CEng 793

On Machine Learning and Optimization

Sinan Kalkan

Optimization Methods for Large-Scale Machine Learning

Léon Bottou* Frank E. Curtis† Jorge Nocedal‡

June 16, 2016

Abstract

This paper provides a review and commentary on the past, present, and future of numerical optimization algorithms in the context of machine learning applications. Through case studies on text classification and the training of deep neural networks, we discuss how optimization problems arise in machine learning and what makes them challenging. A major theme of our study is that large-scale machine learning represents a distinctive setting in which the stochastic gradient (SG) method has traditionally played a central role while conventional gradient-based nonlinear optimization techniques typically falter. Based on this viewpoint, we present a comprehensive theory of a straightforward, yet versatile SG algorithm, discuss its practical behavior, and highlight opportunities for designing algorithms with improved performance. This leads to a discussion about the next generation of optimization methods for large-scale machine learning, including an investigation of two main streams of research on techniques that diminish noise in the stochastic directions and methods that make use of second-order derivative approximations.

Now

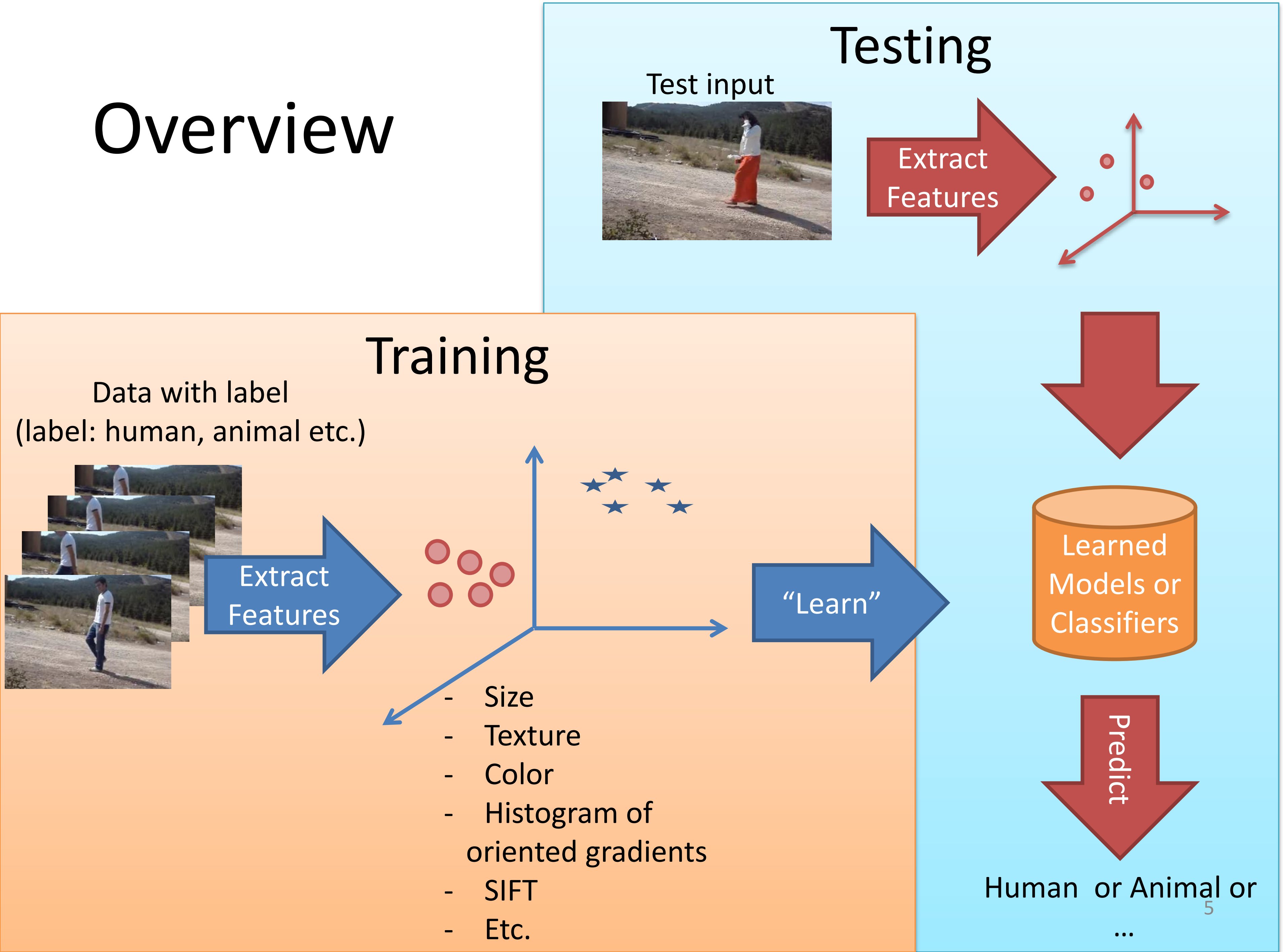
- Introduction to ML
  - Problem definition
  - Classes of approaches
  - K-NN
  - Support Vector Machines
  - Softmax classification / logistic regression
  - Parzen Windows

- Optimization
  - Gradient Descent approaches
  - A flavor of other approaches

Introduction to Machine learning

Uses many figures and material from the following website:
http://www.byclb.com/TR/Tutorials/neural_networks/ch1_1.htm

Overview



- Size
- Texture
- Color
- Histogram of oriented gradients
- SIFT
- Etc.

Content

- Problem definition
- General approaches
- Popular methods
  - kNN
  - Linear classification
  - Support Vector Machines
  - Parzen windows

Problem Definition

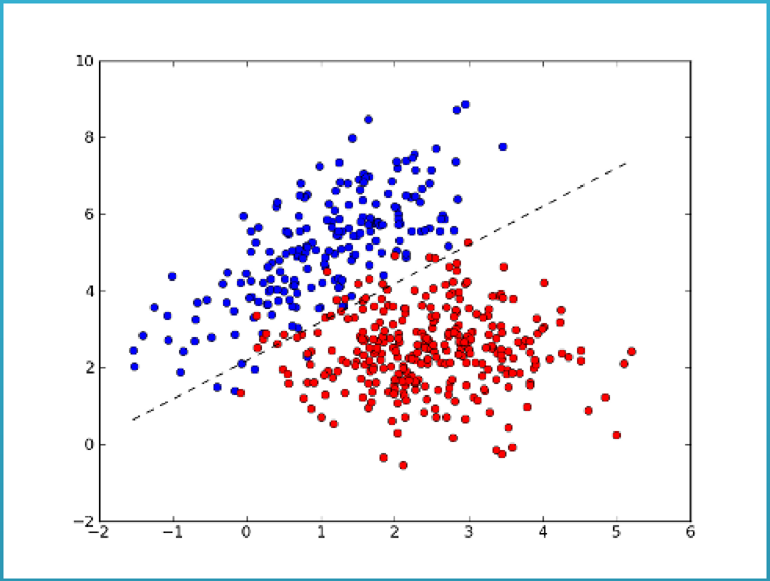
- Given
  - Data: a set of instances $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ sampled from a space $\mathbf{X} \in \mathbb{R}^d$.
  - Labels: the corresponding labels y_i for each $\mathbf{x}_i$. $y_i \in \mathbf{Y} \in \mathbb{R}$.
- Goal:
  - Learn a mapping from the space of “data” to the space of labels, i.e.,
    - $\mathcal{M}: \mathbf{X} \rightarrow \mathbf{Y}$.
    - $y = f(\mathbf{x})$.

Issues in Machine Learning

- Hypothesis space
- Loss / cost / objective function
- Optimization
- Bias vs. variance
- Test / evaluate
- Overfitting, underfitting

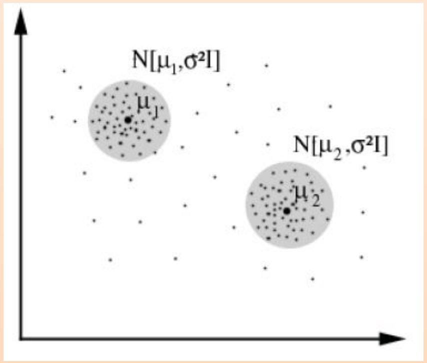
General Approaches

Discriminative

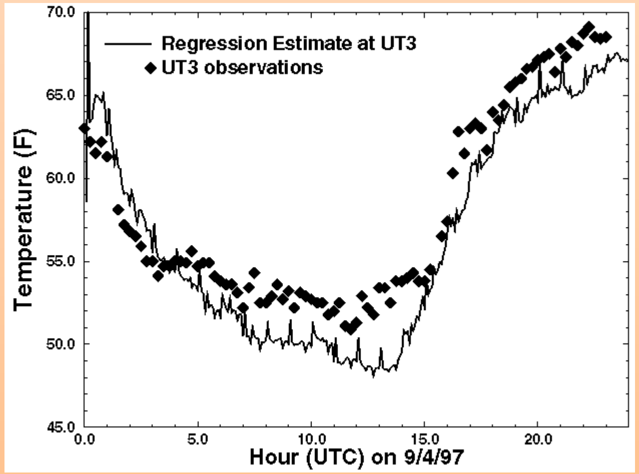


Find separating line
(in general: hyperplane)

Generative



Learn a model for
each class.

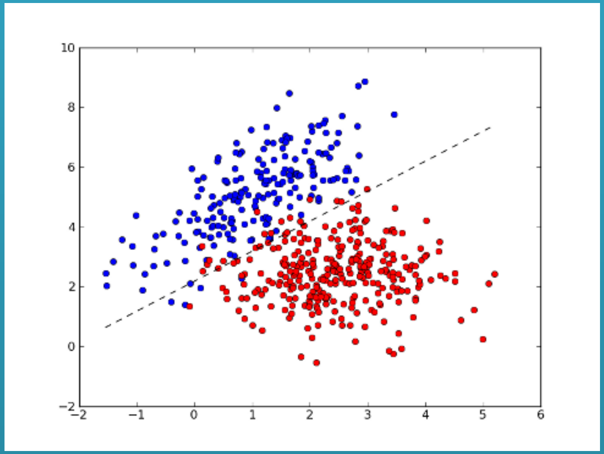


Fit a function to data
(regression).

General Approaches (cont'd)

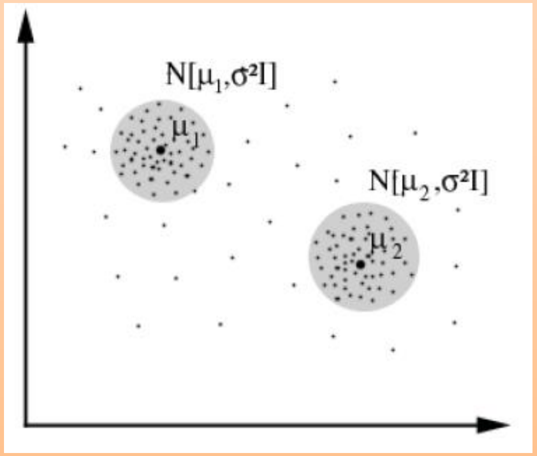
Discriminative

- Support Vector Machines
- Artificial Neural Networks
- Conditional Random Fields
- K-NN



Generative

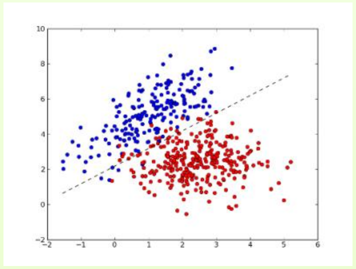
- Regression
- Markov Random Fields
- Bayesian Networks/Learning
- Clustering via Gaussian Mixture Models, Parzen Windows etc.
- …



General Approaches (cont'd)

Supervised

Instance	Label
	elma
	elma
	armut
	elma
	armut

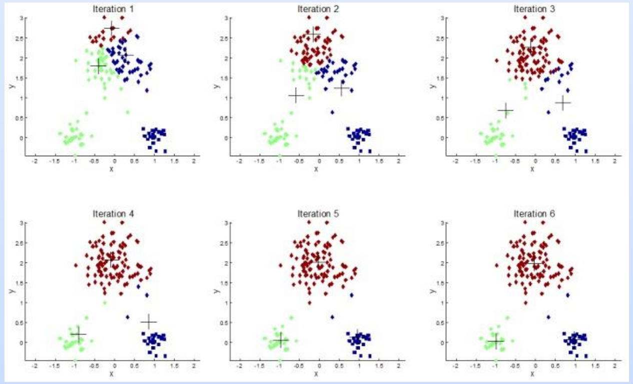


e.g. SVM

Extract Features

Learn a model

Unsupervised



e.g. k-means clustering

Extract Features

Learn a model

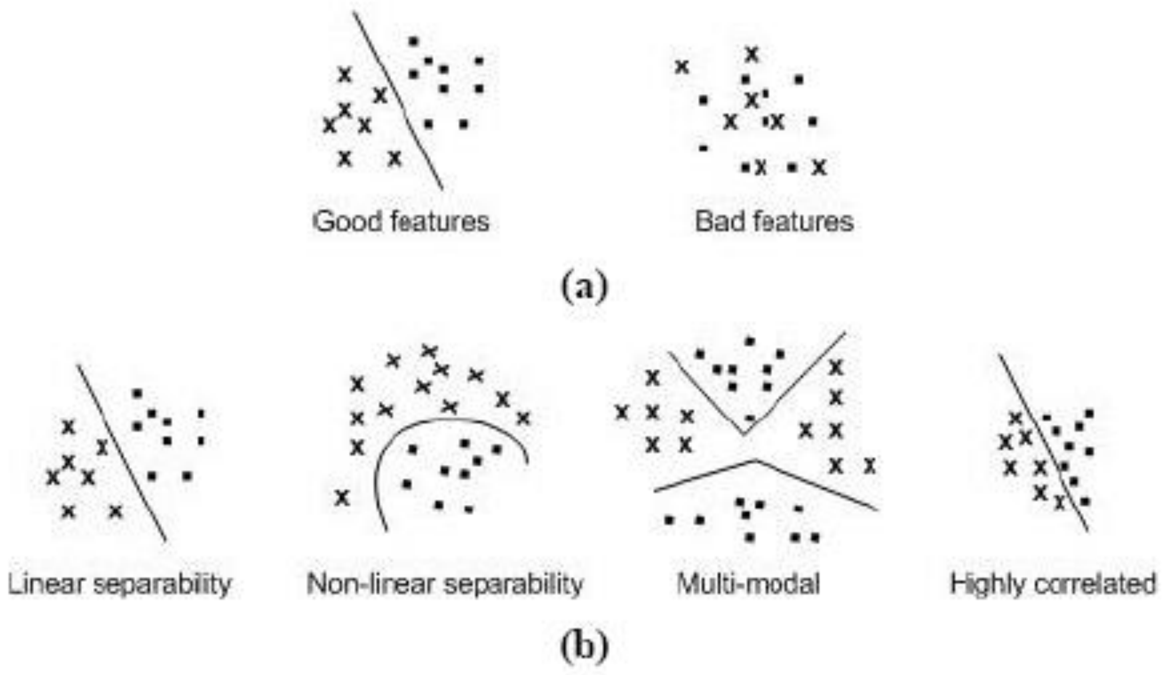
General Approaches (cont'd)

	Generative	**Discriminative**
Supervised	Regression Markov Random Fields Bayesian Networks	Support Vector Machines Neural Networks Conditional Random Fields Decision Tree Learning
Unsupervised	Gaussian Mixture Models Parzen Windows	K-means Self-Organizing Maps

Now

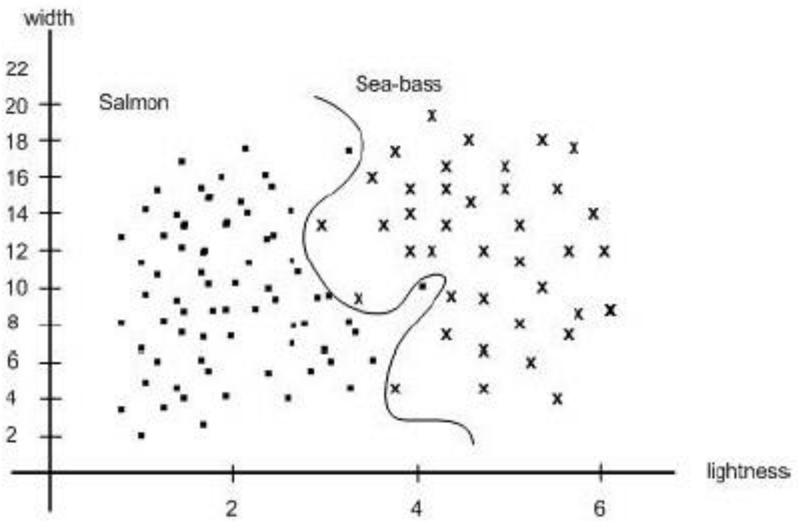
	Generative	Discriminative
Supervised	Softmax/Logistic Regression	Support Vector Machines
Unsupervised	Parzen Windows	

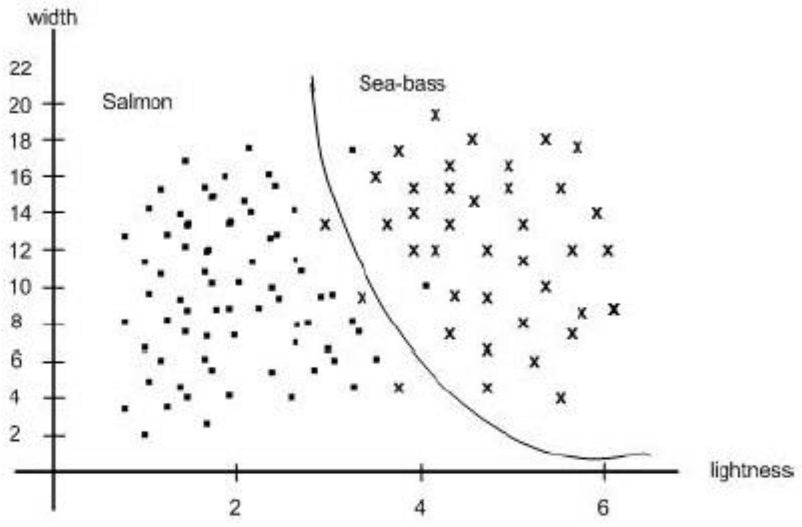
Feature Space

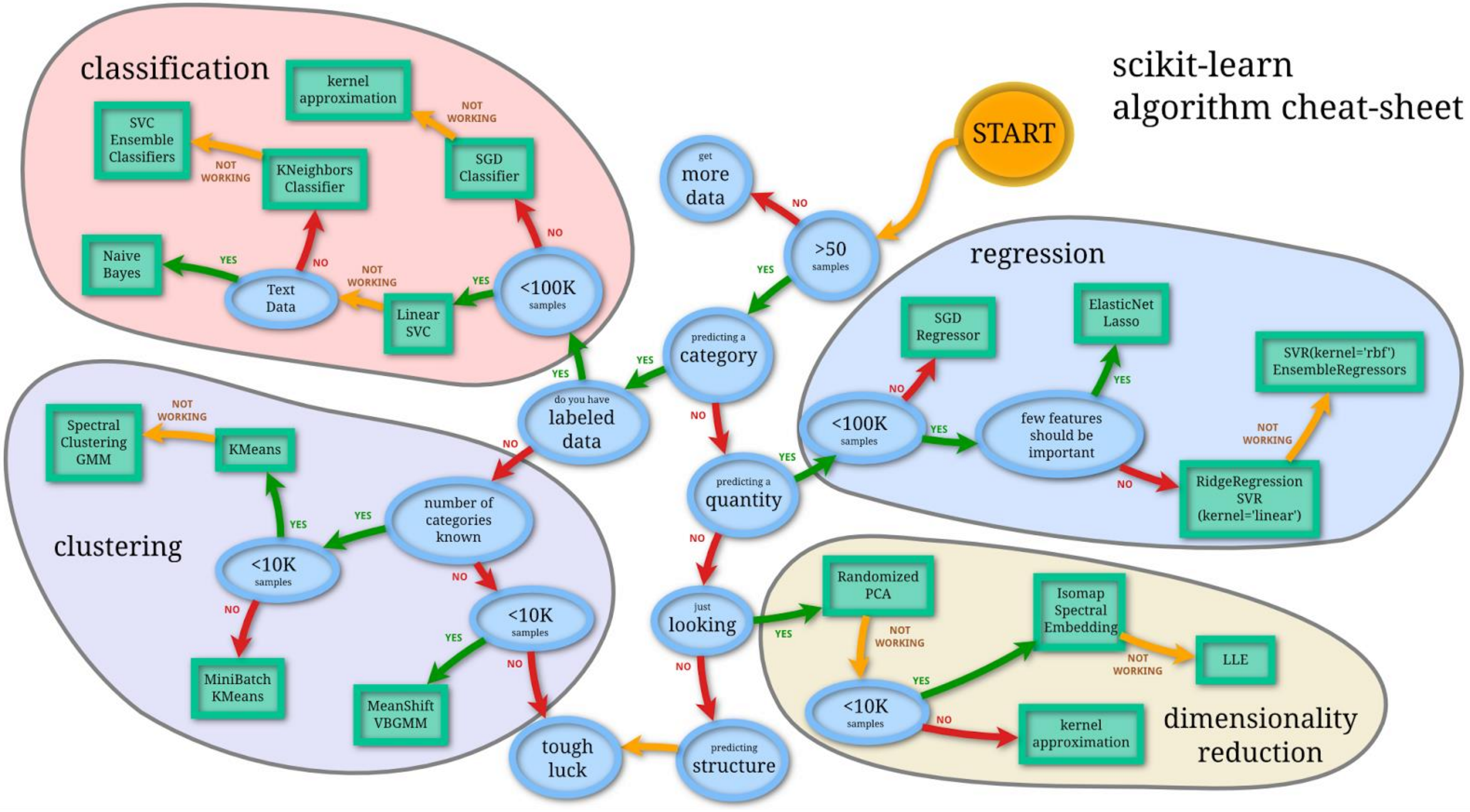


General pitfalls/problems

- Overfitting
- Underfitting
- Occams' razor



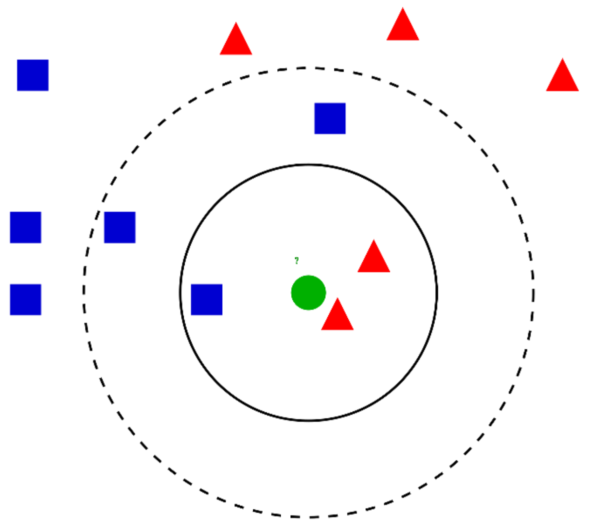



scikit-learn
algorithm cheat-sheet
START
>50 samples
get more data
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
classification
<100K samples
SGD Classifier
kernel approximation
Linear SVC
Text Data
Naive Bayes
KNeighbors Classifier
SVC Ensemble Classifiers
clustering
number of categories known
<10K samples
KMeans
Spectral Clustering GMM
MiniBatch KMeans
MeanShift VBGMM
regression
SGD Regressor
few features should be important
ElasticNet Lasso
RidgeRegression SVR (kernel='linear')
SVR(kernel='rbf') EnsembleRegressors
dimensionality reduction
Randomized PCA
Isomap Spectral Embedding
LLE
YES
NO
NOT WORKING

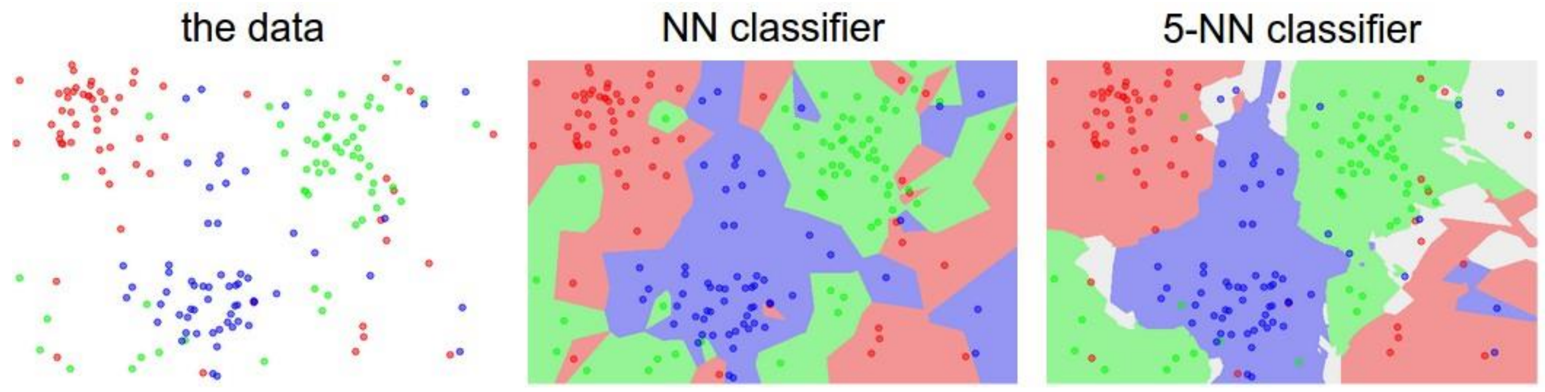
K-NEAREST NEIGHBOR

A very simple algorithm

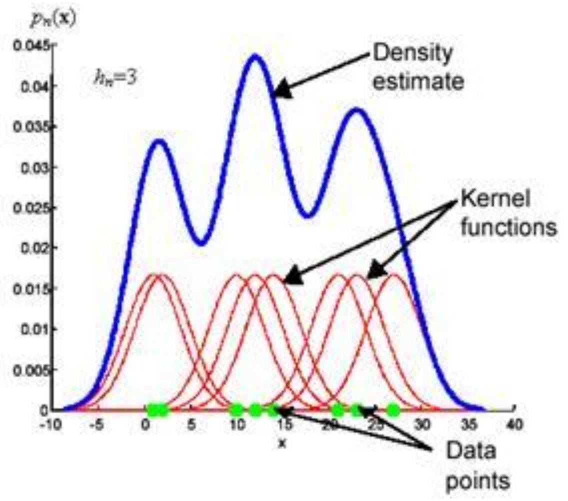
- Advantages:
  - No training
  - Simple
- Disadvantages:
  - Slow testing time (more efficient versions exist)
  - Needs a lot of memory



Wikipedia



http://cs231n.github.io/classification/



A non-parametric method

PARZEN WINDOWS

Probability and density

- Probability of a continuous probability function, $p(x)$, satisfies the following:
  - Probability to be between two values:
$$P(a < x < b) = \int_a^b p(x)dx$$
  - $p(x) > 0$ for all real x.
  - And
$$\int_{-\infty}^{\infty} p(x)dx = 1$$
- In 2-D:
  - Probability for $\mathbf{x}$ to be inside region R:
$$P = \int_R p(\mathbf{x})d\mathbf{x}$$
  - $p(\mathbf{x}) > 0$ for all real $\mathbf{x}$.
  - And
$$\int_{-\infty}^{\infty} p(\mathbf{x})d\mathbf{x} = 1$$

Density Estimation

- Basic idea:

$$P = \int_R p(\mathbf{x}) d\mathbf{x}$$

- If R is small enough, so that $p(\mathbf{x})$ is almost constant in R:
  - $P = \int_R p(\mathbf{x}) d\mathbf{x} \approx \mathrm{p}(\mathbf{x}) \int_R d\mathbf{x} = p(\mathbf{x})V$
  - V: volume of region R.

- If k out of n samples fall into R, then

$$P = k/n$$

- From which we can write:

$$\frac{k}{n} = p(\mathbf{x})V \Rightarrow p(\mathbf{x}) = \frac{k/n}{V}$$

Parzen Windows

- Assume that:
  - R is a hypercube centered at $\mathbf{x}$.
  - h is the length of an edge of the hypercube.
  - Then, $V = h^2$ in 2D and $V = h^3$ in 3D etc.
- Let us define the following window function:

$$w\left(\frac{\mathbf{x}_i - \mathbf{x}_k}{h}\right) = \begin{cases} 1, & |\mathbf{x}_i - \mathbf{x}_k|/h < 1/2 \\ 0, & \text{otherwise} \end{cases}$$

- Then, we can write the number of samples falling into R as follows:

$$k = \sum_{i=1}^{n} w\left(\frac{\mathbf{x}_i - \mathbf{x}_k}{h}\right)$$

Parzen Windows (cont'd)

- Remember: $p(\mathbf{x}) = \frac{k/n}{V}$.
- Using this definition of k, we can rewrite $p(\mathbf{x})$ (in 2D):

$$p(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h^2}w\left(\frac{\mathbf{x}_i - \mathbf{x}}{h}\right)$$

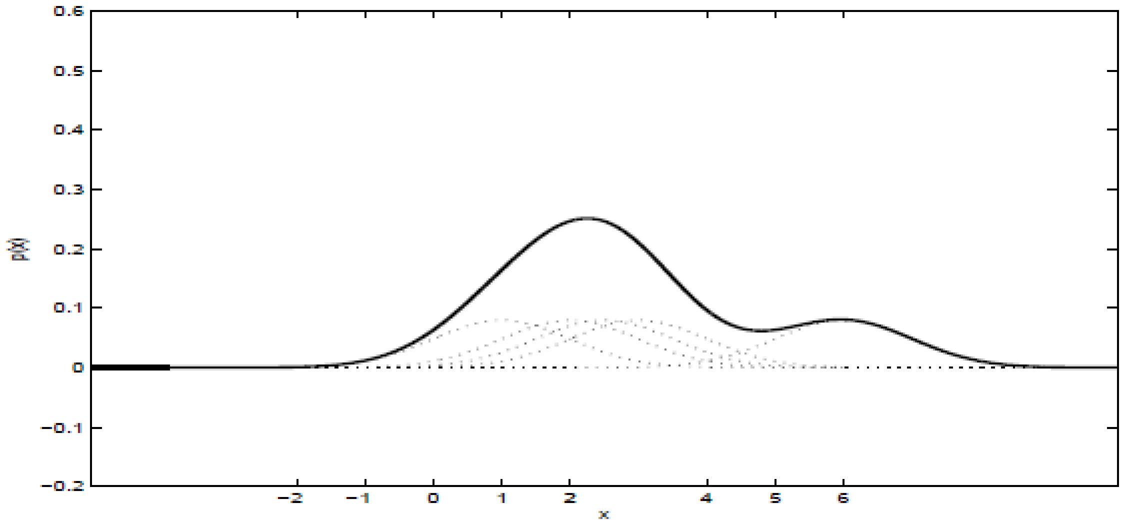
- Interpretation: Probability is the contribution of window functions fitted at each sample!

Parzen Windows (cont'd)

- The type of the window function can add different flavors.
- If we use Gaussian for the window function:

$$p(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(\mathbf{x}_i - \mathbf{x})^{\mathbf{2}}}{2\sigma^2}\right)$$

- Interpretation: Fit a Gaussian to each sample.



0.6
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
p(x)
−2
−1
0
1
2
4
5
6
x

LINEAR CLASSIFICATION

Linear classification

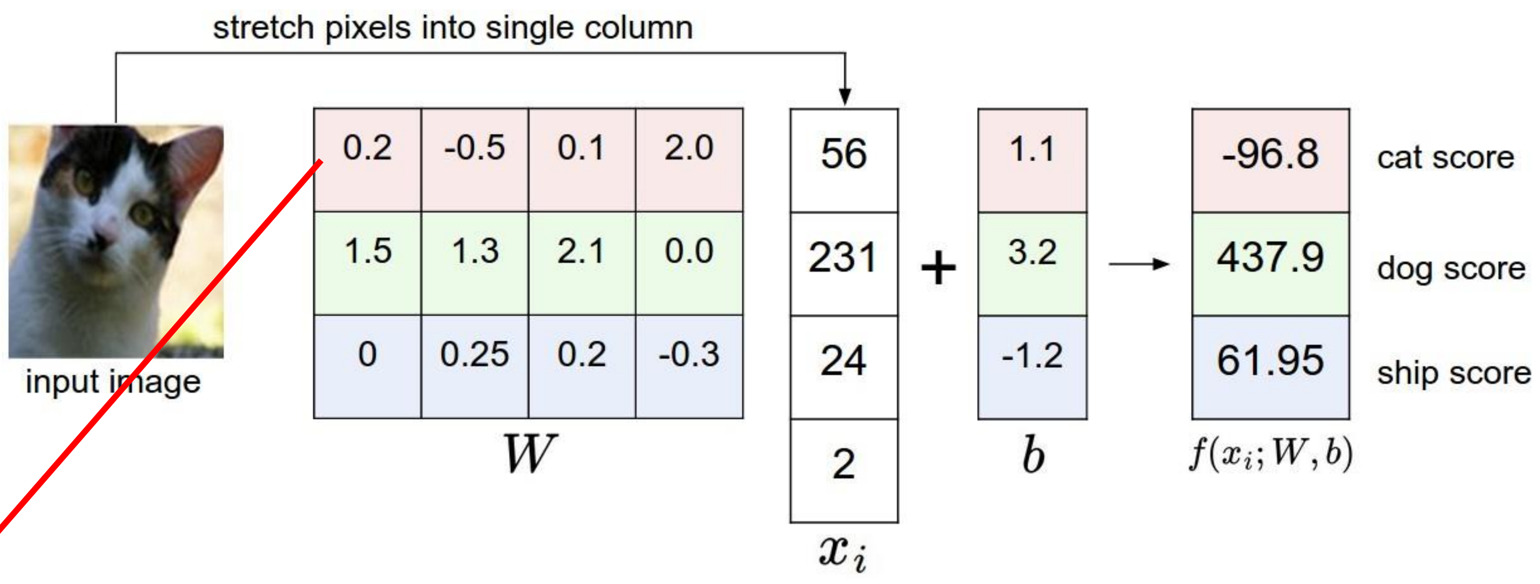
- Linear classification relies on the following score function:

$$f(x_i; W, b) = Wx_i + b$$

Or

$$f(x_i; W, b) = Wx_i$$

- where the bias is implicitly represented in W and x_i



One row per class

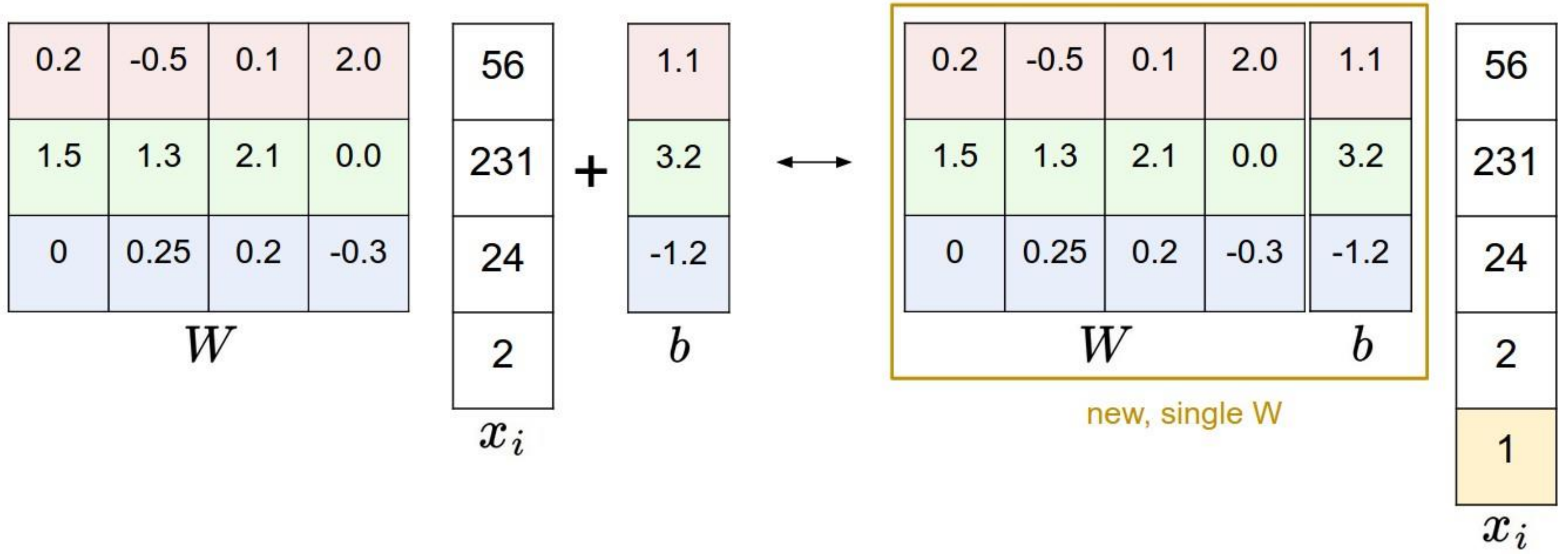
http://cs231n.github.io/linear-classify/

Linear classification

- We can rewrite the score function as:

$$f(x_i; W, b) = W x_i$$

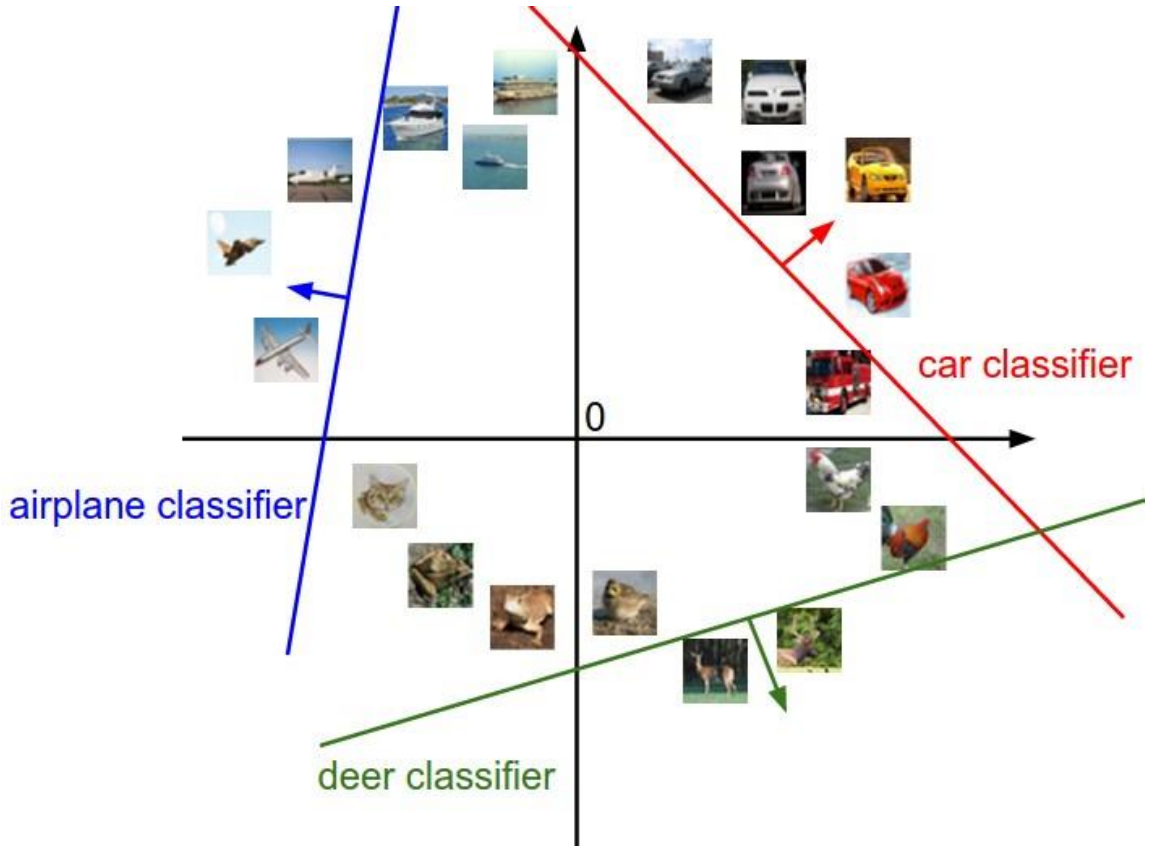
- where the bias is implicitly represented in W and x_i



http://cs231n.github.io/linear-classify/

Linear classification: One interpretation

- Since an image can be thought as a vector, we can consider them as points in high-dimensional space.



One interpretation of:

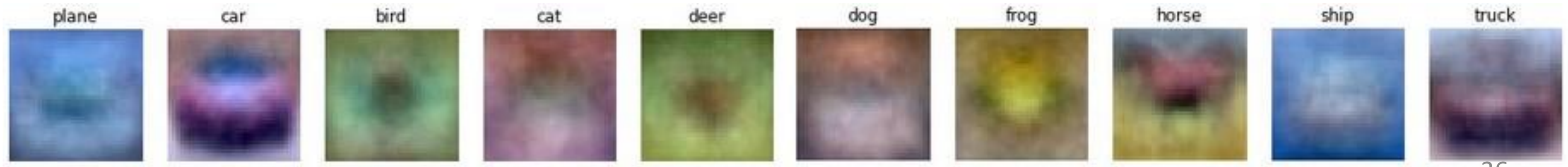
$f(x_i; W, b) = Wx_i + b$

- Each row describes a line for a class, and “b”

http://cs231n.github.io/linear-classify/

Linear classification: Another interpretation

- Each row in W can be interpreted as a template of that class.
  - $f(x_i; W, b) = Wx_i + b$ calculates the inner product to find which template best fits x_i.
  - Effectively, we are doing Nearest Neighbor with the "prototype" images of each class.



http://cs231n.github.io/linear-classify/

Loss function

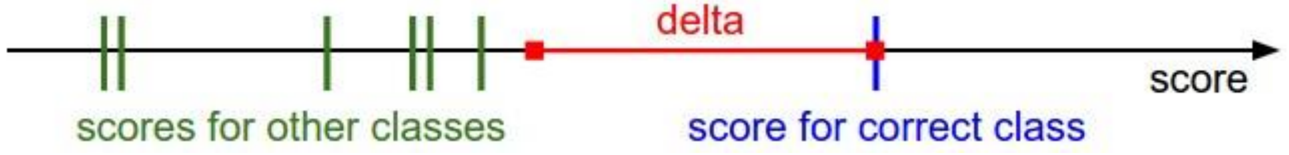
- A function which measures how good our weights are.
  - Other names: cost function, objective function
- Let $s_j = f(x_i; W)_j$
- An example loss function:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Or equivalently:

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

- This directs the distances to other classes to be more than Δ (the margin)



http://cs231n.github.io/linear-classify/

Example

- Consider our scores for x_i to be $s = [13, -7, 11]$ and assume Δ as 10.
- Then,

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

http://cs231n.github.io/linear-classify/

Regularization

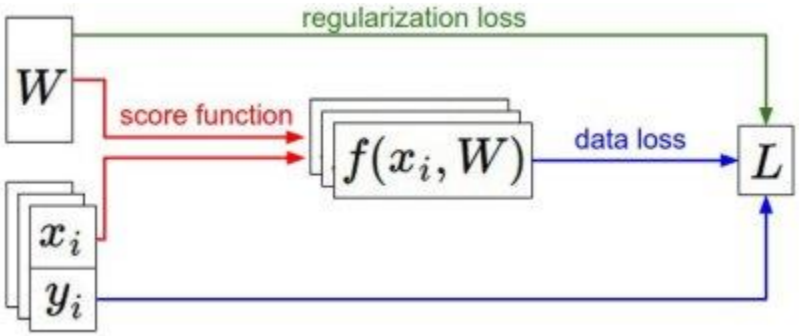
- In practice, there are many possible solutions leading to the same loss value.
  - Based on the requirements of the problem, we might want to penalize certain solutions.
- E.g.,

$$R(W) = \sum_i \sum_j W_{i,j}^2$$

  - which penalizes large weights.
    - Why do we want to do that?

http://cs231n.github.io/linear-classify/

Combined Loss Function



- The loss function becomes:

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

- If you expand it:

$$L = \frac{1}{N}\sum_i \sum_{j \neq y_i} \left[\max\left(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta\right)\right] + \lambda \sum_i \sum_j W_{i,j}^2$$

Hyper parameters
(estimated using validation set)

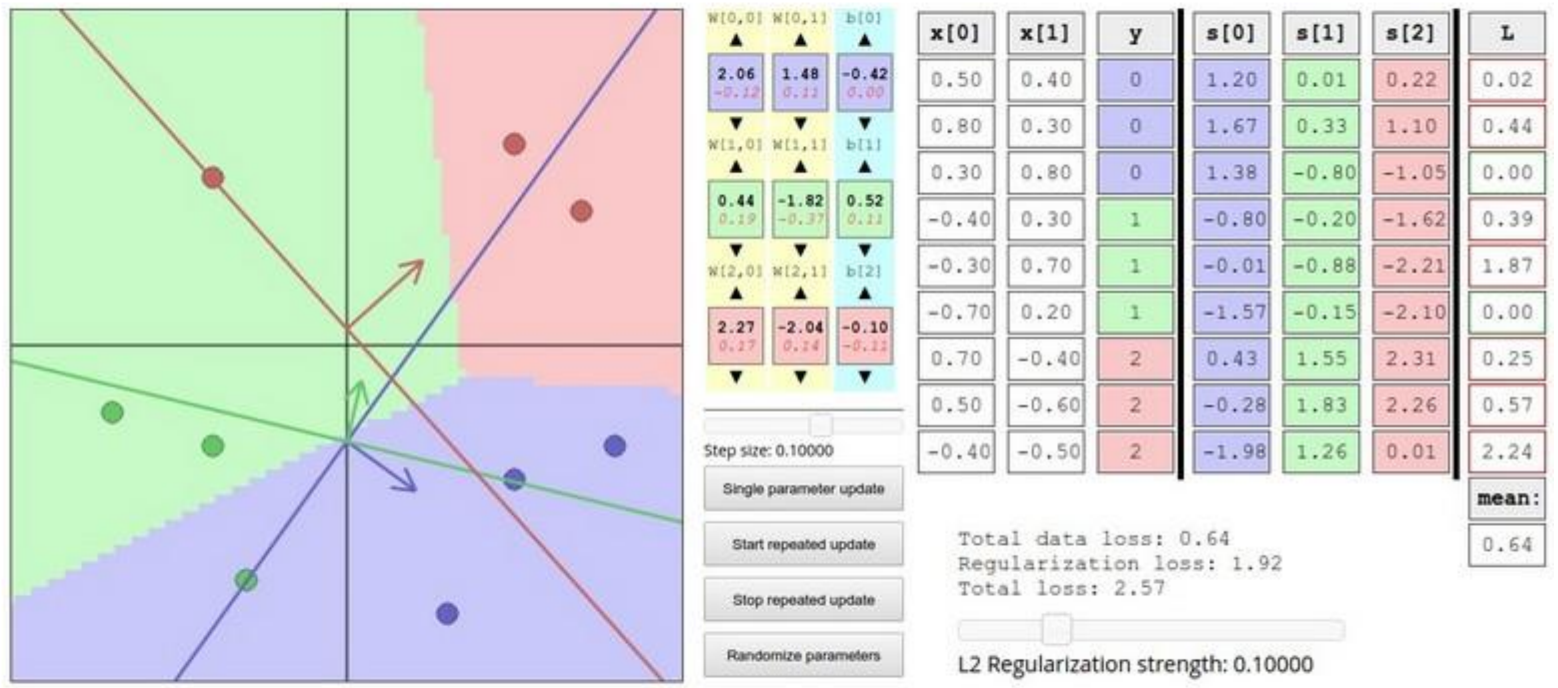
http://cs231n.github.io/linear-classify/

Hinge Loss, or Max-Margin Loss

$$L = \frac{1}{N}\sum_{i}\sum_{j\neq y_i}\left[\max\left(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta\right)\right] + \lambda\sum_{i}\sum_{j} W_{i,j}^2$$

http://cs231n.github.io/linear-classify/

Interactive Demo



W[0,0]	W[0,1]	b[0]
2.06 -0.12	1.48 0.11	-0.42 0.00

W[1,0]	W[1,1]	b[1]
0.44 0.19	-1.82 -0.37	0.52 0.11

W[2,0]	W[2,1]	b[2]
2.27 0.17	-2.04 0.14	-0.10 -0.11

Step size: 0.10000

x[0]	x[1]	y	s[0]	s[1]	s[2]	L
0.50	0.40	0	1.20	0.01	0.22	0.02
0.80	0.30	0	1.67	0.33	1.10	0.44
0.30	0.80	0	1.38	-0.80	-1.05	0.00
-0.40	0.30	1	-0.80	-0.20	-1.62	0.39
-0.30	0.70	1	-0.01	-0.88	-2.21	1.87
-0.70	0.20	1	-1.57	-0.15	-2.10	0.00
0.70	-0.40	2	0.43	1.55	2.31	0.25
0.50	-0.60	2	-0.28	1.83	2.26	0.57
-0.40	-0.50	2	-1.98	1.26	0.01	2.24
						mean:
						0.64

```
Total data loss: 0.64
Regularization loss: 1.92
Total loss: 2.57
```

L2 Regularization strength: 0.10000

http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/

An alternative formulation of

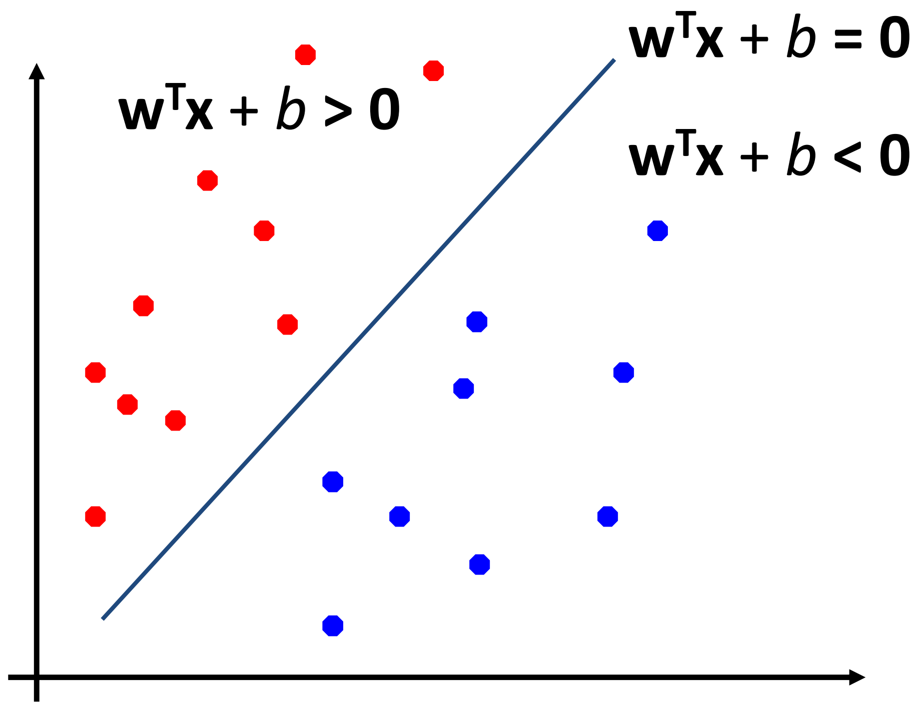
SUPPORT VECTOR MACHINES

Barrowed mostly from the slides of:

- Machine Learning Group, University of Texas at Austin.
- Mingyue Tan, The University of British Columbia

Linear Separators

- Binary classification can be viewed as the task of separating classes in feature space:

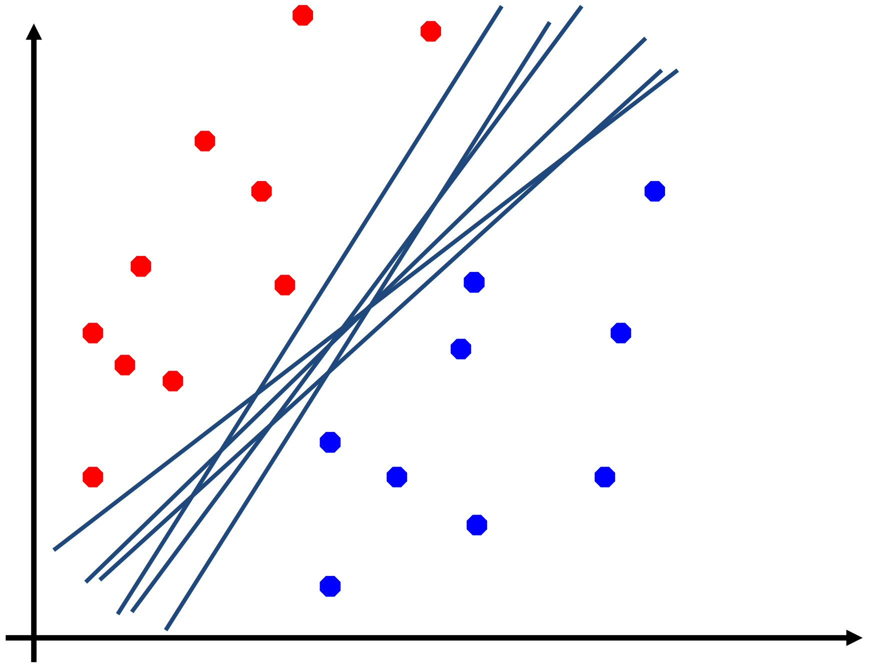


$\mathbf{w}^T\mathbf{x} + b = 0$
$\mathbf{w}^T\mathbf{x} + b > 0$
$\mathbf{w}^T\mathbf{x} + b < 0$

$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$

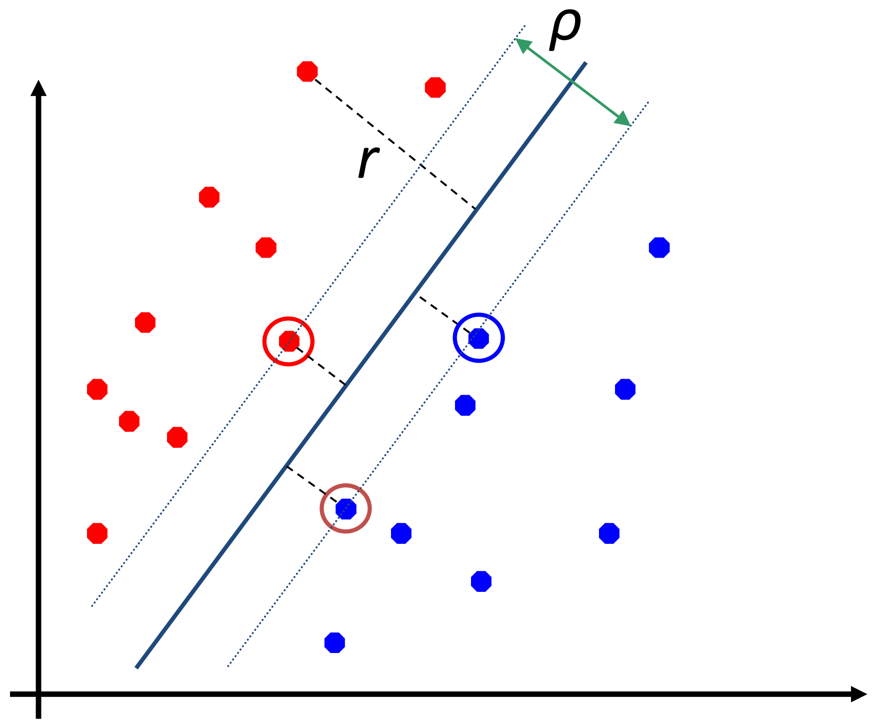
Linear Separators

- Which of the linear separators is optimal?



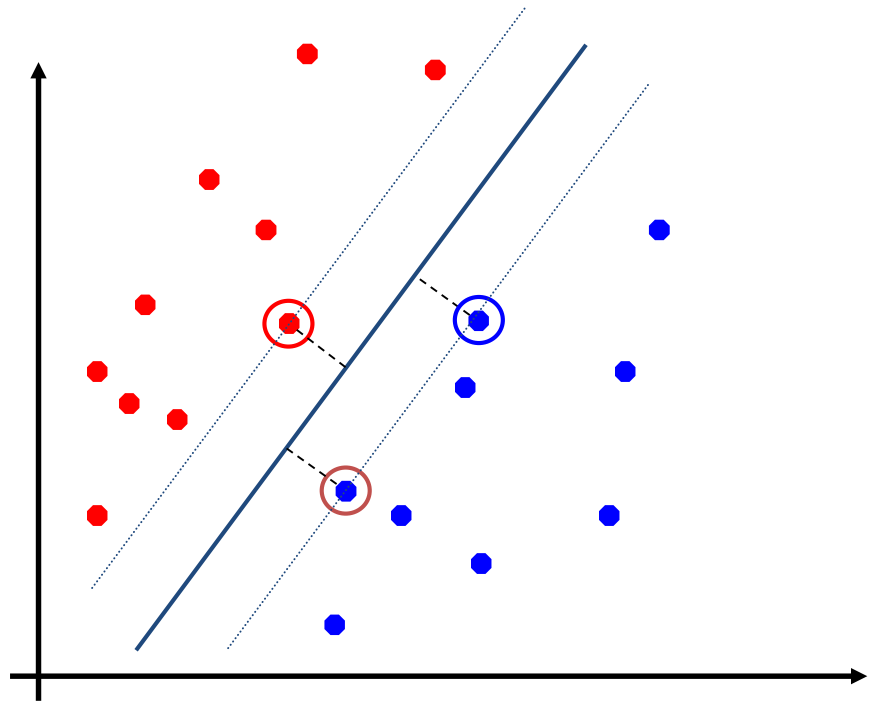
Classification Margin

- Distance from example $\mathbf{x}_i$ to the separator is $r = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are ***support vectors***.
- ***Margin*** ρ of the separator is the distance between support vectors.

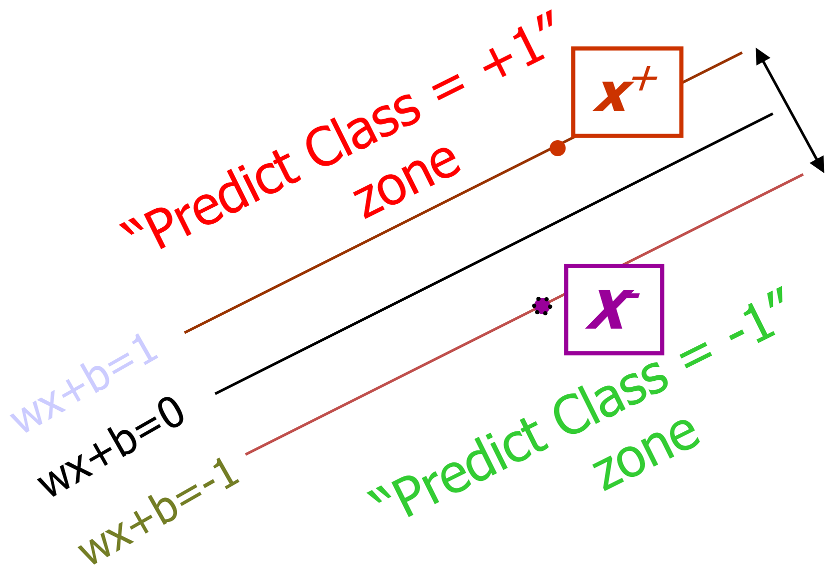


Maximum Margin Classification

- Maximizing the margin is good according to intuition.
- Implies that only support vectors matter; other training examples are ignorable.



Linear SVM Mathematically



ρ=Margin Width

What we know:

- $\boldsymbol{w} \,.\, \boldsymbol{x}^+ + b = +1$
- $\boldsymbol{w} \,.\, \boldsymbol{x}^- + b = -1$
- $\boldsymbol{w} \,.\, (\boldsymbol{x}^+ - \boldsymbol{x}^-) = 2$

$$\rho = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

Linear SVMs Mathematically (cont.)

- Then we can formulate the *optimization problem:*

Find **w** and *b* such that

$\rho = \frac{2}{\|\mathbf{w}\|}$ is maximized

and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Which can be reformulated as:

Find **w** and *b* such that

$\mathbf{\Phi}(\mathbf{w}) = ||\mathbf{w}||^2 = \mathbf{w}^T\mathbf{w}$ is minimized

and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Lagrange Multipliers

- Given the following optimization problem:

  minimize $f(x, y)$ subject to $g(x, y) = c$

- We can formulate it as:

$$L(x, y, \lambda) = f(x, y) + \lambda(g(x, y) - c)$$

- and set the derivative to zero:

$$\nabla_{x,y,\lambda} L(x, y, \lambda) = 0$$

Lagrange Multipliers

- Main intuition:
  - The gradients of f and g are parallel at the maximum

$$\nabla f = \lambda \nabla g$$

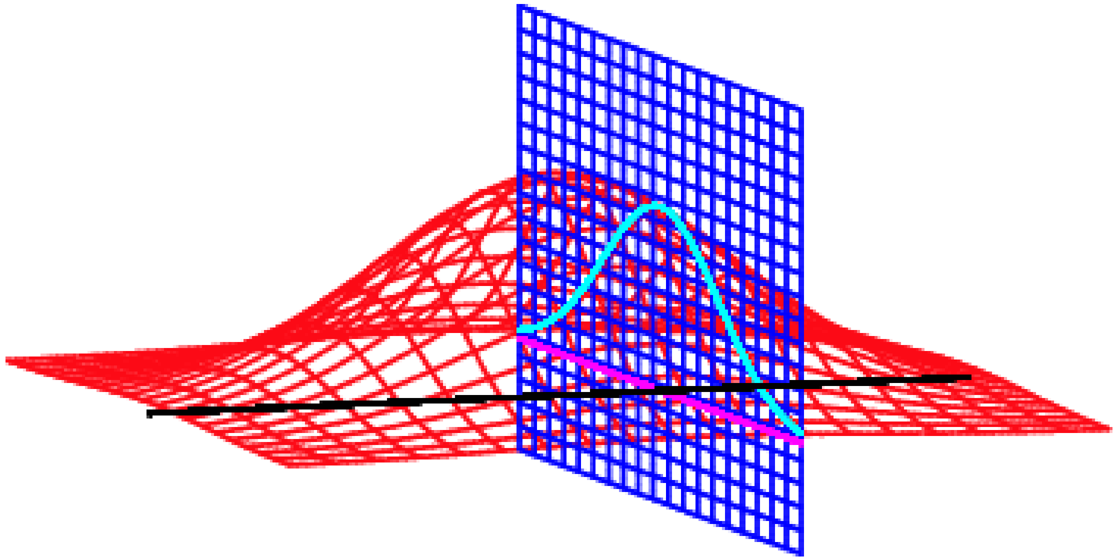


Fig: http://mathworld.wolfram.com/LagrangeMultiplier.html

Lagrange Multipliers

- See the following for proof
  - http://ocw.mit.edu/courses/mathematics/18-02sc-multivariable-calculus-fall-2010/2.-partial-derivatives/part-c-lagrange-multipliers-and-constrained-differentials/session-40-proof-of-lagrange-multipliers/MIT18_02SC_notes_22.pdf
- A clear example:
  - http://tutorial.math.lamar.edu/Classes/CalcIII/LagrangeMultipliers.aspx
- More intuitive explanation:
  - http://www.slimy.com/~steuard/teaching/tutorials/Lagrange.html

- In the SVM problem the Lagrangian is

$$L_P \equiv \tfrac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) + \sum_{i=1}^{l} \alpha_i$$

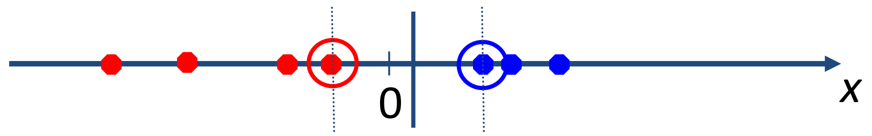
$$\alpha_i \geq 0, \forall i$$

- From the derivatives = 0 we get

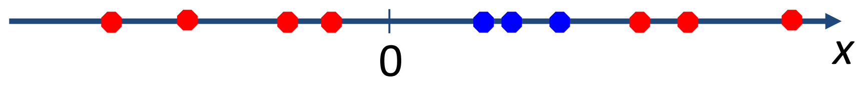
$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i, \sum_{i=1}^{l} \alpha_i y_i = 0$$

Non-linear SVMs

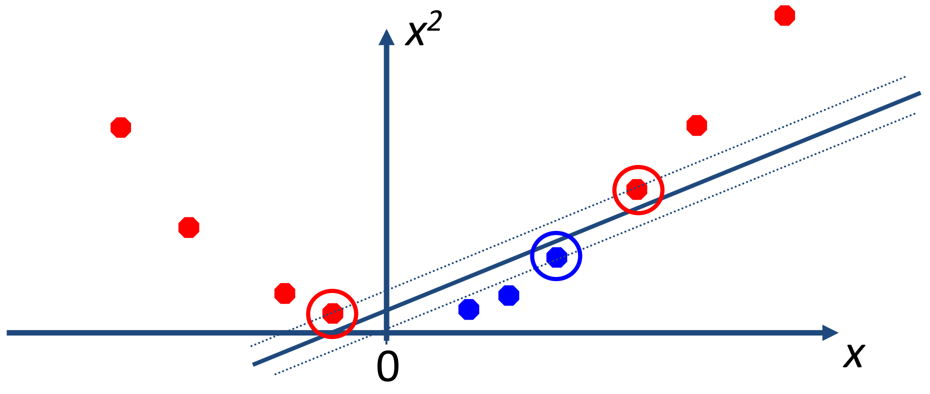
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?



- How about… mapping data to a higher-dimensional space:



SOFTMAX OR LOGISTIC CLASSIFIERS

Cross-entropy

- Uses cross-entropy (t: target probabilities, o: estimated probabilities):

$$H(p,q) = E_p[-\log q] = -\sum_j p_j \log q_j$$

- Wikipedia:
  - “the cross entropy between two probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set, if a coding scheme is used that is optimized for an "unnatural" probability distribution q, rather than the "true" distribution p”

Softmax classifier – cross-entropy **loss**

- Cross-entropy: $H(p,q) = E_p[-\log q] = -\sum_j p_j \log q_j$
- In our case,
  - p denotes the correct probabilities of the categories. In other words, $p_j = 1$ for the correct label and $p_j = 0$ for other categories.
  - q denotes the estimated probabilities of the categories
- But, our scores are not probabilities!
  - One solution: Softmax function: $f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$
  - It maps arbitrary ranges to probabilities
- Using the normalized values, we can define the cross-entropy loss for classification problem now:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) = -f_{y_i} + \log\sum_j e^{f_j}$$

http://cs231n.github.io/

logistic **loss**

- A special case of cross-entropy for binary classification:

$$H(p,q) = -\sum_j p_j \log q_j = -p_j \log q_j - (1-p_j)\log(1-q_j)$$

- Softmax function reduces to the logistic function:

$$\frac{1}{1+e^x}$$

- And the loss becomes:

$$L_i = -\log\left(\frac{1}{1+e^f}\right)$$

http://cs231n.github.io/

Why take logarithm of probabilities?

- Maps probability space to logarithmic space
- Multiplication becomes addition
  - Multiplication is a very frequent operation with probabilities
- Speed:
  - Addition is more efficient
- Accuracy:
  - Considering loss in representing real numbers, addition is friendlier
- Since log-probability is negative, to work with positive numbers, we usually negate the log-probability

Softmax classifier: One interpretation

- Information theory
  - Cross-entropy between a true distribution and an estimated one:

$$H(p,q) = -\sum_x p(x)\log q(x).$$

  - In our case, $p = [0, \ldots, 1, 0, .. 0]$, containing only one 1, at the correct label.
  - Since $H(p,q) = H(p) + D_{KL}(p||q)$, we are minimizing the Kullback-Leibler divergence.

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\log\frac{P(i)}{Q(i)}.$$

http://cs231n.github.io/

Softmax classifier: Another interpretation

- Probabilistic view

$$P(y_i \mid x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}.$$

- In our case, we are minimizing the negative log likelihood.
- Therefore, this corresponds to Maximum Likelihood Estimation (MLE).

http://cs231n.github.io/

Numerical Stability

- Exponentials may become very large. A trick:

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C\sum_j e^{f_j}} = \frac{e^{f_{y_i}+\log C}}{\sum_j e^{f_j+\log C}}$$

- Set $\log C = -\max_j f_j$.

SVM loss vs. cross-entropy loss

- SVM is happy when the classification satisfies the margin
  - Ex: if score values = [10, 9, 9] or [10, -10, -10]
    - SVM loss is happy if the margin is 1
  - SVM is local

- cross-entropy always wants better
  - cross-entropy is global

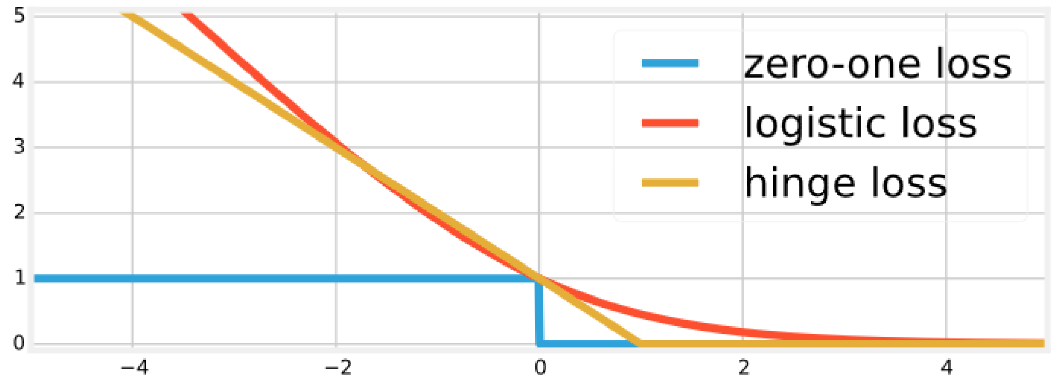
0-1 Loss

- Minimize the # of cases where the prediction is wrong:

$$L = \sum_{\boldsymbol{i}} \mathbf{1}\left(f(x_i; W, b)_{y_i} \neq \hat{y}_i\right)$$

Or equivalently,

$$L = \sum_{\boldsymbol{i}} \mathbf{1}\left(\hat{y}_i f(x_i; W, b)_{y_i} < 0\right)$$



Absolute Value Loss, Squared Error Loss

$$L_i = \sum_j |s_j - y_j|^q$$

- $q = 1$: absolute value loss
- $q = 2$: square error loss.

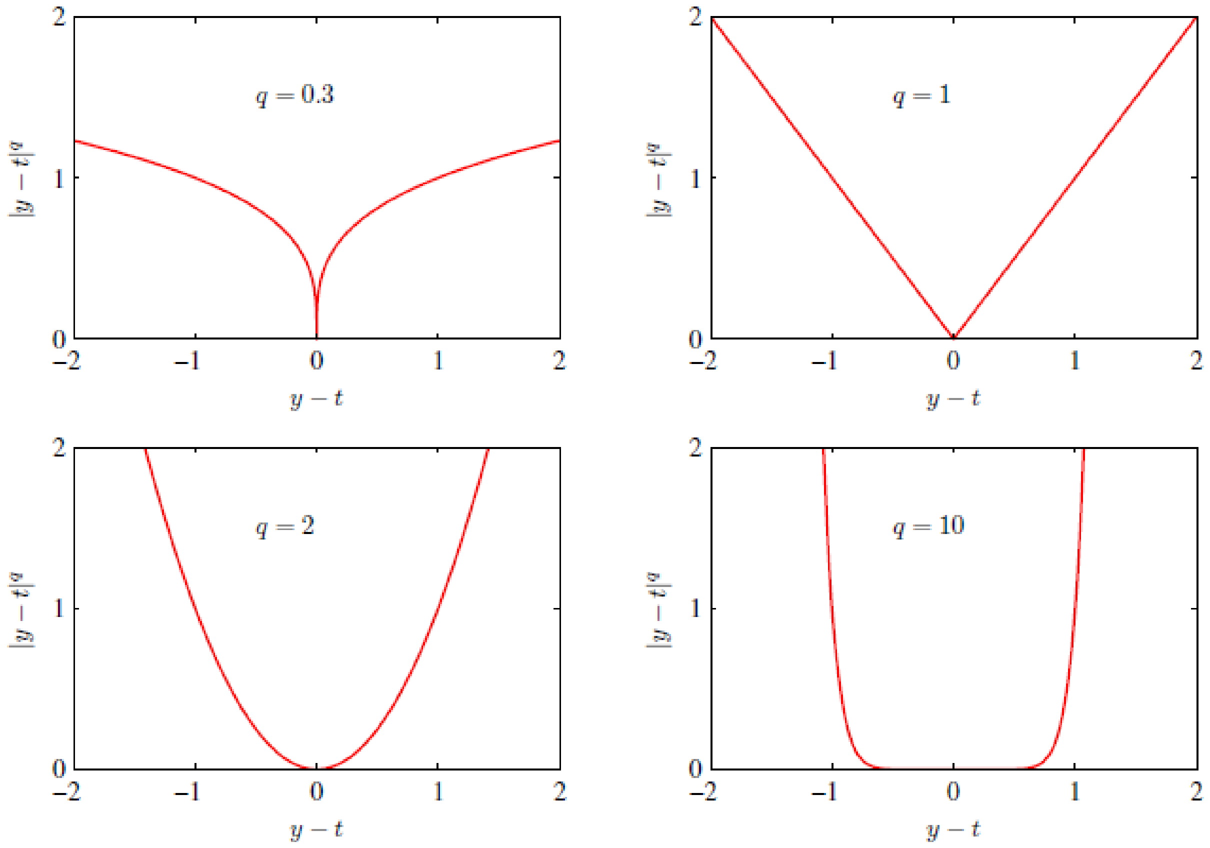


Figure 1.29 Plots of the quantity $L_q = |y - t|^q$ for various values of q.

Bishop

MORE ON LOSS FUNCTIONS

Visualizing Loss Functions

- If you look at one of the example loss functions:

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + 1)$$

- Since W has too many dimensions, this is difficult to plot.
- We can visualize this for one weight direction though, which can give us some intuition about the shape of the function.
  - E.g., start from an arbitrary W_0, choose a direction W_1 and plot $L(W_0 + \alpha W_1)$ for different values of α.

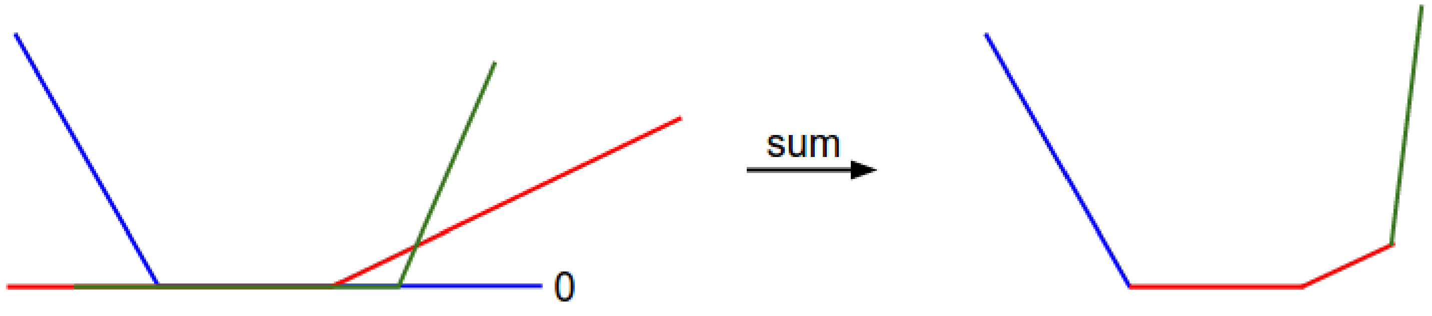
http://cs231n.github.io/

Visualizing Loss Functions

- Example:

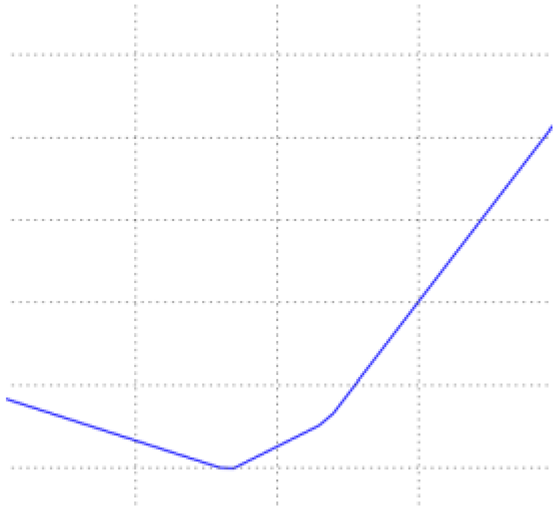
$$
\begin{aligned}
L_0 &= \max(0, w_1^T x_0 - w_0^T x_0 + 1) + \max(0, w_2^T x_0 - w_0^T x_0 + 1) \\
L_1 &= \max(0, w_0^T x_1 - w_1^T x_1 + 1) + \max(0, w_2^T x_1 - w_1^T x_1 + 1) \\
L_2 &= \max(0, w_0^T x_2 - w_2^T x_2 + 1) + \max(0, w_1^T x_2 - w_2^T x_2 + 1) \\
L &= (L_0 + L_1 + L_2)/3
\end{aligned}
$$

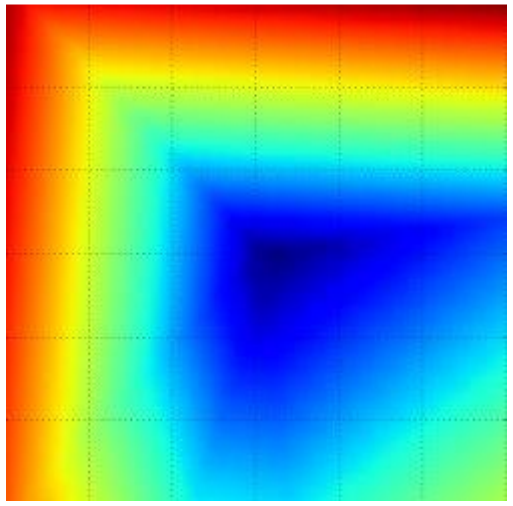
- If we consider just w_0, we have many linear functions in terms of w_0 and loss is a linear combination of them.

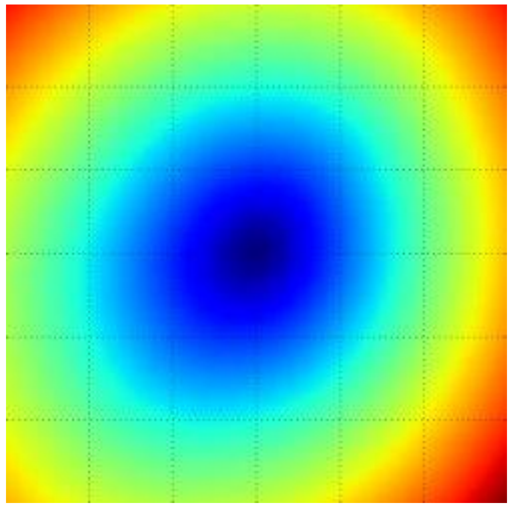


http://cs231n.github.io/

Visualizing Loss Functions


Loss along one direction


Loss along two directions


Loss along two directions
(averaged over many samples)

- You see that this is a convex function.
  - Nice and easy for optimization
- When you combine many of them in a neural network, it becomes non-convex.

Another approach for visualizing loss functions

- 0-1 loss:

$$L = 1(f(x) \neq y)$$

or equivalently as:

$$L = 1(yf(x) > 0)$$

- Square loss:

$$L = (f(x) - y)^2$$

in binary case:

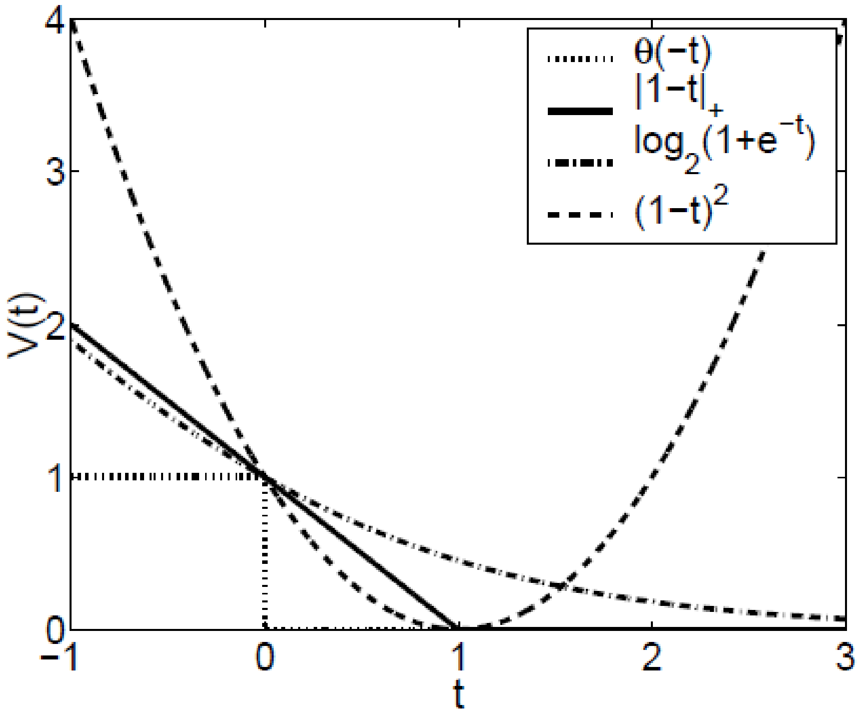
$$L = (1 - yf(x))^2$$

- Hinge-loss

$$L = \max(1 - yf(x), 0)$$

- Logistic loss:

$$L = (\ln 2)^{-1} \ln(1 + e^{-yf(x)})$$



Various loss functions used in classification. Here $t = yf(\mathbf{x})$.

Rosacco et al., 2003

SUMMARY OF LOSS FUNCTIONS

Linear Classifier: SVM

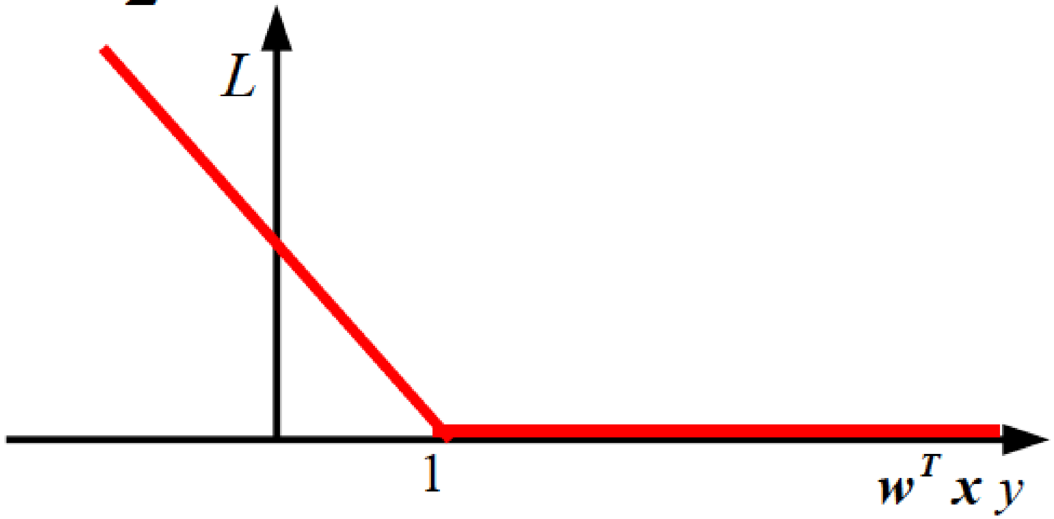
Input: $\boldsymbol{x} \in R^D$

Binary label: $y \in \{-1, +1\}$

Parameters: $\boldsymbol{w} \in R^D$

Output prediction: $\boldsymbol{w}^T \boldsymbol{x}$

Loss: $L = \frac{1}{2} ||\boldsymbol{w}||^2 + \lambda \, max[0, 1 - \boldsymbol{w}^T \boldsymbol{x} \, y]$



Hinge Loss

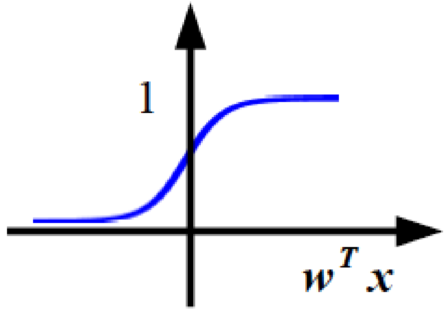
Linear Classifier: Logistic Regression

Input: $\boldsymbol{x} \in R^D$

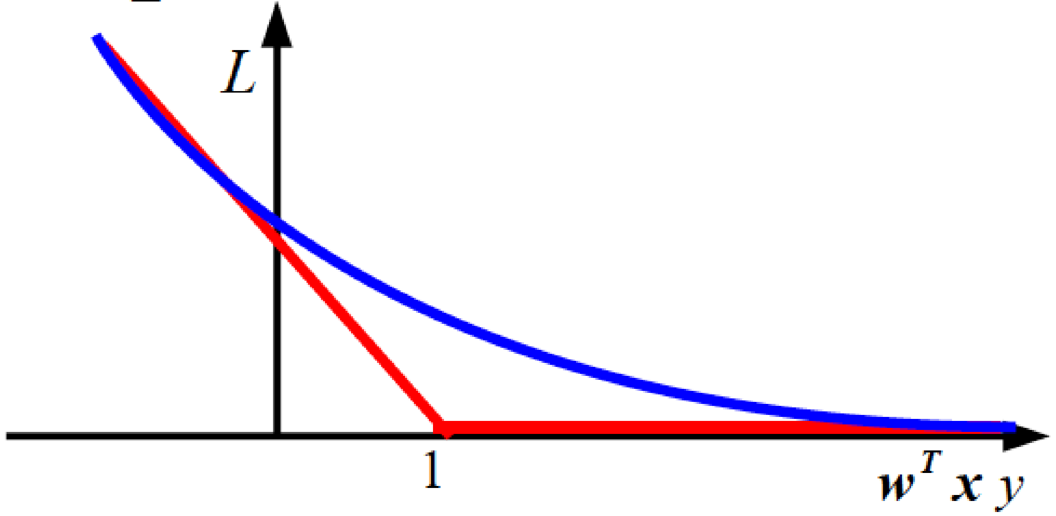
Binary label: $y \in \{-1, +1\}$

Parameters: $\boldsymbol{w} \in R^D$

Output prediction: $p(y=1|\boldsymbol{x}) = \dfrac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}$



Loss: $L = \dfrac{1}{2}\|\boldsymbol{w}\|^2 - \lambda \log(p(y|\boldsymbol{x}))$



Log Loss

Ranzato

Side Note: Different Losses

Logistic regression:

$$\boxed{\sum_{i=1}^{m} \ln(1+\exp(-y_i f(x_i)))}$$

Boosting :

$$\boxed{\frac{1}{m}\sum_{i} \exp(-y_i f(x_i)) = \prod_t Z_t}$$

SVM:

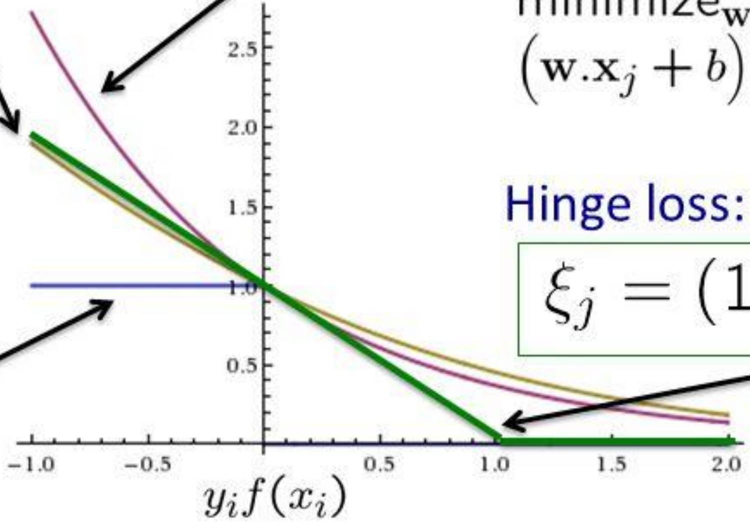
$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w} + C\sum_j \xi_j$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j,\ \forall j$$
$$\xi_j \geq 0,\ \forall j$$

Hinge loss:

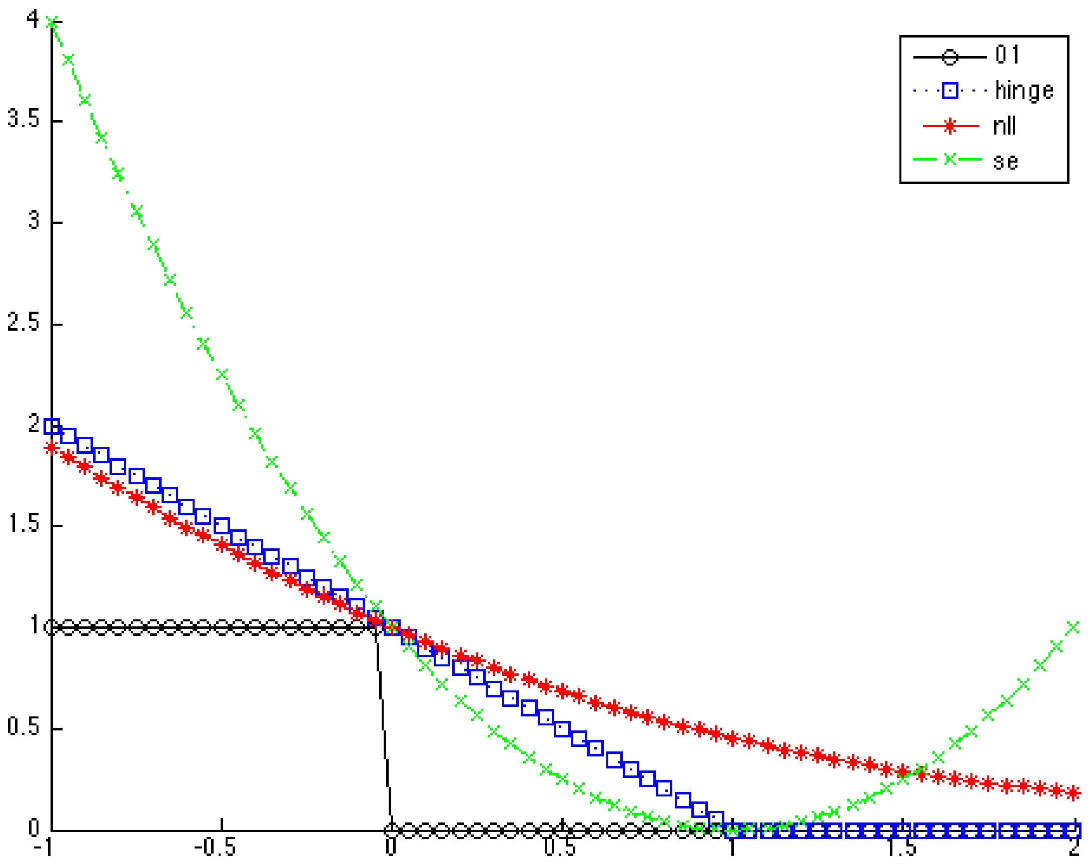
$$\boxed{\xi_j = (1 - f(x_i)y_i)_+}$$

0-1 Loss:

$$\boxed{\delta(H(x_i) \neq y_i)}$$

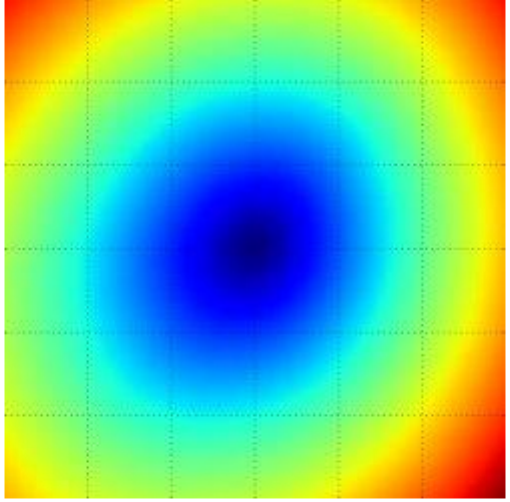


All our new losses approximate 0/1 loss!


01
hinge
nll
se
4
3.5
3
2.5
2
1.5
1
0.5
0
-1
-0.5
0
0.5
1
1.5
2

Sum up

- 0-1 loss is not differentiable/helpful at training
  - It is used in testing
- Other losses try to cover the “weakness” of 0-1 loss
- Hinge-loss imposes weaker constraint compared to cross-entropy
- For classification: use hinge-loss or cross-entropy loss
- For regression: use squared-error loss, or absolute difference loss



SO, WHAT DO WE DO WITH A LOSS FUNCTION?

Optimization strategies

- We want to find W that minimizes the loss function.
  - Remember that W has lots of dimensions.
- Naïve idea: random search
  - For a number of iterations:
    - Select a W randomly
    - If it leads to better loss than the previous ones, select it.
  - This yields 15.5% accuracy on CIFAR after 1000 iterations (chance: 10%)

http://cs231n.github.io/

Optimization strategies

- Second idea: random local search
  - Start at an arbitrary position (weight W)
  - Select an arbitrary direction $W + \delta W$ and see if it leads to a better loss
    - If yes, move along
  - This leads to 21.4% accuracy after 1000 iterations
  - This is actually a variation of simulated annealing
- A better idea: follow the gradient

A quick reminder on gradients / partial derivatives

- In one dimension: $\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$
- In practice:
  - $\frac{df[n]}{dn} = \frac{f[n+h]-f[n]}{h}$
  - $\frac{df[n]}{dn} = \frac{f[n+h]-f[n-h]}{2h}$ (centered difference – works better)
- In many dimensions:

1. Compute gradient numerically with finite differences
   - Slow
   - Easy
   - Approximate
2. Compute the gradient analytically
   - Fast
   - Exact
   - Error-prone to implement

- In practice: implement (2) and check against (1) before testing

http://cs231n.github.io/

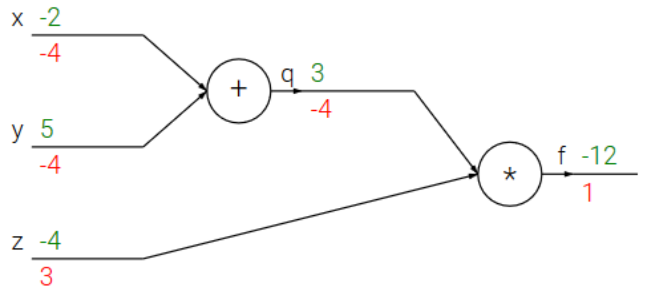
A quick reminder on gradients / partial derivatives

- If you have a many-variable function, e.g., $f(x, y) = x + y$, you can take its derivative wrt either x or y:
  - $\frac{df(x,y)}{dx} = \lim_{h \to 0} \frac{f(x+h,y) - f(x,y)}{h} = 1$
  - Similarly, $\frac{df(x,y)}{dy} = 1$
  - In fact, we should denote them as follows since they are "partial derivatives" or "gradients on x or y":
    - $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$
- Partial derivative tells you the rate of change along a single dimension at a point.
  - E.g., if $\partial f / \partial x$=1, it means that a change of x_0 in x leads to the same amount of change in the value of the function.
- Gradient is a vector of partial derivatives:
  - $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

http://cs231n.github.io/

A quick reminder on gradients / partial derivatives

- A simple example:
  - $f(x, y) = \max(x, y)$
  - $\nabla f = [\mathbf{1}(x \geq y),\ \mathbf{1}(y \geq x)]$
- Chaining:
  - What if we have a composition of functions?
  - E.g., $f(x, y, z) = q(x, y)z$ and $q(x, y) = x + y$
  - $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = z$
  - $\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = z$
  - $\frac{\partial f}{\partial z} = q(x, y) = x + y$



- Back propagation
  - Local processing to improve the system globally
  - Each gate locally determines to increase or decrease the inputs

http://cs231n.github.io/

Partial derivatives and backprop

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

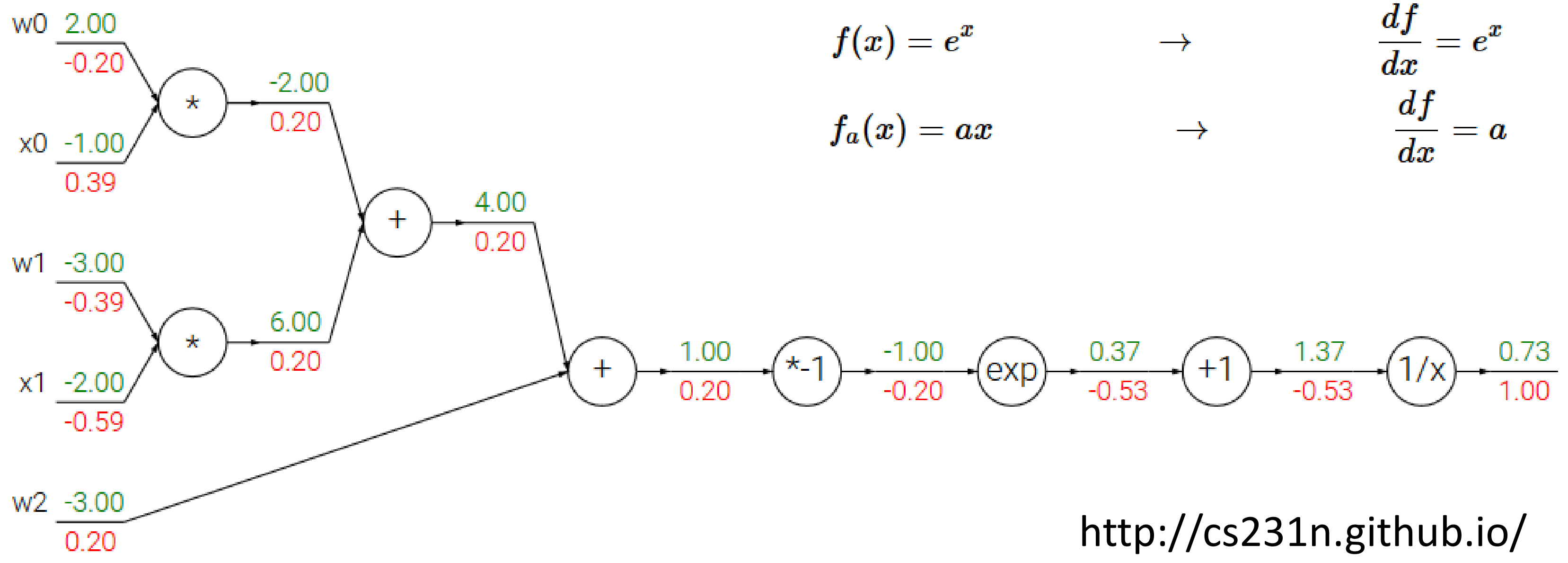
- Which has many gates:

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$



http://cs231n.github.io/

In fact, that was the sigmoid function

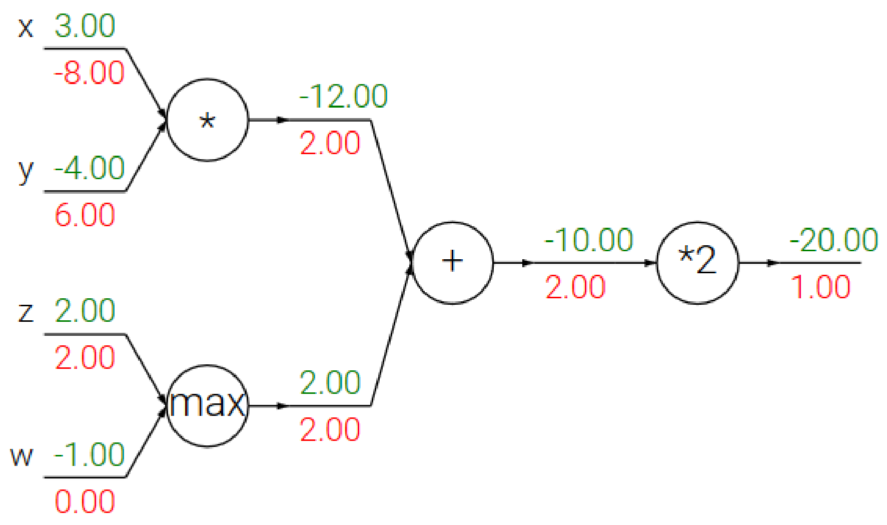
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\rightarrow \qquad \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

- We will combine all these gates into a single gate and call it a neuron

Intuitive effects

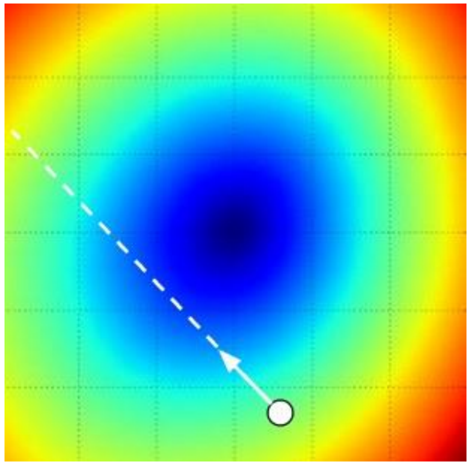
- Commonly used operations
  - Add gate
  - Max gate
  - Multiply gate
- Due to the effect of the multiply gate, when one of the inputs is large, the small input gets the large gradient
  - This has a negative effect in NNs
  - When one of the weights is unusually large, it effects the other weights.
  - Therefore, it is better to normalize the input / weights



http://cs231n.github.io/

Optimization strategies: gradient

- Move along the negative gradient (since we wish to go down)
  - $W - s\frac{\partial L(W)}{\partial W}$
  - s: step size



- Gradient tells us the direction
  - Choosing how much to move along that direction is difficult to determine
  - This is also called the learning rate
  - If it is small: too slow to converge
  - If it is big: you may overshoot and skip the minimum

http://cs231n.github.io/

Optimization strategies: gradient

- Take our loss function:

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + 1)$$

- Its gradient wrt w_j is:

$$\frac{\partial L_i}{\partial w_j} = \mathbf{1}\left(w_j^T x_i - w_{y_i}^T x_i + 1 > 0\right) x_i$$

- For the "winning" weight, this is:

$$\nabla_{w_{y_i}} L_i = -\left(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)\right) x_i$$

http://cs231n.github.io/

Gradient Descent

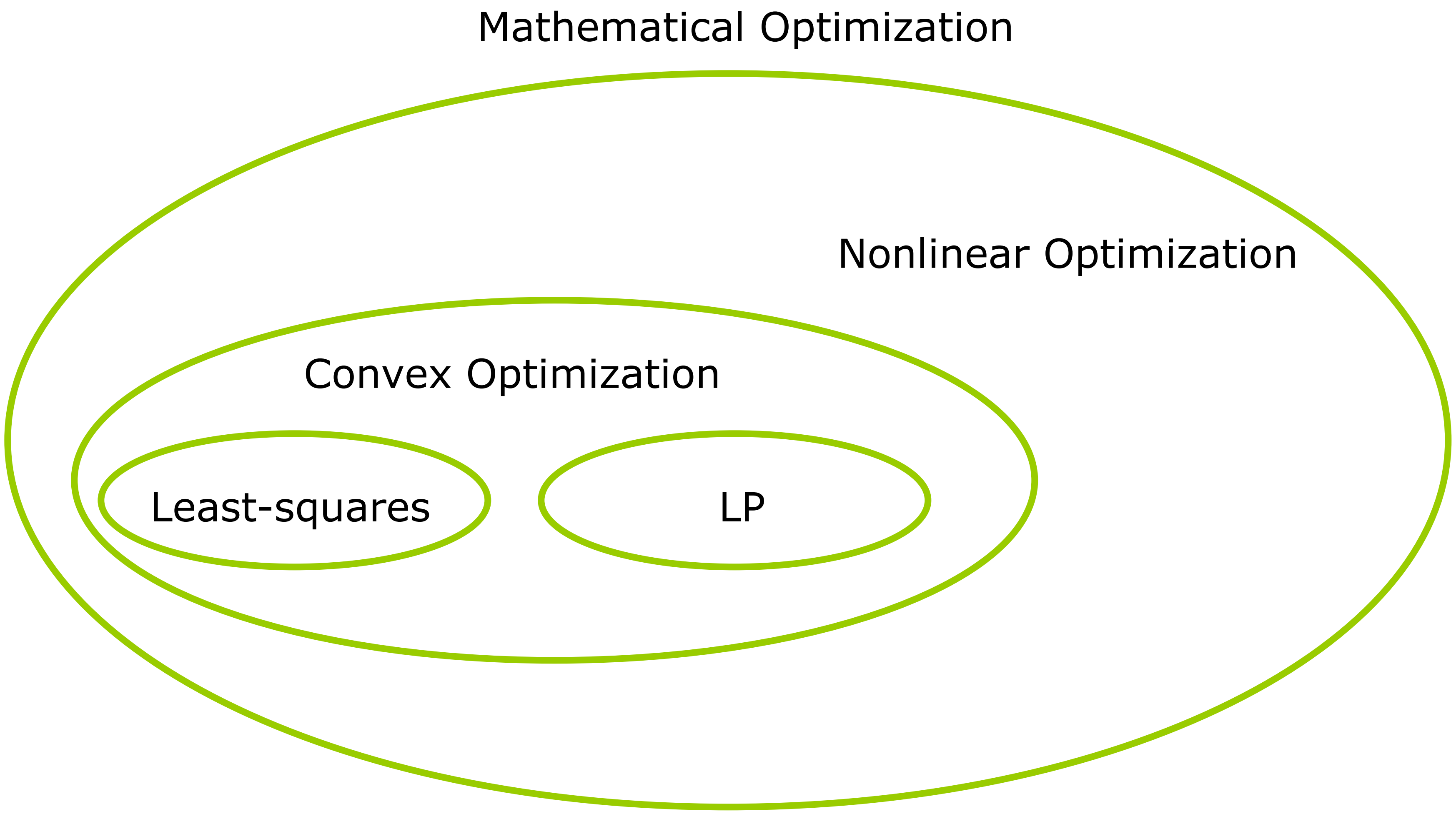
- Update the weight:

$$W_{new} \leftarrow W - s\frac{\partial L(W)}{\partial W}$$

- This computes the gradient after seeing all examples to update the weight.
  - Examples can be on the order of millions or billions
- Alternative:
  - Mini-batch gradient descent: Update the weights after, e.g., 256 examples
  - Stochastic (or online) gradient descent: Update the weights after each example
  - People usually use batches and call it stochastic.
  - Performing an update after one example for 100 examples is more expensive than performing an update at once for 100 examples due to matrix/vector operations

http://cs231n.github.io/

A GENERAL LOOK AT OPTIMIZATION


Mathematical Optimization
Nonlinear Optimization
Convex Optimization
Least-squares
LP

Mathematical Optimization

(mathematical) optimization problem

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq b_i, \quad i = 1, \ldots, m \end{array}$$

- $x = (x_1, \ldots, x_n)$: optimization variables
- $f_0 : \mathbf{R}^n \to \mathbf{R}$: objective function
- $f_i : \mathbf{R}^n \to \mathbf{R}$, $i = 1, \ldots, m$: constraint functions

optimal solution $x^\star$ has smallest value of f_0 among all vectors that satisfy the constraints

Convex Optimization

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq b_i, \quad i = 1, \ldots, m \end{array}$$

- objective and constraint functions are convex:

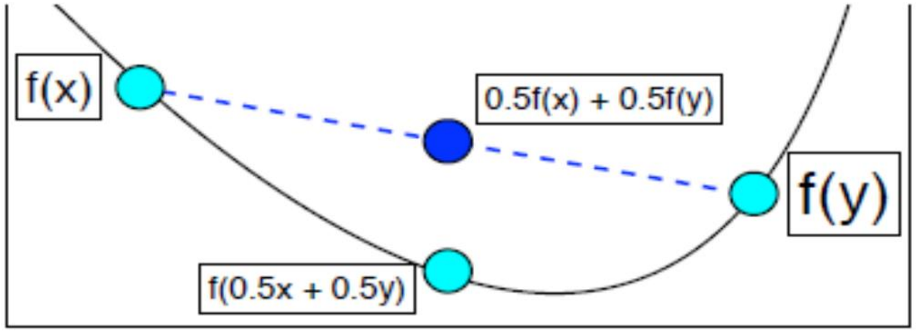
$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y)$$

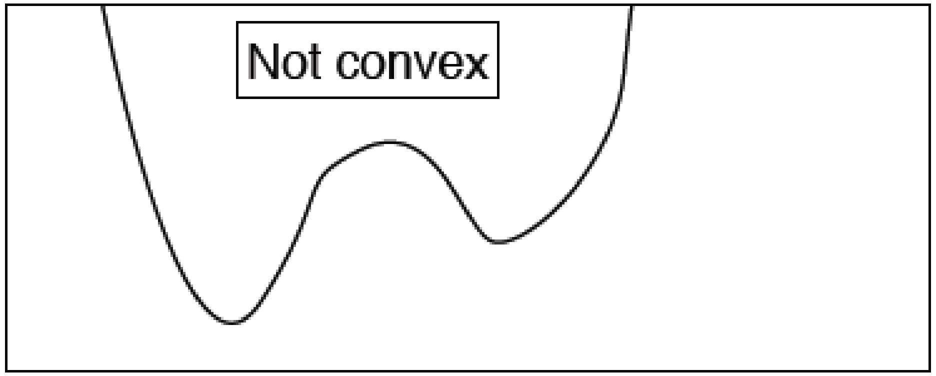
  if $\alpha + \beta = 1$, $\alpha \geq 0$, $\beta \geq 0$

- includes least-squares problems and linear programs as special cases

Interpretation

- Function's value is below the line connecting two points





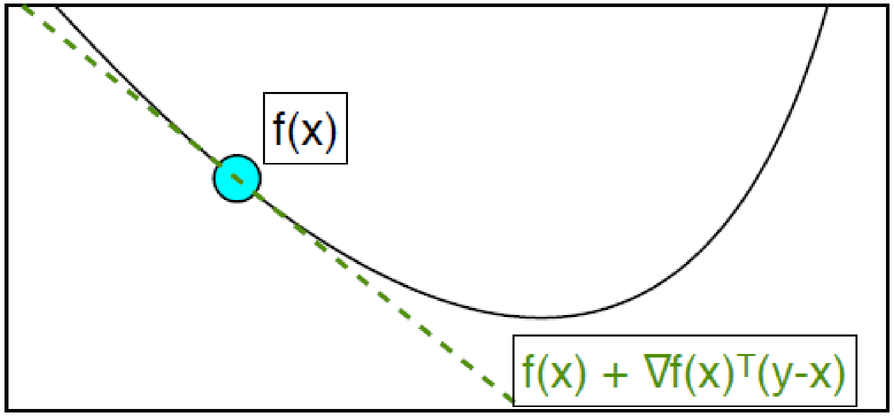
Mark Schmidt

Another interpretation

A differentiable function f is convex if for all x and y we have

$$f(y) \geq f(x) + \nabla f(x)^T (y - x),$$

- The function is globally above the tangent at x.



Example convex functions

Some simple convex functions:

- $f(x) = c$
- $f(x) = a^T x$
- $f(x) = x^T A x$ (for $A \succeq 0$)
- $f(x) = \exp(ax)$
- $f(x) = x \log x$ (for $x > 0$)
- $f(x) = \|x\|^2$
- $f(x) = \|x\|_p$
- $f(x) = \max_i \{x_i\}$

Some other notable examples:

- $f(x, y) = \log(e^x + e^y)$
- $f(X) = \log \det X$ (for X positive-definite).
- $f(x, Y) = x^T Y^{-1} x$ (for Y positive-definite)

Mark Schmidt

Operations that conserve convexity

1. Non-negative weighted sum:

$$f(x) = \theta_1 f_1(x) + \theta_2 f_2(x).$$

2. Composition with affine mapping:

$$g(x) = f(Ax + b).$$

3. Pointwise maximum:

$$f(x) = \max_i \{f_i(x)\}.$$

Show that least-residual problems are convex for any ℓ_p-norm:

$$f(x) = ||Ax - b||_p$$

We know that $||\cdot||_p$ is a norm, so it follows from (2).

$$||x + y||_p <= ||x||_p + ||y||_p$$

Show that SVMs are convex:

$$f(x) = \frac{1}{2}||x||^2 + C\sum_{i=1}^{n} \max\{0, 1 - b_i a_i^T x\}.$$

Know first term is convex, for the other terms use (3) on the two (convex) arguments, then use (1) to put it all together.

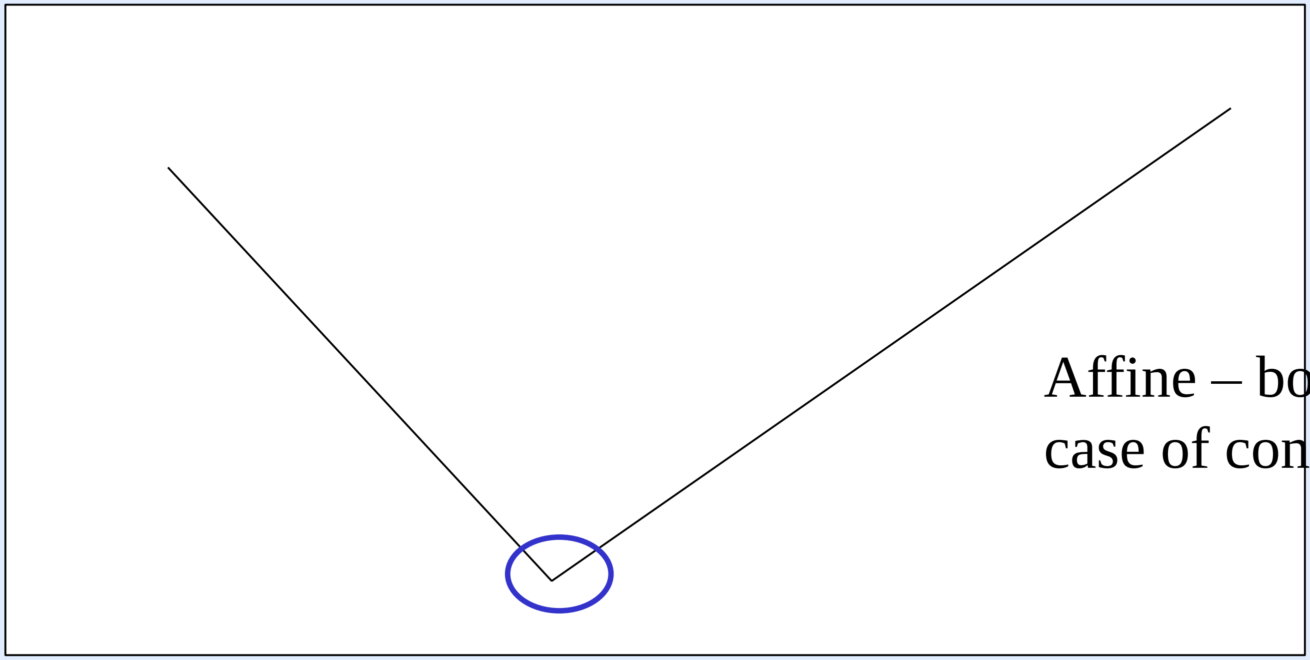
Why convex optimization?

- Can’t solve most OPs
  - E.g. NP Hard, even high polynomial time too slow
- Convex OPs
  - (Generally) No analytic solution
  - Efficient algorithms to find (global) solution
  - Interior point methods (basically Iterated Newton) can be used:
    - ~[10-100]*max{p^3 , p^2m, F} ; F cost eval. obj. and constr. f
  - At worst solve with general IP methods (CVX), faster specialized

Convex Function

- Easy to see why convexity allows for efficient solution
- Just “slide” down the objective function as far as possible and will reach a minimum

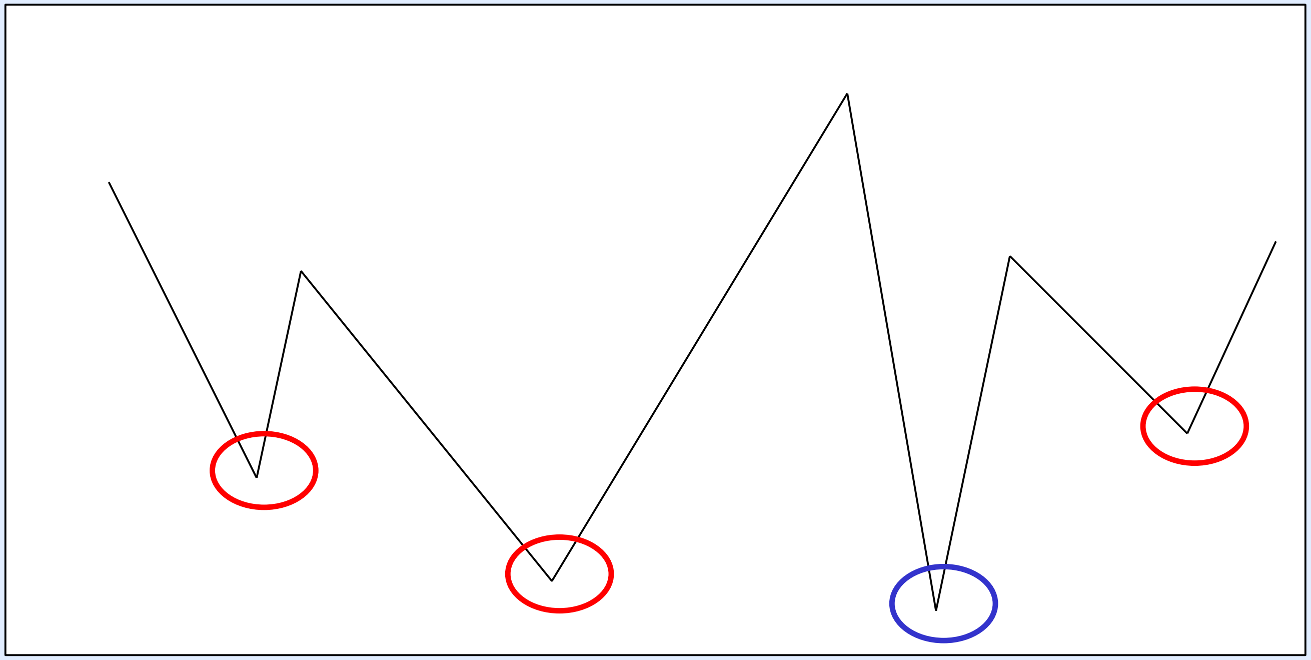
Convex vs. Non-convex Ex.



Affine – border case of convexity

- Convex, min. easy to find

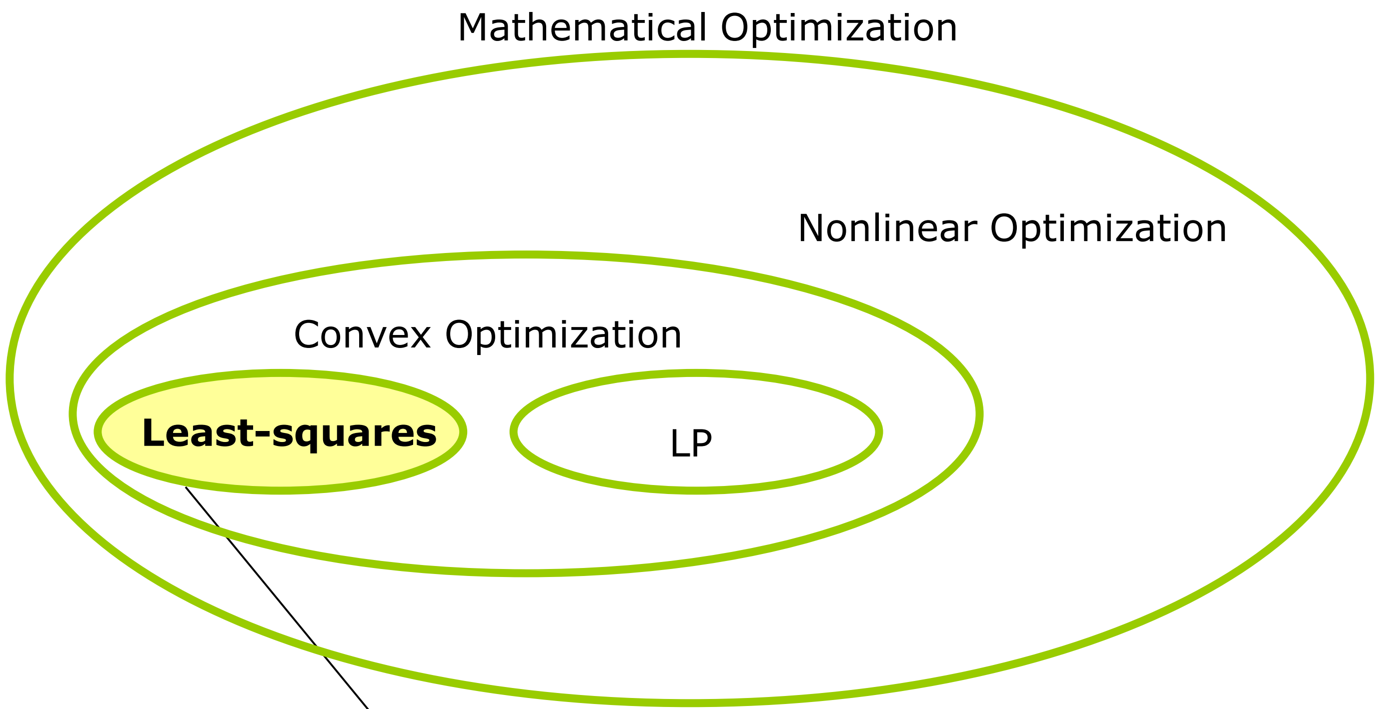
Convex vs. Non-convex Ex.



- Non-convex, easy to get stuck in a local min.
- **Can't rely on only local search techniques**

Non-convex

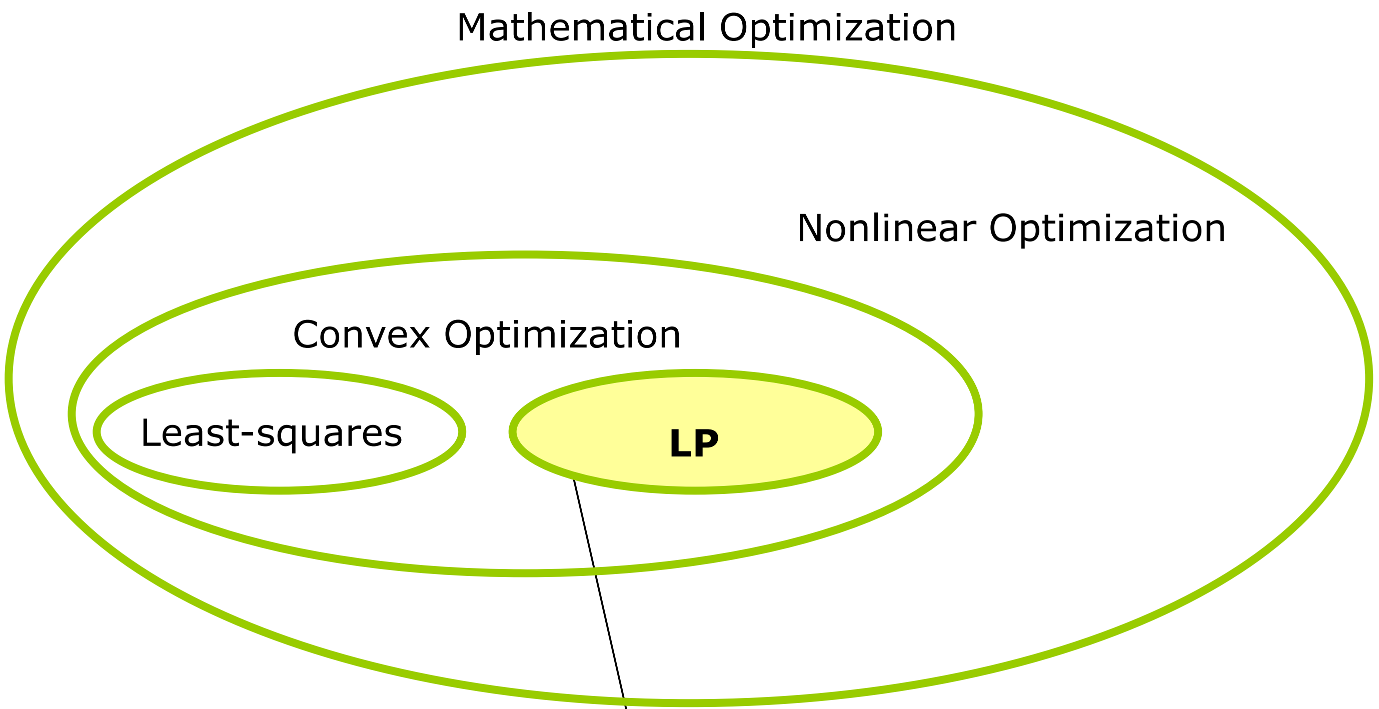
- Some non-convex problems highly multi-modal, or NP hard
- Could be forced to search all solutions, or hope stochastic search is successful
- Cannot guarantee best solution, inefficient
- Harder to make performance guarantees with approximate solutions



minimize $\|Ax - b\|_2^2$

- Analytical solution
- Good algorithms and software
- High accuracy and high reliability
- Time complexity: $C \cdot n^2 k$

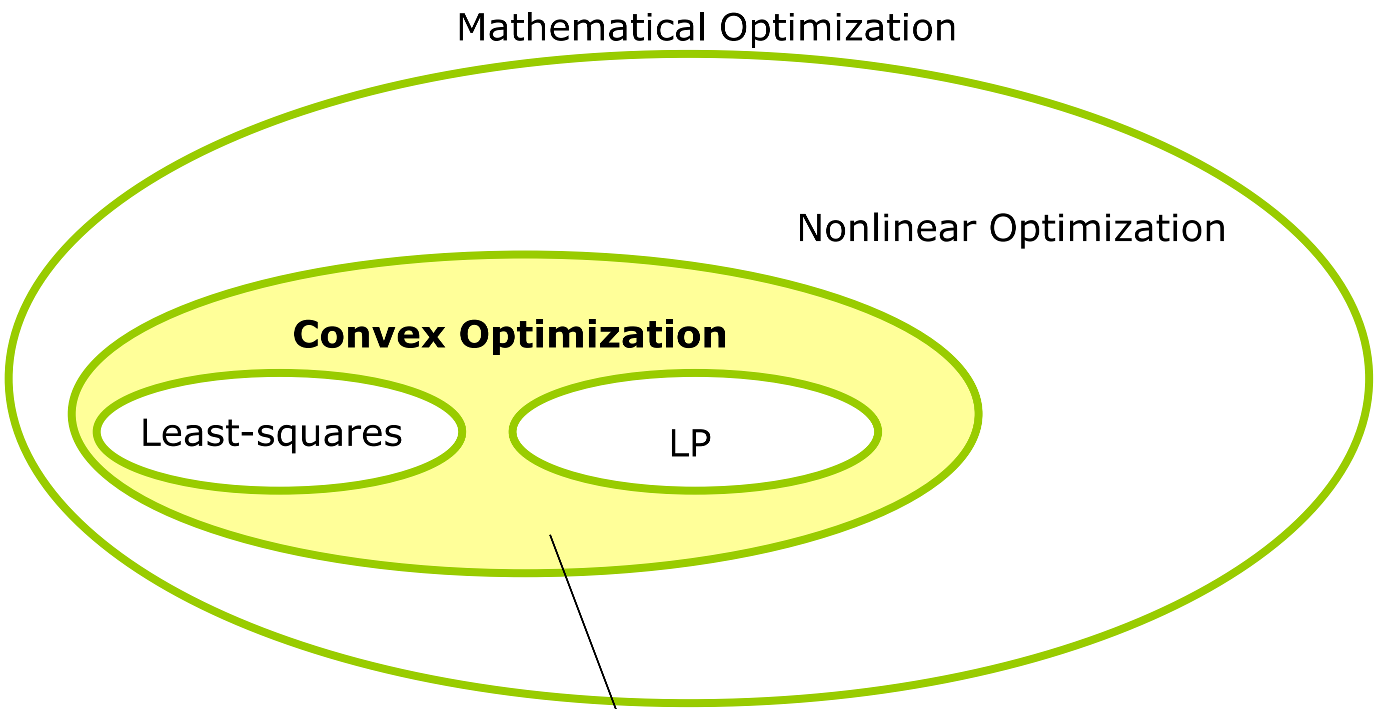
A mature technology!



minimize $c^T x$
subject to $a_i^T x \le b_i$
$i = 1, \ldots, m$

- No analytical solution
- Algorithms and software
- Reliable and efficient
- Time complexity: $C \cdot n^2 m$

Also a mature technology!



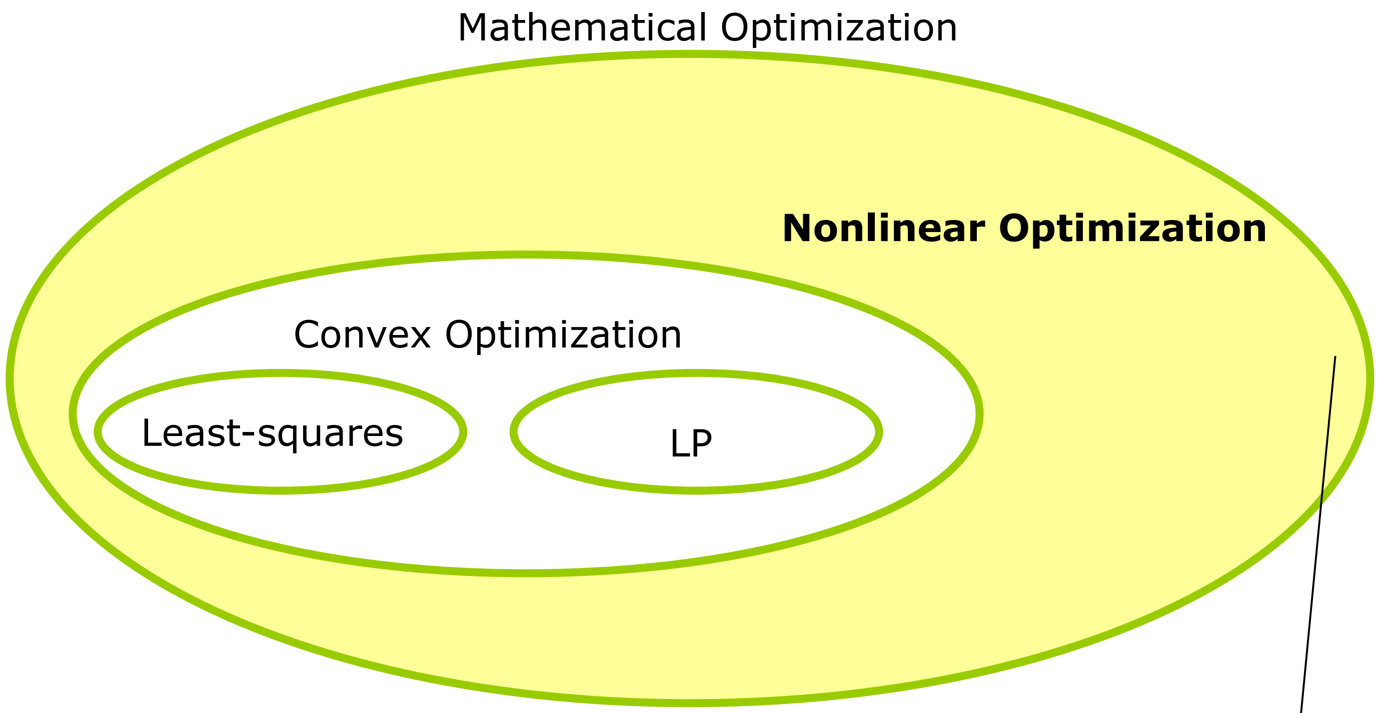
$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \le b_i, \quad i = 1, \dots, m \end{array}$$

- No analytical solution
- Algorithms and software
- Reliable and efficient
- Time complexity (roughly)

  $\propto \max\{n^3, n^2 m, F\}$

F is cost of evaluating f_i's and their first and second derivatives

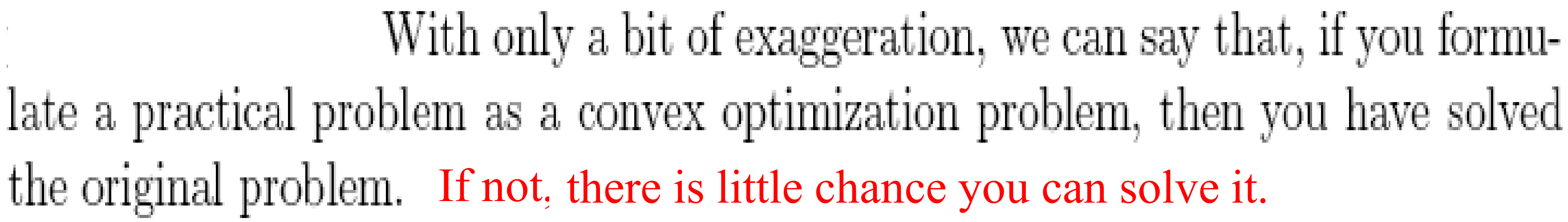
Almost a ~~mature~~ technology!



- Sadly, no effective methods to solve
- Only approaches with some compromise
- Local optimization: *"more art than technology"*
- Global optimization: greatly compromised efficiency
- Help from convex optimization
  1) Initialization 2) Heuristics 3) Bounds

Far from a technology! (something to avoid)

Why Study Convex Optimization



With only a bit of exaggeration, we can say that, if you formulate a practical problem as a convex optimization problem, then you have solved the original problem. If not, there is little chance you can solve it.

-- Section 1.3.2, p8, *Convex Optimization*

Recognizing Convex Optimization Problems

- often difficult to recognize
- many tricks for transforming problems into convex form
- surprisingly many problems can be solved via convex optimization

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \le b_i, \quad i = 1, \ldots, m \end{array}$$

Least-squares

A *least-squares* problem is an optimization problem with no constraints (*i.e.*, $m = 0$) and an objective which is a sum of squares of terms of the form $a_i^T x - b_i$:

$$\text{minimize} \quad f_0(x) = \|Ax - b\|_2^2 = \sum_{i=1}^k (a_i^T x - b_i)^2. \tag{1.4}$$

Here $A \in \mathbf{R}^{k \times n}$ (with $k \ge n$), a_i^T are the rows of A, and the vector $x \in \mathbf{R}^n$ is the optimization variable.

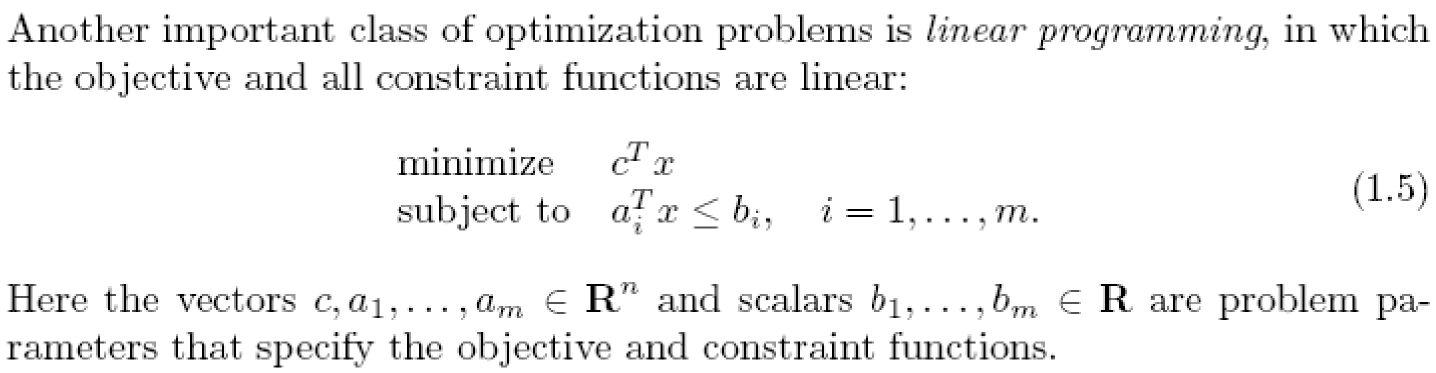
$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^n |x_i|^p \right)^{1/p}.$$

Analytical Solution of Least-squares

$$\text{minimize} \quad f_0(x) = \|Ax - b\|_2^2 = \sum_{i=1}^{k} (a_i^T x - b_i)^2.$$

- Set the derivative to zero:
  - $\frac{df_0(x)}{dx} = 0$
  - $(A^T A)2x - 2Ab = 0$
  - $(A^T A)x = Ab$
- Solve this system of linear equations

Linear Programming (LP)



Another important class of optimization problems is *linear programming*, in which the objective and all constraint functions are linear:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & a_i^T x \leq b_i, \quad i = 1, \ldots, m. \end{array} \tag{1.5}$$

Here the vectors $c, a_1, \ldots, a_m \in \mathbf{R}^n$ and scalars $b_1, \ldots, b_m \in \mathbf{R}$ are problem parameters that specify the objective and constraint functions.

- no analytical formula for solution
- reliable and efficient algorithms and software

Rong Jin

To sum up

- We introduced some important concepts in machine learning and optimization
- We introduced popular machine learning methods
- We talked about loss functions and how we can optimize them using gradient descent