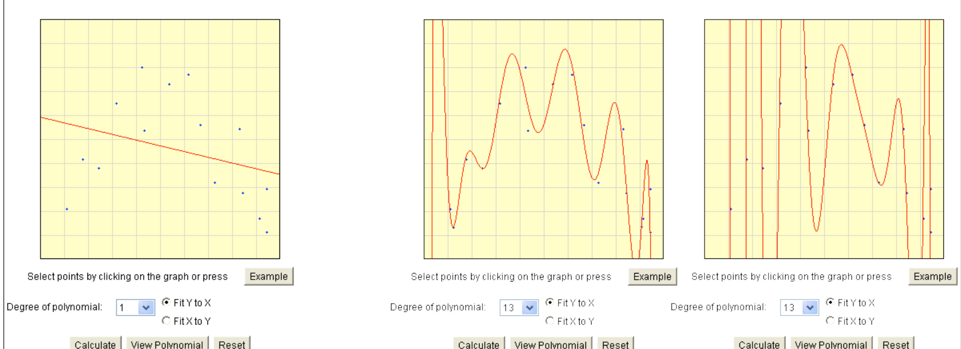
What's learning, revisited
Overfitting
Generative versus Discriminative
Logistic Regression

Machine Learning – 10701/15781
Carlos Guestrin
Carnegie Mellon University

September 19th, 2007

Bias-Variance Tradeoff

- Choice of hypothesis class introduces learning bias
  - More complex class → less bias
  - More complex class → more variance



Training set error

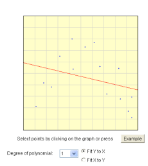
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

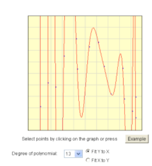
- Given a dataset (Training data)
- Choose a loss function
  - e.g., squared error (L_2) for regression
- **Training set error:** For a particular set of parameters, loss function on training data:

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

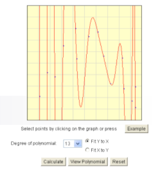
Training set error as a function of model complexity

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$





Prediction error



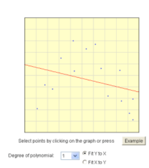
- Training set error can be poor measure of "quality" of solution
- **Prediction error:** We really care about error over all possible input points, not just training data:

$$
\begin{aligned}
error_{true}(\mathbf{w}) &= E_{\mathbf{x}}\left[\left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2\right] \\
&= \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x}
\end{aligned}
$$

Prediction error as a function of model complexity

$$
error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2
$$

$$
error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x}
$$



Computing prediction error

- Computing prediction
  - hard integral
  - May not know t(**x**) for every **x**

$$error_{true}(\mathbf{w}) \quad = \quad \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x}$$

- Monte Carlo integration (sampling approximation)
  - Sample a set of i.i.d. points $\{\mathbf{x}_1,\ldots,\mathbf{x}_M\}$ from p(**x**)
  - Approximate integral with sample average

$$error_{true}(\mathbf{w}) \quad \approx \quad \frac{1}{M} \sum_{j=1}^{M} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

Why training set error doesn't approximate prediction error?

- Sampling approximation of prediction error:

$$error_{true}(\mathbf{w}) \quad \approx \quad \frac{1}{M} \sum_{j=1}^{M} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

- Training error :

$$error_{train}(\mathbf{w}) \quad = \quad \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

- Very similar equations!!!
  - Why is training set a bad measure of prediction error???

Why training set error doesn't approximate prediction error?

Because you cheated!!!

Training error good estimate for a single **w,**
But you optimized **w** with respect to the training error, and found **w** that is good for this set of samples

Training error is a (optimistically) biased estimate of prediction error

- Very similar equations!!!
  - Why is training set a bad measure of prediction error???

Test set error

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

- Given a dataset, **randomly** split it into two parts:
  - Training data – {$\mathbf{x}_1, \ldots, \mathbf{x}_{Ntrain}$}
  - Test data – {$\mathbf{x}_1, \ldots, \mathbf{x}_{Ntest}$}
- Use training data to optimize parameters **w**
- **Test set error:** For the ***final solution*** **w***, evaluate the error using:

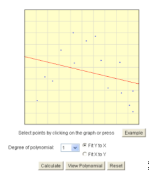
$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

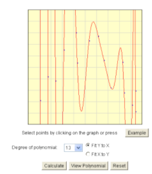
Test set error as a function of model complexity

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x}$$

$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$





Overfitting

- **Overfitting:** a learning algorithm overfits the training data if it outputs a solution **w** when there exists another solution **w'** such that:

$$[error_{train}(\mathbf{w}) < error_{train}(\mathbf{w}')] \wedge [error_{true}(\mathbf{w}') < error_{true}(\mathbf{w})]$$

How many points to I use for training/testing?

- Very hard question to answer!
  - Too few training points, learned **w** is bad
  - Too few test points, you never know if you reached a good solution
- Bounds, such as Hoeffding's inequality can help:

$$P(\mid \widehat{\theta} - \theta^* \mid \geq \epsilon) \;\; \leq \;\; 2e^{-2N\epsilon^2}$$

- More on this later this semester, but still hard to answer
- Typically:
  - if you have a reasonable amount of data, pick test set "large enough" for a "reasonable" estimate of error, and use the rest for learning
  - if you have little data, then you need to pull out the big guns…
    - e.g., bootstrapping

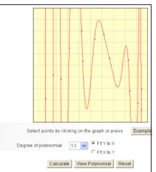
Error estimators

$$error_{true}(\mathbf{w}) \;\; = \;\; \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x}$$

$$error_{train}(\mathbf{w}) = \;\; \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

$$error_{test}(\mathbf{w}) \;\; = \;\; \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

Error as a function of number of training examples for a fixed model complexity



$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

little data → infinite data

Error estimators

err

err

Be careful!!!

Test set only unbiased if you never never never never do any any any any learning on the test data

For example, if you use the test set to select the degree of the polynomial… no longer unbiased!!! (We will address this problem later in the semester)

$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

Announcements

- First homework is out:
  - Programming part and Analytic part
  - Remember collaboration policy: can discuss questions, but need to write your own solutions and code
  - Remember you are not allowed to look at previous years' solutions, search the web for solutions, use someone else's solutions, etc.
  - Due Oct. 3rd **beginning of class**
  - Start early!
- Recitation this week:
  - Bayes optimal classifiers, Naïve Bayes

What's (supervised) learning, more formally

- Given:
  - **Dataset**: Instances $\{\langle \mathbf{x}_1;t(\mathbf{x}_1)\rangle,\ldots, \langle \mathbf{x}_N;t(\mathbf{x}_N)\rangle\}$
    - e.g., $\langle \mathbf{x}_i;t(\mathbf{x}_i)\rangle = \langle$(GPA=3.9,IQ=120,MLscore=99);150K$\rangle$
  - **Hypothesis space**: H
    - e.g., polynomials of degree 8
  - **Loss function**: measures quality of hypothesis $h \in H$
    - e.g., squared error for regression
- Obtain:
  - **Learning algorithm**: obtain $h \in H$ that minimizes loss function
    - e.g., using matrix operations for regression
    - Want to minimize prediction error, but can only minimize error in dataset

Types of (supervised) learning problems, revisited

- **Regression**, e.g.,
  - **dataset**: ⟨position; temperature⟩
  - **hypothesis space**:
  - **Loss function**:

- **Density estimation**, e.g.,
  - **dataset**: ⟨grades⟩
  - **hypothesis space**:
  - **Loss function**:

- **Classification**, e.g.,
  - **dataset**: ⟨brain image; {verb v. noun}⟩
  - **hypothesis space**:
  - **Loss function**:

Learning is (simply) function approximation!

- The general (supervised) learning problem:
  - Given some data (including features), hypothesis space, loss function
  - Learning is no magic!
  - Simply trying to find a function that fits the data

- **Regression**

- **Density estimation**

- **Classification**

- (Not surprisingly) Seemly different problem, very similar solutions…

What is NB really optimizing?

- Naïve Bayes assumption:
  - Features are independent given class:

$$P(X_1, X_2|Y) = P(X_1|X_2, Y)P(X_2|Y)$$

  - More generally:

$$\boxed{P(X_1 ... X_n|Y) = \prod_i P(X_i|Y)}$$

- NB Classifier:

MLE for the parameters of NB

- Given dataset
  - Count(A=a,B=b) $\leftarrow$ number of examples where A=a and B=b
- MLE for NB, simply:
  - Prior: P(Y=y) =
  - Likelihood: $P(X_i=x_i|Y_i=y_i)$ =

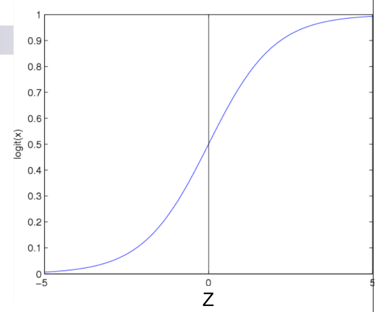
What is NB really optimizing? Let's use an example

Generative v. Discriminative classifiers – Intuition

- **Want to Learn**: h:**X** $\mapsto$ Y
  - **X** – features
  - Y – target classes
- **Bayes optimal classifier** – P(Y|**X**)
- **Generative classifier**, e.g., Naïve Bayes:
  - Assume some **functional form for P(X|Y), P(Y)**
  - Estimate parameters of P(X|Y), P(Y) directly from training data
  - Use Bayes rule to calculate P(Y|X= x)
  - This is a **'*generative*' model**
    - **Indirect** computation of P(Y|X) through Bayes rule
    - But, **can generate a sample of the data**, $P(X) = \sum_y P(y) P(X|y)$
- **Discriminative classifiers**, e.g., Logistic Regression:
  - Assume some **functional form for P(Y|X)**
  - Estimate parameters of P(Y|X) directly from training data
  - This is the **'*discriminative*' model**
    - Directly learn P(Y|X)
    - But **cannot obtain a sample of the data**, because P(X) is not available

Logistic Regression

Logistic function (or Sigmoid): $\frac{1}{1+exp(-z)}$



- Learn P(Y|**X**) directly!
  - Assume a particular functional form
  - Sigmoid applied to a linear function of the data:

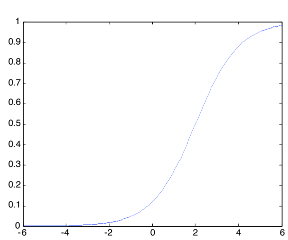
$$P(Y=1|X) = \frac{1}{1+\exp(w_0+\sum_{i=1}^{n} w_i X_i)}$$

Features can be discrete or continuous!

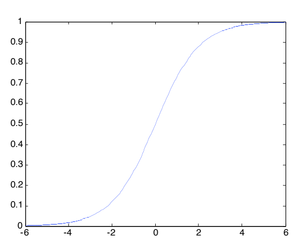
Understanding the sigmoid

$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1+e^{w_0+\sum_i w_i x_i}}$$

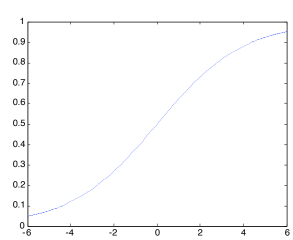
w_0=-2, w_1=-1



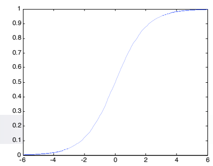
w_0=0, w_1=-1



w_0=0, w_1=-0.5



Logistic Regression – a Linear classifier



$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{w_0 + \sum_i w_i x_i}}$$

Very convenient!

$$P(Y = 1 | X = < X_1, ... X_n >) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

implies

$$P(Y = 0 | X = < X_1, ... X_n >) = \frac{exp(w_0 + \sum_i w_i X_i)}{1 + exp(w_0 + \sum_i w_i X_i)}$$

implies

$$\frac{P(Y = 0|X)}{P(Y = 1|X)} = exp(w_0 + \sum_i w_i X_i)$$

linear classification rule!

implies

$$\ln \frac{P(Y = 0|X)}{P(Y = 1|X)} = w_0 + \sum_i w_i X_i$$

Logistic regression v. Naïve Bayes

- Consider learning f: X → Y, where
  - X is a vector of real-valued features, < X1 … Xn >
  - Y is boolean
- Could use a Gaussian Naïve Bayes classifier
  - assume all X_i are conditionally independent given Y
  - model $P(X_i \mid Y = y_k)$ as Gaussian $N(\mu_{ik},\sigma_i)$
  - model P(Y) as Bernoulli(θ,1-θ)
- What does that imply about the form of P(Y|X)?

$$P(Y=1|X=<X_1,...X_n>) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

Cool!!!!

Derive form for P(Y|X) for continuous X_i

$$P(Y=1|X) = \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1)+P(Y=0)P(X|Y=0)}$$

$$= \frac{1}{1+\frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}}$$

$$= \frac{1}{1+\exp(\ln\frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})}$$

$$= \frac{1}{1+\exp(\ (\ln\frac{1-\theta}{\theta})+\boxed{\sum_i \ln\frac{P(X_i|Y=0)}{P(X_i|Y=1)})}}$$

Ratio of class-conditional probabilities

$$\ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)}$$

$$\boxed{P(X_i = x \mid Y = y_k) = \frac{1}{\sigma_i \sqrt{2\pi}} \; e^{\frac{-(x-\mu_{ik})^2}{2\sigma_i^2}}}$$

Derive form for P(Y|X) for continuous X_i

$$P(Y=1|X) = \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1) + P(Y=0)P(X|Y=0)}$$

$$= \frac{1}{1 + \exp(\;(\ln \frac{1-\theta}{\theta}) + \boxed{\sum_i \ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)}})}$$

$$\boxed{\sum_i \left(\frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2} \right)}$$

$$P(Y=1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

Gaussian Naïve Bayes v. Logistic Regression

Set of Gaussian Naïve Bayes parameters (feature variance independent of class label)

Set of Logistic Regression parameters

- Representation equivalence
  - **But only in a special case!!!** (GNB with class-independent variances)
- But what's the difference???
- **LR makes no assumptions about** P(**X**|Y) **in learning**!!!
- **Loss function!!!**
  - Optimize different functions $\rightarrow$ Obtain different solutions

Logistic regression for more than 2 classes

- Logistic regression in more general case, where $Y \in \{Y_1 \ldots Y_R\}$: learn R-1 sets of weights

Logistic regression more generally

- Logistic regression in more general case, where $Y \in \{Y_1 \ldots Y_R\}$: learn R-1 sets of weights

for $k<R$

$$P(Y = y_k|X) = \frac{\exp(w_{k0} + \sum_{i=1}^{n} w_{ki}X_i)}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji}X_i)}$$

for $k=R$ (normalization, so no weights for this class)

$$P(Y = y_R|X) = \frac{1}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji}X_i)}$$

Features can be discrete or continuous!

Announcements

- Don't forget recitation tomorrow
- And start the homework early

Loss functions: Likelihood v. Conditional Likelihood

- Generative (Naïve Bayes) Loss function:
  Data likelihood

$$\begin{aligned} \ln P(\mathcal{D} \mid \mathbf{w}) &= \sum_{j=1}^{N} \ln P(\mathbf{x}^j, y^j \mid \mathbf{w}) \\ &= \sum_{j=1}^{N} \ln P(y^j \mid \mathbf{x}^j, \mathbf{w}) + \sum_{j=1}^{N} \ln P(\mathbf{x}^j \mid \mathbf{w}) \end{aligned}$$

- Discriminative models cannot compute P($\mathbf{x}^j$|**w**)!
- But, discriminative (logistic regression) loss function:
  Conditional Data Likelihood

$$\ln P(\mathcal{D}_Y \mid \mathcal{D}_{\mathbf{X}}, \mathbf{w}) = \sum_{j=1}^{N} \ln P(y^j \mid \mathbf{x}^j, \mathbf{w})$$

  - Doesn't waste effort learning P(X) – focuses on P(Y|**X**) all that matters for classification

Expressing Conditional Log Likelihood

$$l(\mathbf{w}) \equiv \sum_j \ln P(y^j | \mathbf{x}^j, \mathbf{w})$$

$$P(Y=0|\mathbf{X},\mathbf{w}) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

$$P(Y=1|\mathbf{X},\mathbf{w}) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

$$l(\mathbf{w}) = \sum_j y^j \ln P(y^j = 1|\mathbf{x}^j, \mathbf{w}) + (1-y^j) \ln P(y^j = 0|\mathbf{x}^j, \mathbf{w})$$

Maximizing Conditional Log Likelihood

$$P(Y=0|X,W) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

$$P(Y=1|X,W) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

$$\begin{aligned} l(\mathbf{w}) &\equiv \ln \prod_j P(y^j|\mathbf{x}^j,\mathbf{w}) \\ &= \sum_j y^j(w_0+\sum_i^n w_i x_i^j) - \ln(1+exp(w_0+\sum_i^n w_i x_i^j)) \end{aligned}$$

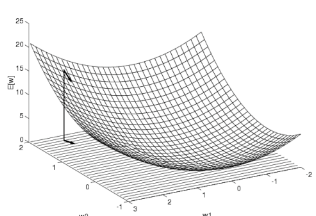
Good news: $l(\mathbf{w})$ is concave function of $\mathbf{w}$ $\rightarrow$ no locally optimal solutions

Bad news: no closed-form solution to maximize $l(\mathbf{w})$

Good news: concave functions easy to optimize

Optimizing concave function – Gradient ascent

- Conditional likelihood for Logistic Regression is concave $\rightarrow$ Find optimum with gradient ascent



Gradient: $$\nabla_{\mathbf{w}} l(\mathbf{w}) = [\frac{\partial l(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial l(\mathbf{w})}{\partial w_n}]'$$

Learning rate, η>0

Update rule: $$\Delta \mathbf{w} = \eta \nabla_{\mathbf{w}} l(\mathbf{w})$$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \frac{\partial l(\mathbf{w})}{\partial w_i}$$

- Gradient ascent is simplest of optimization approaches
  - e.g., Conjugate gradient ascent much better (see reading)

Maximize Conditional Log Likelihood: Gradient ascent

$$l(\mathbf{w}) = \sum_j y^j (w_0 + \sum_i^n w_i x_i^j) - \ln(1 + exp(w_0 + \sum_i^n w_i x_i^j))$$

Gradient Descent for LR

Gradient ascent algorithm: iterate until change < ε

$$w_0^{(t+1)} \leftarrow w_0^{(t)} + \eta \sum_j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w})]$$

For $i = 1 \ldots n$,

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w})]$$

repeat

That's all M(C)LE. How about MAP?

$$p(\mathbf{w} \mid Y, \mathbf{X}) \quad \propto \quad P(Y \mid \mathbf{X}, \mathbf{w})p(\mathbf{w})$$

- One common approach is to define priors on **w**
  - Normal distribution, zero mean, identity covariance
  - "Pushes" parameters towards zero
- Corresponds to ***Regularization***
  - Helps avoid very large weights and overfitting
  - More on this later in the semester
- MAP estimate

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \ln \left[p(\mathbf{w}) \prod_{j=1}^{N} P(y^j \mid \mathbf{x}^j, \mathbf{w}) \right]$$

M(C)AP as Regularization

$$\ln \left[p(\mathbf{w}) \prod_{j=1}^{N} P(y^j \mid \mathbf{x}^j, \mathbf{w}) \right]$$

$$p(\mathbf{w}) = \prod_i \frac{1}{\kappa\sqrt{2\pi}} \, e^{\frac{-w_i^2}{2\kappa^2}}$$

Penalizes high weights, also applicable in linear regression

Gradient of M(C)AP

$$\frac{\partial}{\partial w_i} \ln \left[p(\mathbf{w}) \prod_{j=1}^{N} P(y^j \mid \mathbf{x}^j, \mathbf{w}) \right]$$

$$p(\mathbf{w}) = \prod_i \frac{1}{\kappa\sqrt{2\pi}} \; e^{\frac{-w_i^2}{2\kappa^2}}$$

MLE vs MAP

- Maximum conditional likelihood estimate

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \ln \left[\prod_{j=1}^{N} P(y^j \mid \mathbf{x}^j, \mathbf{w}) \right]$$

$$\boxed{w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w})]}$$

- Maximum conditional a posteriori estimate

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \ln \left[p(\mathbf{w}) \prod_{j=1}^{N} P(y^j \mid \mathbf{x}^j, \mathbf{w}) \right]$$

$$\boxed{w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w})] \right\}}$$

Naïve Bayes vs Logistic Regression

Consider Y boolean, X_i continuous, $X=<X_1 \ldots X_n>$

Number of parameters:

- NB: 4n +1
- LR: n+1

Estimation method:

- NB parameter estimates are uncoupled
- LR parameter estimates are coupled

G. Naïve Bayes vs. Logistic Regression 1

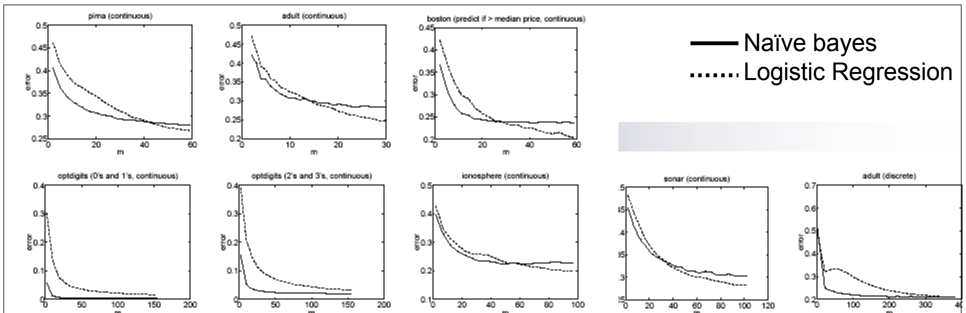
[Ng & Jordan, 2002]

- Generative and Discriminative classifiers
- Asymptotic comparison (# training examples → infinity)
  - when model correct
    - GNB, LR produce identical classifiers
  - when model incorrect
    - LR is less biased – does not assume conditional independence
      - **therefore LR expected to outperform GNB**

G. Naïve Bayes vs. Logistic Regression 2

[Ng & Jordan, 2002]

- Generative and Discriminative classifiers
- Non-asymptotic analysis
  - convergence rate of parameter estimates, n = # of attributes in X
    - Size of training data to get close to infinite data solution
    - GNB needs *O*(log n) samples
    - LR needs *O*(n) samples
  - **GNB converges more quickly to its (perhaps less helpful) asymptotic estimates**



Naïve bayes

Logistic Regression

Some experiments from UCI data sets

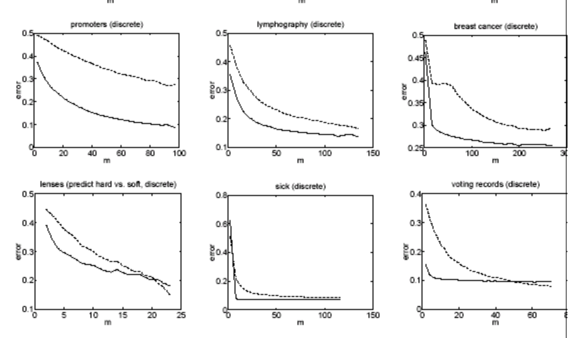


Figure 1: Results of 15 experiments on datasets from the UCI Machine Learnin repository. Plots are of generalization error vs. m (averaged over 1000 rando train/test splits). Dashed line is logistic regression; solid line is naive Bayes.

What you should know about Logistic Regression (LR)

- Gaussian Naïve Bayes with class-independent variances representationally equivalent to LR
  - Solution differs because of objective (loss) function
- In general, NB and LR make different assumptions
  - NB: Features independent given class $\rightarrow$ assumption on P(**X**|Y)
  - LR: Functional form of P(Y|**X**), no assumption on P(**X**|Y)
- LR is a linear classifier
  - decision rule is a hyperplane
- LR optimized by conditional likelihood
  - no closed-form solution
  - concave $\rightarrow$ global optimum with gradient ascent
  - Maximum conditional a posteriori corresponds to regularization
- Convergence rates
  - GNB (usually) needs less data
  - LR (usually) gets to better solutions in the limit