Lecture 3:

Classification II

Sam Roweis

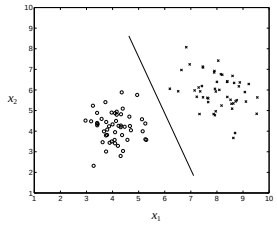
September 23, 2003

Probabilistic Classification: Bayes Classifiers

- Generative model: $p(\mathbf{x}, y) = p(y)p(\mathbf{x}|y)$.
  $p(y)$ are called class *priors*.
  $p(\mathbf{x}|y)$ are called *class conditional feature distributions*.
- For the prior we use a Bernoulli or multinomial:
  $p(y = k|\pi) = \pi_k$ with $\sum_k \pi_k = 1$.
- Classification rules:
  ML?: $\operatorname{argmax}_y p(\mathbf{x}|y)$ (no! can behave badly if skewed priors)
  MAP: $\operatorname{argmax}_y p(y|\mathbf{x}) = \operatorname{argmax}_y \log p(\mathbf{x}|y) + \log p(y)$ (yes!)
- Fitting: maximize $\sum_n \log p(\mathbf{x}^n, y^n) = \sum_n \log p(\mathbf{x}^n|y^n) + \log p(y^n)$
  1) Sort data into batches by class label.
  2) Estimate $p(y)$ by counting size of batches (plus regularization).
  3) Estimate $p(\mathbf{x}|y)$ separately within each batch using ML.
  (also with regularization).

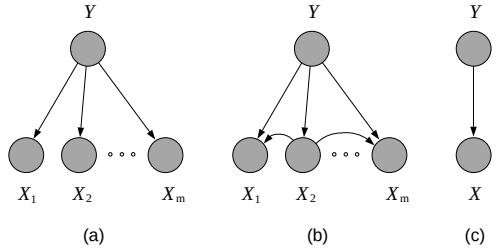
Review: Classification

- Given examples of a discrete *class label* y and some *features* $\mathbf{x}$.
- Goal: compute y for new $\mathbf{x}$.
- Two approaches:
  Generative: model $p(\mathbf{x}, y) = p(y)p(\mathbf{x}|y)$;
  then use Bayes' rule to infer conditional $p(y|\mathbf{x})$.
  Discriminative: model discriminants $f(y|\mathbf{x})$ directly and take max.
- Generative approach is related to conditional *density estimation* while discriminative approach is closer to *regression*.



Three Key Regularization Ideas

- To avoid overfitting, we can put *priors* on the parameters of the class and class conditional feature distributions.
- We can also *tie* some parameters together so that fewer of them are estimated using more data.
- Finally, we can make *factorization* or *independence* assumptions about the distributions. In particular, for the class conditional distributions we can assume the features are fully dependent, partly dependent, or independent (!).



Class Priors and Multinomial Smoothing

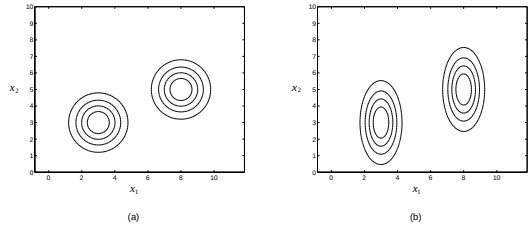
- Let's say you were trying to estimate the bias of a coin. You flip it K times; what is your estimate of the probability z of heads?
- One answer: maximum likelihood. $z = \#h/K$.
- What if you flip it 2 times and you get both heads? Do you think that $z = 1$? Would you be infinitely surprised to see a tail?
- ML is almost always a bad idea. We need to incorporate a *prior* belief to modulate the results of small numbers of trials.
- We do this with a technique called *smoothing*: $z^* = \frac{\#h+\alpha}{K+2\alpha}$
  α are the number of "pseudo-counts" you use for your prior.
- Same situation occurs for estimating class priors:

$$p^*(c) = \frac{\#c + \alpha}{N + C\alpha}$$

- A very common setting is $\alpha = 1$ which is called *Laplace Smoothing*.

Regularized Gaussians

- Idea 1: assume all the covariances are the same (tie parameters). This is exactly Fisher's linear discriminant analysis.



- Idea 2: use a Wishart prior on the covariance matrix. (Smoothing!) This "fattens up" the posteriors by making the MAP estimates the sample covariances plus a bit of the identity matrix.
- Idea 3: Make independence assumptions to get diagonal or identity-multiple covariances. (i.e. sparse inverse covariances.) More on this in a few minutes...

Gaussian Class-Conditional Distributions

- If all features are continuous, a popular choice is a Gaussian class-conditional model.

$$p(\mathbf{x}|y = k, \theta) = |2\pi\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(\mathbf{x} - \mu_k)\Sigma^{-1}(\mathbf{x} - \mu_k)\right\}$$

- Fitting: use the following amazing and useful fact.
  The maximum likelihood fit of a Gaussian to some data is the Gaussian whose mean is equal to the data mean and whose covariance is equal to the sample covariance.
  [Try to prove this as an exercise in understanding likelihood, algebra, and calculus all at once!]
- Seems easy. And works amazingly well.
  But we can do even better with some simple regularization...

Gaussian Bayes Classifier

- Maximum likelihood estimates for parameters:
  priors π_k: use observed frequencies of classes (plus smoothing)
  means μ_k: use class means
  covariance Σ: use data from single class or pooled data
  $(\mathbf{x}^m - \mu_{y^m})$ to estimate (full/diagonal) covariances
- Compute the posterior via Bayes' rule (equal covars):

$$\begin{aligned} p(y = k|\mathbf{x}, \theta) &= \frac{p(\mathbf{x}|y = k, \theta)p(y = k|\pi)}{\sum_j p(\mathbf{x}|y = j, \theta)p(y = j|\pi)} \\ &= \frac{\exp\{\mu_k^\top \Sigma^{-1}\mathbf{x} - \mu_k^\top \Sigma^{-1}\mu_k/2 + \log \pi_k\}}{\sum_j \exp\{\mu_j^\top \Sigma^{-1}\mathbf{x} - \mu_j^\top \Sigma^{-1}\mu_j/2 + \log \pi_j\}} \\ &= \frac{e^{\beta_k^\top \mathbf{x}}}{\sum_j e^{\beta_j^\top \mathbf{x}}} = \exp\{\beta_k^\top \mathbf{x}\}/Z \end{aligned}$$

e.g. $\beta_k = [\Sigma^{-1}\mu_k \,;\, (\mu_k^\top \Sigma^{-1}\mu_k + \log \pi_k)]$ (last term is bias)

Linear Geometry

- Taking the ratio of any two posteriors (the "odds") shows that the contours of equal pairwise probability are linear surfaces in the feature space if the covariances of all classes are equal:

$$\frac{p(y=k|\mathbf{x},\theta)}{p(y=j|\mathbf{x},\theta)} = \exp\left\{(\beta_k - \beta_j)^\top \mathbf{x}\right\}$$

- The pairwise discrimination contours $p(y_k) = p(y_j)$ are orthogonal to the differences of the means in feature space when $\Sigma = \sigma I$. For general Σ shared b/w all classes the same is true in the transformed feature space $\mathbf{w} = \Sigma^{-1}\mathbf{x}$.
- The priors do not change the geometry, they only shift the operating point on the logit by the log-odds $\log(\pi_k/\pi_j)$.
- Summary: for equal class-covariances, we obtain a *linear classifier*.
- If we use difference covariances, the decision surfaces are conic sections and we have a quadratic classifier.

Naive (Idiot's) Bayes Classifier

- Assumption: conditioned on class, attributes are independent.

$$p(\mathbf{x}|y) = \prod_i p(x_i|y)$$

- Sounds crazy right? Right! But it works.
- Algorithm: sort data cases into bins according to y_n. Compute marginal probabilities $p(y=c)$ using frequencies.
- For each class, estimate distribution of i^{th} variable: $p(x_i|y=c)$.
- At test time, compute $\mathrm{argmax}_c\, p(c|\mathbf{x})$ using

$$\begin{aligned} c(\mathbf{x}) = \mathrm{argmax}_c\, p(c|\mathbf{x}) &= \mathrm{argmax}_c\, [\log p(\mathbf{x}|c) + \log p(c)] \\ &= \mathrm{argmax}_c\, [\log p(c) + \sum_i \log p(x_i|c)] \end{aligned}$$

Discrete Bayesian Classifier

- If the inputs are discrete (categorical), what should we do?
- The simplest class conditional model is a joint multinomial (table):

$$p(x_1 = a, x_2 = b, \ldots | y = c) = \eta^c_{ab\ldots}$$

- This is conceptually correct, but there's a big practical problem.
- Fitting: ML params are observed counts:

$$\eta^c_{ab\ldots} = \frac{\sum_n [y_n = c][x_1 = a][x_2 = b][\ldots][\ldots]}{\sum_n [y_n = c]}$$

- Consider the 16x16 digits at 256 gray levels.
- How many entries in the table? How many will be zero? What happens at test time? Doh!
- We obviously need some regularlization. Smoothing will not help much here. Unless we know about the relationships between inputs beforehand, sharing parameters is hard also (what to share?). But what about independence?

Discrete (Multinomial) Naive Bayes

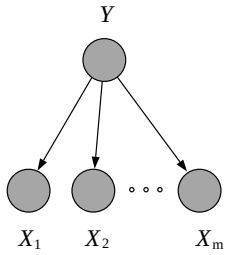
Discrete features x_i, assumed independent given the class label y.

$$p(x_i = j|y = k) = \eta_{ijk}$$
$$p(\mathbf{x}|y=k,\eta) = \prod_i \prod_j \eta_{ijk}^{[x_i=j]}$$

Classification rule:

$$\begin{aligned} p(y=k|\mathbf{x},\eta) &= \frac{\pi_k \prod_i \prod_j \eta_{ijk}^{[x_i=j]}}{\sum_q \pi_q \prod_i \prod_j \eta_{ijq}^{[x_i=j]}} \\ &= \frac{e^{\beta_k^\top \mathbf{x}}}{\sum_q e^{\beta_q^\top \mathbf{x}}} \end{aligned}$$

$\beta_k = \log[\eta_{11k} \ldots \eta_{1jk} \ldots \eta_{ijk} \ldots \log \pi_k]$
$\mathbf{x} = [x_1{=}1; x_1{=}2; \ldots; x_i{=}j; \ldots; 1]$



(a)

Fitting Discrete Naive Bayes

- ML parameters are class-conditional frequency counts:

$$\eta^*_{ijk} = \frac{\sum_m [x_i{}^m = j][y^m = k]}{\sum_m [y^m = k]}$$

- How do we know? Write down the likelihood:

$$\ell(\theta; \mathcal{D}) = \sum_m \log p(y^m|\pi) + \sum_{mi} \log p(x_i{}^m|y^m, \eta)$$

and optimize it by setting its derivative to zero
(careful! enforce normalization with Lagrange multipliers):

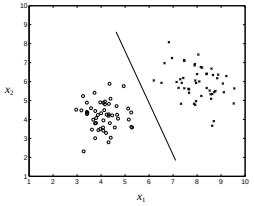
$$\ell(\eta; \mathcal{D}) = \sum_m \sum_{ijk} [x_i{}^m = j][y^m = k] \log \eta_{ijk} + \sum_{ik} \lambda_{ik}(1 - \textstyle\sum_j \eta_{ijk})$$

$$\frac{\partial \ell}{\partial \eta_{ijk}} = \frac{\sum_m [x_i{}^m = j][y^m = k]}{\eta_{ijk}} - \lambda_{ik}$$

$$\frac{\partial \ell}{\partial \eta_{ijk}} = 0 \quad \Rightarrow \lambda_{ik} = \sum_m [y^m = k] \quad \Rightarrow \eta^*_{ijk}$$

Discriminative Models

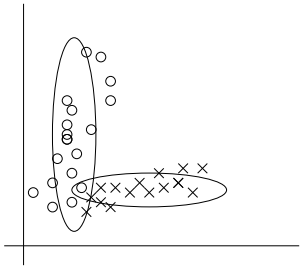
- Observation: if $p(y|\mathbf{x})$ are linear functions of $\mathbf{x}$ (or monotone transforms), decision surfaces will be piecewise linear.



- Idea: parametrize $p(y|\mathbf{x})$ directly, forget $p(\mathbf{x}, y)$ and Bayes' rule.
- Bolder idea: forget $p(y|\mathbf{x})$, just use discriminants $f(y|\mathbf{x})$.
- Don't need to model the density of the features.
  Some density models have lots of parameters.
  Many densities give same linear classifier.
  But we cannot generate new labeled data.
- Why not optimize the same cost function we use at test time?

Gaussian Naive Bayes

- This is just a Gaussian Bayes Classifier with a separate but diagonal covariance matrix for each class.
- Equivalent to fitting a 1D Gaussian to each input for each class.
- NB: Decision surfaces are quadratics, not linear...
- Even better idea:
  Blend between full, diagonal and identity covaraince.



Logistic/Softmax Regression

- Model: y is a multinomial random variable whose posterior is the softmax of linear functions of *any* feature vector $\mathbf{x}$.

$$p(y = k|\mathbf{x}, \theta) = \frac{e^{\theta_k^\top \mathbf{x}}}{\sum_j e^{\theta_j^\top \mathbf{x}}}$$

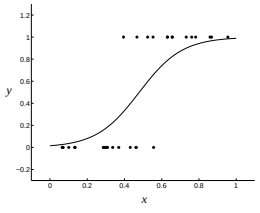
- Fitting: now we optimize the *conditional* likelihood:

$$\ell(\theta; \mathcal{D}) = \sum_{mk} [y^m = k] \log p(y = k|\mathbf{x}^m, \theta) = \sum_{mk} y_k^m \log p_k^m$$

$$\frac{\partial \ell}{\partial \theta_i} = \sum_{mk} \frac{\partial \ell_k^m}{\partial p_k^m} \frac{\partial p_k^m}{\partial z_i^m} \frac{\partial z_i^m}{\partial \theta_i}$$

$$= \sum_{mk} \frac{y_k^m}{p_k^m} p_k^m (\delta_{ik} - p_i^m) \mathbf{x}^m$$

$$= \sum_m (y_k^m - p_k^m) \mathbf{x}^m$$



Softmax/Logit

- The squashing function is known as the *softmax* or *logit*:

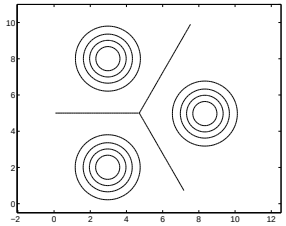
$$\phi_k(\mathbf{z}) \equiv \frac{e^{z_k}}{\sum_j e^{z_j}} \qquad g(\eta) = \frac{1}{1+e^{-\eta}}$$

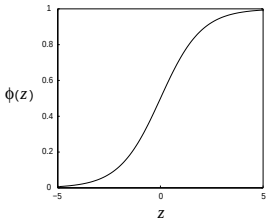
- It is invertible (up to a constant):

$$z_k = \log \phi_k + c \qquad \eta = \log(g/1-g)$$

- Derivative is easy:

$$\frac{\partial \phi_k}{\partial z_j} = \phi_k(\delta_{kj} - \phi_j) \qquad \frac{dg}{d\eta} = g(1-g)$$



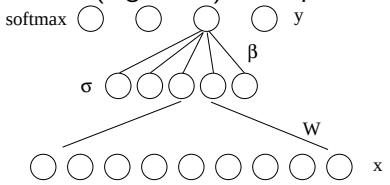


Neural Networks for Classification

- Neural nets with one hidden layer trained for classification are doing nonlinear logistic regression:

$$p(y=k|\mathbf{x}) = \text{softmax}[\beta_k^\top \sigma(\mathbf{W}\mathbf{x})]$$

where $\mathbf{W}$ and β are the first and second layer weights and $\sigma()$ is a squashing function (e.g. tanh) that operates componentwise.



- Gradient of conditional likelihood still easily computable.
- We lose the convexity property – local minima problems.

More on Logistic Regression

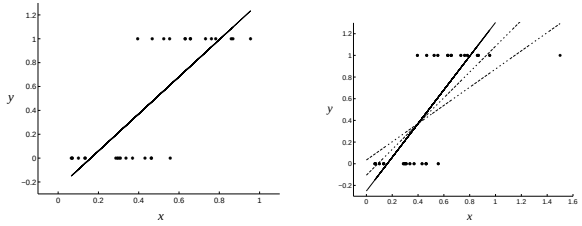
- Hardest Part: picking the feature vector $\mathbf{x}$.
- Amazing fact: the conditional likelihood is (almost) convex in the parameters θ. Still no local minima!
- Gradient is easy to compute; so easy to optimize.
  Slow: gradient descent, IIS. Fast: BFGS, Newton-Raphson, IRLS.
- Why almost? Consider what happens if there are two features with identical classification patterns in our training data. Logistic Regression can only see the sum of the corresponding weights.
- Solution? Weight decay: add $\epsilon \sum \theta^2$ to the cost function, which subtracts $2\epsilon\theta$ from each gradient.
- Why is this method called logistic regression?
- It should really be called "softmax linear regression".
- Log odds (logit) between any two classes is linear in parameters.

Joint vs. Conditional Models

- Many of the methods we have seen so far have linear or piecewise linear decision surfaces in some space $\mathbf{x}'$:
  LDA, perceptron, Gaussian Bayes, Naive Bayes, KNN,...
- But the criteria used to find this hyperplane is different:
- Gauss/Naive Bayes are joint models; optimize $p(\mathbf{x}, y) = p(\mathbf{x})p(y|\mathbf{x})$.
- Logistic Regression/NN are conditional: optimize $p(y|\mathbf{x})$ directly.
- See reading...

Classification via Regression?

- We could forget that y was a discrete (categorical) random variable and just attempt to model $p(y|\mathbf{x})$ using regression.
- Idea: do regression to an *indicator matrix*.
  (in binary case $p(y = 1|\mathbf{x})$ is also the conditional expectation)
- For two classes, this is equivalent* to LDA. For 3 or more, disaster...
- Very bad idea! Noise models (e.g. Gaussian) for regression are totally inappropriate, and fits are oversensitive to outliers.
  Furthermore, gives unreasonable predictions < 0 and > 1.



Exponential Family Class-Conditionals

- Bayes Classifier has the same form whenever the class-conditional densities are *any* exponential family density:

$$\begin{aligned} p(\mathbf{x}|y = k, \eta_k) &= h(\mathbf{x}) \exp\{\eta_k^\top \mathbf{x} - a(\eta_k)\} \\ p(y = k|\mathbf{x}, \eta) &= \frac{p(\mathbf{x}|y = k, \eta_k)p(y = k|\pi)}{\sum_j p(\mathbf{x}|y = j, \eta_j)p(y = j|\pi)} \\ &= \frac{\exp\{\eta_k^\top \mathbf{x} - a(\eta_k)\}}{\sum_j \exp\{\eta_j^\top \mathbf{x} - a(\eta_j)\}} \\ &= \frac{e^{\beta_k^\top \mathbf{x}}}{\sum_j e^{\beta_j^\top \mathbf{x}}} \end{aligned}$$

  where $\beta_k = [\eta_k\,;\, -a(\eta_k)]$ and we have augmented $\mathbf{x}$ with a constant component always equal to 1 (bias term).
- Resulting classifier is linear in the sufficient statistics.

Noisy-OR Classifier

- Many probabilistic models can be obtained as noisy versions of formulas from propositional logic.
- Noisy-OR: each input x_i activates output y w/some probability.

$$p(y = 0|\mathbf{x}, \alpha) = \prod_i \alpha_i^{x_i} = \exp\left\{\sum_i x_i \log \alpha_i\right\}$$

- Letting $\theta_i = -\log \alpha_i$ we get yet another linear classifier:

$$p(y = 1|\mathbf{x}, \theta) = 1 - e^{-\theta^\top \mathbf{x}}$$

Futher points...

- Some classifiers return a single guess for y without a distribution.
- Last class: non-parametric (e.g. K-nearest-neighbour).
- This class: generative & discriminative models.
  (plus many more, e.g. probit regression, complementary log-log, generalized linear models, neural networks with hidden layers, ...)
- Advanced topic: kernel machine classifiers. (e.g. kernel voted perceptron, support vector machines, Gaussian processes).
- Advanced topic: combining multiple weak classifiers into a single stronger one using boosting, bagging, stacking...

Readings: Hastie et. al, Ch4; Duda&Hart, Ch3,4.10