Midterm Review

David Rosenberg

New York University

November 1, 2015

Typical Sequence of Events at Deployment Time

Many problem domains can be formalized as follows:

1. Observe input x in input space $\mathcal{X}$.
2. Take action a in action space $\mathcal{A}$.
3. Observe outcome y in output space $\mathcal{Y}$.
4. Evaluate action in relation to the outcome: $\ell(a, y)$.

Some Formalization

The Spaces

- $\mathcal{X}$: input space
- $\mathcal{Y}$: output space
- $\mathcal{A}$: action space

Decision Function

A **decision function** produces an action $a \in \mathcal{A}$ for any input $x \in \mathcal{X}$:

$$\begin{array}{rccc} f: & \mathcal{X} & \to & \mathcal{A} \\ & x & \mapsto & f(x) \end{array}$$

Loss Function

A **loss function** evaluates an action in the context of the output y.

$$\begin{array}{rccc} \ell: & \mathcal{A} \times \mathcal{Y} & \to & \mathbf{R}^{\geqslant 0} \\ & (a, y) & \mapsto & \ell(a, y) \end{array}$$

Action Spaces

- $\mathcal{A} = \{-1, 1\}$ [hard classification, as used in AdaBoost]
- $\mathcal{A} = \mathbf{R}$ [regression or soft classification]
- $\mathcal{A}$=$\{$Probability distributions a space $\mathcal{Y}\}$

Setup for Statistical Learning Theory

Data Generating Assumption

All pairs $(X, Y) \in \mathcal{X} \times \mathcal{Y}$ are drawn i.i.d. from some **unknown** $P_{\mathcal{X} \times \mathcal{Y}}$.

Definition

The **expected loss** or **"risk"** of a decision function $f : \mathcal{X} \to \mathcal{A}$ is

$$R(f) = \mathbb{E}\ell(f(X), Y),$$

where the expectation taken is over $(X, Y) \sim P_{\mathcal{X} \times \mathcal{Y}}$.

Definition

A **Bayes decision function** $f^* : \mathcal{X} \to \mathcal{A}$ is a function that achieves the *minimal risk* (called the **Bayes risk)** among all possible functions:

$$R(f^*) = \inf_f R(f).$$

The Empirical Risk Functional

Can we estimate $R(f)$ without knowing $\mathcal{P}_{\mathcal{X}\times\mathcal{Y}}$?

Assume we have sample data

Let $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ be drawn i.i.d. from $\mathcal{P}_{\mathcal{X}\times\mathcal{Y}}$.

- The **empirical risk** of $f : \mathcal{X} \to \mathcal{A}$ with respect to $\mathcal{D}_n$ is

$$\hat{R}_n(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(X_i), Y_i).$$

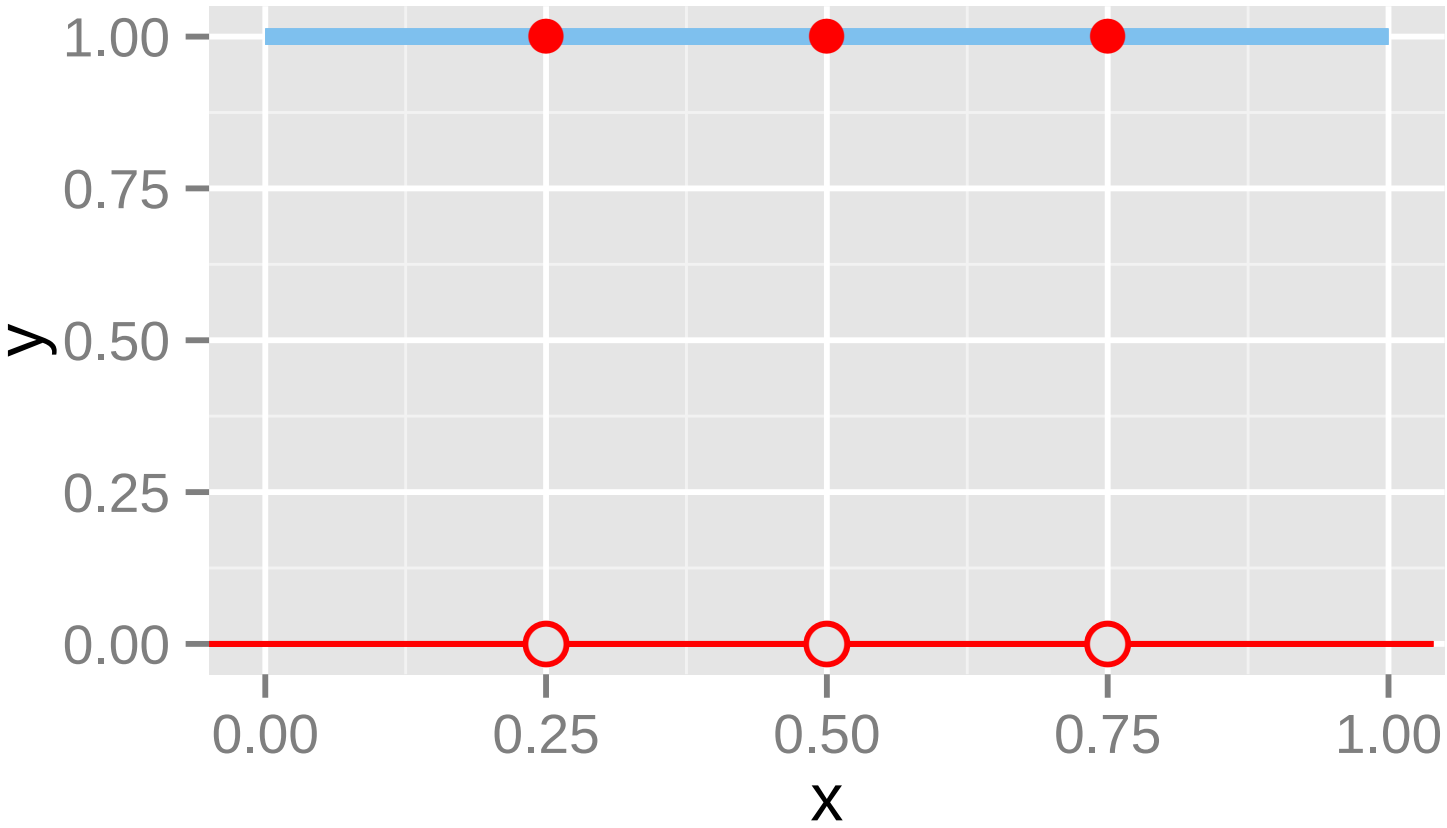
- A function $\hat{f}$ is an **empirical risk minimizer** if

$$\hat{R}_n(\hat{f}) = \inf_f \hat{R}_n(f),$$

where the minimum is taken over all functions.

Empirical Risk Minimization

$P_X = \text{Uniform}[0,1]$, $Y \equiv 1$ (i.e. Y is always 1).



Under square loss or 0/1 loss: Empirical Risk $= 0$. Risk $= 1$.
So unconstrained ERM doesn't work here.

Constrained Empirical Risk Minimization

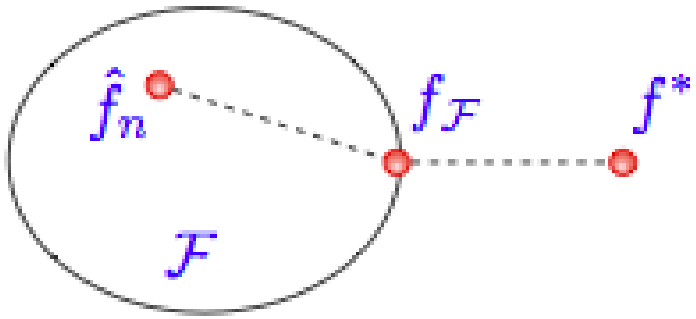
- Hypothesis space $\mathcal{F}$ is a set of functions mapping $\mathcal{X} \to \mathcal{A}$
- **Empirical risk minimizer** (ERM) in $\mathcal{F}$ is $\hat{f} \in \mathcal{F}$, where

$$\hat{R}(\hat{f}) \quad = \quad \inf_{f \in \mathcal{F}} \hat{R}(f) = \inf_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(f(X_i), Y_i).$$

- **Risk minimizer** in $\mathcal{F}$ is $f^*_{\mathcal{F}} \in \mathcal{F}$, where

$$R(f^*_{\mathcal{F}}) = \inf_{f \in \mathcal{F}} R(f) = \inf_{f \in \mathcal{F}} \mathbb{E}\ell(f(X), Y)$$

Error Decomposition



$$f^* = \arg\min_f \mathbb{E}\ell(f(X), Y)$$

$$f_{\mathcal{F}} = \arg\min_{f \in \mathcal{F}} \mathbb{E}\ell(f(X), Y))$$

$$\hat{f}_n = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i)$$

- **Approximation Error** (of $\mathcal{F}$) $= R(f_{\mathcal{F}}) - R(f^*)$
- **Estimation error** (of $\hat{f}_n$ in $\mathcal{F}$) $= R(\hat{f}_n) - R(f_{\mathcal{F}})$

Approximation Error

- Approximation error is a property of the class $\mathcal{F}$
- It's our penalty for restricting to $\mathcal{F}$ rather than considering all measurable functions
  - Approximation error is the minimum risk possible with $\mathcal{F}$ (even with infinite training data)
- *Bigger* $\mathcal{F}$ mean *smaller* approximation error.

Estimation Error

- *Estimation error*: The performance hit for choosing f using finite training data
  - *Equivalently*: It's the hit for not knowing the true risk, but only the empirical risk.
- *Smaller* $\mathcal{F}$ means *smaller* estimation error.
- *Under typical conditions:* "With infinite training data, estimation error goes to zero."
  - Infinite training data solves the *statistical* problem, which is not knowing the true risk.]

Optimization Error

- Does unlimited data solve our problems?
- There's still the *algorithmic* problem of *finding* $\hat{f}_n \in \mathcal{F}$.
- For nice choices of loss functions and classes $\mathcal{F}$, the algorithmic problem can be solved (to any desired accuracy).
  - Takes time! Is it worth it?
- For trees, can't optimize exactly.
- **Optimization error:** If $\tilde{f}_n$ is the function our optimization method returns, and $\hat{f}_n$ is the empirical risk minimizer, then the optimization error is $R(\tilde{f}_n) - R(\hat{f}_n)$
- NOTE: May have $R(\tilde{f}_n) < R(\hat{f}_n)$, since $\hat{f}_n$ may overfit more than $\tilde{f}_n$!

Error Decomposition

Definition

The **excess risk** of f is the amount by which the risk of f exceeds the Bayes risk.

$$\begin{aligned}\textbf{Excess Risk}(\tilde{f}_n) &= R(\tilde{f}_n) - R(f^*) \\ &= \underbrace{R(\tilde{f}_n) - R(\hat{f}_n)}_{\text{optimization error}} + \underbrace{R(\hat{f}_n) - R(f^*_{\mathcal{F}})}_{\text{estimation error}} + \underbrace{R(f^*_{\mathcal{F}}) - R(f^*)}_{\text{approximation error}}\end{aligned}$$

Complexity Measures for Decision Functions

- Depth of a decision tree
- Degree of a polynomial
- How about for **linear** models?
  - ℓ_0 complexity: number of non-zero coefficients
  - ℓ_1 "lasso" complexity: $\sum_{i=1}^{d} |w_i|$, for coefficients $w_1, \ldots, w_d$
  - ℓ_2 "ridge" complexity: $\sum_{i=1}^{d} w_i^2$ for coefficients $w_1, \ldots, w_d$

Nested Hypothesis Spaces from Complexity Measure

- Hypothesis space: $\mathcal{F}$
- Complexity measure $\Omega : \mathcal{F} \to \mathbf{R}^{\geqslant 0}$
- Consider all functions in $\mathcal{F}$ with complexity **at most** r:

$$\mathcal{F}_r = \{f \in \mathcal{F} \mid \Omega(f) \leqslant r\}$$

  - If Ω is a norm on $\mathcal{F}$, this is a **ball of radius** r in $\mathcal{F}$.
- Increasing complexities: $r = 0, 1.2, 2.6, 5.4, \ldots$ gives nested spaces:

$$\mathcal{F}_0 \subset \mathcal{F}_{1.2} \subset \mathcal{F}_{2.6} \subset \mathcal{F}_{5.4} \subset \cdots \subset \mathcal{F}$$

Constrained Empirical Risk Minimization

Constrained ERM (Ivanov regularization)

For complexity measure $\Omega : \mathcal{F} \to \mathbf{R}^{\geqslant 0}$ and fixed $r \geqslant 0$,

$$\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \ell(f(x_i), y_i)$$
$$\text{s.t. } \Omega(f) \leqslant r$$

- Choose r using validation data or cross-validation.
- Each r corresponds to a different hypothesis spaces. Could also write:

$$\min_{f \in \mathcal{F}_r} \sum_{i=1}^{n} \ell(f(x_i), y_i)$$

Penalized Empirical Risk Minimization

Penalized ERM (Tikhonov regularization)

For complexity measure $\Omega : \mathcal{F} \to \mathbf{R}^{\geqslant 0}$ and fixed $\lambda \geqslant 0$,

$$\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \ell(f(x_i), y_i) + \lambda \Omega(f)$$

- Choose λ using validation data or cross-validation.

Ridge Regression: Workhorse of Modern Data Science

Ridge Regression (Tikhonov Form)

The ridge regression solution for regularization parameter $\lambda \geqslant 0$ is

$$\hat{w} = \arg\min_{w \in \mathbf{R}^d} \sum_{i=1}^{n} \left\{ w^T x_i - y_i \right\}^2 + \lambda \|w\|_2^2,$$

where $\|w\|_2^2 = w_1^2 + \cdots + w_d^2$ is the square of the ℓ_2-norm.

Ridge Regression (Ivanov Form)

The ridge regression solution for complexity parameter $r \geqslant 0$ is

$$\hat{w} = \arg\min_{\|w\|_2^2 \leqslant r} \sum_{i=1}^{n} \left\{ w^T x_i - y_i \right\}^2.$$

Lasso Regression: Workhorse (2) of Modern Data Science

Lasso Regression (Tikhonov Form)

The lasso regression solution for regularization parameter $\lambda \geqslant 0$ is

$$\hat{w} = \arg\min_{w \in \mathbf{R}^d} \sum_{i=1}^{n} \left\{ w^T x_i - y_i \right\}^2 + \lambda \|w\|_1,$$

where $\|w\|_1 = |w_1| + \cdots + |w_d|$ is the ℓ_1-norm.

Lasso Regression (Ivanov Form)

The lasso regression solution for complexity parameter $r \geqslant 0$ is

$$\hat{w} = \arg\min_{\|w\|_1 \leqslant r} \sum_{i=1}^{n} \left\{ w^T x_i - y_i \right\}^2.$$

Lasso Gives Feature Sparsity: So What?

- Time/expense to compute/buy features
- Memory to store features (e.g. real-time deployment)
- Identifies the important features
- Better prediction? sometimes
- As a feature-selection step for training a slower non-linear model

Loss Functions for Regression

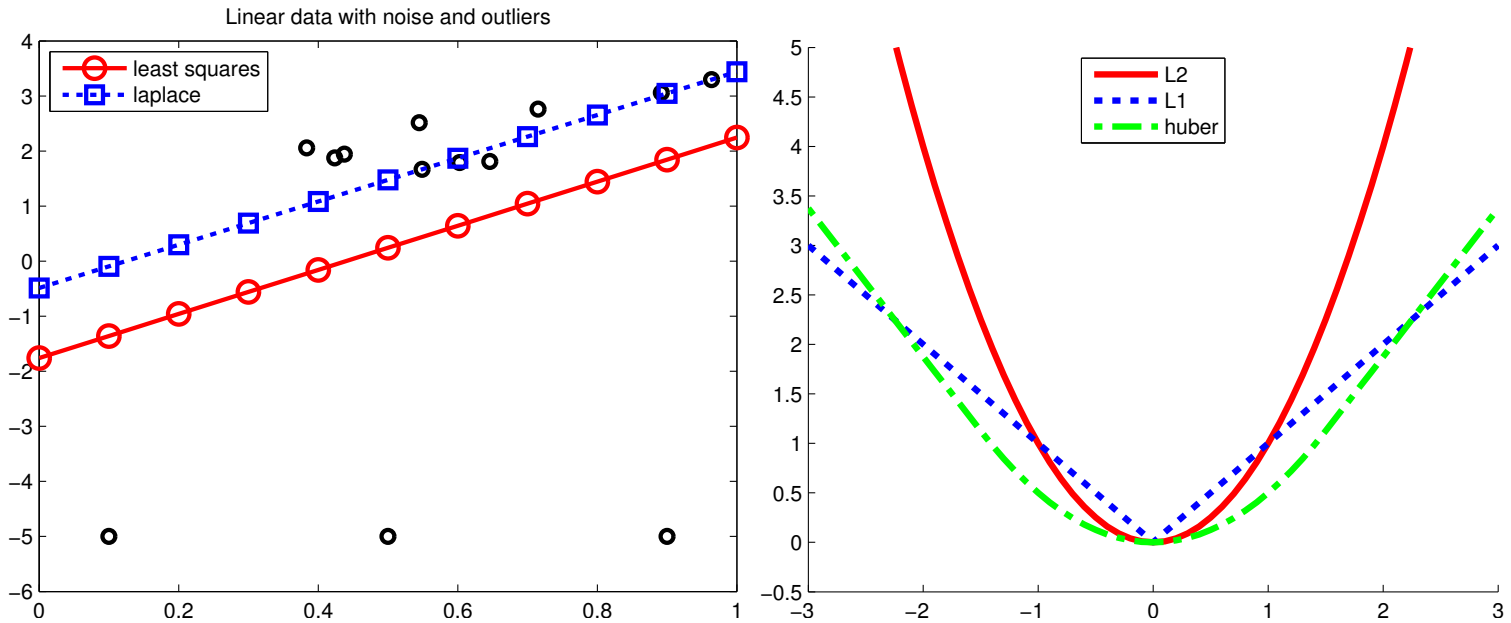
- Regression losses usually only depend on the **residual**:

$$r = y - \hat{y}$$

$$(\hat{y}, y) \mapsto \ell(r) = \ell(y - \hat{y})$$

Some Losses for Regression

- **Square** or ℓ_2 Loss: $\ell(r) = r^2$ (not robust)
- **Absolute** or **Laplace** or ℓ_1 Loss: $\ell(r) = |r|$ (not differentiable)
  - gives **median regression**
- **Huber** Loss: Quadratic for $|r| \leqslant \delta$ and linear for $|r| > \delta$ (robust and differentiable)



KPM Figure 7.6

The Classification Problem: Real-Valued Predictions

- Action space $\mathcal{A} = \mathbf{R}$ Output space $\mathcal{Y} = \{-1, 1\}$
- Prediction function $f : \mathcal{X} \to \mathbf{R}$

Definition

The value $f(x)$ is called the **score** for the input x. Generally, the magnitude of the score represents the **confidence of our prediction**.

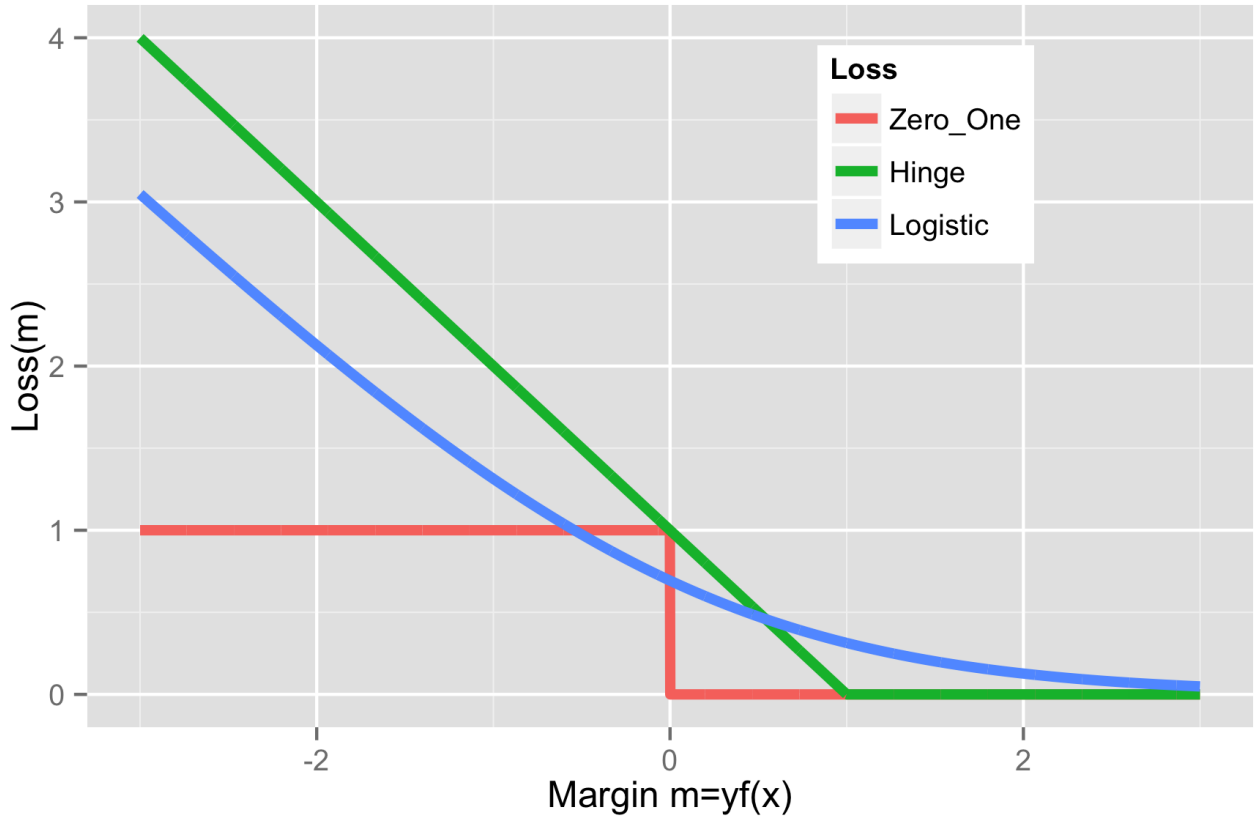
Definition

The **margin** on an example (x, y) is $yf(x)$. The margin is a measure of how **correct** we are.

- We want to **maximize the margin**.
- Most classification losses depend only on the margin.

Classification Losses

Logistic/Log loss: $\ell_{\text{Logistic}} = \log\left(1 + e^{-m}\right)$



Logistic loss is differentiable. Never enough margin for logistic loss.
How many support vectors?

(Soft Margin) Linear Support Vector Machine

- Hypothesis space $\mathcal{F} = \left\{ f(x) = w^T x \mid w \in \mathbf{R}^d \right\}$.
- Loss $\ell(m) = (1-m)_+$
- ℓ_2 regularization

$$\min_{w \in \mathbf{R}^d, b \in \mathbf{R}} \frac{1}{2}\|w\|^2 + \frac{c}{n} \sum_{i=1}^{n} \left(1 - y_i \left[w^T x_i + b\right]\right)_+ .$$

- unconstrained optimization
- not differentiable
- Can we reformulate into a differentiable problem?

SVM as a Quadratic Program

- The SVM optimization problem is equivalent to

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|^2 + \frac{c}{n}\sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & \xi_i \geqslant 0 \text{ for } i = 1, \ldots, n \\ & \xi_i \geqslant \left(1 - y_i \left[w^T x_i + b\right]\right) \text{ for } i = 1, \ldots, n \end{aligned}$$

- Differentiable objective function
- A quadratic program that can be solved by any off-the-shelf QP solver.

SVM Dual Problem

- Can eliminate the λ variables:

$$\begin{aligned} \sup_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_j^T x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & \alpha_i \in \left[0, \frac{c}{n}\right] \; i = 1, \ldots, n. \end{aligned}$$

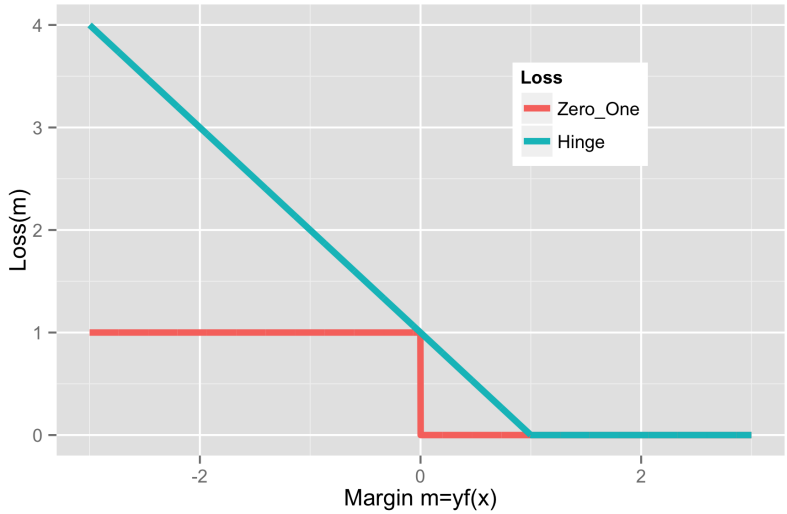
  Constraints are **box constraints**. (Simpler than primal constraints.)
- If α^* is a solution to the dual problem, then

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i x_i.$$

- Since $\alpha_i \in [0, \frac{c}{n}]$, we see that c controls the amount of weight we can put on any single example

The Margin

- For notational convenience, define $f^*(x) = x_i^T w^* + b^*$.
- Margin $yf^*(x)$



- Incorrect classification: $yf^*(x) \leqslant 0$.
- Margin error: $yf^*(x) < 1$.
- "On the margin": $yf^*(x) = 1$.
- "Good side of the margin": $yf^*(x) > 1$.

Complementary Slackness Results: Summary

$$\begin{aligned}
\alpha_i^* = 0 &\implies y_i f^*(x_i) \geqslant 1 \\
\alpha_i^* \in \left(0, \frac{c}{n}\right) &\implies y_i f^*(x_i) = 1 \\
\alpha_i^* = \frac{c}{n} &\implies y_i f^*(x_i) \leqslant 1
\end{aligned}$$

$$\begin{aligned}
y_i f^*(x_i) < 1 &\implies \alpha_i^* = \frac{c}{n} \\
y_i f^*(x_i) = 1 &\implies \alpha_i^* \in \left[0, \frac{c}{n}\right] \\
y_i f^*(x_i) > 1 &\implies \alpha_i^* = 0
\end{aligned}$$

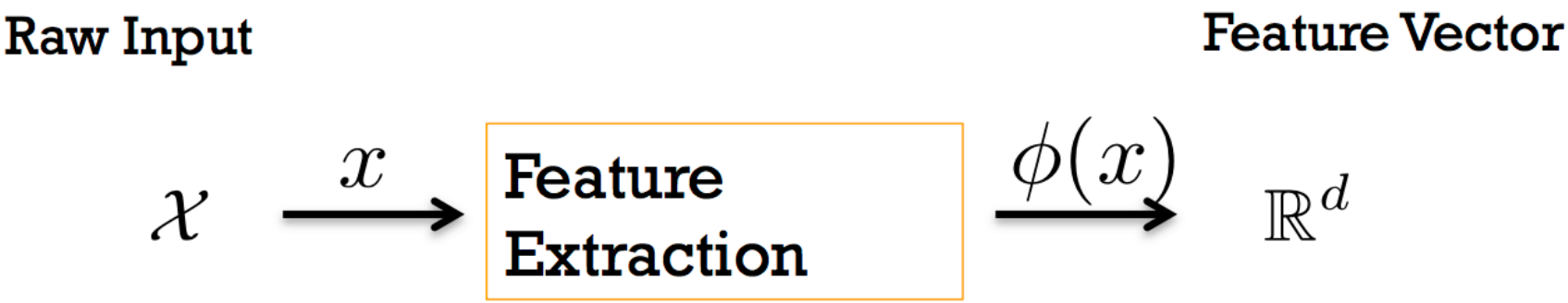
The Input Space $\mathcal{X}$

- Our general learning theory setup: no assumptions about $\mathcal{X}$
- But $\mathcal{X} = \mathbf{R}^d$ for the specific methods we've developed:
  - Ridge regression
  - Lasso regression
  - Linear SVM

Feature Extraction

Definition

Mapping an input from $\mathcal{X}$ to a vector in $\mathbf{R}^d$ is called **feature extraction** or **featurization**.



- e.g. Quadratic feature map: $\mathcal{X} = \mathbf{R}^d$

$$\phi(x) = (x_1, \ldots, x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_ix_j, \ldots \sqrt{2}x_{d-1}x_d)^T.$$

High-Dimensional Features Good but Expensive

- To get **expressive** hypothesis spaces using linear models,
  - need high-dimensional feature spaces
- But more costly in terms of computation and memory.

Some Methods Can Be "Kernelized"

Definition

A method is **kernelized** if inputs only appear inside inner products: $\langle \phi(x), \phi(y) \rangle$ for $x, y \in \mathcal{X}$.

- The function

$$k(x, y) = \langle \phi(x), \phi(y) \rangle$$

  is called the **kernel** function.

Kernel Evaluation Can Be Fast

Example

Quadratic feature map

$$\phi(x) = (x_1, \ldots, x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_ix_j, \ldots \sqrt{2}x_{d-1}x_d)^T$$

has dimension $O(d^2)$, but

$$k(w,x) = \langle \phi(w), \phi(x) \rangle = \langle w, x \rangle + \langle w, x \rangle^2$$

- Naively explicit computation of $k(w,x)$: $O(d^2)$
- Implicit computation of $k(w,x)$: $O(d)$

Recap

1. Given a kernelized ML algorithm.
2. Can swap out the inner product for a new kernel function.
3. New kernel may correspond to a high dimensional feature space.
4. Computational cost is independent of feature dimension.
    1. However, now has a quadratic dependence on the size of the data set.

Ridge Regression

- Recall the ridge regression objective:

$$J(w) = \|Xw - y\|^2 + \lambda \|w\|^2.$$

- Differentiating and setting equal to zero ,we get

$$\left(X^T X + \lambda I\right) w \quad = \quad X^T y$$

Kernelizing Ridge Regression

- So we have, for $\lambda > 0$:

$$\begin{aligned} (X^T X + \lambda I) w &= X^T y \\ w &= \frac{1}{\lambda} X^T (y - Xw) \\ w &= X^T \alpha \end{aligned}$$

  for $\alpha = \lambda^{-1}(y - Xw) \in \mathbf{R}^n$.

- So w is "**in the span of the data**":

$$w = \begin{pmatrix} | & \cdots & | \\ x_1 & \cdots & x_n \\ | & \cdots & | \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \alpha_1 x_1 + \cdots \alpha_n x_n$$

Kernelizing Ridge Regression

- So plugging in $w = X^T \alpha$ to

$$\begin{aligned} \alpha &= \lambda^{-1}(y - Xw) \\ \lambda\alpha &= y - XX^T\alpha \\ XX^T\alpha + \lambda\alpha &= y \\ (XX^T + \lambda I)\,\alpha &= y \\ \alpha &= (\lambda I + XX^T)^{-1} y \end{aligned}$$

- When can we swap in a new kernel matrix for XX^T?

Mercer's Theorem

Theorem

A symmetric function $k(w,x)$ can be expressed an inner product

$$k(w,x) = \langle \phi(w), \phi(x) \rangle$$

for some ϕ if and only if $k(w,x)$ is ***positive semidefinite.***

- If we start with a psd kernel, can we generate more?

The Kernel Matrix (or the Gram Matrix)

Definition

For a set of $\{x_1, \ldots, x_n\}$ and an inner product $\langle \cdot, \cdot \rangle$on the set, the **kernel matrix** or the **Gram matrix** is defined as

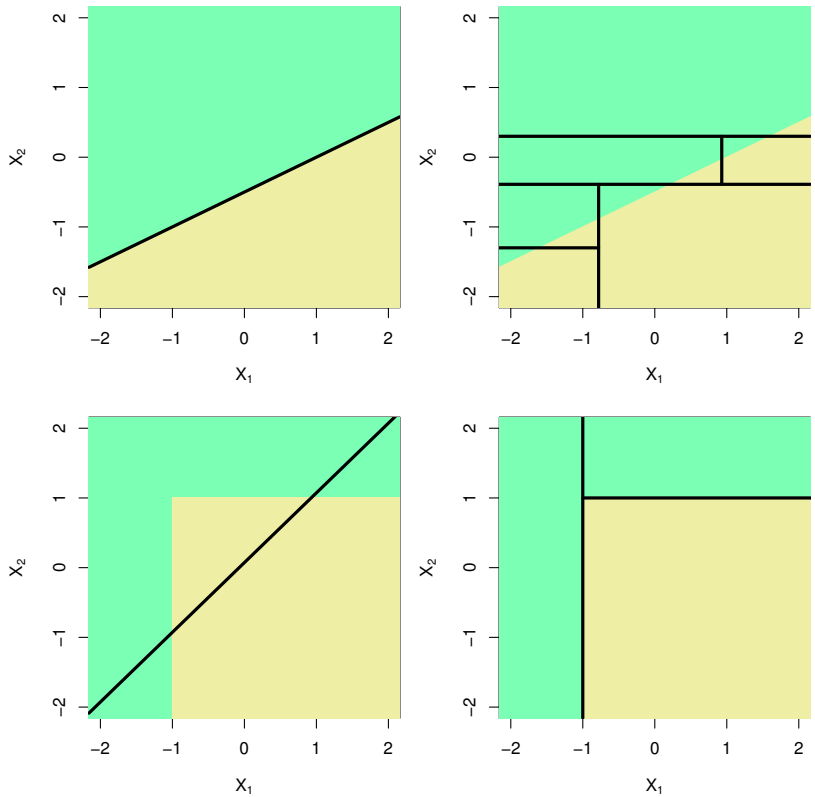
$$K = \left(\langle x_i, x_j \rangle \right)_{i,j} = \begin{pmatrix} \langle x_1, x_1 \rangle & \cdots & \langle x_1, x_n \rangle \\ \vdots & \ddots & \cdots \\ \langle x_n, x_1 \rangle & \cdots & \langle x_n, x_n \rangle \end{pmatrix}.$$

Then for the standard Euclidean inner product $\langle x_i, x_j \rangle = x_i^T x_j$, we have

$$K = XX^T$$

Trees vs Linear Models

- Trees have to work much harder to capture linear relations.



From *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

Comments about Trees

- Trees make no use of **geometry**
  - No inner products or distances
  - called a "nonmetric" method
  - **Feature scale irrelevant**
- Predictions are not continuous
  - not so bad for classification
  - may not be desirable for regression

Ensembles: Parallel vs Sequential

- Ensemble methods combine multiple models
- **Parallel ensembles**: each model is built independently
  - e.g. bagging and random forests
  - Main Idea: Combine many (high complexity, low bias) models to reduce variance
- **Sequential ensembles**:
  - Models are generated sequentially
  - Try to add new models that do well where previous models lack

Averaging Independent Prediction Functions

- Let $Z_1, \ldots, Z_n$ be independent r.v's with mean μ and variance σ^2.
- Average has the same expected value but smaller variance:

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} Z_i\right] = \mu \qquad \text{Var}\left[\frac{1}{n}\sum_{i=1}^{n} Z_i\right] = \frac{\sigma^2}{n}.$$

- Prediction functions? Suppose we have B independent training sets.
- Let $\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)$ be the prediction models for each set.
- Define the average prediction function as:

$$\hat{f}_{\text{avg}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x).$$

- Variance of average?
- In practice we don't have B independent training sets...
  - Instead, we can use **the bootstrap**....

The Bootstrap Sample

Definition

A **bootstrap sample** from $\mathcal{D} = \{X_1, \ldots, X_n\}$ is a sample of size n drawn *with replacement* from $\mathcal{D}$.

- In a bootstrap sample, some elements of $\mathcal{D}$
  - will show up multiple times,
  - some won't show up at all.
- So we expect ~63.2% of elements of $\mathcal{D}$ will show up at least once.

Bagging

- Suppose we had B bootstrap samples from a training set.
- **Bagging estimator** given as

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b^*(x),$$

where $\hat{f}_b^*$ is trained on the b'th bootstrap sample.

Random Forest

Main idea of random forests

Use **bagged decision trees**, but modify the tree-growing procedure to reduce the correlation between trees.

- **Key step** in random forests:
  - When constructing each tree node, restrict choice of splitting variable to a randomly chosen subset of features of size m.
- Typically choose $m \approx \sqrt{p}$, where p is the number of features.
- Can choose m using cross validation.

AdaBoost - Rough Sketch

- Training set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$.
- Start with equal weight on all training points $w_1 = \cdots = w_n = 1$.
- Repeat for $m = 1, \ldots, M$:
  - Fit weak classifier $G_m(x)$ to weighted training points
  - Increase weight on points $G_m(x)$ misclassifies
- Final prediction $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.
- The α_m's are nonnegative,
  - larger when G_m fits its weighted $\mathcal{D}$ well
  - smaller when G_m fits weighted $\mathcal{D}$ less well

Adaptive Basis Function Model

- Hypothesis space $\mathcal{F}$
  - Can be classifiers or regression functions
  - These would be the "weak classifiers" or "base classifiers"
- An **adaptive basis function expansion** over $\mathcal{F}$ is

$$f(x) = \sum_{m=1}^{M} \nu_m h_m(x),$$

  - Each $h_m \in \mathcal{F}$ is chosen in a learning process, and
  - ν_m are **expansion coefficients.**
- For example, $\mathcal{F}$ could be all decision trees of depth at most 4.
- We now discuss one approach to fitting such a model.

Forward Stagewise Additive Modeling

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   1. Compute:

$$(\nu_m, h_m) = \underset{\nu \in \mathbf{R}, h \in \mathcal{F}}{\arg\min} \sum_{i=1}^{n} \ell \left\{ y_i, f_{m-1}(x_i) \underbrace{+\nu h(x_i)}_{\text{new piece}} \right\}.$$

   2. Set $f_m(x) = f_{m-1}(x) + \nu_m h(x)$.
3. Return: $f_M(x)$.

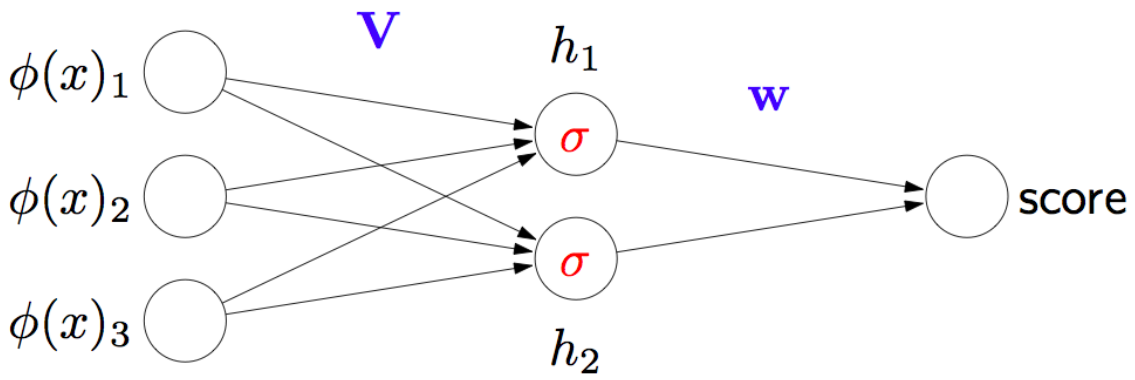
Exponential Loss and AdaBoost

- Take loss function to be

$$\ell(y, f(x)) = \exp(-yf(x)).$$

- Let $\mathcal{F} = \{h(x) : \mathcal{X} \to \{-1, 1\}\}$ be a hypothesis space of weak classifiers.
- Then Forward Stagewise Additive Modeling (FSAM) reduces to AdaBoost.
  - (See HTF Section 10.4 for proof.)

Neural Network



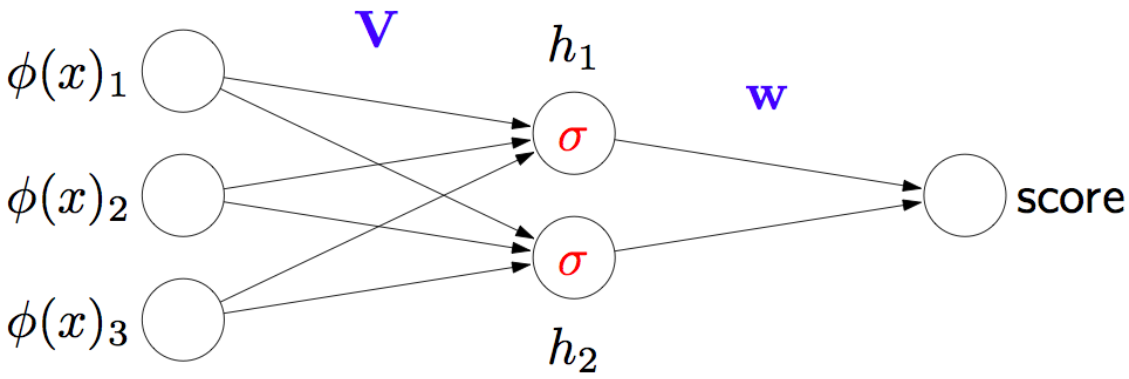
- Score is just

$$\begin{aligned} \text{score} &= w_1 h_1 + w_2 h_2 \\ &= w_1 \sigma(v_1^T \phi(x)) + w_2 \sigma\left(v_2^T \phi(x)\right) \end{aligned}$$

- This is the basic recipe.
  - We can add more hidden nodes.
  - We can add more hidden layers.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Neural Network: Hidden Nodes as Learned Features



- Can interpret h_1 and h_2 as nonlinear features learned from data.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Neural Network: The Hypothesis Space

- What hyperparameters describe a neural network?
  - Number of layers
  - Number of nodes in each hidden layer
  - Activation function (but so many to choose from)
- Example neural network hypothesis space:

$$\mathcal{F} = \left\{ f : \mathbf{R}^d \to \mathbf{R} \mid f \text{ is a NN with 2 hidden layers, 500 nodes in each} \right\}$$

- Functions in $\mathcal{F}$ **parameterized by the weights between nodes**.

Neural Network: Loss Functions and Learning

- Neural networks give a **new hypothesis space**.
- But we can use all the **same loss functions** we've used before.
- Optimization method of choice: **stochastic gradient descent**.

Neural Network: Objective Function

- In our simple network, the output score is given by

$$f(x) = w_1 \sigma(v_1^T \phi(x)) + w_2 \sigma\left(v_2^T \phi(x)\right)$$

- Objective with square loss is then

$$J(w, v) = \sum_{i=1}^{n} \left(y_i - f_{w,v}(x_i)\right)^2$$

- Note: $J(w, v)$ is **not convex**.
  - makes optimization much more difficult
  - accounts for many of the "tricks of the trade"

Learning with Back-Propagation

- Back-propagation is an **algorithm** for computing the SGD gradient
- Mathematically, it's not necessary.
- With lots of chain rule, you can work out the gradient by hand.
- Back-propagation is
  - a clean way to organize the computation of the gradient
  - an efficient way to compute the gradient

Likelihood of a Predicted Distribution

- Suppose we have

$$\mathcal{D} = \{y_1, \ldots, y_n\} \text{ sampled i.i.d. from } p(y).$$

- Then the **likelihood** of $\hat{p}$ for the data $\mathcal{D}$ is defined to be

$$\hat{p}(\mathcal{D}) = \prod_{i=1}^{n} \hat{p}(y_i).$$

- We'll write this as

$$L_{\mathcal{D}}(\hat{p}) := \hat{p}(\mathcal{D})$$

- Special case: If $\hat{p}$ is a probability mass function, then
  - $L_{\mathcal{D}}(\hat{p})$ is the probability of $\mathcal{D}$ under $\hat{p}$.

Probability Estimation as Statistical Learning

- Output space $\mathcal{Y}$ (containing observations from distribution P)
- **Action space**
  $\mathcal{A} = \{p(y) \mid p \text{ is a probability density or mass function on } \mathcal{Y}\}$.
- How to encode our objective of "high likelihood" as a loss function?

- Define loss function as the negative log-likelihood of y under $p(\cdot)$:

$$\begin{array}{rccc} \ell: & \mathcal{A} \times \mathcal{Y} & \to & \mathbf{R} \\ & (p, y) & \mapsto & -\log p(y) \end{array}$$

Generalized Regression as Statistical Learning

- Input space $\mathcal{X}$
- Output space $\mathcal{Y}$
- All pairs (X, Y) are independent with distribution $P_{\mathcal{X} \times \mathcal{Y}}$.
- **Action space**
  $\mathcal{A} = \{p(y) \mid p \text{ is a probability density or mass function on } \mathcal{Y}\}$.
- Hypothesis spaces comprise decision functions $f : \mathcal{X} \to \mathcal{A}$.
  - Given an $x \in \mathcal{X}$, predict a probability distribution $p(y)$ on $\mathcal{Y}$.
- Loss function as before:

$$\begin{array}{rccc} \ell: & \mathcal{A} \times \mathcal{Y} & \to & \mathbf{R} \\ & (p, y) & \mapsto & -\log p(y) \end{array}$$

- ERM gives MLE.

Generalized Regression as Statistical Learning

- The risk of decision function $f : \mathcal{X} \to \mathcal{A}$

$$R(f) = -\mathbb{E}_{X,Y} \log \left[f(X) \right] (Y),$$

where $f(X)$ is a PDF or PMF on $\mathcal{Y}$, and we're evaluating it on Y.

- The empirical risk of f for a sample $\mathcal{D} = \{y_1, \ldots, y_n\} \in \mathcal{Y}$ is

$$\hat{R}(f) = -\sum_{i=1}^{n} \log \left[f(x_i) \right] (y_i).$$

This is called the negative **conditional log-likelihood**.

Linear Probabilistic Classifiers

- Setting: $\mathcal{X} = \mathbf{R}^d$, $\mathcal{Y} = \{0, 1\}$
- For each $X = x$, $p(Y = 1 \mid x) = \theta$. (i.e. Y has a Bernoulli(θ) distribution)
- θ may vary with x.
- For each $x \in \mathbf{R}^d$, just want to predict $\theta \in [0, 1]$.
- Two steps:

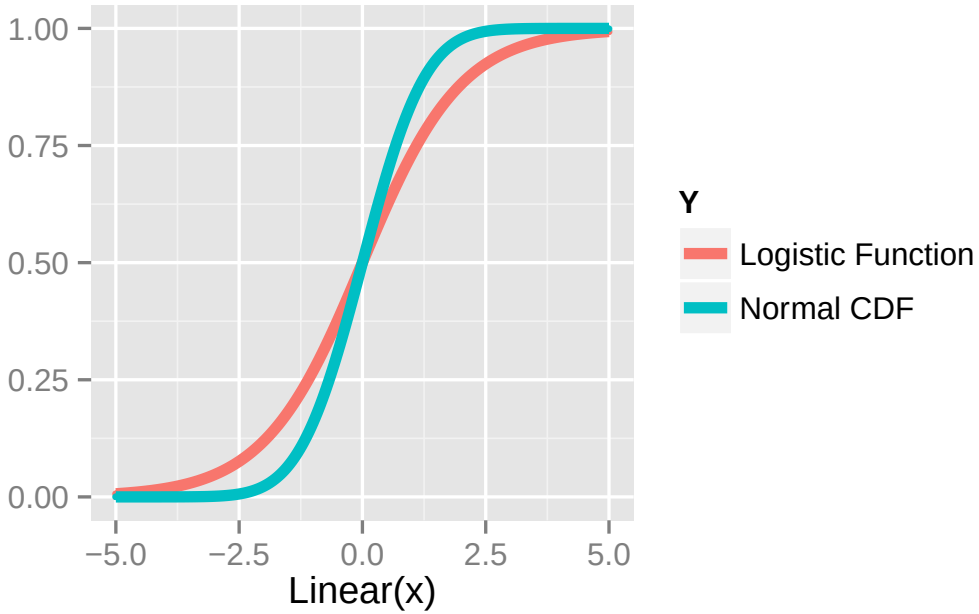
$$\underbrace{x}_{\in \mathbf{R}^D} \mapsto \underbrace{w^T x}_{\in \mathbf{R}} \mapsto \underbrace{f(w^T x)}_{\in [0,1]},$$

  where $f : \mathbf{R} \to [0, 1]$ is called the **transfer** or **inverse link** function.
- Probability model is then

$$p(Y = 1 \mid x) = f(w^T x)$$

Inverse Link Functions

- Two commonly used "inverse link" functions to map from $w^T x$ to θ:



- Logistic function $\Longrightarrow$ Logistic Regression
- Normal CDF $\Longrightarrow$ Probit Regression

Specifying a Natural Exponential Family

- The family is a **natural exponential family** with parameter θ if

$$p_\theta(y) = \frac{1}{Z(\theta)} h(y) \exp\left[\theta^T y\right].$$

- To specify a natural exponential family, we need to choose $h(y)$.
  - Everything else is determined.
- Implicit in choosing $h(y)$ is the choice of the support of the distribution.

Natural Exponential Families: Examples

The following are univariate natural exponential families:

1. Normal distribution with known variance.
2. Poisson distribution
3. Gamma distribution (with known k parameter)
4. Bernoulli distribution (and Binomial with known number of trials)

Generalized Linear Models [with Canonical Link]

- In GLMs, we first choose a natural exponential family.
  - (This amounts to choosing $h(y)$.)
- The idea is to plug in $w^T x$ for the natural parameter.
- This gives models of the following form:

$$p_\theta(y \mid x) = \frac{1}{Z(w^T x)} h(y) \exp\left[(w^T x) y\right].$$

- This is the form we had for Poisson regression.

Generalized Linear Models [with General Link]

- More generally, choose a function ψ so that

$$x \mapsto w^T x \mapsto \psi(w^T x),$$

  where $\theta = \psi(w^T x)$ is the natural parameter for the family.
- So our final prediction (for one-parameter families) is:

$$p_\theta(y \mid x) = \frac{1}{Z(\psi(w^T x))} h(y) \exp\left[\psi(w^T x) y\right].$$

Gradient Descent

Gradient Descent

- Initialize $x = 0$
- repeat
  - $x \leftarrow x - \underbrace{\eta}_{\text{step size}} \nabla f(x)$
- until stopping criterion satisfied

Gradient Descent: Does it scale?

- At every iteration, we compute the gradient at current w:

$$\nabla_w \hat{R}_n(w) = \frac{2}{n} \sum_{i=1}^{n} \underbrace{\left(w^T x_i - y_i\right)}_{i\text{th residual}} x_i$$

- We have to touch all n training points to take a single step. $[O(n)]$
  - Called a **batch optimization** method
- Can we make progress without looking at all the data?

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent

- initialize $w = 0$
- repeat
  - randomly choose training point $(x_i, y_i) \in \mathcal{D}_n$
  - $w \leftarrow w - \eta \underbrace{\nabla_w \ell(f_w(x_i), y_i)}_{\text{Grad(Loss on i'th example)}}$
- until stopping criteria met

How to find the Lasso solution?

- How to solve the Lasso?

$$\min_{w \in \mathbf{R}^d} \sum_{i=1}^{n} \left(w^T x_i - y_i \right)^2 + \lambda |w|_1$$

- $|w|_1$ is not differentiable!

The Lasso as a Quadratic Program

- Substituting $w = w^+ - w^-$ and $|w| = w^+ + w^-$, Lasso problem is:

$$\begin{aligned} \min_{w^+, w^- \in \mathbf{R}^d} & \sum_{i=1}^{n} \left(\left(w^+ - w^- \right)^T x_i - y_i \right)^2 + \lambda \left(w^+ + w^- \right) \\ \text{subject to } & w_i^+ \geqslant 0 \text{ for all } i \\ & w_i^- \geqslant 0 \text{ for all } i \end{aligned}$$

- Objective is **differentiable** (in fact, **convex and quadratic**)
- $2d$ variables vs d variables
- $2d$ constraints vs no constraints
- A "**quadratic program**": a convex quadratic objective with linear constraints.
  - Could plug this into a generic QP solver.

Projected SGD

$$\min_{w^+, w^- \in \mathbf{R}^d} \sum_{i=1}^{n} \left(\left(w^+ - w^- \right)^T x_i - y_i \right)^2 + \lambda \left(w^+ + w^- \right)$$

$$\text{subject to } w_i^+ \geqslant 0 \text{ for all } i$$

$$w_i^- \geqslant 0 \text{ for all } i$$

- Solution:
  - Take a stochastic gradient step
  - "Project" w^+ and w^- into the constraint set
    - In other words, any component of w^+ or w^- is negative, make it 0 .
- Note: Sparsity pattern may change frequently as we iterate

Coordinate Descent Method

Coordinate Descent Method

Goal: Minimize $L(w) = L(w_1, \dots w_d)$ over $w = (w_1, \dots, w_d) \in \mathbf{R}^d$.

- **Initialize** $w^{(0)} = 0$
- **while** not converged:
  - Choose a coordinate $j \in \{1, \dots, d\}$
  - $w_j^{\text{new}} \leftarrow \arg\min_{w_j} L(w_1^{(t)}, \dots, w_{j-1}^{(t)}, \mathbf{w_j}, w_{j+1}^{(t)}, \dots, w_d^{(t)})$
  - $w^{(t+1)} \leftarrow w^{(t)}$
  - $w_j^{(t+1)} \leftarrow w_j^{\text{new}}$
  - $t \leftarrow t+1$

- For when it's easier to minimize w.r.t. one coordinate at a time
- Random coordinate choice $\implies$ **stochastic coordinate descent**
- Cyclic coordinate choice $\implies$ **cyclic coordinate descent**

Coordinate Descent Method for Lasso

- Why mention coordinate descent for Lasso?
- In Lasso, the coordinate minimization has a **closed form solution**!

The Lagrangian

Recall the general optimization problem:

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leqslant 0, \;\; i = 1, \ldots, m \\ & h_i(x) = 0, \;\; i = 1, \ldots p, \end{aligned}$$

Definition

The **Lagrangian** for the general optimization problem is

$$L(x, \lambda, \nu) = f_0(x) + \sum_{I=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x),$$

- λ_i's and ν's are called **Lagrange multipliers**
- λ and ν also called the **dual variables** .

The Primal and the Dual

- Original optimization problem in **primal form:**

$$p^* = \inf_x \sup_{\lambda \succeq 0, \nu} L(x, \lambda, \nu)$$

- The **Lagrangian dual problem**:

$$d^* = \sup_{\lambda \succeq 0, \nu} \inf_x L(x, \lambda, \nu)$$

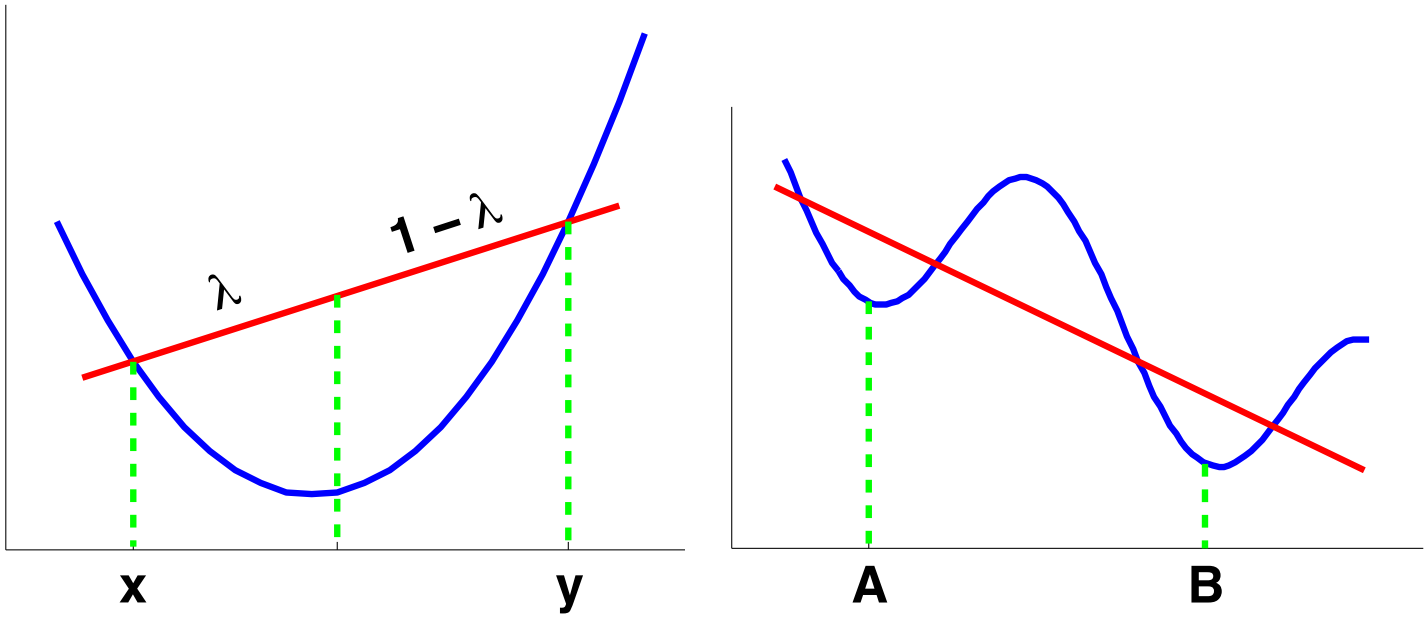
- We showed **weak duality**: $p^* \geqslant d^*$ for any optimization problem

Convex and Concave Functions

Definition

A function $f : \mathbf{R}^n \to \mathbf{R}$ is **convex** if **dom** f is a convex set and if for all $x, y \in$ **dom** f, and $0 \leqslant \theta \leqslant 1$, we have

$$f(\theta x + (1-\theta)y) \leqslant \theta f(x) + (1-\theta) f(y).$$



Convex Optimization Problem: Standard Form

Convex Optimization Problem: Standard Form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leqslant 0, \;\; i = 1, \ldots, m \\ & a_i^T x = b_i, \;\; i = 1, \ldots p \end{array}$$

where $f_0, \ldots, f_m$ are convex functions.
Note: Equality constraints are now linear. Why? [otherwise feasible set won't be convex]

Slater's Constraint Qualifications for Strong Duality

- Sufficient conditions for strong duality in a **convex** problem.
- Roughly: the problem must be **strictly** feasible.
- The domain $\mathcal{D} \subset \mathbf{R}^n$ of an optimization problem is the set on which all the functions are defined.
  - i.e. $f_0, f_1, \ldots, f_m$ are all defined.
  - the domain $\mathcal{D}$ is NOT the feasible set.
- Qualifications when problem domain $\mathcal{D} \subset \mathbf{R}^n$ is an open set:
  - $\exists x$ such that $Ax = b$ and $f_i(x) < 0$ for $i = 1, \ldots, m$
  - For any affine inequality constraints, $f_i(x) \leqslant 0$ is sufficient

Complementary Slackness

- Consider a general optimization problem (i.e. not necessarily convex).
- If we have **strong duality**, we get an interesting relationship between
  - the optimal Lagrange multiplier λ_i and
  - the ith constraint at the optimum: $f_i(x^*)$
- Relationship is called "**complementary slackness**":

$$\lambda_i^* f_i(x^*) = 0$$

- Lagrange multiplier is zero unless the constraint is active at the optimum.