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

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

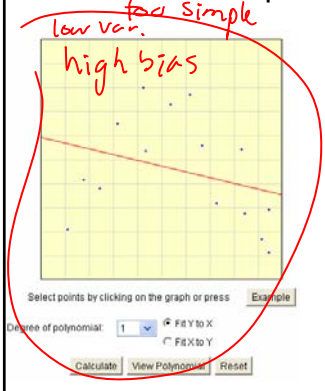
September 19th, 2007

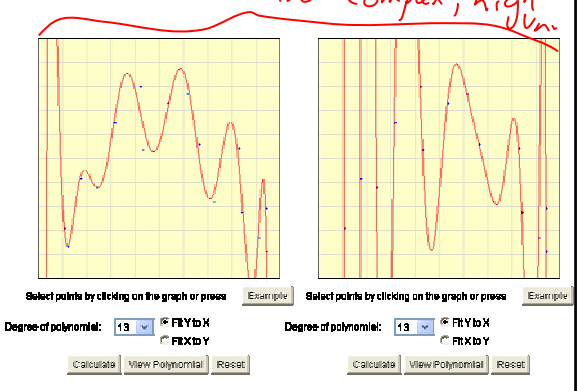
Bias-Variance Tradeoff

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

too simple
low var.
high bias

too complex, high var.





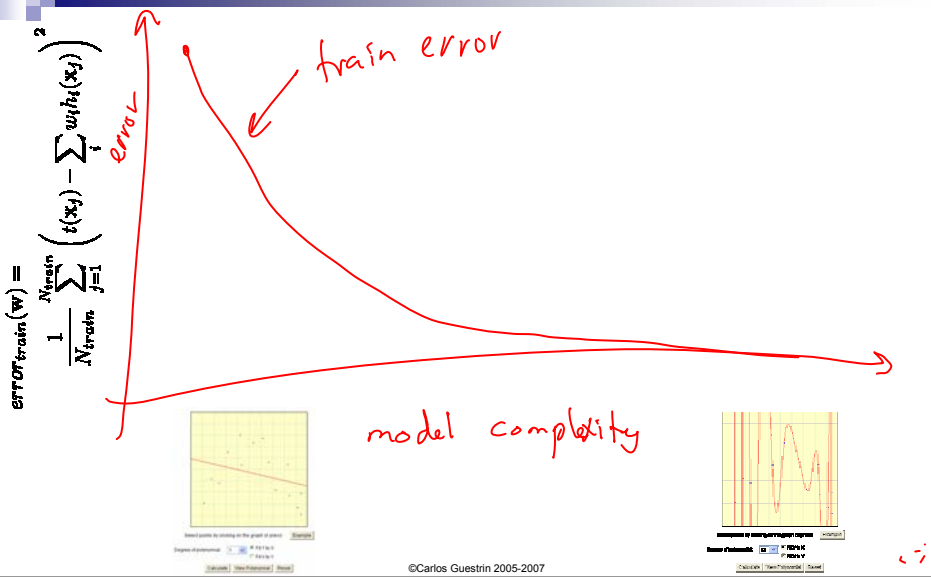
Training set error

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

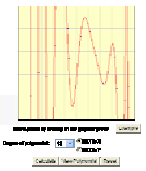
- Given a dataset (Training data)
- Choose a loss function
  - e.g., squared error (L_2) for regression
- **Training set error:** For a particular set of parameters, loss function on training data:

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

Training set error as a function of model complexity



Prediction error



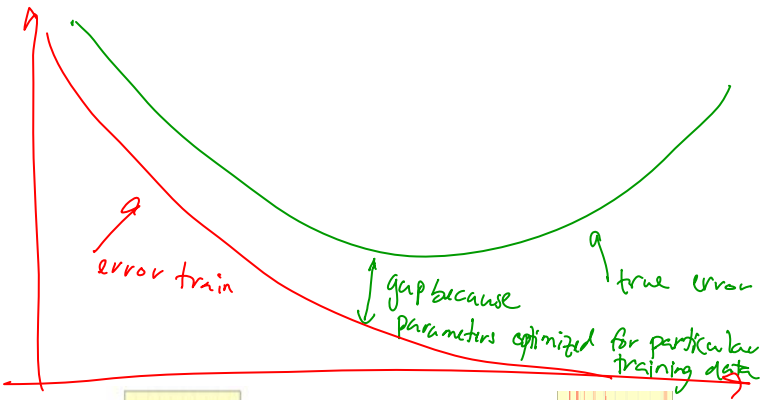
- Training set error can be poor measure of "quality" of solution
- **Prediction error:** We really care about error over all possible input points, not just training data:

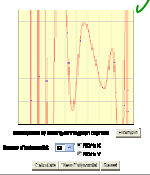
$$error_{true}(\mathbf{w}) = E_{\mathbf{x}}\left[\left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2\right]$$
$$= \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x}) d\mathbf{x}$$

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**

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

- 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

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

$= error_{test}(\mathbf{w})$

Why training set error doesn't approximate prediction error?

- Sampling approximation of prediction error:

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

- Training error :

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

- 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:

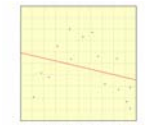
$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

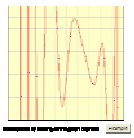
Test set error as a function of model complexity

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

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

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





Overfitting

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

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

- w was better in train than w', but worse in true error,
- w' worse in train, but better in true than 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:

$$P(|\ \hat{\theta} - \theta^* \ | \geq \epsilon) \ \leq \ 2e^{-2N\epsilon^2}$$

- 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

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

← gold standard

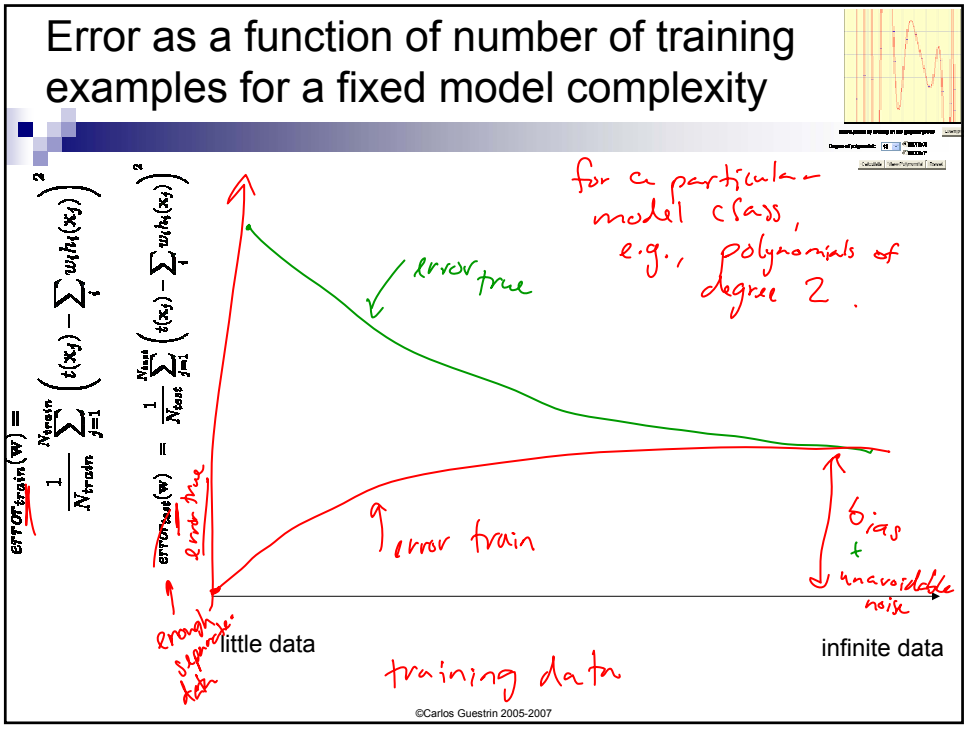
$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

← what optimize, but optimistic

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

← estimate of $error_{true}$

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



Error estimators

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:

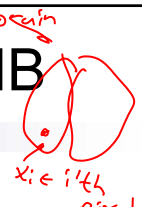
$$P(X_1, X_2|Y) = P(X_1|X_2, Y)P(X_2|Y)$$

  - More generally:

$$P(X_1 \ldots X_n|Y) = \prod_i P(X_i|Y)$$

- NB Classifier: $P(Y, X_1 \ldots X_n) = P(Y) \prod_{i=1}^{n} P(X_i|Y)$

MLE for the parameters of NB



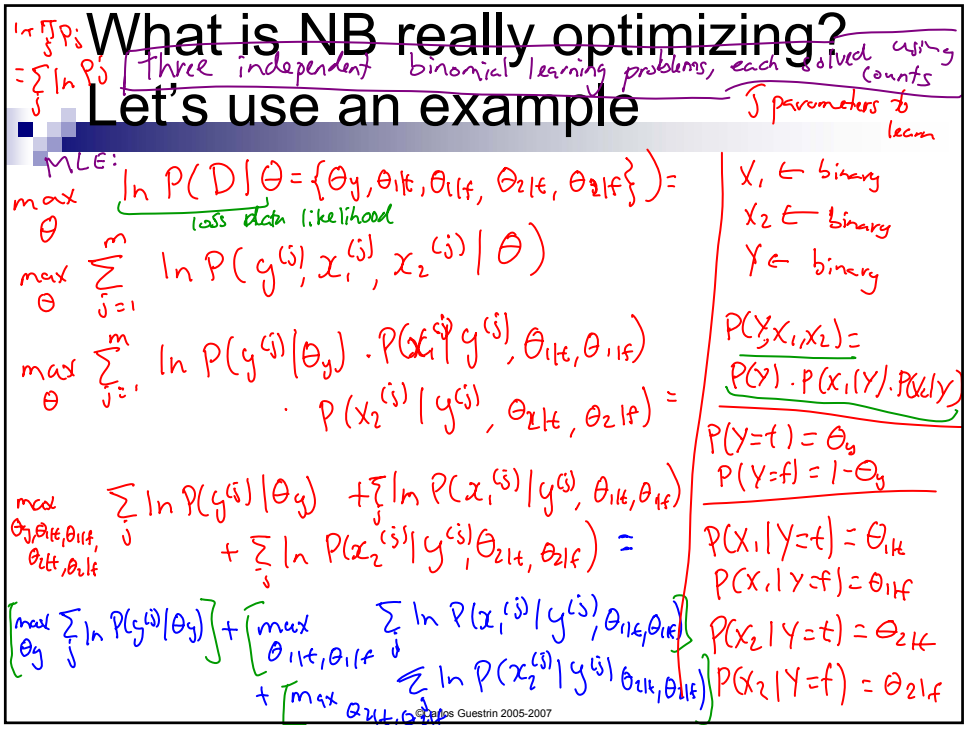
- Given dataset
  - Count(A=a,B=b) ← number of examples where A=a and B=b

total data : m data point

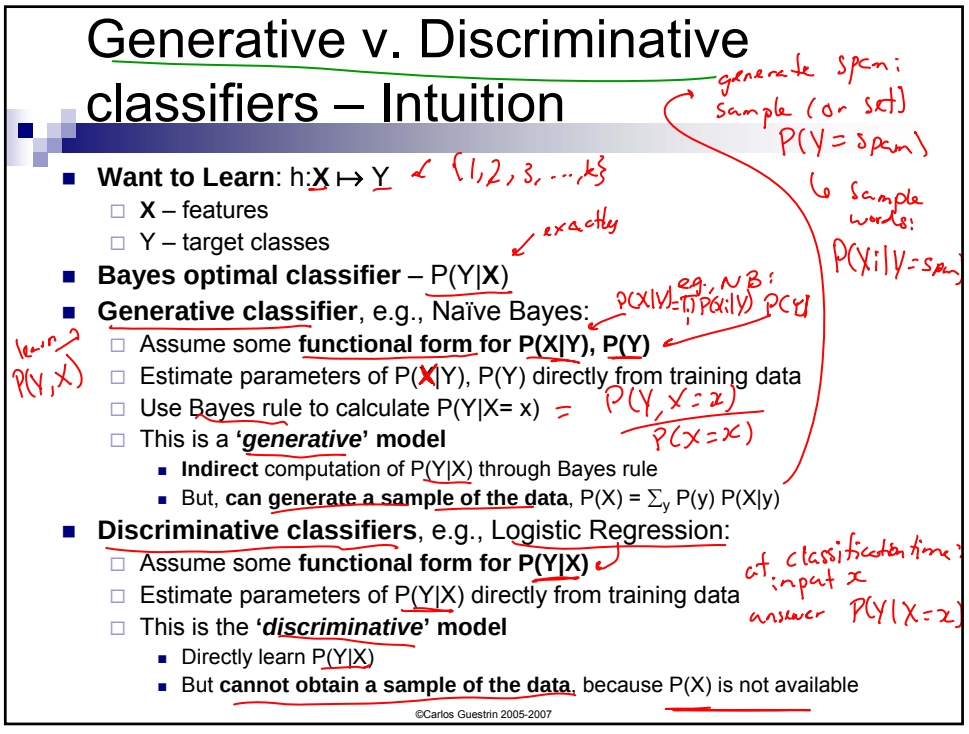
- MLE for NB, simply:
  - Prior: P(Y=y) = $\frac{Count(Y=y)}{m}$
  - Likelihood: $P(X_i=x_i|Y=y) = \frac{Count(X_i=x_i, Y=y)}{Count(Y=y)}$

e.g., ith pixel is on given read a verb

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



Generative v. Discriminative classifiers – Intuition



- **Want to Learn**: h: **X** ↦ 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) = Σ_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