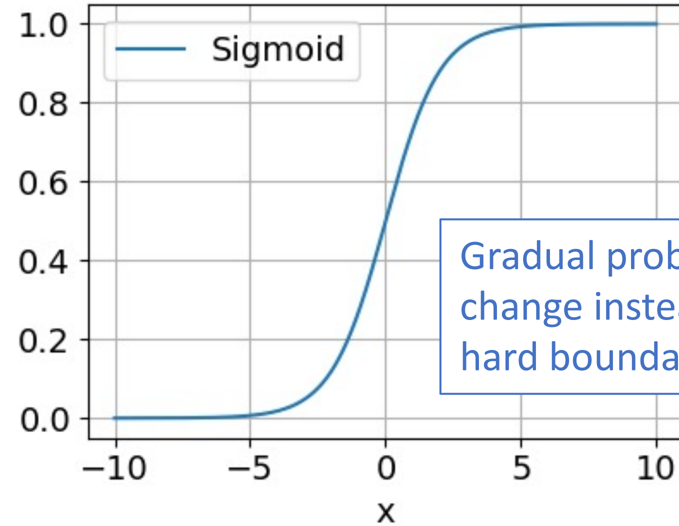
Linear model

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \boldsymbol{x}^T \cdot \boldsymbol{\theta}$$

+ Logistic function (sigmoid)

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

= Logistic model (regression)



Gradual probability change instead of hard boundary

Logistic fct. between 0 and 1
Negative inputs σ(t)<0.5
Positive inputs σ(t)>0.5

Logistic Regression = Logit Regression

Linear model

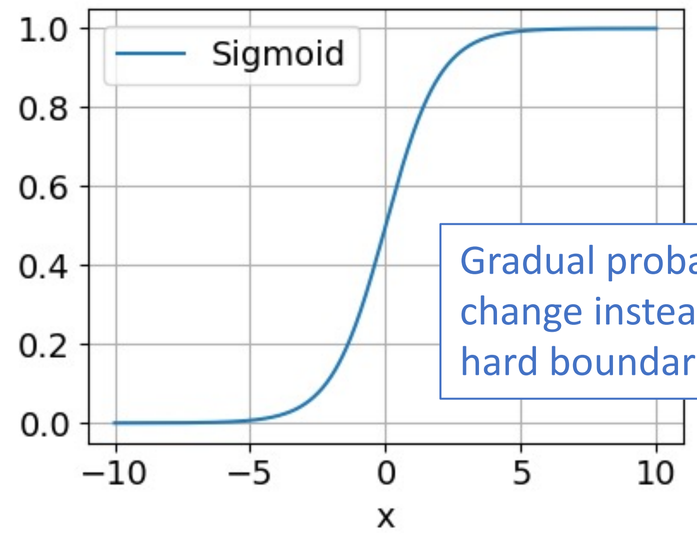
$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \boldsymbol{x}^T \cdot \boldsymbol{\theta}$$

+ Logistic function (sigmoid)

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

= Logistic model (regression)

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{x}^\top \boldsymbol{\theta})$$



Gradual probability change instead of hard boundary

Logistic fct. between 0 and 1
Negative inputs σ(t)<0.5
Positive inputs σ(t)>0.5

Logit = inverse of logistic fct.

$$logit(p) = \log\left(\frac{p}{1-p}\right)$$

Logistic Regression = Logit Regression

Linear model

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \boldsymbol{x}^T \cdot \boldsymbol{\theta}$$

+ Logistic function (sigmoid)

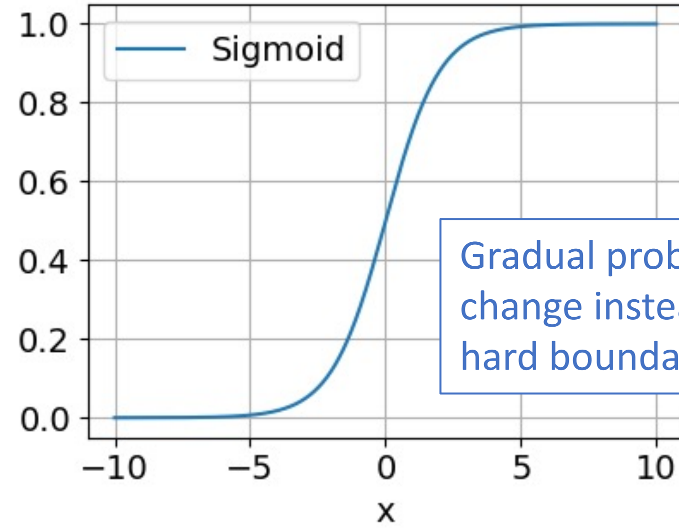
$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

= Logistic model (regression)

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{x}^\top \boldsymbol{\theta})$$

+ Threshold = binary classifier

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$$



Gradual probability change instead of hard boundary

Logistic fct. between 0 and 1
Negative inputs σ(t)<0.5
Positive inputs σ(t)>0.5

Logit = inverse of logistic fct.

$$logit(p) = \log\left(\frac{p}{1-p}\right)$$

Training a Logistic Model

- We are maximising probability of observing our targets y, given data x
- Probability of observing binary data y if the real probability is p:

$$P(y) = \begin{cases} p & \text{if} \quad y = 1 \\ 1 - p & \text{if} \quad y = 0 \end{cases}$$

or

$$P(y) = p^{\,y} \, (1 - p)^{1-y}$$

Training a Logistic Model

- We are maximising probability of observing our targets y, given data x
- Probability of observing binary data y if the real probability is p:

$$P(y) = \begin{cases} p & \text{if} \quad y = 1 \\ 1 - p & \text{if} \quad y = 0 \end{cases} \qquad \text{or} \qquad P(y) = p^{\,y} \, (1-p)^{1-y}$$

- For a set of observations: $$P(\mathbf{y}) = \prod_i \hat{p}_i^{\,y_i} \, (1 - \hat{p}_i)^{1-y_i}$$

or $$\log\left(P(\mathbf{y})\right) = \sum_i y_i \log\left(\hat{p}_i\right) + (1 - y_i) \log\left(1 - \hat{p}_i\right)$$

Training a Logistic Model

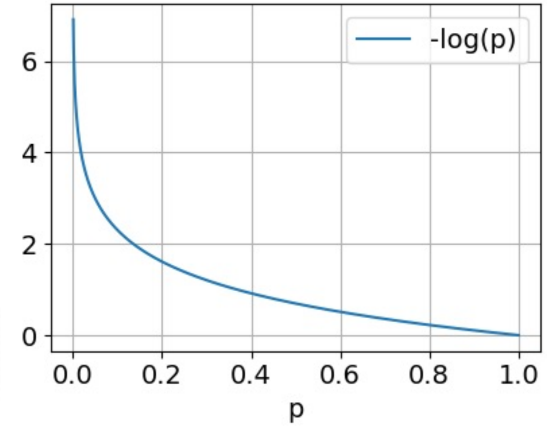
- We are maximising probability of observing our targets y, given data x
- Probability of observing binary data y if the real probability is p:

$$P(y) = \begin{cases} p & \text{if} \quad y = 1 \\ 1 - p & \text{if} \quad y = 0 \end{cases} \qquad \text{or} \qquad P(y) = p^{\,y} \, (1-p)^{1-y}$$

- For a set of observations: $P(\mathbf{y}) = \prod_i \hat{p}_i^{\,y_i} \, (1 - \hat{p}_i)^{1-y_i}$

or $\log\left(P(\mathbf{y})\right) = \sum_i y_i \log\left(\hat{p}_i\right) + (1 - y_i) \log\left(1 - \hat{p}_i\right)$

- Where the probability is the estimated probability output by the regression $\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{x}^\top \boldsymbol{\theta})$
- *Minimise* this *negative* log probability: called *log loss* or *binary cross-entropy* $J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right) log\left(1 - \hat{p}^{(i)}\right) \right]$



- Cost function is convex → good for optimisation, as they have no local minima

Decision Boundaries

```
from sklean.linear_model import LogisticRegression
log_model=LogisticRegression()
log_model.fit(X, y)
log_model.predict(X_new) # predict class
log_model.predict_proba(X_new) # predict probabilities
```

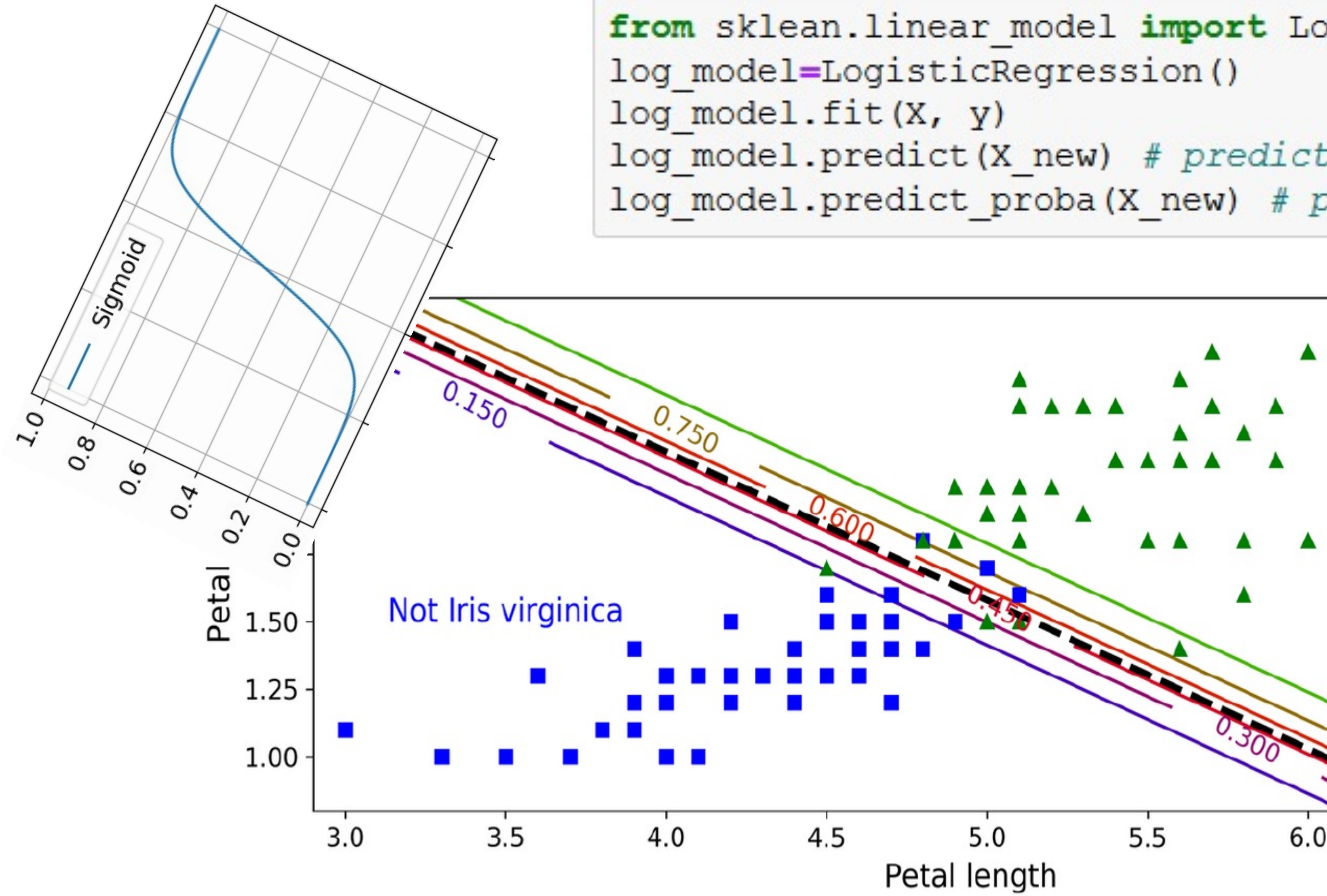


Image: Geron, Hands On ML

Softmax Regression

- Extend logistic regression to multiple classes
- Fit a linear model for each class k $s_k(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\theta}^{(k)}$

Called "logits"

- Map the class scores into a probability distribution using:
- *Softmax* function:

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

Outputs in range (0,1)

Monotonic

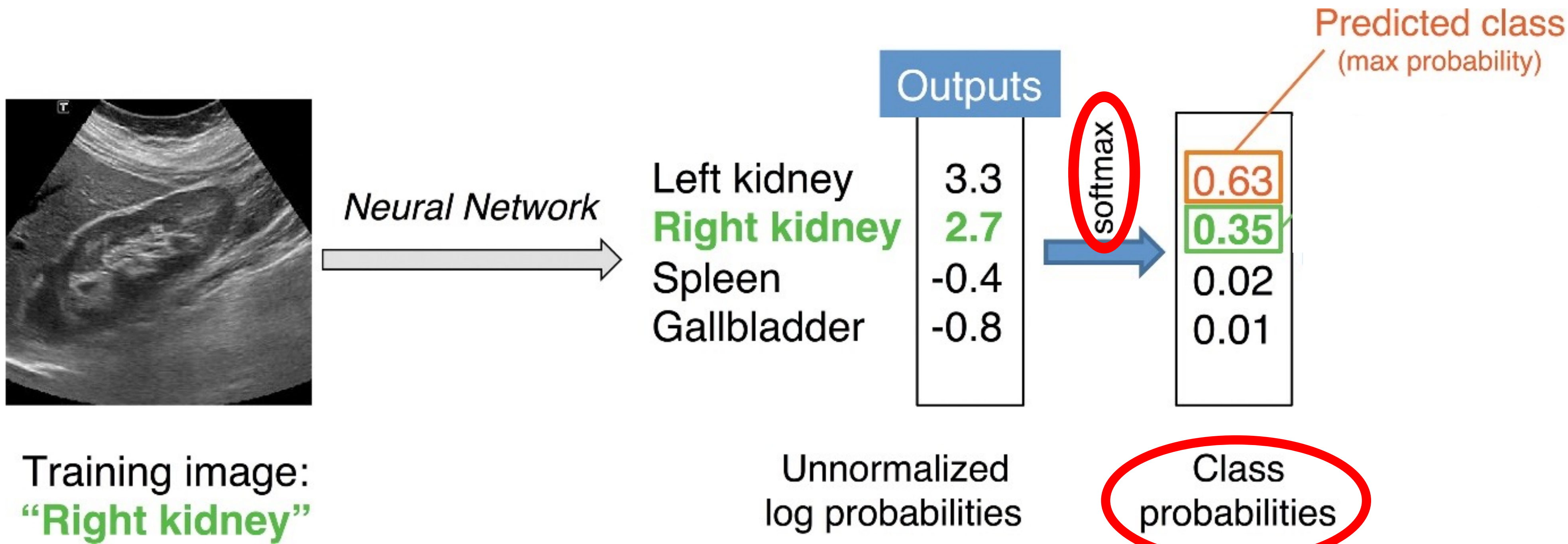
$$\sum_k \hat{p}_k = 1$$

- Cost function: *Cross entropy*

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

Differentiable, so use Gradient Descent

Softmax Normalisation in Neural Networks



$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

(Chartrand et al. RadioGraphics 2017)

Part 2

Support Vector Machines

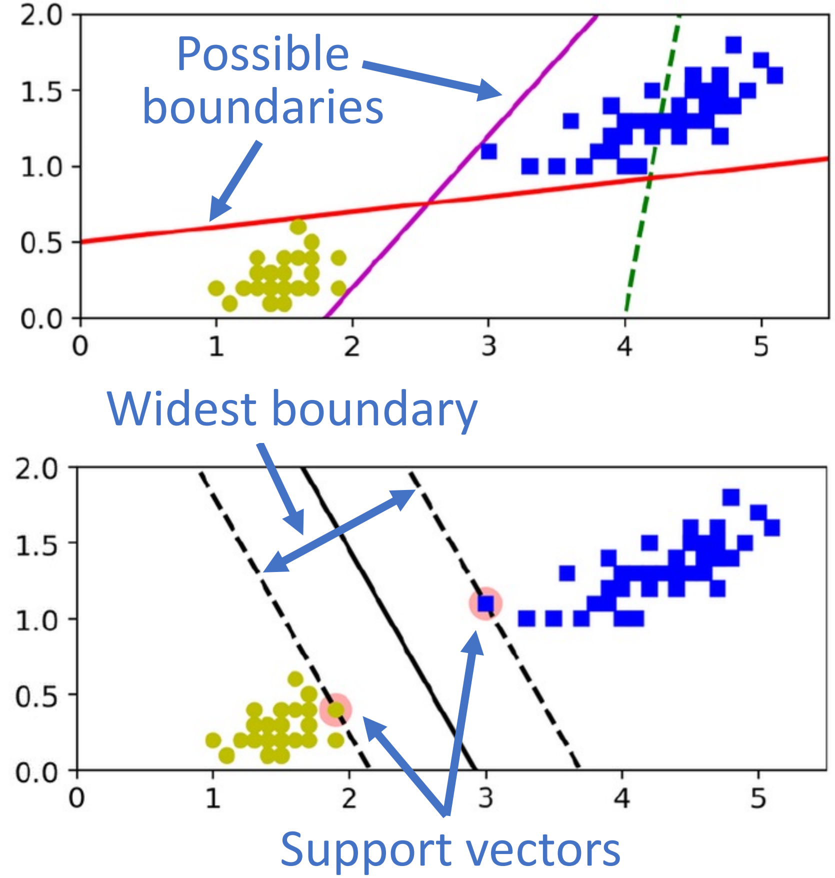
Support Vector Machine (SVM)

- Key idea: fit the widest possible boundary between two classes while limiting boundary violations
- Turns out that the boundary is defined by a small set of samples at the boundary edge → support vectors
- Good for complex boundaries & small to medium datasets (becomes quite slow for large sets)
- Can be linear or non-linear (boundary shape)
- Internally, uses a smart mathematical approach, called the "kernel trick"

Linear SVM - Hard Margin

- Fit the widest possible boundary between two classes with zero boundary violations

- As for any method based on *distances* an SVM is sensitive to the scale of inputs, especially relative scaling
  - Make sure pre-processing does scaling!

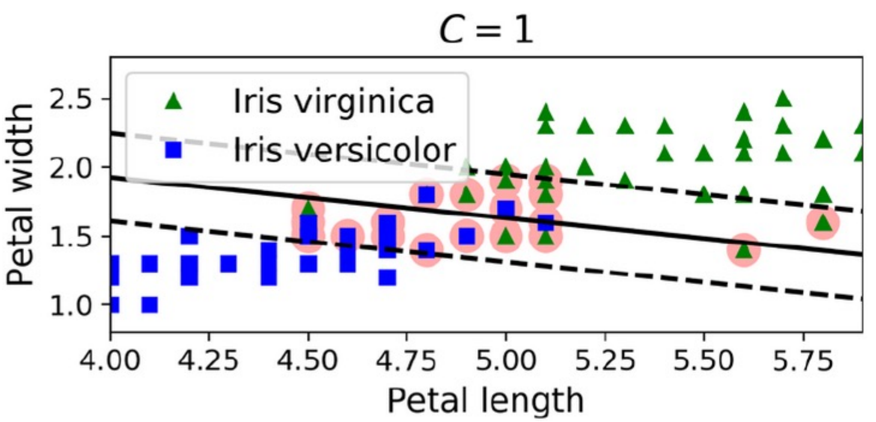


Images: Geron, Hands On ML

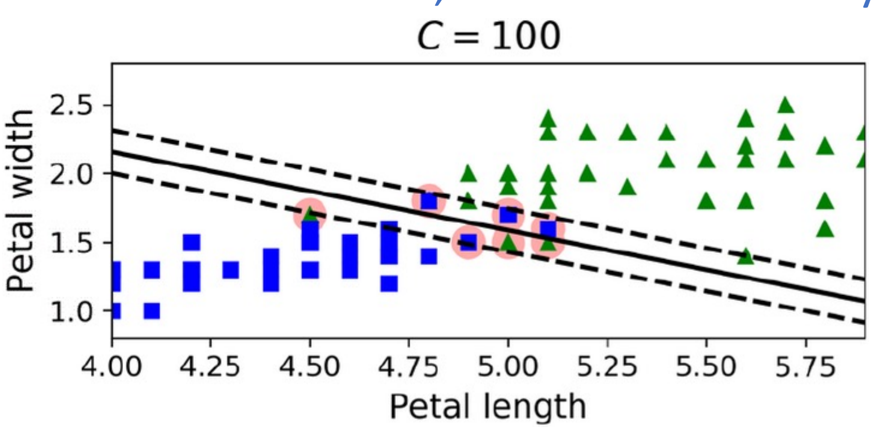
Linear SVM – Soft Margin

- Fit the *widest possible boundary* between two classes while limiting boundary violations
  - Trade-off between these
- Parameter C controls how important boundary violations are in the loss function
  - C near zero → Hard Margin where boundary width is most important
  - Large C → boundary width less important compared to boundary violations

Wider boundary, more violations



Less violations, narrower boundary

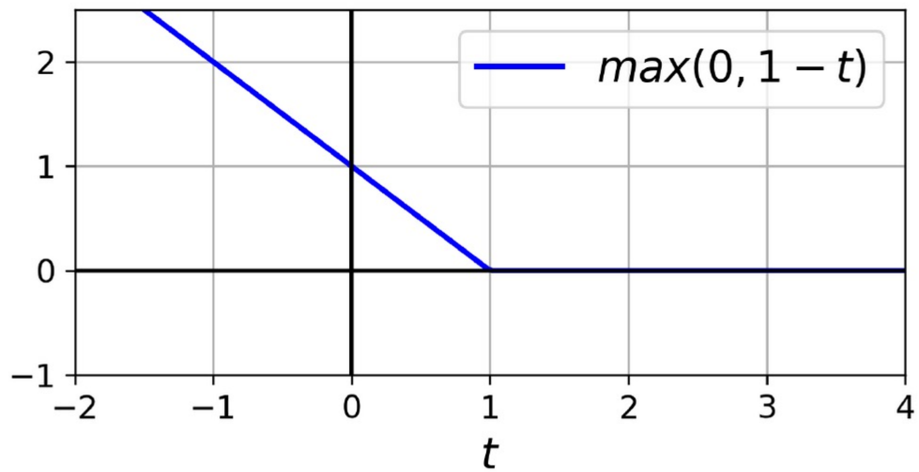


Images: Geron, Hands On ML

Hinge loss function & SVM

- Can write the SVM loss using the Hinge loss function:

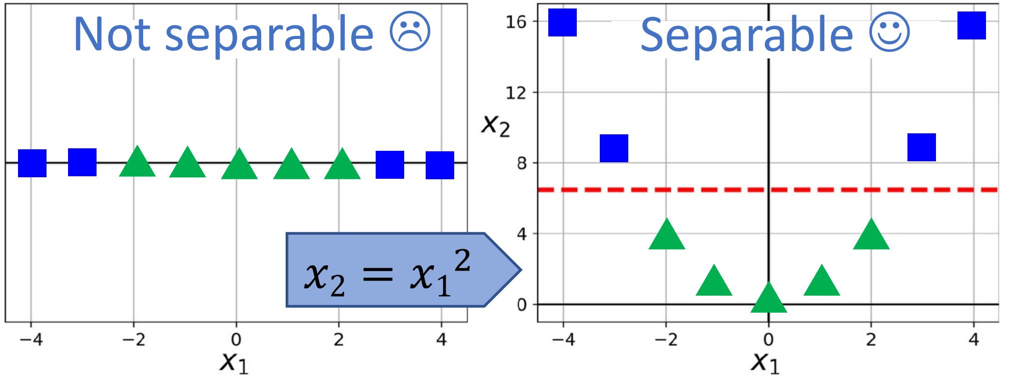
$$\max(0, 1 - t)$$



- You will see this as an option in other machine learning methods

Non-Linear Boundaries

- Problem: Many decision boundaries are not linear
- Idea: Add non-linear functions of the input data as extra features and use a linear boundary in this higher dimensional space
  - Longer computation time
- e.g. Gaussian radial basis function (RBF)

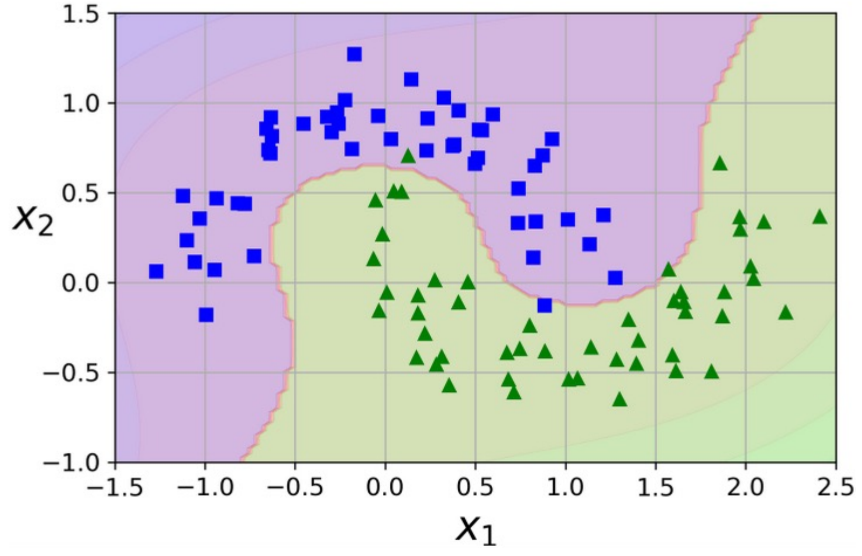


Distance between samples

$$\phi_\gamma(\mathbf{x}, \ell) = \exp\left(-\gamma \overbrace{\| \mathbf{x} - \ell \|}^{}{}^2\right)$$

New non-linear feature

Hyperparameter Gamma

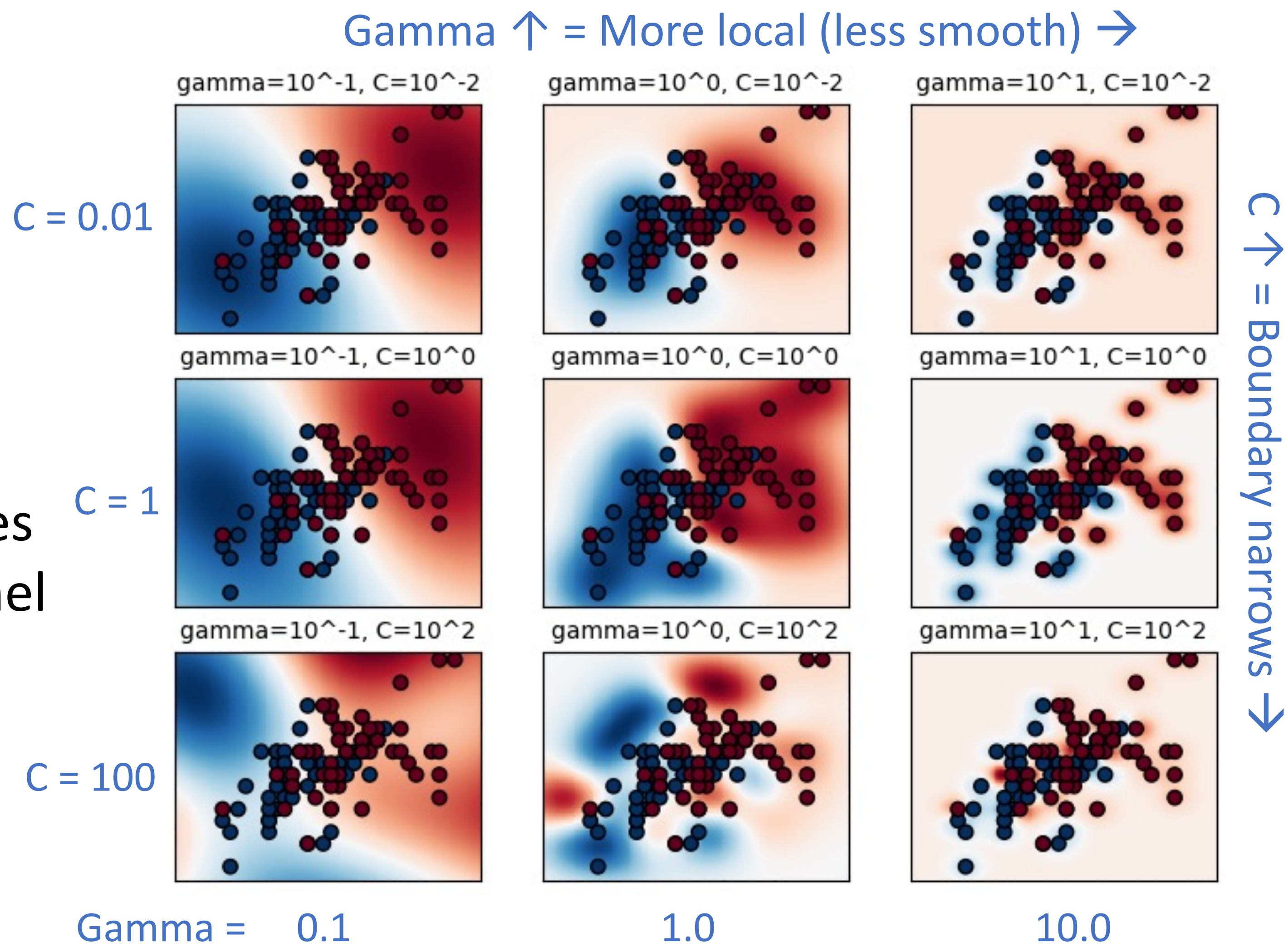


Images: Geron, Hands On ML

SVM Hyperparameters: C & Gamma

An example:

- 2 classes
- 50 samples each
- 2 (original) features
- Gaussian RBF kernel



Kernel “trick”

- Problem: we have introduced many extra features
  - More powerful classifier, but SLOW!
- Turns out that SVM optimisation only needs to know the *dot product* between each sample’s new high dimensional feature set (i.e. vector)
- Store these dot products in a matrix: N_{samp} x N_{samp}
- **Kernel** = a function K(a, b) that specifies the dot product
- Kernel trick: use kernel functions to get dot products without ever actually calculating the extra features
- Common kernels (best practice to start simple)

Linear: $K(\mathbf{a},\mathbf{b}) = \mathbf{a}^{\mathsf{T}}\mathbf{b}$

Linear classifier

Polynomial: $K(\mathbf{a},\mathbf{b}) = (\gamma\mathbf{a}^{\mathsf{T}}\mathbf{b} + r)^{d}$

Use polynomial feature combinations without having to compute them

Gaussian RBF: $K(\mathbf{a},\mathbf{b}) = \exp\left(-\gamma\|\mathbf{a}-\mathbf{b}\|^{2}\right)$

Use similarity to support vectors with Gaussian drop off

Sigmoid: $K(\mathbf{a},\mathbf{b}) = \tanh\left(\gamma\mathbf{a}^{\mathsf{T}}\mathbf{b} + r\right)$

Summary

- Logistic regression
  - Probability-based method
  - Uses sigmoid (logistic) function to get outputs in [0,1] range
  - *Sigmoid & Softmax* functions used in many deep learning models

- Support Vector Machines
  - Fit the *widest possible boundary* between two classes while limiting boundary violations
  - Create extra non-linear features → high dimensional space where they can be separated with a linear boundary in this space
  - Best to try simple kernels first, compare different kernels
  - Hyperparameters also need to be optimised
  - Data needs to be scaled uniformly (internally it is distance-based)