SAIR
Spatial AI & Robotics Lab

CSE 473/573-A

L19: CLASSIFICATION

Chen Wang
Spatial AI & Robotics Lab
Department of Computer Science and Engineering
University at Buffalo The State University of New York

Machine Learning

	Supervised Learning	Unsupervised Learning
Discrete	classification or categorization	clustering
Continuous	regression	dimensionality reduction

The machine learning framework

- Apply a prediction function to a feature representation of the image to get the desired output:

f() = “apple”

f() = “tomato”

f() = “cow”

SAIR
Spatial AI & Robotics Lab

Slide credit: L. Lazebnik

The machine learning framework

$$y = f(\mathbf{x})$$

output

prediction function

Image feature

- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to an unseen *test example* $\mathbf{x}$ and output the predicted value $y = f(\mathbf{x})$

Image Classification

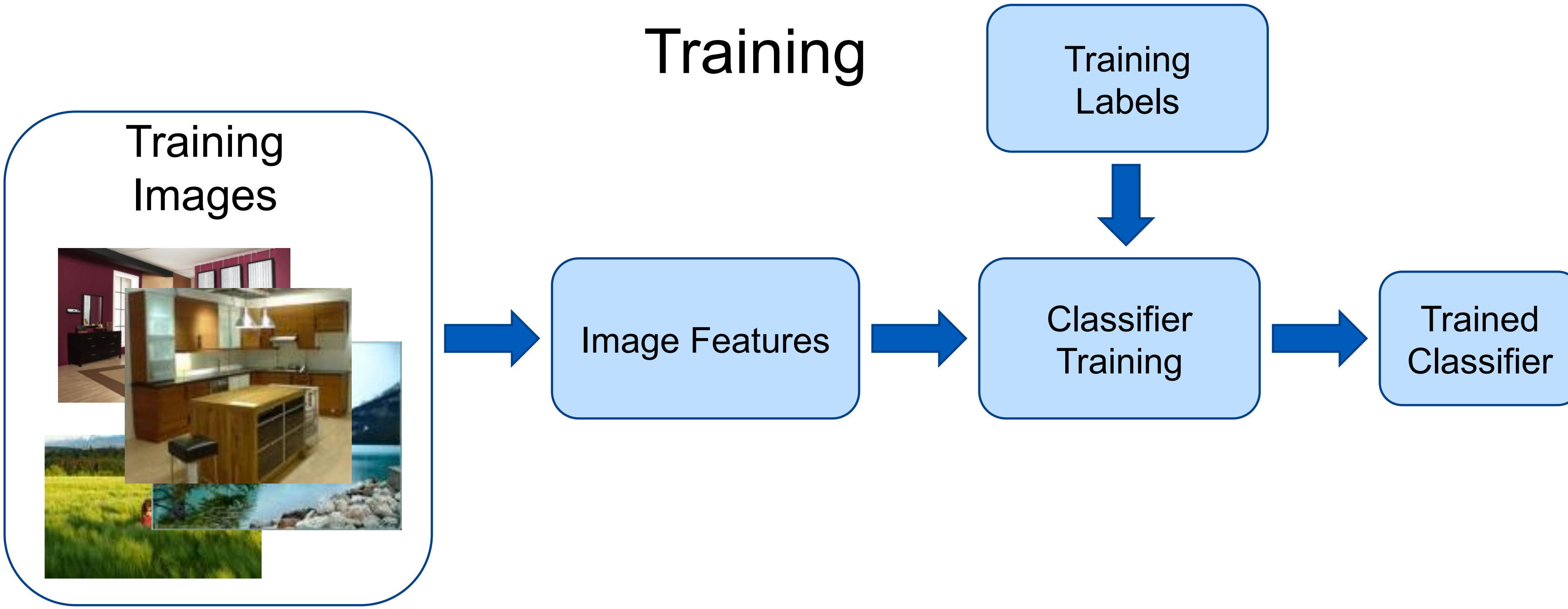
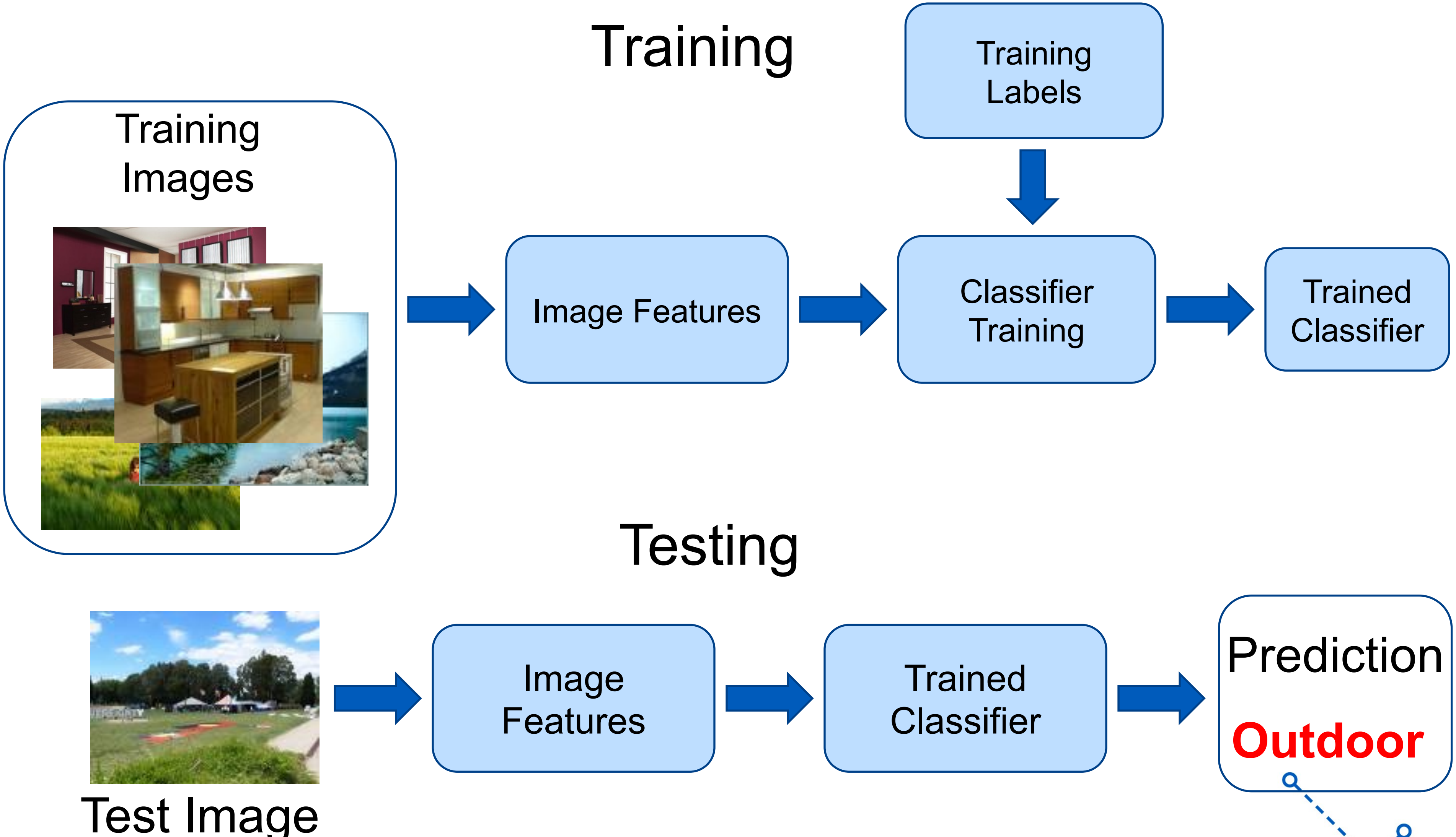


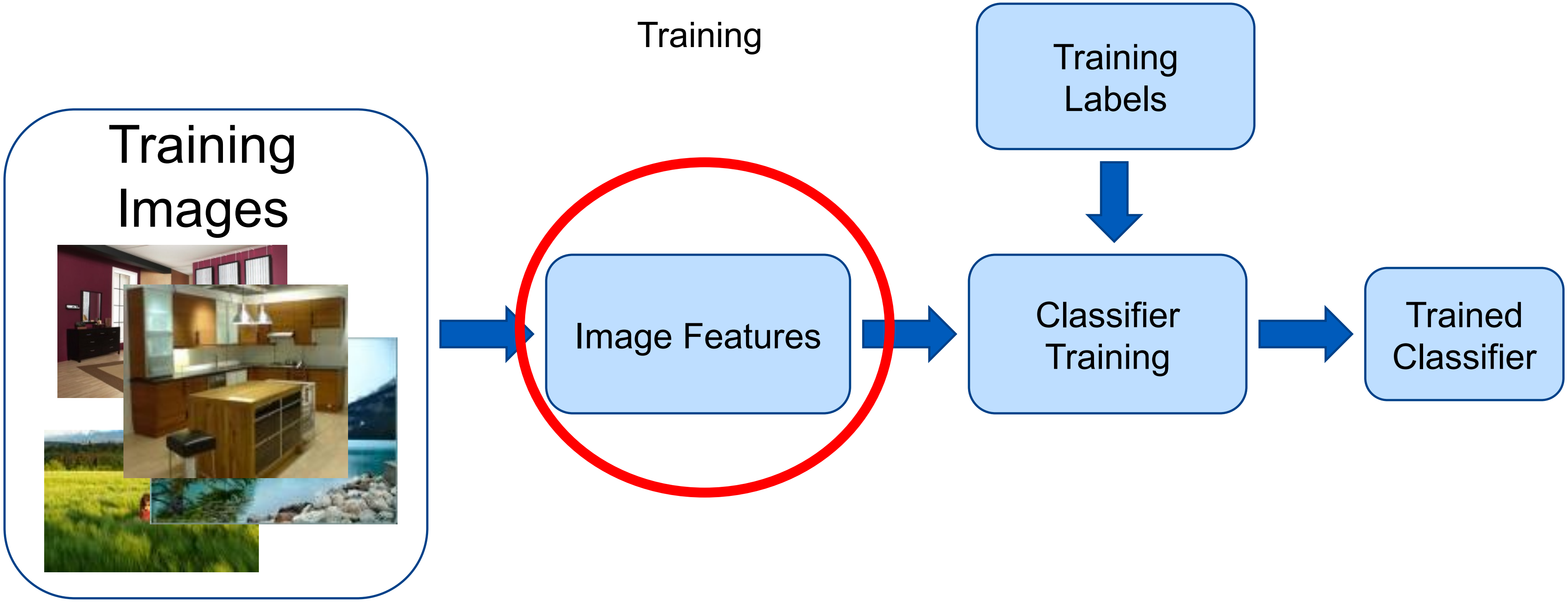
Image Categorization



Example: Scene Classification

- Is this a kitchen?

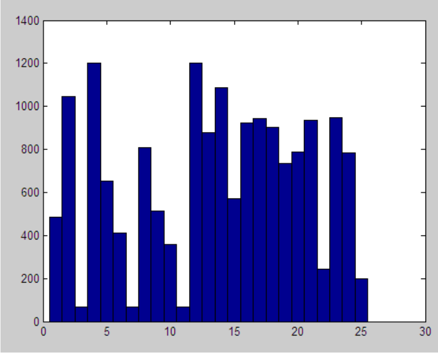
Image features

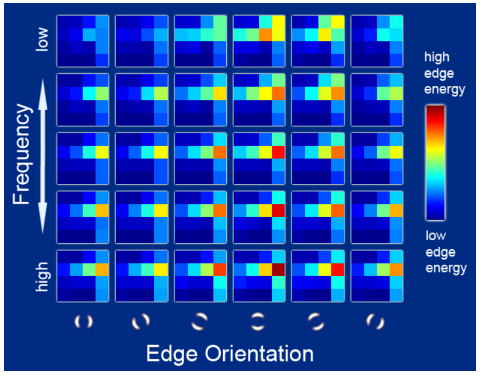


S A I R
Spatial AI & Robotics Lab

Types of Features

- Raw pixels
- Histograms
- SIFT descriptors
- CNN features
- …

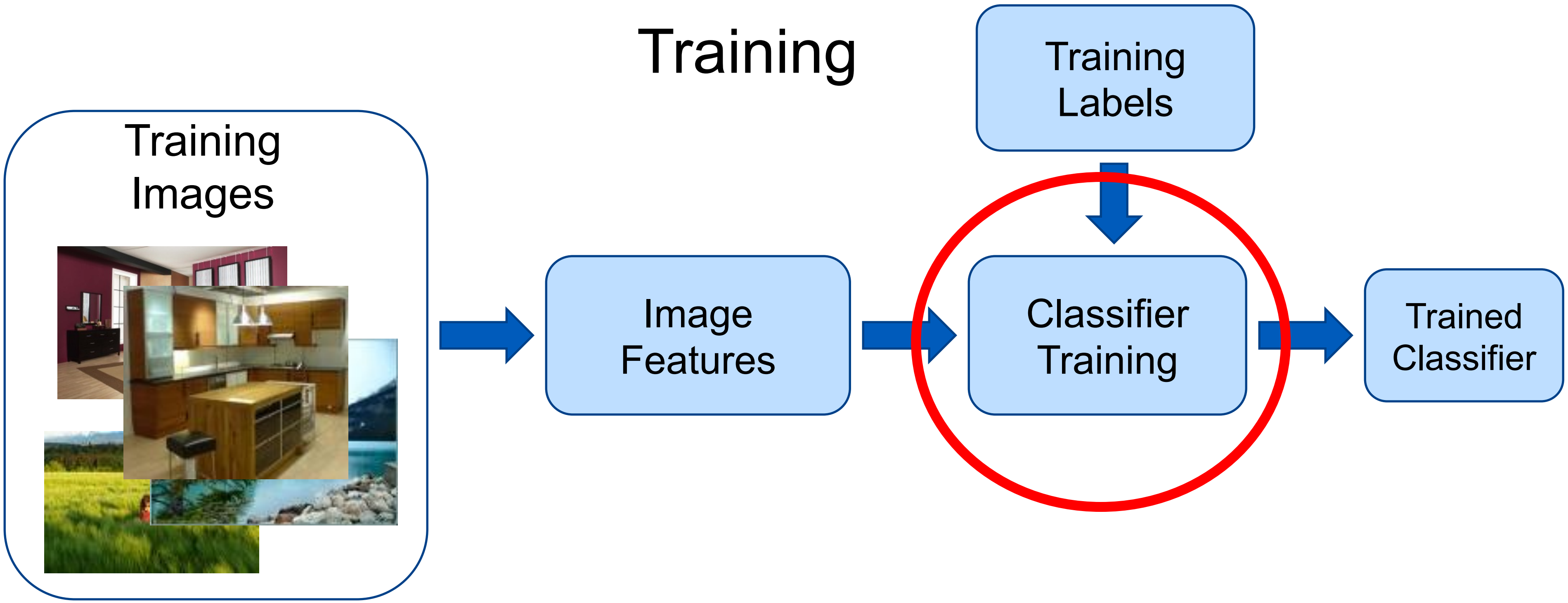




Desirable Properties of Features

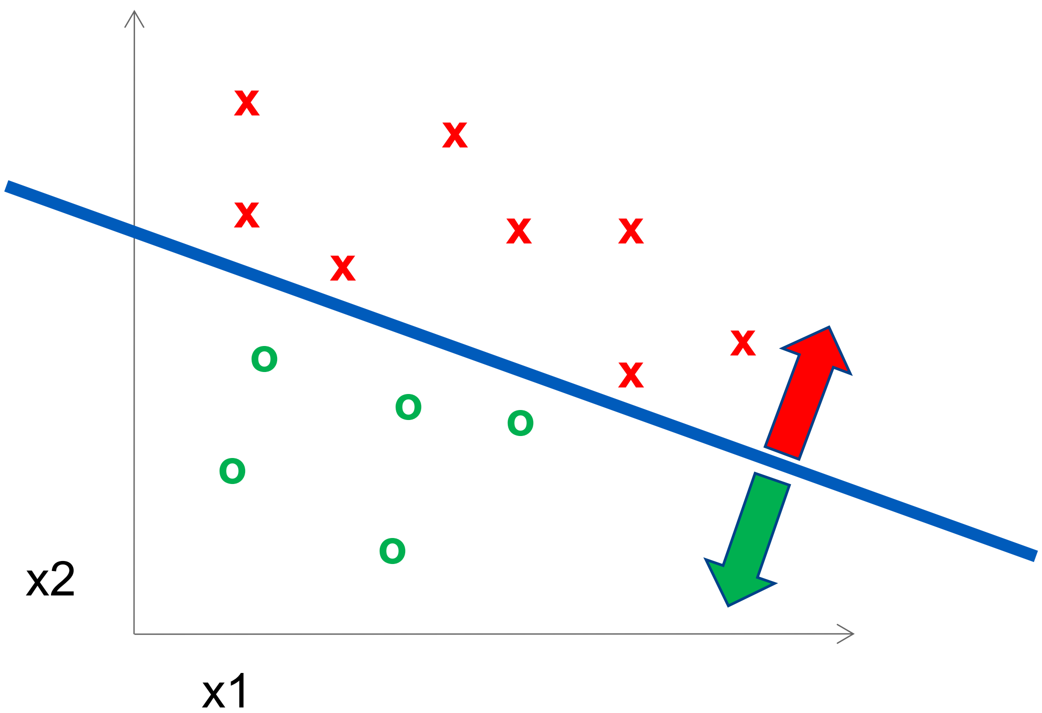
- Coverage
  - Ensure that all relevant info is captured
- Concision
  - Minimize number of features without sacrificing coverage
- Directness/Uniqueness
  - Ideal features are independently useful for prediction

Classifiers



Learning a classifier

Given subset of features with corresponding labels, learn a function to predict the unseen labels.



Desirable properties of classifiers

- Can you list desirable properties of a classifier.
  - Minimize Prediction Error
  - Minimize Training Time
  - Minimize Training Data
  - Nonlinear Separability
  - Makes Hard and Soft Decisions
  - Generalizable
  - Can be Tuned
  - On-line adaptation
  - Non-Parametric

Generalization

- How well does a learned model generalize from the data it was trained on to a new test set?

Training set (labels known)

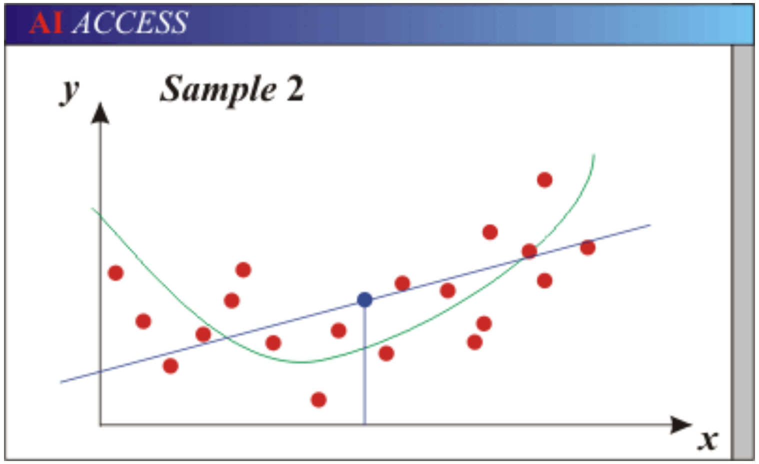
Test set
(labels unknown)

L. Lazebnik

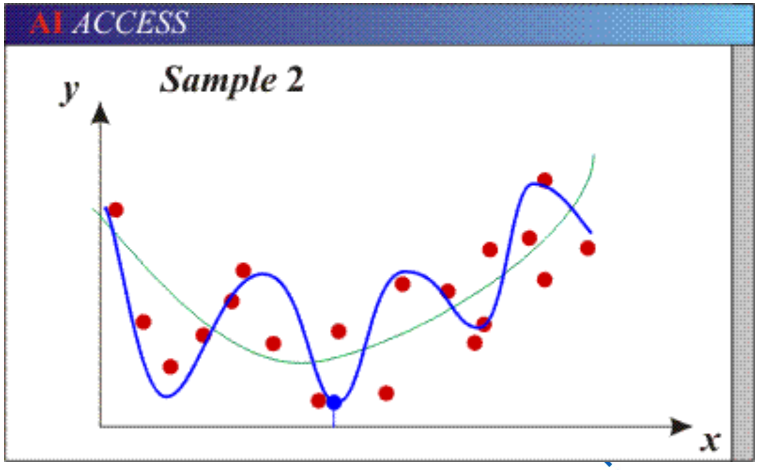
Bias-Variance Trade-off

- No Free Lunch Theorem

- Models with too few parameters are inaccurate because of a large bias (not enough flexibility).



- Models with too many parameters are inaccurate because of a large variance (too much sensitivity to the sample).



Bias-Variance Trade-off

$$E(MSE) = Var\{noise\} + bias^2 + Var\{label\}$$

Inherent / Unavoidable error

Error due to incorrect assumptions

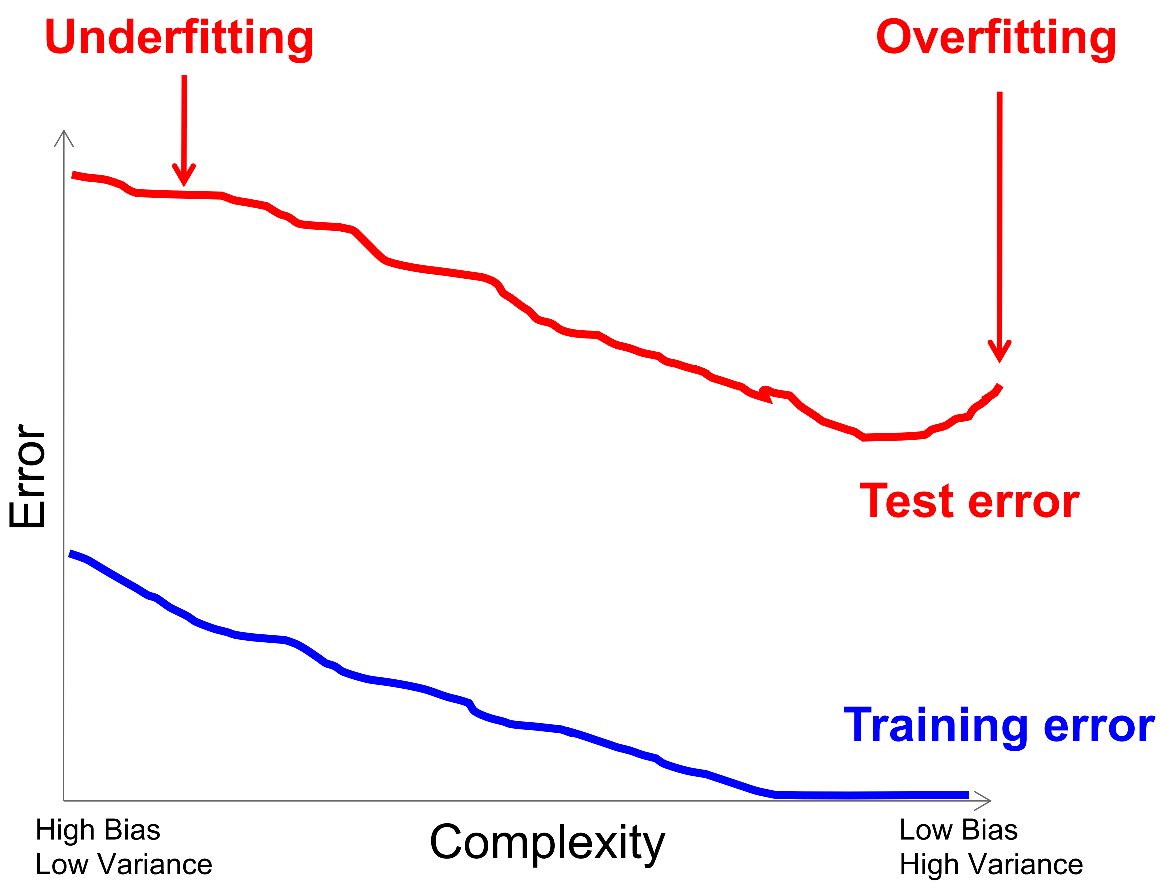
Error due to variance of training samples

See explanations of bias-variance and Bishop's "Neural Networks" book

Generalization

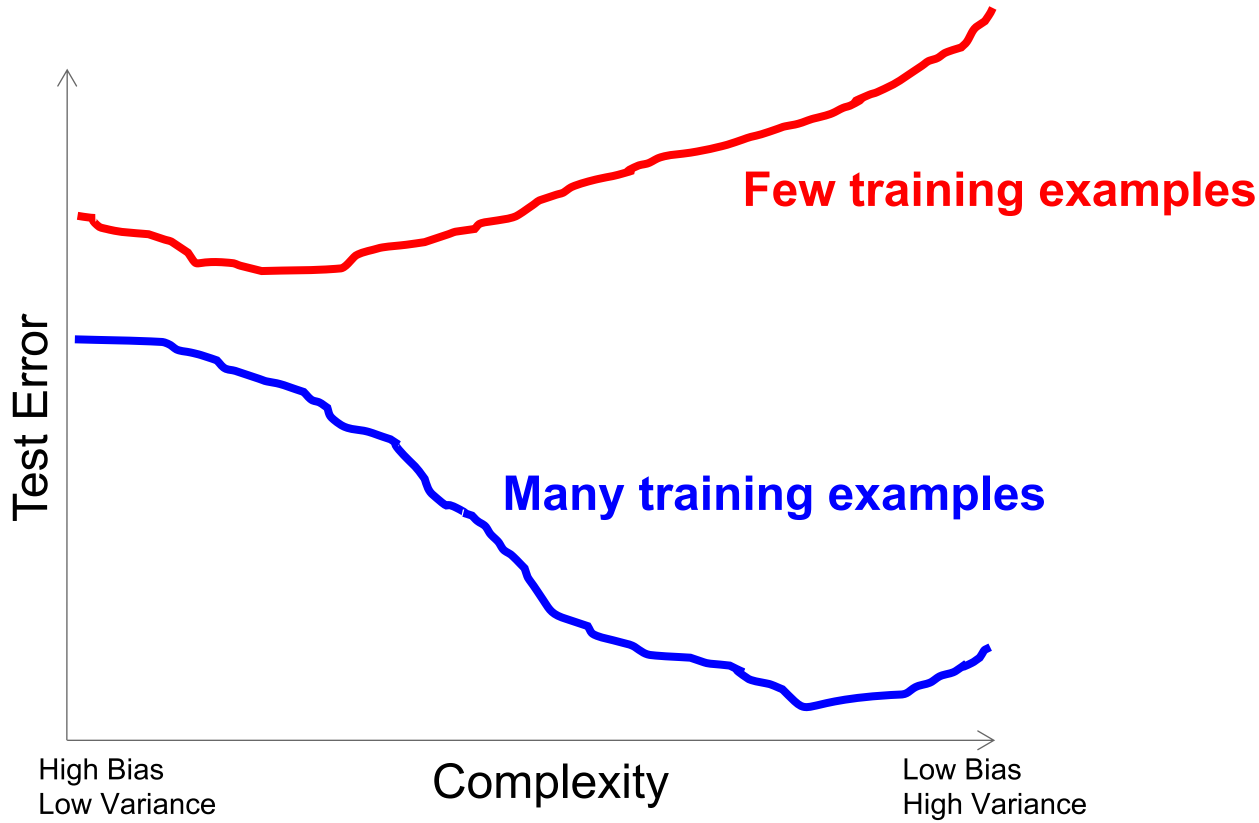
- Components of generalization error
  - **Bias**: how much the average model differs from the true model, over all training sets?
    - Error due to inaccurate assumptions made by the model.
  - **Variance**: how much models estimated from different training sets differ from each other.
  - **Underfitting**: model is too “simple” to represent all the relevant class characteristics
    - High bias (few degrees of freedom) and low variance
    - High training error and high test error
  - **Overfitting**: model is too “complex” and fits irrelevant characteristics (noise) in the data
    - Low bias (many degrees of freedom) and high variance
    - Low training error and high test error

Bias-variance tradeoff



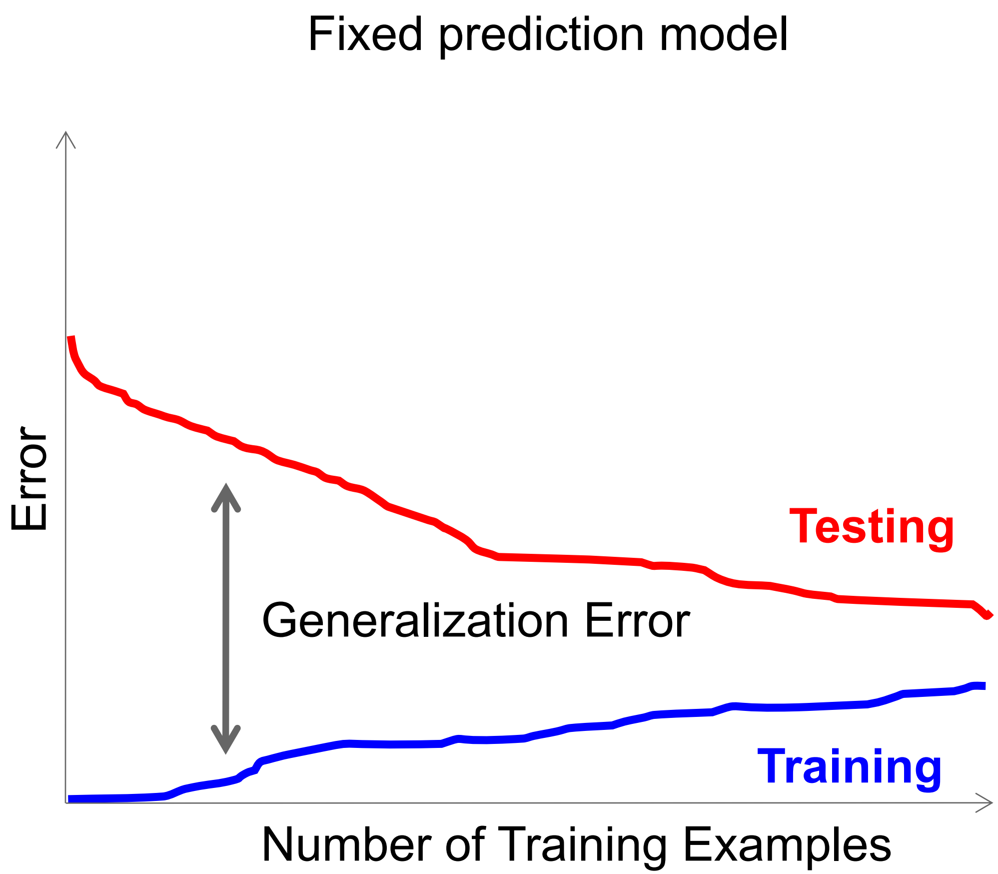
D. Hoiem

Bias-variance tradeoff



D. Hoiem

Effect of Training Size



The perfect classification algorithm

- Objective function: encodes the right loss for the problem

- Parameterization: makes assumptions that fit the problem

- Regularization: right level of regularization for amount of training data

- Training algorithm: can find parameters that maximize objective on training set

- Inference algorithm: can solve for objective function in evaluation

Things to Remember

- No classifier is inherently better than any other
  - You need to make assumptions to generalize

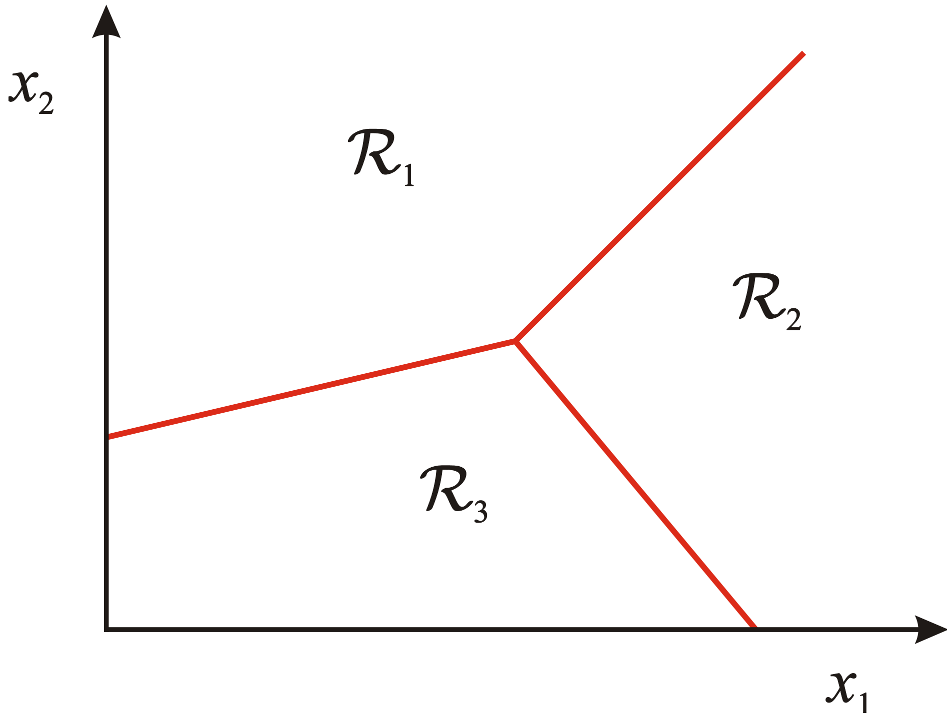
- Three kinds of error
  - Inherent: unavoidable
  - Bias: due to inaccurate model
  - Variance: due to inability to perfectly estimate parameters from limited data

Some classifiers

- **K-nearest neighbor**
- **Support Vector Machine (SVM)**
- Boosted Decision Trees
- Neural networks
- Naïve Bayes
- Bayesian network
- Logistic regression
- Randomized Forests
- …

Classification

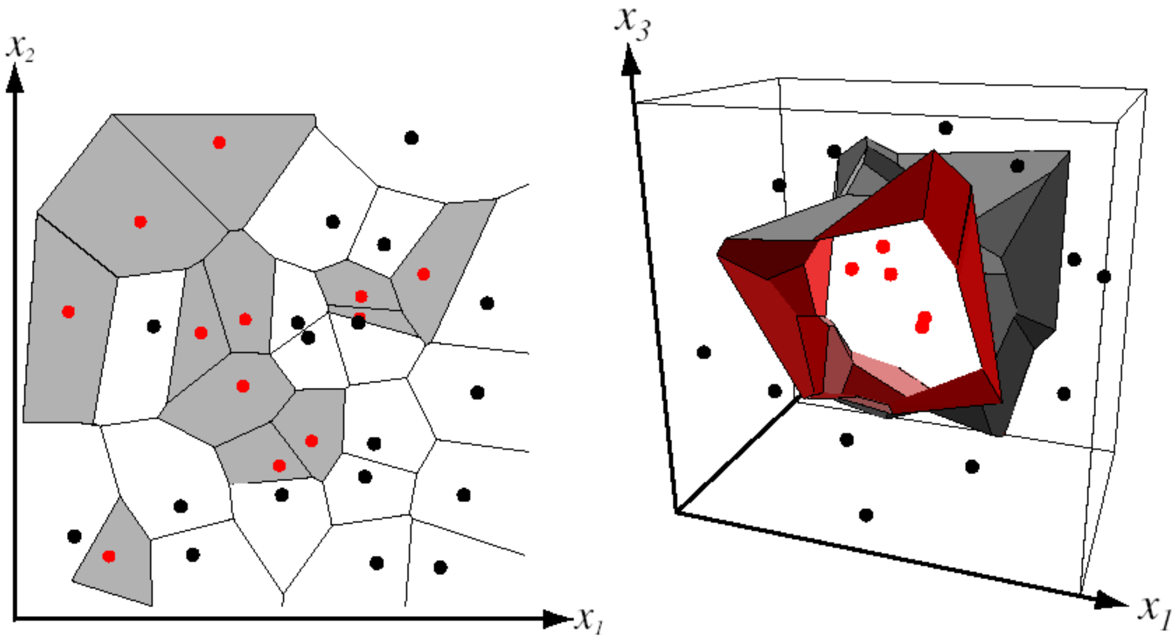
- Assign input vectors to one of two (more) classes
- Any **decision rule** divides input space into ***decision regions*** separated by ***decision boundaries***



L. Lazebnik

Nearest Neighbor Classifier

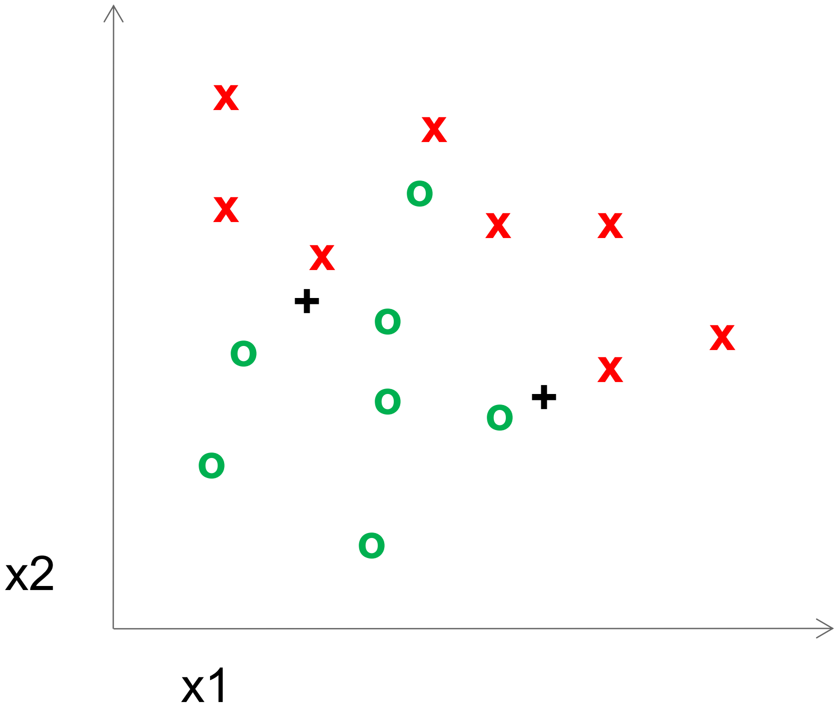
- Assign label of nearest training data point to each test data point



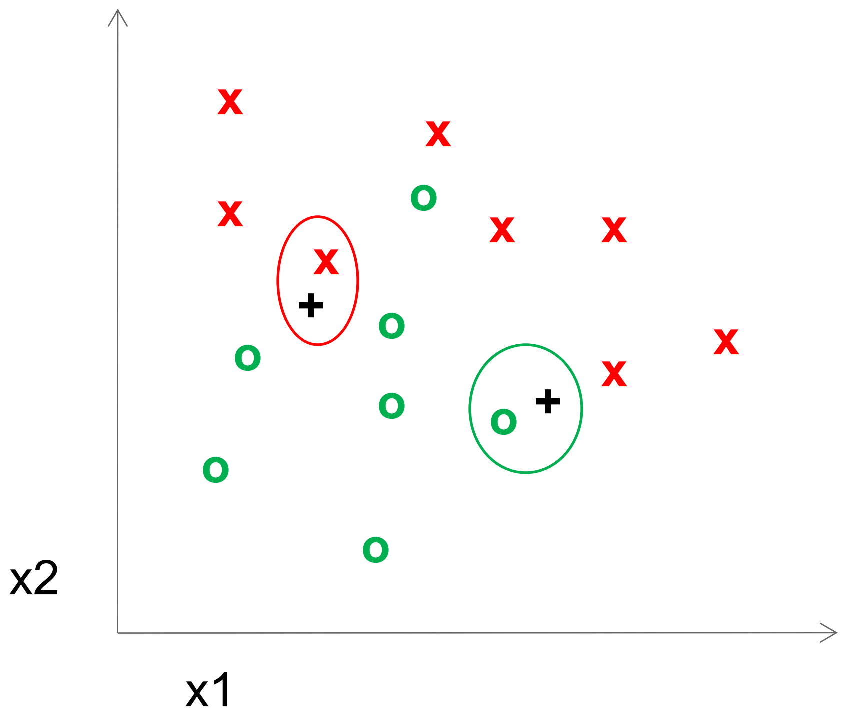
Voronoi partitioning of feature space for two-category 2D and 3D data

D. Lowe

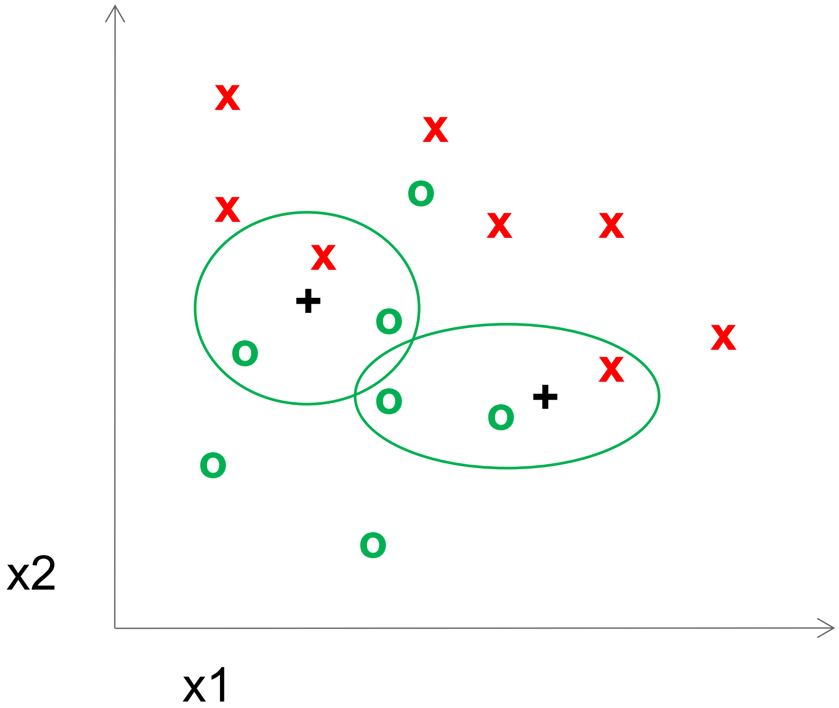
K-nearest neighbor



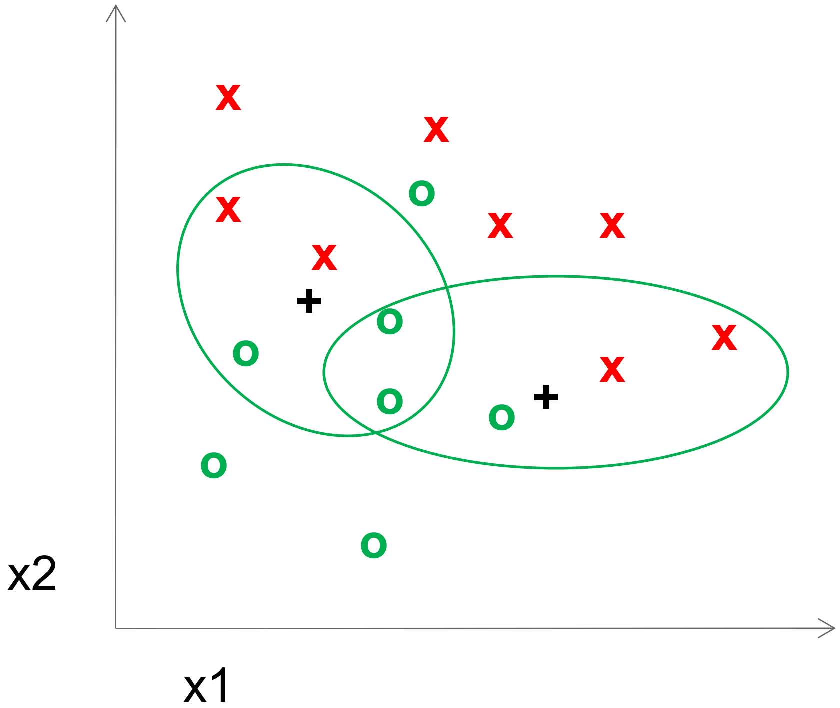
1-nearest neighbor



3-nearest neighbor



5-nearest neighbor

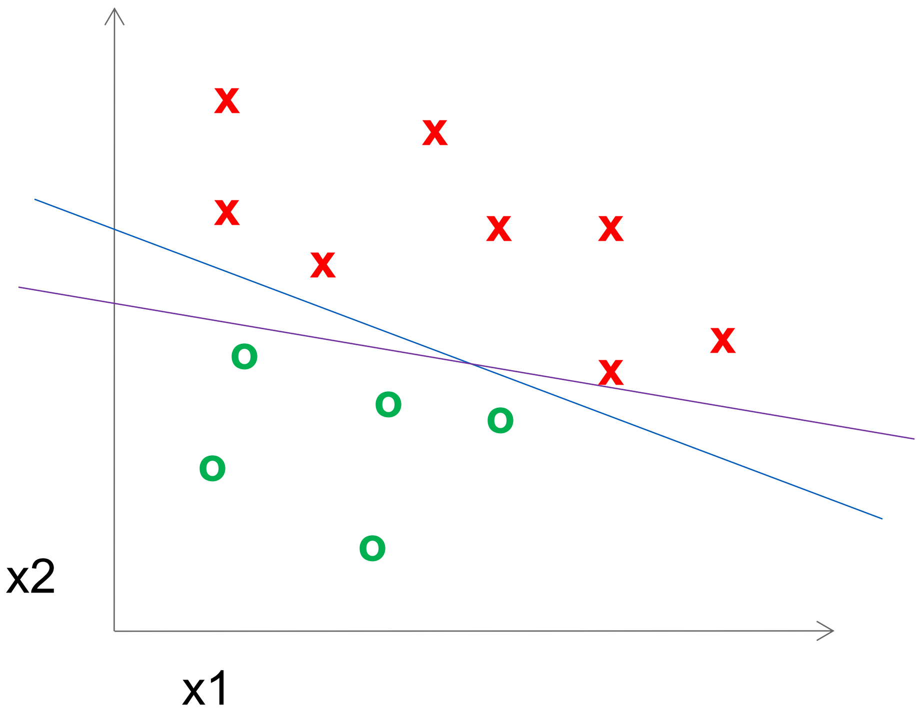


Using K-NN

- Simple, a good one to try first

- With infinite examples, 1-NN provably has error that is at most twice Bayes optimal error

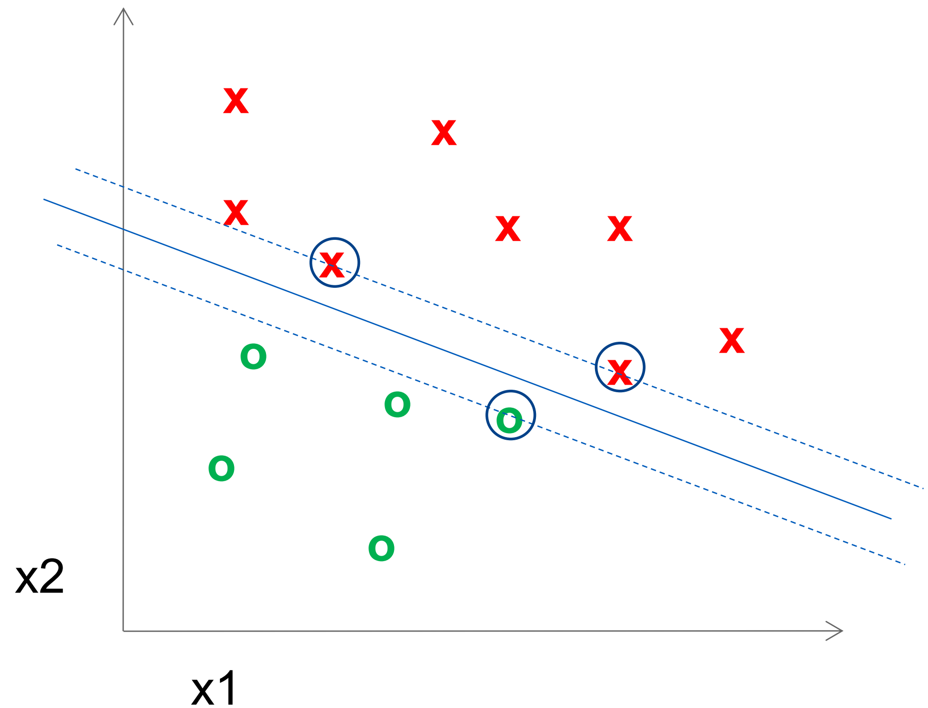
Linear Support Vector Machine (SVM)



- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

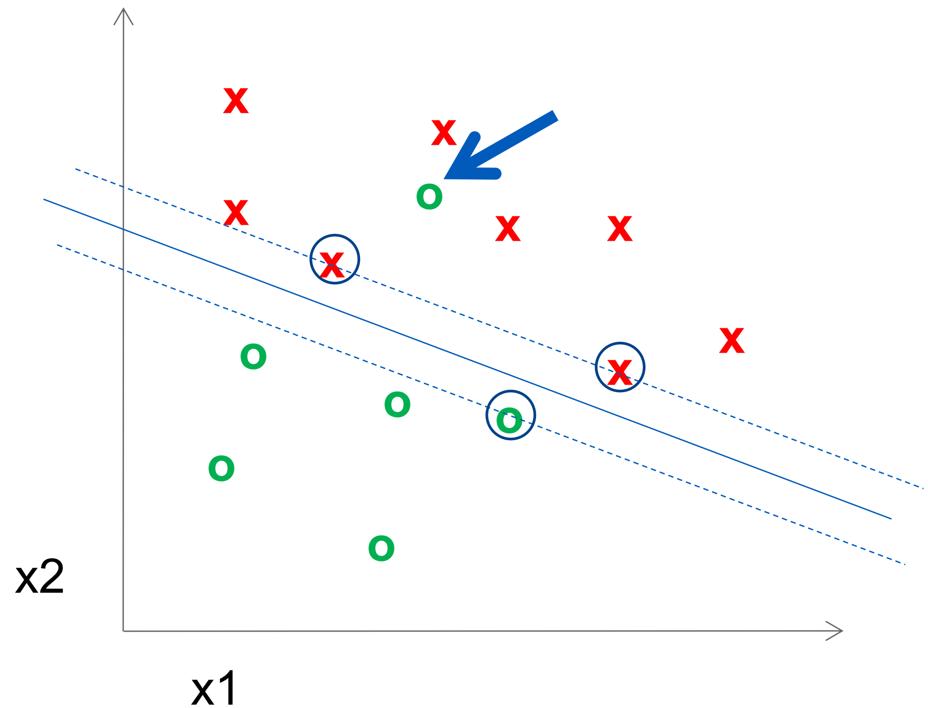
Linear Support Vector Machine (SVM)



- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

Linear Support Vector Machine (SVM)



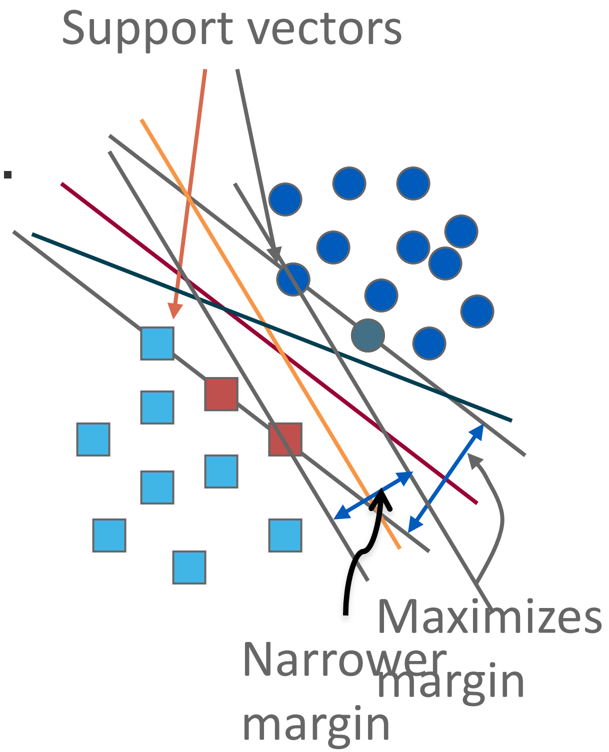
- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

Support Vector Machine (SVM)

- SVMs maximize the margin around the separating hyperplane.
  - A.k.a, large margin classifiers

- The decision function is fully specified by a subset of training samples, the support vectors.



Soft SVM

- Cost Function (Hinge loss)

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

$$= \max\{0, 1 - y * f(x)\}$$

$$\stackrel{\text{def}}{=} \left(1 - y * f(x)\right)_+$$

SAIR
Spatial AI & Robotics Lab

Linear Soft SVM

- Assume sample x is in a higher dimension.
  - The linear model is $f(x) = \langle x, w \rangle$.
- The loss function with a regularization term.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

- Can be solved via gradient descent.

Stochastic Gradient Descent (SGD)

- Given loss function, we calculate the gradient (partial derivative) with respective the model weights.

$$\frac{\delta}{\delta w_k} \lambda \parallel w \parallel^2 \quad = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k} \left(1 - y_i \langle x_i, w \rangle\right)_+ \quad = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

- We update the weights by gradient descent.

$$w = w - \alpha \cdot (2\lambda w)$$

Gradient Update — No misclassification

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

Gradient Update — Misclassification

SGD for solving linear SVM

- Assume there is **NO regularization**, during each iteration
  - Set the batch size T and do:

parameter: T;

initialization: $\theta^{(1)} = \mathbf{0}$;

for $t = 1, \cdots, T$ **do**

$\quad \mathbf{w}^{(t)} = \frac{1}{\lambda t}\theta^{(t)}$;

$\quad$ Choose i randomly from $[N]$;

$\quad$ **if** $y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \geqslant 1$ **then**

$\quad\quad \theta^{(t+1)} = \theta^{(t)}$;

$\quad$ **else**

$\quad\quad \theta^{(t+1)} = \theta^{(t)} + y_i \mathbf{x}_i$;

Result: $\bar{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}^{(t)}$

Number of sample in each iteration (batch)

Record weight gradient update from each sample

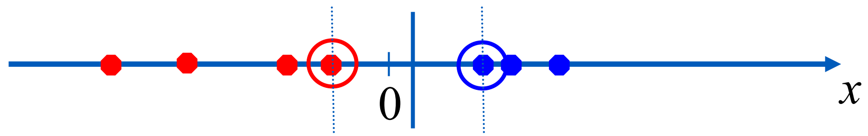
Averaged updated weights in the batch

- Update the solution $w = w + \bar{w}$

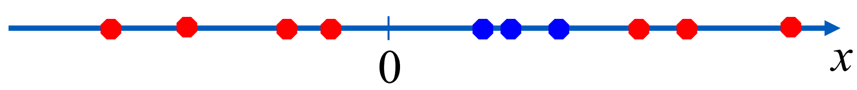
SAIR
Spatial AI & Robotics Lab

Nonlinear SVMs

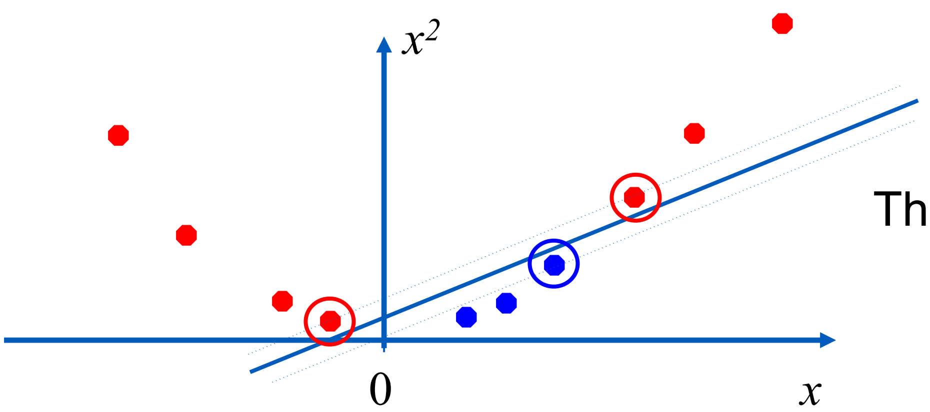
- Datasets that are linearly separable work out great:



- But what if the dataset is NOT linearly separable?



- We can map it to a higher-dimensional space:

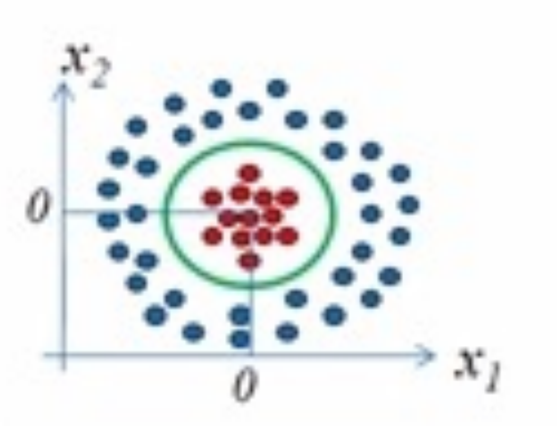


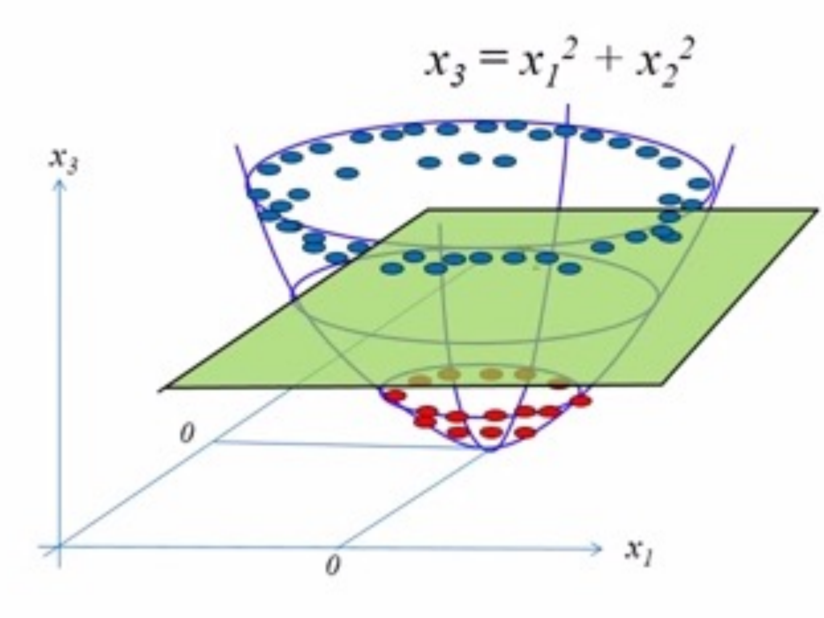
The new feature: $[x, x^2]$

Andrew Moore

Nonlinear SVMs

- Design new features with any higher dimension.
- Nonlinear classification becomes a linear problem!





The new feature: $\boldsymbol{x} = [x_1,\ x_2,\ x_1^2 + x_2^2]$

Nonlinear SVMs with Kernel Trick

- Higher feature dimension means more computation.
- Kernel trick is to bypass the high dimension data.
- Recall that during training and testing we only want the **inner product (scalar) of training samples, instead of explicit high dimension features**.
- Assume nonlinear mapping $\psi(x): \mathbb{R}^M \rightarrow \mathbb{R}^n$, $M \ll N$
- Can define a kernel function to directly calculate inner products

$$
\begin{tikzcd}
 & \psi(x), \psi(x') \arrow[dr] & \\
x, x' \arrow[ur] \arrow[rr, "k(x, x')"] & & \langle \psi(x), \psi(x') \rangle
\end{tikzcd}
$$

Kernel functions

- Widely-used kernel functions

Polynomial: $k(x_m, x_n) = (x_m^T x_n + r)^d$

Linear: $k(x_m, x_n) = x_m^T x_n$

Gaussian: $k(x_m, x_n) = e^{-\frac{||x_m - x_n||^2}{2\sigma^2}}$

Laplace: $k(x_m, x_n) = e^{-\alpha||x_m - x_n||}$

Kernel Trick

- Gaussian Kernel

Let the original space $\mathcal{X} = \mathbb{R}$ and consider the mapping ψ given by

$$\psi(x) = \left\{ \frac{1}{\sqrt{m!}} e^{-\frac{x^2}{2}} x^m \right\}_{m=0}^{\infty}.$$

- Then we can prove

$$\begin{aligned}\langle \psi(x), \psi(x') \rangle &= \sum_{m=0}^{\infty} \left(\frac{1}{\sqrt{m!}} e^{-\frac{x^2}{2}} x^m \right) \left(\frac{1}{\sqrt{m!}} e^{-\frac{(x')^2}{2}} (x')^m \right) \\ &= e^{-\frac{x^2+(x')^2}{2}} \sum_{m=0}^{\infty} \left(\frac{(xx')^m}{m!} \right) \\ &= e^{-\frac{(x-x')^2}{2}}.\end{aligned}$$

- So we have $k(x, x') = e^{-\frac{(x-x')^2}{2}}$

SAIR
Spatial AI & Robotics Lab

Kernel Matrix

- For training and testing, we only need to calculate the kernel matrix instead of the explicit infinite high dimension features.
- Use the values when necessary, during training.

$$\boldsymbol{\phi}\boldsymbol{\phi}^T = K = \begin{bmatrix} \phi(x_1)^T\phi(x_1) & \phi(x_1)^T\phi(x_2) & . & \phi(x_1)^T\phi(x_n) \\ \phi(x_2)^T\phi(x_1) & \phi(x_2)^T\phi(x_2) & . & . \\ . & . & . & . \\ \phi(x_m)^T\phi(x_1) & \phi(x_m)^T\phi(x_2) & . & \phi(x_m)^T\phi(x_n) \end{bmatrix}$$

$$= \begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & . & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & . & . \\ . & . & . & . \\ k(x_m,x_1) & k(x_m,x_2) & . & k(x_m,x_n) \end{bmatrix}$$

Dive into nonlinear kernel SVM

- Let the weights w being $\mathbf{w} = \sum_{i=0}^{N-1} a_i \phi(\mathbf{x}_i)$
- Then inner products $f(x)$ and $||w||^2$ become

$$f(x) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle = \sum_{j=0}^{N-1} a_j \langle \phi(\mathbf{x}_j), \phi(\mathbf{x}) \rangle = \sum_{i=0}^{N-1} a_i \kappa(\mathbf{x}_i, \mathbf{x}).$$

$$\|\mathbf{w}\|^2 = \left\langle \sum_{j=0}^{N-1} a_j \phi(\mathbf{x}_j), \sum_{j=0}^{N-1} a_j \phi(\mathbf{x}_j) \right\rangle = \sum_{i,j=0}^{N-1} a_i a_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle .$$

- Then the loss function in page 38 becomes

$$\min_{\mathbf{a} \in \mathbb{R}^N} \left(\lambda \mathbf{a}^T \mathbf{K} \mathbf{a} + \frac{1}{N} \sum_{i=0}^{N-1} \max\{0, 1 - y_i (\mathbf{K}\mathbf{a})_i\} \right),$$

i-th element of the vector Ka.

- Then solution is like linear SVM!

Page 14-16 in C. Wang, Kernel learning for visual perception.

Summary: SVMs for image classification

1. Pick an image representation
   - In our case, **bag of features (next lecture)**
2. Pick a kernel function for that representation
3. Compute the kernel matrix.
4. Feed the kernel matrix into the SVM solver, e.g., SGD, to obtain support vectors and weights.
5. At test time: compute kernel values for your test example and each support vector, and combine them with the learned weights to get the value of the decision function.

Slide credit: L. Lazebnik

What about multi-class SVMs?

- Unfortunately, there is no “definitive” multi-class SVM formulation
- In practice, we have to obtain a multi-class SVM by combining multiple two-class SVMs
- One vs. others
  - Training: learn an SVM for each class vs. the others
  - Testing: apply each SVM to test example and assign to it the class of the SVM returning highest decision value
- One vs. one
  - Training: learn an SVM for each pair of classes
  - Testing: each learned SVM “votes” for a class to assign to the test example

SVMs: Pros and Cons

- Pros
  - Many publicly available SVM packages: http://www.kernel-machines.org/software
  - Kernel-based framework is very powerful, flexible
  - SVMs work very well in practice, even with very small training sample sizes

- Cons
  - No “direct” multi-class SVM, combine two-class SVMs
  - Computation, memory
    - During training time, must compute the big kernel matrix.
    - Learning can take a very long time for large-scale problems

What to remember about classifiers

- No free lunch: machine learning algorithms are tools, not dogmas

- Try SIMPLE classifiers first

- Better to have smart features and simple classifiers than simple features and smart classifiers

- Use increasingly powerful classifiers with more training data (bias-variance tradeoff)

Machine Learning Considerations

- 3 important design decisions:
  - *** What data do I use?
  - ** How do I represent my data (what feature)?
  - * What classifier / regressor / models do I use?
- These are in decreasing order of importance
- Deep learning addresses 2 and 3 simultaneously (and blurs the boundary between them).
- You can take the features/representation from deep learning and use it with any classifier.

Machine Learning Considerations

- How to overcome overfitting?
  - Simpler Model
  - Data augmentation
  - Early stopping
  - Regularization
  - Ensembling
  - …