SVMs, Duality and the Kernel Trick (cont.)

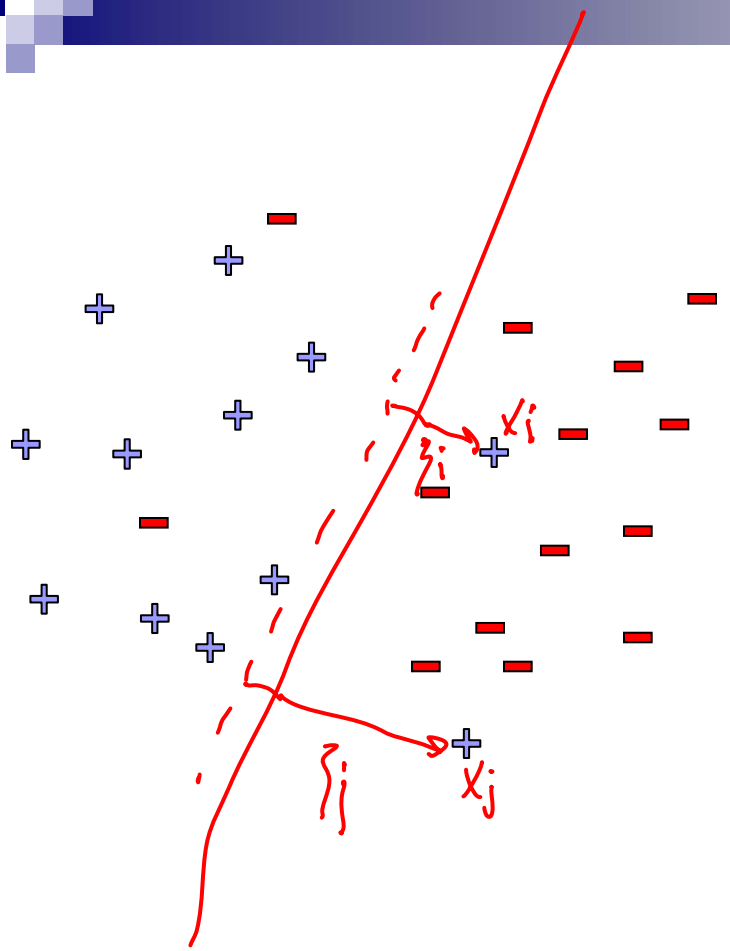
Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

February 28th, 2007

SVMs reminder



$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w} + C\sum_j \xi_j$$

$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \quad \forall j$$

$$\xi_j \geq 0, \quad \forall j$$

slack penalty

each point must have margin ≥ 1

Dual SVM formulation – the non-separable case

one α_i for each data point

maximize$_\alpha$ $\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j$

irrelevant $\alpha_i = 0$

otherwise $\alpha_i > 0$

$\sum_i \alpha_i y_i = 0$

$C \geq \alpha_i \geq 0$ $\forall i$

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

linear combination

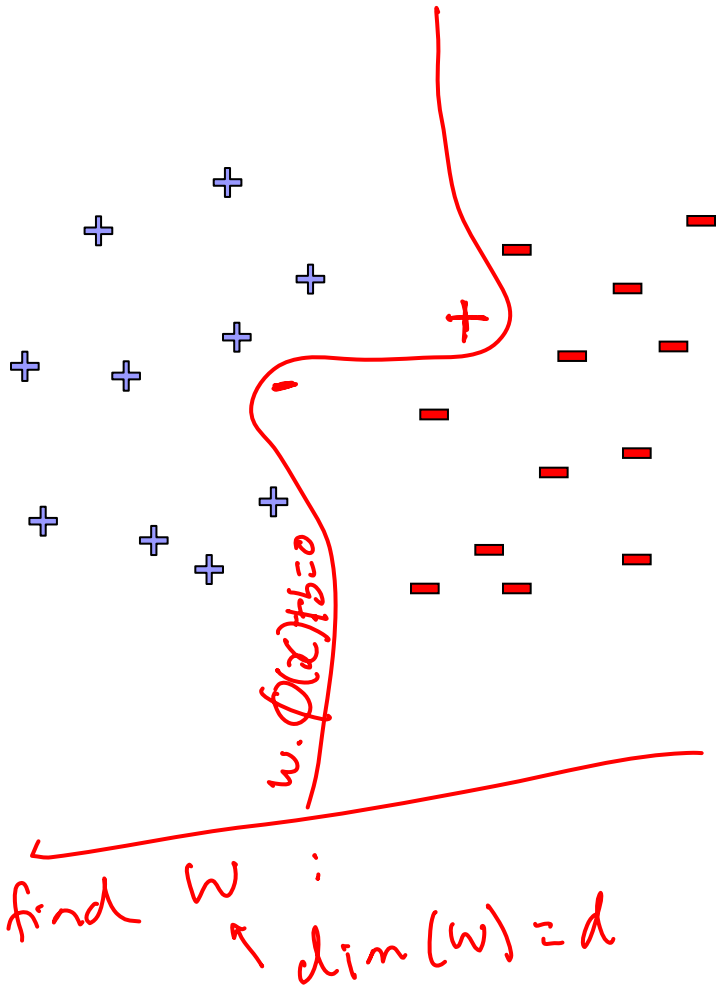
$$b = y_k - \mathbf{w} . \mathbf{x}_k$$

for any k where $C > \alpha_k > 0$

compared to separable case: only difference $\alpha_i \leq C$ $\forall i$

intuitively, don't give me alphas that are too large

Reminder from last time: What if the data is not linearly separable?



Use features of features of features of features….

$$\Phi(\mathbf{x}) : R^m \mapsto F$$

$\Phi(x) =$

x_2

$x_1 x_2$

x_1^2

$\sin x_1$

e^{-x_1/x_2}

d

mapping to high dim features

Feature space can get really large really quickly!

Dual formulation only depends on dot-products, not on **w**!

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

$x_i = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{pmatrix}$ $x_j = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_d \end{pmatrix}$

$x_i \cdot x_j = v_1 u_1 + v_2 u_2 + \cdots + v_d u_d$

$\phi(x) = \begin{pmatrix} v_1 \\ v_2 \\ v_1 v_2 \\ v_1^2 \\ v_2^2 \\ \vdots \end{pmatrix}$

$\phi(x_i) \cdot \phi(x_j) = K(x_i, x_j)$

how many terms?

m^2

m = # data points

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$
$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

Dot-product of polynomials

$u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$ $v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = \text{polynomials of degree } d$$

$d=1 \quad \phi(u) = u$

$\phi(u) \cdot \phi(v) = u \cdot v = u_1 v_1 + u_2 v_2$

$d=2 \quad \phi(u) = \begin{pmatrix} u_1^2 \\ u_1 u_2 \\ u_2 u_1 \\ u_2^2 \end{pmatrix}$

$\phi(u) \cdot \phi(v) =$

$u_1^2 \cdot v_1^2 + 2 u_1 u_2 \cdot v_1 v_2 + u_2^2 \cdot v_2^2$

$= (u_1 v_1 + u_2 v_2)^2 = (u \cdot v)^2$

degree $= d$ $\quad K(u,v) = \phi(u) \cdot \phi(v) = (u \cdot v)^d$

$= O(d)$ multiplications

Finally: the "kernel trick"!

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0$$

$$\mathbf{w} = \sum_i \alpha_i y_i \Phi(\mathbf{x}_i)$$

$$b = y_k - \mathbf{w}.\Phi(\mathbf{x}_k)$$

for any k where $C > \alpha_k > 0$

- Never represent features explicitly
  - Compute dot products in closed form
- Constant-time high-dimensional dot-products for many classes of features

- Very interesting theory – Reproducing Kernel Hilbert Spaces
  - Not covered in detail in 10701/15781, more in 10702

Common kernels

- Polynomials of degree d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d$$

- Polynomials of degree up to d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d$$

- Gaussian kernels

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u} - \mathbf{v}\|_2^2}{2\sigma^2}\right)$$

- Sigmoid

$$K(\mathbf{u}, \mathbf{v}) = \tanh(\eta \mathbf{u} \cdot \mathbf{v} + \nu)$$

Overfitting?

- Huge feature space with kernels, what about overfitting???
  - Maximizing margin leads to sparse set of support vectors
  - Some interesting theory says that SVMs search for simple hypothesis with large margin
  - Often robust to overfitting

What about at classification time

- For a new input **x**, if we need to represent $\Phi(\mathbf{x})$, we are in trouble!
- Recall classifier: sign($\mathbf{w}.\Phi(\mathbf{x})$+b)
- Using kernels we are cool!

$$K(\mathbf{u},\mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v})$$

$$\mathbf{w} = \sum_i \alpha_i y_i \Phi(\mathbf{x}_i)$$

$$b = y_k - \mathbf{w}.\Phi(\mathbf{x}_k)$$

for any k where $C > \alpha_k > 0$

SVMs with kernels

- Choose a set of features and kernel function
- Solve dual problem to obtain support vectors α_i
- At classification time, compute:

$$\mathbf{w} \cdot \Phi(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$$

$$b = y_k - \sum_i \alpha_i y_i K(\mathbf{x}_k, \mathbf{x}_i)$$

for any k where $C > \alpha_k > 0$

Classify as

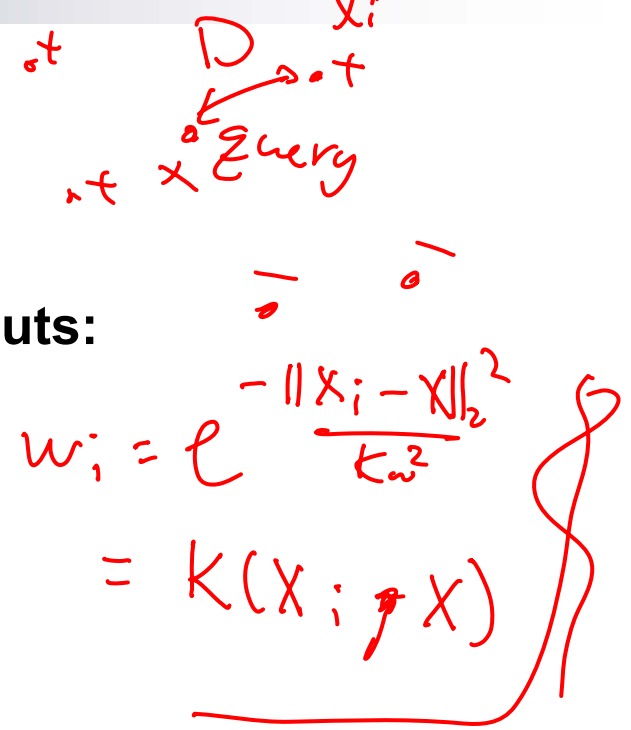
$$sign\left(\mathbf{w} \cdot \Phi(\mathbf{x}) + b\right)$$

Remember kernel regression

Remember kernel regression???

1. $w_i = exp(-D(x_i, query)^2 / K_w^2)$
2. *How to fit with the local points?*

   Predict the weighted average of the outputs:
   predict = $\Sigma w_i y_i / \Sigma w_i$



$$\hat{y}(x) = \frac{\sum_i w_i y_i}{\sum_i w_i}$$

$$\hat{y}(x) = \frac{\sum_i K(x, x_i) y_i}{\sum K(x, x_i)}$$

SVMs v. Kernel Regression

SVMs

$$sign\left(\mathbf{w}\cdot\Phi(\mathbf{x})+b\right)$$

or

$$sign\left(\sum_i \alpha_i y_i K(\mathbf{x},\mathbf{x}_i)+b\right)$$

if I say $b=0$ & $\forall i\ \alpha_i = 1$

↳ then I get kernel regression

⇒ but, SVM optimizing over α's s.t. $\alpha_j \geq 0$

Kernel Regression

$y_i \in \{+1, -1\}$

Classification

$$sign\left(\frac{\sum_i y_i K(\mathbf{x},\mathbf{x}_i)}{\sum_j K(\mathbf{x},\mathbf{x}_j)}\right)$$

$$= \alpha(x)$$

$$\frac{1}{\sum_j k(x,x_j)}$$

$$= sign\left(\sum_i \alpha(x)\cdot y_i\, K(x,x_i)\right)$$

$$\equiv \alpha(x) \geq 0$$

$$sign\left(\sum_i y_i\, K(x,x_i)\right)$$

SVMs v. Kernel Regression

SVMs

$$sign\left(\mathbf{w}\cdot\Phi(\mathbf{x})+b\right)$$

or

$$sign$$

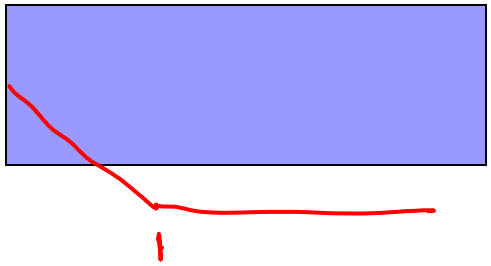
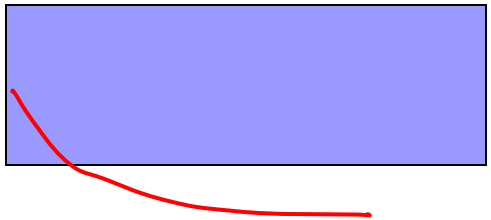
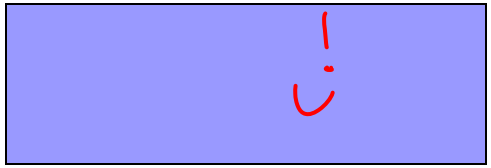
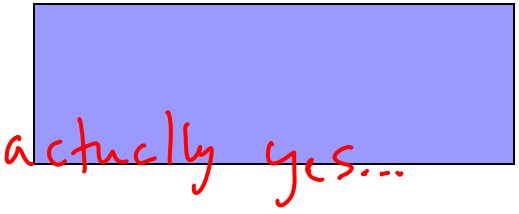
Kernel Regression

$$sign\left(\frac{\sum_i y_i K(\mathbf{x},\mathbf{x}_i)}{\sum_i K(\mathbf{x},\mathbf{x}_i)}\right)$$

Differences:

- SVMs:
  - Learn weights α_i (and bandwidth)
  - Often sparse solution
- KR:
  - Fixed "weights", learn bandwidth
  - Solution may not be sparse
  - Much simpler to implement

What's the difference between SVMs and Logistic Regression?

	SVMs	Logistic Regression
Loss function		
High dimensional features with kernels		

Kernels in logistic regression

features $\in \Phi(x)$

$$P(Y = +1 \mid x, \mathbf{w}, b) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \Phi(\mathbf{x}) + b)}}$$

- Define weights in terms of support vectors:

$$\mathbf{w} = \sum_i \alpha_i \Phi(\mathbf{x}_i)$$

← representer theorem

$$P(Y = 1 \mid x, \mathbf{w}) = \frac{1}{1 + e^{-(\sum_i \alpha_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) + b)}}$$

$$= \frac{1}{1 + e^{-(\sum_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b)}}$$

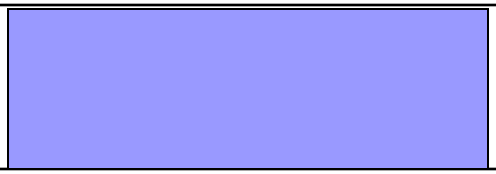
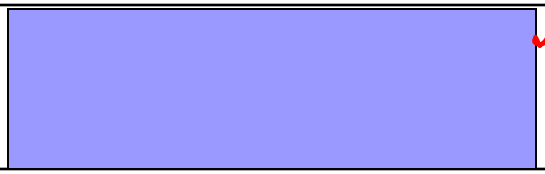
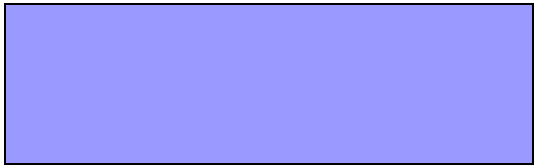
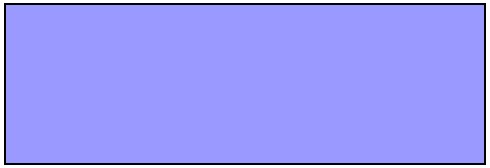
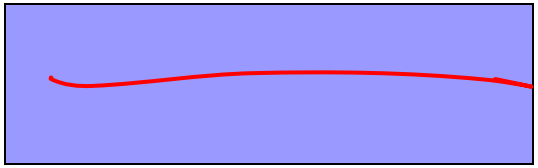
$K(x, x_i) = \Phi(x_i) \cdot \Phi(x)$

- Derive simple gradient descent rule on α_i, b

must learn α_i's & b ⇒ take derivative & do gradient descent

still convex, because (*) is linear in α_i's & b

What's the difference between SVMs and Logistic Regression? (Revisited)

	SVMs	**Logistic Regression**
Loss function	Hinge loss	Log-loss
High dimensional features with kernels	Yes!!	Yes!!
			
			

What you need to know

- Dual SVM formulation
  - How it's derived (intuition)
- The kernel trick
- Derive polynomial kernel
- Common kernels
- Kernelized logistic regression
- Differences between SVMs and logistic regression

D SVMs with kernels v. kernel regression

Announcements

- Class projects out next week

Learning Theory

PAC-learning, VC Dimension and Margin-based Bounds

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

February 28th, 2007

What now…

- We have explored **many** ways of learning from data
- But…
  - How good is our classifier, really?
  - How much data do I need to make it “good enough”?

A simple setting…

- Classification
  - m data points
  - **Finite** number of possible hypothesis (e.g., dec. trees of depth d)
- A learner finds a hypothesis *h* that is **consistent** with training data
  - Gets zero error in training – $error_{train}(h) = 0$
- What is the probability that *h* has more than ε true error?
  - $error_{true}(h) \geq \varepsilon$

How likely is a bad hypothesis to get *m* data points right?

- Hypothesis *h* that is **consistent** with training data ⇒ got *m* i.i.d. points right
- Prob. *h* with $error_{true}(h) \geq \varepsilon$ gets one data point right

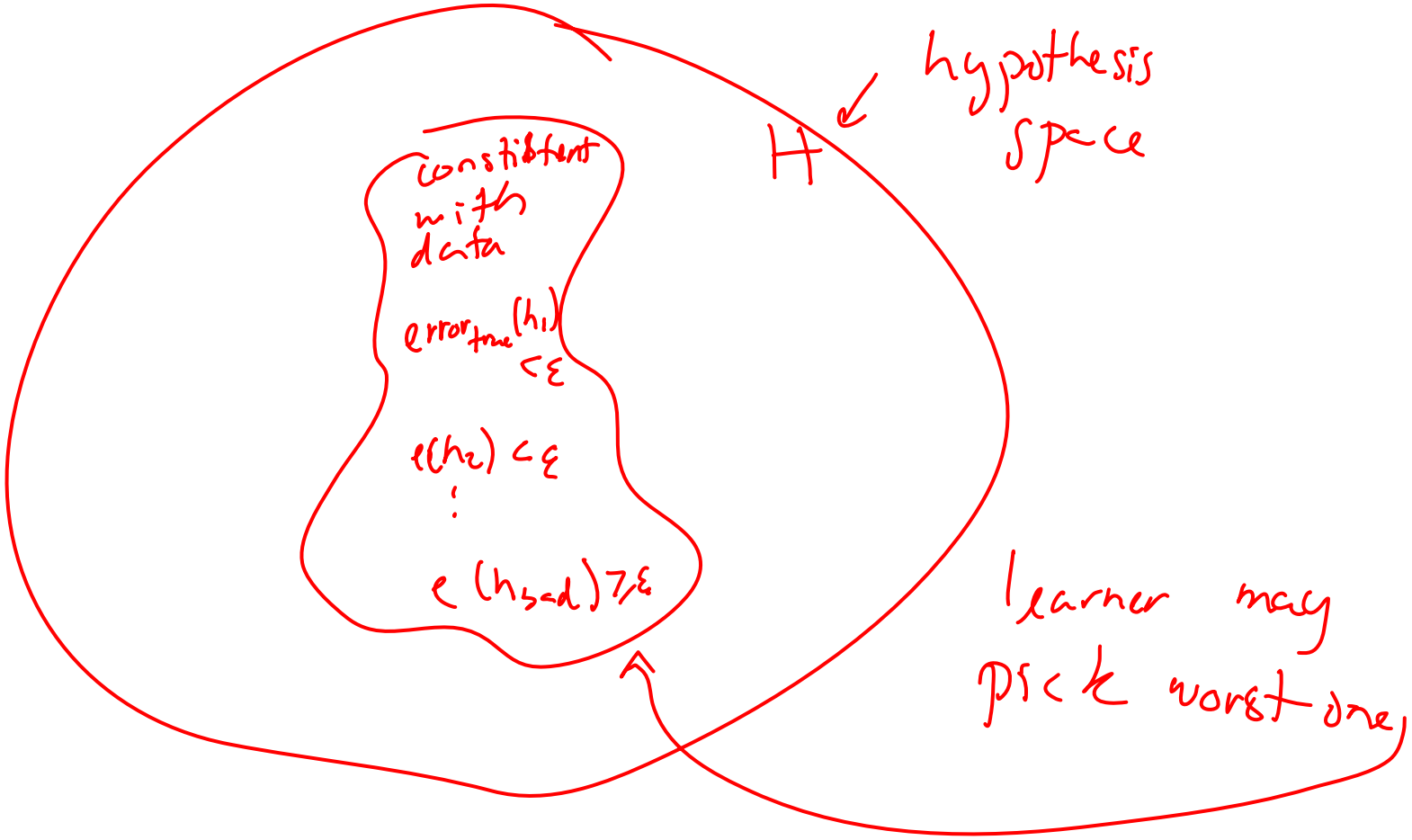
$P(h \text{ gets } 1 \text{ point right}) \leq 1-\varepsilon$

- Prob. *h* with $error_{true}(h) \geq \varepsilon$ gets *m* data points right

$P(h \text{ gets } m \text{ i.i.d. points right}) \leq (1-\varepsilon)^m$

have a bad h ($error_{true}(h) \geq \varepsilon$), then $P(\text{gets } m \text{ points right})$ is exponentially small

But there are many possible hypothesis that are consistent with training data



How likely is learner to pick a bad hypothesis

- Prob. *h* with $error_{true}(h) \geq \varepsilon$ gets *m* data points right

$P(\text{bad } h \text{ gets } m \text{ points right}) \leq (1-\varepsilon)^m$

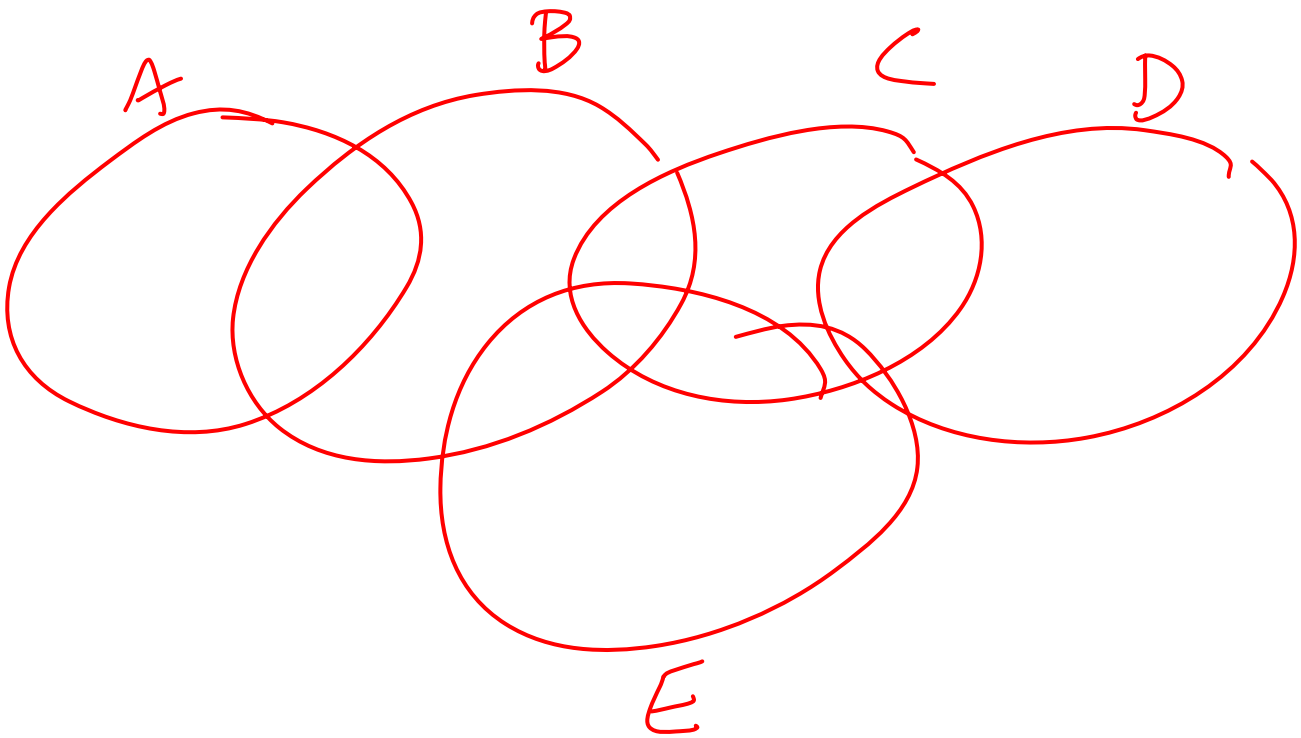
- There are *k* hypothesis consistent with data $= \{h_1 \dots h_k\}$
  - How likely is learner to pick a bad one?

$P(\exists \text{ hypothesis consistent with data, } \exists h \text{ in there s.t. } error_{true}(h) \geq \varepsilon)$?

$= P(error_t(h_1) \geq \varepsilon \vee e_t(h_2) \geq \varepsilon \vee \dots e_t(h_n) \geq \varepsilon)$

Union bound

- P(A or B or C or D or …) $\leq P(A) + P(B) + P(C) + \cdots$



How likely is learner to pick a bad hypothesis

$(1-\varepsilon)^m \le (e^{-\varepsilon})^m = e^{-\varepsilon m}$

- Prob. h_i with $error_{true}(h) \ge \varepsilon$ gets m data points right

$P(error_t(h_i) \ge \varepsilon \text{ \& consistent with } m \text{ data points}) \le (1-\varepsilon)^m$

- There are k hypothesis consistent with data
  - How likely is learner to pick a bad one?

$P(error_t(h_1) \ge \varepsilon \text{ \& } h_1 \text{ consistent with } m \text{ OR } error_t(h_2) \ge \varepsilon \text{ \& consistent OR } \ldots \text{ OR } error_t(h_k) \ge \varepsilon \text{ \& consistent})$

$\le \sum_i P(error_t(h_i) \ge \varepsilon \text{ \& consistent with } m \text{ data points})$

$\le k(1-\varepsilon)^m$

$\le |H|(1-\varepsilon)^m \le |H| e^{-\varepsilon m}$

$1-\varepsilon \le e^{-\varepsilon}$ for $\varepsilon \ge 0$

simplify eqns.

Review: Generalization error in finite hypothesis spaces [Haussler '88]

- ***Theorem***: Hypothesis space *H* finite, dataset *D* with *m* i.i.d. samples, $0 < \varepsilon < 1$: for any learned hypothesis *h* that is consistent on the training data:

$$P(\text{error}_{true}(h) \geq \epsilon) \leq |H|e^{-m\epsilon}$$

as $m \to$ increases $\Rightarrow$ Prob. make a bad decision decrease exponentially fast

as $|H| \to$ increases $\Rightarrow$ Chances of making a bad decision increase linearly with $|H|$

Using a PAC bound

I want: $error_{true}(h) \leq \epsilon$ guarantee with high prob.
guarantee with prob. $\geq 1-\delta$

PAC: probably Approximately Correct

- Typically, 2 use cases: $P(\text{error}_{true}(h) > \epsilon) \leq |H|e^{-m\epsilon}$
  - 1: Pick ε and δ, give you m
  - 2: Pick m and δ, give you ε

1: e.g., $\varepsilon \leq 0.1$

$1-\delta \geq 0.95$ I am right

$\delta \geq |H| e^{-m\varepsilon}$

$\ln \delta \geq \ln|H| - m\varepsilon$

$m \geq \frac{1}{\varepsilon}\left(\ln|H| + \ln\frac{1}{\delta}\right)$

(# points you need)

Review: Generalization error in finite hypothesis spaces [Haussler '88]

- ***Theorem***: Hypothesis space *H* finite, dataset *D* with *m* i.i.d. samples, $0 < \varepsilon < 1$: for any learned hypothesis *h* that is consistent on the training data:

$$P(\text{error}_{true}(h) > \epsilon) \leq |H|e^{-m\epsilon}$$

Even if *h* makes zero errors in training data, may make errors in test

Limitations of Haussler ‘88 bound

$$P(\text{error}_{true}(h) > \epsilon) \leq |H| e^{-m\epsilon}$$

- Consistent classifier

- Size of hypothesis space

What if our classifier does not have zero error on the training data?

- A learner with zero training errors may make mistakes in test set
- What about a learner with $error_{train}(h)$ in training set?

Simpler question: What's the expected error of a hypothesis?

- The error of a hypothesis is like estimating the parameter of a coin!

- Chernoff bound: for *m* i.i.d. coin flips, $x_1,\ldots,x_m$, where $x_i \in \{0,1\}$. For $0<\varepsilon<1$:

$$P\left(\theta - \frac{1}{m}\sum_i x_i > \epsilon\right) \leq e^{-2m\epsilon^2}$$

Using Chernoff bound to estimate error of a single hypothesis

$$P\left(\theta - \frac{1}{m}\sum_i x_i > \epsilon\right) \leq e^{-2m\epsilon^2}$$

But we are comparing many hypothesis: **Union bound**

For each hypothesis h_i:

$$P\left(\text{error}_{true}(h_i) - \text{error}_{train}(h_i) > \epsilon\right) \leq e^{-2m\epsilon^2}$$

What if I am comparing two hypothesis, h_1 and h_2?

Generalization bound for |H| hypothesis

- ***Theorem***: Hypothesis space *H* finite, dataset *D* with *m* i.i.d. samples, $0 < \varepsilon < 1$: for any learned hypothesis *h*:

$$P\left(\text{error}_{true}(h) - \text{error}_{train}(h) > \epsilon\right) \leq |H| e^{-2m\epsilon^2}$$

PAC bound and Bias-Variance tradeoff

$$P\left(\text{error}_{true}(h) - \text{error}_{train}(h) > \epsilon\right) \leq |H| e^{-2m\epsilon^2}$$

or, after moving some terms around, with probability at least 1-δ:

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{\ln |H| + \ln \frac{1}{\delta}}{2m}}$$

- **Important: PAC bound holds for all *h*,**

but doesn't guarantee that algorithm finds best *h*!!!

What about the size of the hypothesis space?

$$m \geq \frac{1}{2\epsilon^2}\left(\ln|H| + \ln\frac{1}{\delta}\right)$$

- How large is the hypothesis space?

Boolean formulas with *n* binary features

$$m \geq \frac{1}{2\epsilon^2}\left(\ln|H| + \ln\frac{1}{\delta}\right)$$

Number of decision trees of depth k

$$m \geq \frac{1}{2\epsilon^2}\left(\ln|H| + \ln\frac{1}{\delta}\right)$$

Recursive solution

Given *n* attributes

H_k = Number of decision trees of depth k

H_0 =2

H_{k+1} = (#choices of root attribute) *
(# possible left subtrees) *
(# possible right subtrees)

= n * H_k * H_k

Write $L_k = \log_2 H_k$

$L_0 = 1$

$L_{k+1} = \log_2 n + 2L_k$

So $L_k = (2^k-1)(1+\log_2 n) +1$

PAC bound for decision trees of depth k

$$m \geq \frac{\ln 2}{2\epsilon^2}\left((2^k - 1)(1 + \log_2 n) + 1 + \ln\frac{1}{\delta}\right)$$

- Bad!!!
  - Number of points is exponential in depth!

- But, for *m* data points, decision tree can't get too big…

Number of leaves never more than number data points

Number of decision trees with k leaves

$$m \geq \frac{1}{2\epsilon^2}\left(\ln|H| + \ln\frac{1}{\delta}\right)$$

H_k = Number of decision trees with k leaves

H_0 =2

$$H_{k+1} = n\sum_{i=1}^{k} H_i H_{k+1-i}$$

Loose bound:

$$H_k = n^{k-1}(k+1)^{2k-1}$$

Reminder:

$$|\text{DTs depth } k| = 2 * (2n)^{2^k-1}$$

PAC bound for decision trees with k leaves – Bias-Variance revisited

$$H_k = n^{k-1}(k+1)^{2k-1}$$

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{\ln |H| + \ln \frac{1}{\delta}}{2m}}$$

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{(k-1)\ln n + (2k-1)\ln(k+1) + \ln \frac{1}{\delta}}{2m}}$$

What did we learn from decision trees?

- Bias-Variance tradeoff formalized

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{(k-1)\ln n + (2k-1)\ln(k+1) + \ln\frac{1}{\delta}}{2m}}$$

- Moral of the story:

  Complexity of learning not measured in terms of size hypothesis space, but in maximum *number of points* that allows consistent classification

  - Complexity *m* – no bias, lots of variance
  - Lower than *m* – some bias, less variance

What about continuous hypothesis spaces?

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{\ln |H| + \ln \frac{1}{\delta}}{2m}}$$

- Continuous hypothesis space:
  - |H| = 1
  - Infinite variance???

- **As with decision trees, only care about the maximum number of points that can be classified exactly!**

How many points can a linear boundary classify exactly? (1-D)

How many points can a linear boundary classify exactly? (2-D)

How many points can a linear boundary classify exactly? (d-D)

PAC bound using VC dimension

- **Number of training points that can be classified exactly is VC dimension!!!**
  - **Measures relevant size of hypothesis space, as with decision trees with k leaves**

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{VC(H)\left(\ln \frac{2m}{VC(H)} + 1\right) + \ln \frac{4}{\delta}}{m}}$$

Shattering a set of points

Definition: a **dichotomy** of a set S is a partition of S into two disjoint subsets.

Definition: a set of instances S is **shattered** by hypothesis space H if and only if for every dichotomy of S there exists some hypothesis in H consistent with this dichotomy.

VC dimension

Definition: The **Vapnik-Chervonenkis dimension**, $VC(H)$, of hypothesis space H defined over instance space X is the size of the largest finite subset of X shattered by H. If arbitrarily large finite sets of X can be shattered by H, then $VC(H) \equiv \infty$.

Examples of VC dimension

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{VC(H)\left(\ln\frac{2m}{VC(H)}+1\right)+\ln\frac{4}{\delta}}{m}}$$

- Linear classifiers:
  - VC(H) = d+1, for *d* features plus constant term *b*
- Neural networks
  - VC(H) = #parameters
  - Local minima means NNs will probably not find best parameters
- 1-Nearest neighbor?

PAC bound for SVMs

- SVMs use a linear classifier
  - For *d* features, VC(H) = d+1:

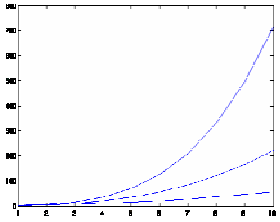
$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{(d+1)\left(\ln \frac{2m}{d+1} + 1\right) + \ln \frac{4}{\delta}}{m}}$$

VC dimension and SVMs: Problems!!!

Doesn't take margin into account

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{(d+1)\left(\ln\frac{2m}{d+1}+1\right)+\ln\frac{4}{\delta}}{m}}$$

- What about kernels?
  - Polynomials: num. features grows really fast = Bad bound



$$\text{num. terms} = \binom{p+n-1}{p} = \frac{(p+n-1)!}{p!(n-1)!}$$

n – input features
p – degree of polynomial

  - Gaussian kernels can classify any set of points exactly

Margin-based VC dimension

- H: Class of linear classifiers: $\mathbf{w}.\Phi(\mathbf{x})$ (b=0)
  - Canonical form: $\min_j |\mathbf{w}.\Phi(\mathbf{x}_j)| = 1$
- VC(H) = R^2 **w.w**
  - Doesn't depend on number of features!!!
  - $R^2 = \max_j \Phi(\mathbf{x}_j).\Phi(\mathbf{x}_j)$ – magnitude of data
  - R^2 is bounded even for Gaussian kernels ! bounded VC dimension
- Large margin, low **w.w**, low VC dimension – Very cool!

Applying margin VC to SVMs?

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{VC(H)\left(\ln\frac{2m}{VC(H)}+1\right)+\ln\frac{4}{\delta}}{m}}$$

- VC(H) = R^2 **w.w**
  - $R^2 = \max_j \Phi(\mathbf{x}_j).\Phi(\mathbf{x}_j)$ – magnitude of data, doesn't depend on choice of **w**
- SVMs minimize **w.w**

- **SVMs minimize VC dimension to get best bound?**
- **Not quite right: ☹**
  - **Bound assumes VC dimension chosen before looking at data**
  - **Would require union bound over infinite number of possible VC dimensions…**
  - **But, it can be fixed!**

Structural risk minimization theorem

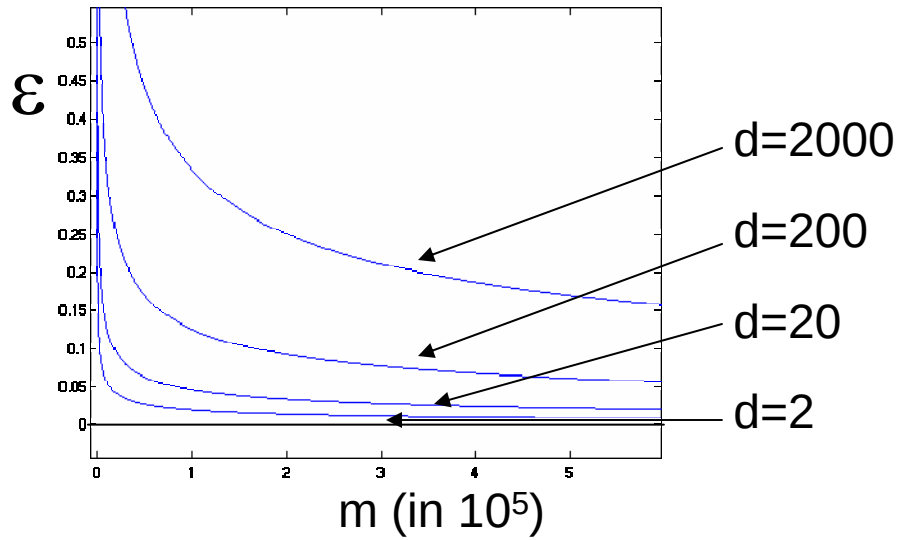
$$\text{error}_{true}(h) \leq \text{error}^{\gamma}_{train}(h) + C\sqrt{\frac{\frac{R^2}{\gamma^2}\ln m + \ln\frac{1}{\delta}}{m}}$$

$$\text{error}^{\gamma}_{train}(h) = \text{num. points with margin} < \gamma$$

- For a family of hyperplanes with margin $\gamma>0$
  - **w.w** · 1
- SVMs maximize margin γ + hinge loss
  - Optimize tradeoff training error (bias) versus margin γ (variance)

Reality check – Bounds are loose

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \underbrace{\sqrt{\frac{(d+1)\left(\ln\frac{2m}{d+1}+1\right)+\ln\frac{4}{\delta}}{m}}}_{\epsilon}$$



- Bound can be very loose, why should you care?
  - There are tighter, albeit more complicated, bounds
  - Bounds gives us formal guarantees that empirical studies can't provide
  - Bounds give us intuition about complexity of problems and convergence rate of algorithms

What you need to know

- Finite hypothesis space
  - Derive results
  - Counting number of hypothesis
  - Mistakes on Training data
- Complexity of the classifier depends on number of points that can be classified exactly
  - Finite case – decision trees
  - Infinite case – VC dimension
- Bias-Variance tradeoff in learning theory
- Margin-based bound for SVM
- Remember: will your algorithm find best classifier?