ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Data Mining
Learning from Large Data Sets

Lecture 5 – Large-scale supervised learning

263-5200-00L

Andreas Krause

Announcement

- No recitations this week
- No lecture next week (Easter holiday)

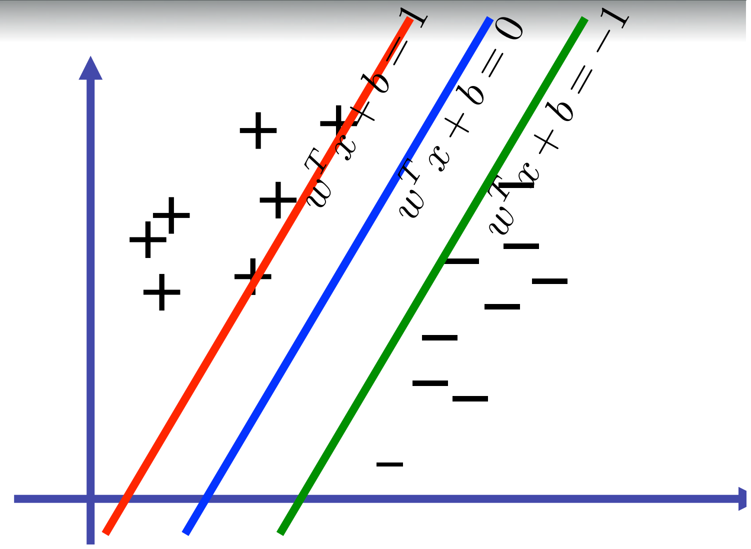
Course organization

- **Retrieval**
  - Given a query, find "most similar" item in a large data set
  - *Applications*: GoogleGoggles, Shazam, …
- **Supervised learning** (Classification, Regression)
  - Learn a concept (function mapping queries to labels)
  - *Applications*: Spam filtering, predicting price changes, …
- **Unsupervised learning** (Clustering, dimension reduction)
  - Identify clusters, "common patterns"; anomaly detection
  - *Applications*: Recommender systems, fraud detection, …
- **Learning with limited feedback**
  - Learn to optimize a function that's expensive to evaluate
  - *Applications*: Online advertising, opt. UI, learning rankings, …

Support Vector Machine

$$\min_{w,b} w^T w$$

$$\text{s.t.} y_i(w^T x_i + b) \geq 1$$



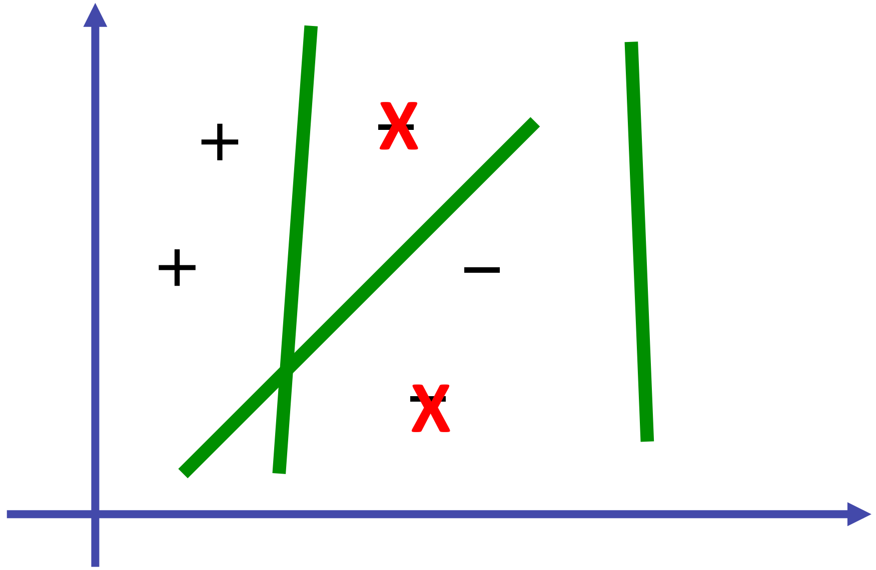
- How can we solve this optimization?
- What about local minima?

- This is a convex (quadratic) program

Dealing with massive data sets

- Are we done??
- Complexity of quadratic programming
  - Naïve implementations: $\Omega(n^3)$
- What if the data doesn't even fit in memory??
- Will see how one can reformulate the SVM optimization problem so that one can solve it on web-scale problems…

Online classification



X: Classification error

- Data arrives sequentially
- Need to classify one data point at a time
- Use a different decision rule (lin. separator) each time
- Can't remember all data points!

Generally: Online convex programming

- Input:
  - Feasible set $S \subseteq R^d$
  - Starting point $w_0 \in S$
- Each round t do
  - Pick new feasible point $w_t \in S$
  - Receive convex function $f_t : S \to \mathbb{R}$
  - Incur loss $\ell_t = f_t(w_t)$
- Regret:

$$R_T = \left(\sum_{t=1}^{T} \ell_t\right) - \min_{w \in S} \sum_{t=1}^{T} f_t(w)$$

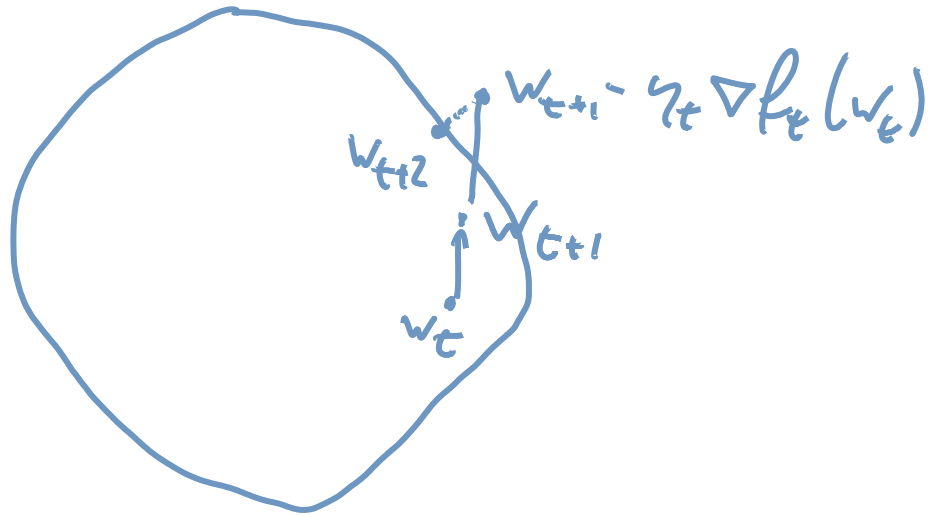
Solve: $\min \sum_{t=1}^{N} f_t(x)$ s.t. $x \in S$

E.g., SVM

Online convex programming

- Simple update rule:

$$w_{t+1} = \qquad w_t - \eta_t \nabla f_t(w_t)$$



$$\mathrm{Proj}_S(x) = \underset{x' \in S}{\operatorname{argmin}} \|x' - x\|_2$$

- How well does this simple algorithm do??

Regret for online convex programming

Theorem [Zinkevich '03]

Let $f_1, \ldots, f_T$ be an arbitrary sequence of convex functions with feasible set S

Set $\eta_t = 1/\sqrt{t}$

Then, the regret of online convex programming is bounded by

$$R_T \leq \frac{||S||^2\sqrt{T}}{2} + \left(\sqrt{T} - \frac{1}{2}\right)||\nabla f||^2$$

additional loss in accuracy due to online setting

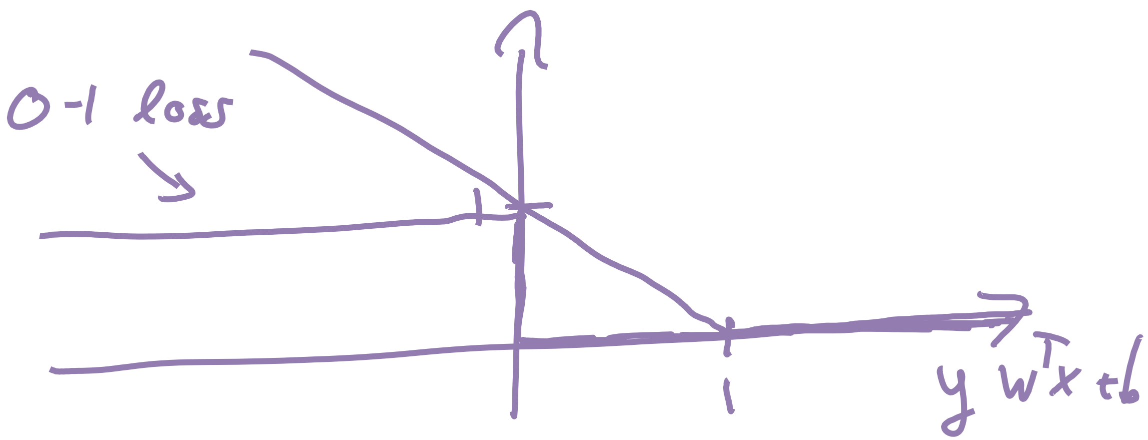
$$\frac{R_T}{T} = O\left(\frac{\sqrt{T}}{T}\right) = O\left(\frac{1}{\sqrt{T}}\right) \to 0$$

OCP for SVM formulation

$$\min_{w,b} \sum_{i=1}^{N} \max(0, 1 - y_i(w^T x_i + b))$$

hinge loss

$$\text{s.t.} ||w||_2 \leq \frac{1}{\lambda}$$



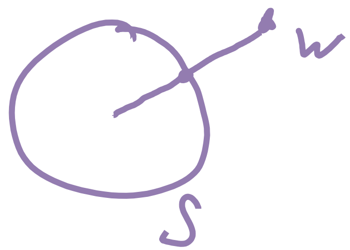
Online convex programming for SVM

$$w_{t+1} = \mathrm{Proj}_S\left(w_t - \eta_t \nabla f_t(w_t)\right)$$

- Feasible set: $S = \{w : ||w|| \leq \frac{1}{\lambda}\}$
- Projection:

$w \in \mathbb{R}^d$

$$\mathrm{Proj}_S(w) = \begin{cases} w & \text{if } w \in S \\ \frac{w}{||w||} \cdot \frac{1}{\lambda} & \text{if } w \notin S \end{cases}$$



- Gradient: $f_t(w) = \max(0, 1 - y_t(w^T x_t))$

Subgradient for SVM

- Hinge loss: $f_t(w) = \max(0, 1 - y_t(w^T x_t + b))$

- Subgradient:

If	$\partial_w f_t(w)$
$1 - y_t(w^T x_t + b) < 0$	0
$1 - y_t(w^T x_t + b) > 0$	$-y_t x_t$

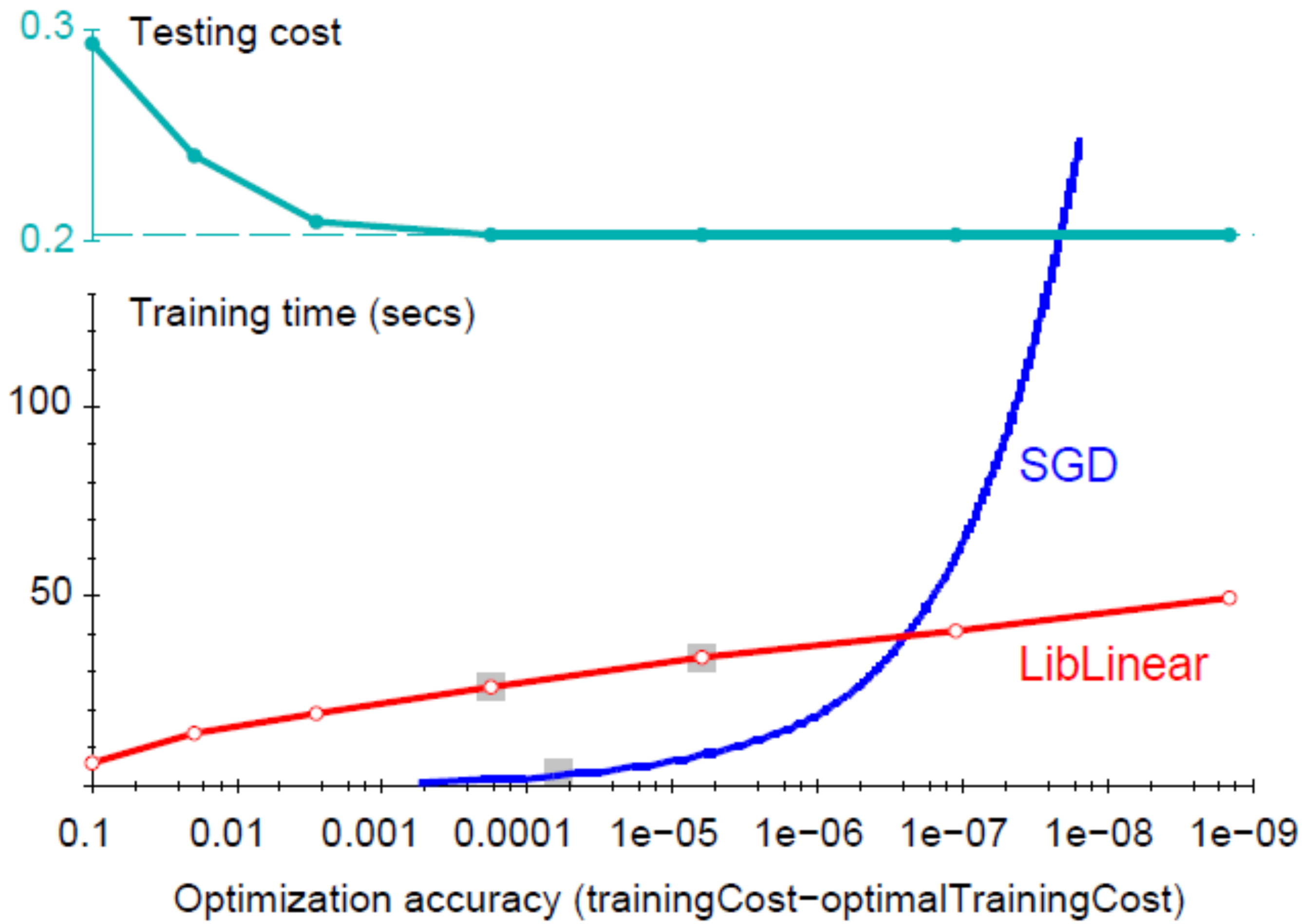
$$w_{t+1} = \mathrm{Proj}_S\left(w_t - \eta_t \partial_w f_t(w_t)\right)$$

Example [Bottou]

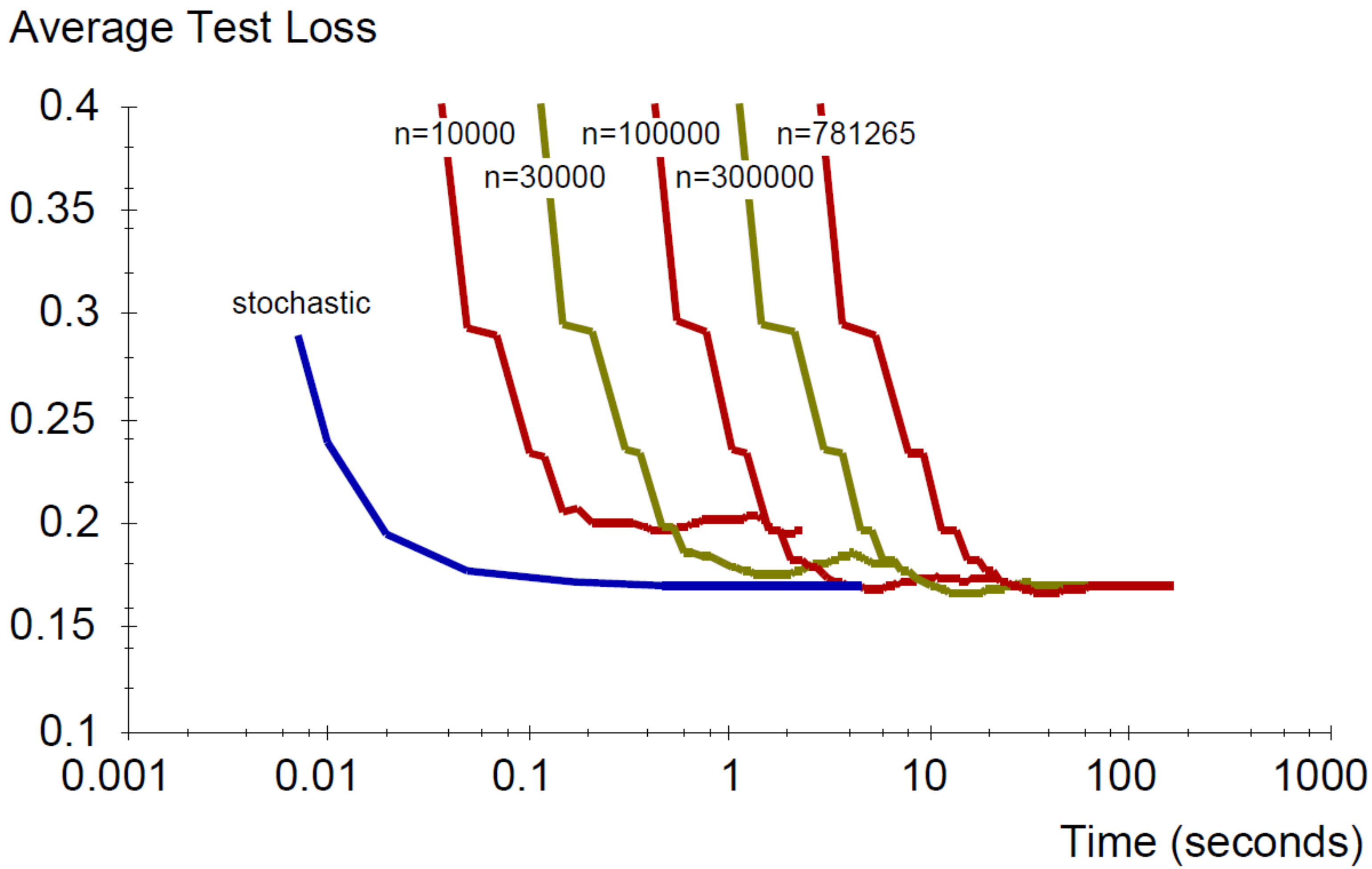
- Stochastic gradient descent
  - Online convex programming with training samples picked at random
- Data set:
  - Reuters RCV1
  - 780k training examples, 23k test examples
  - 50k dimensions

	Training Time	Primal cost	Test Error
SVMLight	23,642 secs	*0.2275*	*6.02%*
SVMPerf	66 secs	0.2278	6.03%
SGD	1.4 secs	*0.2275*	*6.02%*

Error


0.3
Testing cost
0.2
Training time (secs)
100
50
SGD
LibLinear
0.1
0.01
0.001
0.0001
1e−05
1e−06
1e−07
1e−08
1e−09
Optimization accuracy (trainingCost−optimalTrainingCost)

Subsampling



State of the art: PEGASOS

INPUT: training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$,

Regularization parameter λ,

Number of iterations T

INITIALIZE: Choose $\mathbf{w}_1$ s.t. $\|\mathbf{w}_1\| \leq 1/\sqrt{\lambda}$

FOR $t = 1, 2, \ldots, T$

Choose $A_t \subseteq S$ ← "Batch"

$A_t^+ = \{(\mathbf{x}, y) \in A_t : y\langle \mathbf{w}_t, \mathbf{x}\rangle < 1\}$

$\nabla_t = \lambda \mathbf{w}_t - \frac{\eta_t}{|A_t|} \sum_{(\mathbf{x},y)\in A_t^+} y\,\mathbf{x}$

$\eta_t = \frac{1}{t\lambda}$

$\mathbf{w}_t' = \mathbf{w}_t - \eta_t \nabla_t$

$\mathbf{w}_{t+1} = \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_t'\|}\right\} \mathbf{w}_t'$

OUTPUT: $\mathbf{w}_{T+1}$

$\min_w \lambda \|w\|^2 + \frac{1}{N}\sum_{i=1}^{N} \text{hinge}_i(w)$

$= \min_w \sum_{i=1}^{N} f_i(w)$

$f_i(w) = \lambda \|w\|_2^2 + \text{hinge}(\cdot)$ ($w^T w$)

$\frac{\partial}{\partial w} f_i(w) = 2\lambda w + \frac{\partial}{\partial w}\text{hinge}\ldots$

Performance for PEGASOS

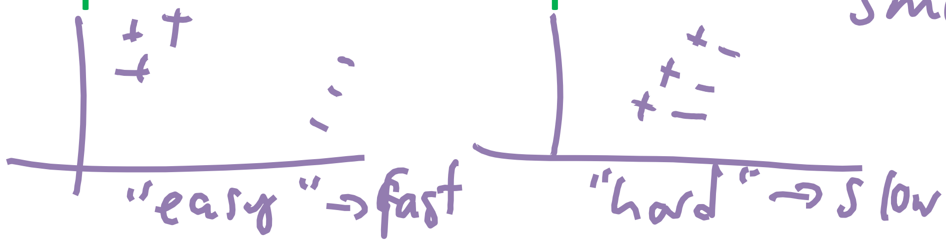
- <u>Theorem</u> [Shalev-Shwartz et al. '07]:
  - Run-time required for Pegasos to find ε-accurate solution with probability at least 1-δ:

$$O^*\left(\frac{d \log \frac{1}{\delta}}{\lambda\varepsilon}\right) = O\left(\frac{d}{\lambda\varepsilon}\right)$$

See paper for details; d = dim.

- Depends on
  - number of dimensions d
  - "difficulty" of problem (λ and ε)
- Does **not** depend on #examples n

Small λ ⇓ Large w ⇓ Small margin γ, $\gamma = \frac{1}{\|w\|}$



Difference between PEGASOS and standard OCP / SGD

- Uses batches of training examples
  ➔ empirically more efficient
- Uses «strongly convex» loss functions
  ➔ improved convergence rate, and better empirical performance
- Only guaranteed to work in the stochastic setting (i.e., can't handle arbitrary ordering of data)

Dealing with massive data

- Online convex programming lets one train an SVM, processing one data point at a time
  - No need to store data in memory
  - Order doesn't matter (for general OCP)!
- What about truly massive data?
  - Streaming 1 TB ~4-5 hours
- Can we do parallel processing in data centers?
  - Map reduce for SVM??

Parallel online learning

- Various different approaches [Zinkevich et al ’10]

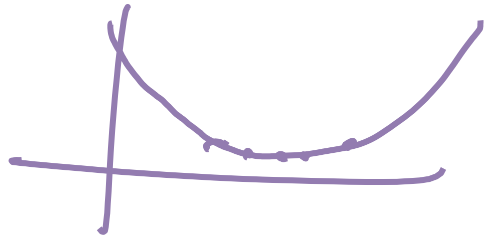
Algorithm	Latency tolerance	MapReduce	Network IO	Scalability
Distributed subgradient [3, 9]	moderate	**yes**	high	**linear**
Distributed convex solver [7]	**high**	**yes**	**low**	unclear
Multicore stochastic gradient [5]	low	no	n.a.	**linear**
Parallel stochastic gradient descent [Zinkevich ‘10]	**high**	**yes**	**low**	**linear**

- *Still active area of research*

Parallel stochastic gradient descent
[Zinkevich et al '10]

- "Data parallel" method for solving

$$\min_w \lambda ||w||^2 + \frac{1}{T}\sum_{t=1}^{T} f_t(w)$$



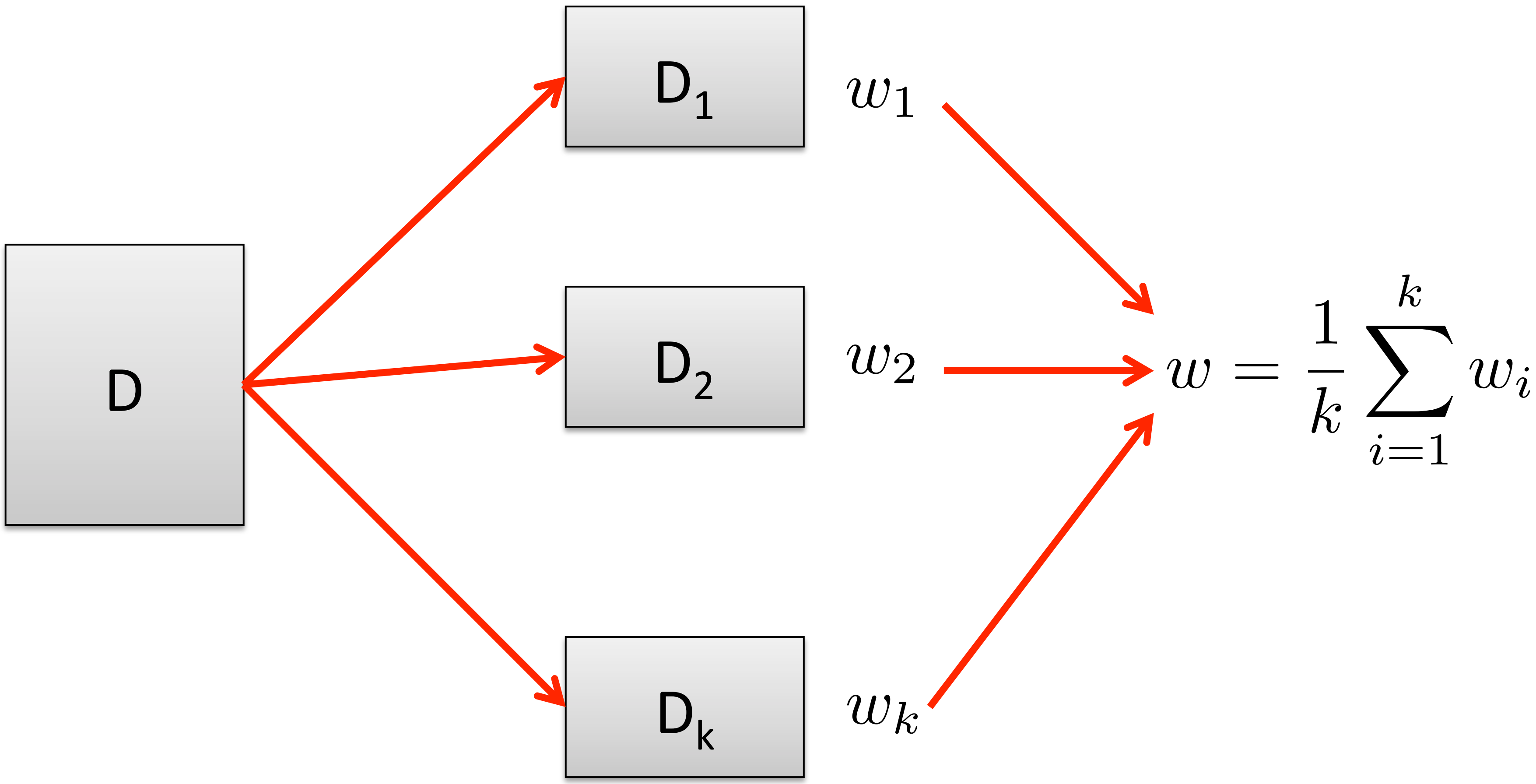
- Randomly partition data set to k machines
- Each machine runs SGD independently, produces w_i
- After T iterations, compute

$$w = \frac{1}{k}\sum_{i=1}^{k} w_i$$

- How well does this algorithm do?
- Does parallelism help?

Parallel stochastic gradient descent

[Zinkevich et al '10]



Parallel stochastic gradient descent
[Zinkevich et al '10]

Theorem: Suppose each of the *k* machines runs for

$$T = \Omega\Big(\frac{\log \frac{k\lambda}{\varepsilon}}{\varepsilon\lambda}\Big)$$

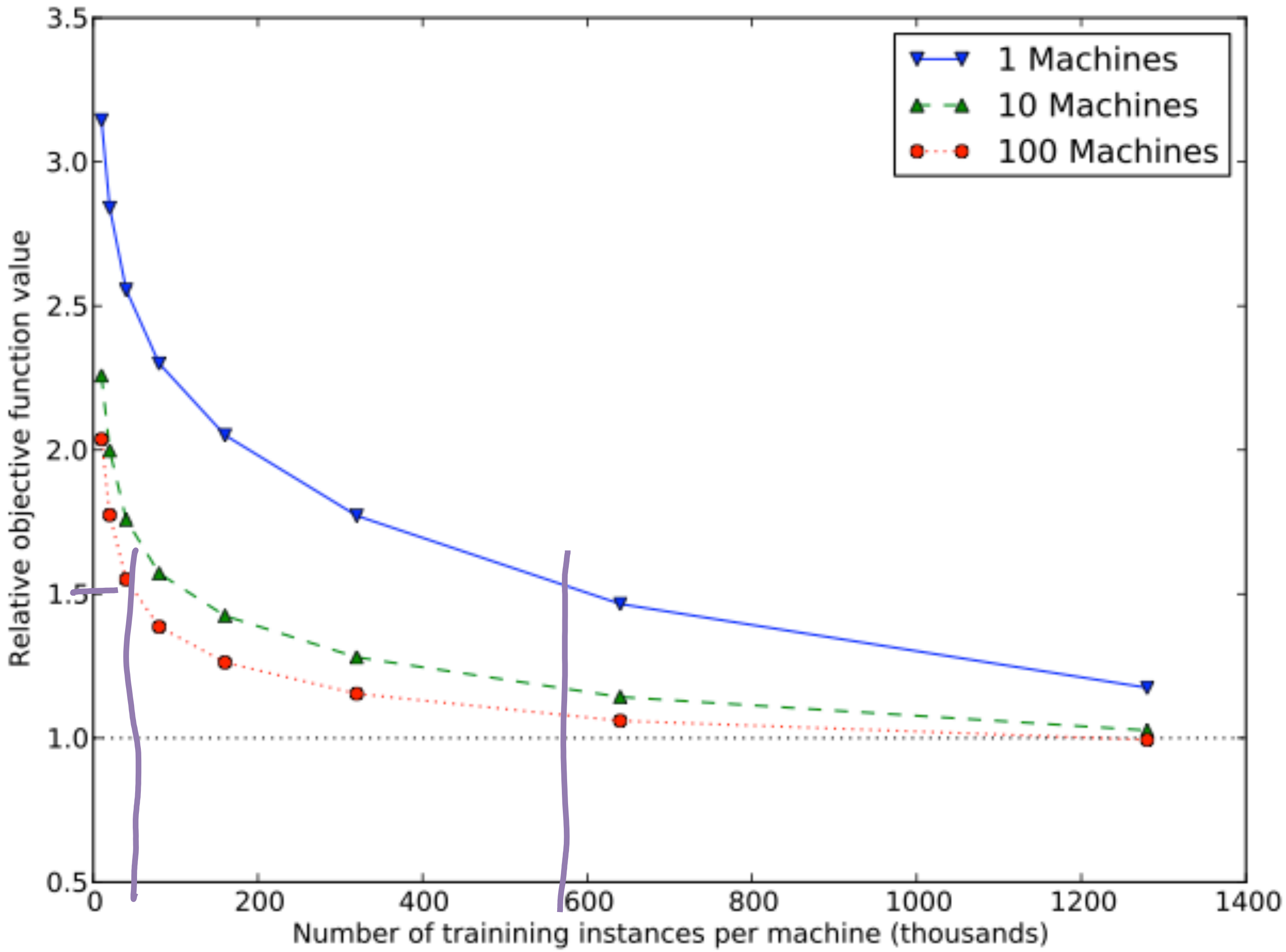
Then: $$\mathbb{E}[\text{ error }] \leq \mathcal{O}\Big(\varepsilon\big(\frac{1}{\sqrt{k\lambda}} + 1\big)\Big)$$

Parallelization helps, but only if $k = \mathcal{O}\Big(\frac{1}{\lambda}\Big)$

The "more difficult" the learning problem (the smaller λ), the more parallelization helps!

Performance of parallel online SGD

[Zinkevich et al '10]



Summary so far

- Support Vector Machines
  - State of the art linear classifier
  - Requires solving convex program
- Online convex programming
  - Simple, online algorithm for approximately minimizing additive loss functions
  - Only require (sub-)gradients and reprojection
- Stochastic gradient descent
  - Online convex programming in random order
- Parallelized stochastic gradient descent
  - Compute gradients independently, then average
  - Amount of effective parallelism depends on "hardness" of problem

More results on supervised learning

- **Feature selection**
- Dealing with multiple classes
- Linear regression
- Nonlinear classification / regression

Feature selection

- In many high-dimensional problems, we may prefer "sparse" solutions: $\text{sign}(w^T x + b)$

  where w contains only few nonzero entries)

- Reasons:
  - Interpretability (would like to "understand" the classifier, identify important variables)
  - Generalization (simpler models may generalize better)
  - Storage / computation (don't need to store / sum data for 0 coefficients…)

Feature selection

- Suppose we would like to identify top *k* features
- Approach 1
  - Try out all sets of at most k variables
  - Fit a classifier to each set, ignoring the non-selected variables
  - Pick the best set
  - Problem?
- Approach 2
  - Greedily select the features: Add one at a time to maximize improvement in accuracy
  - Problem?
- Ideally: Solve classification and feature selection in one fell-swoop!

Sparsity enforcing regularizers

- Before:
  - Support vector machine

$$\|w\|_2^2 = \sum_{i=1}^{D} w_i^2$$

$$\min_{w,b} \lambda ||w||_2^2 + \sum_i \max(0, 1 - y_i(w^T x_i + b))$$

  - Uses $||w||_2$ to control the weights
- Slight modification: replace $||w||_2$ by $||w||_1 = \sum_{i=1}^{D} |w_i|$
  - L1-SVM

$$\min_{w,b} \lambda ||w||_1 + \sum_i \max(0, 1 - y_i(w^T x_i + b))$$

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

Feature selection with L1-SVM

[Zhu et al NIPS '03]

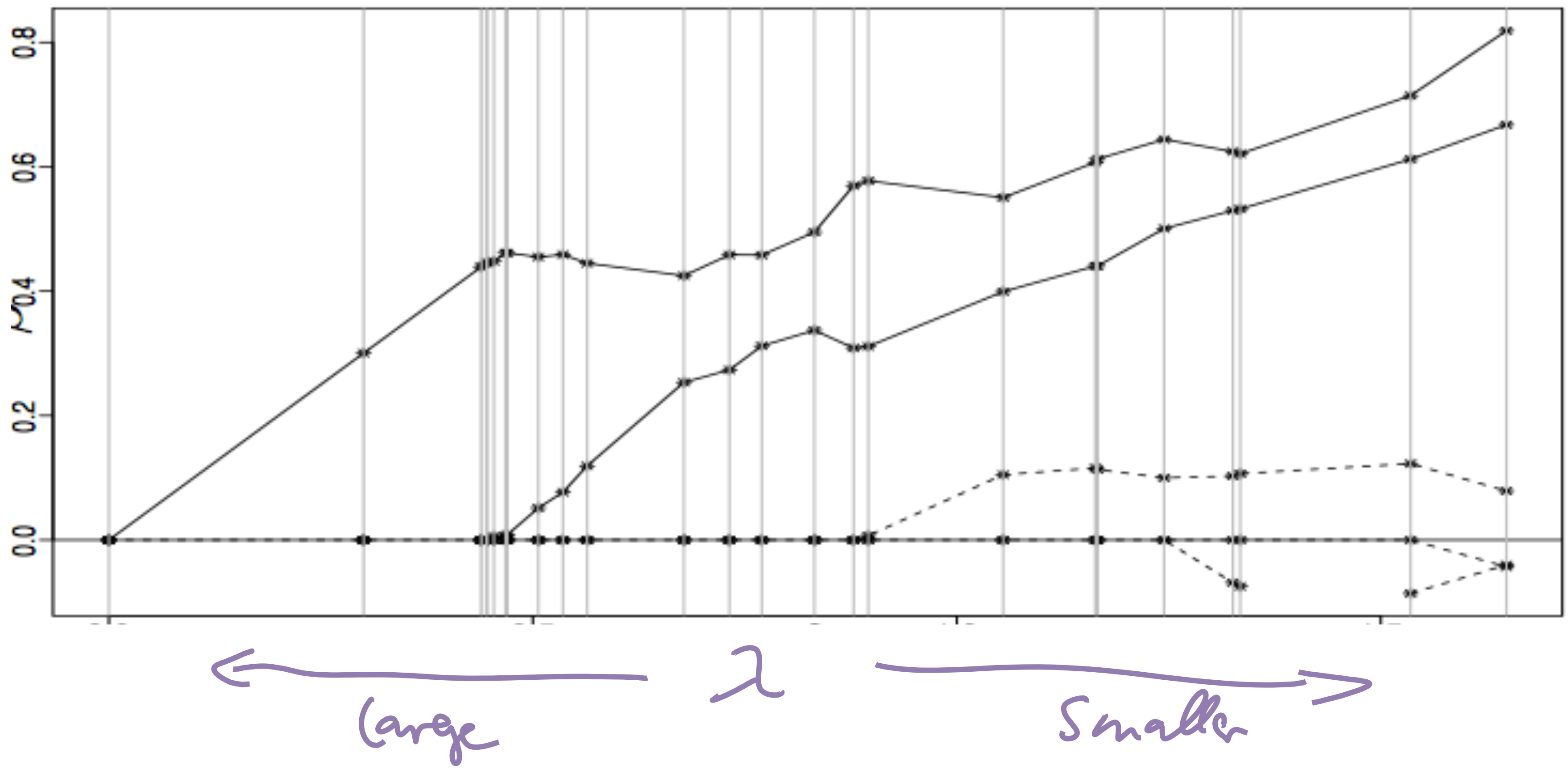
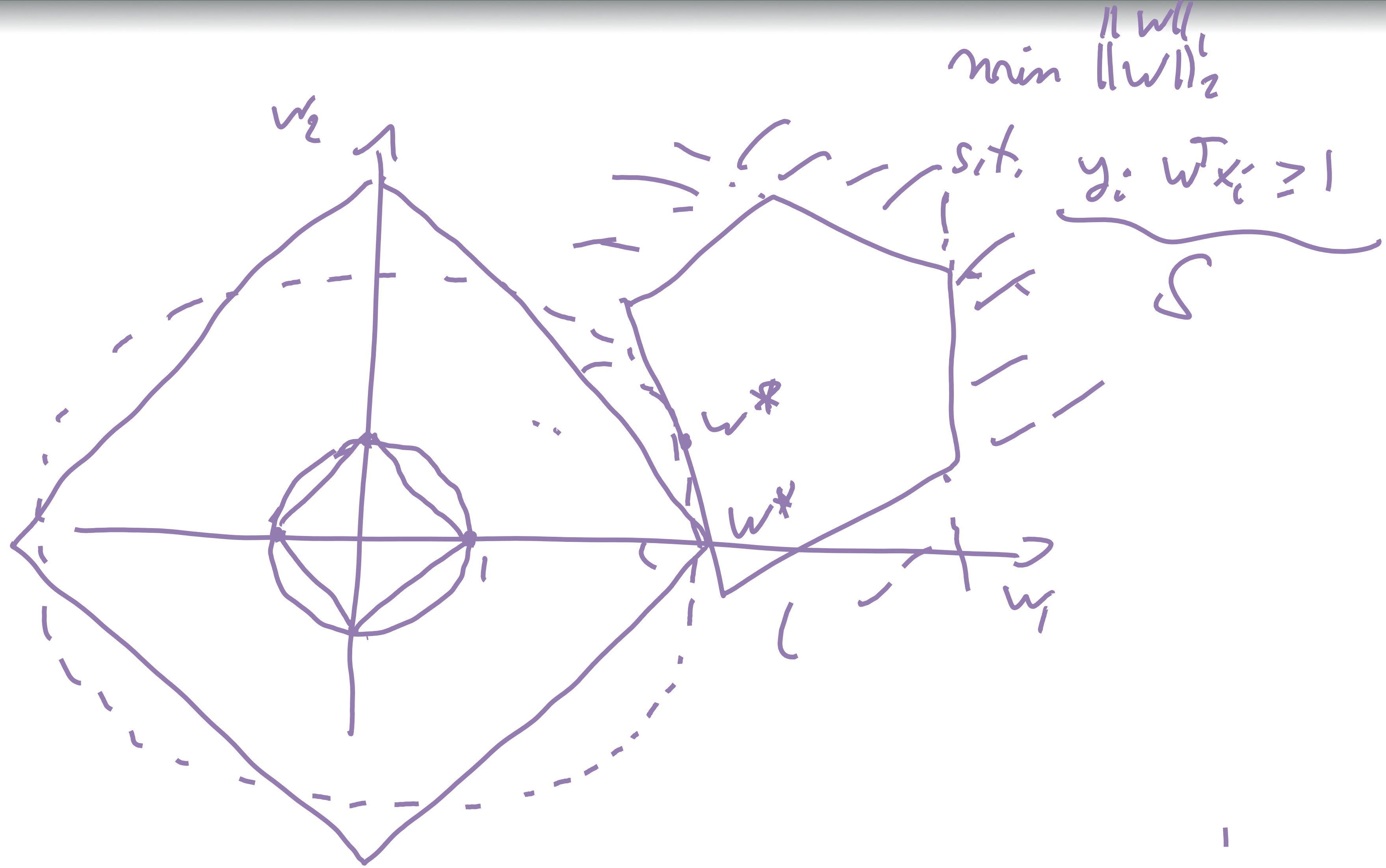


Illustration of l1-regularization



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

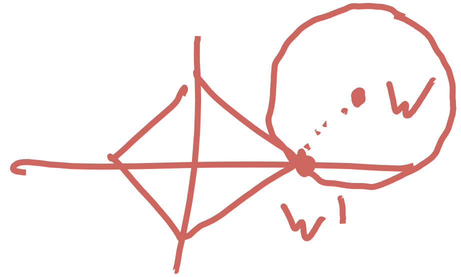
Online L1-SVM

- Can solve L1-SVM using online convex programming

$$\min_{w,b} \sum_i \max(0, 1 - y_i(w^T x_i + b)) \quad \text{s.t. } ||w||_1 \leq \frac{1}{\lambda}$$

S

- Subgradient:
  - calculation stays the same as in SVM!
- Reprojection:
  - Need to solve:

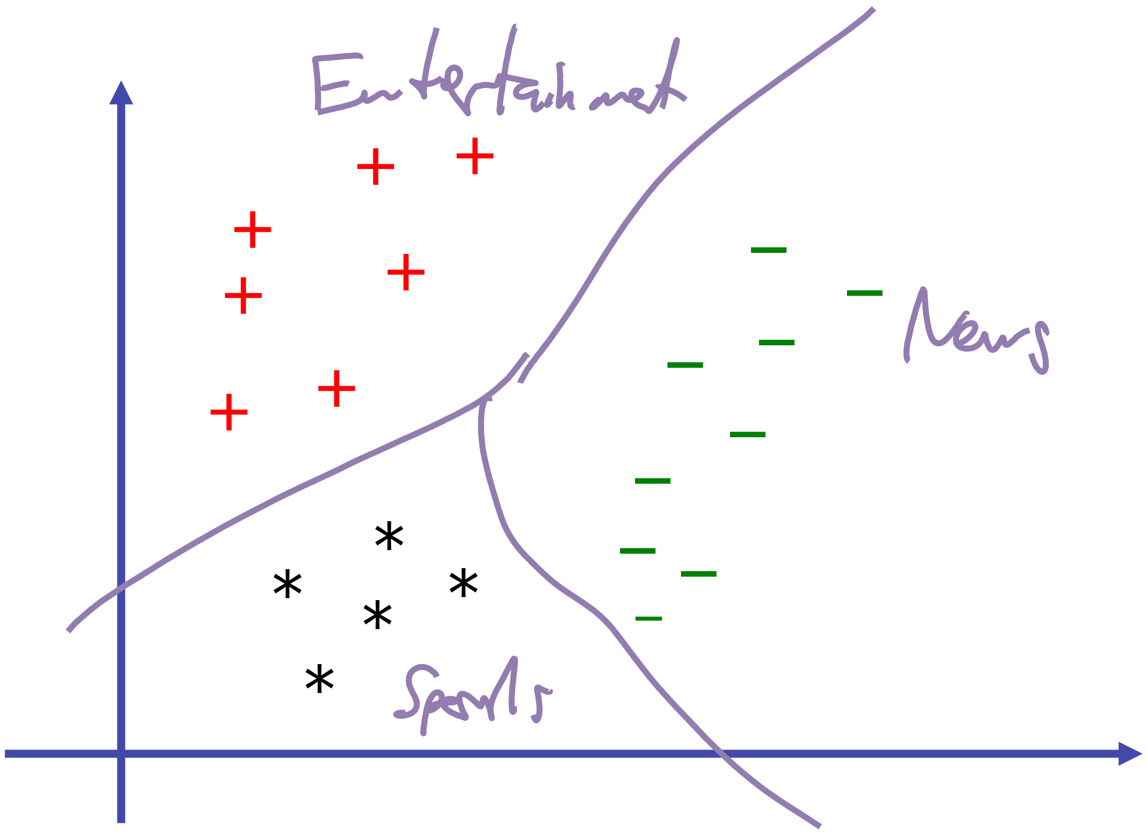


$$\text{Proj}_S(w) = \arg\min_{w' \in S} ||w - w'||_2$$

More results on supervised learning

- Feature selection
- **Dealing with multiple classes**
- Regression
- Nonlinear methods

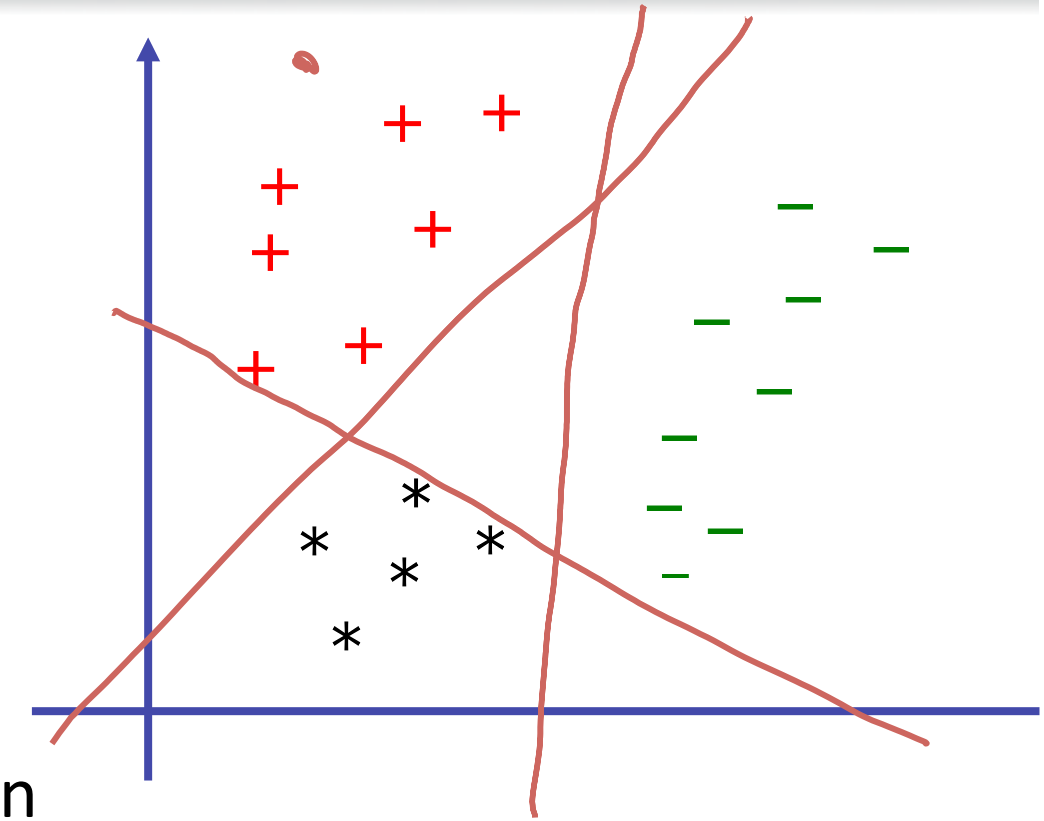
Dealing with multiple classes



One-vs-all

$y_e \cdot w_e^T x_i + b_e$

- Solve c SVMs, one for each class
  - Positive examples: all points from class l
  - Negative examples all other points
- Classify using the SVM with largest margin
- Problems?



- Ideally want to optimize all SVMs at the same time

Multi-class SVM

$\|w_y\|_2^2$

$$\min_{w,b,\xi} \sum_y w_{(y)}^T w_{(y)} + C \sum_i \xi_i$$

$$\text{s.t. } w_{(y_i)}^T x_i + b_{(y_i)} \geq w_{(y')}^T x_i + b_{(y')} + 1 - \xi_j$$

- Can be solved using same techniques as single-class SVM
- Multi-class hinge loss:

$$\ell(W; (\mathbf{x}, y)) = \max_{r \in [k] \setminus \{y\}} [1 - (W\mathbf{x})_y + (W\mathbf{x})_r]_+$$

z

$\max(1 - z, 0)$

More results on supervised learning

- Feature selection
- Dealing with multiple classes
- **Regression**
- Nonlinear methods

Regression

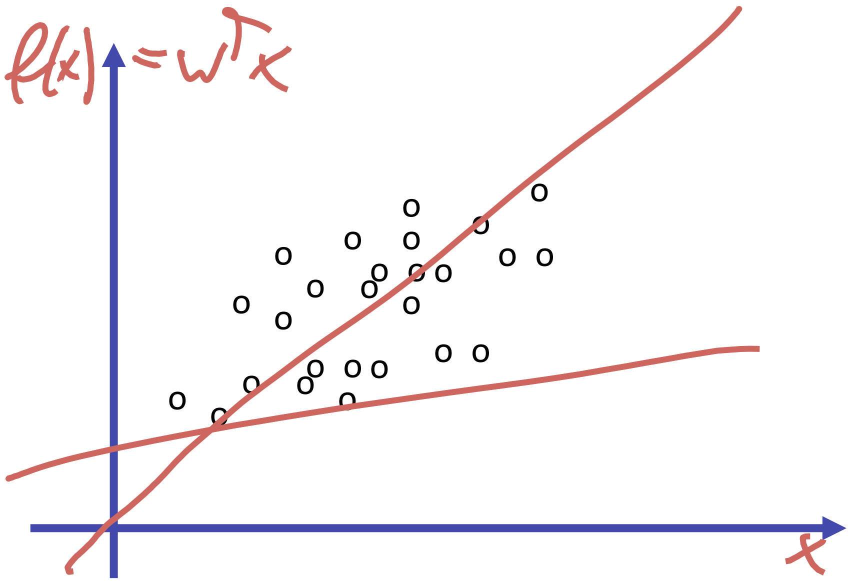
- So far, our goal was to predict a discrete label
- In many problems, we need to predict a real-valued output

$$y = f(x; w) + noise$$

- E.g.:
  - Predict grade based on #homeworks solved
  - Predict flight delay at one airport given delays at other airports
  - …

Linear regression

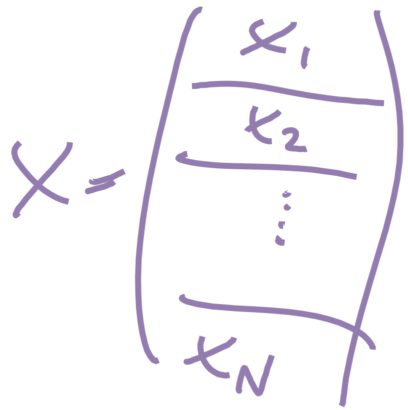
- Given $(x_1, y_1), \ldots, (x_n, y_n)$
- Assume: $y_i = w^T x_i + noise$
- To optimize *w* need to quantify goodness of fit



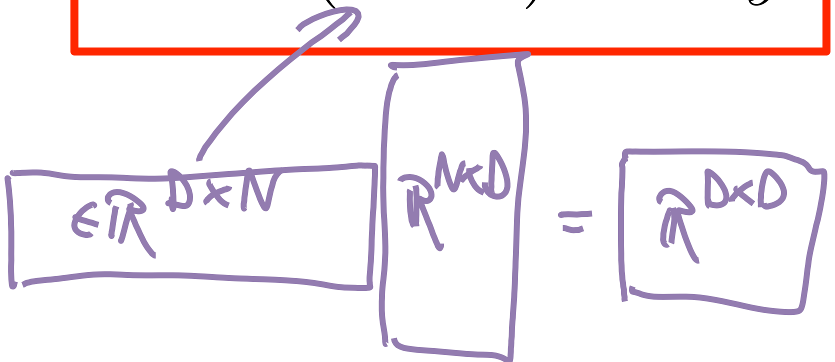
Square loss

- Want to solve

$$w^* = \arg\min_w \sum_{i=1}^{n} (y_i - w^T x_i)^2$$



- Closed form solution: $w^* = (X^T X)^{-1} X^T y$



- Complexity? $O(D^3)$
- Intractable for large # of dimensions!
- Will see how we can efficiently compute with OCP!