Module 5: Classification

Mixture Models for Classification

STAT/BIOSTAT 527, University of Washington
Emily Fox
May 27th, 2014

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**

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

- \( \pi_k \): mix weights
- \( \mu_k \): shape params
- \( \Sigma_k \): mixture comp.

\( \ln \text{ID:} \)

Our actual observations

- Imagine we have an assignment of each \( x_i \) to a Gaussian.

C. Bishop, *Pattern Recognition & Machine Learning*
Clustering our Observations

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

  - Introduce latent cluster indicator variable $z_i$
    
    $z_i \in \{1, \ldots, K\}$
    
    $\Pr(z_i = k) = \pi_k$

  - Then we have
    
    $p(x_i \mid z_i, \pi, \mu, \Sigma) = \frac{\prod_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_j \prod_k \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$

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

  - “Responsibility”
    
    $r_{ik} = p(z_i = k \mid x_i, \pi, \theta) = \frac{\prod_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_j \prod_k \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$

  - 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
    \[ \hat{\pi}_{ik} = \frac{\hat{\pi}_k N(x_i; \hat{\mu}_k, \hat{\Sigma}_k)}{\sum_{\ell} \hat{\pi}_\ell N(x_i; \hat{\mu}_\ell, \hat{\Sigma}_\ell)} \]
  - Maximize parameters given responsibilities

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

- 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 27th, 2014

Estimating Click Probabilities

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

Query

Ad Info

Features of user

MODEL

Yes!

No
Challenge 1: Complexity of Computing Gradients

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

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

For each \( j \)

Naively, \( O(n d^2) \)

but cache \( \hat{p} \) (same \( x_j \)) \( \Rightarrow O(n d) \)

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 \( x_i \), 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!!

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 $y$ closer to $\beta^*$:
  - Mistake at time $t$: $\beta^{(t+1)} = \beta^{(t)} + y_t x_t$
  - Taking dot product with $\beta^*$:
  - 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:
    \[ \text{Ve} \{ f(x) \text{ subgradient} \}
    \text{if}
    \[ f(y) \geq f(x) + \sqrt{y-x} \]
    \[ \text{Ve} \{ [-1,1] \} \]

Subgradient of Hinge

- Hinge loss:

- Subgradient of hinge loss:
  - If \( y_i(\beta \cdot x_i) > 0 \):
  - If \( y_i(\beta \cdot x_i) < 0 \):
  - If \( y_i(\beta \cdot x_i) = 0 \):
  - 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