CSC2515 – Machine Learning Sam Roweis

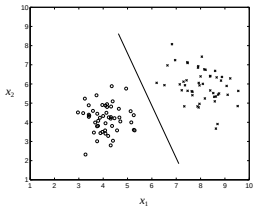
Lecture 3:

Classification II

September 26, 2006

Review: Classification 1

- Given examples of a discrete *class label* y and some *features* $\mathbf{x}$.
- Goal: compute y for new $\mathbf{x}$.
- Last class: compute a discriminant $f(y|\mathbf{x})$ and then take max y.
- This class: probabilistic classifiers. 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 posterior $p(y|\mathbf{x})$ directly.
- Generative approach is related to joint *density estimation* while discriminative approach is closer to *regression*.

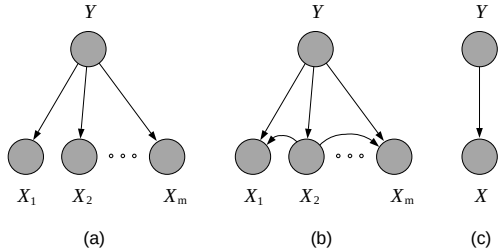


Probabilistic Classification: Bayes Classifiers 2

- 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 $\sum_k \pi_k = 1$; $\pi_k > 0$.
- What classification rule should we use? Pick the class that best models the data, ie $\mathrm{argmax}_y\ p(\mathbf{x}|y)$? No! This behaves very badly if the class priors are skewed. MAP is best:
  $\mathrm{argmax}_y\ p(y|\mathbf{x}) = \mathrm{argmax}_y\ p(\mathbf{x}, y) = \mathrm{argmax}_y\ \log p(\mathbf{x}|y) + \log p(y)$
- How should we fit model parameters? *Maximum joint likelihood.*
  $\Phi = \sum_n \log p(\mathbf{x}^n, y^n) = \sum_n \log p(\mathbf{x}^n|y^n) + \log p(y^n)$
  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 on the class-conditional model (also with regularization).

Three Key Regularization Ideas 3

- To avoid overfitting, we can put *priors* on the parameters of both 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 (!).



- 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.
- The same situation occurs when estimating class priors from data:

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

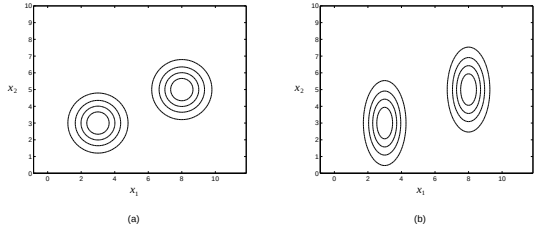
- A very common setting is $\alpha = 1$ which is called *Laplace Smoothing*.

- If all the input features x_i 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 simple but 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!]
- One very nice feature of this model is that the maximum likelihood parameters can be found in closed-form, so we don't have to worry about numerical optimization, local minima, search, etc.
- Seems easy. And works surprisingly well.
  But we can do even better with some simple regularization...

- 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...

- 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 $(\mathbf{x}^n - \mu_{y^n})$ to estimate (full/diagonal) covariances
- Compute the posterior via Bayes' rule. For equal covariances:

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

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

- 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:

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

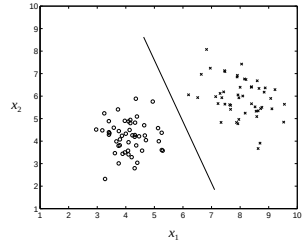
- The pairwise discrimination contours $p(y_k) = p(y_j)$ are orthogonal to the differences of the means in feature space when $\Sigma = \sigma I$. For general Σ (shared b/w all classes) the same is true in the transformed feature space $\mathbf{u} = \Sigma^{-1}\mathbf{x}$.
- The priors do not change 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 different covariances for each class, we have a quadratic classifier with *conic section* decision surfaces.

- 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 $\mathbf{x}$.

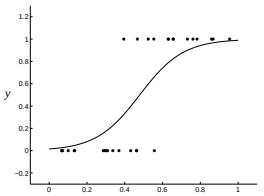
- Observation: if $p(y|\mathbf{x})$ are linear functions of $\mathbf{x}$ (or monotone transforms), decision surfaces will be piecewise linear.
- Idea: parametrize $p(y|\mathbf{x})$ directly, forget $p(\mathbf{x}, y)$ and Bayes' rule.



- Advantages: We don't need to model the density of the features $p(x)$ which often takes lots of parameters and seems redundant since many densities give the same linear classifier.
- Disadvantages: We cannot detect outliers, compare models using likelihoods or generate new labelled data.
- What should our objective function be? We'll try to use one that is closer to the one we care about at test time (ie error rate).

- Model: y is a multinomial random variable whose posterior is the "softmax" of linear functions of the 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* log-likelihood:

$$\begin{aligned}
\ell(\theta;\mathcal{D}) &= \sum_n \log p(y=y^n|\mathbf{x}^n,\theta) = \sum_{nk} y_k^n \log p_k^n \qquad y_k^n \equiv [y^n == k] \\
\frac{\partial \ell}{\partial \theta_i} &= \sum_{nk} y_k^n \frac{\partial \log p_k^n}{\partial p_k^n}\frac{\partial p_k^n}{\partial \theta_i} \qquad p_k^n \equiv p(y=k|\mathbf{x}^n) \\
&= \sum_{nk} y_k^n \frac{1}{p_k^n} p_k^n(\delta_{ik} - p_i^n)\mathbf{x}^n \qquad \delta_{ik} \equiv [i == k] \\
&= \sum_n (y_i^n - p_i^n)\mathbf{x}^n \qquad \sum_n (\text{target}^n - \text{prediction}^n) * \text{feature}^n
\end{aligned}$$

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

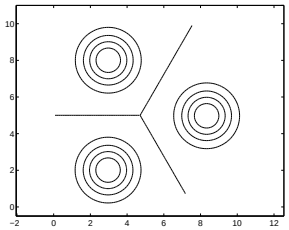
$$\phi_k(\mathbf{z}) \equiv \frac{e^{z_k}}{\sum_j e^{z_j}} \qquad g(\eta) = \frac{1}{1+e^{-\eta}}$$

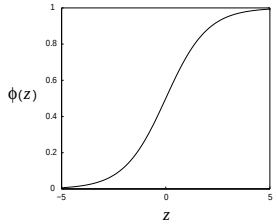
- It is invertible (up to a constant):

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

- Derivative is easy:

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





- Hardest Part: picking the feature vector $\mathbf{x}$ (see next slide!).
- Amazing fact: the conditional likelihood is convex* in the parameters θ (assuming regularization). Still no local minima! But the optimal parameters cannot be computed in closed form.
- However, the gradient is easy to compute; so easy to optimize. Slow: gradient descent, IIS. Fast: BFGS, Newton-Raphson, IRLS.
- Regularization? Gaussian prior on θ (weight decay): add $\epsilon \sum \theta^2$ to the cost function, which subtracts $2\epsilon\theta$ from each gradient.
- Logistic regression could really be called "softmax linear regression". Log odds (logit) between any two classes is linear in parameters.
- * 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. Luckily, weight decay will solve this. Moral: always regularize!

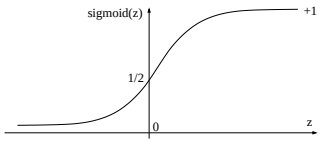
- Historically motivated by relations to biology, but for our purposes, ANNs are just nonlinear classification machines of the form:

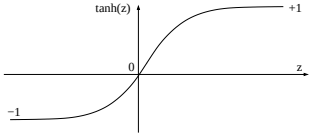
$$p(y=k|\mathbf{x},\theta) = \frac{e^{\theta_k^\top \mathbf{h}(\mathbf{x})}}{\sum_j e^{\theta_j^\top \mathbf{h}(\mathbf{x})}} \qquad h_j = \sigma(\mathbf{b}_j^\top \mathbf{x})$$

  where $h_j = \sigma(\mathbf{b}_j^\top \mathbf{x})$ are known as the *hidden unit activations*; y_k are the output units and x_i are the input units.
- The nonlinear scalar function σ is called an *activation function*. We usually use *invertible* and *differentiable* activation functions.
- If the activation function is *linear*, the whole network reduces* to a linear network: equivalent to logistic regression. [*Only if there are at least as many hiddens as inputs and outputs.]
- It is often a good idea to add "skip weights" directly connecting inputs to outputs to take care of this linear component directly.

- Two common activation functions: *sigmoid* and *hyperbolic tangent*

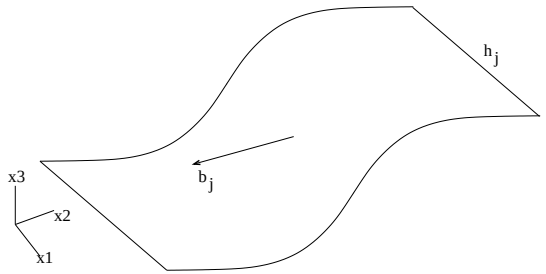
$$\text{sigmoid(z)} = \frac{1}{1+\exp(-z)} \qquad \tanh(\text{z}) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$$





- For small weights, these functions will be operating near zero and their behaviour will be almost linear.
- Thus, for small weights, the network behaves essentially linearly.
- But for larger weights, we are effectively *learning the input feature functions* for a non-linear version of logistic regression.
- In general we want a saturating activation function (why?).

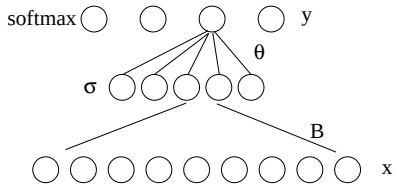
- ANNs can be thought of as generalized linear models, where the basis functions (hidden units) are sigmoidal "cliffs".
- The cliff direction is determined by the input-to-hidden weights, and the cliffs are combined by the hidden-to-output weights.
- We include bias units of course, and these set where the cliff is positioned relative to the origin.



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

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

where θ and $\mathbf{B}$ 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, using the efficient *backpropagation algorithm* which we'll see later.
- But: We lose the convexity property – local minima problems.

- 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?

- 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 $\text{argmax}_c\ p(c|\mathbf{x})$ using

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

- Even if the assumption is wrong, this does well on 0-1 loss. [Domingos & Pazzani]

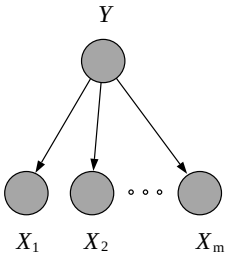
Discrete features x_i, assumed independent given the class label y.

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

Classification rule:

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

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



(a)

- ML parameters are class-conditional frequency counts:

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

- How do we know? Write down the likelihood:

$$\ell(\theta; \mathcal{D}) = \sum_n \log p(y^n | \pi) + \sum_{ni} \log p(x_i{}^n | y^n, \eta)$$

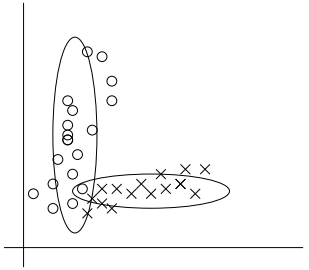
and optimize it by setting its derivative to zero
(careful! enforce normalization with Lagrange multipliers):

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

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

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

- 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.



- 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.

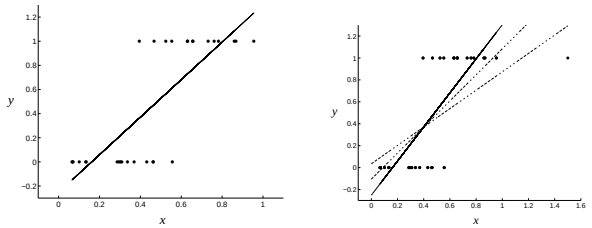
$$p(y = 0 | \mathbf{x}, \alpha) = \prod_i \alpha_i^{x_i} = \exp\left\{ \sum_i x_i \log \alpha_i \right\}$$

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

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

- 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, Noisy-OR, KNN,...
- But the criteria used to find this hyperplane is different:
- KNN/perceptron optimize training set classification error.
- 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.
- Very important point: in general there is a large difference between the *architecture* used for classification and the *objective function* used to optimize the parameters of the architecture.
- See reading...

- 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...
- Generally a 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.



- Last class: non-parametric (e.g. K-nearest-neighbour).
  Those classifiers return a single guess for y without a distribution.
- This class: probabilistic generative models $p(\mathbf{x}, y)$ (e.g. Gaussian class-conditional, Naive Bayes) & discriminative (conditional) models $p(y|\mathbf{x})$ (e.g. logistic regression, ANNs, noisy-OR).
  (Plus many more we didn't talk about , e.g. probit regression, complementary log-log, generalized linear models, ...)
- 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