cs221.stanford.edu/q

Question

What's the true objective of machine learning?

- minimize error on the training set
- minimize training error with regularization
- minimize error on unseen future examples
- learn about machines

Training error

Loss minimization:

$$\min_{\mathbf{w}} \mathsf{TrainLoss}(\mathbf{w})$$

$$\mathsf{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\mathsf{train}}|} \sum_{(x,y)\in\mathcal{D}_{\mathsf{train}}} \mathsf{Loss}(x, y, \mathbf{w})$$

Is this a good objective?

A strawman algorithm

Algorithm: rote learning

Training: just store $\mathcal{D}_{\text{train}}$.

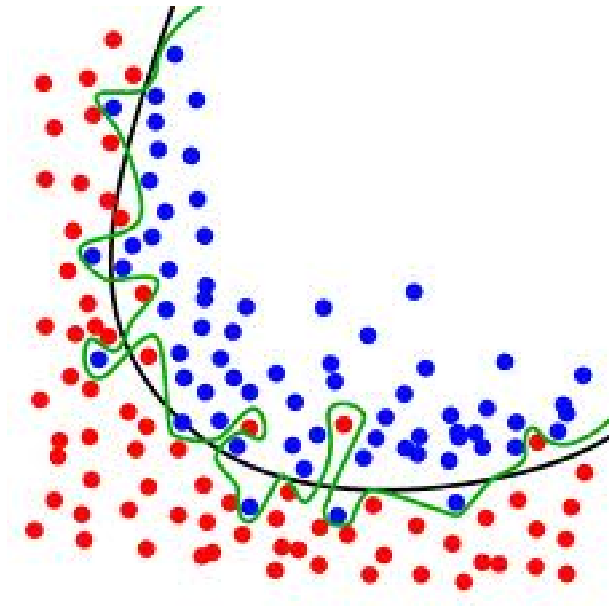
Predictor $f(x)$:

If $(x, y) \in \mathcal{D}_{\text{train}}$: return y.

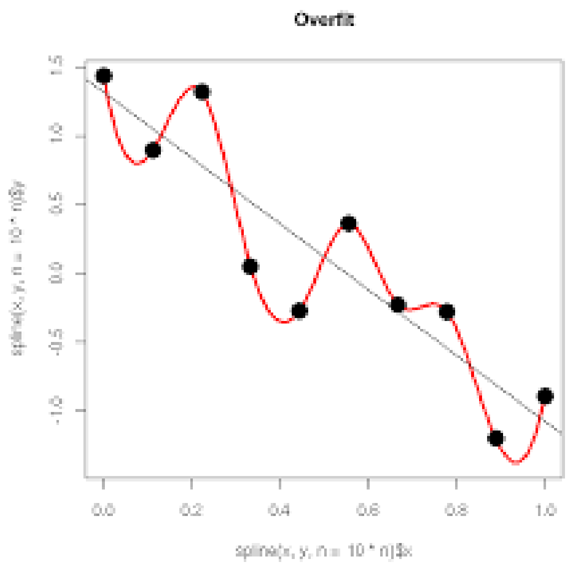
Else: **segfault**.

Minimizes the objective perfectly (zero), but clearly bad...

Overfitting pictures



Classification



Regression

Evaluation

$$\mathcal{D}_{\text{train}} \longrightarrow \boxed{\text{Learner}} \longrightarrow f$$

How good is the predictor f?

Key idea: the real learning objective

Our goal is to minimize **error on unseen future examples**.

Don't have unseen examples; next best thing:

Definition: test set

Test set $\mathcal{D}_{\text{test}}$ contains examples not used for training.

Overfitting example

Example: overfitting

Input: $x \in \{-4, -3, -2, -1, 1, 2, 3, 4\}$

Output: $y = \text{sign}(x)$, but 25% labels flipped

x	-4	-3	-2	-1	1	2	3	4
y (train)	-	+	-	-	+	+	+	-
y (test)	+	-	-	-	+	-	+	+
rote predictions	-	+	-	-	+	+	+	-
linear predictions	-	-	-	-	+	+	+	+

	Train error	Test error	
Rote	0%	50%	— overfits!
Linear	25%	25%	— generalizes!

Another strawman algorithm

Algorithm: majority algorithm

Training: find most frequent output y in $\mathcal{D}_{\text{train}}$.

Predictor $f(x)$: return y.

On the previous example:

x	-4	-3	-2	-1	1	2	3	4
y_{train}	-	+	-	-	+	+	+	-

	Train error	Test error	
Rote	0%	50%	— overfits!
Linear	25%	25%	— generalizes!
Majority	50%	50%	— generalizes!*

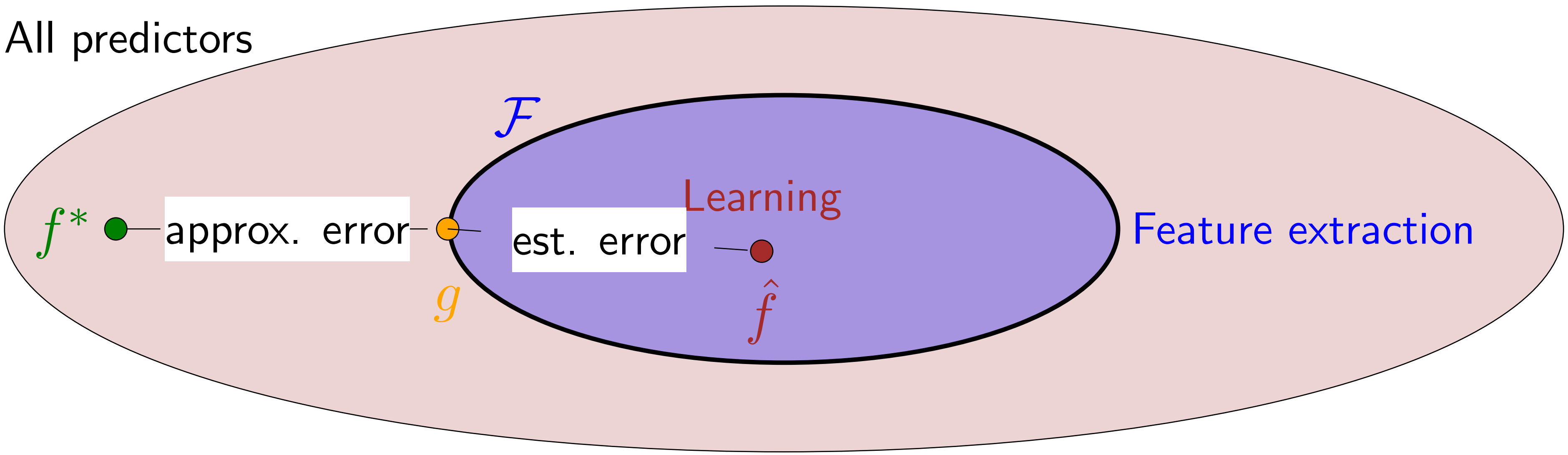
*though the error is high

Generalization

When will a learning algorithm **generalize** well?

$$\boxed{\mathcal{D}_{\text{train}}} \longrightarrow \boxed{\mathcal{D}_{\text{test}}}$$

Approximation and estimation error



- Approximation error: how good is the hypothesis class?
- Estimation Error: how good is the learned predictor **relative to** the hypothesis class?

$$\underbrace{\mathbf{Err}(\hat{f}) - \mathbf{Err}(g)}_{\textbf{estimation}} + \underbrace{\mathbf{Err}(g) - \mathbf{Err}(f^*)}_{\textbf{approximation}}$$

Effect of hypothesis class size

As the hypothesis class size increases...

Approximation error decreases because:

taking min over larger set

Estimation error increases because:

statistical learning theory

Estimation error analogy

Scenario 1: ask few people around

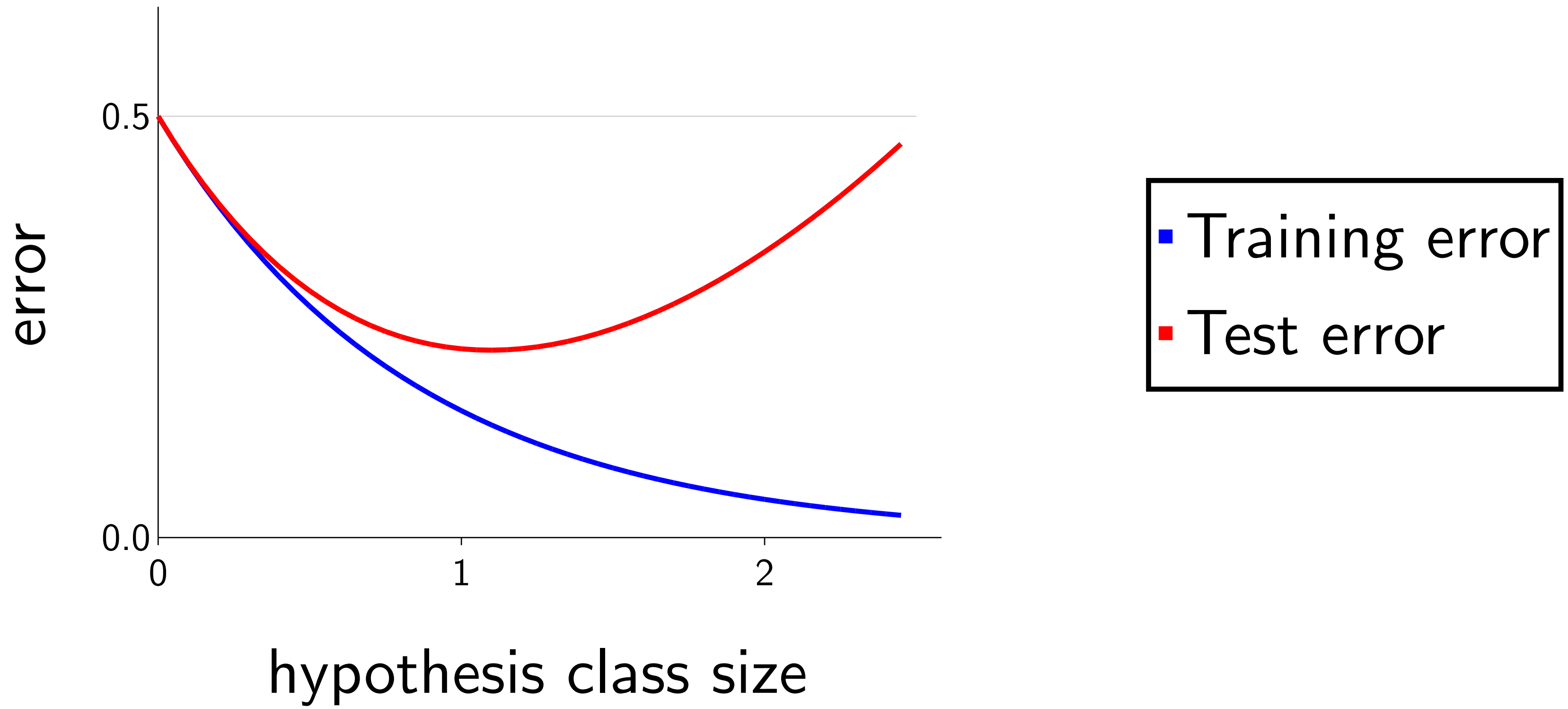
Is your name Joe?

Scenario 2: email all of Stanford

Is your name Joe?

people = hypotheses, questions = examples

Training and test error



Underfitting: majority algorithm

Overfitting: rote learning algorithm

Fitting: reasonable learning algorithms

cs221.stanford.edu/q

Question

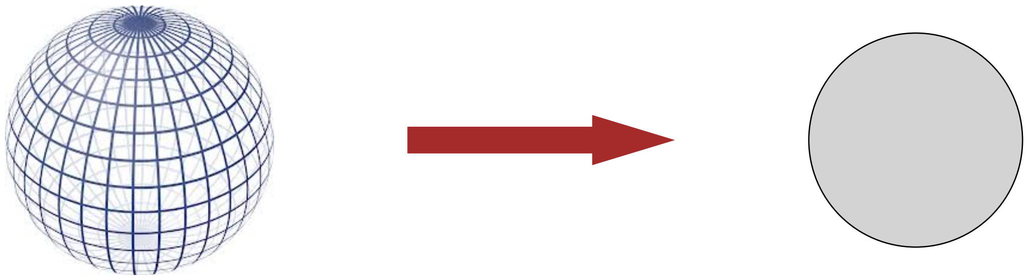
How can you reduce overfitting (select all that apply)?

- remove features
- simplify your features (replace $\cos(x_1)$ with x_1)
- add $0.2\|\mathbf{w}\|^2$ to the objective function
- make sure $\|\mathbf{w}\| \le 1$
- run SGD for fewer iterations

Controlling size of hypothesis class

Linear predictors are specified by weight vector $\mathbf{w} \in \mathbb{R}^d$

Keeping the dimensionality d small:



[whiteboard: linear and quadratic functions]

Keeping the norm (length) $\|\mathbf{w}\|$ small:

[whiteboard: $x \mapsto w_1 x$]

Controlling the dimensionality

Manual feature (template) selection:

- Add features if they help
- Remove features if they don't help

Automatic feature selection (beyond the scope of this class):

- Forward selection
- Boosting
- L_1 regularization

Controlling the norm: regularization

Regularized objective:

$$\min_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \dots, 0]$

For $t = 1, \dots, T$:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta(\nabla_{\mathbf{w}} [\text{TrainLoss}(\mathbf{w})] + \lambda \mathbf{w})$$

Same as gradient descent, except shrink the weights towards zero by λ.

Note: SVM = hinge loss + regularization

Controlling the norm: early stopping

Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$$

Idea: simply make T smaller

Intuition: if have fewer updates, then $\|\mathbf{w}\|$ can't get too big.

Lesson: try to minimize the training error, but don't try too hard.

Summary so far

Key idea: keep it simple

Try to minimize training error, but keep the hypothesis class small.

Hyperparameters

Definition: hyperparameters

Properties of the learning algorithm (features, regularization parameter λ, number of iterations T, step size η, etc.).

How do we choose hyperparameters?

Choose hyperparameters to minimize $\mathcal{D}_{\text{train}}$ error? **No** - solution would be to include all features, set $\lambda = 0$, $T \to \infty$.

Choose hyperparameters to minimize $\mathcal{D}_{\text{test}}$ error? **No** - choosing based on $\mathcal{D}_{\text{test}}$ makes it an unreliable estimate of error!

Validation

Problem: can't use test set!

Solution: randomly take out 10-50% of training and use it instead of the test set to estimate test error.

$\mathcal{D}_{\text{train}} \backslash \mathcal{D}_{\text{val}}$ | $\mathcal{D}_{\text{val}}$ | $\mathcal{D}_{\text{test}}$

Definition: validation set

A **validation (development) set** is taken out of the training data which acts as a surrogate for the **test set**.

Development cycle

Problem: simplified named-entity recognition

Input: a string x (e.g., *President [Barack Obama] in*)

Output: y, whether x contains a person or not (e.g., $+1$)

Algorithm: recipe for success

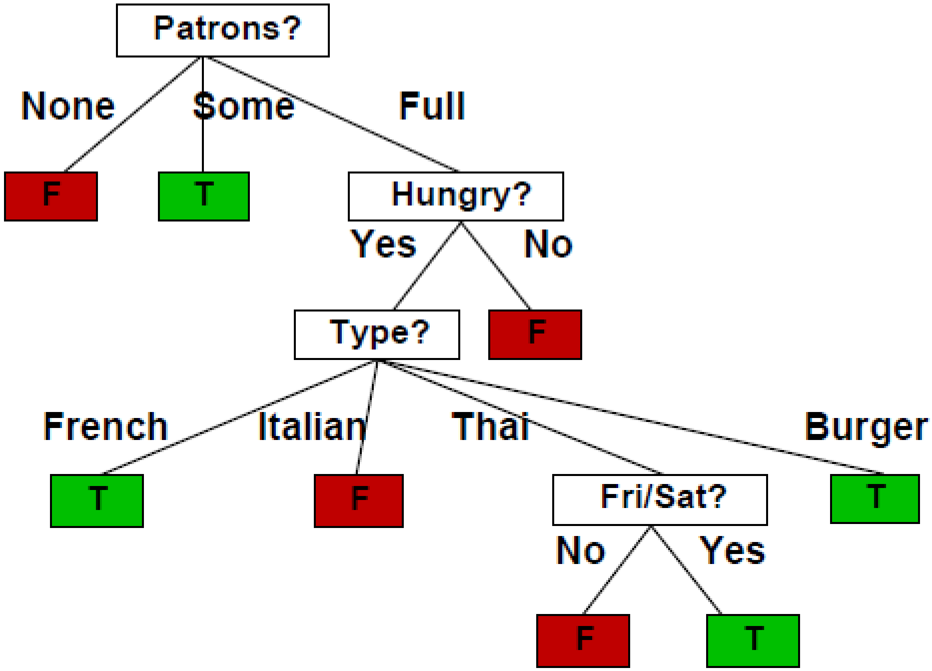
- Split data into train, dev, test
- Look at data to get intuition
- Repeat:
  - Implement feature / tune hyperparameters
  - Run learning algorithm
  - Sanity check train and dev error rates, weights
  - Look at errors to brainstorm improvements
- Run on test set to get final error rates

[semi-live solution]

Learning from Examples: A Review

- Learning principles
- Decision tree
- Linear regression & classification
- Support vector machine
- Neural network & deep learning
- Evaluation (generalization)

Decision Tree



- Choose the attribute that minimizes the remaining information needed

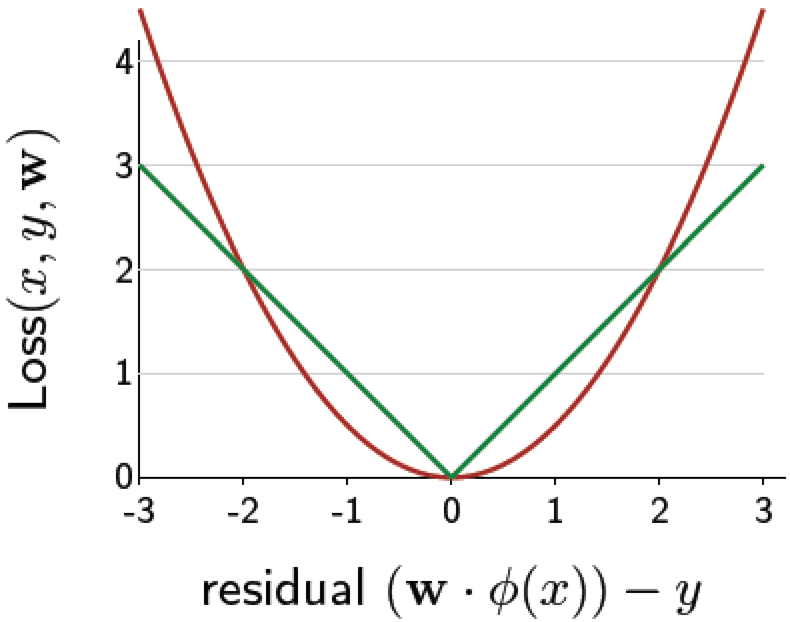
Linear Regression & Classification

$$\underbrace{\mathbf{w} \cdot \phi(x)}_{\text{score}}$$

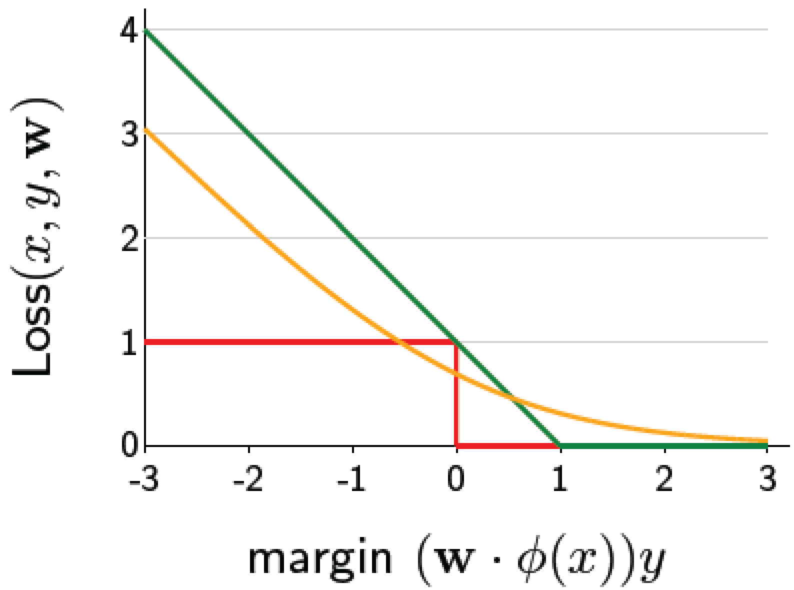
	Classification	Linear regression
Predictor $f_{\mathbf{w}}$	sign(score)	score
Relate to correct y	margin (score y)	residual (score $- y$)
Loss functions	zero-one hinge logistic	squared absolute deviation
Algorithm	SGD	SGD

Linear Regression & Classification

Regression

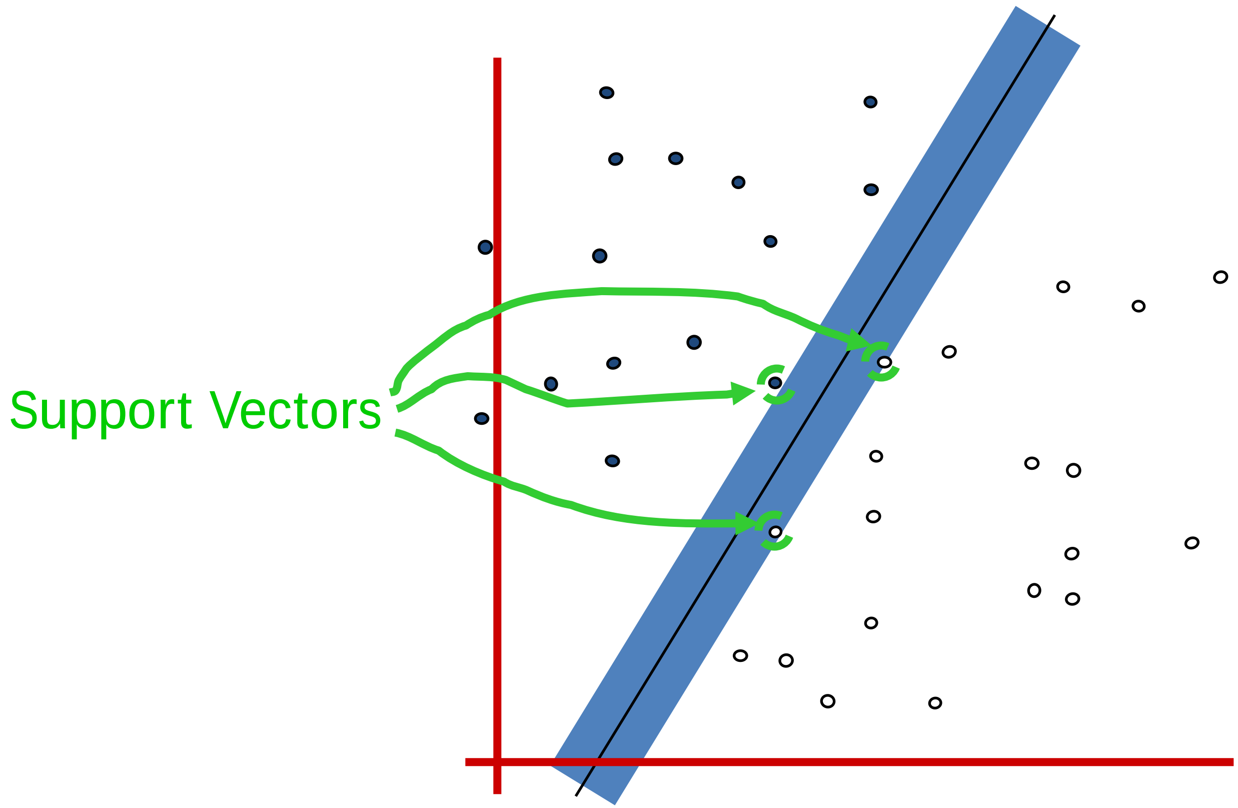


Binary classification



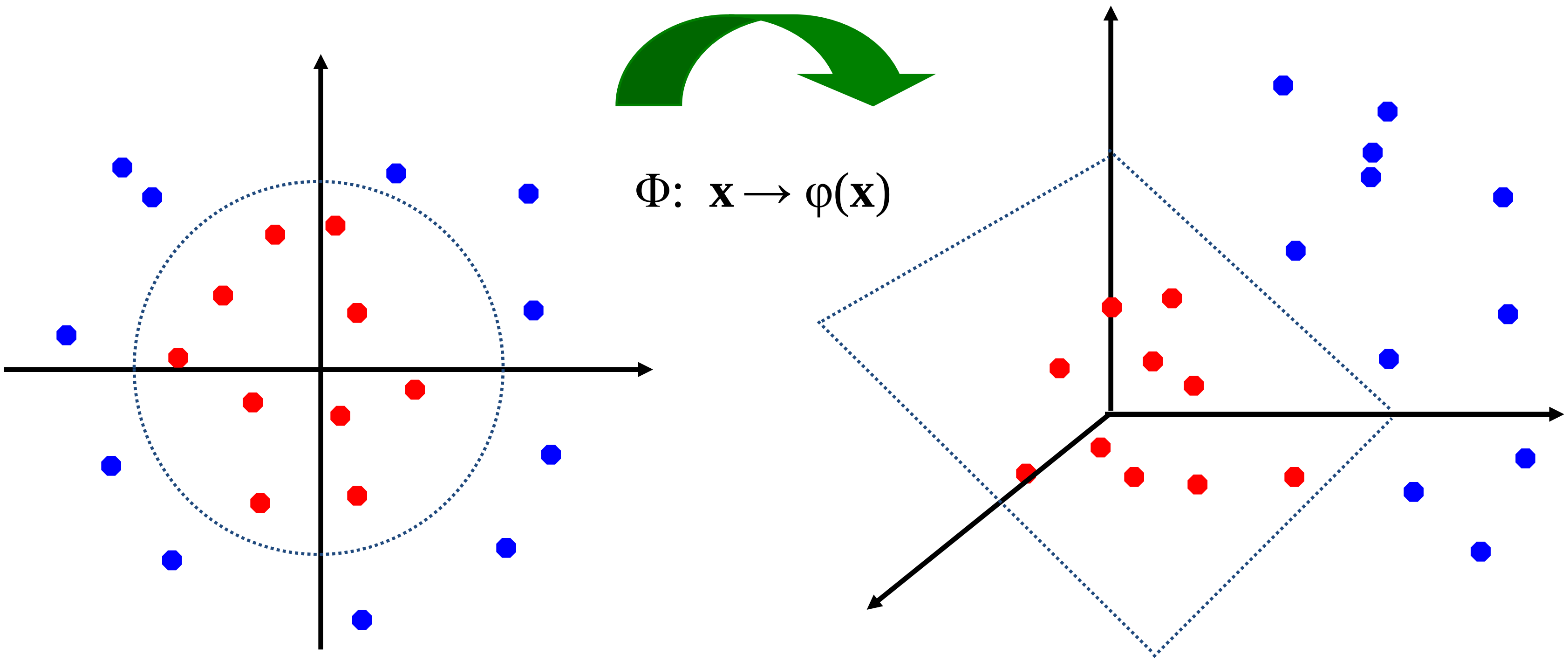
- Minimize the proper loss function

Support Vector Machine



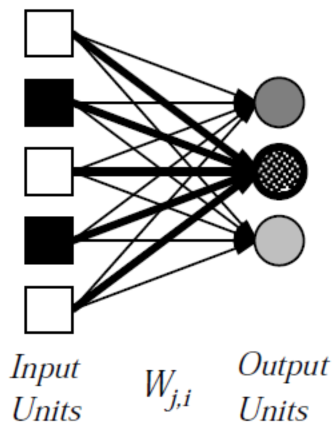
- Maximize the margin: hard or soft

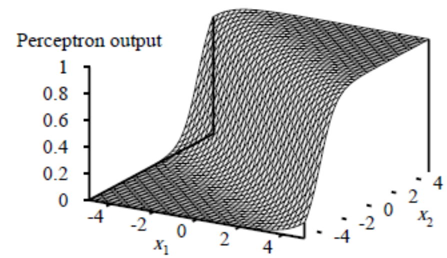
Support Vector Machine

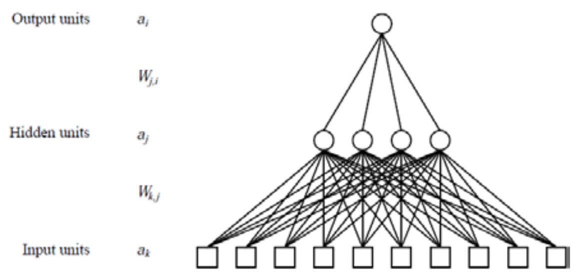


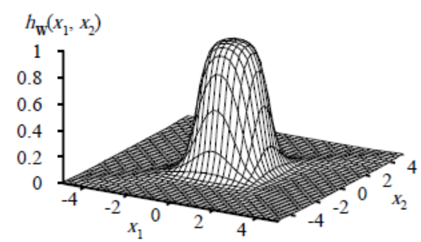
- Kernel trick: from linear to nonlinear

Neural Network & Deep Learning



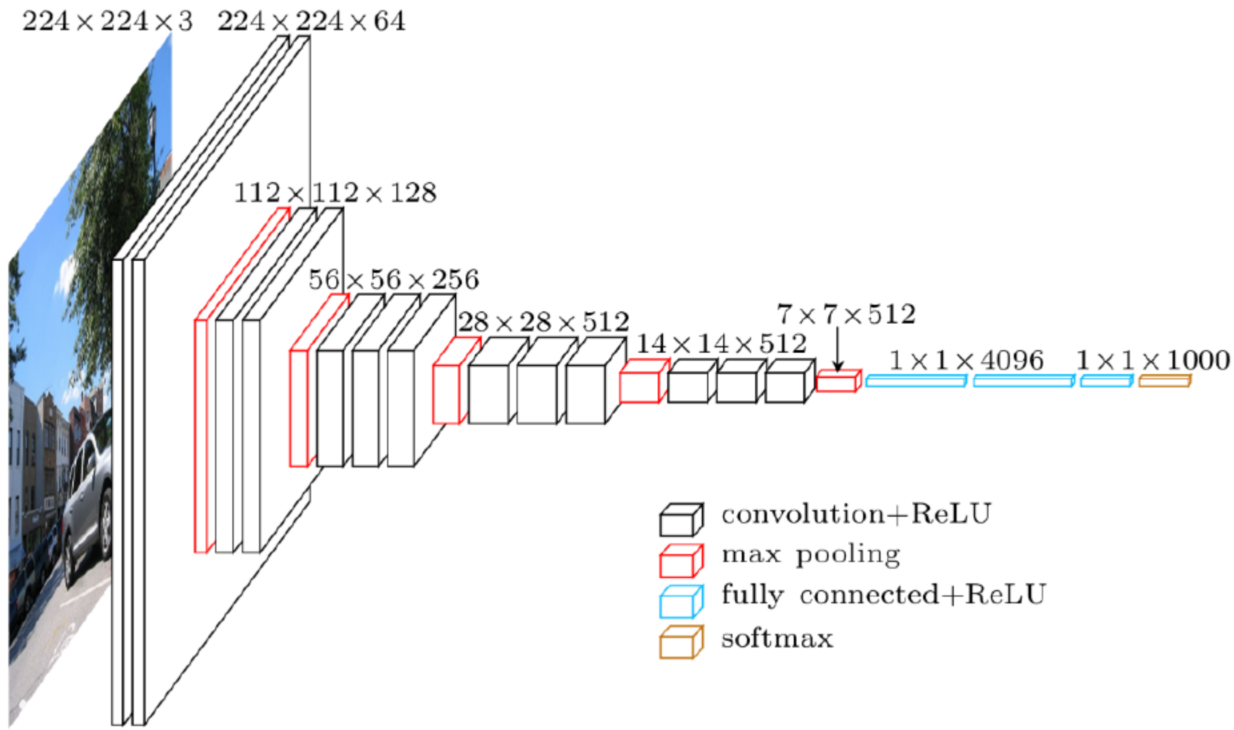






- Hidden unit & back-projection

Neural Network & Deep Learning



- Big data + GPU: NO manual features

Homework & Next Chapter

- Exercise
  - 18.6/18.15/18.16/18.17/18.19/18.22/18.23/18.25
- Machine Learning (practical skills!)
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning