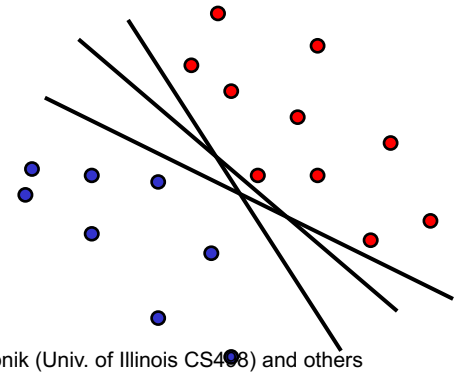
Support vector machines

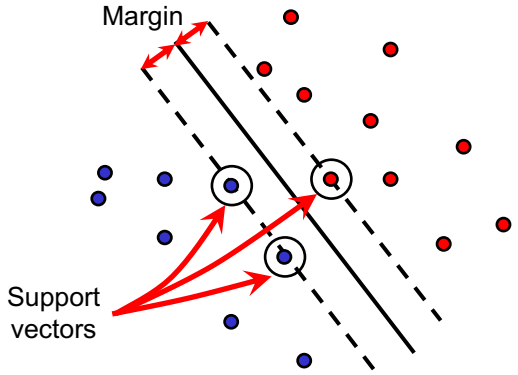
- When the data is linearly separable, which of the many possible solutions should we prefer?
  - SVM criterion: maximize the *margin*, or distance between the hyperplane and the closest training example



Slides courtesy L. Lazebnik (Univ. of Illinois CS498) and others

Support vector machines

- When the data is linearly separable, which of the many possible solutions should we prefer?
  - SVM criterion: maximize the *margin*, or distance between the hyperplane and the closest training example

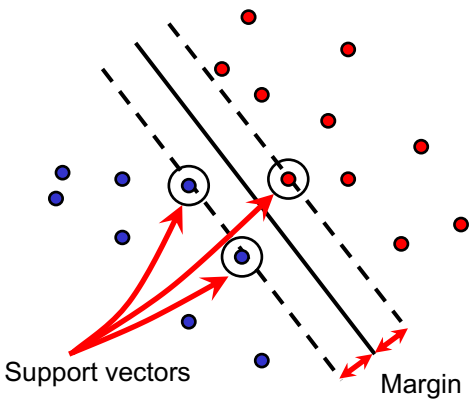


Finding the maximum margin hyperplane

- Define the *margin* as the distance between the hyperplane $w^T x + b = 0$ and the closest point x_i^*
- Distance between point and hyperplane is given by $\frac{|w^T x + b|}{\|w\|}$
- Assuming the data is linearly separable, we can fix the scale of w and b so that $y_i^*(w^T x_i^* + b) = 1$ and $y_i(w^T x_i + b) \geq 1$ for all other points
- Then the margin is $\frac{1}{\|w\|}$

Finding the maximum margin hyperplane

- Find separating hyperplane that maximizes the distance to the closest training example



Positive examples: $w^T x + b \geq 1$
Negative examples: $w^T x + b \leq -1$

For support vectors, $w^T x + b = \pm 1$

The margin is $1/\|w\|$

Finding the maximum margin hyperplane

- Maximize margin $1/\|w\|$ while correctly classifying all training data:

  y_i positive: $w^T x_i + b \geq 1$

  y_i negative: $w^T x_i + b \leq -1$

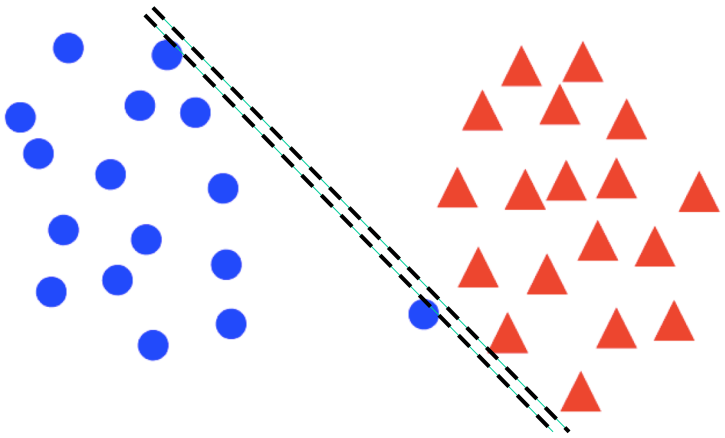
- Equivalent problem:

  $\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 1 \quad \forall i$

  - This is a quadratic objective with linear constraints: convex optimization, global optimum can be found using well-studied methods

"Soft margin" formulation

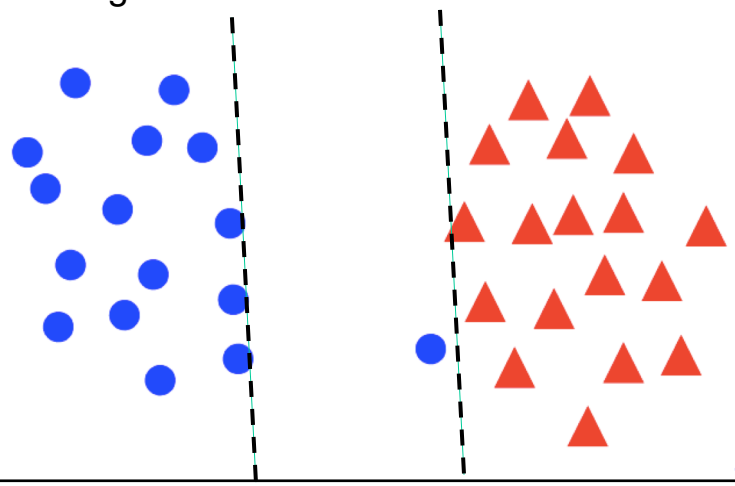
- What about non-separable data?
- And even for separable data, we may prefer a larger margin with a few constraints violated



Source

"Soft margin" formulation

- What about non-separable data?
- And even for separable data, we may prefer a larger margin with a few constraints violated

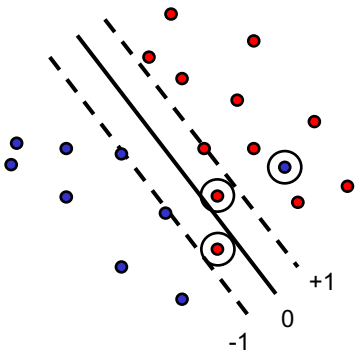


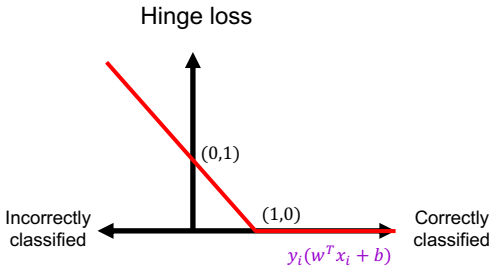
Source

"Soft margin" formulation

- Penalize margin violations using *hinge loss*:

$$\min_{w,b} \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{n} \max[0, 1 - y_i(w^T x_i + b)]$$

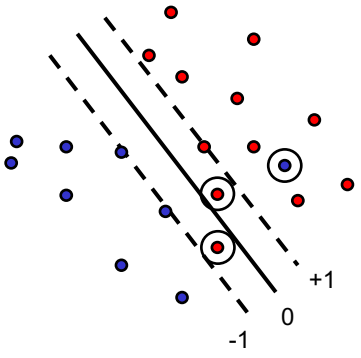


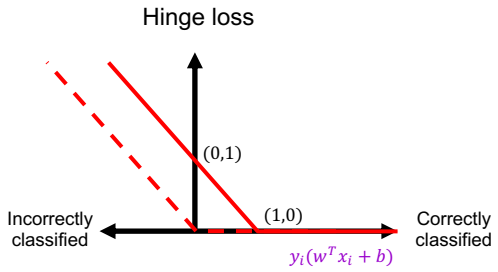


"Soft margin" formulation

- Penalize margin violations using *hinge loss*:

$$\min_{w,b} \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{n} \max[0, 1 - y_i(w^T x_i + b)]$$





Recall hinge loss used by the perceptron update algorithm!

"Soft margin" formulation

- Penalize margin violations using *hinge loss*:

$$\min_{w,b} \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{n} \max[0, 1 - y_i(w^T x_i + b)]$$

Maximize margin – a.k.a. *regularization*

Minimize misclassification loss

SGD optimization (omitting bias)

$$l(w, x_i, y_i) = \frac{\lambda}{2n}\|w\|^2 + \max[0, 1 - y_i w^T x_i]$$

$$\nabla l(w, x_i, y_i) = \frac{\lambda}{n} w - \mathbb{I}[y_i w^T x_i < 1] y_i x_i$$

Recall: $\frac{\partial}{\partial a}\max(0, a) = \mathbb{I}[a > 0]$

SGD optimization (omitting bias)

$$l(w, x_i, y_i) = \frac{\lambda}{2n}\|w\|^2 + \max[0, 1 - y_i w^T x_i]$$

$$\nabla l(w, x_i, y_i) = \frac{\lambda}{n} w - \mathbb{I}[y_i w^T x_i < 1] y_i x_i$$

- SGD update:
  - If $y_i w^T x_i < 1$: $w \leftarrow w + \eta\left(y_i x_i - \frac{\lambda}{n} w\right)$
  - Otherwise: $w \leftarrow w - \eta \frac{\lambda}{n} w$

S. Shalev-Schwartz et al., Pegasos: Primal Estimated sub-GrAdient SOlver for SVM, *Mathematical Programming*, 2011

SVM vs. perceptron

- SVM loss: $l(w, x_i, y_i) = \frac{\lambda}{2n}\|w\|^2 + \max[0, 1 - y_i w^T x_i]$
- SVM update:
  - If $y_i w^T x_i < 1$: $w \leftarrow w + \eta\left(y_i x_i - \frac{\lambda}{n} w\right)$
  - Otherwise: $w \leftarrow w - \eta\frac{\lambda}{n} w$
- Perceptron loss: $l(w, x_i, y_i) = \max[0, -y_i w^T x_i]$
- Perceptron update:
  - If $y_i w^T x_i < 0$: $w \leftarrow w + \eta\, y_i x_i$
  - Otherwise: do nothing
- What are the differences?

Dual SVM formulation

- SVM objective:

$$\min_{w,b} \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{n} \max[0, 1 - y_i(w^T x_i + b)]$$

- Directly solving for w using SGD is called the *primal* approach
- Instead, SVM optimization can be formulated to learn a classifier in the form

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i x_i^T x + b$$

by solving a *dual* optimization problem over α_i

Dual SVM formulation

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i x_i^T x + b, \qquad w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\|w\|^2 = \left(\sum_i \alpha_i y_i x_i\right)\left(\sum_j \alpha_j y_j x_j\right) = \sum_{i,j} \alpha_i \alpha_j y_i y_j \, x_i^T x_j$$

- The dual problem (given without derivation):

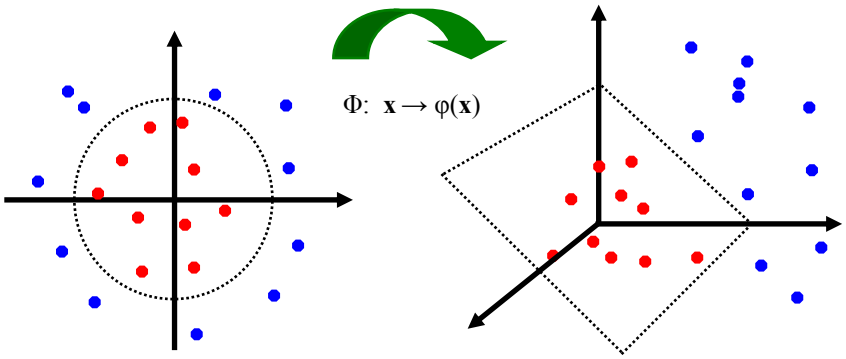
$$\max_{\alpha_i} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \, x_i^T x_j$$

$$\text{s.t.}\ 0 \le \alpha_i \le \frac{1}{\lambda}, \sum_i \alpha_i y_i = 0$$

- Important properties of the dual:
  - At the optimum, α_i are nonzero only for support vectors
  - Feature vectors appear only inside dot products $x_i^T x_j$: this enables nonlinear SVMs via *kernel functions*

Nonlinear SVMs

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable



Slide credit: Andrew Moore

Nonlinear SVMs

- *The kernel trick*: instead of explicitly computing the lifting transformation $\varphi(x)$, define a *kernel function*

$$K(x, x') = \varphi(x)^T \varphi(x')$$

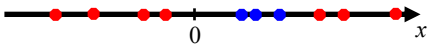
  - To be valid, the kernel function must satisfy *Mercer's condition*
- This gives a nonlinear decision boundary in the original feature space:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \varphi(x_i)^T \varphi(x) + b$$

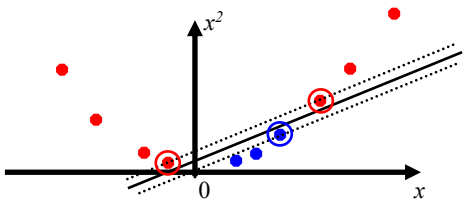
$$= \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b$$

Example

- Non-separable data in 1D:



- Apply mapping $\varphi(x) = (x, x^2)$:



$$\varphi(x)^T \varphi(x') = K(x, x') = xx' + x^2 x'^2$$

Kernel example 1: Polynomial

- Polynomial kernel with degree d and constant c:

$$K(x, x') = (x^T x' + c)^d$$

- What this looks like for $d = 2$:

$$x = (u, v), \qquad x' = (u', v')$$

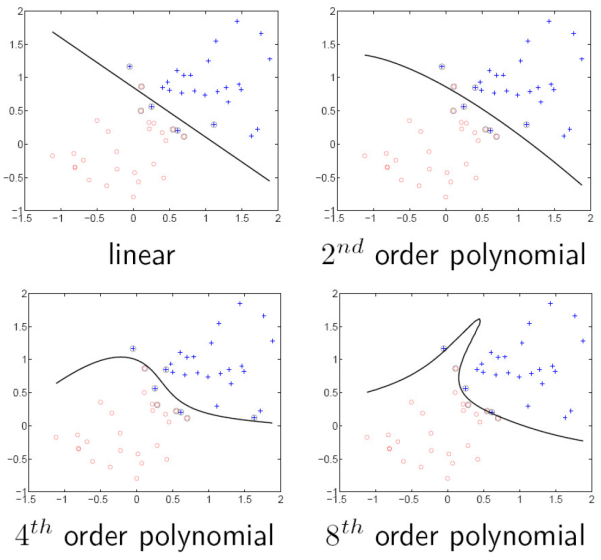
$$K(x, x') = (uu' + vv' + c)^2$$

$$= u^2u'^2 + v^2v'^2 + 2uu'vv' + cuu' + cvv' + c^2$$

$$\varphi(x) = (u^2, v^2, \sqrt{2}uv, \sqrt{c}u, \sqrt{c}v, \sqrt{c})$$

- Thus, the explicit feature transformation consists of all polynomial combinations of individual dimensions of degree up to d

Kernel example 1: Polynomial

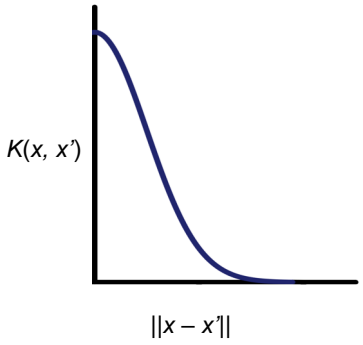


Kernel example 2: Gaussian

- Gaussian kernel with bandwith σ:

$$K(x, x') = \exp\left(-\frac{1}{\sigma^2}\|x - x'\|^2\right)$$

- Fun fact: the corresponding mapping $\varphi(x)$ is infinite-dimensional!

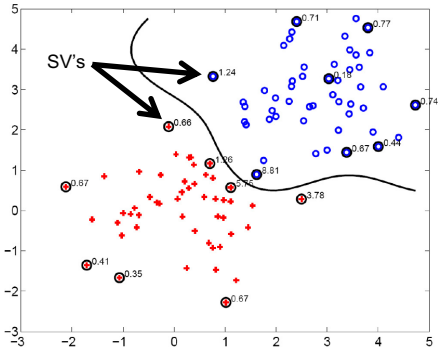


Kernel example 2: Gaussian

- Gaussian kernel with bandwith σ:

$$K(x, x') = \exp\left(-\frac{1}{\sigma^2}\|x - x'\|^2\right)$$

- The predictor $f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b$ is a sum of "bumps" centered on support vectors



It's also called a *Radial Basis Function* (RBF) kernel

Kernel example 2: Gaussian

- Gaussian kernel with bandwith σ:

$$K(x, x') = \exp\left(-\frac{1}{\sigma^2}\|x - x'\|^2\right)$$

- The predictor $f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b$ is a sum of "bumps" centered on support vectors
- How does the value of σ affect the behavior of the predictor?
  - What if σ is close to zero?
  - What if σ is very large?

SVM: Pros and cons

- Pros
  - Margin maximization and kernel trick are elegant, amenable to convex optimization and theoretical analysis
  - SVM loss gives very good accuracy in practice
  - Linear SVMs can scale to large datasets
  - Kernel SVMs are flexible, can be used with problem-specific kernels
  - Perfect "off-the-shelf" classifier, many packages are available
- Cons
  - Kernel SVM training does not scale to large datasets: memory cost is quadratic and computation cost even worse
  - "Shallow" method: predictor is a "flat" combination of kernel functions of support vectors and test example, no explicit feature representations are learned

Training linear classifiers

- **Given:** i.i.d. training data $\{(x_i, y_i), i = 1, \dots, n\}$, $y_i \in \{-1, 1\}$
- **Prediction function:** $f_w(x) = \text{sgn}(w^T x)$
- Classification with *bias*, i.e. $f_w(x) = \text{sgn}(w^T x + b)$, can be reduced to the case w/o bias by letting $w' = [w; b]$ and $x' = [x; 1]$

General recipe

- Find parameters w that minimize the sum of a *regularization loss* and a *data loss*:

$$\underbrace{\hat{L}(w)}_{\text{empirical loss}} = \underbrace{\lambda R(w)}_{\text{regularization}} + \underbrace{\frac{1}{n}\sum_{i=1}^{n} l(w, x_i, y_i)}_{\text{data loss}}$$

- Optimize by *stochastic gradient descent* (SGD): At each iteration, sample a single data point (x_i, y_i) and take a step in the direction *opposite* the gradient of the loss for that point:

$$w \leftarrow w - \eta \nabla_w \left[\frac{\lambda}{n} R(w) + l(w, x_i, y_i)\right]$$

Model 1: Linear regression

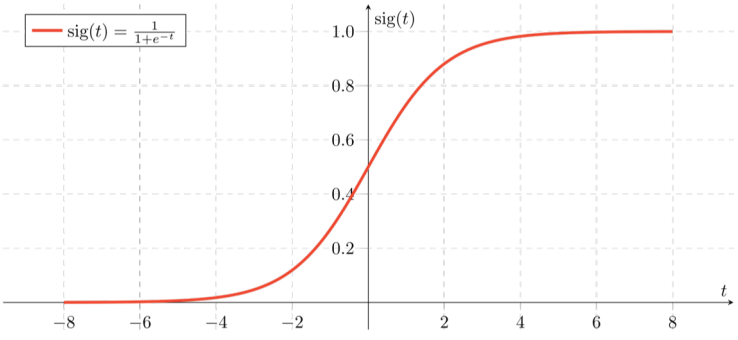
- **Regularization:** none
- **Data loss:** $l(w, x_i, y_i) = (w^T x_i - y_i)^2$
- **Interpretation:** *negative log likelihood* assuming $y|x$ is normally distributed with mean $w^T x$
- **Pros:** convex loss, easy to optimize
- **Cons:** conceptually inappropriate for classification, sensitive to outliers

Model 2: Logistic regression

- Posterior label probability or confidence is given by the *sigmoid function*:

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1+\exp(-w^T x)}$$

$$P_w(y = -1|x) = 1 - \sigma(w^T x) = \sigma(-w^T x)$$



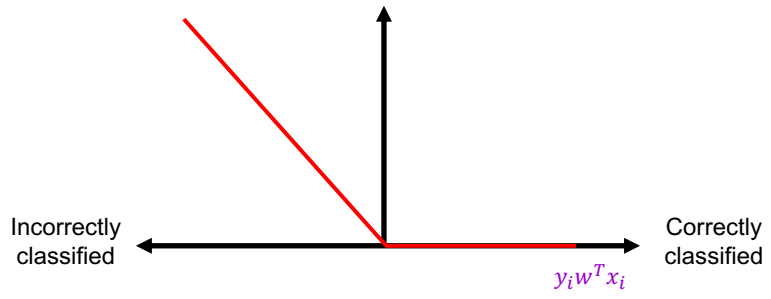
Model 2: Logistic regression

- **Regularization:** none
- **Data loss:**
  $l(w, x_i, y_i) = -\log P_w(y_i|x_i) = -\log \sigma(y_i w^T x_i)$
- **Interpretation:** negative log likelihood assuming Gaussian *class-conditional distributions* $P(x|y = 1)$

Model 3: Perceptron training algorithm

- **Regularization:** none
- **Data loss:** *hinge loss*

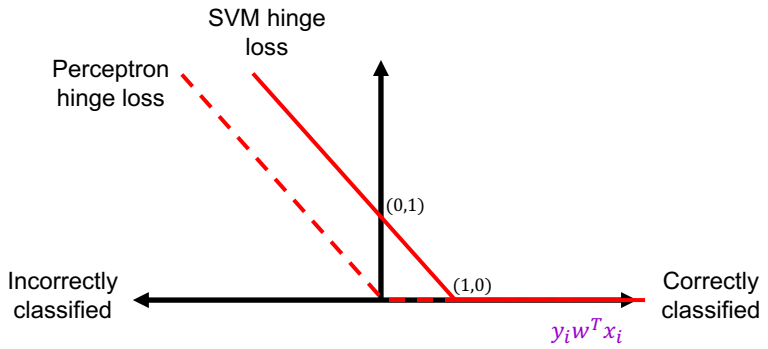
$$l(w, x_i, y_i) = \max(0, -y_i w^T x_i)$$



Model 4: Support vector machines

- **Regularization:** $R(w) = \frac{1}{2}\|w\|^2$
- **Data loss:** *hinge loss*

$$l(w, x_i, y_i) = \max(0, 1 - y_i w^T x_i)$$



Model 4: Support vector machines

- **Regularization:** $R(w) = \frac{1}{2}\|w\|^2$
- **Data loss:** *hinge loss*

$$l(w, x_i, y_i) = \max(0, 1 - y_i w^T x_i)$$

- **Interpretation:** maximize margin while minimizing constraint violations

SGD updates

- Linear regression:

$$w \leftarrow w + \eta\,(y_i - w^T x_i)\,x_i$$

- Logistic regression:

$$w \leftarrow w + \eta\,\sigma(-y_i w^T x_i)\,y_i x_i$$

- Perceptron:

$$w \leftarrow w + \eta\,\mathbb{I}[y_i w^T x_i < 0]\,y_i x_i$$

- SVM:

$$w \leftarrow \left(1 - \frac{\eta\lambda}{n}\right) w + \eta\,\mathbb{I}[y_i w^T x_i < 1]\,y_i x_i$$

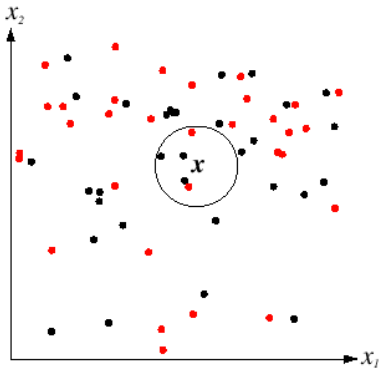
Supervised learning outline revisited

1. **Collect data and labels**
2. **Specify model:** select model type, loss function, *hyperparameters*
3. **Train model:** find the *parameters* of the model that minimize the empirical loss on the training data

- What is the right methodology for step 2?

Hyperparameters

- K in K-nearest-neighbor
  - What if K is too large?
  - What if K is too small?



Hyperparameters

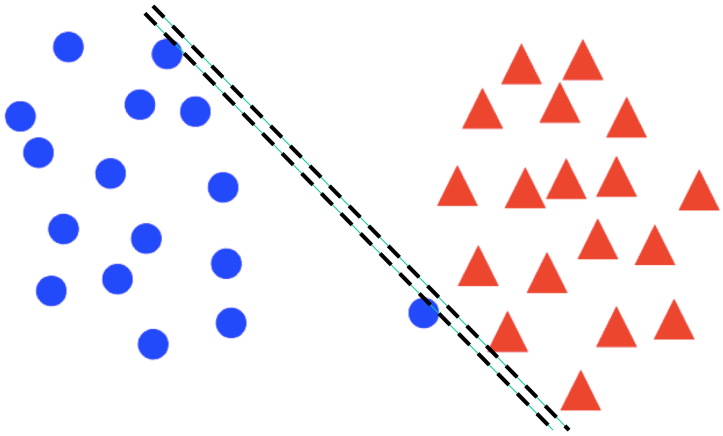
- Regularization constant λ in SVM optimization

$$\min_{w,b} \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{n} \max[0, 1 - y_i(w^T x_i + b)]$$

  - What if λ is too large?
  - What if λ is too small?

Hyperparameters

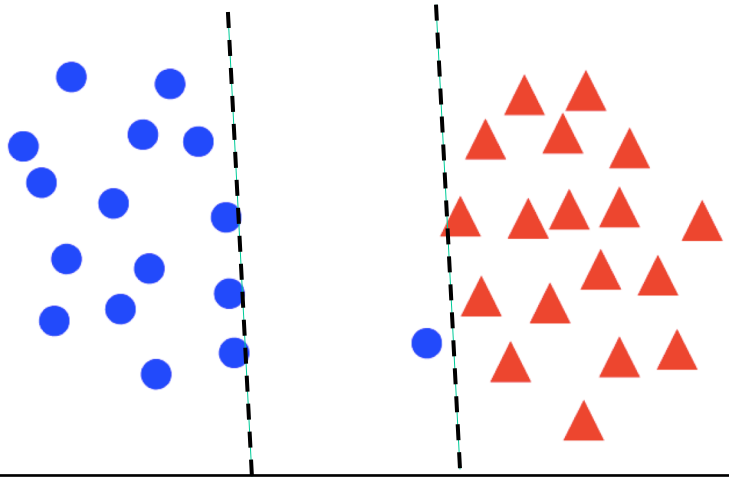
- Regularization constant λ in SVM optimization
- Recall: tradeoff between margin and constraint violations



Source

Hyperparameters

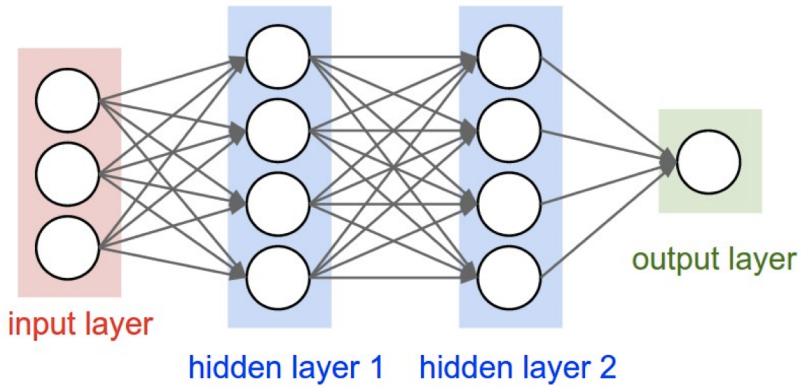
- Regularization constant λ in SVM optimization
- Recall: tradeoff between margin and constraint violations



Source

Hyperparameters in multi-layer networks

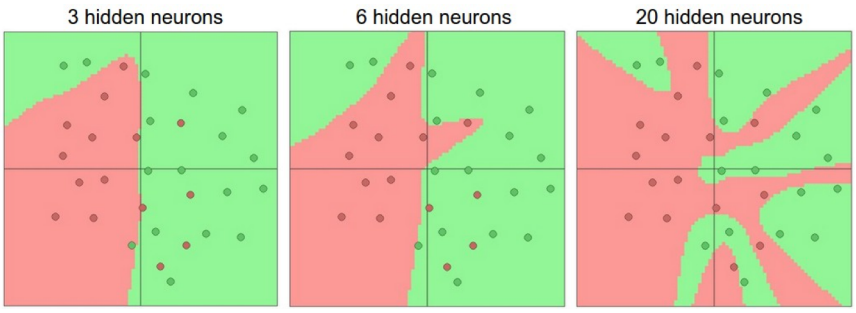
- Number of layers, number of units per layer



Source: Stanford 231n

Hyperparameters in multi-layer networks

- Number of layers, number of units per layer

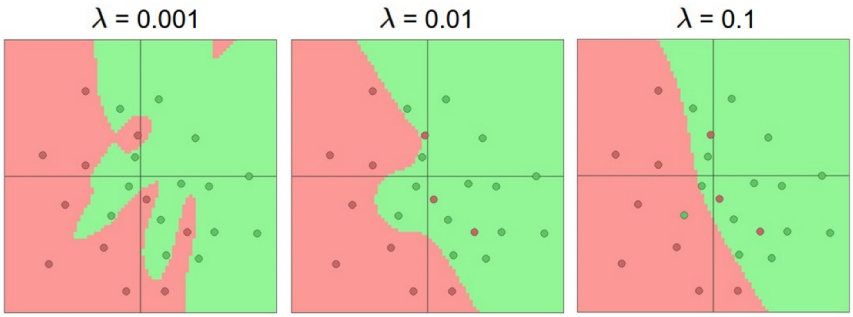


Number of hidden units in a two-layer network

Source: Stanford 231n

Hyperparameters in multi-layer networks

- Number of layers, number of units per layer
- Regularization



Source: Stanford 231n

Hyperparameters in multi-layer networks

- Number of layers, number of units per layer
- Regularization
- SGD settings: learning rate schedule, number of epochs, minibatch size, etc.

Hyperparameters: Summary

- Hyperparameter types
  - K in K-NN
  - In SVMs: regularization constant, kernel type and constants
  - In neural networks: number of layers, number of units per layer, regularization
  - SGD settings: learning rate schedule, number of epochs, minibatch size, etc.
- We can think of our hyperparameter choices as defining the "complexity" of the model and controlling its generalization ability

Bias-variance tradeoff

- Generalization (test) error of learning algorithms has two main components:
  - **Bias:** error due to simplifying model assumptions
  - **Variance:** error due to randomness of training set
- The tradeoff between these components is determined by the complexity of the model and the amount of training data

High bias, low variance

Low bias, high variance

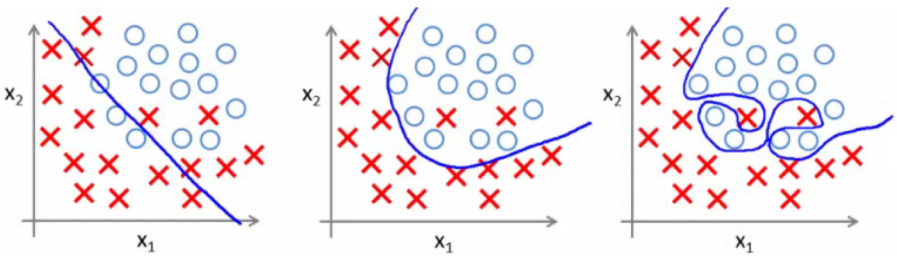


Figure source

Underfitting and overfitting

- **Underfitting:** training and test error are both *high*
  - Model does an equally poor job on the training and the test set
  - The model is too "simple" to represent the data or the model is not trained well
- **Overfitting:** Training error is *low* but test error is *high*
  - Model fits irrelevant characteristics (noise) in the training data
  - Model is too complex or amount of training data is insufficient

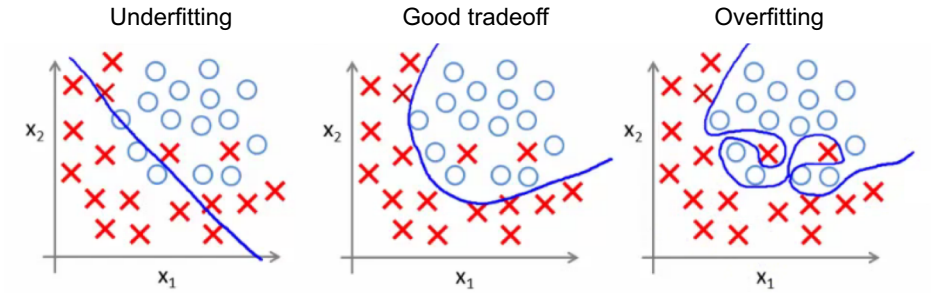
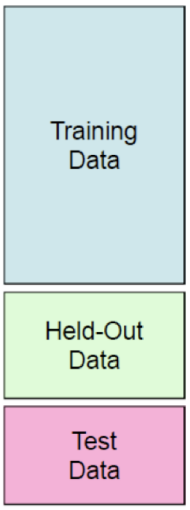


Figure source

Hyperparameter search in practice

- Iterate for a range of hyperparameter choices:
  - Learn parameters on the ***training data***
  - Measure accuracy on the ***held-out*** or ***validation data***
- Finally, measure accuracy on the ***test data***
- **Crucial:** do not peek at test set during hyperparameter search!



Hyperparameter search in practice

- Variant: *K-fold cross-validation*
  - Partition the data into K groups
  - In each run, select one of the groups as the validation set

What's the big deal?

Baidu admits cheating in international supercomputer competition

ZDNet

Baidu recently apologised for violating the rules of an international supercomputer test in May, when the Chinese search engine giant claimed to beat both Google and Microsoft on the ImageNet image-recognition test.

By Cyrus Lee | June 10, 2015 -- 00:15 GMT (17:15 PDT) | Topic: China

TECHNOLOGY

The New York Times

Computer Scientists Are Astir After Baidu Team Is Barred From A.I. Competition

By JOHN MARKOFF JUNE 3, 2015

engadget

Baidu caught gaming recent supercomputer performance test

by Andrew Tarantola | @terrortola | June 3rd 2015 At 11:09pm