ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

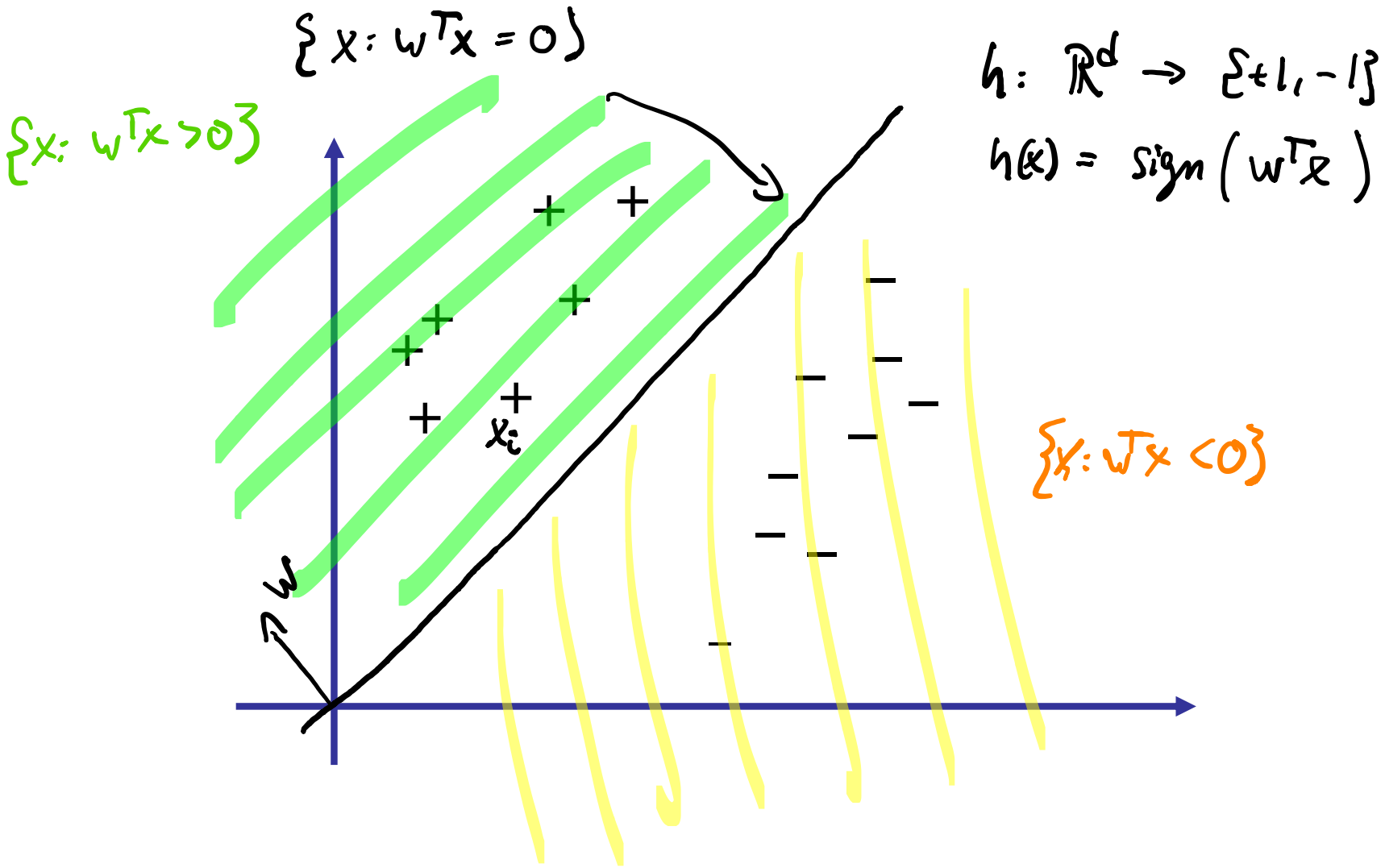
Introduction to Machine Learning

Linear Classification

Prof. Andreas Krause
Learning and Adaptive Systems (las.ethz.ch)

Linear classifiers

- Data set $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$



Surrogate optimization problem

- Instead of $$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell_{0/1}(\mathbf{w}; \mathbf{x}_i, y_i)$$
- Solve $$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell_{P}(\mathbf{w}; \mathbf{x}_i, y_i)$$

where $$\ell_P(\mathbf{w}; y_i, \mathbf{x}_i) = \max(0, -y_i \mathbf{w}^T \mathbf{x}_i)$$

Stochastic Gradient Descent

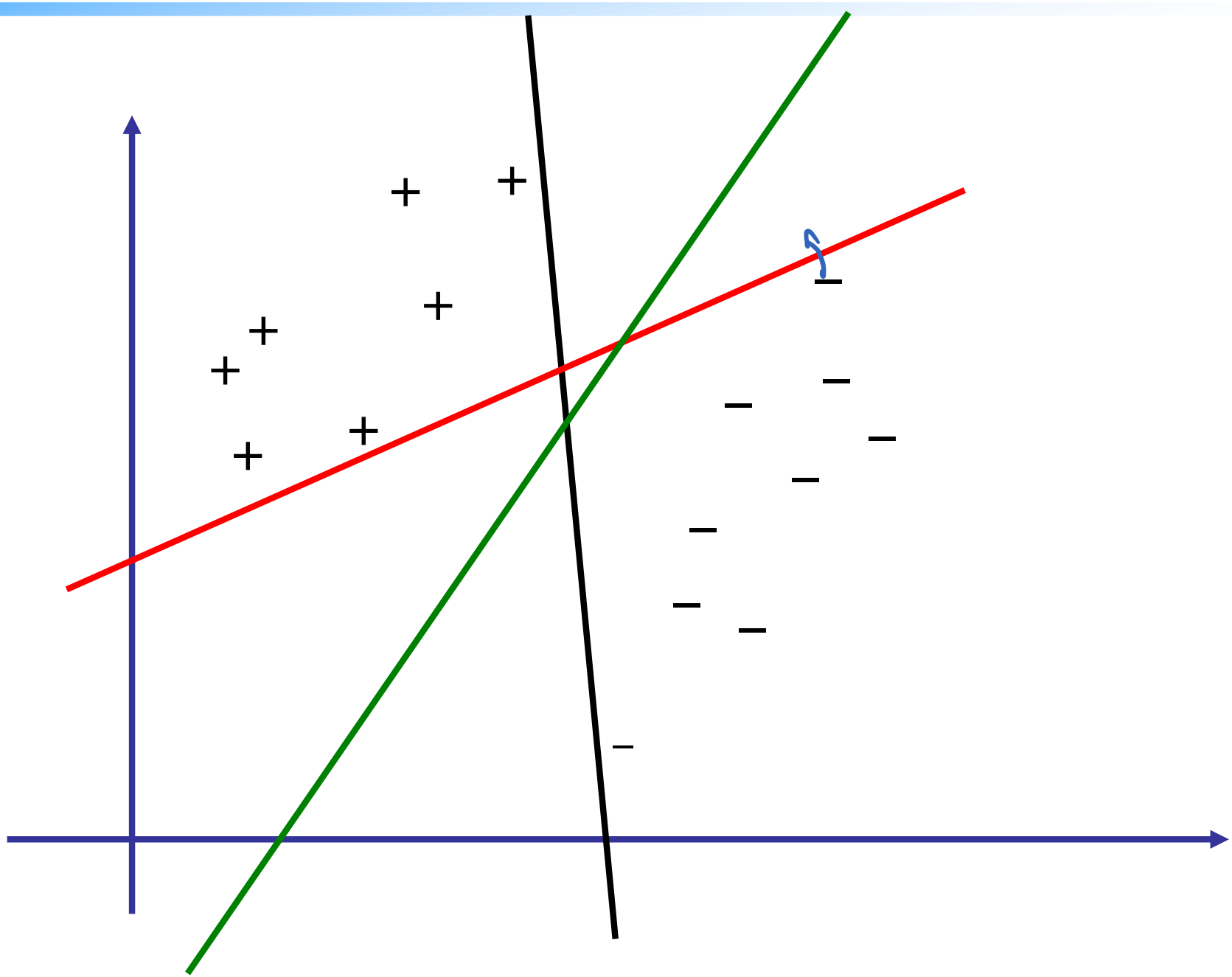
- Start at an arbitrary $\mathbf{w}_0 \in \mathbb{R}^d$
- For t=1,2,... do
  - Pick data point $(\mathbf{x}', y') \in D$ from training set uniformly at random (with replacement), and set

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla \ell(\mathbf{w}_t; \mathbf{x}', y')$$

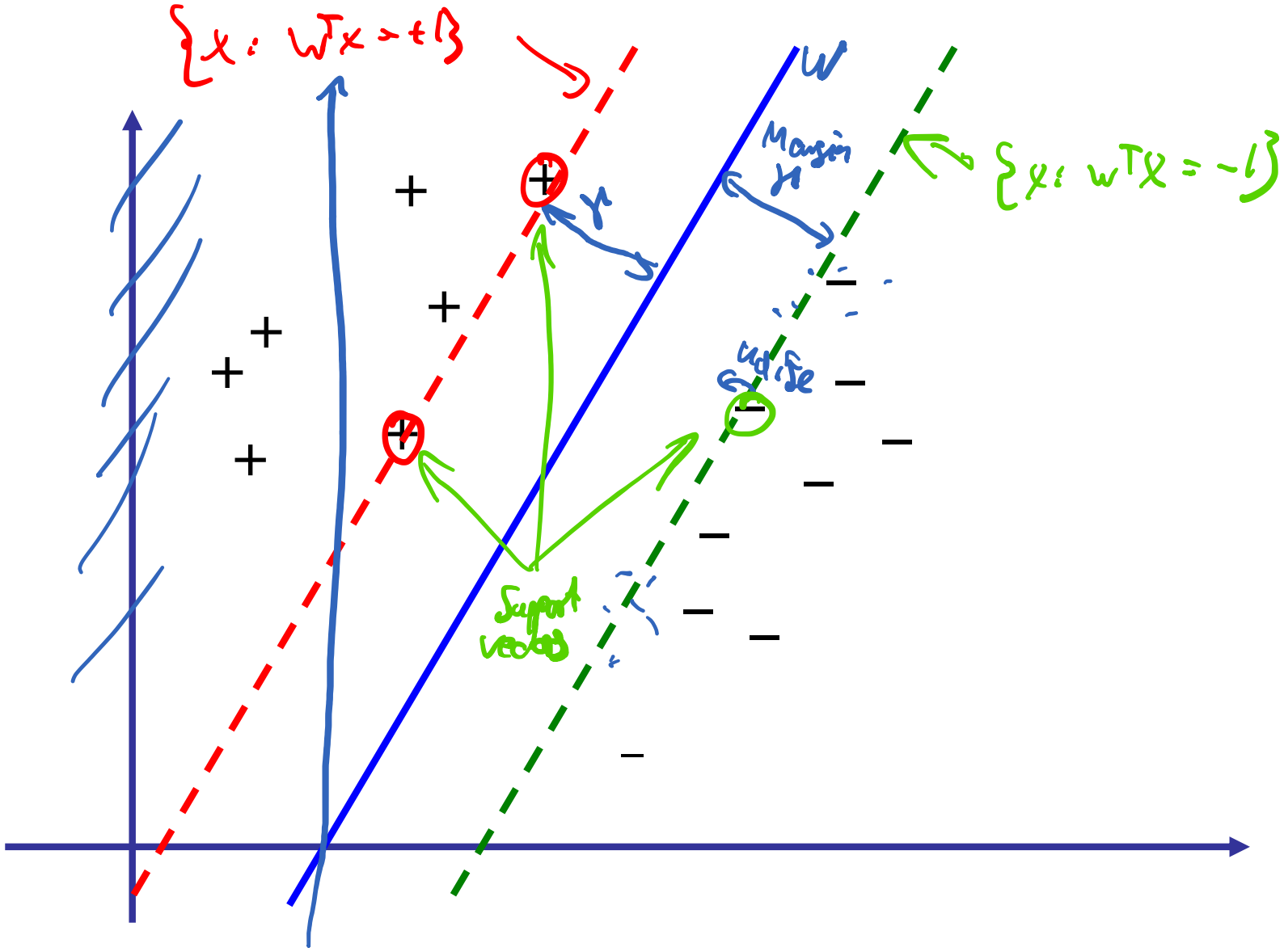
- Hereby, η_t is called learning rate
- Guaranteed to converge under mild conditions, if

$$\sum_t \eta_t = \infty \quad \text{and} \quad \sum_t \eta_t^2 < \infty$$

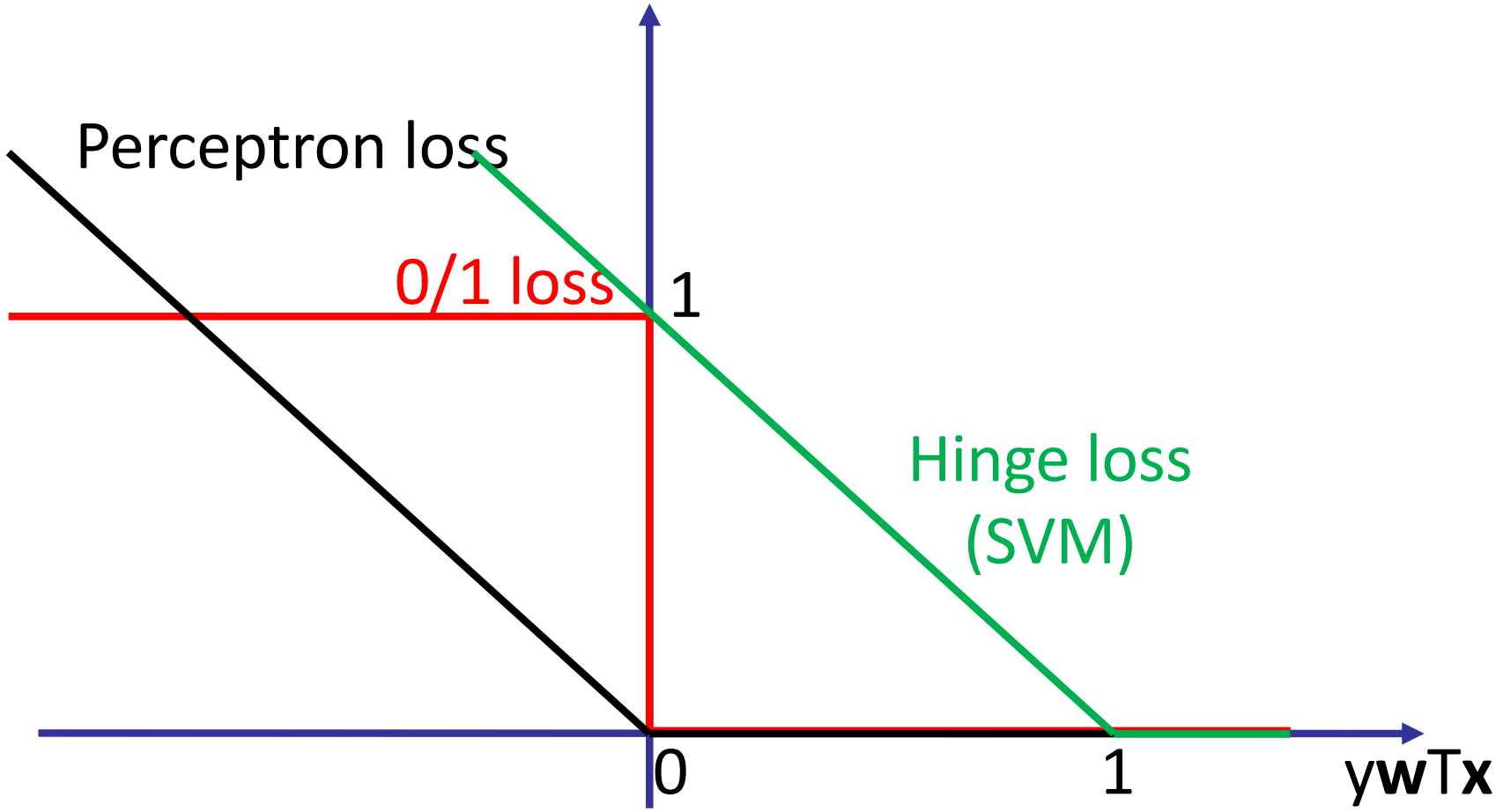
Which of these separators will the Perceptron "prefer"?



Support Vector Machines (SVMs): "max. margin" linear classification



Hinge vs. Perceptron loss



Hinge loss upper bounds #mistakes; encourages „margin“

$$\ell_H(\mathbf{w};\mathbf{x},y) = \max\{0, 1 - y\mathbf{w}^T\mathbf{x}\}$$

SVM vs. Perceptron

- Perceptron:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -y_i \mathbf{w}^T \mathbf{x}_i\}$$

- Support vector machine (SVM):

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} + \lambda ||\mathbf{w}||_2^2$$

Support vector machines

- Widely used, very effective linear classifer
- Almost like Perceptron. Only differences:
  - Optimize slightly different, shifted loss (hinge loss)
  - Regularize weights (like ridge regression)
- Can optimize via stochastic gradient descent
- Safe choice for learning rate: $\eta_t = \frac{1}{\lambda t}$
- More details in Advanced Machine Learning lecture

Choosing the regularization parameter

- Can pick regularization parameter via cross-validation just like in linear regression!
- Note that instead of using the hinge-loss for validation, would use the target performance metric (e.g., #mistakes)

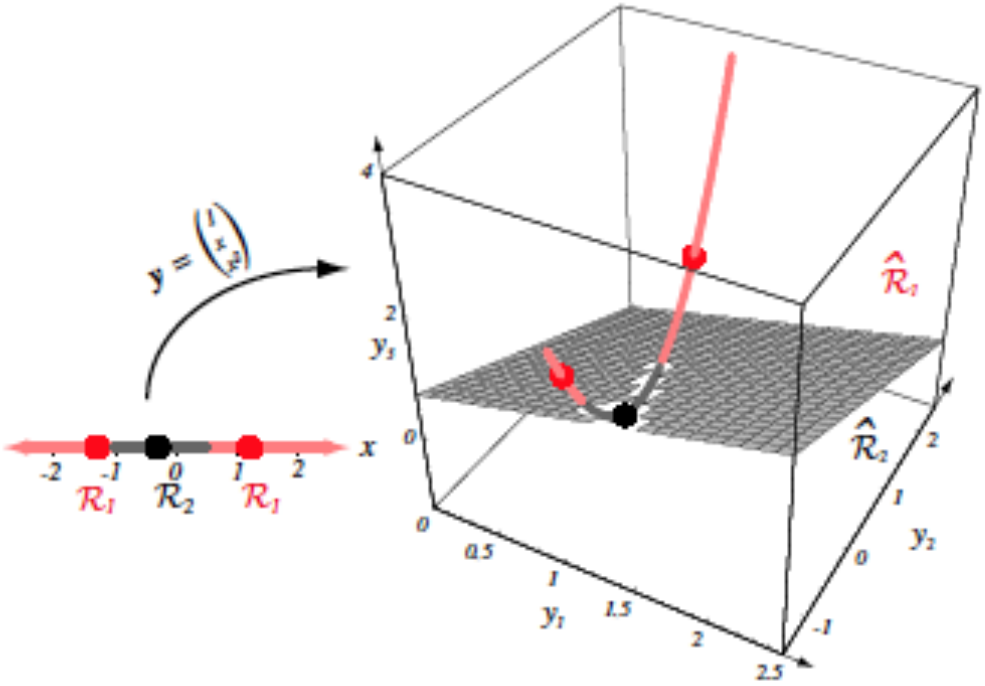
SVM in Scikit-Learn + Demo

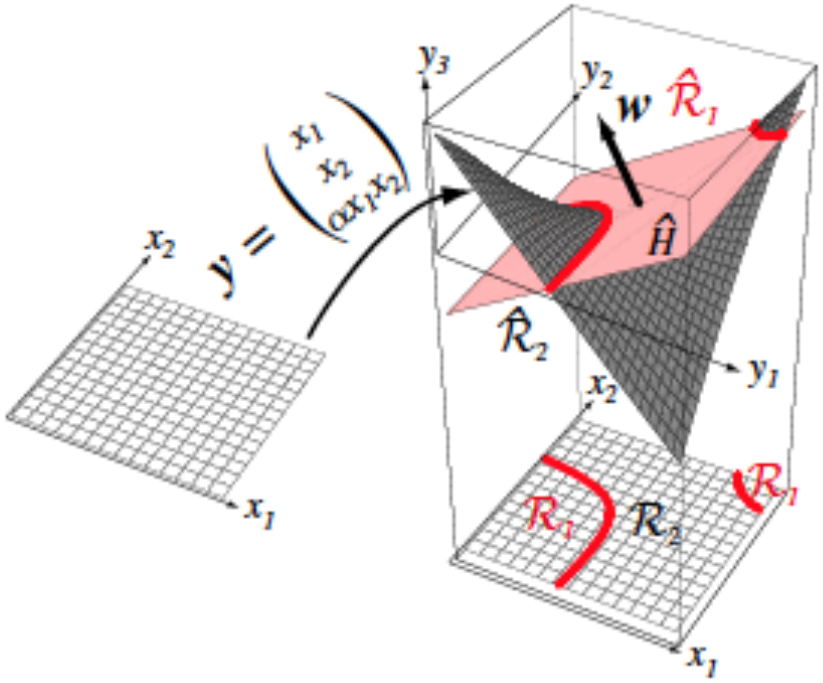
```
from sklearn.svm import LinearSVC

linearsvm = LinearSVC(C=1.0)
linearsvm.fit(X_train, y_train)
y_predict = linearsvm.predict(X_test)
```

Preview: non-linear classification

- How can we find nonlinear classification boundaries?
- Similar as in regression, can use non-linear transformations of the feature vectors





Recall: linear regression for polynomials

- We can fit non-linear classifiers via linear methods, using nonlinear features of our data (basis functions)

$$f(\mathbf{x}) = \sum_{i=1}^{d} w_i \phi_i(\mathbf{x})$$

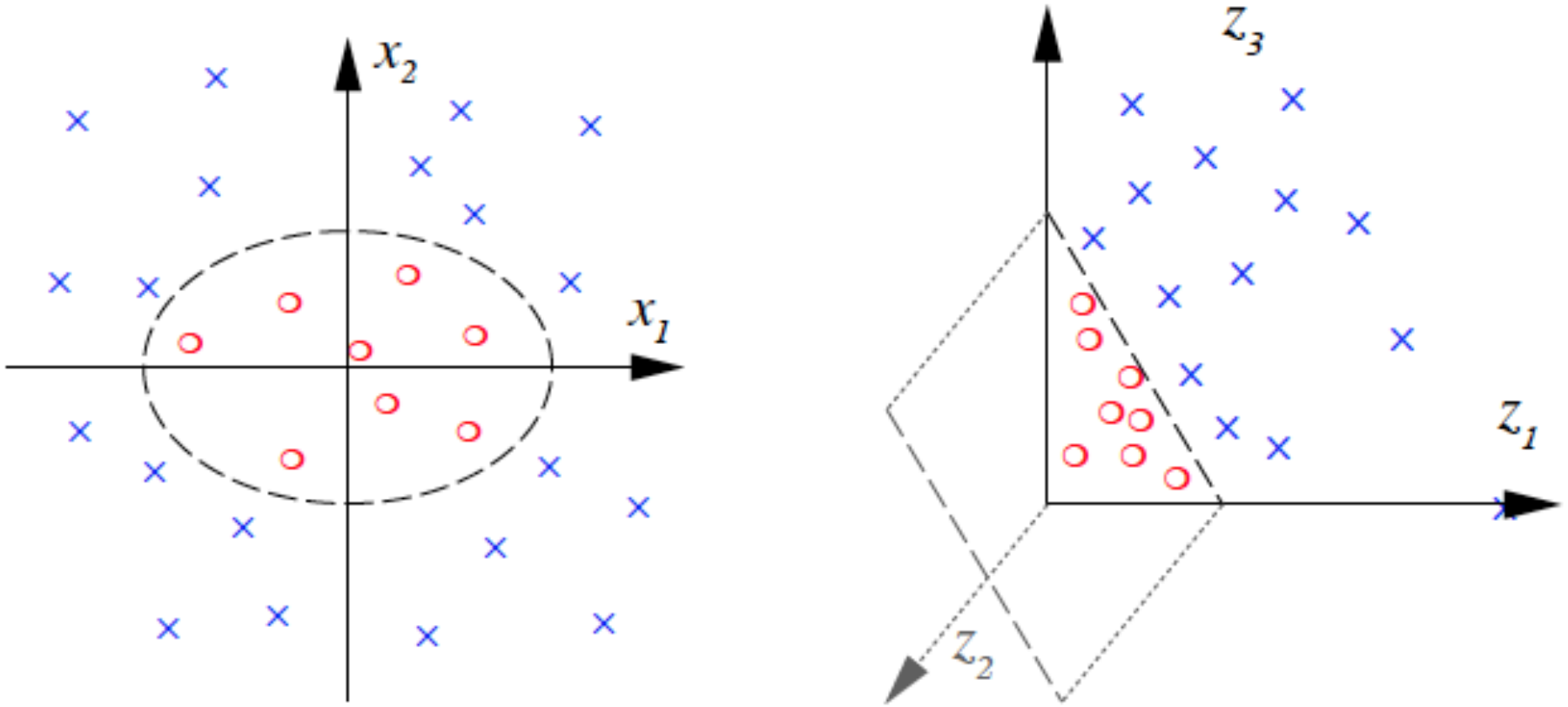
- For example: polynomials (in 1-D)

$$f(x) = \sum_{i=0}^{m} w_i x^i$$

- Higher dimensions -> Monomials

Example

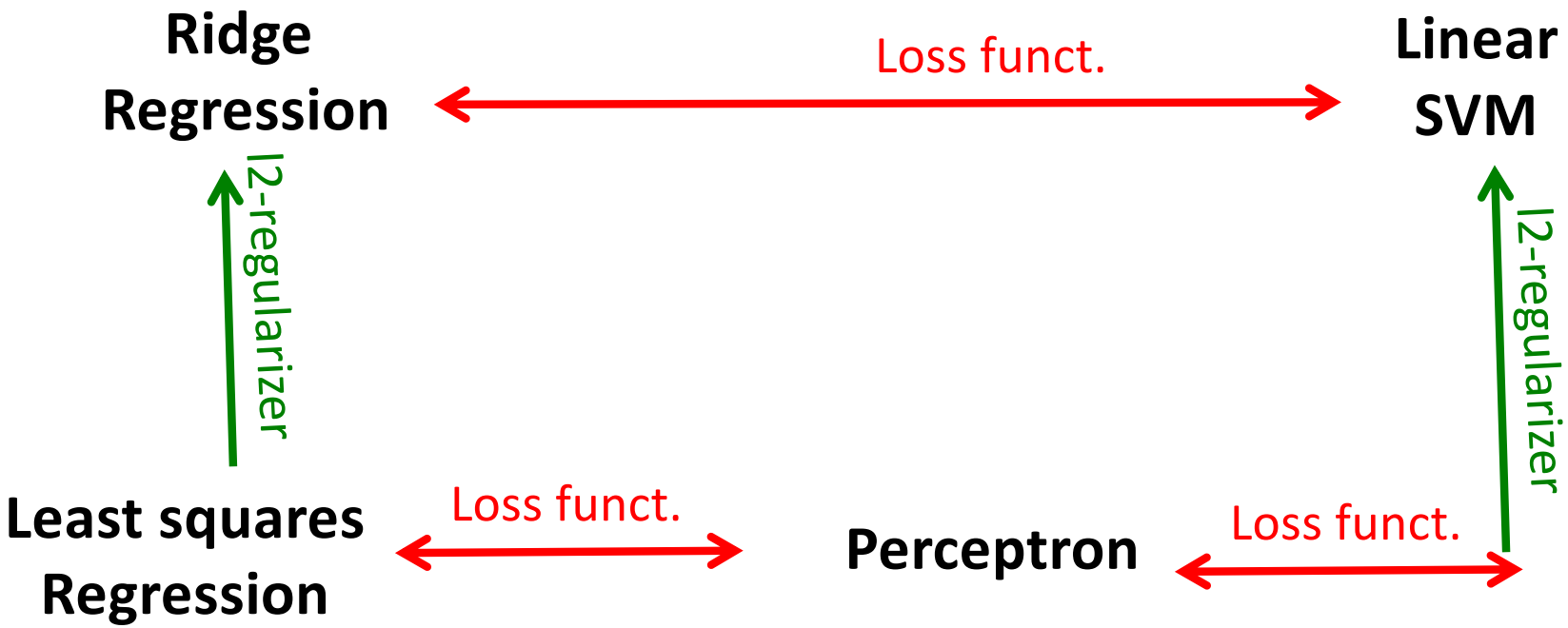
$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$



What you need to know

- The Perceptron is an algorithm for linear classification
- It applies Stochastic Gradient Descent (SGD) on the Perceptron loss
- Mini-batches exploit parallelism, reduce variance
- The Perceptron loss is a convex surrogate function for the 0-1 (misclassification) loss
- It is guaranteed to produce a feasible solution (a linear separator) if the data is separable
- SGD is much more generally applicable
- Support Vector Machines (SVMs) are closely related to Perceptron; use hinge loss and regularization

Supervised learning big picture so far



Supervised learning summary so far

Representation/ features	Linear hypotheses; nonlinear hypotheses with nonlinear feature transforms		
Model/ objective:	Loss-function Squared loss, 0/1 loss, Perceptron loss, Hinge loss	+	Regularization L^2 norm
Method:	Exact solution, Gradient Descent, (mini-batch) SGD, Convex Programming, …		
Evaluation metric:	Mean squared error, Accuracy		
Model selection:	K-fold Cross-Validation, Monte Carlo CV		

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Introduction to Machine Learning

Feature selection

Prof. Andreas Krause
Learning and Adaptive Systems (las.ethz.ch)

Feature selection

- In many high-dimensional problems, we may prefer not to work with all potentially available features

- Why?
  - Interpretability (would like to "understand" the classifier, identify important variables/features)
  - Generalization (simpler models may generalize better)
  - Storage / computation / cost (don't need to store / sum / acquire data for unused features…)

- How?
  - Naïve: try all subsets, and pick best (via crossvalidation)

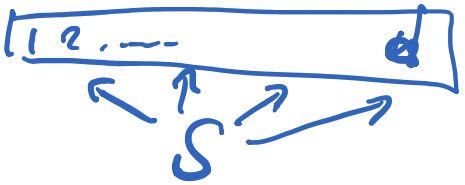
Greedy feature selection

- General purpose approach:
  Greedily add (or remove) features to maximize
  - Cross-validated prediction accuracy
  - Mutual information or other notions of informativeness (not discussed)

- Can be used for *any* method
  (not only linear regression/classification)

Details

- Set of all features: $V=\{1,\ldots,d\}$



- Define cost function for scoring subsets S of V

$k = |S|$

$\hat{L}(S)$ is cross-validation error using features in S only

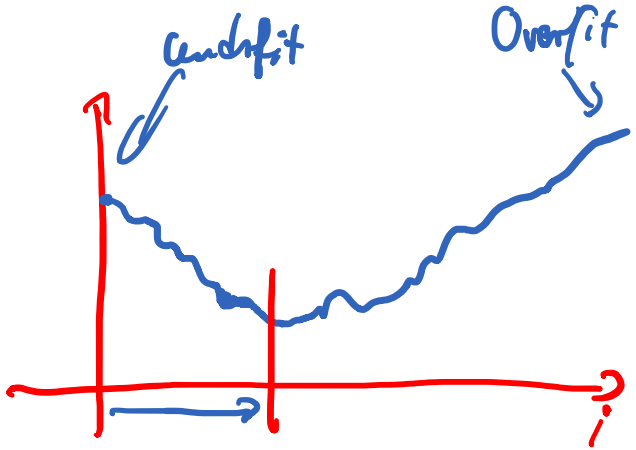
$x_i = [x_{i1} \ldots x_{id}] \longrightarrow x_{S,i} = [x_{i,i_1} \ldots x_{i,i_k}]$ $\quad S = \{i_1 \ldots i_k\}$

Train $\hat{w}_S = \underset{w \in \mathbb{R}^k}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(w; x_{S,i}, y_i) + \lambda \|w\|_2^2$

$\hat{L}(S)$ ← cross-validation error estimate of $\hat{w}_S$

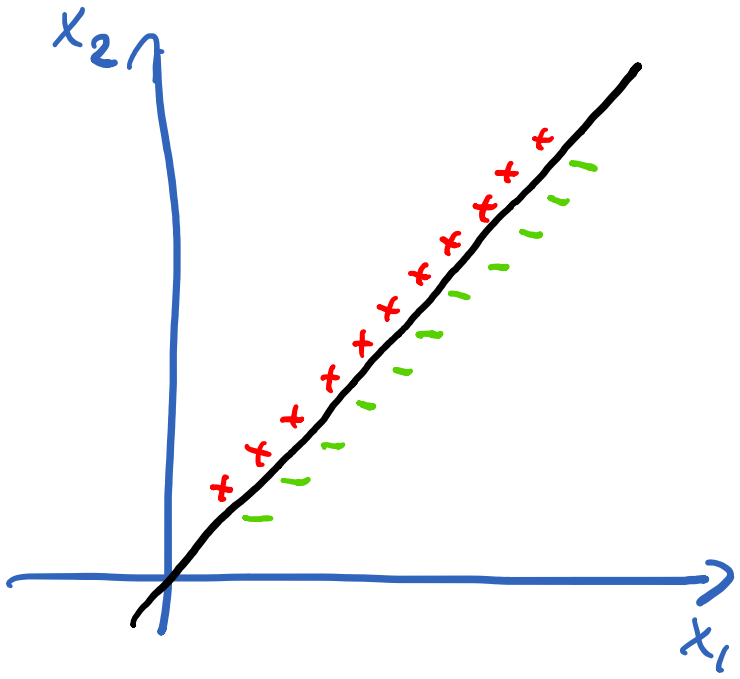
Greedy forward selection

- Start with $S = \emptyset$ and $E_0 = \infty$
- For i = 1:d
  - find best element to add: $s_i = \arg\min_{j \in V \setminus S} \hat{L}(S \cup \{j\})$
  - compute error: $E_i = \hat{L}(S \cup \{s_i\})$
  - If $E_i > E_{i-1}$ break, else set $S \leftarrow S \cup \{s_i\}$



Example

Problems with greedy forward selection

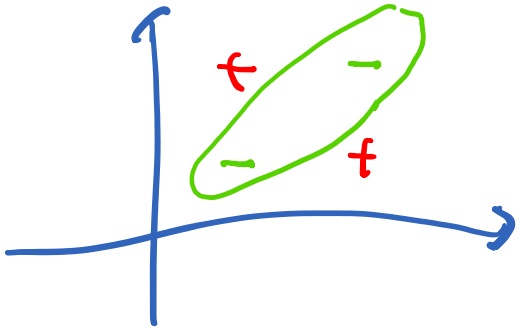


With x_1 or x_2 alone, have ~50% error $\Rightarrow \hat{L}(\{\}) = \hat{L}(\{x_1\}) = \hat{L}(\{x_2\}) = .5$

With x_1 and x_2, data is linearly separable

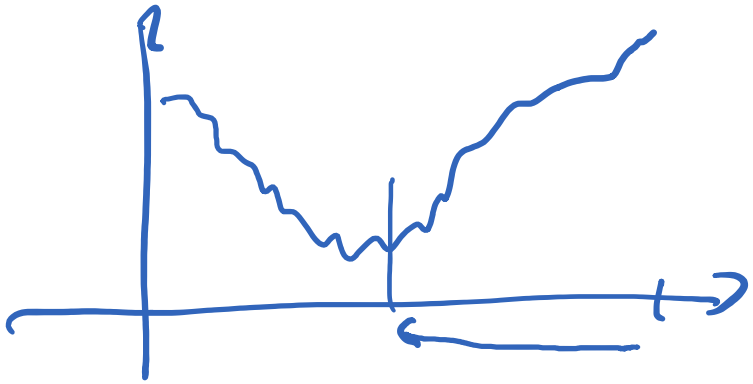
$$\hat{L}(\{x_1, x_2\}) = 0$$

Also, for nonlinear classification



Greedy backward selection

- Start with $S = V$ and $E_{d+1} = \infty$
- For *i = d:-1:1*
  - find best element to remove: $s_i = \arg\min_{j \in S} \hat{L}(S \setminus \{j\})$
  - compute error: $E_i = \hat{L}(S \setminus \{s_i\})$
  - If $E_i > E_{i+1}$ break, else set $S \leftarrow S \setminus \{s_i\}$



Comparison: FW vs. BW selection

Method	*Forward (FW)*	*Backward (BW)*
Advantages	Usually faster (if few relevant features)	Can handle „dependent“ features

Problems with greedy feature selection

- Computational cost (need to retrain models many times for different feature combinations)
- Can be suboptimal

- Can we solve the learning & feature selection problem simultaneously via a single optimization?

Linear models: Feature selection = Sparsity

- So far: explicitly select a subset of features

$$\mathbf{x} = [x_1, \ldots, x_d] \quad \Rightarrow \quad \mathbf{x}_S = [x_{i_1}, \ldots, x_{i_k}]$$

optimize over coefficients $\mathbf{w}_S = [w_{i_1}, \ldots, w_{i_k}]$

$$\hat{\mathbf{w}}_S = \arg\min_{\mathbf{w}_S} \sum_{i=1}^{n} (y_i - \mathbf{w}_S^T \mathbf{x}_{i,S})^2$$

- This is equivalent to constraining $\mathbf{w}$ to be sparse (i.e., contain at most *k* non-zero entries)

Joint feature selection and training

- Would like to solve

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \text{ s.t. } ||\mathbf{w}||_0 \leq k$$

where $||\mathbf{w}||_0$ is the number of non-zeros in $\mathbf{w}$

- Alternatively, can penalize the number of nonzero entries:

$$\hat{w} = \underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_0$$

Making the optimization tractable

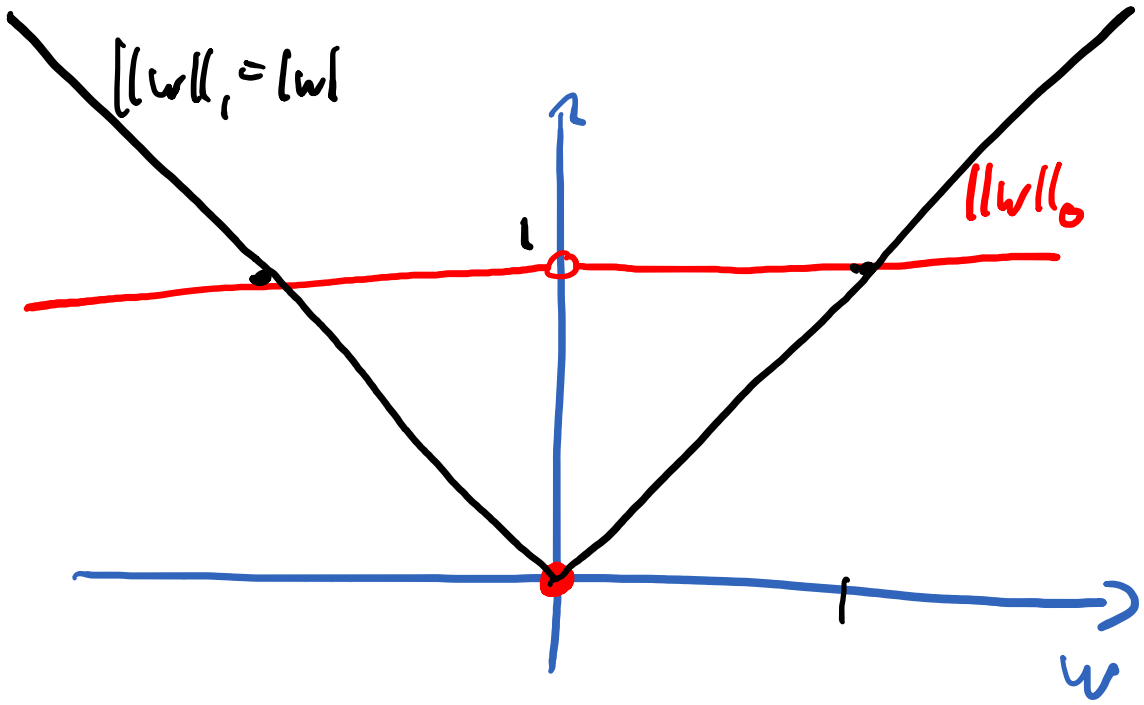
- Want to solve:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda ||\mathbf{w}||_0$$

- This is a difficult combinatorial optimization problem ☹
- Can view greedy algorithms before as heuristics for solving it

- **Key idea**: Replace $||\mathbf{w}||_0$ by a more tractable term

L1 as surrogate for L0



The „sparsity trick“

$$||\mathbf{w}||_0 \Rightarrow ||\mathbf{w}||_1$$

Sparse regression: The Lasso

- Before:
  - Ridge regression

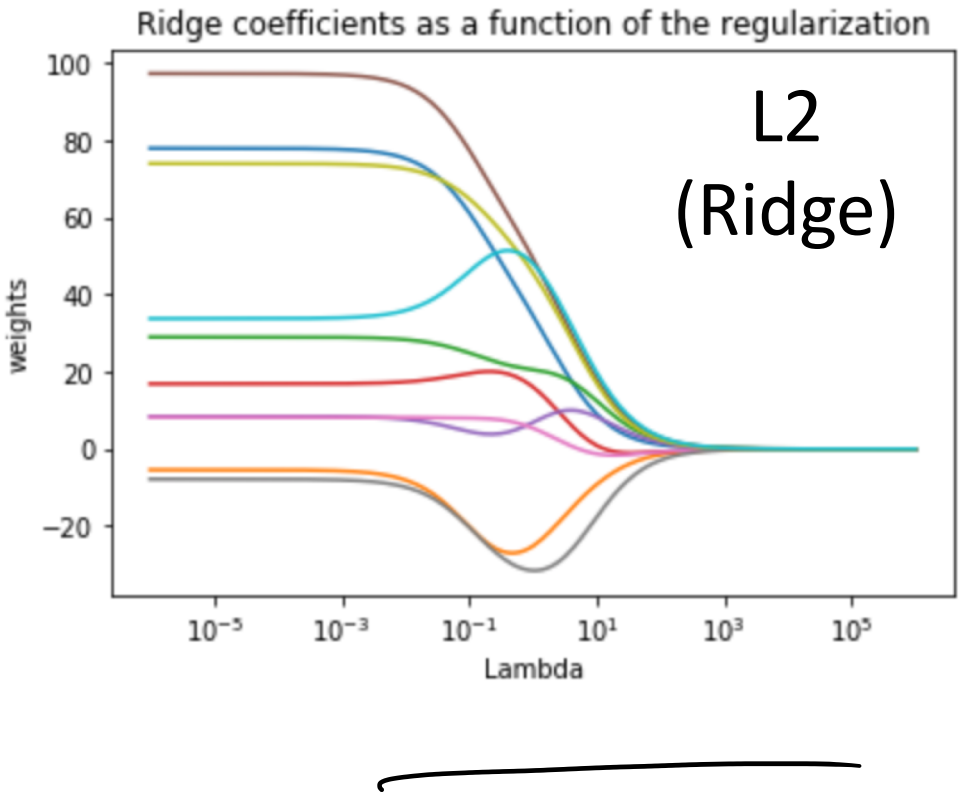
$$\min_{\mathbf{w}} \lambda ||\mathbf{w}||_2^2 + \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

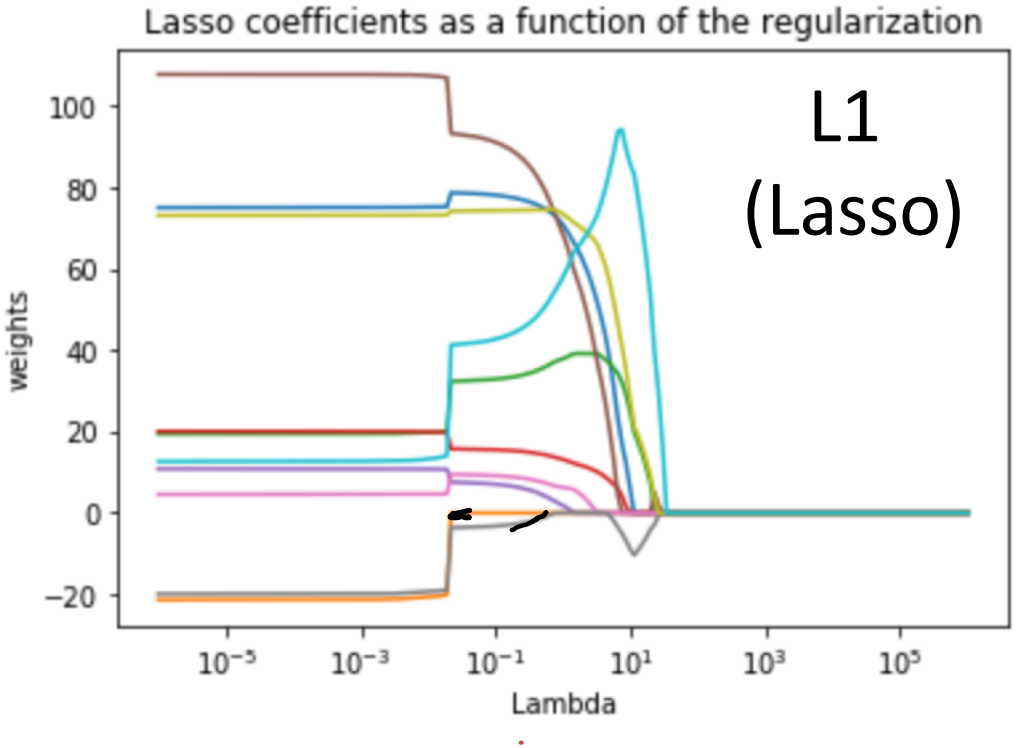
  - Uses $||\mathbf{w}||_2^2$ to control the weights
- Slight modification: replace $||\mathbf{w}||_2^2$ by $||\mathbf{w}||_1$
  - L1-regularized regression (the Lasso)

$$\min_{\mathbf{w}} \lambda ||\mathbf{w}||_1 + \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- This alternative penalty encourages coefficients to be exactly 0 ➔ automatic feature selection!

Regularization paths





How to pick the regularization parameter?

Crossvalidation!

Another example: L1-SVM

- Before:
  - Support vector machine

$$\min_{\mathbf{w}} \lambda||\mathbf{w}||_2^2 + \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)$$

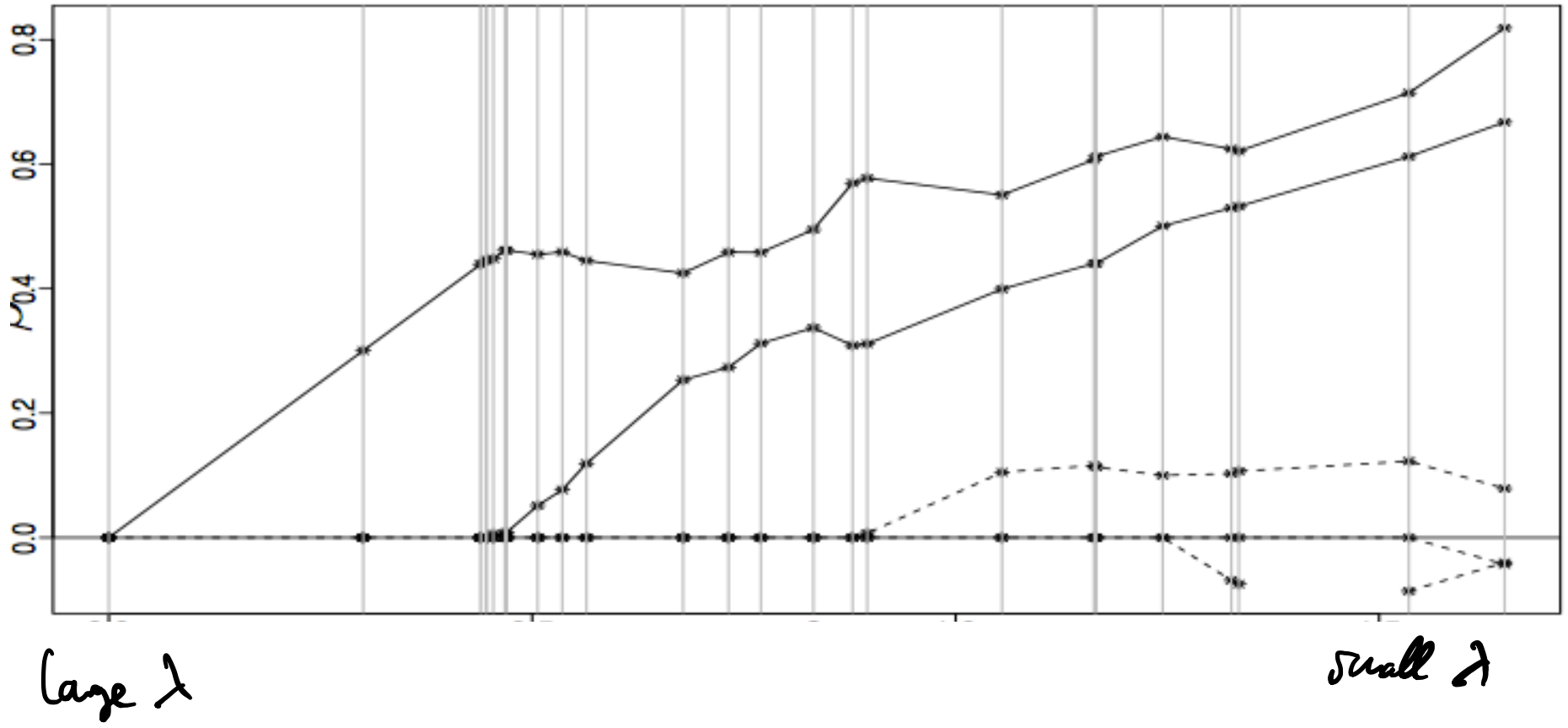
  - Uses $||w||_2$ to control the weights
- Apply sparsity trick: replace $||\mathbf{w}||_2^2$ by $||\mathbf{w}||_1$
  - L1-SVM

$$\min_{\mathbf{w}} \lambda||\mathbf{w}||_1 + \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)$$

- This alternative penalty encourages coefficients to be exactly 0 ➔ ignores those features!

Feature selection with L1-SVM

[Zhu et al NIPS '03]



Experiment

[Zhu et al NIPS '03]

- Data:
  - 38 train, 34 test data from a DNA microarray classification experiment (leukemia diagnosis)
  - 7129 dimensions

Method	CV Error	Test Error	# of Genes
2-norm SVM UR	2/38	3/34	22
2-norm SVM RFE	2/38	1/34	31
1-norm SVM	2/38	2/34	17

l1-SVM demo

Solving l1 regularized problems

- L1-norm is convex
- Combined with convex losses, obtain convex optimization problems (e.g., Lasso, l1-SVM, …)
- Can in principle solve using (stochastic) gradient descent
- However, convergence usually slow, and will rarely get „exact 0“ entries
- Much recent work in convex optimization deals with solving such problems very efficiently
  - Proximal methods (not discussed in this class)

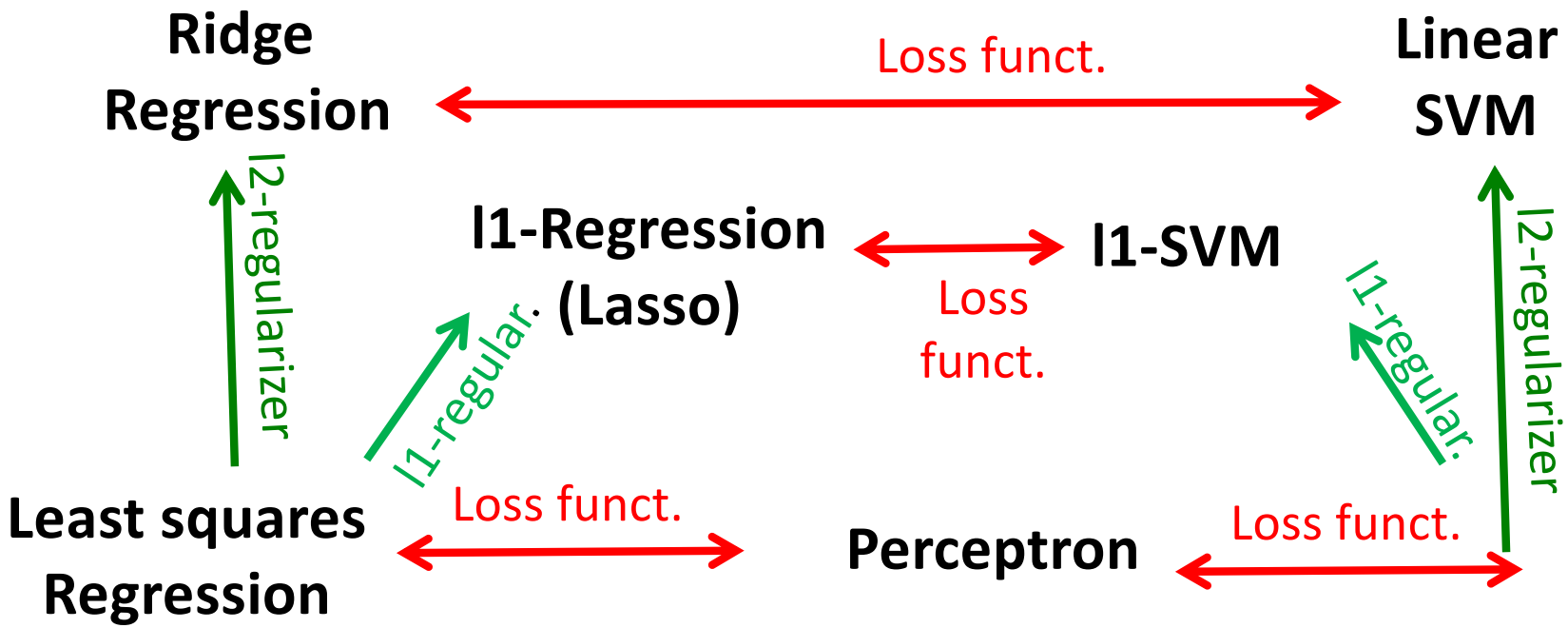
Comparison: Greedy selection vs. L1-Regularization

Method	*Greedy (FW/BW)*	*L1-Regularization*
Advantages	Applies to any prediction method	Faster (training and feature selection happen jointly)
Disadvantages	Slower (need to train many models)	Only works for linear models

What do you need to know

- What is feature selection
- Greedy algorithm (forward and backward)
- l1-regularization to encourage sparsity
  - Example: The Lasso (l1-regression)
  - Example: l1-SVM
- Advantages and disadvantages of the respective methods

Supervised learning big picture so far



Supervised learning summary so far

Representation/ features	Linear hypotheses; nonlinear hypotheses with nonlinear feature transforms		
Model/ objective:	Loss-function Squared loss, 0/1 loss, Perceptron loss, Hinge loss	+	Regularization L^2 norm, L^1 norm , L^0 penalty
Method:	Exact solution, Gradient Descent, (mini-batch) SGD, Convex Programming, …		
Evaluation metric:	Mean squared error, Accuracy		
Model selection:	K-fold Cross-Validation, Monte Carlo CV		