Machine Learning

Slides use Tony Jebara's materials

Topic 1

- Introduction
- Machine Learning: What, Why and Applications
- Syllabus, policies, texts, web page
- Historical Perspective
- Machine Learning Tasks and Tools
- Digit Recognition Example
- Machine Learning Approach
- Deterministic or Probabilistic Approach
- Why Probabilistic?

Machine Learning: What/Why

Statistical Data-Driven Computational Models

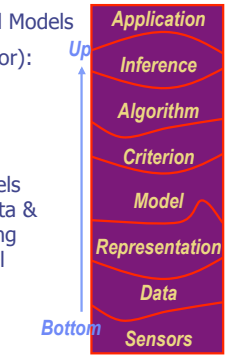
Real domains (vision, speech, behavior):
- no $E=MC^2$
- noisy, complex, nonlinear
- have many variables
- non-deterministic
- incomplete, approximate models

Need: statistical models driven by data & sensors, a.k.a Machine Learning

Bottom-Up: use data to form a model

Why? Complex data everywhere, audio, video, internet

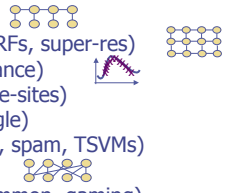
Intelligence = Learning = Prediction



Machine Learning Applications

- ML: Interdisciplinary (CS, Math, Stats, Physics, OR, Psych)
- Data-driven approach to AI
- Many domains are too hard to do manually

Speech Recognition (HMMs, ICA)
Computer Vision (face rec, digits, MRFs, super-res)
Time Series Prediction (weather, finance)
Genomics (micro-arrays, SVMs, splice-sites)
NLP and Parsing (HMMs, CRFs, Google)
Text and InfoRetrieval (docs, google, spam, TSVMs)
Medical (QMR-DT, informatics, ICA)
Behavior/Games (reinforcement, gammon, gaming)

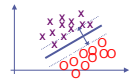


Course Details & Requirements

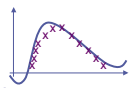
- Probability/Stats, Linear Algebra, Calculus, AI
- Mathematical & Data Driven approach to AI
- Lots of Equations!
- Required Text: Introduction to Graphical Models by M. Jordan & C. Bishop (Online)
  Pattern Recognition & Machine Learning by C. Bishop (Spring 2006 Edition)
- Reference Text: Pattern Classification (3rd Edition) by Duda, Hart and Stork
- Homework: Every 2-3 weeks
- Grading: homework, midterm, 2 quizzes & final examination
- Software Requirements: Matlab software & Acis account

Machine Learning Tasks

Classification y=sign(f(x))

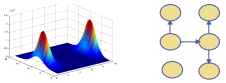


Regression y=f(x)

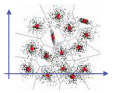


Supervised

Modeling p(x)



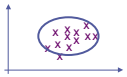
Clustering



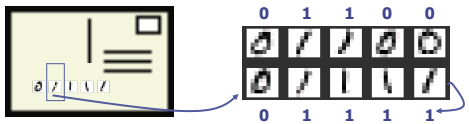
Unsupervised

Feature Selection

Detection p(x)<t



ML Example: Digit Recognition

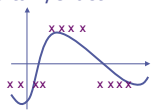


- Want to automate zipcode reading in post office
- Look at an image and say if it is a '1' or '0'
- 8x8 pixels of gray-level (0.0=dark, 0.5=gray, 1.0=white)
- Learn from above labeled training images
- Predict labels on testing images
- Binary Classification [0,1]
- What to do?

Ex: Two Approaches

In ML, we will consider two complementary approaches:

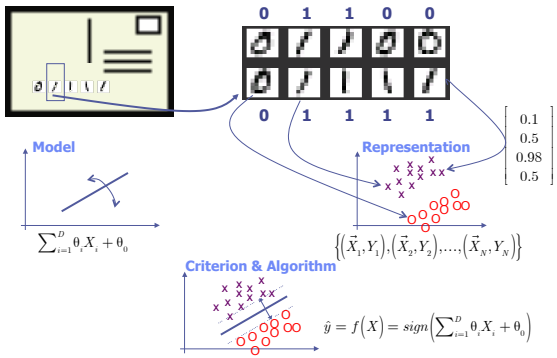
1) Deterministic:
   All variables/observables are treated as certain/exact
   Find/fit a function f(X) on an image X
   Output 0 or 1 depending on input
   Class label given by y=sign(f(X))/2 + 1/2



2) Probabilistic/Bayesian/Stochastic:
   Variables/observables are random (R.V.) and uncertain
   Probability image is a '0' digit: p(y=0|X) = 0.43
   Probability image is a '1' digit: p(y=1|X) = 0.57
   Output label with larger p(y=0|image) or p(y=1|image)

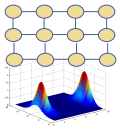
These are interconnected! Deterministic approaches can be generated from (more general) probabilistic approaches

Ex: 1) Deterministic Approach

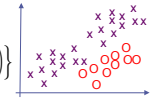


Ex: 2) Probabilistic Approach

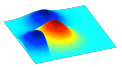
a) Provide Prior Model Parameters & Structure
e.g. nearby pixels are co-dependent



b) Obtain Data and Labels $\{(X_1,Y_1),\ldots,(X_T,Y_T)\}$



c) Learn a probability model with data p(all system variables) $p(X,Y)$



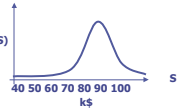
d) Use model for inference (classify/predict) $p(Y \mid X)$

Probability image is '0': p(y=0|X)
Probability image is '1': p(y=1|X)
Output: arg max$_i$ p(y=i|X)

Why Probabilistic Approach?

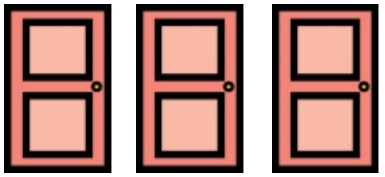
- Decision making often involves uncertainty
- Hidden variables, complexity, randomness in system
- Input data is noisy and uncertain
- Estimated model is noisy and uncertain
- Output data is uncertain (no single correct answer)

- Example: Predict your salary in the future
- Inputs: Field, Degree, University, City, IQ
- Output: $Amount
- There is uncertainty and hidden variables
- No one answer (I.e. $84K) is correct
- Answer = a distribution over salaries



Why Probabilistic? Monty Hall

- Behind one door is a prize (car? 1$?)
- Pick a door



Door A Door B Door C

Monty Hall Solution

Probabilistic Graphical Model Bayesian Network

Probabilistic Interpretation is Best

Bayesian Solution: Change your mind!

Assume we always start by picking A.

If prize behind A: Opens B/C → Change A to C/B → Lose

If prize behind B: Opens C → Change A to B → Win

If prize behind C: Opens B → Change A to C → Win

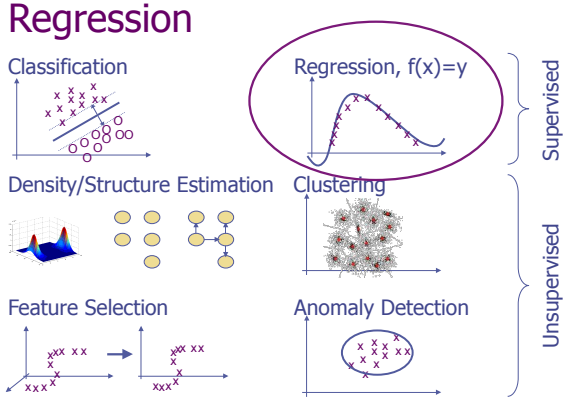
Probability of winning if change your mind = 66%
Probability of winning if stick to your guns = 33%

Machine Learning

Topic 2

- Regression
- Empirical Risk Minimization
- Least Squares
- Higher Order Polynomials
- Under-fitting / Over-fitting
- Cross-Validation

Regression



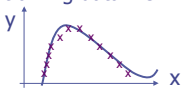
Tony Jebara, Columbia University

Function Approximation

- Start with training dataset

$$\mathcal{X} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \quad x \in \mathbb{R}^D = \begin{bmatrix} x(1) \\ x(2) \\ \ldots \\ x(D) \end{bmatrix} \quad y \in \mathbb{R}^1$$

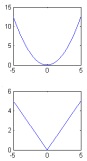
- Have N (input, output) pairs
- Find a function f(x) to predict y from x
  That fits the training data well



- Example: predict the price of house in dollars y using x = [#rooms; latitude; longitude; …]
- Need: a) Way to evaluate how good a fit we have
  b) Class of functions in which to search for f(x)

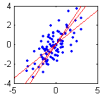
Empirical Risk Minimization

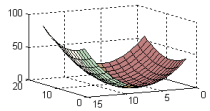
- Idea: minimize 'loss' on the training data set
- Empirical = use the training set to find the best fit
- Define a loss function of how good we fit a single point:
  $L(y, f(x))$
- Empirical Risk = the average loss over the dataset
  $R = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i))$
- Simplest loss: squared error from y value
  $L(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$
- Other possible loss: absolute error
  $L(y_i, f(x_i)) = |y_i - f(x_i)|$



Linear Function Classes

- Linear is simplest class of functions to search over:
  $f(x;\theta) = \theta^T x + \theta_0 = \sum_{d=1}^{D} \theta_d x(d) + \theta_0$
- Start with x being 1-dimensional (D=1):
  $f(x;\theta) = \theta_1 x + \theta_0$
- Plug in the above & minimize empirical risk over θ
  $R(\theta) = \frac{1}{2N}\sum_{i=1}^{N} (y_i - \theta_1 x_i - \theta_0)^2$





- Note: minimum occurs when R(θ) gets flat (not always!)
- Note: when R(θ) is flat, gradient $\nabla_\theta R = 0$

Min by Gradient=0

- Gradient=0 means the partial derivatives are all 0

$$\nabla_\theta R = \begin{bmatrix} \frac{\partial R}{\partial \theta_0} \\ \frac{\partial R}{\partial \theta_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- Take partials of empirical risk:

$$R(\theta) = \frac{1}{2N}\sum_{i=1}^{N} (y_i - \theta_1 x_i - \theta_0)^2$$

$$\frac{\partial R}{\partial \theta_0} = \frac{1}{N}\sum_{i=1}^{N} (y_i - \theta_1 x_i - \theta_0)(-1) = 0$$

$$\frac{\partial R}{\partial \theta_1} = \frac{1}{N}\sum_{i=1}^{N} (y_i - \theta_1 x_i - \theta_0)(-x_i) = 0$$

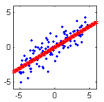
$$\theta_0 = \frac{1}{N}\sum y_i - \theta_1 \frac{1}{N}\sum x_i$$

$$\theta_1 \sum x_i^2 = \sum y_i x_i - \theta_0 \sum x_i$$

$$\theta_1 = \frac{\sum y_i x_i - \frac{1}{N}\sum y_i \sum x_i}{\sum x_i^2 - \frac{1}{N}\sum x_i \sum x_i}$$

Properties of the Solution

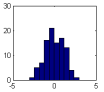
- Setting θ* as before gives least squared error
- Define error on each data point as:
  $e_i = y_i - \theta^*_1 x_i - \theta^*_0$



- Note property #1:

$$\frac{\partial R}{\partial \theta_0} = \frac{1}{N}\sum_{i=1}^{N} (y_i - \theta_1 x_i - \theta_0) = 0$$

…average error is zero $\frac{1}{N}\sum e_i = 0$

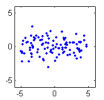


- Note property #2:

$$\frac{\partial R}{\partial \theta_1} = \frac{1}{N}\sum_{i=1}^{N} (y_i - \theta_1 x_i - \theta_0) x_i = 0$$

…error not correlated with data

$$\frac{1}{N}\sum e_i x_i = \frac{1}{N} e^T x = 0$$



Multi-Dimensional Regression

- More elegant/general to do $\nabla_\theta R = 0$ with linear algebra
- Rewrite empirical risk in vector-matrix notation:

$$R(\theta) = \frac{1}{2N}\sum_{i=1}^{N} (y_i - \theta_1 x_i - \theta_0)^2$$

$$= \frac{1}{2N}\sum_{i=1}^{N} \left(y_i - \begin{bmatrix} 1 & x_i \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \right)^2$$

$$= \frac{1}{2N} \left\| \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \right\|^2$$

$$= \frac{1}{2N} \|\mathbf{y} - \mathbf{X}\theta\|^2$$

Can add more dimensions by adding columns to X matrix and rows to θ vector

Multi-Dimensional Regression

- More elegant/general to do $\nabla_\theta R = 0$ with linear algebra
- Rewrite empirical risk in vector-matrix notation:

$$R(\theta) = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \theta_1 x_i - \theta_0\right)^2$$

$$= \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \begin{bmatrix} 1 & x_i \end{bmatrix}\begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}\right)^2$$

$$= \frac{1}{2N}\left\| \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} 1 & x_1(1) & \dots & x_1(D) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N(1) & \dots & x_N(D) \end{bmatrix}\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_D \end{bmatrix}\right\|^2$$

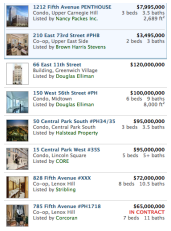
Can add more dimensions by adding columns to X matrix and rows to θ vector

$$= \frac{1}{2N}\left\|\mathbf{y} - \mathbf{X}\theta\right\|^2$$

Multi-Dimensional Regression

- More realistic dataset: many measurements
- Have N apartments each with D measurements
- Each row of X is [#rooms; latitude; longitude,...]

$$\mathbf{X} = \begin{bmatrix} 1 & x_1(1) & \dots & x_1(D) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N(1) & \dots & x_N(D) \end{bmatrix}$$



Multi-Dimensional Regression

- Solving gradient=0

$$\nabla_\theta R = 0$$

$$\nabla_\theta\left(\frac{1}{2N}\left\|\mathbf{y} - \mathbf{X}\theta\right\|^2\right) = 0$$

$$\frac{1}{2N}\nabla_\theta\left(\left(\mathbf{y} - \mathbf{X}\theta\right)^T\left(\mathbf{y} - \mathbf{X}\theta\right)\right) = 0$$

$$\frac{1}{2N}\nabla_\theta\left(\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\theta + \theta^T\mathbf{X}^T\mathbf{X}\theta\right) = 0$$

$$\frac{1}{2N}\left(-2\mathbf{y}^T\mathbf{X} + 2\theta^T\mathbf{X}^T\mathbf{X}\right) = 0$$

$$\mathbf{X}^T\mathbf{X}\theta = \mathbf{X}^T\mathbf{y}$$

$$\theta^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

$$\frac{\partial \vec{u}^T\vec{\theta}}{\partial\vec{\theta}} = \vec{u}^T$$

$$\frac{\partial \vec{\theta}^T\vec{\theta}}{\partial\vec{\theta}} = 2\vec{\theta}^T$$

$$\frac{\partial \vec{\theta}^T A\vec{\theta}}{\partial\vec{\theta}} = \vec{\theta}^T\left(A + A^T\right)$$

- In Matlab: "t=pinv(X)*y" or "t=X\y" or "t=inv(X'*X)*X'*y"

Multi-Dimensional Regression

- Solving gradient=0

$$\mathbf{X}^T\mathbf{X}\theta = \mathbf{X}^T\mathbf{y}$$

$$\theta^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

- In Matlab: "t=pinv(X)*y" or "t=X\y" or "t=inv(X'*X)*X'*y"
- If the matrix X is skinny, the solution is probably unique
- If X is fat (more dimensions than points) we get multiple solutions for theta which give zero error.

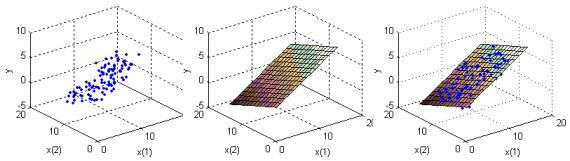
- The pseudeoinverse (pinv(X)) returns the theta with zero error and which has the smallest norm.

$$\min_\theta \left\|\theta\right\|^2 \; such\ that\ \mathbf{X}\theta = \mathbf{y}$$

2D Linear Regression

- Once best θ* is found, we can plug it into the function:

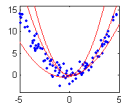
$$f\left(x;\theta^*\right) = \theta_2^* x(2) + \theta_1^* x(1) + \theta_0^*$$



- What would a fat X look like?

Polynomial Function Classes

- Back to 1-dim x (D=1) BUT Nonlinear
- Polynomial: $f\left(x;\theta\right) = \sum_{p=1}^{P}\theta_p x^p + \theta_0$



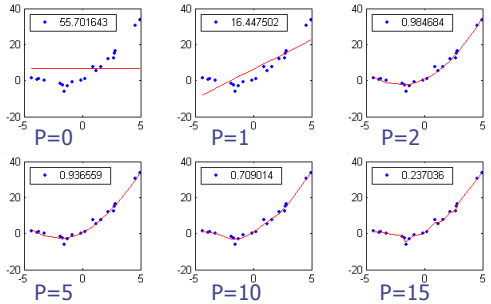
- Writing Risk:

$$R(\theta) = \frac{1}{2N}\left\| \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} 1 & x_1^1 & \dots & x_1^P \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N^1 & \dots & x_N^P \end{bmatrix}\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_P \end{bmatrix}\right\|^2$$

- Order-P polynomial regression fitting for 1D variable is same as P-dimensional linear regression!
- Construct a multidim x-vector from x scalar $\mathbf{x}_i = \begin{bmatrix} x_i^0 & x_i^1 & x_i^2 & x_i^3 \end{bmatrix}^T$
- More generally any $\mathbf{x}_i = \begin{bmatrix} \phi_0(x_i) & \phi_1(x_i) & \phi_2(x_i) & \phi_3(x_i) \end{bmatrix}^T$

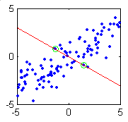
Underfitting/Overfitting

- Try varying P. Higher P fits a more complex function class
- Observe R(θ*) drops with bigger P



Evaluating The Regression

- Unfair to use empirical to find best order P
- High P (vs. N) can overfit, even linear case!
- min R(θ*) not on training but on future data
- Want model to *Generalize* to future data



True loss: $R_{true}(\theta) = \int P(x,y)L\left(y, f\left(x;\theta\right)\right)dx\,dy$

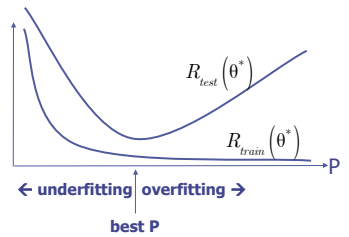
- One approach: split data into training / testing portion

$$\left\{\left(x_1,y_1\right),\dots,\left(x_N,y_N\right)\right\} \qquad \left\{\left(x_{N+1},y_{N+1}\right),\dots,\left(x_{N+M},y_{N+M}\right)\right\}$$

- Estimate θ* with training loss: $R_{train}(\theta) = \frac{1}{N}\sum_{i=1}^{N}L\left(y_i, f\left(x_i;\theta\right)\right)$
- Evaluate P with testing loss: $R_{test}(\theta) = \frac{1}{M}\sum_{i=N+1}^{N+M}L\left(y_i, f\left(x_i;\theta\right)\right)$

Crossvalidation

- Try fitting with different polynomial order P
- Select P which gives lowest $R_{test}(\theta^*)$



- Think of P as a measure of the complexity of the model
- Higher order polynomials are more flexible and complex

Machine Learning

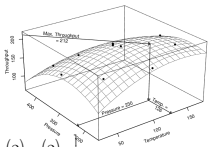
Topic 3

- Additive Models and Linear Regression
- Sinusoids and Radial Basis Functions
- Classification
- Logistic Regression
- Gradient Descent

Polynomial Basis Functions

- To fit a P'th order polynomial function to multivariate data: concatenate columns of all monomials up to power P
- E.g. 2 dimensional data and 2nd order polynomial (quadratic)

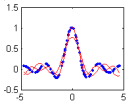
$$\begin{bmatrix} x_1(1) & x_1(2) \\ \vdots & \vdots \\ x_i(1) & x_i(2) \\ \vdots & \vdots \\ x_N(1) & x_N(2) \end{bmatrix} \rightarrow \begin{bmatrix} 1 & x_1(1) & x_1(2) & x_1(1)x_1(1) & x_1(1)x_1(2) & x_1(2)x_1(2) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_i(1) & x_i(2) & x_i(1)x_i(1) & x_i(1)x_i(2) & x_i(2)x_i(2) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_N(1) & x_N(2) & x_N(1)x_N(1) & x_N(1)x_N(2) & x_N(2)x_N(2) \end{bmatrix}$$



Sinusoidal Basis Functions

- More generally, we don't just have to deal with polynomials, use any set of basis fn's:

$$f(x;\theta) = \sum_{p=1}^{P} \theta_p \phi_p(x) + \theta_0$$

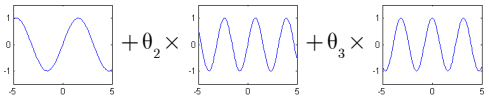


- These are generally called Additive Models
- Regression adds linear combinations of the basis fn's
- For example: Fourier (sinusoidal) basis

$$\phi_{2k}(x_i) = \sin(kx_i) \qquad \phi_{2k+1}(x_i) = \cos(kx_i)$$

- Note, don't have to be a basis per se, usually subset

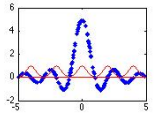
$$\theta_1 \times \quad + \theta_2 \times \quad + \theta_3 \times$$



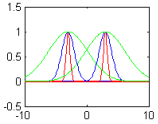
Radial Basis Functions

- Can act as prototypes of the data itself

$$f(\mathbf{x};\theta) = \sum_{k=1}^{N} \theta_k \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}-\mathbf{x}_k\right\|^2\right)$$

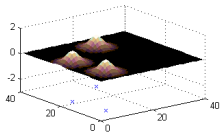


- Parameter σ = standard deviation
  σ^2 = covariance



controls how wide bumps are
what happens if too big/small?

- Also works in multi-dimensions



- Called RBF for short

Radial Basis Functions

- Each training point leads to a bump function

$$f(\mathbf{x};\theta) = \sum_{k=1}^{N} \theta_k \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}-\mathbf{x}_k\right\|^2\right)$$

- Reuse solution from linear regression: $\theta^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$
- Can view the data instead as X, a big matrix of size N x N

$$\mathbf{X} = \begin{bmatrix} \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_1-\mathbf{x}_1\right\|^2\right) & \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_1-\mathbf{x}_2\right\|^2\right) & \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_1-\mathbf{x}_3\right\|^2\right) \\ \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_2-\mathbf{x}_1\right\|^2\right) & \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_2-\mathbf{x}_2\right\|^2\right) & \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_2-\mathbf{x}_3\right\|^2\right) \\ \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_3-\mathbf{x}_1\right\|^2\right) & \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_3-\mathbf{x}_2\right\|^2\right) & \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}_3-\mathbf{x}_3\right\|^2\right) \end{bmatrix}$$

- For RBFs, X is square and symmetric, so solution is just

$$\nabla_\theta R = 0 \rightarrow \mathbf{X}^T\mathbf{X}\theta = \mathbf{X}^T\mathbf{y} \rightarrow \mathbf{X}\theta = \mathbf{y} \rightarrow \theta^* = \mathbf{X}^{-1}\mathbf{y}$$

Evaluating Our Learned Function

- We minimized empirical risk to get θ^*
- How well does $f(x;\theta^*)$ perform on future data?
- It should *Generalize* and have low True Risk:

$$R_{true}(\theta) = \int P(x,y)\,L\left(y, f(x;\theta)\right)dx\,dy$$

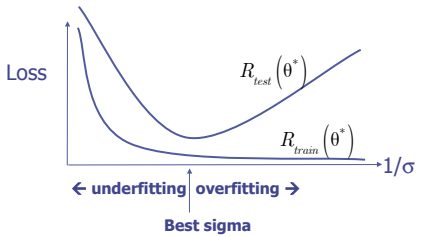
- Can't compute true risk, instead use Testing Empirical Risk
- We randomly split data into training and testing portions

$$\left\{(x_1,y_1),\ldots,(x_N,y_N)\right\} \qquad \left\{(x_{N+1},y_{N+1}),\ldots,(x_{N+M},y_{N+M})\right\}$$

- Find θ^* with training data: $R_{train}(\theta) = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, f(x_i;\theta)\right)$
- Evaluate it with testing data: $R_{test}(\theta) = \frac{1}{M}\sum_{i=N+1}^{N+M} L\left(y_i, f(x_i;\theta)\right)$

Crossvalidation

- Try fitting with different sigma radial basis function widths
- Select sigma which gives lowest $R_{test}(\theta^*)$



- Think of sigma as a measure of the simplicity of the model
- Thinner RBFs are more flexible and complex

Regularized Risk Minimization

- Empirical Risk Minimization gave overfitting & underfitting
- We want to add a penalty for using too many theta values
- This gives us the Regularized Risk

$$R_{regularized}(\theta) = R_{empirical}(\theta) + Penalty(\theta)$$
$$= \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, f(x_i;\theta)\right) + \frac{\lambda}{2N}\left\|\theta\right\|^2$$

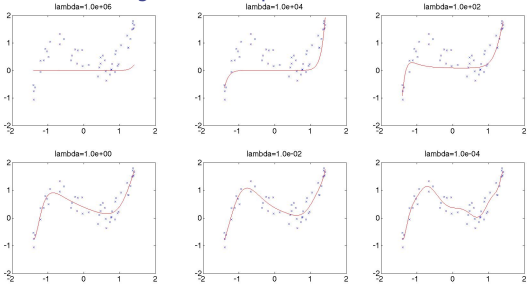
- Solution for Regularized Risk with Least Squares Loss:

$$\nabla_\theta R_{regularized} = 0 \quad \Rightarrow \quad \nabla_\theta\left\{\frac{1}{2N}\left\|\mathbf{y}-\mathbf{X}\theta\right\|^2 + \frac{\lambda}{2N}\left\|\theta\right\|^2\right\} = 0$$

$$\theta^* = \left(\mathbf{X}^T\mathbf{X} + \lambda I\right)^{-1}\mathbf{X}^T\mathbf{y}$$

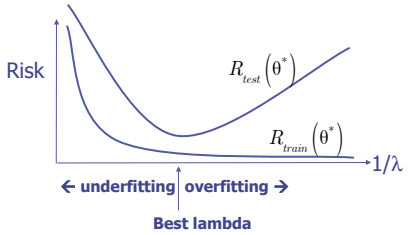
Regularized Risk Minimization

- Have D=16 features (or P=15 throughout)
- Try minimizing $R_{regularized}(\theta)$ to get θ^* with different λ
- Note that λ=0 give back Empirical Risk Minimization



Crossvalidation

- Try fitting with different lambda regularization levels
- Select lambda which gives lowest $R_{test}(\theta^*)$



- Lambda measures simplicity of the model
- Models with low lambda are more flexible

From Regression To Classification

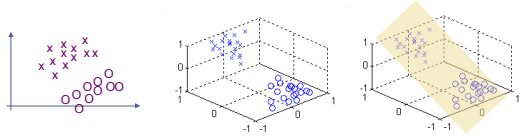
- Classification is another important learning problem

Regression $\mathcal{X} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$ $\mathbf{x} \in \mathbb{R}^D$ $y \in \mathbb{R}^1$

Classification $\mathcal{X} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$ $\mathbf{x} \in \mathbb{R}^D$ $y \in \{0,1\}$

- E.g. Given x = [tumor size, tumor density]
  Predict y in {benign,malignant}

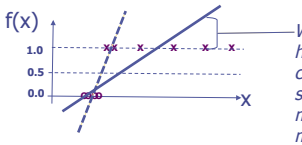
- Should we solve this as a least squares regression problem?



Classification vs. Regression

a) Classification needs binary answers like {0,1}
b) Least squares is an unfair measure of risk here
   e.g. Why penalize a correct but large positive y answer?
   e.g. Why penalize a correct but large negative y answer?

- Example: not good to use regression output for a decision
  f(x)>0.5 → Class 1 f(x)<0.5 → Class 0
  if f(x)=-3.8 & correct class=0, squared error penalizes it…



We pay a hefty squared error loss here even if we got the correct classification result. The thick solid line model makes two mistakes while the dashed model is perfect

Classification vs. Regression

We will consider the following four steps to improve from naïve regression to get better classification learning:

1) Fix functions f(x) to give binary output (logistic neuron)
2) Fix our definition of the Risk we will minimize so that we get good classification accuracy (logistic loss)

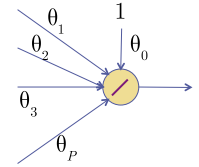
…and later on…

3) Make an even better fix on f(x) to binarize (perceptron)
4) Make an even better risk (perceptron loss)

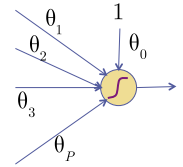
Logistic Neuron (McCullough-Pitts)

- To output binary, use squashing function g().

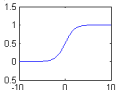
$$f(\mathbf{x};\theta) = \theta^T\mathbf{x} \qquad f(\mathbf{x};\theta) = g(\theta^T\mathbf{x})$$
$$g(z) = (1+\exp(-z))^{-1}$$



Linear neuron



Logistic Neuron



- This squashing is called sigmoid or logistic function

Logistic Regression

- Given a classification problem with binary outputs

$$\mathcal{X} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\} \quad \mathbf{x} \in \mathbb{R}^D \quad y \in \{0,1\}$$

- Use this function and output 1 if f(x)>0.5 and 0 otherwise

$$f(\mathbf{x};\theta) = (1+\exp(-\theta^T\mathbf{x}))^{-1}$$

Short hand for Linear Functions

- What happened to adding the intercept?

$$f(\mathbf{x};\theta) = \theta^T\mathbf{x} + \theta_0$$

$$= \begin{bmatrix} \theta(1) \\ \theta(2) \\ \vdots \\ \theta(D) \end{bmatrix}^T \begin{bmatrix} \mathbf{x}(1) \\ \mathbf{x}(2) \\ \vdots \\ \mathbf{x}(D) \end{bmatrix} + \theta_0 = \begin{bmatrix} \theta_0 \\ \theta(1) \\ \theta(2) \\ \vdots \\ \theta(D) \end{bmatrix}^T \begin{bmatrix} 1 \\ \mathbf{x}(1) \\ \mathbf{x}(2) \\ \vdots \\ \mathbf{x}(D) \end{bmatrix} = \vec{\theta}^T\vec{\mathbf{x}}$$

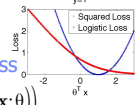
Logistic Regression

- Given a classification problem with binary outputs

$$\mathcal{X} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\} \quad \mathbf{x} \in \mathbb{R}^D \quad y \in \{0,1\}$$

- Fix#1: use f(x) below, output 1 if f(x)>0.5 and 0 otherwise

$$f(\mathbf{x};\theta) = (1+\exp(-\theta^T\mathbf{x}))^{-1}$$



- Fix#2: instead of squared loss, use Logistic Loss

$$L_{log}(y, f(\mathbf{x};\theta)) = (y_i - 1)\log(1 - f(\mathbf{x};\theta)) - y_i \log(f(\mathbf{x};\theta))$$

- This method is called Logistic Regression.
- But Empirical Risk Minimization has no closed-form sol'n:

$$R_{emp}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - 1)\log(1 - f(\mathbf{x}_i;\theta)) - y_i \log(f(\mathbf{x}_i;\theta))$$

Logistic Regression

- With logistic squashing function, minimizing R(θ) is harder

$$R_{emp}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - 1)\log\left(1 - f(\mathbf{x}_i;\theta)\right) - y_i \log\left(f(\mathbf{x}_i;\theta)\right)$$

$$\nabla_\theta R = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1-y_i}{1-f(\mathbf{x}_i;\theta)} - \frac{y_i}{f(\mathbf{x}_i;\theta)}\right) f'(\mathbf{x}_i;\theta) = 0 \quad ???$$

- Can't minimize risk and find best theta analytically!
- Let's try finding best theta numerically.
- Use the following to compute gradient

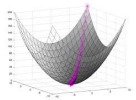
$$f(\mathbf{x};\theta) = \left(1 + \exp\left(-\theta^T\mathbf{x}\right)\right)^{-1} = g\left(\theta^T\mathbf{x}\right)$$

- Here, g() is the logistic squashing function

$$g(z) = \left(1 + \exp(-z)\right)^{-1} \qquad g'(z) = g(z)\left(1 - g(z)\right)$$

Gradient Descent

- Useful when we can't get minimum solution in closed form
- Gradient points in direction of fastest increase
- Take step in the opposite direction!



- Gradient Descent Algorithm

choose scalar step size η, & tolerance ε
initialize θ^0 = small random vector

$$\theta^1 = \theta^0 - \eta \nabla_\theta R_{emp}\Big|_{\theta^0}, \quad t = 1$$

while $\left\|\theta^t - \theta^{t-1}\right\| \geq \epsilon$ *{*

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta R_{emp}\Big|_{\theta^t}, \quad t = t + 1$$

}

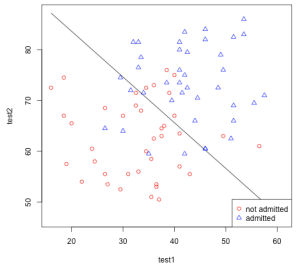
- For appropriate *η*, this will converge to local minimum

Logistic Regression

- Logistic regression gives better classification performance
- Its empirical risk is

$$R_{emp}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - 1)\log\left(1 - f(\mathbf{x}_i;\theta)\right) - y_i \log\left(f(\mathbf{x}_i;\theta)\right)$$

- This R(θ) is convex so gradient descent always converges to the same solution



- Make predictions using

$$f(\mathbf{x};\theta) = \left(1 + \exp\left(-\theta^T\mathbf{x}\right)\right)^{-1}$$

- Output 1 if f > 0.5
- Output 0 otherwise

Machine Learning

Topic 4

- Perceptron, Online & Stochastic Gradient Descent
- Convergence Guarantee
- Perceptron vs. Linear Regression
- Multi-Layer Neural Networks
- Back-Propagation
- Demo: LeNet
- Deep Learning

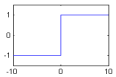
Perceptron (another Neuron)

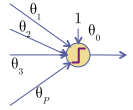
- Classification scenario once again but consider +1, -1 labels

$$\mathcal{X} = \left\{\left(x_1, y_1\right), \left(x_2, y_2\right), \ldots, \left(x_N, y_N\right)\right\} \quad x \in \mathbb{R}^D \quad y \in \left\{-1, 1\right\}$$

- A better choice for a classification squashing function is

$$g\left(z\right) = \begin{cases} -1 \text{ when } z < 0 \\ +1 \text{ when } z \geq 0 \end{cases}$$





- And a better choice is classification loss

$$L\left(y, f\left(\mathbf{x};\theta\right)\right) = \text{step}\left(-yf\left(\mathbf{x};\theta\right)\right)$$

- Actually with above g(z) any loss is like classification loss

$$R\left(\theta\right) = \frac{1}{4N}\sum_{i=1}^{N}\left(y - g\left(\theta^T x_i\right)\right)^2 \equiv \frac{1}{N}\sum_{i=1}^{N}\text{step}\left(-y_i\theta^T x_i\right)$$

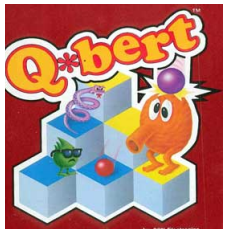
- What does this R(θ) function look like?

Perceptron & Classification Loss

- Classification loss for the Risk leads to hard minimization
- What does this R(θ) function look like?

$$R\left(\theta\right) = \frac{1}{N}\sum_{i=1}^{N}\text{step}\left(-y_i\theta^T x_i\right)$$

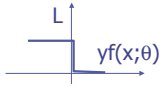
- Qbert-like, can't do gradient descent since the gradient is zero except at edges when a label flips



Perceptron & Perceptron Loss

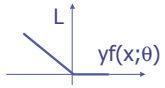
- Instead of Classification Loss

$$R\left(\theta\right) = \frac{1}{N}\sum_{i=1}^{N}\text{step}\left(-y_i\theta^T x_i\right)$$



- Consider Perceptron Loss:

$$R^{per}\left(\theta\right) = -\frac{1}{N}\sum_{i\in misclassified} y_i\left(\theta^T x_i\right)$$



- Instead of staircase-shaped R get smooth piece-wise linear R

- Get reasonable gradients for gradient descent

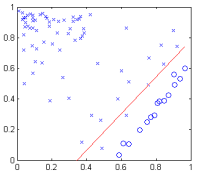
$$\nabla_\theta R^{per}\left(\theta\right) = -\frac{1}{N}\sum_{i\in misclassified} y_i\mathbf{x}_i$$

$$\theta^{t+1} = \theta^t - \eta\nabla_\theta R^{per}\Big|_{\theta^t} = \theta^t + \eta\frac{1}{N}\sum_{i\in misclassified} y_i\mathbf{x}_i$$

Perceptron vs. Linear Regression

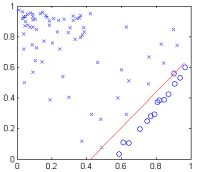
- Linear regression gets close but doesn't do perfectly

classification error = 2
squared error = 0.139



- Perceptron gets zero error

classification error = 0
perceptron err = 0



Stochastic Gradient Descent

- Gradient Descent vs. Stochastic Gradient Descent

- Instead of computing the average gradient for all points and then taking a step

$$\nabla_\theta R^{per}\left(\theta\right) = -\frac{1}{N}\sum_{i\in misclassified} y_i\mathbf{x}_i$$

- Update the gradient for each mis-classified point by itself

$$\nabla_\theta R^{per}\left(\theta\right) = -y_i\mathbf{x}_i \qquad \text{if i mis-classified}$$

- Also, set η to 1 without loss of generality

$$\theta^{t+1} = \theta^t - \eta\nabla_\theta R^{per}\Big|_{\theta^t} = \theta^t + y_i\mathbf{x}_i \qquad \text{if i mis-classified}$$

Online Perceptron

- Apply stochastic gradient descent to a perceptron
- Get the "online perceptron" algorithm:

$$\begin{array}{l} initialize\, t = 0 \;and\; \theta^0 = \vec{0} \\ while\, not\, converged\, \{ \\ \quad pick\, i \in \left\{1, \ldots, N\right\} \\ \quad if\left(y_i x_i^T\theta^t \leq 0\right) \quad \{\; \theta^{t+1} = \theta^t + y_i x_i \\ \qquad\qquad\qquad\qquad t = t+1 \qquad \} \} \end{array}$$

- Either pick i randomly or use a "for i=1 to N" loop
- If the algorithm stops, we have a theta that separates data
- The total number of mistakes we made along the way is t

Online Perceptron Theorem

Theorem: the online perceptron algorithm converges to zero error in finite t if we assume

1) all data inside a sphere of radius r: $\left\|\mathbf{x}_i\right\| \leq r \;\; \forall i$
2) data is separable with margin γ: $y_i\left(\theta^*\right)^T\mathbf{x}_i \geq \gamma \;\; \forall i$

Proof:

- Part 1) Look at inner product of current θ^t with θ^*
  assume we just updated a mistake on point i:

$$\left(\theta^*\right)^T\theta^t = \left(\theta^*\right)^T\theta^{t-1} + y_i\left(\theta^*\right)^T\mathbf{x}_i \geq \left(\theta^*\right)^T\theta^{t-1} + \gamma$$

  after applying t such updates, we must get:

$$\left(\theta^*\right)^T\theta^t = \left(\theta^*\right)^T\theta^t \geq t\gamma$$

Online Perceptron Proof

•Part 1) $\left(\theta^*\right)^T \theta^t = \left(\theta^*\right)^T \theta^t \geq t\gamma$

•Part 2) $\left\|\theta^t\right\|^2 = \left\|\theta^{t-1} + y_i \mathbf{x}_i\right\|^2 = \left\|\theta^{t-1}\right\|^2 + 2y_i\left(\theta^{t-1}\right)^T \mathbf{x}_i + \left\|\mathbf{x}_i\right\|^2$

$\leq \left\|\theta^{t-1}\right\|^2 + \left\|\mathbf{x}_i\right\|^2$ since only update mistakes middle term is negative

$\leq \left\|\theta^{t-1}\right\|^2 + r^2$

$\leq tr^2$

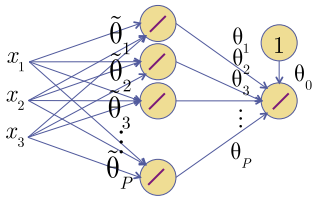
•Part 3) Angle between optimal & current solution

$$\cos\left(\theta^*, \theta^t\right) = \frac{\left(\theta^*\right)^T \theta^t}{\left\|\theta^t\right\|\left\|\theta^*\right\|} \geq \frac{t\gamma}{\left\|\theta^t\right\|\left\|\theta^*\right\|} \geq \frac{t\gamma}{\sqrt{tr^2}\left\|\theta^*\right\|}$$ apply part 1 then part 2

•Since $\cos \leq 1 \Rightarrow \frac{t\gamma}{\sqrt{tr^2}\left\|\theta^*\right\|} \leq 1 \Rightarrow t \leq \frac{r^2}{\gamma^2}\left\|\theta^*\right\|^2$ ...so t is finite!

Multi-Layer Neural Networks

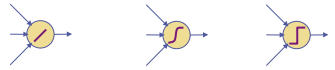
•What if we consider cascading multiple layers of network?
•Each output layer is input to the next layer
•Each layer has its own weights parameters
•Eg: each layer has linear nodes (not perceptron/logistic)



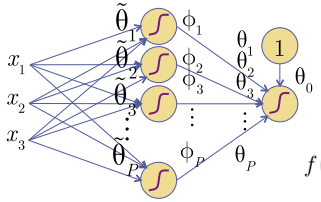
•Above Neural Net has 2 layers. What does this give?

Multi-Layer Neural Networks

•Need to introduce non-linearities between layers
•Avoids previous redundant linear layer problem



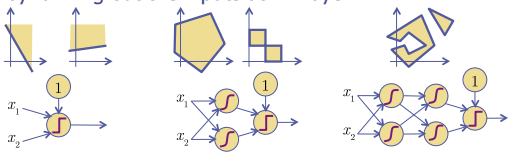
•Neural network can adjust the basis functions themselves...



$$f\left(\mathbf{x}\right) = g\left(\sum_{i=1}^{P} \theta_i g\left(\tilde{\theta}_i^T \mathbf{x}\right)\right)$$

Multi-Layer Neural Networks

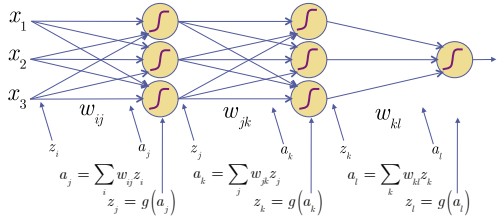
•Multi-Layer Network can handle more complex decisions
•1-layer: is linear, can't handle XOR
•Each layer adds more flexibility (but more parameters!)
•Each node splits its input space with linear hyperplane
•2-layer: if last layer is AND operation, get convex hull
•2-layer: can do almost anything multi-layer can by fanning out the inputs at 2nd layer



•Note: Without loss of generality, we can omit the 1 and θ_0

Back-Propagation

•Gradient descent on squared loss is done layer by layer
•Layers: input, hidden, output. Parameters: $\theta = \left\{w_{ij}, w_{jk}, w_{kl}\right\}$

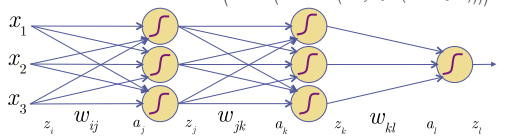


•Each input x_n for n=1..N generates its own a's and z's
•Back-Propagation: Splits layer into its inputs & outputs
•Get gradient on output...back-track chain rule until input

Back-Propagation

•Cost function: $R\left(\theta\right) = \frac{1}{N}\sum_n \frac{1}{2}\left(y^n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_i^n\right)\right)\right)\right)^2$

$= \frac{1}{N}\sum_n \frac{1}{2}\left(y^n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_i^n\right)\right)\right)\right)^2$



•First compute output layer derivative:

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L^n}{\partial a_l^n}\right]\left[\frac{\partial a_l^n}{\partial w_{kl}}\right]$$ **Chain Rule**

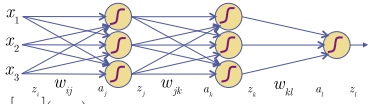
$define\ L^n := \frac{1}{2}\left(y^n - f\left(x^n\right)\right)^2$

Define as δ

$$= \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}\left(y^n - g\left(a_l^n\right)\right)^2}{\partial a_i^n}\right]\left[\frac{\partial a_l^n}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-\left(y^n - z_l^n\right) g'\left(a_l^n\right)\right]\left(z_k^n\right) = \frac{1}{N}\sum_n \delta_l^n z_k^n$$

Back-Propagation

•Cost function: $R\left(\theta\right) = \frac{1}{N}\sum_n \frac{1}{2}\left(y^n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_i^n\right)\right)\right)\right)^2$



$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L^n}{\partial a_l^n}\right]\left[\frac{\partial a_l^n}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-\left(y^n - z_l^n\right) g'\left(a_l^n\right)\right]\left(z_k^n\right) = \frac{1}{N}\sum_n \delta_l^n z_k^n$$

•Next, hidden layer derivative:

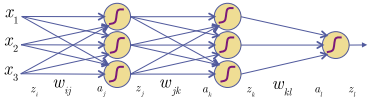
$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\frac{\partial L^n}{\partial a_k^n}\right]\left[\frac{\partial a_k^n}{\partial w_{jk}}\right] = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L^n}{\partial a_l^n}\frac{\partial a_l^n}{\partial a_k^n}\right]\left[\frac{\partial a_k^n}{\partial w_{jk}}\right]$$ **Multivariate Chain Rule**

$$= \frac{1}{N}\sum_n \left[\sum_l \delta_l^n \frac{\partial a_l^n}{\partial a_k^n}\right]\left(z_j^n\right) = \frac{1}{N}\sum_n \left[\sum_l \delta_l^n w_{kl} g'\left(a_k^n\right)\right]\left(z_j^n\right) = \frac{1}{N}\sum_n \delta_k^n z_j^n$$

Recall $a_l = \sum_k w_{kl} g\left(a_k\right)$ **Define as δ**

Back-Propagation

•Cost function: $R\left(\theta\right) = \frac{1}{N}\sum_n \frac{1}{2}\left(y^n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_i^n\right)\right)\right)\right)^2$



$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L^n}{\partial a_l^n}\right]\left[\frac{\partial a_l^n}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-\left(y^n - z_l^n\right) g'\left(a_l^n\right)\right]\left(z_k^n\right) = \frac{1}{N}\sum_n \delta_l^n z_k^n$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\frac{\partial L^n}{\partial a_k^n}\right]\left[\frac{\partial a_k^n}{\partial w_{jk}}\right] = \frac{1}{N}\sum_n \left[\sum_l \delta_l^n w_{kl} g'\left(a_k^n\right)\right]\left(z_j^n\right) = \frac{1}{N}\sum_n \delta_k^n z_j^n$$

•Any previous (input) layer derivative: repeat the formula!

$$\frac{\partial R}{\partial w_{ij}} = \frac{1}{N}\sum_n \left[\frac{\partial L^n}{\partial a_j^n}\right]\left[\frac{\partial a_j^n}{\partial w_{ij}}\right] = \frac{1}{N}\sum_n \left[\sum_k \frac{\partial L^n}{\partial a_k^n}\frac{\partial a_k^n}{\partial a_j^n}\right]\left[\frac{\partial a_j^n}{\partial w_{ij}}\right] = \frac{1}{N}\sum_n \left[\sum_k \delta_k^n w_{jk} g'\left(a_j^n\right)\right]\left(z_i^n\right) = \frac{1}{N}\sum_n \delta_j^n z_i^n$$

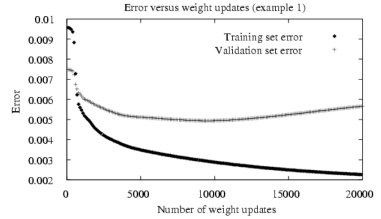
•What is this last z?

Back-Propagation

•Again, take small step in direction opposite to gradient

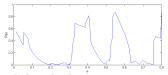
$$w_{ij}^{t+1} = w_{ij}^t - \eta\frac{\partial R}{\partial w_{ij}} \qquad w_{jk}^{t+1} = w_{jk}^t - \eta\frac{\partial R}{\partial w_{jk}} \qquad w_{kl}^{t+1} = w_{kl}^t - \eta\frac{\partial R}{\partial w_{kl}}$$

•Early stop before when error bottoms out on validation set



Neural Networks Demo

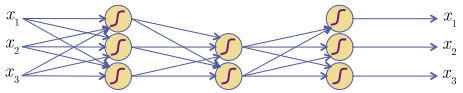
- Again, take small step in direction opposite to gradient
- Digits Demo: LeNet… http://yann.lecun.com
- Problems with back-prop
  is that MLP over-fits…



- Other problems: hard to interpret, black-box
- What are the hidden inner layers doing?

Auto-Encoders

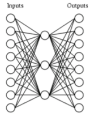
- Make the neural net reconstruct the input vector



- Set the target **y** vector to be the **x** vector again
- But, it gets narrow in the middle!
- So, there is some information "compression"
- This leads to better generalization

- This is unsupervised learning since we only use the **x** data

Auto-Encoders



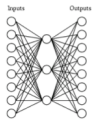
A target function:

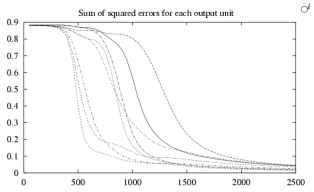
Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

Can this be learned??

Auto-Encoders

A network:



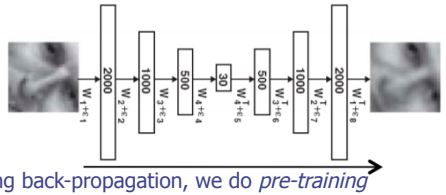


Learned hidden layer representation:

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Deep Learning

- We can stack several independently trained auto-encoders

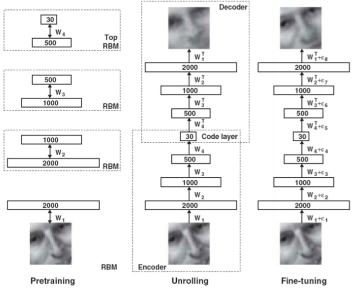


- Using back-propagation, we do *pre-training*
- Train Net 1 to go from 2000 inputs to 1000 to 2000 inputs
- Train Net 2 to go from 1000 hidden values to 500 to 1000
- Train Net 3 to go from 500 hidden to 30 to 500

- Then, do *unrolling* - link up the learned networks as above

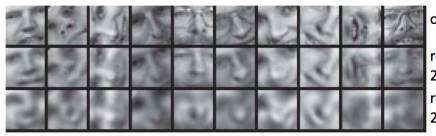
Deep Learning

- Then do *fine-tuning* of the overall neural network by running back-propagation on the whole thing to reconstruct **x** from **x**



Deep Learning

- Does good reconstruction!
- Beats PCA on images of unaligned faces.
- PCA is better when face images are aligned…
- We will cover PCA in a few lectures…



original image

reconstructed from
2000-1000-500-30 DBN

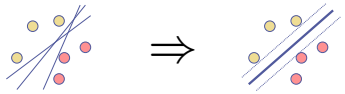
reconstructed from
2000-300, linear PCA

Minimum Training Error?

- Is minimizing Empricial Risk the right thing?
- Are Perceptrons and Neural Networks
  giving the best classifier?

- We are getting: minimum training error
  not minimum testing error

- Perceptrons are giving a bunch of solutions:



… a solution with *guarantees* → SVMs

Machine Learning

Topic 5

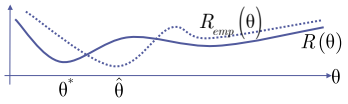
- Generalization Guarantees
- VC-Dimension
- Nearest Neighbor Classification (infinite VC dimension)
- Structural Risk Minimization
- Support Vector Machines

Empirical Risk Minimization

- Example: non-pdf linear classifiers $f(x;\theta) = sign(\theta^T x + \theta_0) \in \{-1,1\}$
- Recall ERM: $R_{emp}(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i;\theta)) \in [0,1]$
- Have loss function: quadratic: $L(y,x,\theta) = \frac{1}{2}(y - f(x;\theta))^2$
  linear: $L(y,x,\theta) = |y - f(x;\theta)|$
  binary: $L(y,x,\theta) = step(-yf(x;\theta))$
- Empirical $R_{emp}(\theta)$ *approximates* the true risk (expected error)
  $R(\theta) = E_P\{L(x,y,\theta)\} = \int_{X\times Y} P(x,y) L(x,y,\theta)\, dx\, dy \in [0,1]$
- But, we don't know the true P(x,y)!
- If infinite data, *law of large numbers* says:
  $\lim_{N\to\infty} \min_\theta R_{emp}(\theta) = \min_\theta R(\theta)$
- But, in general, can't make guarantees for ERM solution:
  $\arg\min_\theta R_{emp}(\theta) \neq \arg\min_\theta R(\theta)$

Bounding the True Risk

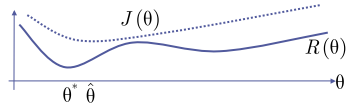
- ERM is inconsistent
  not guaranteed
  may do better on training than on test!



$R(\hat{\theta}) \geq R_{emp}(\hat{\theta})$

- Idea: add a prior or regularizer to $R_{emp}(\theta)$
- Define capacity or confidence = $C(\theta)$ which favors simpler θ

$J(\theta) = R_{emp}(\theta) + C(\theta)$



- If, $R(\theta) \leq J(\theta)$ we have bound $J(\theta)$ is a guaranteed risk
- After train, can guarantee future error rate is $\leq \min_\theta J(\theta)$

Bound the True Risk with VC

- But, how to find a guarantee? Difficult, but there is one…
- Theorem (Vapnik): with probability 1-η where η is a number between [0,1], the following bound holds:

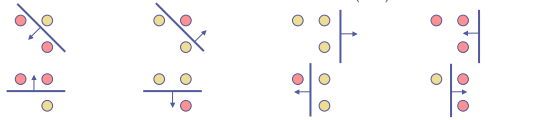
$$R(\theta) \leq J(\theta) = R_{emp}(\theta) + \frac{2h\log\left(\frac{2eN}{h}\right) + 2\log\left(\frac{1}{\eta}\right)}{N}\left(1 + \sqrt{1 + \frac{NR_{emp}(\theta)}{h\log\left(\frac{2eN}{h}\right) + \log\left(\frac{1}{\eta}\right)}}\right)$$

  N = number of data points
  h = Vapnik-Chervonenkis (VC) dimension (1970's)
  = capacity of the classifier class $f(.;\theta)$

- Note, above is independent of the true P(x,y)
- A *worst-case scenario* bound, guaranteed for all P(x,y)
- VC dimension not just the # of parameters a classifier has
- VC measures # of different datasets it can classify perfectly
- Structural Risk Minimization: minimize risk bound J(θ)

VC Dimension & Shattering

- How to compute h or VC for a family of functions $f(.;\theta)$
  h = # of training points that can be shattered
- Recall, classifier maps input to output $f(x;\theta) \to y \in \{-1,1\}$
- Shattering: I pick h points & place them at $x_1,\ldots,x_h$
  You challenge me with 2^h possible labelings $y_1,\ldots,y_h \in \{\pm 1\}^h$
  VC dimension is maximum # of points I can place which a $f(x;\theta)$ can correctly classify for arbitrary labeling $y_1,\ldots,y_h$
- Example: for 2d linear classifier h=3 $f(x;\theta) = x_1\theta_1 + x_2\theta_2 + \theta_0$

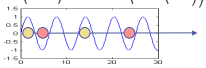


can't ever shatter 4 points! or 3 points on a straight line…

VC Dimension & Shattering

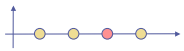
- More generally for higher dimensional linear classifiers, a hyperplane in $\mathbb{R}^d$ shatters any set of linearly independent points. Can choose d+1 linearly indep. points so h=d+1
- Note: VC is *not necessarily proportional* to # of parameters
- Example: sinusoidal 1d classifier $f(x;\theta) = sign(\sin(\theta x))$

  number of parameters=1
  …but… h=infinity!



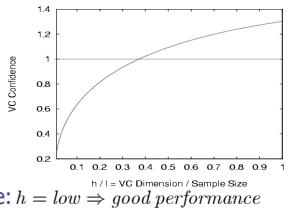
  since I can choose: $x_i = 10^{-i} \quad i = 1,\ldots,h$
  no matter what labeling you challenge: $y_1,\ldots,y_h \in \{\pm 1\}^h$
  using $\theta = \pi\left(1 + \sum_{i=1}^{h} \frac{1}{2}(1 - y_i)10^{-i}\right)$ shatters perfectly

But, as a side note, if I choose 4 equally spaced x's then cannot shatter



VC Dimension & Shattering

- Recall that VC dimension gives an upper bound
- We want to minimize h since that minimizes C(θ) & J(θ)
- If can't compute h exactly but can compute h^+ can plug in h^+ in bound & still guarantee
- Also, sometimes bound is trivial
- Need h/N = 0.3 before C(θ)<1 (recall R(θ) in [0,1])

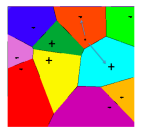


- Note: $h = low \Rightarrow good\ performance \qquad h = \infty \not\Rightarrow poor\ performance$

Nearest Neighbors & VC

- Consider Nearest Neighbors classification algorithm:

  Input a query example x
  Find training example x_i in $\{x_1,\ldots x_N\}$ closest to x
  Predict label for x as y_i of neighbor

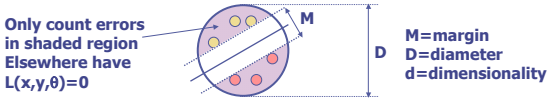


- Often use Euclidean distance $\|x - x_i\|$ to measure closeness
- Nearest neighbors shatters any set of points!
- So VC=infinity, C(θ)=infinity, guaranteed risk=infinity
- But still works well in practice

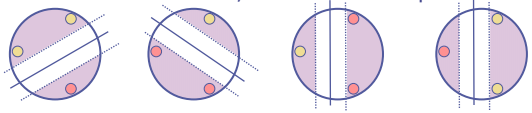
$h = \infty \not\Rightarrow poor\ performance \qquad h = low \Rightarrow good\ performance$

VC Dimension & Large Margins

- Linear classifiers are too big a function class since h=d+1
- Can reduce VC dimension if we restrict them
- Constrain linear classifiers to data living inside a sphere
- Gap-Tolerant classifiers: a linear classifier whose activity is constrained to a sphere & outside a margin

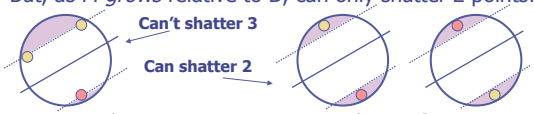


- If M is small relative to D, can still shatter 3 points:



VC Dimension & Large Margins

- But, as M *grows* relative to D, can only shatter 2 points!



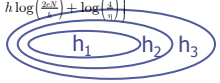
- For hyperplanes, as M grows vs. D, shatter fewer points!
- VC dimension h goes down if gap-tolerant classifier has larger margin, general formula is: $h \le \min\left\{ceil\left[\frac{D^2}{M^2}\right], d\right\}+1$
- Before, just had h=d+1. Now we have a smaller h
- If data is anywhere, D is infinite and back to h=d+1
- Typically real data is bounded (by sphere), D is fixed
- Maximizing M reduces h, improving guaranteed risk J(θ)
- Note: R(θ) doesn't count errors in margin or outside sphere

Structural Risk Minimization

- Structural Risk Minimization: minimize risk bound J(θ) reducing empirical error & reduce VC dimension h

$$R(\theta) \le J(\theta) = R_{emp}(\theta) + \frac{2h\log\left(\frac{2eN}{h}\right) + 2\log\left(\frac{4}{\eta}\right)}{N}\left(1+\sqrt{1+\frac{NR_{emp}(\theta)}{h\log\left(\frac{2eN}{h}\right)+\log\left(\frac{4}{\eta}\right)}}\right)$$

for each model i in list of hypothesis
1) compute its h=h_i
2) $\theta^* = \arg\min_\theta R_{emp}(\theta)$
3) compute $J(\theta^*, h_i)$
choose model with lowest $J(\theta^*, h_i)$



Space of different Classifiers or Hypotheses

- Or, directly optimize over both $(\theta^*, h) = \arg\min_{\theta,h} J(\theta, h)$
- If possible, min empirical error while also minimizing VC
- For gap-tolerant linear classifiers, minimize $R_{emp}(\theta)$ *while* maximizing margin, support vector machines do just that!

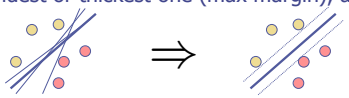
Support Vector Machines

- Support vector machines are (in the simplest case) linear classifiers that do structural risk minimization (SRM)
- Directly maximize margin to reduce guaranteed risk J(θ)
- Assume first the 2-class data is linearly separable:

$have\ \{(x_1,y_1),\ldots,(x_N,y_N)\}\ \ where\ x_i \in \mathbb{R}^D\ and\ y_i \in \{-1,1\}$

$f(x;\theta) = sign(w^T x + b)$

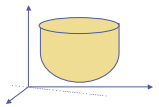
- Decision boundary or hyperplane given by $w^T x + b = 0$
- Note: can scale w & b while keeping same boundary
- Many solutions exist which have empirical error $R_{emp}(\theta)=0$
- Want widest or thickest one (max margin), also it's unique!



Side Note: Constraints

- How to minimize a function subject to equality constraints?

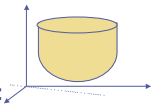
$$\min_{x_1,x_2} f(\vec{x}) = \min_{x_1,x_2} b_1x_1 + b_2x_2 + \tfrac{1}{2}H_{11}x_1^2 + H_{12}x_1x_2 + \tfrac{1}{2}H_{22}x_2^2$$
$$= \min_{\vec{x}} \vec{b}^T\vec{x} + \tfrac{1}{2}\vec{x}^T H\vec{x}$$
$$\Rightarrow \frac{\partial f}{\partial \vec{x}} = \vec{b} + H\vec{x} = 0$$
$$\Rightarrow \vec{x} = -H^{-1}b$$



- Only walk on $x_1=2x_2$ or… $x_1-2x_2=0$…
- Use Lagrange Multipliers, for each constraint, subtract it times a lambda variable. Lambda blows up the minimization if we don't satisfy the constraint:

$$\min_{x_1,x_2} \max_\lambda f(\vec{x}) - \lambda(equality\ condition = 0)$$
$$= \min_{x_1,x_2} \max_\lambda b_1x_1 + b_2x_2 + \tfrac{1}{2}H_{11}x_1^2 + H_{12}x_1x_2 + \tfrac{1}{2}H_{22}x_2^2 - \lambda(x_1 - 2x_2)$$

Side Note: Constraints



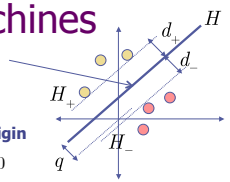
- Cost minimization with equality constraints:
  1) Subtract each constraint times an extra variable (a Lagrange multiplier λ, like an adversary variable)
  2) Take partials with respect to x and set to zero
  3) Plug solution into constraint to find lambda

$$x_1 - 2x_2 = \vec{x}^T\begin{bmatrix}1\\-2\end{bmatrix}$$

$$\min_{\vec{x}} \max_\lambda f(\vec{x}) - \lambda(equality\ condition = 0)$$
$$= \min_{\vec{x}} \max_\lambda b^T\vec{x} + \tfrac{1}{2}\vec{x}^T H\vec{x} - \lambda(x_1 - 2x_2)$$
$$\Rightarrow \frac{\partial f}{\partial \vec{x}} = \vec{b} + H\vec{x} - \lambda\begin{bmatrix}1\\-2\end{bmatrix} = 0 \ \Rightarrow\ \vec{x} = H^{-1}\lambda\begin{bmatrix}1\\-2\end{bmatrix} - H^{-1}b$$
$$\Rightarrow \left(H^{-1}\lambda\begin{bmatrix}1\\-2\end{bmatrix} - H^{-1}b\right)^T\begin{bmatrix}1\\-2\end{bmatrix} = 0 \Rightarrow \lambda = \frac{b^TH^{-1}\begin{bmatrix}1\\-2\end{bmatrix}}{\begin{bmatrix}1\\-2\end{bmatrix}^T H^{-1}\begin{bmatrix}1\\-2\end{bmatrix}}$$

Support Vector Machines

- Define: $w^T x + b = 0$



H_+=positive margin hyperplane
H_-=negative margin hyperplane
q =distance from decision plane to origin

$$q = \min_x \|\vec{x} - \vec{0}\| \quad subject\ to \quad w^T x + b = 0$$
$$\min_x \tfrac{1}{2}\|\vec{x} - \vec{0}\|^2 - \lambda(w^T x + b)$$

1) grad $\frac{\partial}{\partial x}\left(\frac{1}{2}x^T x - \lambda(w^T x + b)\right) = 0$ **2) plug into constraint** $w^T x + b = 0$

$$x - \lambda w = 0 \qquad w^T(\lambda w) + b = 0$$
$$x = \lambda w \qquad \lambda = -\frac{b}{w^T w}$$

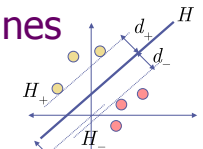
3) Sol'n $\hat{x} = -\left(\frac{b}{w^T w}\right)w$

4) distance $q = \|\hat{x} - \vec{0}\| = \left\|-\frac{b}{w^T w}w\right\| = \frac{|b|}{w^T w}\sqrt{w^T w} = \frac{|b|}{\|w\|}$

5) Define without loss of generality since can scale b & w

$$H \rightarrow w^T x + b = 0$$
$$H_+ \rightarrow w^T x + b = +1$$
$$H_- \rightarrow w^T x + b = -1$$

Support Vector Machines



- The constraints on the SVM for $R_{emp}(\theta)=0$ are thus:

$$w^T x_i + b \ge +1 \quad \forall y_i = +1$$
$$w^T x_i + b \le -1 \quad \forall y_i = -1$$

- Or more simply: $y_i(w^T x_i + b) - 1 \ge 0$
- The margin of the SVM is:

$$m = d_+ + d_-$$

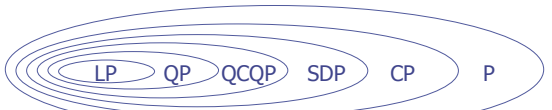
$H_+ \rightarrow w^T x + b = +1$
$H_- \rightarrow w^T x + b = -1$

- Distance to origin: $H \rightarrow q = \frac{|b|}{\|w\|}$ $H_+ \rightarrow q_+ = \frac{|b-1|}{\|w\|}$ $H_- \rightarrow q_- = \frac{|-1-b|}{\|w\|}$
- Therefore: $d_+ = d_- = \frac{1}{\|w\|}$ and margin $m = \frac{2}{\|w\|}$
- Want to max margin, or equivalently minimize: $\|w\|$ *or* $\|w\|^2$
- SVM Problem: $\min \frac{1}{2}\|w\|^2 \quad subject\ to \quad y_i(w^T x_i + b) - 1 \ge 0$
- This is a quadratic program!
- Can plug this into a matlab function called "qp()", done!

Side Note: Optimization Tools

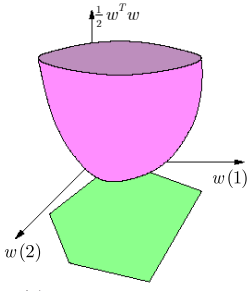
- A hierarchy of Matlab optimization packages to use:

Linear Programming $\min_{\vec{x}} \vec{b}^T\vec{x}\ \ s.t.\ \vec{c}_i^T\vec{x} \ge \alpha_i\ \ \forall i$
<Quadratic Programming $\min_{\vec{x}} \frac{1}{2}\vec{x}^T H\vec{x} + \vec{b}^T\vec{x}\ \ s.t.\ \vec{c}_i^T\vec{x} \ge \alpha_i\ \ \forall i$
<Quadratically Constrained Quadratic Programming
<Semidefinite Programming
<Convex Programming
<Polynomial Time Algorithms



Side Note: Optimization Tools

- Each data point adds $y_i\left(w^T x_i + b\right) - 1 \geq 0$ linear inequality to QP
- Each point cuts a half plane of allowable SVMs and reduces green region
- The SVM is closest point to the origin that is still in the green region
- The preceptron algorithm just puts us randomly in green region
- QP runs in cubic polynomial time
- There are D values in the w vector
- Needs $O(D^3)$ run time… But, there is a DUAL SVM in $O(N^3)$!



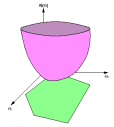
SVM in Dual Form

- We can also solve the problem via convex duality
- Primal SVM problem L_P: $\min \frac{1}{2}\|w\|^2 \;\; subject\ to \;\; y_i\left(w^T x_i + b\right) - 1 \geq 0$
- This is a quadratic program, quadratic cost function with multiple linear inequalities (these carve out a convex hull)
- Subtract from cost each inequality times an α Lagrange multiplier, take derivatives of w & b:

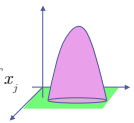
$$L_P = \min_{w,b} \max_{\alpha \geq 0} \frac{1}{2}\|w\|^2 - \sum_i \alpha_i \left(y_i\left(w^T x_i + b\right) - 1\right)$$

$$\frac{\partial}{\partial w} L_P = w - \sum_i \alpha_i y_i x_i = 0 \;\rightarrow\; w = \sum_i \alpha_i y_i x_i$$

$$\frac{\partial}{\partial b} L_P = -\sum_i \alpha_i y_i = 0$$



- Plug back in, dual: $L_D = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j$
- Also have constraints: $\sum_i \alpha_i y_i = 0 \;\; \& \;\; \alpha_i \geq 0$



- Above L_D must be maximized! convex duality… also qp()

Machine Learning

Topic 6

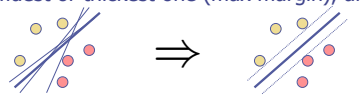
- Review: Support Vector Machines
- Primal & Dual Solution
- Non-separable SVMs
- Kernels
- SVM Demo

Review: SVM

- Support vector machines are (in the simplest case) linear classifiers that do structural risk minimization (SRM)
- Directly maximize margin to reduce guaranteed risk J(θ)
- Assume first the 2-class data is linearly separable:

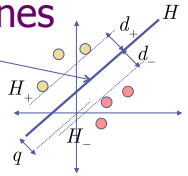
$$have\ \{(x_1,y_1),\ldots,(x_N,y_N)\}\ \ where\ x_i\in\mathbb{R}^D\ and\ y_i\in\{-1,1\}$$
$$f(x;\theta)=sign(w^Tx+b)$$

- Decision boundary or hyperplane given by $w^Tx+b=0$
- Note: can scale w & b while keeping same boundary
- Many solutions exist which have empirical error $R_{emp}(\theta)=0$
- Want widest or thickest one (max margin), also it's unique!



Support Vector Machines

- Define: $w^Tx+b=0$



H_+=positive margin hyperplane
H_-=negative margin hyperplane
q =distance from decision plane to origin

$$q=\min_x\|\vec{x}-\vec{0}\|\ \ subject\ to\ \ w^Tx+b=0$$
$$\min_x \tfrac{1}{2}\|\vec{x}-\vec{0}\|^2-\lambda\left(w^Tx+b\right)$$

1) grad $\frac{\partial}{\partial x}\left(\frac{1}{2}x^Tx-\lambda\left(w^Tx+b\right)\right)=0$

2) plug into constraint $w^Tx+b=0$, $w^T(\lambda w)+b=0$

$$x-\lambda w=0$$
$$x=\lambda w \qquad \lambda=-\frac{b}{w^Tw}$$

3) Sol'n $\hat{x}=-\left(\frac{b}{w^Tw}\right)w$

4) distance $q=\|\hat{x}-\vec{0}\|=\left\|-\frac{b}{w^Tw}w\right\|=\frac{|b|}{w^Tw}\sqrt{w^Tw}=\frac{|b|}{\|w\|}$

5) Define without loss of generality since can scale b & w

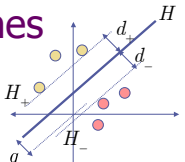
$$H\ \rightarrow w^Tx+b=0$$
$$H_+\rightarrow w^Tx+b=+1$$
$$H_-\rightarrow w^Tx+b=-1$$

Support Vector Machines

- The constraints on the SVM for $R_{emp}(\theta)=0$ are thus:

$$w^Tx_i+b\geq+1\quad\forall y_i=+1$$
$$w^Tx_i+b\leq-1\quad\forall y_i=-1$$

- Or more simply: $y_i\left(w^Tx_i+b\right)-1\geq0$
- The margin of the SVM is:



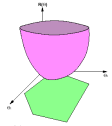
$H_+\rightarrow w^Tx+b=+1$
$H_-\rightarrow w^Tx+b=-1$

$$m=d_++d_-$$

- Distance to origin: $H\rightarrow q=\frac{|b|}{\|w\|}\quad H_+\rightarrow q_+=\frac{|b-1|}{\|w\|}\quad H_-\rightarrow q_-=\frac{|-1-b|}{\|w\|}$
- Therefore: $d_+=d_-=\frac{1}{\|w\|}$ and margin $m=\frac{2}{\|w\|}$
- Want to max margin, or equivalently minimize: $\|w\|$ or $\|w\|^2$
- SVM Problem: $\min\frac{1}{2}\|w\|^2\ \ subject\ to\ \ y_i\left(w^Tx_i+b\right)-1\geq0$
- This is a quadratic program!
- Can plug this into a matlab function called "qp()", done!

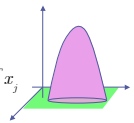
SVM in Dual Form

- We can also solve the problem via convex duality
- Primal SVM problem L_P: $\min\frac{1}{2}\|w\|^2\ \ subject\ to\ \ y_i\left(w^Tx_i+b\right)-1\geq0$
- This is a quadratic program, quadratic cost function with multiple linear inequalities (these carve out a convex hull)



- Subtract from cost each inequality times an α Lagrange multiplier, take derivatives of w & b:

$$L_P=\min_{w,b}\max_{\alpha\geq0}\tfrac{1}{2}\|w\|^2-\sum_i\alpha_i\left(y_i\left(w^Tx_i+b\right)-1\right)$$
$$\frac{\partial}{\partial w}L_P=w-\sum_i\alpha_iy_ix_i=0\ \rightarrow\ w=\sum_i\alpha_iy_ix_i$$
$$\frac{\partial}{\partial b}L_P=-\sum_i\alpha_iy_i=0$$



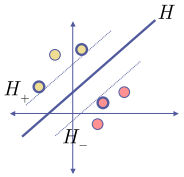
- Plug back in, dual: $L_D=\sum_i\alpha_i-\frac{1}{2}\sum_i\sum_j\alpha_i\alpha_jy_iy_jx_i^Tx_j$
- Also have constraints: $\sum_i\alpha_iy_i=0\ \ \&\ \ \alpha_i\geq0$
- Above L_D must be maximized! convex duality… also qp()

SVM Dual Solution Properties

- We have dual convex program:

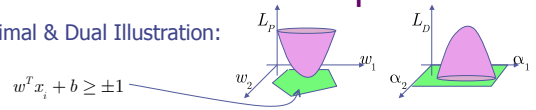
$$\sum_i\alpha_i-\tfrac{1}{2}\sum_{i,j}\alpha_i\alpha_jy_iy_jx_i^Tx_j\ \ \ subject\ to\ \ \sum_i\alpha_iy_i=0\ \ \&\ \ \alpha_i\geq0$$

- Solve for N alphas (one per data point) instead of D w's
- Still convex (qp) so unique solution, gives alphas
- Alphas can be used to get w: $w=\sum_i\alpha_iy_ix_i$
- Support Vectors: have non-zero alphas shown with thicker circles, all live on the margin: $w^Tx_i+b=\pm1$
- Solution is sparse, most alphas=0 these are *non-support vectors* SVM ignores them if they move (without crossing margin) or if they are deleted, SVM doesn't change (stays robust)



SVM Dual Solution Properties

- Primal & Dual Illustration:



$w^Tx_i+b\geq\pm1$

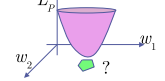
- Recall we could get w from alphas: $w=\sum_i\alpha_iy_ix_i$
- Or could use as is: $f(x)=sign\left(x^Tw+b\right)=sign\left(\sum_i\alpha_iy_ix^Tx_i+b\right)$
- Karush-Kuhn-Tucker Conditions (KKT): solve value of b on margin (for nonzero alphas) have: $w^Tx_i+b=y_i$ using known w, compute b for each support vector

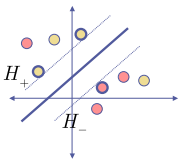
$$\tilde{b}_i=y_i-w^Tx_i\quad\forall i:\alpha_i>0\qquad\text{then…}\qquad b=average\left(\tilde{b}_i\right)$$

- Sparsity (few nonzero alphas) is useful for several reasons
- Means SVM only uses some of training data to learn
- Should help improve its ability to generalize to test data
- Computationally faster when using final learned classifier

Non-Separable SVMs

- What happens when non-separable?
- There is no solution and convex hull shrinks to nothing





- Not all constraints can be resolved, their alphas go to ∞
- Instead of perfectly classifying each point: $y_i\left(w^Tx_i+b\right)\geq1$ we "Relax" the problem with (positive) slack variables xi's allow data to (sometimes) fall on wrong side, for example:

$$w^Tx_i+b\geq-0.03\quad if\ y_i=+1$$

- New constraints: $w^Tx_i+b\geq+1-\xi_i\quad if\ y_i=+1\quad where\ \xi_i\geq0$

$$w^Tx_i+b\leq-1+\xi_i\quad if\ y_i=-1\quad where\ \xi_i\geq0$$

- But too much xi's means too much slack, so penalize them

$$L_P:\ \min\tfrac{1}{2}\|w\|^2+C\sum_i\xi_i\quad subject\ to\ \ y_i\left(w^Tx_i+b\right)-1+\xi_i\geq0$$

Non-Separable SVMs

- This new problem is still convex, still qp()!
- User chooses scalar C (or cross-validates) which controls how much slack xi to use (how non-separable) and how robust to outliers or bad points on the wrong side

Large margin · Low slack · On right side · For xi positivity

$$L_P: \min \tfrac{1}{2}\|w\|^2 + C\sum_i \xi_i - \sum_i \alpha_i \left(y_i\left(w^T x_i + b\right) - 1 + \xi_i\right) - \sum_i \beta_i \xi_i$$

$\frac{\partial}{\partial b} L_P$ and $\frac{\partial}{\partial w} L_P$ as before...

$$\frac{\partial}{\partial \xi_i} L_P = C - \alpha_i - \beta_i = 0$$

$$\alpha_i = C - \beta_i \text{ but... } \alpha_i \& \beta_i \geq 0$$

$$\therefore 0 \leq \alpha_i \leq C$$

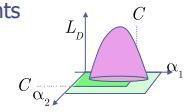
- Can now write dual problem (to maximize):

$$L_D: \max \sum_i \alpha_i - \tfrac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j \text{ subject to } \sum_i \alpha_i y_i = 0 \text{ and } \alpha_i \in [0, C]$$

- Same dual as before but alphas can't grow beyond C

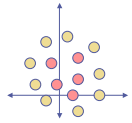
Non-Separable SVMs

- As we try to enforce a classification for a data point its Lagrange multiplier alpha keeps growing endlessly
- Clamping alpha to stop growing at C makes SVM "give up" on those non-separable points
- The dual program is now:
- Solve as before with extra constraints that alphas positive AND less than C... gives alphas... from alphas get $w = \sum_i \alpha_i y_i x_i$



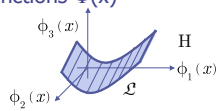
- Karush-Kuhn-Tucker Conditions (KKT): solve value of b on margin for not=zero alphas AND not=C alphas for all others have support vectors, assume $\xi_i = 0$ and use formula $y_i\left(w^T x_i + \tilde{b}_i\right) - 1 + \xi_i = 0$ to get $\tilde{b}_i$ and $b = average\left(\tilde{b}_i\right)$
- Mechanical analogy: support vector forces & torques

Nonlinear SVMs



- What if the problem is not linear?
- We can use our old trick...
- Map d-dimensional x data from L-space to high dimensional H (Hilbert) feature-space via basis functions Φ(x)
- For example, quadratic classifier:

$$x_i \rightarrow \Phi\left(x_i\right) \quad via \quad \Phi\left(\vec{x}\right) = \begin{bmatrix} \vec{x} \\ vec\left(\vec{x}\vec{x}^T\right) \end{bmatrix}$$



- Call phi's feature vectors computed from original x inputs
- Replace all x's in the SVM equations with phi's
- Now solve the following learning problem:

$$L_D: \max \sum_i \alpha_i - \tfrac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \phi\left(x_i\right)^T \phi\left(x_j\right) \quad s.t. \ \alpha_i \in [0, C], \sum_i \alpha_i y_i = 0$$

- Which gives a nonlinear classifier in original space:

$$f\left(x\right) = sign\left(\sum_i \alpha_i y_i \phi\left(x\right)^T \phi\left(x_i\right) + b\right)$$

Kernels (see http://www.youtube.com/watch?v=3liCbRZPrZA)

- One important aspect of SVMs: all math involves only the *inner products* between the phi features!

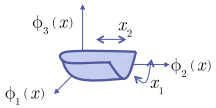
$$f\left(x\right) = sign\left(\sum_i \alpha_i y_i \phi\left(x\right)^T \phi\left(x_i\right) + b\right)$$

- Replace all inner products with a general kernel function
- Mercer kernel: accepts 2 inputs and outputs a scalar via:

$$k\left(x, \tilde{x}\right) = \left\langle \phi\left(x\right), \phi\left(\tilde{x}\right) \right\rangle = \begin{cases} \phi\left(x\right)^T \phi\left(\tilde{x}\right) & for\ finite\ \phi \\ \int_t \phi\left(x, t\right) \phi\left(\tilde{x}, t\right) dt & otherwise \end{cases}$$

- Example: quadratic polynomial

$$\phi\left(x\right) = \begin{bmatrix} x_1^2 & \sqrt{2}x_1 x_2 & x_2^2 \end{bmatrix}^T$$



$$k\left(x, \tilde{x}\right) = \phi\left(x\right)^T \phi\left(\tilde{x}\right) = x_1^2 \tilde{x}_1^2 + 2x_1 \tilde{x}_1 x_2 \tilde{x}_2 + x_2^2 \tilde{x}_2^2 = \left(x_1 \tilde{x}_1 + x_2 \tilde{x}_2\right)^2$$

Kernels

- Sometimes, many Φ(x) will produce the same k(x,x')
- Sometimes k(x,x') computable but features huge or infinite!
- Example: polynomials

If explicit polynomial mapping, feature space Φ(x) is huge d-dimensional data, p-th order polynomial, $\dim(H) = \binom{d+p-1}{p}$ images of size 16x16 with p=4 have dim(H)=183million but can equivalently just use kernel: $k\left(x, y\right) = \left(x^T y\right)^p$

$$k\left(x, \tilde{x}\right) = \left(x^T \tilde{x}\right)^p = \left(\sum_i x_i \tilde{x}_i\right)^p$$

Multinomial Theorem

$$\propto \sum_r \frac{p!}{r_1! r_2! r_3! \ldots \left(p - r_1 - r_2 - \ldots\right)!} x_1^{r_1} x_2^{r_2} \cdots x_d^{r_d} \tilde{x}_1^{r_1} \tilde{x}_2^{r_2} \cdots \tilde{x}_d^{r_d}$$

w=weight on term

$$\propto \sum_r \left(\sqrt{w_r} x_1^{r_1} x_2^{r_2} \cdots x_d^{r_d}\right)\left(\sqrt{w_r} \tilde{x}_1^{r_1} \tilde{x}_2^{r_2} \cdots \tilde{x}_d^{r_d}\right)$$

$$\propto \phi\left(x\right)\phi\left(\tilde{x}\right)$$

Equivalent!

Kernels

- Replace each $x_i^T x_j \rightarrow k\left(x_i, x_j\right)$, for example:

P-th Order Polynomial Kernel: $k\left(x, \tilde{x}\right) = \left(x^T \tilde{x} + 1\right)^p$

RBF Kernel (infinite!): $k\left(x, \tilde{x}\right) = \exp\left(-\frac{1}{2\sigma^2}\left\|x - \tilde{x}\right\|^2\right)$

Sigmoid (hyperbolic tan) Kernel: $k\left(x, \tilde{x}\right) = \tanh\left(\kappa x^T \tilde{x} - \delta\right)$

- Using kernels we get generalized inner product SVM:

$$L_D: \max \sum_i \alpha_i - \tfrac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j k\left(x_i, x_j\right) \quad s.t. \ \alpha_i \in [0, C], \sum_i \alpha_i y_i = 0$$

$$f\left(x\right) = sign\left(\sum_i \alpha_i y_i k\left(x_i, x\right) + b\right)$$

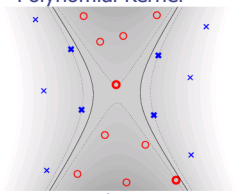
- Still qp solver, just use Gram matrix K (positive definite)

$$K = \begin{bmatrix} k\left(x_1, x_1\right) & k\left(x_1, x_2\right) & k\left(x_1, x_3\right) \\ k\left(x_1, x_2\right) & k\left(x_2, x_2\right) & k\left(x_2, x_3\right) \\ k\left(x_1, x_3\right) & k\left(x_2, x_3\right) & k\left(x_3, x_3\right) \end{bmatrix} \qquad K_{ij} = k\left(x_i, x_j\right)$$

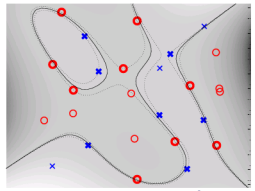
Kernelized SVMs

- Polynomial kernel: $k\left(x_i, x_j\right) = \left(x_i^T x_j + 1\right)^p$
- Radial basis function kernel: $k\left(x_i, x_j\right) = \exp\left(-\frac{1}{2\sigma^2}\left\|x_i - x_j\right\|^2\right)$

Polynomial Kernel



RBF kernel

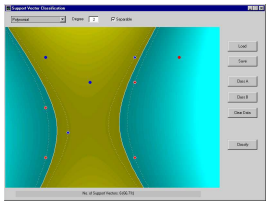


- Least-squares, logistic-regression, perceptron are also kernelizable

SVM Demo

- SVM Demo by Steve Gunn:
  http://www.isis.ecs.soton.ac.uk/resources/svminfo/
- In svc.m replace
  [alpha lambda how] = qp(...);
  with
  [alpha lambda how] = quadprog(H,c,[],[],A,b,vlb,vub,x0);

This replaces the old Matlab command qp (quadratic programming) with the new one for more recent versions

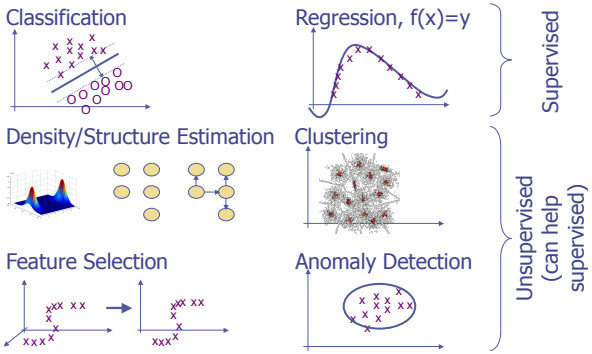


Machine Learning

Topic 7

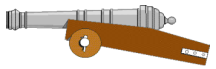
- Unsupervised Learning
- Statistical Perspective
- Probability Models
- Discrete & Continuous: Gaussian, Bernoulli, Multinomial
- Maximum Likelihood → Logistic Regression
- Conditioning, Marginalizing, Bayes Rule, Expectations
- Classification, Regression, Detection
- Dependence/Independence
- Maximum Likelihood → Naïve Bayes

Unsupervised Learning



Statistical Perspective

- Several problems with framework so far:
  - Only have input-output approaches (SVM, Neural Net)
  - Pulled non-linear squashing functions out of a hat
  - Pulled loss functions (squared error, etc.) out of a hat
- Better approach for classification?
- What if we have multi-class classification?
- What if other problems, i.e. unobserved values of x,y,etc…
- Also, what if we don't have a true function?
- Example of Projectile Cannon (c.f. Distal Learning)

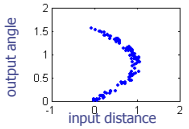


- Would like to train a regression function to control a cannon's angle of fire (y) given target distance (x)

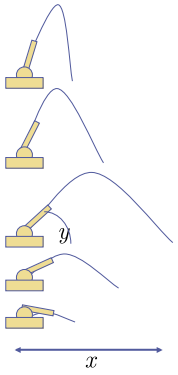
Statistical Perspective

- Example of Projectile Cannon (45 degree problem)
  x = input target distance
  y = output cannon angle

$$x = \frac{v(0)^2}{g}\sin(2y) + noise$$

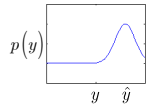


- What does least squares do?
- Conditional statistical models address this problem…



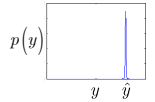
Probability Models

- Instead of deterministic functions, output is a probability
- Previously: our output was a scalar $\hat{y} = f(x) = \theta^T x + b$
- Now: our output is a probability $p(y)$
  e.g. a probability bump:



- $p(y)$ subsumes or is a superset of $\hat{y}$
- Why is this representation for our answer more general?
  → A deterministic answer $\hat{y}$ with complete confidence is like putting a probability $p(y)$ where all the mass is at $\hat{y}$!

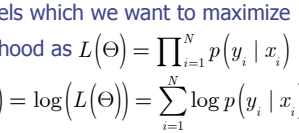
$$\hat{y} \Leftrightarrow p(y) = \delta(y - \hat{y})$$



Probability Models

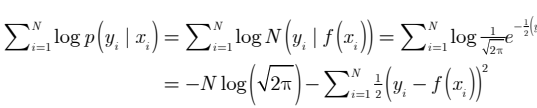
- Now: our output is a probability density function (pdf) $p(y)$
- Probability Model: a family of pdf's with adjustable parameters which lets us select one of many $p(y) \rightarrow p(y \mid \Theta)$
- E.g.: 1-dim Gaussian distribution
  'given' 'mean' parameter μ:

$$p(y \mid \mu) = N(y \mid \mu) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y-\mu)^2}$$



- Want mean centered on f(x)'s value $p(y) = N(y \mid f(x))$
- Now, linear regression is:

$$N(y \mid f(x)) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y-f(x))^2} = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y-\theta^T x-b)^2}$$



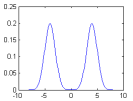
Probability Models

- To fit to data, we typically "maximize likelihood" of the probability model
- Log-likelihood = objective function (i.e. negative of cost) for probability models which we want to maximize
- Define (conditional) likelihood as $L(\Theta) = \prod_{i=1}^{N} p(y_i \mid x_i)$
  or log-Likelihood as $l(\Theta) = \log(L(\Theta)) = \sum_{i=1}^{N} \log p(y_i \mid x_i)$
- For Gaussian p(y|x), maximum likelihood is least squares!

$$\sum_{i=1}^{N} \log p(y_i \mid x_i) = \sum_{i=1}^{N} \log N(y_i \mid f(x_i)) = \sum_{i=1}^{N} \log \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y_i - f(x_i))^2}$$
$$= -N \log(\sqrt{2\pi}) - \sum_{i=1}^{N} \frac{1}{2}(y_i - f(x_i))^2$$

Probability Models

- Can extend probability model to 2 bumps:
  $p(y \mid \Theta) = \frac{1}{2} N(y \mid \mu_1) + \frac{1}{2} N(y \mid \mu_2)$



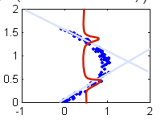
- Each mean can be a linear regression fn.

$$p(y \mid x, \Theta) = \frac{1}{2} N(y \mid f_1(x)) + \frac{1}{2} N(y \mid f_2(x))$$
$$= \frac{1}{2} N(y \mid \theta_1^T x + b_1) + \frac{1}{2} N(y \mid \theta_2^T x + b_2)$$

- Therefore the (conditional) log-likelihood to maximize is:

$$l(\Theta) = \sum_{i=1}^{N} \log\left(\frac{1}{2} N(y_i \mid \theta_1^T x_i + b_1) + \frac{1}{2} N(y_i \mid \theta_2^T x_i + b_2)\right)$$

- Maximize l(θ) using gradient ascent
- Nicely handles the "cannon firing" data



Probability Models

- Now classification: can also go beyond deterministic!
- Previously: wanted output to be binary $\hat{y} = \{0,1\}$
- Now: our output is a probability $p(y)$

  e.g. a probability table:

y=0	y=1
0.73	0.27

α

- This subsumes or is a superset again…
- Consider probability over binary events (coin flips!):

  e.g. Bernoulli distribution (i.e 1x2 probability table) with parameter α

  $p(y \mid \alpha) = \alpha^y (1-\alpha)^{1-y} \qquad \alpha \in [0,1]$
- Linear classification can be done by setting α equal to f(x):

  $p(y \mid x) = f(x)^y (1-f(x))^{1-y} \qquad f(x) \in [0,1]$

Probability Models

- Now linear classification is:

  $p(y \mid x) = f(x)^y (1-f(x))^{1-y} \qquad f(x) \equiv \alpha \in [0,1]$
- Log-likelihood is (negative of cost function):

$$\sum_{i=1}^{N} \log p(y_i \mid x_i) = \sum_{i=1}^{N} \log f(x_i)^{y_i} (1-f(x_i))^{1-y_i}$$
$$= \sum_{i=1}^{N} y_i \log f(x_i) + (1-y_i) \log(1-f(x_i))$$
$$= \sum_{i \in class1} \log f(x_i) + \sum_{i \in class0} \log(1-f(x_i))$$

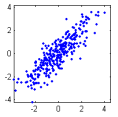
- But, need a squashing function since f(x) in [0,1]
- Use sigmoid or logistic again…

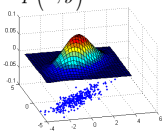
  $f(x) = sigmoid(\theta^T x + b) \in [0,1]$
- Called logistic regression → *new loss function*
- Do gradient descent, similar to logistic output neural net!
- Can also handle multi-layer f(x) and do backprop again!

Generative Probability Models

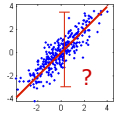
- Idea: Extend probability to describe *both* X and Y
- Find probability density function over both: $p(x,y)$

  E.g. *describe* data with Multi-Dim. Gaussian (later…)





- Called a 'Generative Model' because we can use it to synthesize or re-generate data similar to the training data we learned from
- Regression models & classification boundaries are not as flexible
  don't keep info about X
  don't model noise/uncertainty



Properties of PDFs

- Let's review some basics of probability theory
- First, pdf is a function, multiple inputs, one output:

  $p(x_1,\ldots,x_n) \qquad p(x_1 = 0.3,\ldots,x_n = 1) = 0.2$
- Function's output is always non-negative:

  $p(x_1,\ldots,x_n) \geq 0$
- Can have discrete or continuous or both inputs:

  $p(x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 3.1415)$
- Summing over the domain of all inputs gives unity:

$$\int_{y=-\infty}^{\infty} \int_{x=-\infty}^{\infty} p(x,y)\,dx\,dy = 1 \qquad \sum_y \sum_x p(x,y) = 1$$

0.4	0.1
0.3	0.2

Continuous→integral, Discrete→sum

Properties of PDFs

- Marginalizing: integrate/sum out a variable leaves a marginal distribution over the remaining ones…

  $\sum_y p(x,y) = p(x)$
- Conditioning: if a variable 'y' is 'given' we get a conditional distribution over the remaining ones…

  $p(x \mid y) = \frac{p(x,y)}{p(y)}$
- Bayes Rule: mathematically just redo conditioning but has a deeper meaning (1764)… if we have X being data and θ being a model

likelihood **posterior** **prior** **evidence**

$$p(\theta \mid \mathcal{X}) = \frac{p(\mathcal{X} \mid \theta)\, p(\theta)}{p(\mathcal{X})}$$

Properties of PDFs

- Expectation: can use pdf p(x) to compute averages and expected values for quantities, denoted by:

  $E_{p(x)}\{f(x)\} = \int_x p(x) f(x)\,dx \quad or \quad = \sum_x p(x) f(x)$
- Properties:

$$E\{cf(x)\} = cE\{f(x)\}$$
$$E\{f(x)+c\} = E\{f(x)\} + c$$
$$E\{E\{f(x)\}\} = E\{f(x)\}$$

example: speeding ticket

Fine=0$	Fine=20$
0.8	0.2

expected cost of speeding?
f(x=0)=0, f(x=1)=20
p(x=0)=0.8, p(x=1)=0.2

- Mean: expected value for x

  $E_{p(x)}\{x\} = \int_{-\infty}^{\infty} p(x)\,x\,dx$
- Variance: expected value of (x-mean)^2, how much x varies

$$Var\{x\} = E\left\{\left(x - E\{x\}\right)^2\right\} = E\{x^2 - 2xE\{x\} + E\{x\}^2\}$$
$$= E\{x^2\} - 2E\{x\}E\{x\} + E\{x\}^2 = E\{x^2\} - E\{x\}^2$$

Properties of PDFs

- Covariance: how strongly x and y vary together

$$Cov\{x,y\} = E\{(x - E\{x\})(y - E\{y\})\} = E\{xy\} - E\{x\}E\{y\}$$

- Conditional Expectation: $E\{y \mid x\} = \int_y p(y \mid x)\,y\,dy$

$$E\{E\{y \mid x\}\} = \int_x p(x) \int_y p(y \mid x)\,y\,dy\,dx = E\{y\}$$

- Sample Expectation: If we don't have pdf p(x,y) can approximate expectations using samples of data

  $E_{p(x)}\{f(x)\} \simeq \frac{1}{N}\sum_{i=1}^{N} f(x_i)$
- Sample Mean: $E\{x\} \simeq \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$
- Sample Var: $E\left\{\left(x - E(x)\right)^2\right\} \simeq \frac{1}{N}\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2$
- Sample Cov: $E\{(x - E(x))(y - E(y))\} \simeq \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})$

More Properties of PDFs

- Independence: probabilities of independent variables multiply. Denote with the following notation:

  $x \perp\!\!\!\perp y \;\rightarrow\; p(x,y) = p(x)p(y)$
  $x \perp\!\!\!\perp y \;\rightarrow\; p(x \mid y) = p(x)$

  also note in this case:

$$E_{p(x,y)}\{xy\} = \int_x \int_y p(x)\,p(y)\,xy\,dx\,dy$$
$$= \int_x p(x)\,x\,dx \int_y p(y)\,y\,dy = E_{p(x)}\{x\}E_{p(y)}\{y\}$$

- Conditional independence: when two variables become independent only if another is observed

  $x \perp\!\!\!\perp y \mid z \;\rightarrow\; p(x \mid y,z) = p(x \mid z)$
  $x \perp\!\!\!\perp y \mid z \;\rightarrow\; p(x \mid y) \neq p(x)$

The IID Assumption

- Most of the time, we will assume that a dataset independent and identically distributed (IID)
- In many real situations, data is generated by some black box phenomenon in an arbitrary order.
- Assume we are given a dataset:

  $\mathcal{X} = \{x_1,\ldots,x_N\}$

  "Independent" means that (given the model θ) the probability of our data multiplies:

  $p(x_1,\ldots,x_N \mid \Theta) = \prod_{i=1}^{N} p_i(x_i \mid \Theta)$

  "Identically distributed" means that each marginal probability is the same for each data point

  $p(x_1,\ldots,x_N \mid \Theta) = \prod_{i=1}^{N} p_i(x_i \mid \Theta) = \prod_{i=1}^{N} p(x_i \mid \Theta)$

The IID Assumption

- Bayes rule says likelihood is probability of data given model

$$p(\theta \mid \mathcal{X}) = \frac{p(\mathcal{X} \mid \theta)\, p(\theta)}{p(\mathcal{X})}$$

(posterior: $p(\theta \mid \mathcal{X})$; likelihood: $p(\mathcal{X} \mid \theta)$; prior: $p(\theta)$; evidence: $p(\mathcal{X})$)

- The likelihood of $\mathcal{X} = \{x_1, \ldots, x_N\}$ under IID assumptions is:

$$p(\mathcal{X} \mid \Theta) = p(x_1, \ldots, x_N \mid \Theta) = \prod_{i=1}^{N} p_i(x_i \mid \Theta) = \prod_{i=1}^{N} p(x_i \mid \Theta)$$

- Learn joint distribution $p(x \mid \Theta)$ by maximum likelihood:

$$\Theta^* = \arg\max_{\Theta} \prod_{i=1}^{N} p(x_i \mid \Theta) = \arg\max_{\Theta} \sum_{i=1}^{N} \log p(x_i \mid \Theta)$$

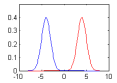
- Learn conditional $p(y \mid x, \Theta)$ by max conditional likelihood:

$$\Theta^* = \arg\max_{\Theta} \prod_{i=1}^{N} p(y_i \mid x_i, \Theta) = \arg\max_{\Theta} \sum_{i=1}^{N} \log p(y_i \mid x_i, \Theta)$$

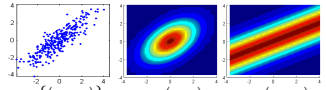
Uses of PDFs

- Classification: have p(x,y) and given x. Asked for discrete y output, give most probable one

$$p(x, y) \rightarrow p(y \mid x) \rightarrow \hat{y} = \arg\max_m p(y = m \mid x)$$



- Regression: have p(x,y) and given x. Asked for a scalar y output, give most probable or expected one

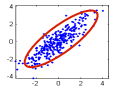


$$\{(x_i, y_i)\} \rightarrow p(x, y) \rightarrow p(y \mid x)$$

$$\hat{y} = \begin{cases} \arg\max_y p(y \mid x) \\ E_{p(y|x)}\{y\} \end{cases}$$

- Anomaly Detection: if have p(x,y) and given both x,y. Asked if it is similar → threshold

$$p(x, y) \geq threshold \rightarrow \{normal, anomaly\}$$



Machine Learning

Topic 8

- Discrete Probability Models
- Independence
- Bernoulli Distribution
- Text: Naïve Bayes
- Categorical / Multinomial Distribution
- Text: Bag of Words

Bernoulli Probability Models

- Bernoulli: recall binary (coin flip) probability, just 1x2 table

$$p(x) = \alpha^x (1-\alpha)^{1-x} \qquad \alpha \in [0,1] \quad x \in \{0,1\}$$

x=0	x=1
0.73	0.27

- Multidimensional Bernoulli: multiple binary events

$p(x_1, x_2)$

	x_2=0	x_2=1
x_1=0	0.4	0.1
x_1=1	0.3	0.2

$p(x_1, x_2, x_3)$

- Why do we write these as an equations instead of tables?

- To do things like… maximum likelihood…
- Fill in the table so that it matches real data…
- Example: coin flips H,H,T,T,T,H,T,H,H,H ???

x=T	x=H

Bernoulli Probability Models

- Bernoulli: recall binary (coin flip) probability, just 1x2 table

$$p(x) = \alpha^x (1-\alpha)^{1-x} \qquad \alpha \in [0,1] \quad x \in \{0,1\}$$

x=0	x=1
0.73	0.27

- Multidimensional Probability Table: multiple binary events

$p(x_1, x_2)$

	x_2=0	x_2=1
x_1=0	0.4	0.1
x_1=1	0.3	0.2

$p(x_1, x_2, x_3)$

- Why do we write these as an equations instead of tables?

- To do things like… maximum likelihood…
- Fill in the table so that it matches real data…
- Example: coin flips H,H,T,T,T,H,T,H,H,H
- Why is this correct?

x=T	x=H
0.4	0.6

Bernoulli Maximum Likelihood

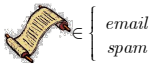
- Bernoulli: $p(x) = \alpha^x (1-\alpha)^{1-x} \qquad \alpha \in [0,1] \quad x \in \{0,1\}$
- Log-Likelihood (IID): $\sum_{i=1}^N \log p(x_i \mid \alpha) = \sum_{i=1}^N \log \alpha^{x_i} (1-\alpha)^{1-x_i}$
- Gradient=0:

$$\frac{\partial}{\partial \alpha} \sum_{i=1}^N \log \alpha^{x_i} (1-\alpha)^{1-x_i} = 0$$
$$\frac{\partial}{\partial \alpha} \sum_{i=1}^N x_i \log \alpha + (1-x_i) \log(1-\alpha) = 0$$
$$\frac{\partial}{\partial \alpha} \sum_{i \in class1} \log \alpha + \sum_{i \in class0} \log(1-\alpha) = 0$$
$$\sum_{i \in class1} \frac{1}{\alpha} - \sum_{i \in class0} \frac{1}{1-\alpha} = 0$$
$$N_1 \frac{1}{\alpha} - N_0 \frac{1}{1-\alpha} = 0$$
$$N_1 (1-\alpha) - N_0 \alpha = 0$$
$$N_1 - (N_1 + N_0)\alpha = 0$$
$$\alpha = \frac{N_1}{N_1 + N_0}$$

x=0	x=1
$\frac{N_0}{N_0+N_1}$	$\frac{N_1}{N_0+N_1}$

Text Modeling via Naïve Bayes

- Naïve Bayes: the simplest model of text



$\in \{ email, spam \}$

- There are about 50,000 words in English
- Each document is D=50,000 dimensional binary vector $\vec{x}_i$
- Each dimension is a word, set to 1 if word in the document

Dim1: "the" = 1
Dim2: "hello" = 0
Dim3: "and" = 1
Dim4: "happy" = 1
…

- Naïve Bayes: assumes each word is independent

$$p(\vec{x}) = p(\vec{x}(1), \ldots, \vec{x}(D)) = \prod_{d=1}^D p(\vec{x}(d))$$
$$= \prod_{d=1}^D \vec{\alpha}(d)^{\vec{x}(d)} (1-\vec{\alpha}(d))^{(1-\vec{x}(d))}$$

- Each 1 dimensional alpha(d) is a Bernoulli parameter
- The whole alpha vector is multivariate Bernoulli

Text Modeling via Naïve Bayes

- Maximum likelihood: assume we have several IID vectors
- Have N documents, each a 50,000 dimension binary vector
- Each dimension is a word, set to 1 if word in the document

		$\vec{x}_1$	$\vec{x}_2$	$\vec{x}_3$	$\vec{x}_4$
Dim1:	**"the" =**	**1**	**0**	**1**	**1**
Dim2:	**"hello" =**	**0**	**1**	**0**	**1**
Dim3:	**"and" =**	**1**	**1**	**0**	**1**
Dim4:	**"happy" =**	**1**	**0**	**0**	**1**

- Likelihood $= \prod_{i=1}^N p(\vec{x}_i \mid \vec{\alpha}) = \prod_{i=1}^N \prod_{d=1}^{50000} \vec{\alpha}(d)^{\vec{x}_i(d)} (1-\vec{\alpha}(d))^{(1-\vec{x}_i(d))}$
- Max likelihood solution: for each word d count number of documents it appears in divided by total N documents $\vec{\alpha}(d) = \frac{N_d}{N}$
- To classify a new document x, build two models α_{+1} α_{-1} & compare $prediction = \arg\max_{y \in \{\pm 1\}} p(\vec{x} \mid \vec{\alpha}_y)$

Categorical Probability Models

- Categorical: a distribution over a single multi-category event

1	2	3	4	5	6
$\vec{\alpha}(1)$	$\vec{\alpha}(2)$	$\vec{\alpha}(3)$	$\vec{\alpha}(4)$	$\vec{\alpha}(5)$	$\vec{\alpha}(6)$

$p(x) = \prod_{m=1}^M \vec{\alpha}(m)^{\vec{x}(m)} \qquad \sum_m \vec{\alpha}(m) = 1 \qquad \vec{x} \in \mathbb{B}^M ; \sum_m \vec{x}(m) = 1$

- Encode events as binary indicator vectors

$\vec{x}(1)$	$\vec{x}(2)$	$\vec{x}(3)$	$\vec{x}(4)$	$\vec{x}(5)$	$\vec{x}(6)$

- Related to the more general *multinomial* distribution
- Find α using Maximum Likelihood (with IID assumption):

$$\sum_{i=1}^N \log p(\vec{x}_i \mid \vec{\alpha}) = \sum_{i=1}^N \log \prod_{m=1}^M \vec{\alpha}(m)^{\vec{x}_i(m)} = \sum_{i=1}^N \sum_{m=1}^M \vec{x}_i(m) \log(\vec{\alpha}(m))$$

- Can't just take gradient over α, use sum= 1 constraint:
- Insert constraint using Lagrange multipliers

$$\frac{\partial}{\partial \alpha_q} \sum_{i=1}^N \sum_{m=1}^M \vec{x}_i(m) \log(\vec{\alpha}(m)) - \lambda \left(\sum_{m=1}^M \vec{\alpha}(m) - 1 \right) = 0$$
$$\sum_{i=1}^N \left(\vec{x}_i(q) \frac{1}{\vec{\alpha}(q)} \right) - \lambda = 0 \quad \Rightarrow \quad \vec{\alpha}(q) = \frac{1}{\lambda} \sum_{i=1}^N \vec{x}_i(q)$$

Categorical Maximum Likelihood

- Taking the gradient with Lagrangian gives this formula for each q: $\vec{\alpha}(q) = \frac{1}{\lambda} \sum_{i=1}^N \vec{x}_i(q)$
- Recall the constraint: $\sum_m \vec{\alpha}(m) - 1 = 0$
- Plug in α's solution: $\sum_m \frac{1}{\lambda} \sum_{i=1}^N \vec{x}_i(m) - 1 = 0$
- Gives the lambda: $\lambda = \sum_m \sum_{i=1}^N \vec{x}_i(m)$
- Final answer: $\vec{\alpha}(q) = \frac{\sum_{i=1}^N \vec{x}_i(q)}{\sum_m \sum_{i=1}^N \vec{x}_i(m)} = \frac{N_q}{N}$
- Example: Rolling dice 1,6,2,6,3,6,4,6,5,6

x=1	x=2	x=3	x=4	x=5	x=6
0.1	0.1	0.1	0.1	0.1	0.5

Multinomial Probability Model

- The multinomial is a categorical over *counts* of events
  Dice: 1,3,1,4,6,1,1 Word Dice: the, dog, jumped, the
- Say document i has W_i=2000 words, each an IID dice roll

$$p(doc_i) = p\left(\vec{x}_i^1, \vec{x}_i^2, \ldots, \vec{x}_i^{W_i}\right) = \prod_{w=1}^{W_i} p\left(\vec{x}_i^w\right) \propto \prod_{w=1}^{W_i} \prod_{d=1}^{D} \vec{\alpha}\left(d\right)^{\vec{x}_i^w(d)}$$

- Get count of each time an event occurred

$$p(doc_i) \propto \prod_{w=1}^{W_i} \prod_{d=1}^{D} \vec{\alpha}\left(d\right)^{\vec{x}_i^w(d)} = \prod_{d=1}^{D} \vec{\alpha}\left(d\right)^{\sum_{w=1}^{W_i} \vec{x}_i^w(d)} = \prod_{d=1}^{D} \vec{\alpha}\left(d\right)^{\vec{X}_i(d)}$$

- BUT: order shouldn't matter when "counting" so multiply by # of possible choosings. Choosing X(1),…X(D) from N

$$\left\{ \begin{matrix} W_i \\ \vec{X}_i(1), \ldots, \vec{X}_i(D) \end{matrix} \right\} = \frac{W_i!}{\prod_{d=1}^{D} \vec{X}_i(d)!} = \frac{\left(\sum_{d=1}^{D} \vec{X}_i(d)\right)!}{\prod_{d=1}^{D} \vec{X}_i(d)!}$$

- Multinomial: over discrete integer vectors X summing to W

$$p\left(\vec{X}_i\right) = \frac{W_i!}{\prod_{d=1}^{D} \vec{X}(d)!} \prod_{d=1}^{D} \vec{\alpha}\left(d\right)^{\vec{X}(d)} \quad s.t. \sum_d \vec{\alpha}\left(d\right) = 1, \vec{X} \in \mathbb{Z}_+^D, \sum_{d=1}^{D} \vec{X}\left(d\right) = W$$

Text Modeling via Multinomial

- Also known as the bag-of-words model

$\in \left\{ \begin{matrix} email \\ spam \end{matrix} \right.$

- Each document is 50,000 dimensional vector
- Each dimension is a word, set to # times word in doc

		X_1	X_2	X_3	X_4
Dim1:	**"the" =**	**9**	**3**	**1**	**0**
Dim2:	**"hello" =**	**0**	**5**	**3**	**0**
Dim3:	**"and" =**	**6**	**2**	**2**	**2**
Dim4:	**"happy" =**	**2**	**5**	**1**	**0**

- Each document is a vector of multinomial counts

$$p\left(doc_i\right) = p\left(\vec{X}_i\right) = \frac{\left(\sum_{d=1}^{D} \vec{X}_i(d)\right)!}{\prod_{d=1}^{D} \vec{X}_i(d)!} \prod_{d=1}^{D} \vec{\alpha}\left(d\right)^{\vec{X}_i(d)} \quad \sum_d \vec{\alpha}\left(d\right) = 1 \quad X \in \mathbb{Z}_+^D$$

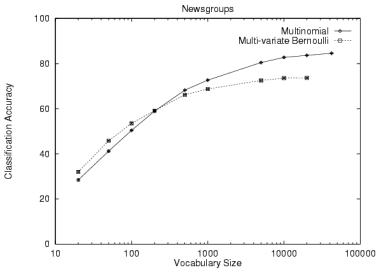
- Log-likelihood: $l\left(\vec{\alpha}\right) = \sum_{i=1}^{N} \log p\left(\vec{X}_i\right) = \sum_{i=1}^{N} \log \frac{\left(\sum_{d=1}^{D} \vec{X}_i(d)\right)!}{\prod_{d=1}^{D} \vec{X}_i(d)!} \prod_{d=1}^{D} \vec{\alpha}\left(d\right)^{\vec{X}_i(d)}$

$$= \sum_{i=1}^{N} \sum_{d=1}^{D} \vec{X}_i\left(d\right) \log \vec{\alpha}\left(d\right) + const$$

- Find α just like the multinomial maximum likelihood formula!

Text Modeling Experiments

- For text modeling (McCallum & Nigam '98)
  Bernoulli better for small vocabulary
  Multinomial better for large vocabulary



Machine Learning

Topic 9

- Continuous Probability Models
- Gaussian Distribution
- Maximum Likelihood Gaussian
- Sampling from a Gaussian

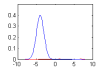
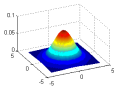
Continuous Probability Models

- Probabilities can have both discrete & continuous variables
- We will discuss:
  1) discrete probability tables

x=T	x=H
0.4	0.6

x=1	x=2	x=3	x=4	x=5	x=6
0.1	0.1	0.1	0.1	0.1	0.5

  2) continuous probability distributions

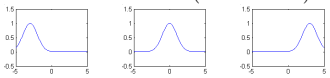



- Most popular continuous distribution = Gaussian

Gaussian Distribution

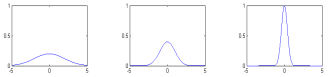
- Recall 1-dimensional Gaussian with mean parameter μ translates Gaussian left & right

$$p(x \mid \mu) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(x-\mu)^2\right)$$



- Can also have variance parameter σ^2 widens or narrows the Gaussian

$$p(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$



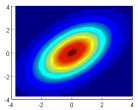
Note: $\int_{x=-\infty}^{\infty} p(x)dx = 1$

Multivariate Gaussian

- Gaussian can extend to D-dimensions
- Gaussian mean parameter μ vector, it translates the bump
- Covariance matrix Σ stretches and rotates bump

$$p(\vec{x} \mid \vec{\mu}, \Sigma) = \frac{1}{(2\pi)^{D/2}\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu})\right)$$

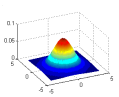
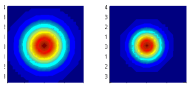
- Mean is any real vector $\quad \vec{x} \in \mathbb{R}^D, \vec{\mu} \in \mathbb{R}^D, \Sigma \in \mathbb{R}^{D \times D}$
- Max and expectation = μ
- Variance parameter is now Σ matrix
- Covariance matrix is positive definite
- Covariance matrix is symmetric
- Need matrix inverse (inv)
- Need matrix determinant (det)
- Need matrix trace operator (trace)



Multivariate Gaussian

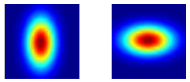
- Spherical:

$$\Sigma = \sigma^2 I = \begin{bmatrix} \sigma^2 & 0 & 0 \\ 0 & \sigma^2 & 0 \\ 0 & 0 & \sigma^2 \end{bmatrix}$$

- Diagonal Covariance: dimensions of x are independent product of multiple 1d Gaussians

$$p(\vec{x} \mid \vec{\mu}, \Sigma) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\vec{\sigma}(d)}} \exp\left(-\frac{(\vec{x}(d)-\vec{\mu}(d))^2}{2\vec{\sigma}(d)^2}\right)$$



$$\Sigma = \begin{bmatrix} \vec{\sigma}(1)^2 & 0 & 0 & 0 \\ 0 & \vec{\sigma}(2)^2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \vec{\sigma}(D)^2 \end{bmatrix}$$

Max Likelihood Gaussian

- Have IID samples as vectors i=1..N: $\quad \mathcal{X} = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$
- How do we recover the mean and covariance parameters?
- Standard approach: Maximum Likelihood (IID)
- Maximize probability of data given model (likelihood)

$$p(\mathcal{X} \mid \theta) = p(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N \mid \theta)$$
$$= \prod_{i=1}^{N} p(\vec{x}_i \mid \vec{\mu}_i, \Sigma_i) \quad \textit{independent Gaussian samples}$$
$$= \prod_{i=1}^{N} p(\vec{x}_i \mid \vec{\mu}, \Sigma) \quad \textit{identically distributed}$$

- Instead, work with maximum of log-likelihood

$$\sum_{i=1}^{N} \log p(\vec{x}_i \mid \vec{\mu}, \Sigma) = \sum_{i=1}^{N} \log \frac{1}{(2\pi)^{D/2}\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(\vec{x}_i-\vec{\mu})^T \Sigma^{-1} (\vec{x}_i-\vec{\mu})\right)$$

Max Likelihood Gaussian

- Max over μ

$$\frac{\partial}{\partial \mu}\left[\sum_{i=1}^{N} \log \frac{1}{(2\pi)^{D/2}\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(\vec{x}_i-\vec{\mu})^T \Sigma^{-1} (\vec{x}_i-\vec{\mu})\right)\right] = 0$$
$$\frac{\partial}{\partial \mu}\left[\sum_{i=1}^{N} -\frac{D}{2}\log 2\pi - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(\vec{x}_i-\vec{\mu})^T \Sigma^{-1} (\vec{x}_i-\vec{\mu})\right] = 0$$
$$\frac{\partial \vec{x}^T \vec{x}}{\partial \vec{x}} = 2\vec{x}^T \qquad \sum_{i=1}^{N} (\vec{x}_i-\vec{\mu})^T \Sigma^{-1} = \vec{0}$$

see Jordan Ch. 12, get sample mean… $\quad \vec{\mu} = \frac{1}{N}\sum_{i=1}^{N} \vec{x}_i$

- For Σ need Trace operator: $\quad tr(A) = tr(A^T) = \sum_{d=1}^{D} A(d,d)$

and several properties:

$$tr(AB) = tr(BA)$$
$$tr(BAB^{-1}) = tr(A)$$
$$tr(\vec{x}\vec{x}^T A) = tr(\vec{x}^T A \vec{x}) = \vec{x}^T A \vec{x}$$

Max Likelihood Gaussian

- Likelihood rewritten in trace notation:

$$l = \sum_{i=1}^{N} -\frac{D}{2}\log 2\pi - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(\vec{x}_i-\vec{\mu})^T \Sigma^{-1} (\vec{x}_i-\vec{\mu})$$
$$= -\frac{ND}{2}\log 2\pi + \frac{N}{2}\log|\Sigma^{-1}| - \frac{1}{2}\sum_{i=1}^{N} tr\left[(\vec{x}_i-\vec{\mu})^T \Sigma^{-1} (\vec{x}_i-\vec{\mu})\right]$$
$$= -\frac{ND}{2}\log 2\pi + \frac{N}{2}\log|\Sigma^{-1}| - \frac{1}{2}\sum_{i=1}^{N} tr\left[(\vec{x}_i-\vec{\mu})(\vec{x}_i-\vec{\mu})^T \Sigma^{-1}\right]$$
$$= -\frac{ND}{2}\log 2\pi + \frac{N}{2}\log|A| - \frac{1}{2}\sum_{i=1}^{N} tr\left[(\vec{x}_i-\vec{\mu})(\vec{x}_i-\vec{\mu})^T A\right]$$

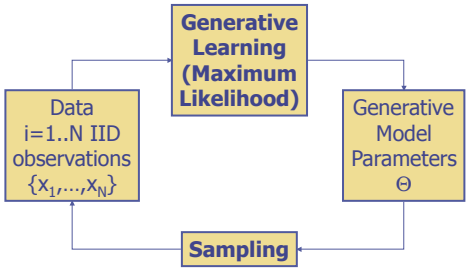
$$\frac{\partial \log|A|}{\partial A} = (A^{-1})^T \qquad \frac{\partial tr[BA]}{\partial A} = B^T$$

- Max over $A=\Sigma^{-1}$ use properties:

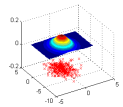
$$\frac{\partial l}{\partial A} = -0 + \frac{N}{2}(A^{-1})^T - \frac{1}{2}\sum_{i=1}^{N}\left[(\vec{x}_i-\vec{\mu})(\vec{x}_i-\vec{\mu})^T\right]^T$$
$$= \frac{N}{2}\Sigma - \frac{1}{2}\sum_{i=1}^{N}(\vec{x}_i-\vec{\mu})(\vec{x}_i-\vec{\mu})^T$$

- Get sample covariance: $\quad \frac{\partial l}{\partial A} = 0 \rightarrow \Sigma = \frac{1}{N}\sum_{i=1}^{N}(\vec{x}_i-\vec{\mu})(\vec{x}_i-\vec{\mu})^T$

Sampling & Max Likelihood



Sampling from a Gaussian



- Fit Gaussian to data, how is this Generative?
- Sampling! Generating discrete data easy:

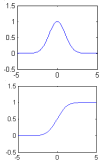
0.73	0.1	0.17

- Assume we can do uniform sampling:
  i.e. rand between (0,1)
  if 0.00 <= rand < 0.73 get A
  if 0.73 <= rand < 0.83 get B
  if 0.83 <= rand < 1.00 get C

0.73	0.83	1.00

- What are we doing?
  Sum up the Probability Density Function (PDF) to get Cumulative Density Function (CDF)
- For 1d Gaussian, Integrate Probability Density Function get Cumulative Density Function
  Integral is like summing many discrete bars

Sampling from a Gaussian



- Integrate 1d Gaussian to get CDF:
  $p(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right)$
  $F(x) = \int_{-\infty}^{x} p(t)\,dt = \frac{1}{2} erf\left(\frac{1}{\sqrt{2}}x\right) + \frac{1}{2}$
- If sample from uniform, get: $u \sim uniform(0,1)$
- Compute mapping: $x = F^{-1}(u) = \sqrt{2}\, erfinv(2u-1)$
- This is a Gaussian sample: $x \sim N(x \mid 0,1)$
- For D-dimensional Gaussian N(z|0,I) concatenate samples:
  $\vec{x} = \left[\vec{x}(1)\ldots\vec{x}(D)\right]^T \sim p(\vec{x} \mid 0, I) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\vec{x}(d)^2\right)$

- For N(z|μ,Σ), add mean & multiply by root cov
  $\vec{z} = \Sigma^{1/2}\vec{x} + \vec{\mu} \sim p(\vec{z} \mid \vec{\mu}, \Sigma)$

- Example code: gendata.m

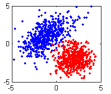
Machine Learning

Topic 10

- Classification with Gaussians
- Regression with Gaussians
- Principal Components Analysis

Classification with Gaussians

- Have two classes, each with their own Gaussian:



$\{(x_1,y_1),\ldots,(x_N,y_N)\}\quad x\in R^D\ \ y\in\{0,1\}$

- Given parameters $\theta=\{\alpha,\mu_0,\Sigma_0,\mu_1,\Sigma_1\}$ we can generate iid data from $p(x,y\mid\theta)=p(y\mid\theta)p(x\mid y,\theta)$ by:

1) flipping a coin to get y via Bernoulli $p(y\mid\theta)=\alpha^y(1-\alpha)^{1-y}$

2) sampling an x from y'th Gaussian $p(x\mid y,\theta)=N(x\mid\mu_y,\Sigma_y)$

- Or, recover parameters from data using maximum likelihood

$$\begin{aligned}l(\theta)&=\log p(data\mid\theta)=\textstyle\sum_{i=1}^N\log p(x_i,y_i\mid\theta)\\&=\textstyle\sum_{i=1}^N\log p(y_i\mid\theta)+\sum_{i=1}^N\log p(x_i\mid y_i,\theta)\\&=\textstyle\sum_{i=1}^N\log p(y_i\mid\alpha)+\sum_{y_i\in 0}\log p(x_i\mid\mu_0,\Sigma_0)+\sum_{y_i\in 1}\log p(x_i\mid\mu_1,\Sigma_1)\end{aligned}$$

Classification with Gaussians

- Max Likelihood can be done separately for the 3 terms

$$l=\textstyle\sum_{i=1}^N\log p(y_i\mid\alpha)+\sum_{y_i\in 0}\log p(x_i\mid\mu_0,\Sigma_0)+\sum_{y_i\in 1}\log p(x_i\mid\mu_1,\Sigma_1)$$

- Count # of pos & neg examples (class prior): $\alpha=\frac{N_1}{N_0+N_1}$
- Get mean & cov of negatives and mean & cov of positives:

$$\mu_0=\tfrac{1}{N_0}\textstyle\sum_{y_i\in 0}x_i\qquad\Sigma_0=\tfrac{1}{N_0}\sum_{y_i\in 0}(x_i-\mu_0)(x_i-\mu_0)^T$$
$$\mu_1=\tfrac{1}{N_1}\textstyle\sum_{y_i\in 1}x_i\qquad\Sigma_1=\tfrac{1}{N_1}\sum_{y_i\in 1}(x_i-\mu_1)(x_i-\mu_1)^T$$

- Given (x,y) pair, can now compute likelihood $p(x,y)$
- To make classification, a bit of Decision Theory
- Without x, can compute prior guess for y $p(y)$
- Give me x, want y, I need posterior $p(y\mid x)$
- Bayes Optimal Decision: $\hat{y}=\arg\max_{y=\{0,1\}}p(y\mid x)$
- Optimal iff we have true probability

Posterior gives Logistic

- Bayes Optimal Decision: $\hat{y}=\arg\max_{y=\{0,1\}}p(y\mid x)$
- To get conditional:

$$p(y\mid x)=\frac{p(x,y)}{p(x)}=\frac{p(x,y)}{\sum_y p(x,y)}=\frac{p(x,y)}{p(x,y=0)+p(x,y=1)}$$

- Check which is greater: $p(y=0\mid x)\geq ?\leq p(y=1\mid x)$
- Or check if this is > 0.5

$$\begin{aligned}p(y=1\mid x)&=\frac{p(x,y=1)}{p(x,y=0)+p(x,y=1)}\\&=\frac{1}{\frac{p(x,y=0)}{p(x,y=1)}+1}\\&=\frac{1}{\exp\left(-\log\frac{p(x,y=1)}{p(x,y=0)}\right)+1}\\&=sigmoid\left(\log\tfrac{p(x,y=1)}{p(x,y=0)}\right)\end{aligned}$$

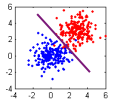
- Get logistic squashing function of log-ratio of probability models

Linear or Quadratic Decisions

- Example cases, plotting decision boundary when = 0.5

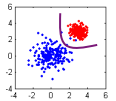
$$\begin{aligned}p(y=1\mid x)&=\frac{p(x,y=1)}{p(x,y=0)+p(x,y=1)}\\&=\frac{\alpha N(x\mid\mu_1,\Sigma_1)}{(1-\alpha)N(x\mid\mu_0,\Sigma_0)+\alpha N(x\mid\mu_1,\Sigma_1)}\end{aligned}$$

- If covariances are equal:



linear decision

- If covariances are different:



quadratic decision

Regression with Gaussians

- Have input and output, each Gaussian:

$\{(x_1,y_1),\ldots,(x_N,y_N)\}\quad x\in R^{D_x}\ \ y\in R^{D_y}$

concatenate $z_i=\begin{bmatrix}x_i\\y_i\end{bmatrix}$

$$p(z\mid\mu,\Sigma)=\frac{1}{(2\pi)^{D/2}\sqrt{|\Sigma|}}\exp\left(-\tfrac{1}{2}(z-\mu)^T\Sigma^{-1}(z-\mu)\right)$$

- Maximum Likelihood is as usual for a multivariate Gaussian

$$\mu=\tfrac{1}{N}\textstyle\sum_{i=1}^N z_i\qquad\Sigma=\tfrac{1}{N}\sum_{i=1}^N(z_i-\mu)(z_i-\mu)^T$$

- Bayes optimal decision: $\hat{y}=\arg\max_{y\in\mathbb{R}}p(y\mid x)$
- Or we can use: $\hat{y}=E_{p(y\mid x)}\{y\}$
- Have joint, need conditional: $p(y\mid x)=\frac{p(x,y)}{p(x)}=\frac{p(x,y)}{\int_y p(x,y)}$

Gaussian Marginals/Conditionals

- Conditional & marginal from joint: $p(y\mid x)=\frac{p(x,y)}{p(x)}=\frac{p(x,y)}{\int_y p(x,y)}$
- Conditioning the Gaussian:

$$p(z\mid\mu,\Sigma)=\frac{1}{(2\pi)^{D/2}\sqrt{|\Sigma|}}\exp\left(-\tfrac{1}{2}(z-\mu)^T\Sigma^{-1}(z-\mu)\right)$$

$$p(x,y)=\frac{1}{(2\pi)^{D/2}\sqrt{|\Sigma|}}\exp\left(-\tfrac{1}{2}\left(\begin{bmatrix}x\\y\end{bmatrix}-\begin{bmatrix}\mu_x\\\mu_y\end{bmatrix}\right)^T\begin{bmatrix}\Sigma_{xx}&\Sigma_{xy}\\\Sigma_{yx}&\Sigma_{yy}\end{bmatrix}^{-1}\left(\begin{bmatrix}x\\y\end{bmatrix}-\begin{bmatrix}\mu_x\\\mu_y\end{bmatrix}\right)\right)$$

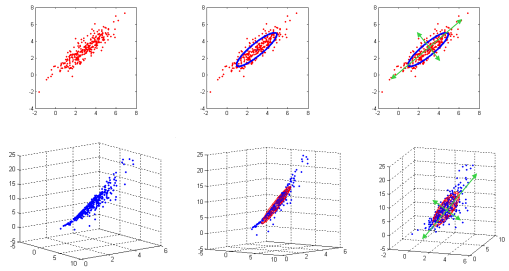
$$p(x)=\frac{1}{(2\pi)^{D_x/2}\sqrt{|\Sigma_{xx}|}}\exp\left(-\tfrac{1}{2}(x-\mu_x)^T\Sigma_{xx}^{-1}(x-\mu_x)\right)$$
$$=N(x\mid\mu_x,\Sigma_{xx})$$

$$p(y\mid x)=N(y\mid\mu_y+\Sigma_{yx}\Sigma_{xx}^{-1}(x-\mu_x),\Sigma_{yy}-\Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy})$$

- Here argmax is expectation which is conditional mean: $\hat{y}=\mu_y+\Sigma_{yx}\Sigma_{xx}^{-1}(x-\mu_x)$

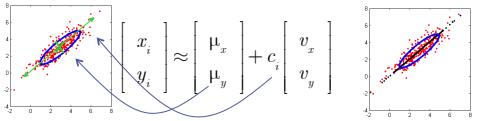
Principal Components Analysis

- Gaussians: for Classification, Regression… & Compression!
- Data can be constant in some directions, changes in others
- Use Gaussian to find directions of high/low variance



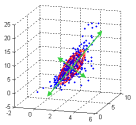
Principal Components Analysis

- Idea: instead of writing data in all its dimensions, only write it as mean + steps along one direction



$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} \approx \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} + c_i \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

- More generally, keep a subset of dimensions C from D (i.e. 2 of 3)



$$\vec{x}_i \approx \vec{\mu} + \sum_{j=1}^{C} c_{ij} \vec{v}_j$$

- Compression method: $\vec{x}_i \gg \vec{c}_i$
- Optimal directions: along eigenvectors of covariance
- Which directions to keep: highest eigenvalues (variances)

Principal Components Analysis

- If we have eigenvectors, mean and coefficients:

$$\vec{x}_i \approx \vec{\mu} + \sum_{j=1}^{C} c_{ij} \vec{v}_j$$

- Get eigenvectors (use eig() in Matlab): $\Sigma = V \Lambda V^T$

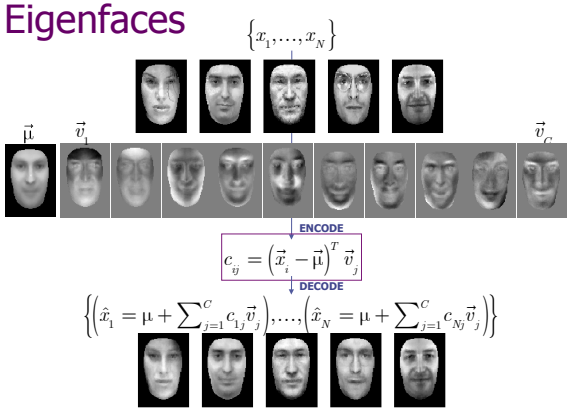
$$\begin{bmatrix} \Sigma(1,1) & \Sigma(1,2) & \Sigma(1,3) \\ \Sigma(1,2) & \Sigma(2,2) & \Sigma(2,3) \\ \Sigma(1,3) & \Sigma(2,3) & \Sigma(3,3) \end{bmatrix} = \begin{bmatrix} [\vec{v}_1] & [\vec{v}_2] & [\vec{v}_3] \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \begin{bmatrix} [\vec{v}_1] & [\vec{v}_2] & [\vec{v}_3] \end{bmatrix}^T$$

- Eigenvectors are orthonormal: $\vec{v}_i^T \vec{v}_j = \delta_{ij}$
- In coordinates of v, Gaussian is diagonal, cov = Λ
- All eigenvalues are non-negative $\lambda_i \geq 0$
- Higher eigenvalues are higher variance, use the top C ones

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4 \geq \ldots$$

- To compute the coefficients: $c_{ij} = \left(\vec{x}_i - \vec{\mu}\right)^T \vec{v}_j$

Eigenfaces



Machine Learning

Instructor: Tony Jebara

Topic 11

- Maximum Likelihood as Bayesian Inference
- Maximum A Posteriori
- Bayesian Gaussian Estimation

Why Maximum Likelihood?

- So far, assumed max (log) likelihood (IID or otherwise)
- Philosophical: Why? $\max_\theta L(\theta) = \max_\theta p(x_1,\ldots,x_N \mid \theta)$
$= \max_\theta \prod_{i=1}^N p(x_i \mid \theta)$

- Also, why ignore p(θ)?
- Hint: Recall Bayes rule:

likelihood, prior, posterior, evidence

$$p(\theta \mid x) = \frac{p(x \mid \theta)\,p(\theta)}{p(x)}$$

- Everyone agrees on probability theory: inference and use of probability models when we have computed p(x)
- But how get to p(x) from data? Debate…
- Two schools of thought: Bayesians and Frequentists

Bayesians & Frequentists

- Frequentists (Neymann/Pearson/Wald). An orthodox view that sampling is infinite and decision rules can be sharp.
- Bayesians (Bayes/Laplace/de Finetti). Unknown quantities are treated probabilistically and the state of the world can always be updated.

de Finetti: p(event) = price I would pay for a contract that pays 1$ when event happens

- Likelihoodists (Fisher). Single sample inference based on maximizing the likelihood function and relying on the Birnbaum's Theorem. Bayesians – But they don't know it.

Bayesians & Frequentists

- Frequentists:
  - Data are a repeatable random sample- there is a frequency
  - Underlying parameters remain constant during this repeatable process
  - Parameters are fixed
- Bayesians:
  - Data are observed from the realized sample.
  - Parameters are unknown and described probabilistically
  - Data are fixed

Bayesians & Frequentists

- Frequentists: classical / objective view / no priors
  every statistician should compute same p(x) so no priors
  can't have a p(event) if it never happened
  avoid p(θ), there is 1 true model, not distribution of them
  permitted: $p_\theta(x,y)$ forbidden: p(x,y|θ)
  Frequentist inference: estimate one best model θ
  use the ML estimator (unbiased & minimum variance)
  do not depend on Bayes rule for learning
- Bayesians: subjective view / priors are ok
  put a distribution or pdf on all variables in the problem
  even models & deterministic quantities (i.e. speed of light)
  use a prior p(θ), on the model θ before seeing any data
  Bayesian inference: use Bayes rule for learning, integrate over all model (θ) unknown variables

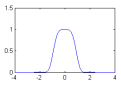
Bayesian Inference

- Bayes rule gives rise to maximum likelihood
- Assume we have a prior over models p(θ)

likelihood, prior, posterior, evidence

$$p(\theta \mid x) = \frac{p(x \mid \theta)\,p(\theta)}{p(x)}$$

- How to pick p(θ)?
  Pick simpler θ is better
  Pick form for mathematical convenience

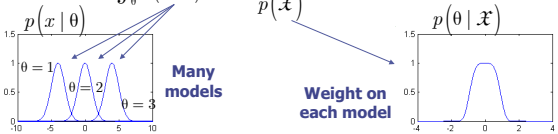


- We have data (can assume IID): $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$
- Want to get a model to compute: $p(x)$
- Want p(x) given our data… How to proceed?

Bayesian Inference

- Want p(x) given our data… $p(x \mid \mathcal{X}) = p(x \mid x_1, x_2, \ldots, x_n)$

$$p(x \mid \mathcal{X}) = \int_\theta p(x, \theta \mid \mathcal{X})\,d\theta$$
$$= \int_\theta p(x \mid \theta, \mathcal{X})\,p(\theta \mid \mathcal{X})\,d\theta \quad \text{Prior}$$
$$= \int_\theta p(x \mid \theta, \mathcal{X})\,\frac{p(\mathcal{X} \mid \theta)\,p(\theta)}{p(\mathcal{X})}\,d\theta$$
$$= \int_\theta p(x \mid \theta)\,\frac{\prod_{i=1}^N p(x_i \mid \theta)\,p(\theta)}{p(\mathcal{X})}\,d\theta$$

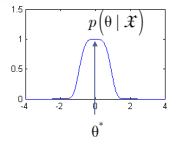


Bayesian Inference to MAP & ML

- The full Bayesian Inference integral can be mathematically tricky. Maximum likelihood is an approximation of it…

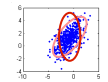
$$p(x \mid \mathcal{X}) = \int_\theta p(x \mid \theta)\,\frac{\prod_{i=1}^N p(x_i \mid \theta)\,p(\theta)}{p(\mathcal{X})}\,d\theta$$
$$\approx \int_\theta p(x \mid \theta)\,\delta(\theta - \theta^*)\,d\theta$$

$$\text{where } \theta^* = \begin{cases} \arg\max_\theta \frac{\prod_{i=1}^N p(x_i \mid \theta)\,p(\theta)}{p(\mathcal{X})} & MAP \\ \arg\max_\theta \frac{\prod_{i=1}^N p(x_i \mid \theta)\,uniform(\theta)}{p(\mathcal{X})} & ML \end{cases}$$



- Maximum A Posteriori (MAP) is like Maximum Likelihood (ML) with a prior p(θ) which lets us prefer some models over others

$$l_{MAP}(\theta) = l_{ML}(\theta) + \log p(\theta) = \sum_{i=1}^N \log p(x_i \mid \theta) + \log p(\theta)$$



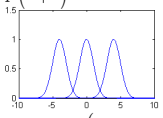
Bayesian Inference Example

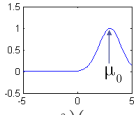
- For Gaussians, we CAN compute the integral (but hard!)

$$p(x \mid \mathcal{X}) = \int_\theta p(x \mid \theta)\frac{\prod_{i=1}^N p(x_i \mid \theta)p(\theta)}{p(\mathcal{X})}d\theta$$
$$\propto \int_\theta p(x \mid \theta)\prod_{i=1}^N p(x_i \mid \theta)p(\theta)d\theta$$

- Example:… assume 1d Gaussian & Gaussian prior on mean

$p(x \mid \theta) = Gaussian$ $\quad$ $p(\theta) = Gaussian$





$$p(x \mid \mathcal{X}) \propto \int_\mu \left(\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(x-\mu)^2}\right)\prod_{i=1}^N \left(\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(x_i-\mu)^2}\right)\left(\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(\mu-\mu_0)^2}\right)d\mu$$

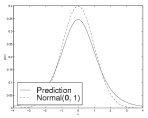
Bayesian Inference Example

- Solve integral over all Gaussian means with variance=1

$$p(x \mid \mathcal{X}) \propto \int_{\mu=-\infty}^{\mu=\infty}\left(\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(x-\mu)^2}\right)\prod_{i=1}^N\left(\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(x_i-\mu)^2}\right)\left(\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(\mu_0-\mu)^2}\right)d\mu$$
$$\propto \int_{\mu=-\infty}^{\mu=\infty}\exp\left(-\frac{1}{2}(x-\mu)^2-\sum_i\frac{1}{2}(x_i-\mu)^2-\frac{1}{2}(\mu_0-\mu)^2\right)d\mu$$
$$\propto \int_{\mu=-\infty}^{\mu=\infty}\exp\left(-\frac{1}{2}\left[(N+2)\mu^2-2\mu\left(x+\mu_0+\sum_i x_i\right)+x^2\right]\right)d\mu$$
$$\propto \int_{\mu=-\infty}^{\mu=\infty}\exp\left(-\frac{1}{2}\left[(N+2)\mu^2-2\mu\left(x+\mu_0+\sum_i x_i\right)+x^2\right]+[\ ]^2-[\ ]^2\right)d\mu$$
$$\propto \exp\left(-\frac{1}{2}\left[\frac{-\left(x+\mu_0+\sum_i x_i\right)^2}{N+2}+x^2\right]\right) \qquad \tilde{\mu} = \frac{\mu_0+\sum_i x_i}{N+1}$$
$$= N\left(x \mid \tilde{\mu}, \tilde{\sigma}^2\right) \qquad \tilde{\sigma}^2 = \frac{N+2}{N+1}$$

- Can integrate over μ and Σ for multivariate Gaussian (Jordan ch. 4 and Minka Tutorial)

$$p(x \mid \mathcal{X}) = \frac{\Gamma\left[(N+1)/2\right]}{\Gamma\left[(N+1-d)/2\right]}\left|\frac{1}{(N+1)\pi}\bar{\Sigma}^{-1}\right|^{1/2}\left(\frac{1}{N+1}\left(x-\bar{\mu}\right)^T\bar{\Sigma}^{-1}\left(x-\bar{\mu}\right)+1\right)^{-(N+1)/2}$$



Machine Learning

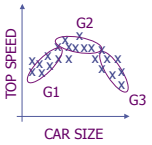
Instructor: Tony Jebara

Topic 12

- Mixture Models and Hidden Variables
- Clustering
- K-Means
- Expectation Maximization

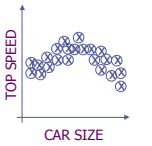
Mixtures for More Flexibility

- With mixtures (e.g. mixtures of Gaussians) we can handle more complicated (e.g. multi-bump, nonlinear) distributions.



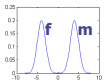
subpopulations: G1=compact car
G2=mid-size car
G3=cadillac

- In fact, if we have enough Gaussians (maybe infinite) we can approximate any distribution...

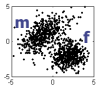


Mixtures as Hidden Variables

- Consider a dataset with K subpopulations but don't know which subpopulation each point belongs to



I.e. looking at height of adult people, we see K=2 subpopulations: males & females



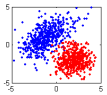
I.e. looking at weight and height of people we see K=2 subpopulations: males & females

- Because of the 'hidden' variable (y can be 1 or 2), these distributions are not Gaussians but Mixture of Gaussians

$$p(\vec{x}) = \sum_y p(\vec{x}, y) = \sum_y p(y)\, p(\vec{x} \mid y) = \sum_y \pi_y N(\vec{x} \mid \vec{\mu}_y, \Sigma_y)$$
$$= \sum_{y=1}^{K} \pi_y \frac{1}{(2\pi)^{D/2}\sqrt{|\Sigma_y|}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu}_y)^T \Sigma_y^{-1} (\vec{x}-\vec{\mu}_y)\right)$$

Unlabeled data → Clustering

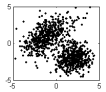
- Recall classification problem:



maximize the log-likelihood of data given models:

$$l = \sum_{n=1}^{N} \log p(\vec{x}_n, y_n \mid \pi, \mu, \Sigma)$$
$$= \sum_{n=1}^{N} \log \pi_{y_n} N(\vec{x}_n \mid \vec{\mu}_{y_n}, \Sigma_{y_n})$$

- If we don't know the class treat it as a hidden variable

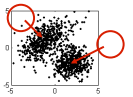


maximize the log-likelihood with unlabeled data:

$$l = \sum_{n=1}^{N} \log p(\vec{x}_n \mid \pi, \mu, \Sigma) = \sum_{n=1}^{N} \log \sum_{y=1}^{K} p(\vec{x}_n, y \mid \pi, \mu, \Sigma)$$
$$= \sum_{n=1}^{N} \log\left(\pi_1 N(\vec{x}_n \mid \vec{\mu}_1, \Sigma_1) + \ldots + \pi_K N(\vec{x}_n \mid \vec{\mu}_K, \Sigma_K)\right)$$

- Instead of classification, we now have a clustering problem

K-Means Clustering

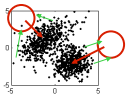


- K-means solves a Chicken-and-Egg problem:
  If knew classes, we can get model (max likelihood!)
  If knew the model, we can predict the classes (classifier!)
- Kmeans: guess a model, use it to classify the data, use classified data as labeled data to update the model, repeat.

- Assumes each point x has a discrete multinomial vector z

0) Input dataset $\{\vec{x}_1, \ldots, \vec{x}_N\}$
1) Randomly initialize means $\vec{\mu}_1, \ldots, \vec{\mu}_K$
2) Find closest mean for each point $\vec{z}_n(i) = \begin{cases} 1 & \text{if } i = \arg\min_j \|\vec{x}_n - \vec{\mu}_j\|^2 \\ 0 & \text{otherwise} \end{cases}$
3) Update means $\vec{\mu}_i = \sum_{n=1}^{N} \vec{x}_n \vec{z}_n(i) / \sum_{n=1}^{N} \vec{z}_n(i)$
4) If any z has changed go to 2

K-Means Clustering

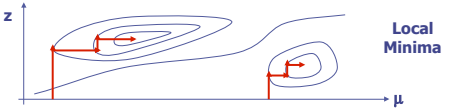


- Geometric, each point goes to closest Gaussian
- Recompute the means by their assigned points
- Essentially minimizing the following cost function:

$$\min_{\mu} \min_{z} J(\vec{\mu}_1, \ldots, \vec{\mu}_K, \vec{z}_1, \ldots, \vec{z}_N) = \sum_{n=1}^{N}\sum_{i=1}^{K} \vec{z}_n(i) \|\vec{x}_n - \vec{\mu}_i\|^2$$

$$\vec{z}_n(i) = \begin{cases} 1 & \text{if } i = \arg\min_j \|\vec{x}_n - \vec{\mu}_j\|^2 \\ 0 & \text{otherwise} \end{cases} \qquad \vec{\mu}_i = \frac{\sum_{n=1}^{N} \vec{x}_n \vec{z}_n(i)}{\sum_{n=1}^{N} \vec{z}_n(i)}$$

- Guaranteed to improve per iteration and converge
- Like Coordinate Descent (lock one var, maximize the other)
- A.k.a. Axis-Parallel Optimization or Alternating Minimization

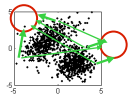


Expectation-Maximization (EM)

- EM is a soft/fuzzy version of K-Means (which does winner-takes-all, closest Gaussian Mean completely wins datapoint)

$$\vec{z}_n(i) = \begin{cases} 1 & \text{if } i = \arg\min_j \|\vec{x}_n - \vec{\mu}_j\|^2 = \arg\max_j N(\vec{x}_n \mid \vec{\mu}_j, I) = \arg\max_j p(\vec{x}_n \mid \vec{\mu}_j) \\ 0 & \text{otherwise} \end{cases}$$

- Instead, consider soft percentage assignment of datapoint



$$assign \propto \pi_j \frac{1}{(2\pi)^{D/2}} \exp\left(-\frac{1}{2}\|\vec{x}_n - \vec{\mu}_j\|^2\right)$$

- EM is 'less greedy' than K-Means uses $\tau_{n,i} = p(\vec{z} = \vec{\delta}_i \mid \vec{x}_n, \theta)$ as shared responsibility for $\vec{x}_n$

$$\tau_{n,1}, \ldots, \tau_{n,K} =$$

- Update for the means are then 'weighted' by responsibilities:

$$\mu_i = \frac{\sum_{n=1}^{N} \tau_{n,i} \vec{x}_n}{\sum_{n=1}^{N} \tau_{n,i}}$$

Expectation-Maximization

- EM uses expected value of $\vec{z}_n(i)$ rather than max

$$\tau_{n,i} = E\{\vec{z}_n(i) \mid \vec{x}_n\} = p(\vec{z}_n = \vec{\delta}_i \mid \vec{x}_n, \theta)$$

- EM updates covariances, mixing proportions AND means…
- The algorithm for Gaussian mixtures:

EXPECTATION: $\tau_{n,i}^{(t)} = \frac{\pi_i^{(t)} N(\vec{x}_n \mid \vec{\mu}_i^{(t)}, \Sigma_i^{(t)})}{\sum_j \pi_j^{(t)} N(\vec{x}_n \mid \vec{\mu}_j^{(t)}, \Sigma_j^{(t)})}$

MAXIMIZATION: $\vec{\mu}_i^{(t+1)} = \frac{\sum_n \tau_{n,i}^{(t)} \vec{x}_n}{\sum_n \tau_{n,i}^{(t)}} \qquad \pi_i^{(t+1)} = \frac{\sum_n \tau_{n,i}^{(t)}}{N}$

$$\Sigma_i^{(t+1)} = \frac{\sum_n \tau_{n,i}^{(t)} (\vec{x}_n - \vec{\mu}_i^{(t+1)})(\vec{x}_n - \vec{\mu}_i^{(t+1)})^T}{\sum_n \tau_{n,i}^{(t)}}$$

- DEMO… like an iterative divide-and-conquer algorithm
- But, divide&conquer is not a guarantee. Can we prove EM?

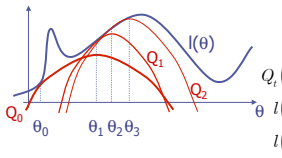
Machine Learning

Topic 13

- Expectation Maximization as Bound Maximization
- EM for Maximum A Posteriori

EM as Bound Maximization

- Let's now show that EM indeed maximizes likelihood
- Bound Maximization: optimize a lower bound on l(θ)
- Since log-likelihood l(θ) not concave, can't max it directly
- Consider an auxiliary function Q(θ) which is concave
- Q(θ) kisses l(θ) at a point and is less than it elsewhere



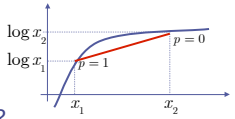
$$l(\theta) \geq Q_t(\theta) \quad \forall\theta \ \forall t$$
$$l(\theta_t) = Q_t(\theta_t)$$
$$Q_t(\theta_{t+1}) > Q_t(\theta_t) \qquad \theta_{t+1} = \arg\max_\theta Q_t(\theta_t)$$
$$l(\theta_{t+1}) \geq Q_t(\theta_{t+1}) > Q_t(\theta_t)$$
$$l(\theta_{t+1}) > l(\theta_t)$$

- Monotonically increases log-likelihood
- But how to find a bound and guarantee we max it?

Jensen's Inequality

- An important general bound from Jensen (1906)
- For convex f: $f\left(E\left\{x\right\}\right) \leq E\left\{f\left(x\right)\right\}$
- For concave f: $f\left(E\left\{x\right\}\right) \geq E\left\{f\left(x\right)\right\}$
- Expectation in discrete case is sum weight by probability
- For convex f: $f\left(\sum_{i=1}^{M} p_i x_i\right) \leq \sum_{i=1}^{M} p_i f\left(x_i\right) \ when \ \sum_{i=1}^{M} p_i = 1, \ p_i \geq 0$
- For concave f: $f\left(\sum_{i=1}^{M} p_i x_i\right) \geq \sum_{i=1}^{M} p_i f\left(x_i\right) \ when \ \sum_{i=1}^{M} p_i = 1, \ p_i \geq 0$
- Example: f(x)=log(x)=concave and M=2

$$\log\left(p x_1 + (1-p) x_2\right) \geq p \log x_1 + (1-p) \log x_2$$



- Bound log(sum) with sum(log)
- How to apply this to mixture models?

Expectation-Maximization

$$l(\theta) = \sum_{n=1}^{N} \log p\left(x_n \mid \theta\right) \qquad \text{Original Log-Likelihood}$$
$$= \sum_{n=1}^{N} \log \sum_z p\left(x_n, z \mid \theta\right) \qquad \text{Has Hidden Variables (messy)}$$
$$= \sum_{n=1}^{N} \log \sum_z p\left(x_n, z \mid \theta\right) \frac{p\left(z \mid x_n, \theta_t\right)}{p\left(z \mid x_n, \theta_t\right)} \qquad \text{Multiply by 1 Ratio of hidden posterior density}$$
$$= \sum_{n=1}^{N} \log \sum_z p\left(z \mid x_n, \theta_t\right) \frac{p\left(x_n, z \mid \theta\right)}{p\left(z \mid x_n, \theta_t\right)} \qquad \text{Rearrange}$$
$$\geq \sum_{n=1}^{N} \sum_z p\left(z \mid x_n, \theta_t\right) \log \frac{p\left(x_n, z \mid \theta\right)}{p\left(z \mid x_n, \theta_t\right)} \qquad \text{Jensen log(}\Sigma_i \, p_i \, x_i \text{)}$$
$$= \sum_{n=1}^{N} \sum_z p\left(z \mid x_n, \theta_t\right) \log p\left(x_n, z \mid \theta\right) - \sum_{n=1}^{N} \sum_z p\left(z \mid x_n, \theta_t\right) \log p\left(z \mid x_n, \theta_t\right) \qquad \text{New auxiliary function called Q (not messy)}$$
$$= Q\left(\theta \mid \theta_t\right) - const$$

EM as Bound Maximization

- Now have the following bound and maximize it:

$$l(\theta) \geq Q(\theta \mid \theta_t) - \sum_{n=1}^{N} \sum_z p\left(z \mid x_n, \theta_t\right) \log p\left(z \mid x_n, \theta_t\right)$$
$$\theta^{t+1} = \arg\max_\theta Q\left(\theta \mid \theta_t\right) = \arg\max_\theta \sum_{n=1}^{N} \sum_z p\left(z \mid x_n, \theta_t\right) \log p\left(x_n, z \mid \theta\right)$$
$$= \arg\max_\theta \sum_{n=1}^{N} \sum_z \tau_{n,z} \log p\left(x_n, z \mid \theta\right)$$

- $Q(\theta|\theta_t)$ is called Auxiliary Function… take derivatives of it
- This is easy for e-families… just weighted max likelihood!
- For example, Gaussian mixture:

$$\frac{\partial Q\left(\theta\right)}{\partial \vec{\mu}_k} = \frac{\partial}{\partial \vec{\mu}_k} \sum_{n=1}^{N} \sum_k \tau_{n,k} \log \pi_k N\left(\vec{x}_n \mid \vec{\mu}_k, \Sigma_k\right)$$
$$0 = \sum_{n=1}^{N} \tau_{n,k} \frac{\partial}{\partial \vec{\mu}_k} \left(-\frac{1}{2}\left(\vec{x}_n - \vec{\mu}_k\right)^T \Sigma_k^{-1} \left(\vec{x}_n - \vec{\mu}_k\right)\right)$$
$$\vec{\mu}_k = \frac{\sum_{n=1}^{N} \tau_{n,k} \vec{x}_n}{\sum_{n=1}^{N} \tau_{n,k}}$$

… similarly get π_k and Σ_k

EM as Expected Log-Likelihood

- Incomplete Log-Likelihood $l(\theta) = \log p\left(observed \mid \theta\right) = \sum_{n=1}^{N} \log \sum_z p\left(x_n, z \mid \theta\right)$
- Complete Log-Likelihood $l^C(\theta) = \log p\left(observed, hidden \mid \theta\right) = \sum_{n=1}^{N} \log p\left(x_n, z_n \mid \theta\right)$
- We don't know the hidden variables z
- EM computes expected values of hidden z under current θ_t
- EM chooses Q to be the Expected Complete Log-Likelihood

$$\begin{aligned} E\left\{l^C(\theta)\right\} &= \sum_{hidden} p\left(hidden \mid observed, \theta_t\right) l^C(\theta) \\ &= \sum_{z_1} \cdots \sum_{z_N} p\left(z_1, \ldots, z_n \mid x_1, \ldots, x_n, \theta_t\right) l^C(\theta) \\ &= \sum_{z_1} \cdots \sum_{z_N} \prod_n p\left(z_n \mid x_n, \theta_t\right) l^C(\theta) \\ &= \sum_{z_1} \cdots \sum_{z_N} \prod_n p\left(z_n \mid x_n, \theta_t\right) \sum_n \log p\left(x_n, z_n \mid \theta\right) \\ &= \sum_n \sum_{z_n} p\left(z_n \mid x_n, \theta_t\right) \log p\left(x_n, z_n \mid \theta\right) \sum_{z_1} \cdots \sum_{z_{n-1}} \sum_{z_{n+1}} \cdots \sum_{z_N} \prod_{i \neq n} p\left(z_i \mid x_i, \theta_t\right) \\ &= \sum_n \sum_{z_n} p\left(z_n \mid x_n, \theta_t\right) \log p\left(x_n, z_n \mid \theta\right) = Q\left(\theta \mid \theta_t\right) \end{aligned}$$

EM for Max A Posteriori

- We can also do MAP instead of ML with EM (stabilizes sol'n)

$$\log posterior\left(\theta\right) = \sum_{n=1}^{N} \log \sum_z p\left(x_n, z \mid \theta\right) + \log p\left(\theta\right)$$

- Prior doesn't have log-sum
- The E-step remains the same: lower bound log-sum

$$\log posterior\left(\theta\right) = l\left(\theta\right) + \log p\left(\theta\right) \geq E\left\{l^C\left(\theta\right)\right\} + const + \log p\left(\theta\right)$$

- The M-step becomes slightly different for each model
- For example, mixture of Gaussians with prior on covariance

$$\log posterior\left(\theta\right) = \sum_{n=1}^{N} \log \sum_k \pi_k N\left(\vec{x}_n \mid \vec{\mu}_k, \Sigma_k\right) + \log \prod_k p\left(\Sigma_k \mid S, \eta\right)$$
$$\log posterior\left(\theta\right) \geq \sum_{n=1}^{N} \sum_k \tau_{n,k} \log \pi_k N\left(\vec{x}_n \mid \vec{\mu}_k, \Sigma_k\right) + \sum_k \log p\left(\Sigma_k \mid S, \eta\right) + const$$

- Updates on π and μ stay the same, only Σ is:

$$\Sigma_k \leftarrow \frac{1}{\sum_{n=1}^{N} \tau_{n,k} + \eta} \left(\sum_{n=1}^{N} \tau_{n,k} \left(\vec{x}_n - \vec{\mu}_k\right)\left(\vec{x}_n - \vec{\mu}_k\right)^T + \eta S\right)$$

- Typically, we use the identity matrix I for S and a small eta.