CS60021: Scalable Data Mining

Large Scale Machine Learning

Sourangshu Bhattacharya

Supervised Learning

- **Example: Spam filtering**

	viagra	learning	the	dating	nigeria	*spam?*
$\vec{x}_1 = ($	1	0	1	0	0)	$y_1 = 1$
$\vec{x}_2 = ($	0	1	1	0	0)	$y_2 = -1$
$\vec{x}_3 = ($	0	0	0	0	1)	$y_3 = 1$

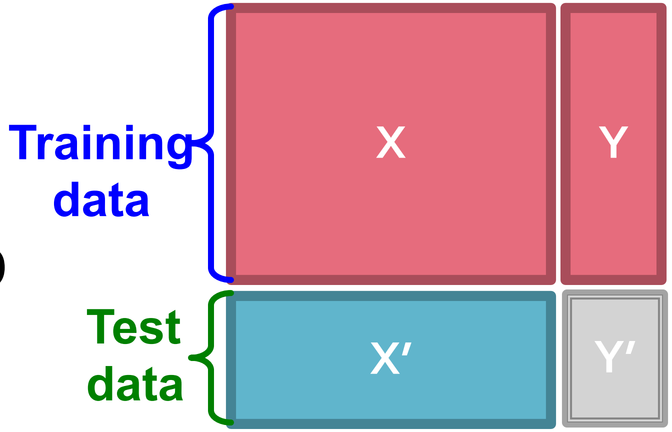
- **Instance space x ∈ X** (|**X**|= **n** data points)
  - **Binary or real-valued feature vector *x* of word occurrences**
  - ***d*** features (words + other things, **d**~100,000)
- **Class y ∈ Y**
  - ***y***: Spam (+1), Ham (-1)
- **Goal: Estimate** a function **f(x)** so that ***y = f(x)***

More generally: Supervised Learning

- **Would like to do prediction:**
  estimate a function **f(x)** so that ***y = f(x)***
- **Where *y* can be:**
  - **Real number**: Regression
  - **Categorical: Classification**
  - **Complex object:**
    - Ranking of items, Parse tree, etc.
- **Data is labeled:**
  - Have many pairs **{(x, y)}**
    - **x** … vector of binary, categorical, real valued features
    - **y** … class ({+1, -1}, or a real number)

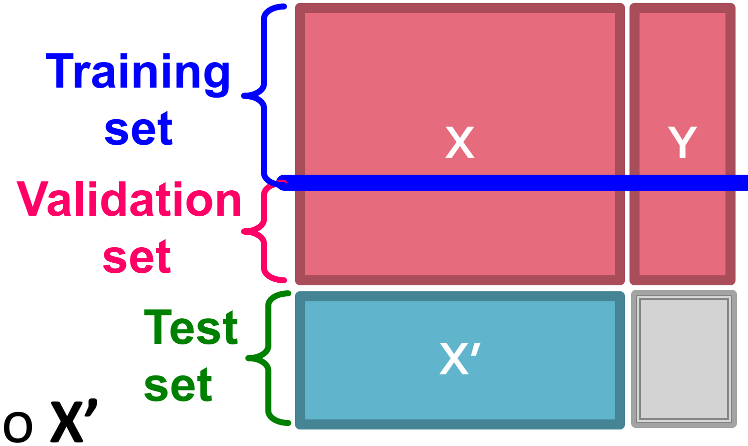
Supervised Learning

- **Task:** Given data **(X,Y)** build a model **f()** to predict **Y'** based on **X'**
- **Strategy:** **Estimate $y = f(x)$ on (X, Y). Hope that the same $f(x)$ also works to predict unknown Y'**
  - The **"hope"** is called **generalization**
    - **Overfitting:** **If f(x) predicts well Y but is unable to predict Y'**
  - **We want to build a model that generalizes well to unseen data**
    - But Jure, how can we well on data we have never seen before?!?



Supervised Learning

- **Idea: Pretend we do not know the data/labels we actually do know**
  - Build the model **f(x)** on the training data
    See how well **f(x)** does on the test data
    - If it does well, then apply it also to **X'**



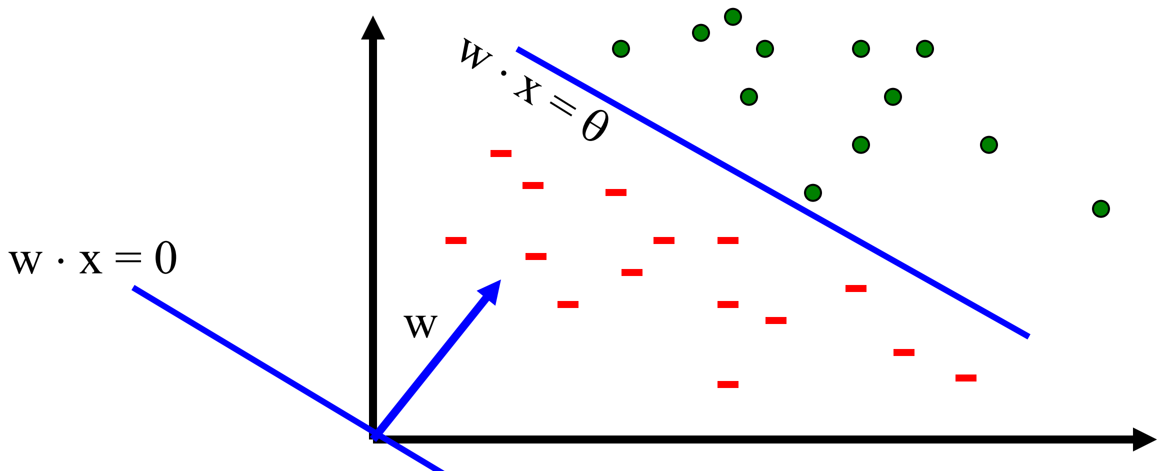
- **Refinement: Cross validation**
  - Splitting into training/validation set is brutal
  - Let's split our data **(X,Y)** into 10-folds (buckets)
  - Take out 1-fold for validation, train on remaining 9
  - Repeat this 10 times, report average performance

Linear models for classification

- **Binary classification:**

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{w}^{(1)}\mathbf{x}^{(1)} + \mathbf{w}^{(2)}\mathbf{x}^{(2)} + \ldots \mathbf{w}^{(d)}\mathbf{x}^{(d)} \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

- **Input:** Vectors $\boldsymbol{x}_j$ and labels $\boldsymbol{y}_j$
  - Vectors $\boldsymbol{x}_j$ are real valued where $\|x\|_2 = 1$
- **Goal:** Find vector $w = (w^{(1)}, w^{(2)}, \ldots, w^{(d)})$
  - Each $\boldsymbol{w}^{(i)}$ is a real number



Decision boundary is **linear**

Note:

$$\mathbf{x} \rightarrow \langle \mathbf{x}, 1 \rangle \quad \forall \mathbf{x}$$

$$\mathbf{w} \rightarrow \langle \mathbf{w}, -\theta \rangle$$

SVM: How to estimate *w*?

$$\min_{w,b} \frac{1}{2} w \cdot w + C \cdot \sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, y_i \cdot (x_i \cdot w + b) \geq 1 - \xi_i$$

- **Want to estimate *w* and *b*!**
  - **Standard way:** Use a solver!
    - **Solver:** software for finding solutions to "common" optimization problems
- **Use a quadratic solver:**
  - Minimize quadratic function
  - Subject to linear constraints
- **Problem:** Solvers are inefficient for big data!

SVM: How to estimate *w*?

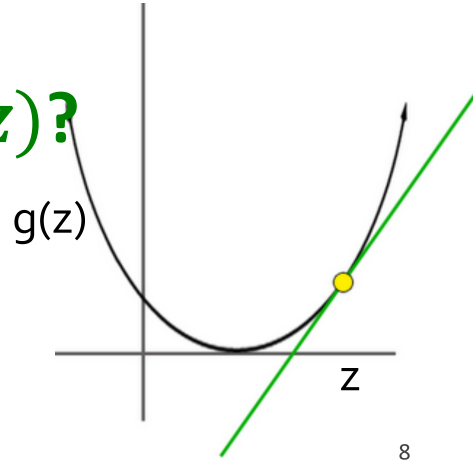
- **Want to estimate w, b!**
- **Alternative approach:**
  - **Want to minimize *f(w,b)*:**

$$\min_{w,b} \tfrac{1}{2} w \cdot w + C \sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, y_i \cdot (x_i \cdot w + b) \geq 1 - \xi_i$$

$$f(w,b) = \tfrac{1}{2} w \cdot w + C \cdot \sum_{i=1}^{n} \max\left\{0, 1 - y_i \left(\sum_{j=1}^{d} w^{(j)} x_i^{(j)} + b\right)\right\}$$

- **Side note:**
  - **How to minimize convex functions $g(z)$?**
  - Use gradient descent: **$\min_z g(z)$**
  - Iterate: **$z_{t+1} \leftarrow z_t - \eta \nabla g(z_t)$**



What is Optimization?

Find the minimum or maximum of an objective function given a set of constraints:

$$\arg\min_{x} f_0(x)$$
$$\text{s.t.}\, f_i(x) \leq 0, i = \{1, \ldots, k\}$$
$$h_j(x) = 0, j = \{1, \ldots l\}$$

Why Do We Care?

Linear Classification

Maximum Likelihood

$$\arg\min_{w} \sum_{i=1}^{n} ||w||^2 + C \sum_{i=1}^{n} \xi_i$$

$$\text{s.t. } 1 - y_i x_i^T w \leq \xi_i$$

$$\xi_i \geq 0$$

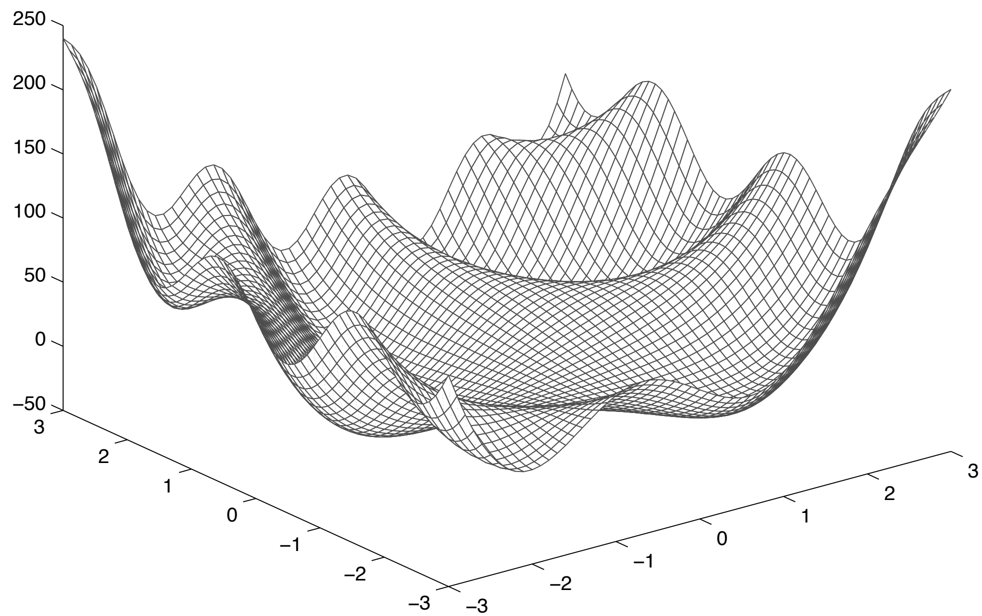
$$\arg\max_{\theta} \sum_{i=1}^{n} \log \, p_\theta(x_i)$$

K-Means

$$\arg \min_{\mu_1, \mu_2, \ldots, \mu_k} J(\mu) = \sum_{j=1}^{k} \sum_{i \in C_j} ||x_i - \mu_j||^2$$

Prefer Convex Problems

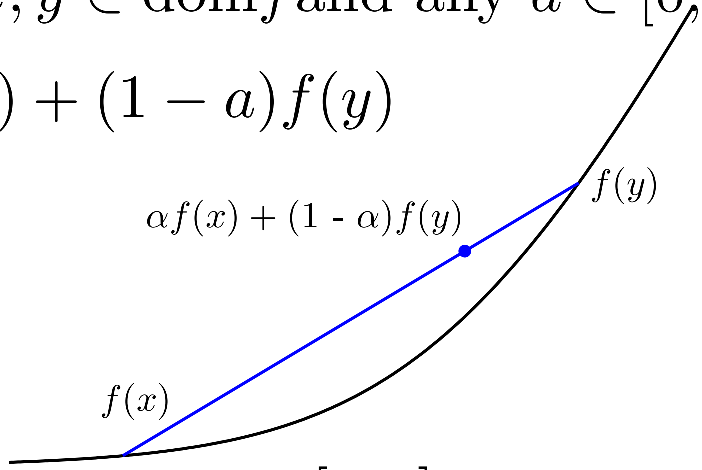
Local (non global) minima and maxima:



Convex Functions and Sets

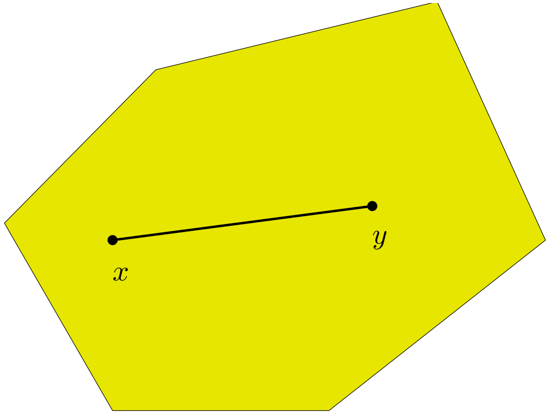
A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if for $x, y \in \text{dom} f$ and any $a \in [0, 1]$,

$$f(ax + (1-a)y) \leq af(x) + (1-a)f(y)$$



A set $C \subseteq \mathbb{R}^n$ is convex if for $x, y \in C$ and any $a \in [0, 1]$,

$$ax + (1-a)y \in C$$



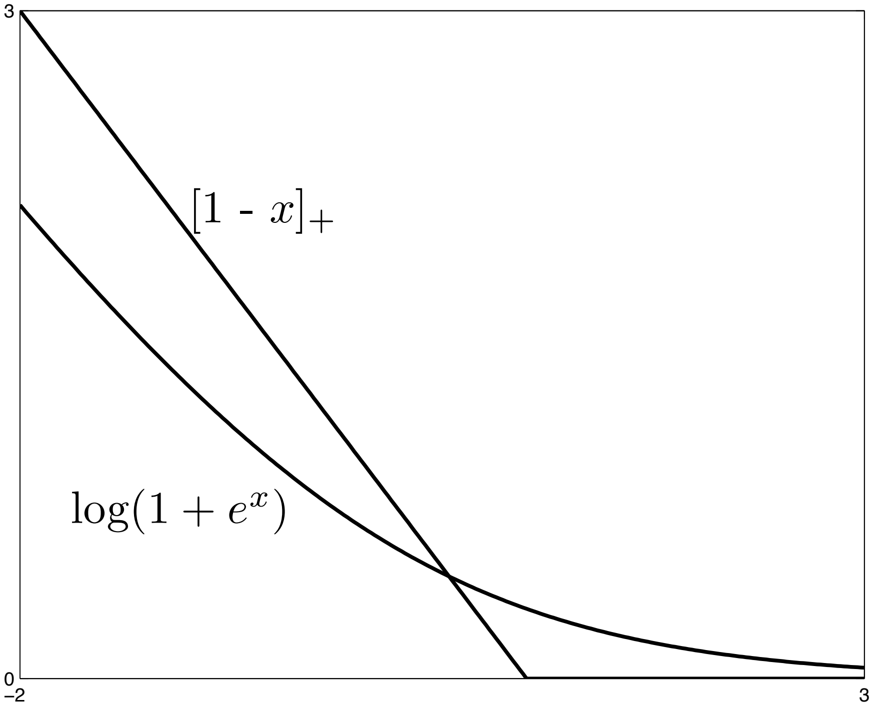
Important Convex Functions

SVM loss:

$$f(w) = \left[1 - y_i x_i^T w\right]_+$$

Binary logistic loss:

$$f(w) = \log\left(1 + \exp(-y_i x_i^T w)\right)$$



Convex Optimization Problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x) && \text{(Convex function)} \\ \text{s.t.} \quad & f_i(x) \leq 0 && \text{(Convex sets)} \\ & h_j(x) = 0 && \text{(Affine)} \end{aligned}$$

Lagrangian Dual

Start with optimization problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \;& f_0(x) \\ \text{s.t.} \;& f_i(x) \leq 0,\; i = \{1, \ldots, k\} \\ & h_j(x) = 0,\; j = \{1, \ldots, l\} \end{aligned}$$

Form *Lagrangian* using Lagrange multipliers $\lambda_i \geq 0$, $\nu_i \in \mathbb{R}$

$$\mathcal{L}(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{k} \lambda_i f_i(x) + \sum_{j=1}^{l} \nu_j h_j(x)$$

Form *dual function*

$$g(\lambda, \nu) = \inf_x \mathcal{L}(x, \lambda, \nu) = \inf_x \left\{ f_0(x) + \sum_{i=1}^{k} \lambda_i f_i(x) + \sum_{j=1}^{l} \nu_j h_j(x) \right\}$$

Gradient Descent

The simplest algorithm in the world (almost). Goal:

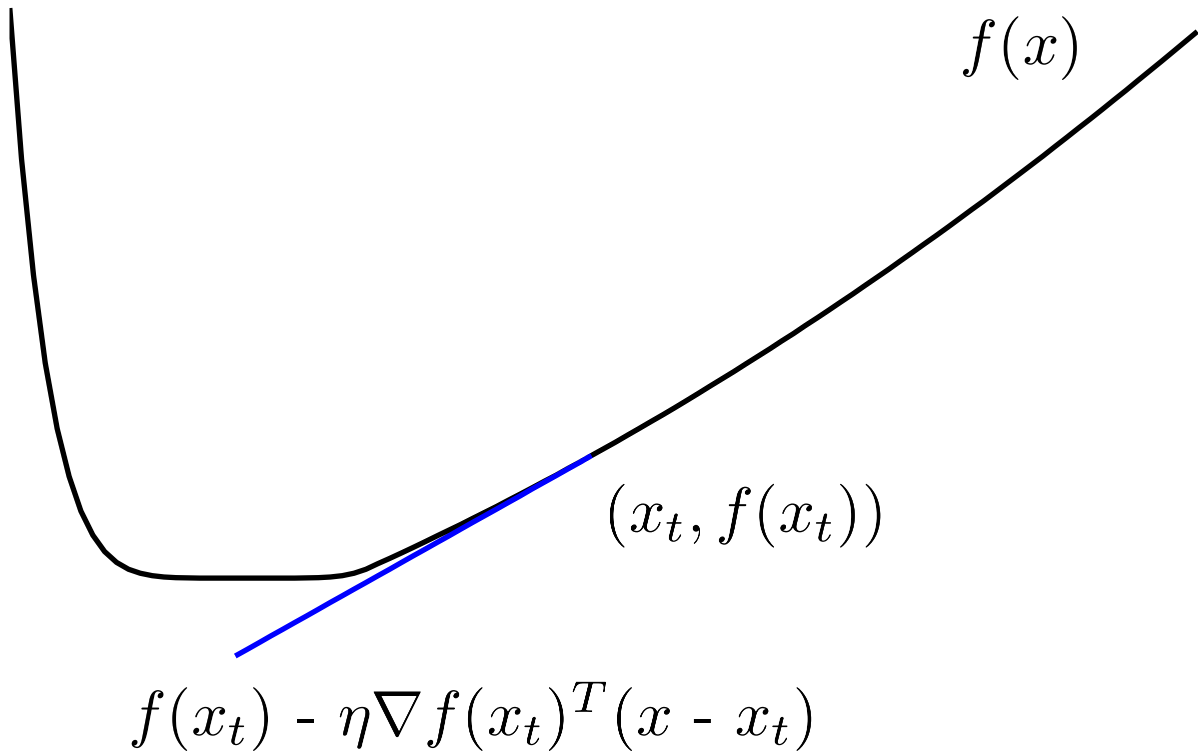
$$\underset{x}{\text{minimize}} \;\; f(x)$$

Just iterate

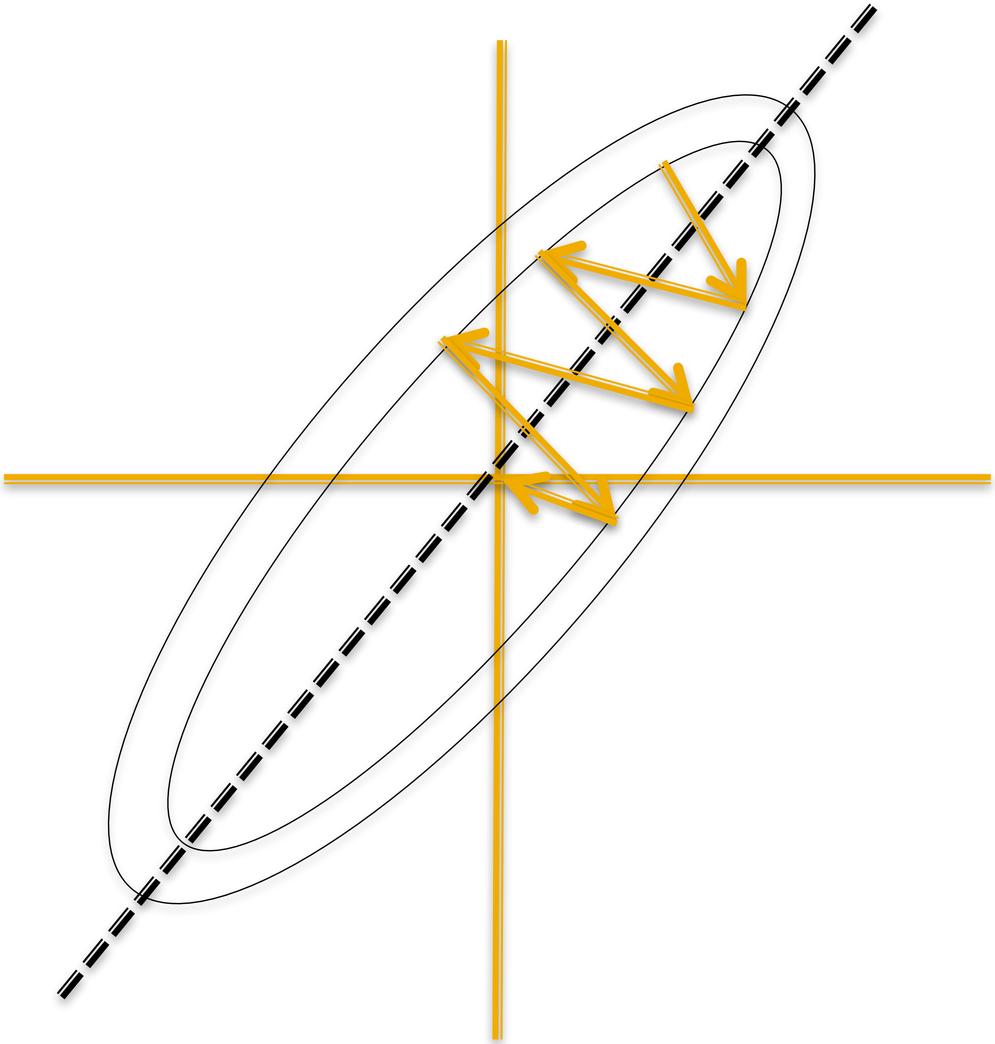
$$x_{t+1} = x_t - \eta_t \nabla f(x_t)$$

where η_t is stepsize.

Single Step Illustration



Full Gradient Descent Illustration



Newton's Method

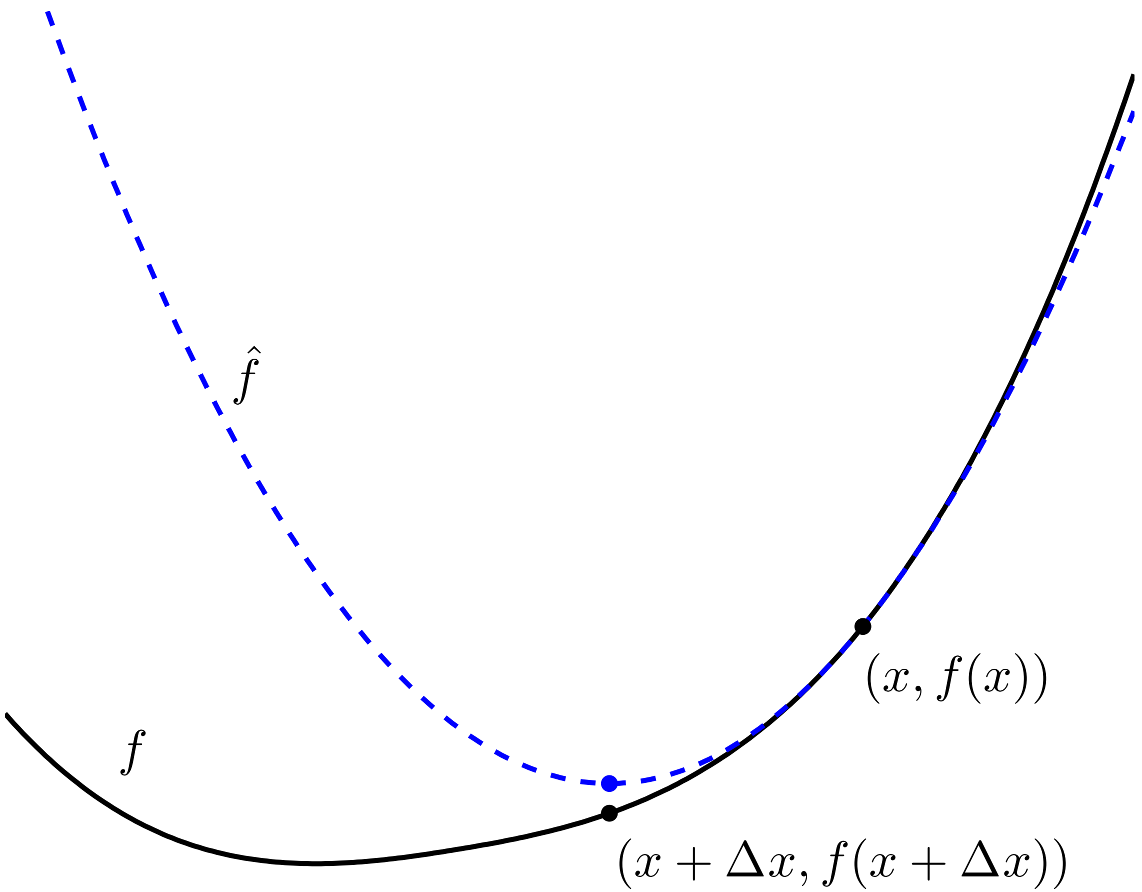
Idea: use a second-order approximation to function.

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x + \frac{1}{2} \Delta x^T \nabla^2 f(x) \Delta x$$

Choose Δx to minimize above:

$$\Delta x = - \underbrace{\left[\nabla^2 f(x)\right]^{-1}}_{\text{Inverse Hessian}} \underbrace{\nabla f(x)}_{\text{Gradient}}$$

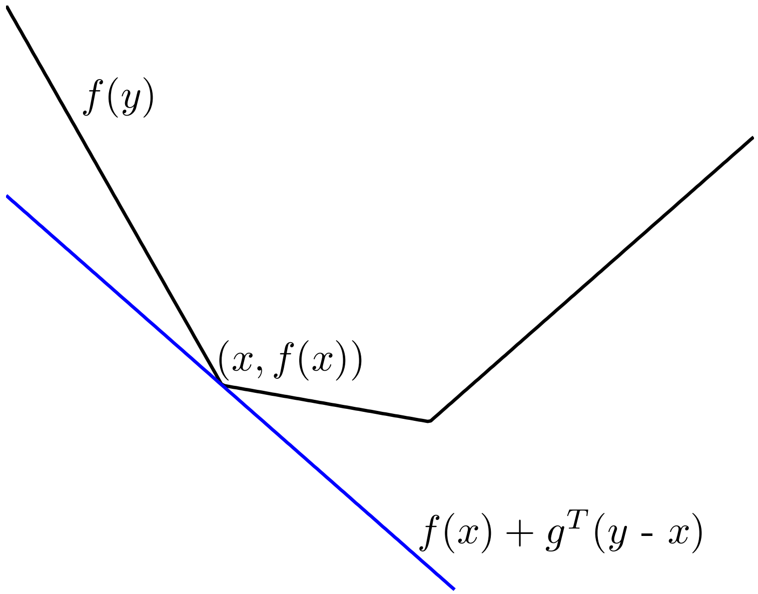
Newton's Method Picture



$\hat{f}$ is 2^{nd}-order approximation, f is true function.

Subgradient Descent Motivation

Lots of non-differentiable convex functions used in machine learning:



The *subgradient set*, or subdifferential set, $\partial f(x)$ of f at x is

$$\partial f(x) = \left\{ g : f(y) \geq f(x) + g^T(y - x) \ \text{ for all } y \right\}.$$

Subgradient Descent – Algorithm

Really, the simplest algorithm in the world. Goal:

$$\underset{x}{\text{minimize}} \;\; f(x)$$

Just iterate

$$x_{t+1} = x_t - \eta_t g_t$$

where η_t is a stepsize, $g_t \in \partial f(x_t)$.

Online learning and optimization

- Goal of machine learning :
  - Minimize expected loss

$$\min_h L(h) = \mathbf{E}\left[\text{loss}(h(x), y)\right]$$

given samples $(x_i, y_i) \; i = 1, 2 \ldots m$

- This is Stochastic Optimization
  - Assume loss function is convex

Batch (sub)gradient descent for ML

- Process all examples together in each step

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \left(\frac{1}{n} \sum_{i=1}^{n} \frac{\partial L(w, x_i, y_i)}{\partial w} \right)$$

where L is the regularized loss function

- Entire training set examined at each step
- Very slow when *n* is very large

Stochastic (sub)gradient descent

- "Optimize" one example at a time
- Choose examples randomly (or reorder and choose in order)
  - Learning representative of example distribution

for $i = 1$ to n:

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \frac{\partial L(w, x_i, y_i)}{\partial w}$$

where L is the regularized loss function

Stochastic (sub)gradient descent

$$\text{for } i = 1 \text{ to } n\text{:}$$
$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \frac{\partial L(w, x_i, y_i)}{\partial w}$$

$$\text{where } L \text{ is the regularized loss function}$$

- Equivalent to online learning (the weight vector *w* changes with every example)
- Convergence guaranteed for convex functions (to local minimum)

SVM: How to estimate *w*?

- **Gradient descent:**

Iterate until convergence:

- For j = 1 … d
  - Evaluate: $\nabla f^{(j)} = \frac{\partial f(w,b)}{\partial w^{(j)}} = w^{(j)} + C\sum_{i=1}^{n}\frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$
  - Update:

    $w^{(j)} \leftarrow w^{(j)} - \eta \nabla f^{(j)}$

η…learning rate parameter
C… regularization parameter

- **Problem:**
  - **Computing $\nabla f^{(j)}$ takes O(n) time!**
    - **n** … size of the training dataset

SVM: How to estimate w?

We just had:

$$\nabla f^{(j)} = w^{(j)} + C\sum_{i=1}^{n}\frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

- **Stochastic Gradient Descent**
  - Instead of evaluating gradient over all examples evaluate it for each **individual** training example

$$\nabla f^{(j)}(x_i) = w^{(j)} + C \cdot \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

Notice: no summation over ***i*** anymore

- **Stochastic gradient descent:**

Iterate until convergence:
- **For i = 1 … n**
  - **For j = 1 … d**
    - **Compute: $\nabla f^{(j)}(x_i)$**
    - **Update: $w^{(j)} \leftarrow w^{(j)} - \eta \nabla f^{(j)}(x_i)$**

SGD - Issues

- Convergence very sensitive to learning rate (η_t) (oscillations near solution due to probabilistic nature of sampling)
  - Might need to decrease with time to ensure the algorithm converges eventually
- Basically – SGD good for machine learning with large data sets!

Hybrid!

- Stochastic – 1 example per iteration
- Batch – All the examples!
- Sample Average Approximation (SAA):
  - Sample *m* examples at each step and perform SGD on them
- Allows for parallelization, but choice of *m* based on heuristics

Example: Text categorization

- **Example by Leon Bottou:**
  - **Reuters RCV1** document corpus
    - Predict a category of a document
      - One **vs.** the rest classification
  - ***n* = 781,000** training examples (documents)
  - 23,000 test examples
  - ***d* = 50,000** features
    - One feature per word
    - Remove stop-words
    - Remove low frequency words

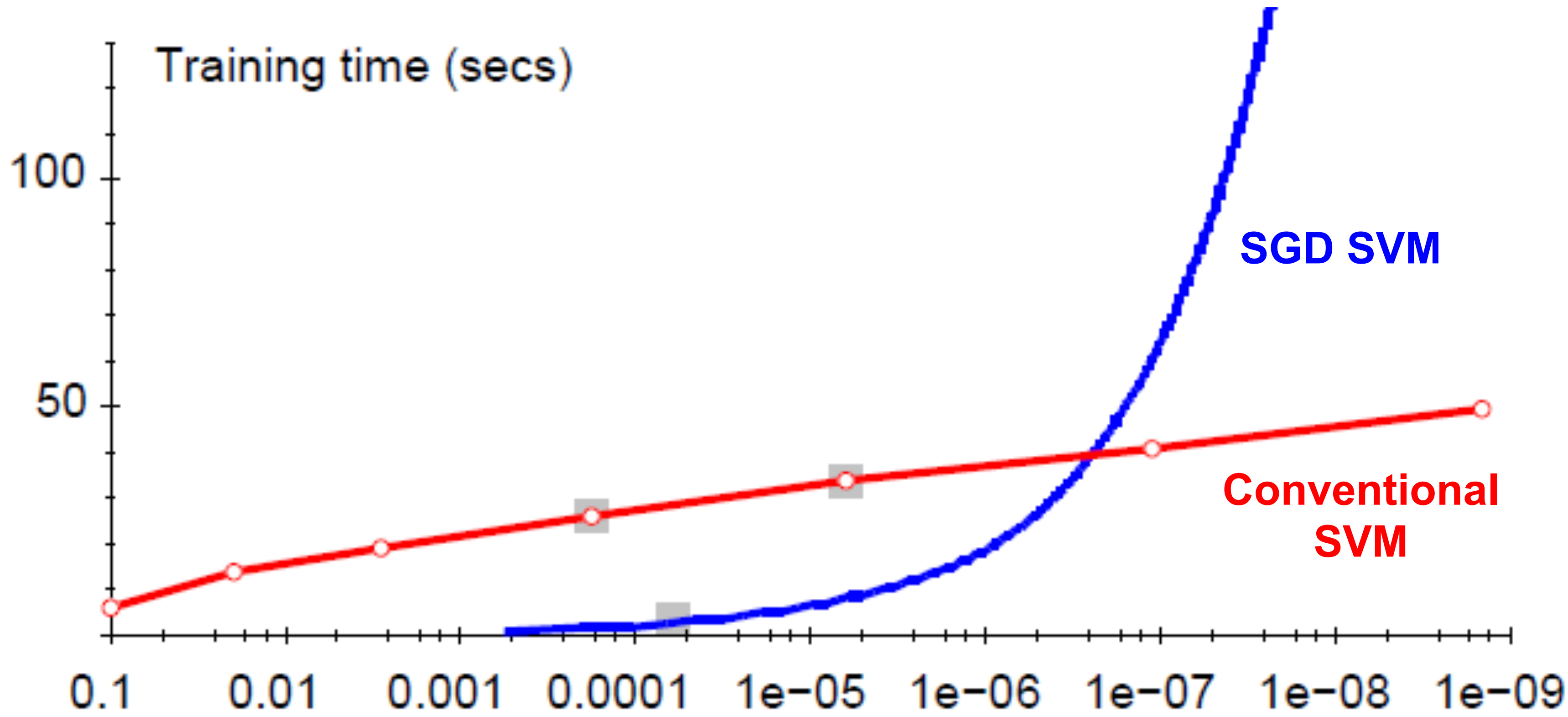
Example: Text categorization

- **Questions:**
  - **(1)** Is **SGD** successful at minimizing ***f(w,b)***?
  - **(2)** How quickly does **SGD** find the min of ***f(w,b)***?
  - **(3)** What is the error on a test set?

	Training time	*Value of f(w,b)*	*Test error*
Standard SVM	23,642 secs	0.2275	6.02%
"Fast SVM"	66 secs	0.2278	6.03%
SGD SVM	1.4 secs	0.2275	6.02%

(1) SGD-SVM is successful at minimizing the value of ***f(w,b)***
(2) SGD-SVM is super fast
(3) SGD-SVM test set error is comparable

Optimization “Accuracy”

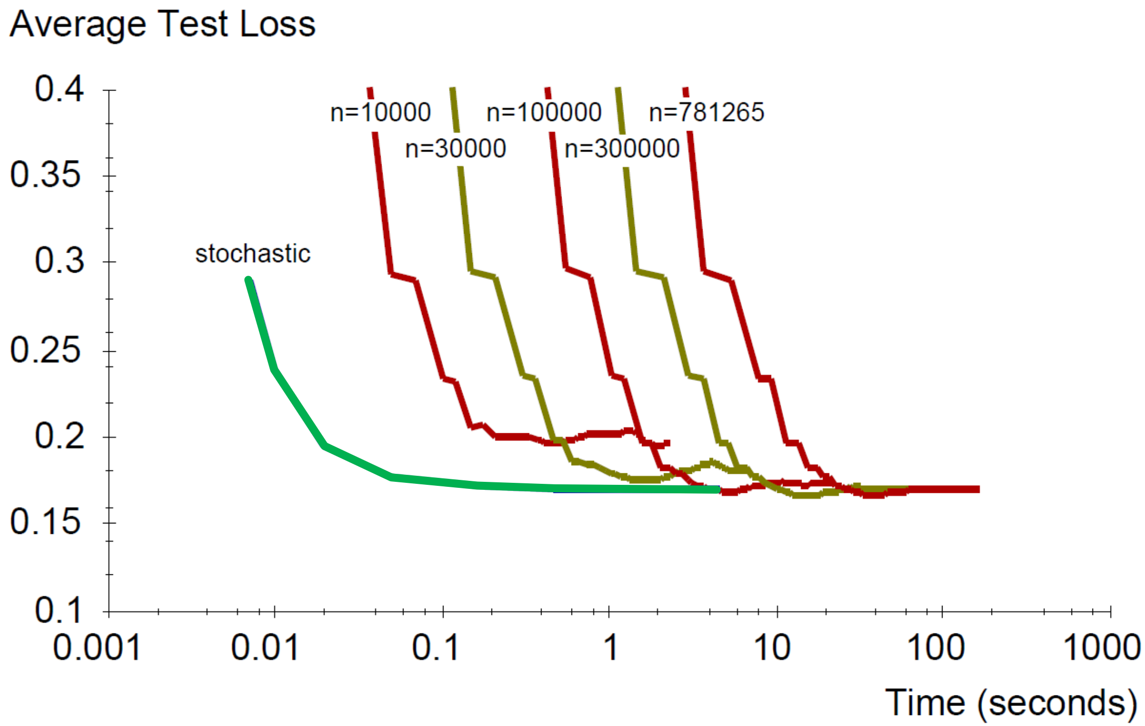


Optimization quality: | *f(w,b)* – *f* (*w*^{opt},*b*^{opt}) |

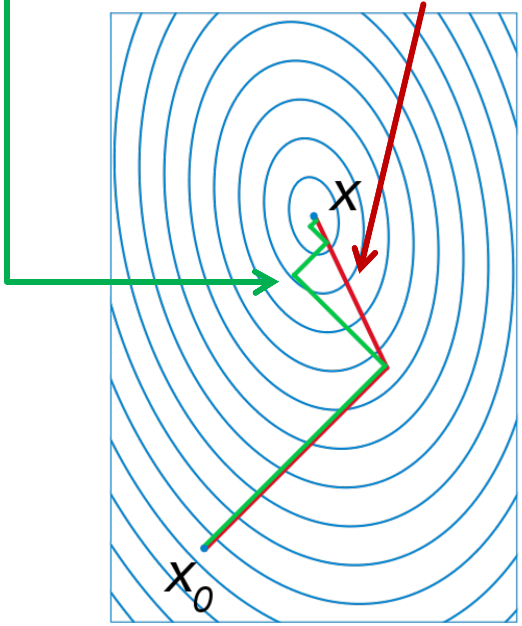
For optimizing *f(w,b) within reasonable* quality
SGD-SVM is super fast

SGD vs. Batch Conjugate Gradient

- **SGD** on full dataset vs. **Conjugate Gradient** on a sample of ***n*** training examples



Theory says: Gradient descent converges in linear time k. **Conjugate gradient** converges in $\sqrt{k}$.



k… condition number

Bottom line: Doing a simple (but fast) SGD update many times is better than doing a complicated (but slow) CG update a few times

Practical Considerations

- **Need to choose learning rate η and t_0**

$$w_{t+1} \leftarrow w_t - \frac{\eta_t}{t+t_0}\left(w_t + C\frac{\partial L(x_i, y_i)}{\partial w}\right)$$

- **Leon suggests:**
  - Choose $\mathbf{t_0}$ so that the expected initial updates are comparable with the expected size of the weights
  - Choose η:
    - Select a **small subsample**
    - Try various rates η (e.g., 10, 1, 0.1, 0.01, …)
    - Pick the one that most reduces the cost
    - Use η for next 100k iterations on the full dataset

Practical Considerations

- **Sparse Linear SVM:**
  - **Feature vector x_i is sparse (contains many zeros)**
    - Do not do: $\boldsymbol{x_i} = [0,0,0,1,0,0,0,0,5,0,0,0,0,0,0,\ldots]$
    - But represent $\mathbf{x_i}$ as a sparse vector $\mathbf{x_i}$**=[*(4,1)*, *(9,5)*, …]**
  - **Can we do the SGD update more efficiently?**

$$w \leftarrow w - \eta\left(w + C\frac{\partial L(x_i,y_i)}{\partial w}\right)$$

  - **Approximated in 2 steps:**

$$w \leftarrow w - \eta C\frac{\partial L(x_i,y_i)}{\partial w}$$

cheap: $\boldsymbol{x_i}$ is sparse and so few coordinates ***j*** of ***w*** will be updated

$$w \leftarrow w(1-\eta)$$

expensive: ***w*** is not sparse, all coordinates need to be updated

Practical Considerations

- **Solution 1:** $\boldsymbol{w} = \boldsymbol{s} \cdot \boldsymbol{v}$
  - Represent vector ***w*** as the product of scalar ***s*** and vector ***v***
  - Then the update procedure is:
    - *(1)* $\boldsymbol{v} = \boldsymbol{v} - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$
    - *(2)* $\boldsymbol{s} = \boldsymbol{s}(1 - \eta)$
- **Solution 2:**
  - Perform only step **(1)** for each training example
  - Perform step **(2)** with lower frequency and higher η

Two step update procedure:

(1) $w \leftarrow w - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$

(2) $w \leftarrow w(1 - \eta)$

Practical Considerations

- **Stopping criteria:**
  How many iterations of SGD?
  - **Early stopping with cross validation**
    - Create a validation set
    - Monitor cost function on the validation set
    - Stop when loss stops decreasing
  - **Early stopping**
    - Extract two disjoint subsamples **A** and **B** of training data
    - Train on **A**, stop by validating on **B**
    - Number of epochs is an estimate of ***k***
    - Train for ***k*** epochs on the full dataset

Stochastic gradient descent

- Reference: http://alex.smola.org/teaching/10-701-15/math.html
- Given dataset $D = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
- Loss function: $L(\theta, D) = \frac{1}{N}\sum_{i=1}^{N} l(\theta; x_i, y_i)$
- For linear models: $l(\theta; x_i, y_i) = l(y_i, \theta^T \phi(x_i))$
- Assumption D is drawn IID from some distribution $\mathcal{P}$.
- Problem:

$$\min_{\theta} L(\theta, D)$$

Stochastic gradient descent

- Input: D
- Output: $\bar{\theta}$

Algorithm:

- Initialize θ^0
- For $t = 1, \dots, T$
$$\theta^{t+1} = \theta^t - \eta_t \nabla_\theta l(y_t, \theta^T \phi(x_t))$$
- $\bar{\theta} = \frac{\sum_{t=1}^T \eta_t \theta^t}{\sum_{t=1}^T \eta_t}$.

SGD convergence

- Expected loss: $s(\theta) = E_{\mathcal{P}}[l(y, \theta^T \phi(x)]$
- Optimal Expected loss: $s^* = s(\theta^*) = \min_{\theta} s(\theta)$
- Convergence:

$$E_{\bar{\theta}}[s(\bar{\theta})] - s^* \leq \frac{R^2 + L^2 \sum_{t=1}^{T} \eta_t^2}{2 \sum_{t=1}^{T} \eta_t}$$

- Where: $R = \|\theta^0 - \theta^*\|$
- $L = \max \nabla l(y, \theta^T \phi(x))$

SGD convergence proof

- Define $r_t = \|\theta^t - \theta^*\|$ and $g_t = \nabla_\theta l(y_t, \theta^T \phi(x_t))$
- $r_{t+1}^2 = r_t^2 + \eta_t^2 \|g_t\|^2 - 2\eta_t (\theta^t - \theta^*)^T g_t$
- Taking expectation w.r.t $\mathcal{P}, \bar{\theta}$ and using $s^* - s(\theta^t) \geq g_t^T(\theta^* - \theta^t)$, we get: $E_{\bar{\theta}}[r_{t+1}^2 - r_t^2] \leq \eta_t^2 L^2 + 2\eta_t(s^* - E_{\bar{\theta}}[s(\theta^t)])$
- Taking sum over $t = 1, \dots, T$ and using

$$E_{\bar{\theta}}[r_{t+1}^2 - r_0^2] \leq L^2 \sum_{t=0}^{T-1} \eta_t^2 + 2 \sum_{t=0}^{T-1} \eta_t (s^* - E_{\bar{\theta}}[s(\theta^t)])$$

SGD convergence proof

- Using convexity of s:
$$\left(\sum_{t=0}^{T-1}\eta_t\right)E_{\bar{\theta}}[s(\bar{\theta})] \le E_{\bar{\theta}}[\sum_{t=0}^{T-1}\eta_t s(\theta^t)]$$
- Substituting in the expression from previous slide:
$$E_{\bar{\theta}}[r_{t+1}^2 - r_0^2]$$
$$\le L^2\sum_{t=0}^{T-1}\eta_t^2 + 2\sum_{t=0}^{T-1}\eta_t(s^* - E_{\bar{\theta}}[s(\bar{\theta})])$$
- Rearranging the terms proves the result.