

# Machine Learning Crash Course

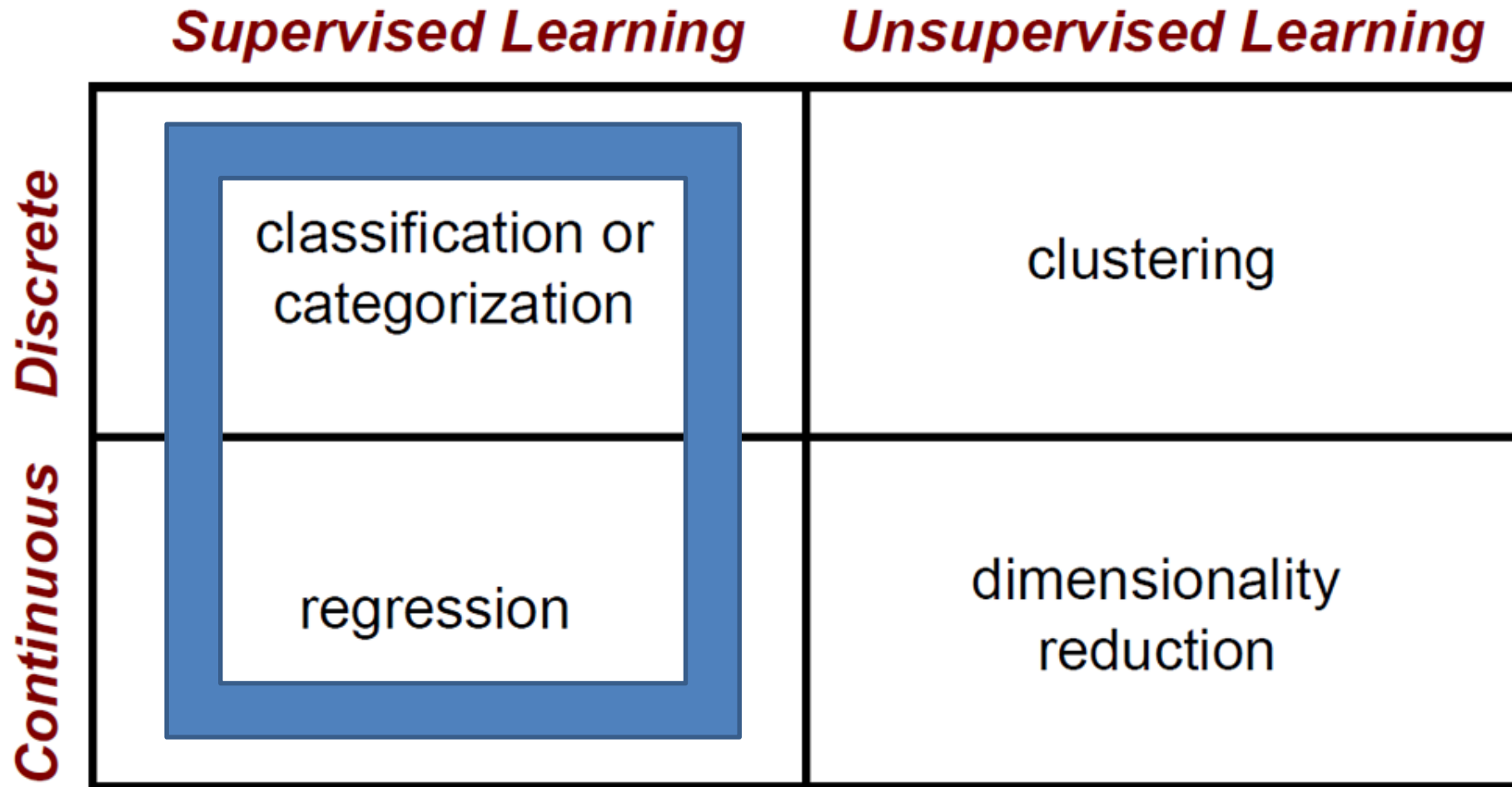


Photo: CMU Machine Learning  
Department protests G20

Computer Vision  
James Hays

Slides: Isabelle Guyon,  
Erik Sudderth,  
Mark Johnson,  
Derek Hoiem

# Machine Learning Problems



# The machine learning framework

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

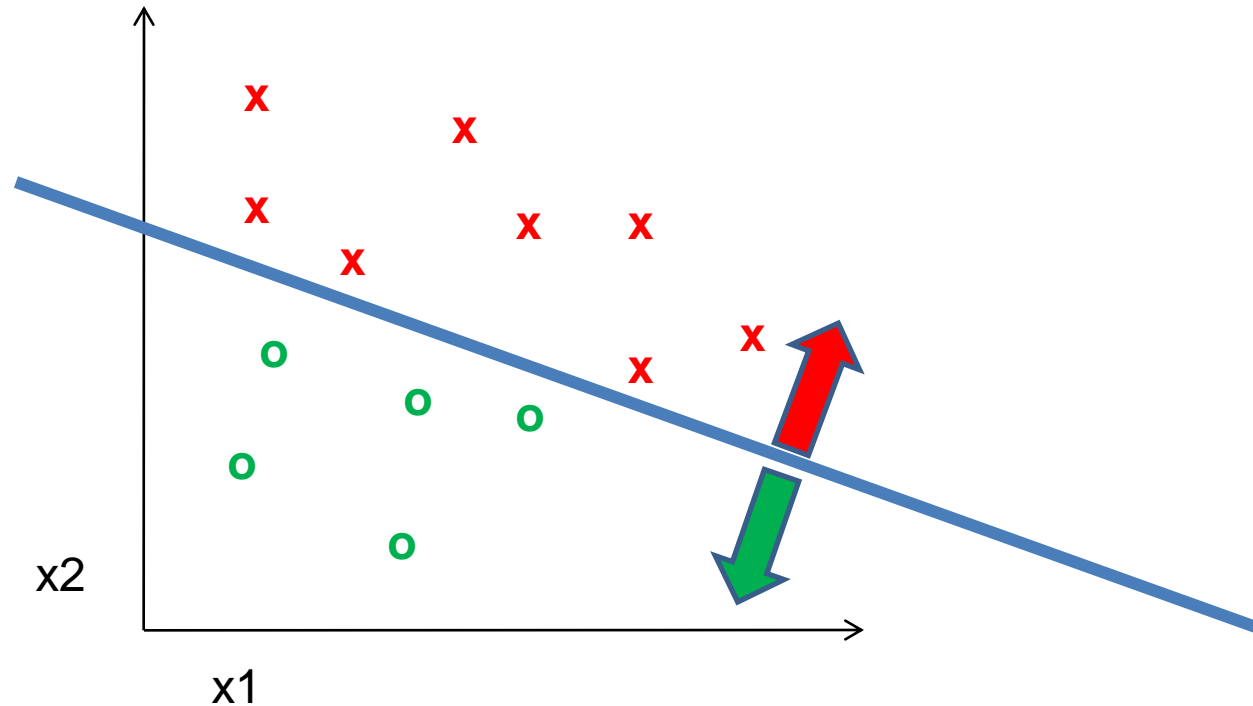
$f(\text{apple image}) = \text{"apple"}$

$f(\text{tomato image}) = \text{"tomato"}$

$f(\text{cow image}) = \text{"cow"}$

# Learning a classifier

Given some set of features with corresponding labels, learn a function to predict the labels from the features



# Generalization



Training set (labels known)



Test set (labels unknown)

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

# Very brief tour of some classifiers

- **K-nearest neighbor**
- **SVM**
- Boosted Decision Trees
- Neural networks
- Naïve Bayes
- Bayesian network
- Logistic regression
- Randomized Forests
- RBMs
- Etc.

# Generative vs. Discriminative Classifiers

## Generative Models

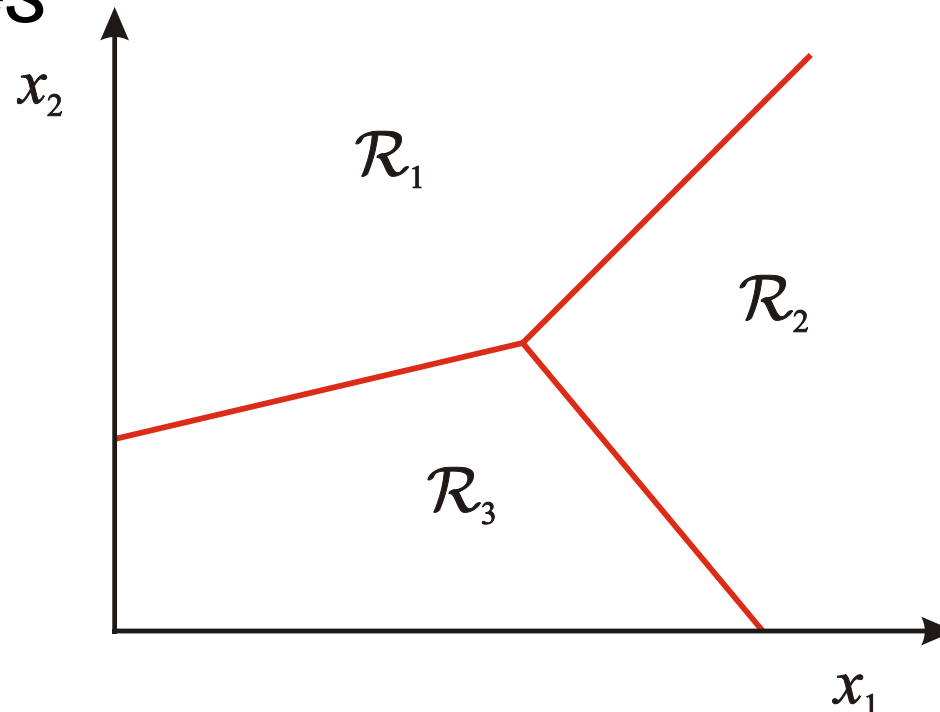
- Represent both the data and the labels
- Often, makes use of conditional independence and priors
- Examples
  - Naïve Bayes classifier
  - Bayesian network
- Models of data may apply to future prediction problems

## Discriminative Models

- Learn to directly predict the labels from the data
- Often, assume a simple boundary (e.g., linear)
- Examples
  - Logistic regression
  - SVM
  - Boosted decision trees
- Often easier to predict a label from the data than to model the data

# Classification

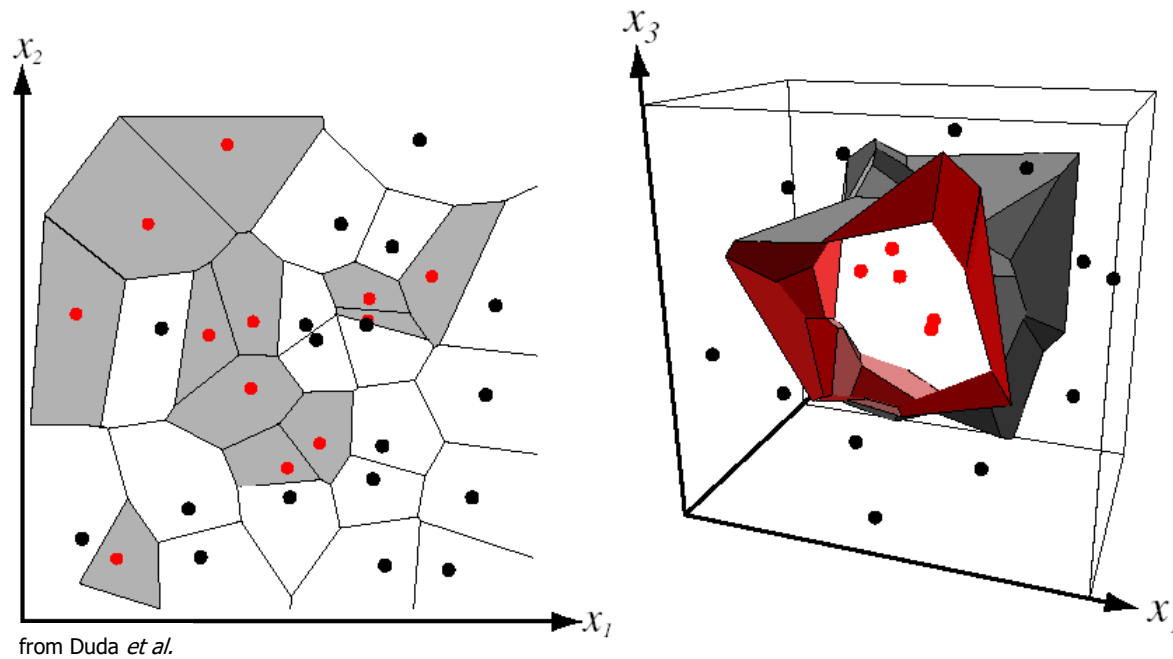
- Assign input vector to one of two or more classes
- Any decision rule divides input space into *decision regions* separated by *decision boundaries*





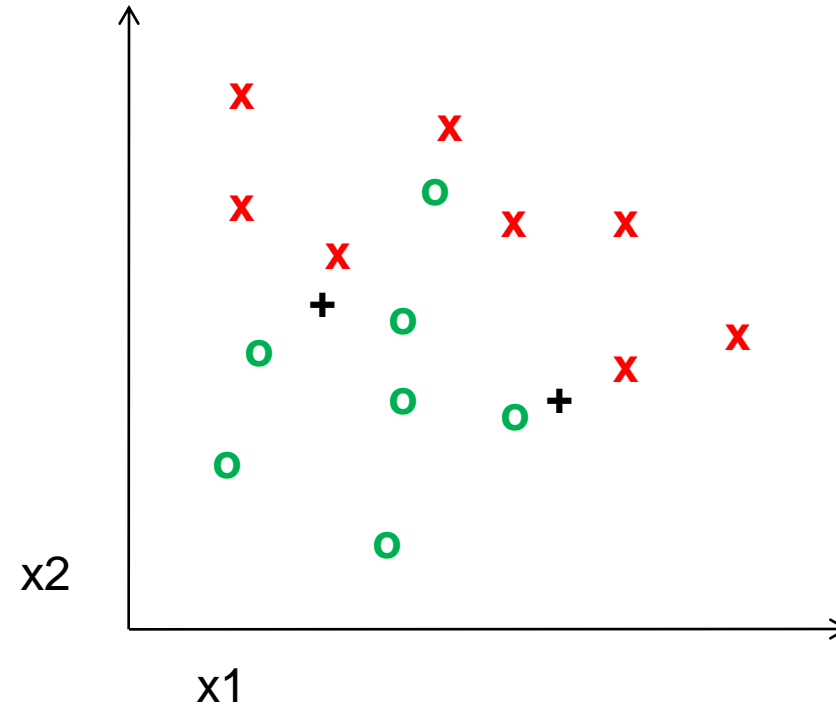
# Nearest Neighbor Classifier

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



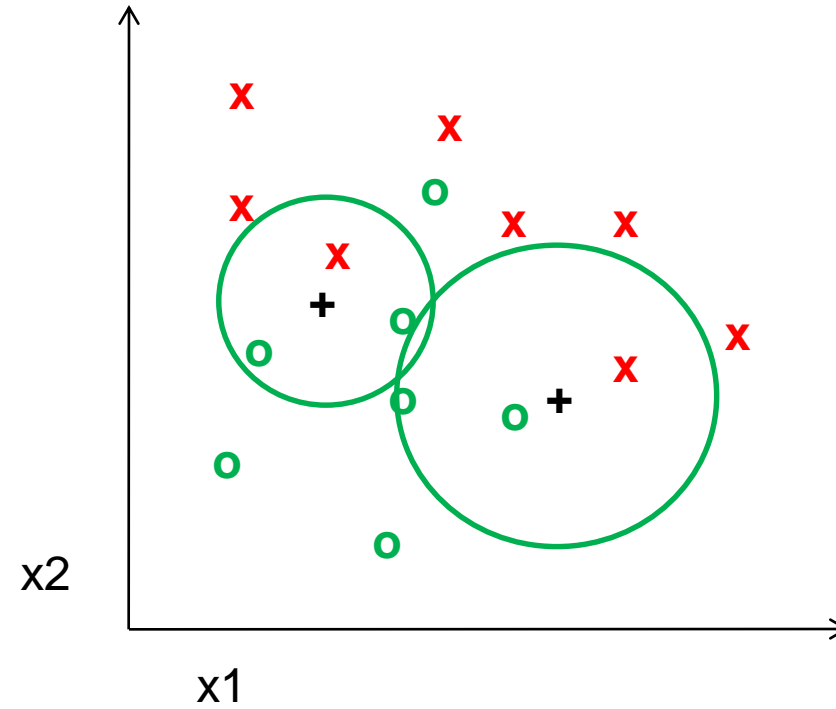
Voronoi partitioning of feature space  
for two-category 2D and 3D data

# K-nearest neighbor

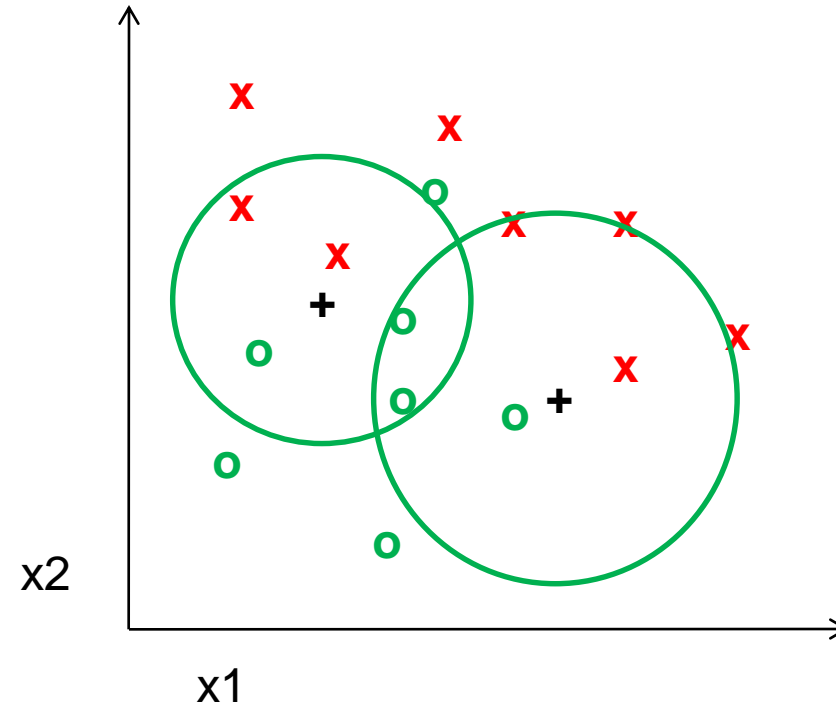




# 3-nearest neighbor



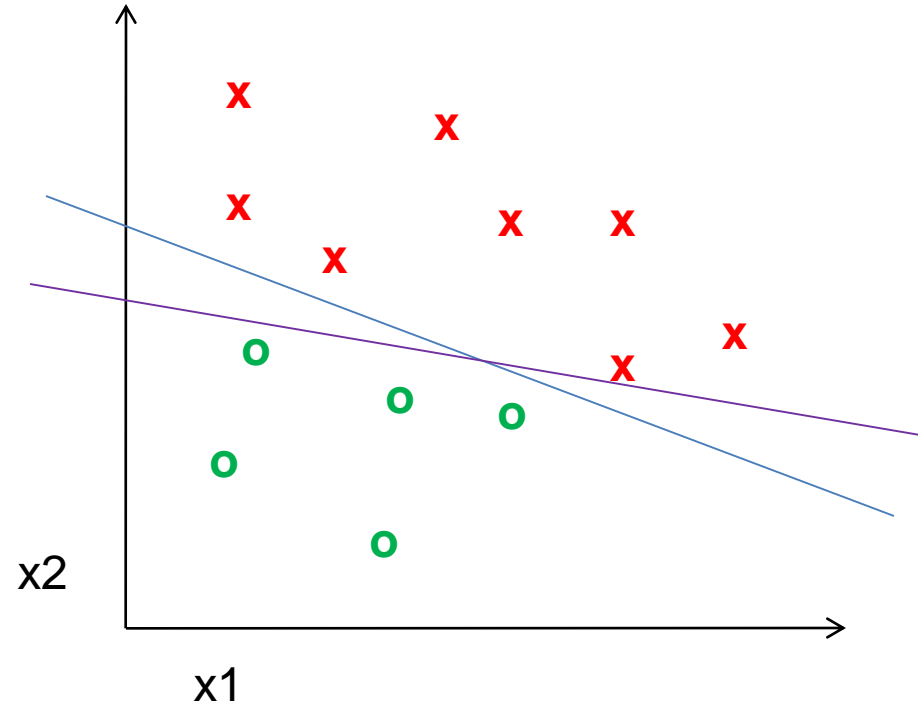
# 5-nearest neighbor



# 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

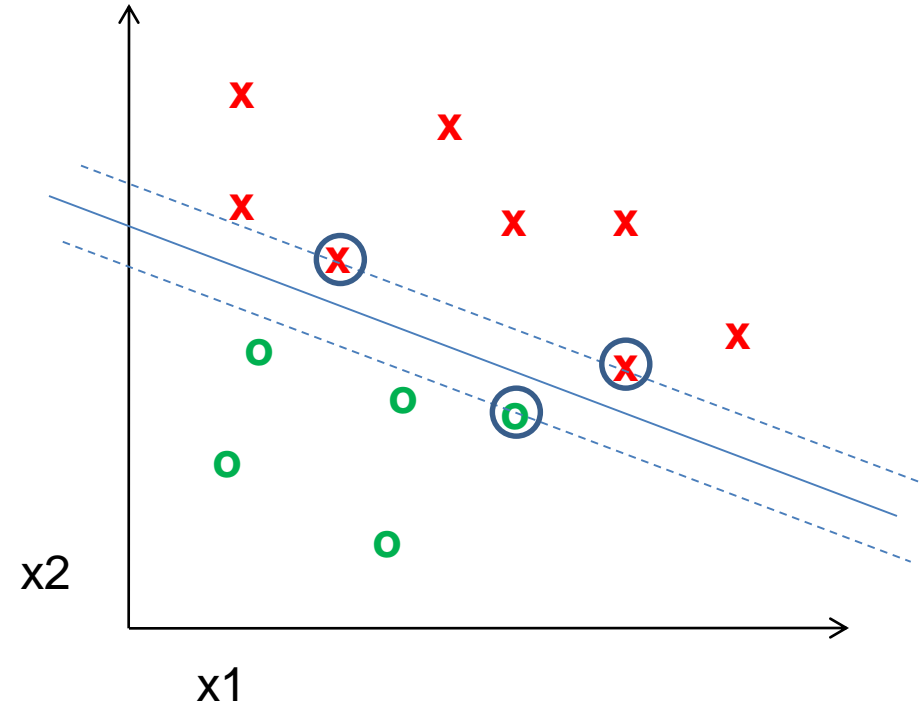
# Classifiers: Linear SVM



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

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

# Classifiers: Linear SVM

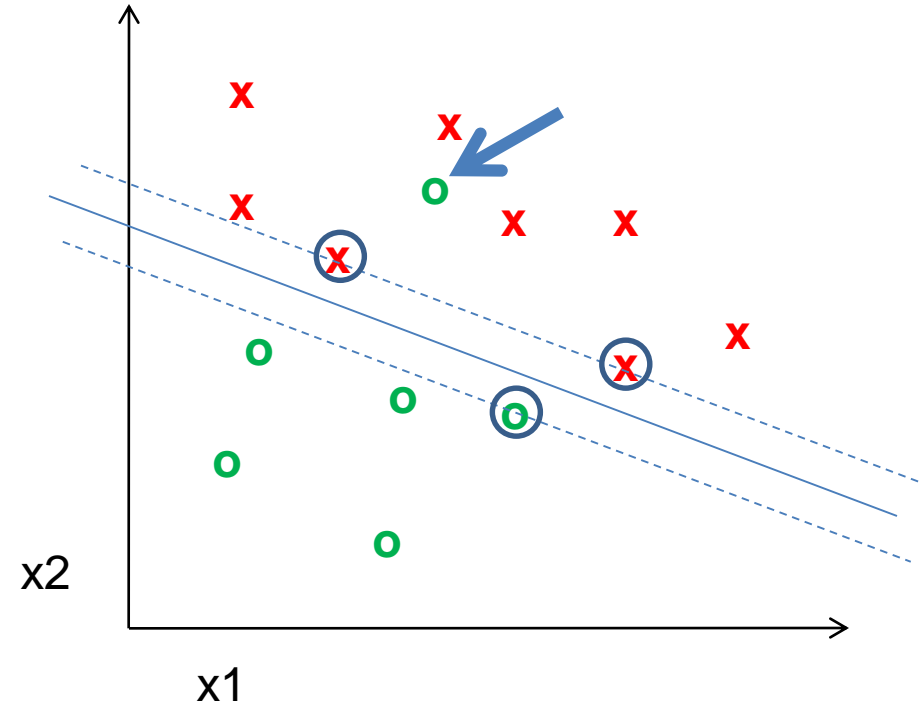


- Find a *linear function* to separate the classes:

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



# Classifiers: Linear SVM



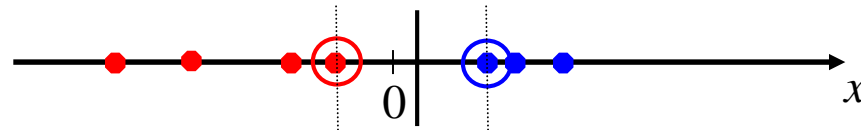
- Find a *linear function* to separate the classes:

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

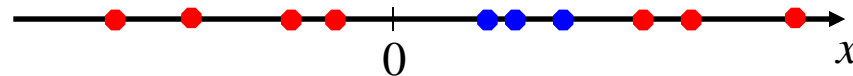
# Nonlinear SVMs

---

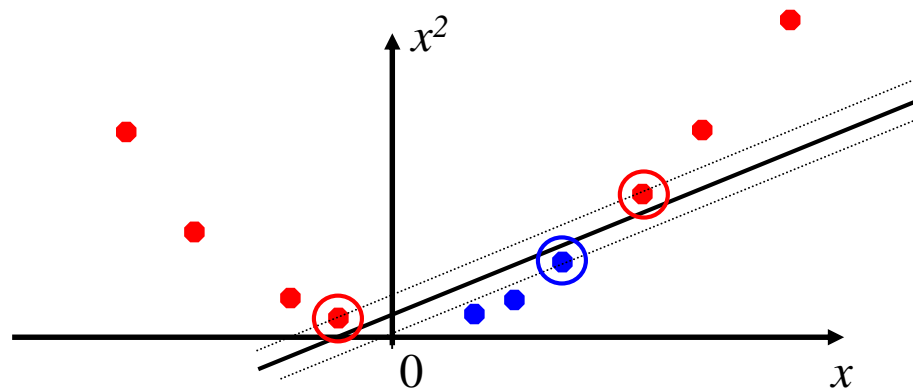
- Datasets that are linearly separable work out great:



- But what if the dataset is just too hard?



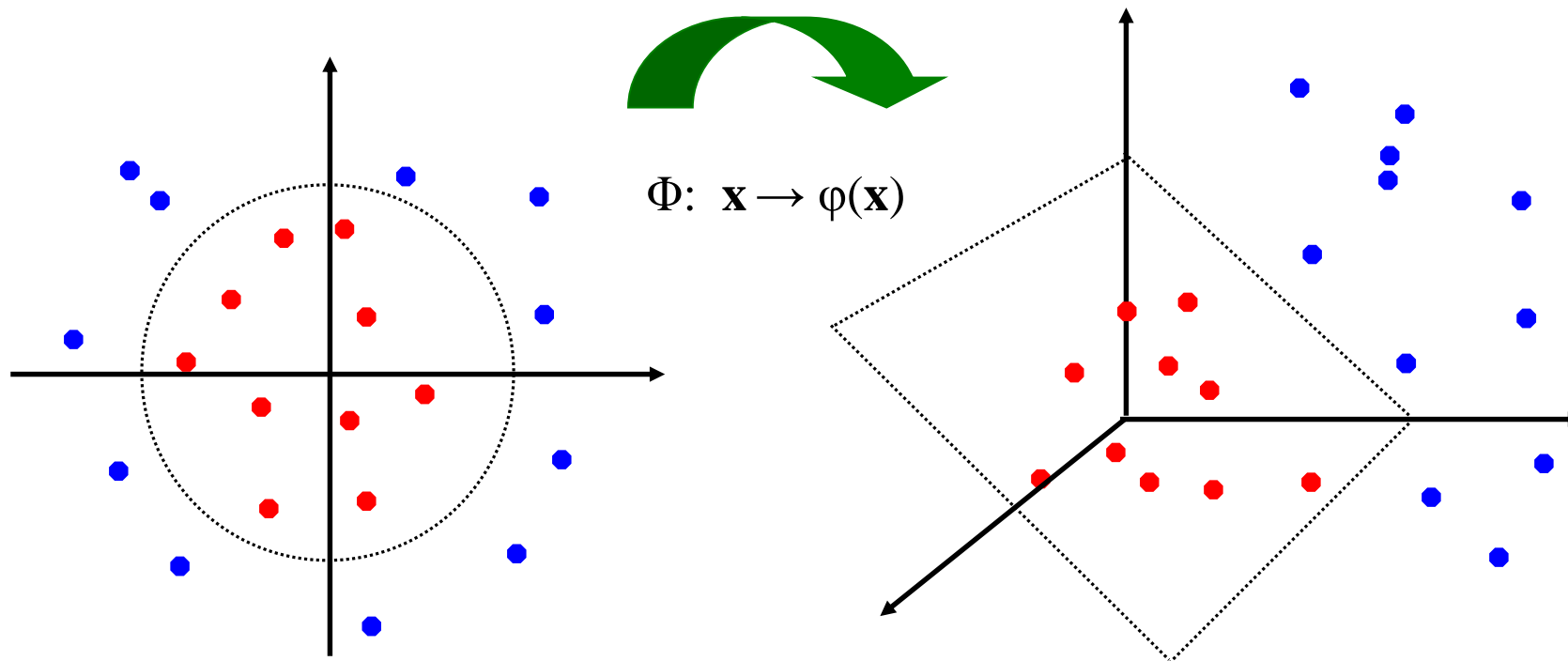
- We can map it to a higher-dimensional space:



# Nonlinear SVMs

---

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



# Nonlinear SVMs

---

- *The kernel trick*: instead of explicitly computing the lifting transformation  $\varphi(\mathbf{x})$ , define a kernel function  $K$  such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)$$

(to be valid, the kernel function must satisfy *Mercer's condition*)

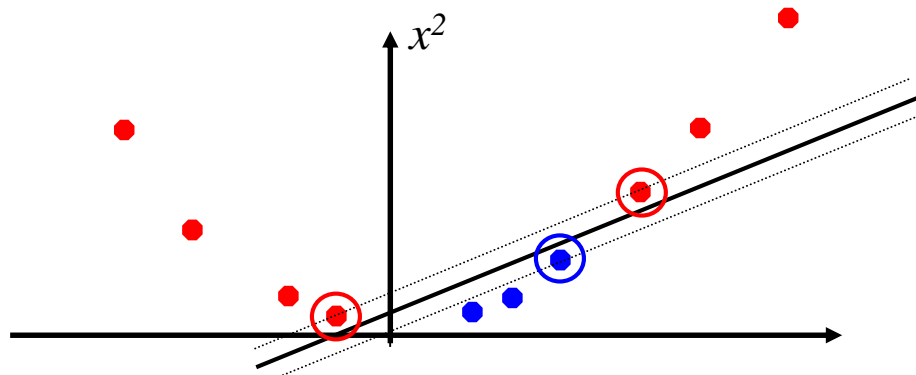
- This gives a nonlinear decision boundary in the original feature space:

$$\sum_i \alpha_i y_i \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}) + b = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

# Nonlinear kernel: Example

---

- Consider the mapping  $\varphi(x) = (x, x^2)$



$$\varphi(x) \cdot \varphi(y) = (x, x^2) \cdot (y, y^2) = xy + x^2 y^2$$

$$K(x, y) = xy + x^2 y^2$$

# Kernels for bags of features

---

- Histogram intersection kernel:

$$I(h_1, h_2) = \sum_{i=1}^N \min(h_1(i), h_2(i))$$

- Generalized Gaussian kernel:

$$K(h_1, h_2) = \exp\left(-\frac{1}{A} D(h_1, h_2)^2\right)$$

- $D$  can be (inverse) L1 distance, Euclidean distance,  $\chi^2$  distance, etc.

# Summary: SVMs for image classification

---

1. Pick an image representation (e.g. histogram of quantized sift features)
2. Pick a kernel function for that representation
3. Compute the matrix of kernel values between every pair of training examples
4. Feed the kernel matrix into your favorite SVM solver 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

# 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 that returns the 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, must combine two-class SVMs
- Computation, memory
  - During training time, must compute matrix of kernel values for every pair of examples
  - Learning can take a very long time for large-scale problems

# Very brief tour of some classifiers

- **K-nearest neighbor**
- **SVM**
- Boosted Decision Trees
- Neural networks
- Naïve Bayes
- Bayesian network
- Logistic regression
- Randomized Forests
- RBMs
- Etc.

# Generalization



Training set (labels known)



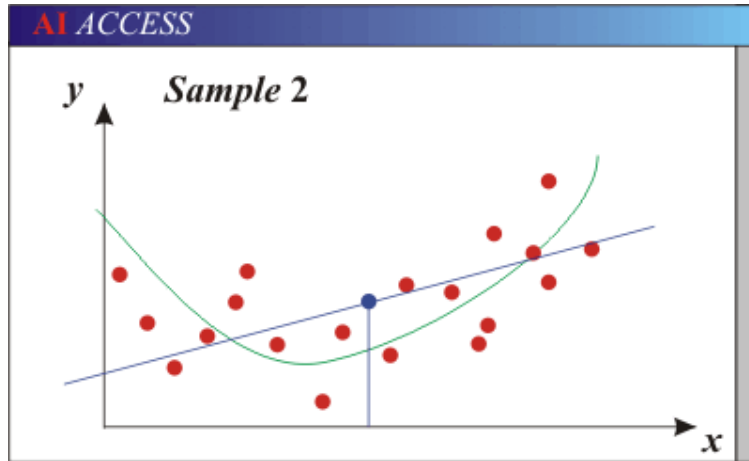
Test set (labels unknown)

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

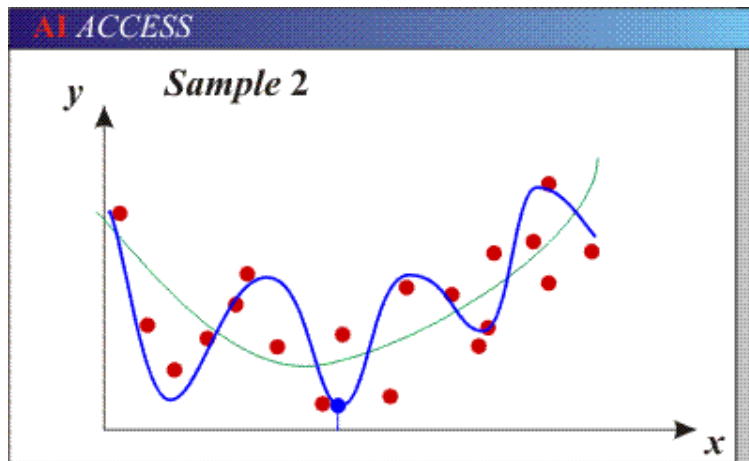
# Generalization

- Components of generalization error
  - **Bias:** how much the average model over all training sets differ from the true model?
    - Error due to inaccurate assumptions/simplifications made by the model. “Bias” sounds negative. “Regularization” sounds nicer.
  - **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 Trade-off

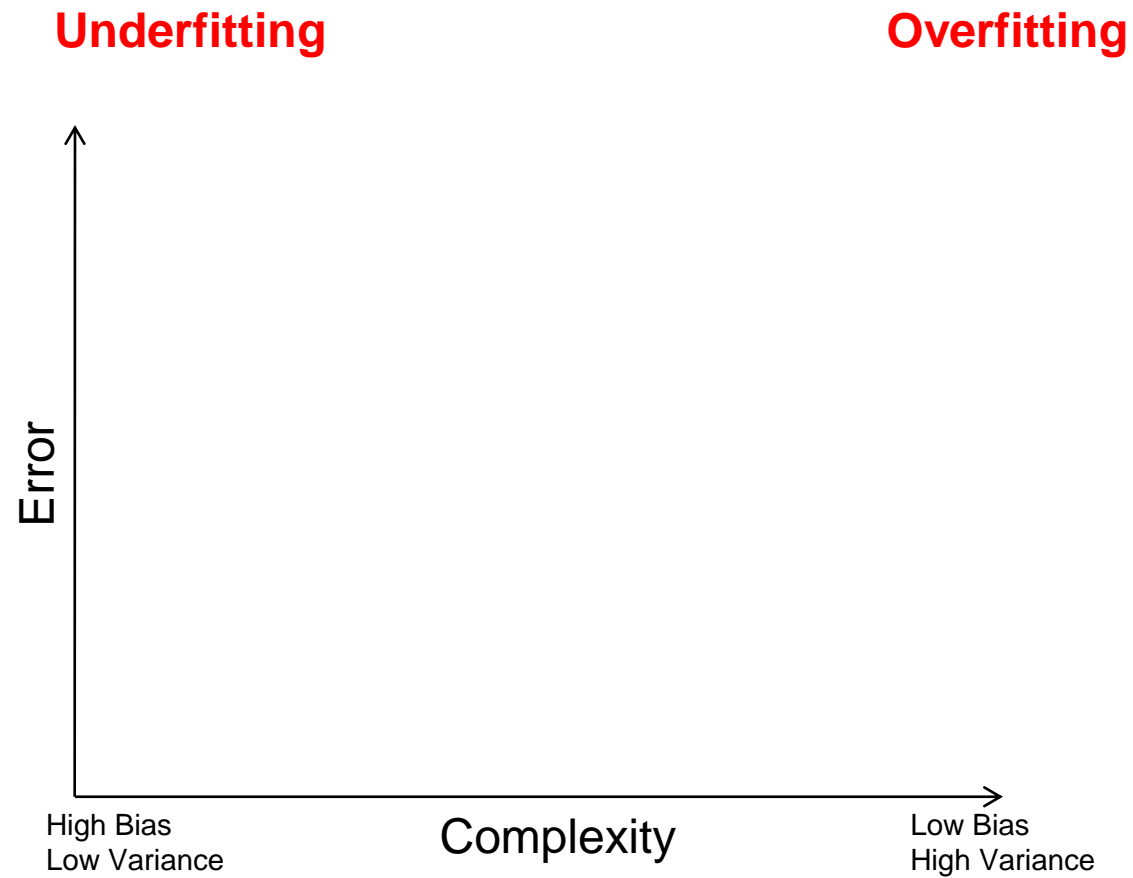


- Models with too few parameters are inaccurate because of a large bias (not enough flexibility).

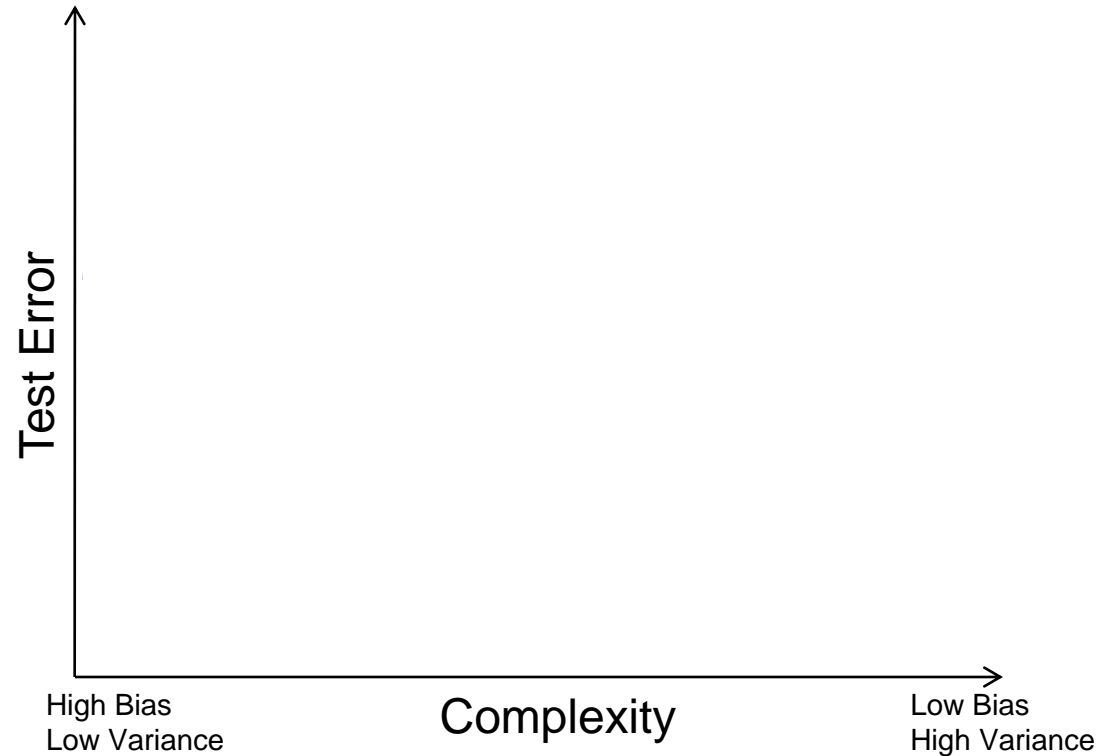


- Models with too many parameters are inaccurate because of a large variance (too much sensitivity to the sample).

# Bias-variance tradeoff

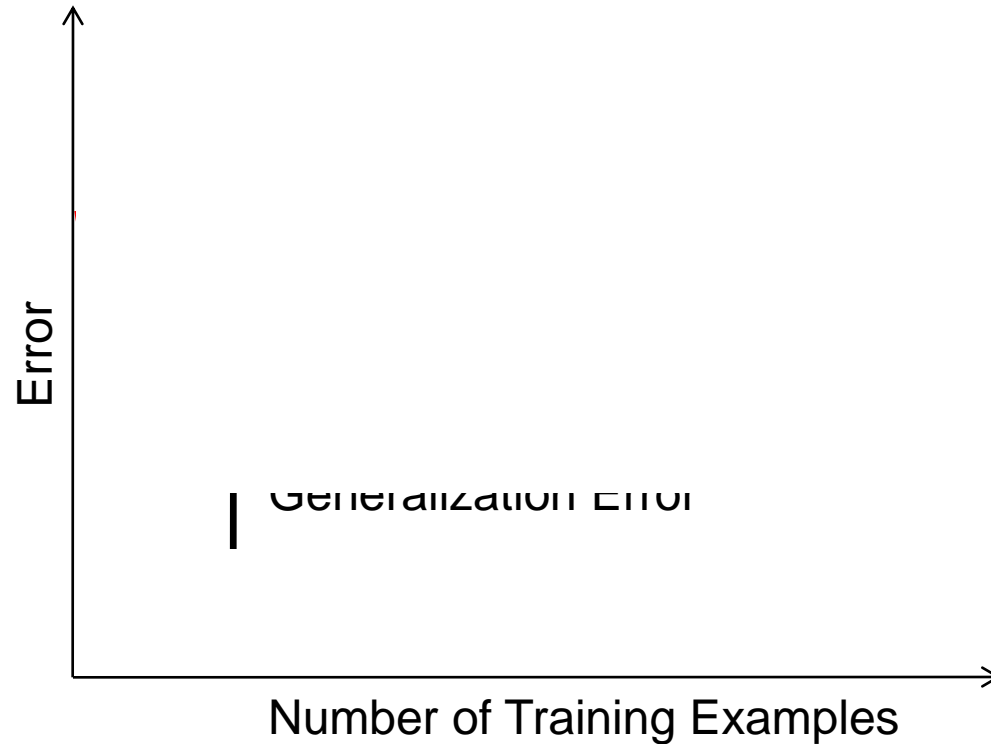


# Bias-variance tradeoff



# Effect of Training Size

Fixed prediction model





# 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 over-simplifications
  - Variance: due to inability to perfectly estimate parameters from limited data



- How to reduce variance?
  - Choose a simpler classifier
  - Regularize the parameters
  - Get more training data
- How to reduce bias?
  - Choose a more complex, more expressive classifier
  - Remove regularization
  - (These might not be safe to do unless you get more training data)

# 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:
  - 1) What data do I use?
  - 2) How do I represent my data (what feature)?
  - 3) What classifier / regressor / machine learning tool 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 representation from deep learning and use it with any classifier.

# Machine Learning Problems

	<i>Supervised Learning</i>	<i>Unsupervised Learning</i>
<i>Discrete</i>	classification or categorization	clustering
<i>Continuous</i>	regression	dimensionality reduction

- Andrew Ng's ranking of machine learning impact
  1. Supervised Learning
  2. Transfer Learning
  3. Unsupervised Learning\* (Often "self-supervised" learning)
  4. Reinforcement Learning

James thinks 2 and 3 might have switched ranks.

