Module 5: Classification

Mixture Models for Classification

STAT/BIOSTAT 527, University of Washington
Emily Fox
May 30th, 2013

Overview of Classification So Far

- Supervised methods

  Generative Discriminative

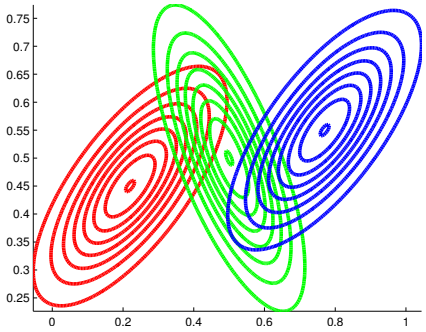
- Objectives:

- Unsupervised methods (generative)

Density as Mixture of Gaussians

- Approximate density with a mixture of Gaussians

Mixture of 3 Gaussians



K = 3

$$p(x_i \mid \pi, \mu, \Sigma) = \sum_{k=1}^{K} \pi_k \, N(x_i \mid \mu_k, \Sigma_k)$$

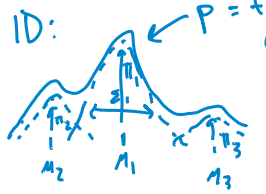
Gauss. kernel, just like in KDE, but not centered at obs.

# of mix comp.

mix. weights

shape params

In 1D:

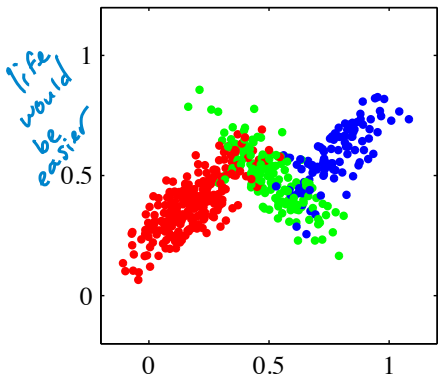


$\Sigma \pi_k = 1$

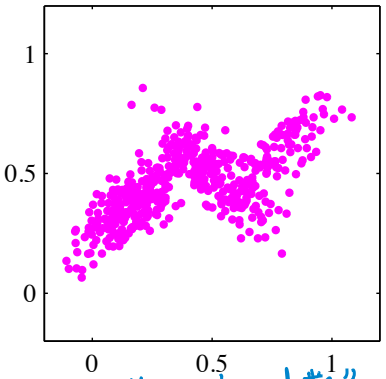
Clustering our Observations

- Imagine we have an assignment of each x_i to a Gaussian

Our actual observations



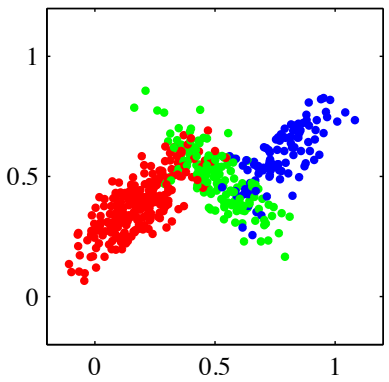
Complete data labeled by true cluster assignments



C. Bishop, Pattern Recognition & Machine Learning

Clustering our Observations

- Imagine we have an assignment of each x_i to a Gaussian



Complete data labeled by true cluster assignments

- Introduce latent cluster indicator variable z_i

$z_i \in \{1, \ldots, K\}$ ← think of as class indicator y_i, but no training data labels

$\Pr(z_i = k) = \pi_k$

- Then we have

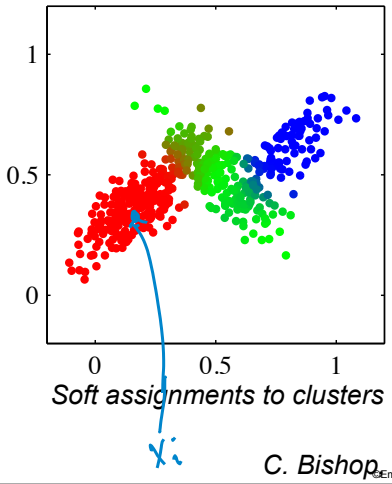
$$p(x_i \mid z_i = k, \pi, \mu, \Sigma) = N(x_i \mid \mu_{z_i = k}, \Sigma_{z_i = k})$$

Param. est. is easy if we have $\{z_i\}$ ⇒ decoupled into K Gauss. est.

C. Bishop, Pattern Recognition & Machine Learning

Clustering our Observations

- We must infer the cluster assignments from the observations



Soft assignments to clusters

- Posterior probabilities of assignments to each cluster *given* model parameters:

"responsibilities"

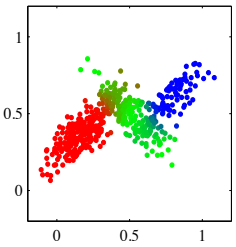
$$r_{ik} = p(z_i = k \mid x_i, \pi, \theta) = \frac{\pi_k N(x_i \mid \mu_k, \Sigma_k)}{\sum_j \pi_j N(x_i \mid \mu_j, \Sigma_j)}$$

motivates an iterative alg.

C. Bishop, Pattern Recognition & Machine Learning

Mixture Models for Classification

- Can use mixture models as a generative classifier in the unsupervised setting



- EM algorithm = iteratively:
  - Estimate responsibilities given parameter estimates

$$\hat{r}_{ik} = \frac{\hat{\pi}_k N(x_i, \hat{\mu}_k, \hat{\Sigma}_k)}{\sum_\ell \hat{\pi}_\ell N(x_i, \hat{\mu}_\ell, \hat{\Sigma}_\ell)}$$

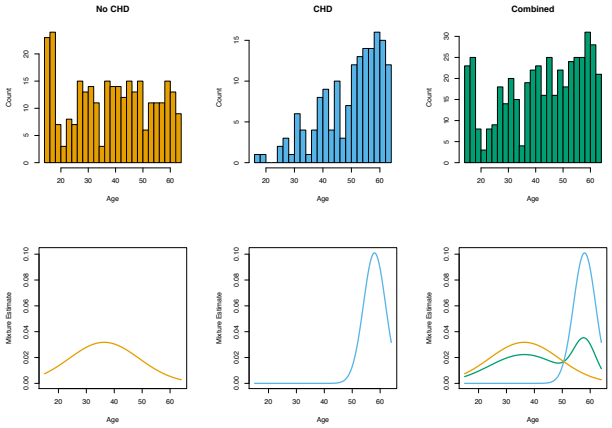
  - Maximize parameters given responsibilities

- For classification, threshold the estimated responsibilities
  - E.g., $\hat{g}(x_i) = \arg\max_k \hat{r}_{ik}$

- Note: allows non-linear boundaries as in QDA

Example: Heart Disease Data

- Binary response = CHD (coronary heart disease)
- Predictor = systolic blood pressure



From Hastie, Tibshirani, Friedman book

What you need to know

- Discriminative vs. Generative classifiers
- LDA and QDA assume Gaussian class-conditional densities
  - Results in linear and quadratic decision boundaries, respectively
- KDE for classification
  - Challenging in areas with little data or in high dimensions
  - Estimating class-conditionals is not optimizing classification objective
- Naïve Bayes assumes factored form
  - Results in log odds that have GAM form
- Mixture models allow for unsupervised generative approach

Readings

- Hastie, Tibshirani, Friedman – 4.3, 4.4.5, 6.6.2-6.6.3, 6.8

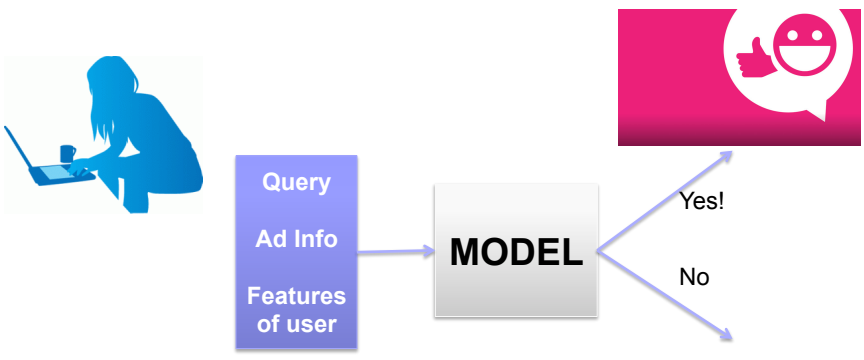
Module 5: Classification

Online Learning Perceptron Algorithm

STAT/BIOSTAT 527, University of Washington
Emily Fox
May 28th, 2013

Estimating Click Probabilities

- **Goal:** Predict whether a person clicks on an ad
- **Basic approach:** Logistic regression



Challenge 1: Complexity of Computing Gradients

in terms of n, d

- What's the cost of a gradient update step for LR???

$$\beta_j^{(t+1)} \leftarrow \beta_j^{(t)} + \eta \left\{ -\lambda \beta_j^{(t)} + \sum_i x_{ij} \left(y_i - \hat{p}(y = 1 \mid x_i, \beta^{(t)}) \right) \right\}$$

for each j

O(d)

O(nd)

Naively, $O(nd^2)$

but cache $\hat{p}$ (same $\forall j$) $\rightarrow O(nd)$

However, if n is huge (or streaming), this is very slow (infeasible) per little gradient step

Challenge 2: Data is streaming

- Assumption thus far: **Batch data**
- But, e.g., click prediction for ads is a streaming data task:
  - User enters query, and ad must be selected:
    - Observe $\mathbf{x}_i$, and must predict y_i
  - User either clicks or doesn't click on ad:
    - Label y_i is revealed afterwards
      - Google gets a reward if user clicks on ad
  - Weights must be updated for next time:

Online Learning Problem

- At each time step t:
  - Observe features (covariates) of data point:
    - Note: many assumptions are possible, e.g., data is iid, data is adversarially chosen… details beyond scope of course
  - Make a prediction:
    - Note: many models are possible, we focus on linear models
  - Observe true label:
    - Note: other observation models are possible, e.g., we don't observe the label directly, but only a noisy version... Details beyond scope of course
  - Update model:

The Perceptron Algorithm [Rosenblatt '58, '62]

- Classification setting: y in {-1,+1}
- Linear model
  - Prediction:
- Training:
  - Initialize weight vector:
  - At each time step:
    - Observe covariates:
    - Make prediction:
    - Observe true class:
    - Update model:
      - If prediction is not equal to truth

Intuition

If $\hat{y} = y_t$,

$\beta^{(t+1)} \leftarrow \beta^{(t)}$

else

$\beta^{(t+1)} \leftarrow \beta^{(t)} + y_t x_t$

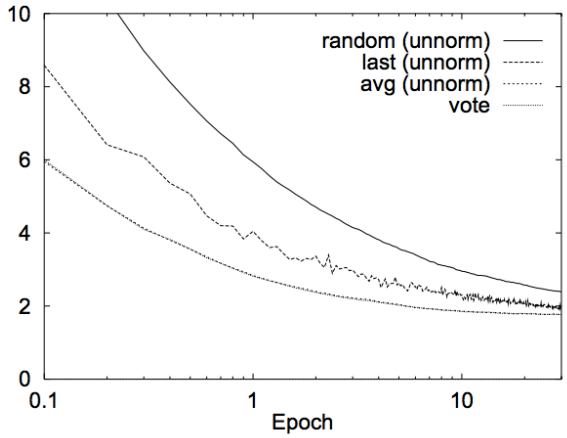
$$\hat{y} = \text{sign}(\beta^{(t)} \cdot x_t)$$

- Why is this a reasonable update rule?

Which weight vector to report?

- Practical problem for all online learning methods
- Suppose you run online learning method and want to sell your learned weight vector… Which one do you sell???
- Last one?
- Random
- Average
- Voting + more advanced

Choice can make a huge difference!!



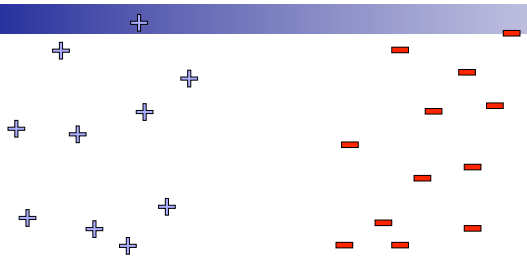
[Freund & Schapire '99]

Mistake Bounds

- Algorithm "pays" every time it makes a mistake:

- How many mistakes is it going to make?

Linear Separability: More formally, Using Margin



- Data linearly separable, if there exists
  - a vector
  - a margin
- Such that

Perceptron Analysis: Linearly Separable Case

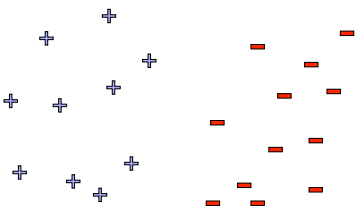
- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples:
  - Each covariate vector has bounded norm:
  - If dataset is linearly separable:
- Then the number of mistakes made by the online perceptron on this sequence is bounded by

Perceptron Proof for Linearly Separable case

- Every time we make a mistake, we get γ closer to β^*:
  - Mistake at time t: $\beta^{(t+1)} = \beta^{(t)} + y_t x_t$
  - Taking dot product with β^*:
  - Thus after m mistakes:
- Similarly, norm of $\beta^{(t+1)}$ doesn't grow too fast:
  - $||\beta^{(t+1)}||^2 = ||\beta^{(t)}||^2 + 2y_t(\beta^{(t)} \cdot x_t) + ||x_t||^2$
  - Thus, after m mistakes:
- Putting all together:

Beyond Linearly Separable Case

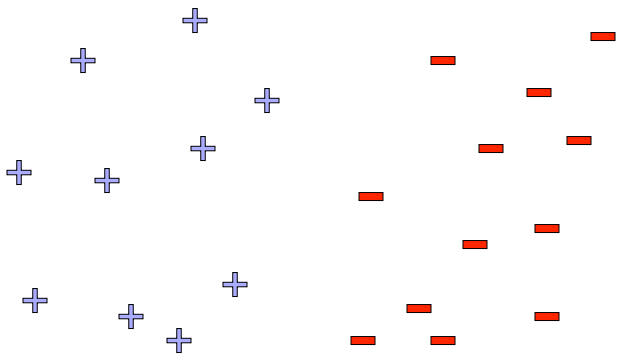
- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an oblivious adversary, no need to be iid
  - Makes a fixed number of mistakes, and it's done for ever!
    - Even if you see infinite data
- However, real world not linearly separable
  - Can't expect never to make mistakes again
  - Analysis extends to non-linearly separable case
  - Very similar bound, see Freund & Schapire
  - Converges, but ultimately may not give good accuracy (make many many many mistakes)



What is the Perceptron Doing???

- When we discussed logistic regression:
  - Started from maximizing conditional log-likelihood
- When we discussed the perceptron:
  - Started from description of an algorithm
- What is the perceptron optimizing????

Perceptron Prediction: Margin of Confidence



Hinge Loss

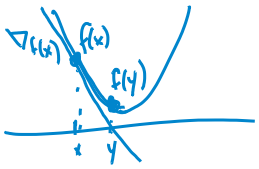
- Perceptron prediction:
- Makes a mistake when:
- Hinge loss (same as maximizing the margin used by SVMs)

Minimizing Hinge Loss in Batch Setting

- Given a dataset:
- Minimize average hinge loss:
- How do we compute the gradient?

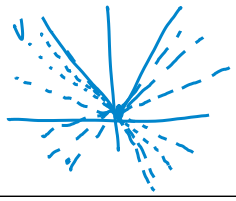
Subgradients of Convex Functions

- Gradients lower bound convex functions:



$f(y) \geq f(x) + \nabla f(x)(y - x)$

- Gradients are unique at **x** if function differentiable at **x**
- Subgradients: Generalize gradients to non-differentiable points:
  - Any plane that lower bounds function:



For $|\beta_j|$: $v \in [-1, 1]$

$v \in \partial f(x)$ subgradient if $f(y) \geq f(x) + v(y - x)$

Subgradient of Hinge

- Hinge loss:

- Subgradient of hinge loss:
  - If $y_t(\beta.\mathbf{x}_t) > 0$:
  - If $y_t(\beta.\mathbf{x}_t) < 0$:
  - If $y_t(\beta.\mathbf{x}_t) = 0$:
  - In one line:

Subgradient Descent for Hinge Minimization

- Given data: $(x_1, y_1), \ldots, (x_n, y_n)$
- Want to minimize:

$$\frac{1}{n}\sum_{i=1}^{n} \ell(\beta, x_i) = \frac{1}{n}\sum_{i=1}^{n} (-y_i(\beta \cdot x_i))_+$$

- Subgradient descent works the same as gradient descent:
  - But if there are multiple subgradients at a point, just pick (any) one:

Perceptron Revisited

- Perceptron update:

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \mathbb{I}\left[y_t(\beta^{(t)} \cdot x_t) \leq 0\right] y_t x_t$$

- Batch hinge minimization update:

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \eta \frac{1}{n}\sum_{i=1}^{n} \left\{\mathbb{I}\left[y_i(\beta^{(t)} \cdot x_i) \leq 0\right] y_i x_i\right\}$$

- Difference?

What you need to know

- Notion of online learning
- Perceptron algorithm
- Mistake bounds and proof
- In online learning, report averaged weights at the end
- Perceptron is optimizing hinge loss
- Subgradients and hinge loss
- (Sub)gradient decent for hinge objective

Module 5: Classification

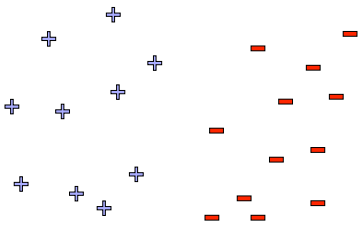
Kernels, again!

STAT/BIOSTAT 527, University of Washington

Emily Fox

May 30th, 2013

What if the data are not linearly separable?



Use features of features of features of features….

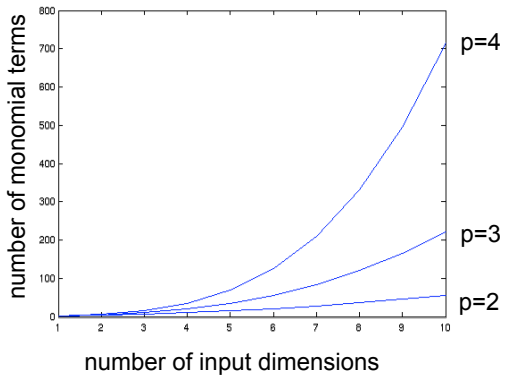
$$\phi(x) : \mathbb{R}^d \rightarrow F$$

Feature space can get really large really quickly!

Higher Order Polynomials

$$\#\text{ terms} = \binom{p+d-1}{p} = \frac{(p+d-1)!}{p!(d-1)!}$$

d – covariate dimension
p – degree of polynomial



grows fast!
p = 6, d = 100
about 1.6 billion terms

Perceptron Revisited

- Given weight vector $\beta^{(t)}$, predict point **x** by:
- Mistake at time *t*: $\beta^{(t+1)} \leftarrow \beta^{(t)} + y_t x_t$
- Thus, write weight vector in terms of mistaken data points only:
  - Let $M^{(t)}$ be time steps up to *t* when mistakes were made:
- Prediction rule now:
- When using high dimensional features:

Dot-Product of Polynomials

$\phi(u) \cdot \phi(v) =$ polynomials of degree exactly *p*

The Kernel Trick Again: Kernelized Perceptron

- Every time you make a mistake, remember (x_t, y_t)
- Kernelized perceptron prediction for **x**:

$$\text{sign}(\beta^{(t)} \cdot \phi(x)) = \sum_{i \in M^{(t)}} y_i (\phi(x_i) \cdot \phi(x))$$
$$= \sum_{i \in M^{(t)}} y_i k(x_i, x)$$

Going Infinite…

Change of notation:
$h_j(x) \rightarrow \phi_j(x)$

- Nonparametric Gaussian regression: Would like to let the number of "features" *M* → ∞ (basis fcns)
- *Prior:* $p(\beta \mid 0, \alpha^{-1} I_M)$
- ~~*Predictions:*~~ $f = \Phi\beta$ (Dist. on f; $(f(x_1), \dots, f(x_n))^T$; $(\beta_1, \dots, \beta_n)^T$; linear comb. of Gaussians β_j; $\Rightarrow$ f Gauss.; $E[f] = \Phi E[\beta] = 0$; $\text{cov}(f) = \Phi E[\beta\beta^T]\Phi^T = \frac{1}{\alpha}\Phi\Phi^T$; $p(f) = N(f \mid 0, \alpha^{-1}\Phi\Phi^T)$; $\begin{bmatrix} \\ \\ \end{bmatrix}_{n\times 1} = \begin{bmatrix} \\ \\ \end{bmatrix}_{n\times M} \begin{bmatrix} \\ \\ \end{bmatrix}_{M\times 1}$; $[\phi_1(x_i) \dots \phi_M(x_i)]$)
- Gaussian process models replace explicit basis function representation with a direct specification in terms of a ***positive definite kernel function***

Mercer Kernel Functions

- Predictions are of the form

$$p(f) = N(f \mid 0, \alpha^{-1}\Phi\Phi^T) = N(f \mid 0, K)$$

where the ***Gram matrix*** *K* is defined as

$$K_{ij} = k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

- *K* is a ***Mercer kernel*** if the Gram matrix is positive definite for any *n* and any $x_1, \ldots, x_n$

Note: K is nxn regardless of M (dim of basis/features)

Example of the "kernel trick"

Polynomial kernels

- All monomials of degree d in O(d) operations:

$\phi(u) \cdot \phi(v) = (u \cdot v)^p =$ polynomials of degree exactly *p*

- How about all monomials of degree up to *p*?
  - Solution 0:
  - Better solution:

Common Kernels

- Polynomials of degree exactly d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d$$

- Polynomials of degree up to d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d$$

- Gaussian (squared exponential) kernel

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u} - \mathbf{v}||}{2\sigma^2}\right)$$

- Sigmoid

$$K(\mathbf{u}, \mathbf{v}) = \tanh(\eta \mathbf{u} \cdot \mathbf{v} + \nu)$$

What you need to know

- Notion of online learning
- Perceptron algorithm
- Mistake bounds and proofs
- The kernel trick
- Kernelized perceptron
- Derive polynomial kernel
- Common kernels
- In online learning, report averaged weights at the end