COMP 6321
Machine Learning

Instructor: Adam Krzyżak
Email: krzyzak@cs.concordia.ca

Lecture 6: Non-linear support vector machines. Kernels. Gaussian Processes. Learning Theory.

- SVMs for non-linearly separable data
- The kernel trick
- Mercer's theorem
- Kernelizing other machine learning methods
  - Kernelized linear regression
  - Kernelized logistic regression
- If we have time: Gaussian Processes
- True error of a hypothesis
- Probably Approximately Correct (PAC) model
- VC-dimension
- Other computational learning theory models

Recall: Linear support vector machines

- Classification method for linearly separable data
- Designed to maximize the *margin* of the data: the minimum distance between any instance and the decision boundary
- Last time: phrase this as a quadratic program, and *solve the dual*
- Solution can be represented as a linear combination of a *set of instances (support vectors)*
- Both the set of support vectors and their coefficients are obtained automatically as the solution to the quadratic program.
- If the data is not linearly separable, or if we want to avoid overfitting: *soft margins*

Recall: Soft margin

- Given $\mathbf{w}, w_0$, an example $(\mathbf{x}_i, y_i)$ is at least distance $M = 1/\|\mathbf{w}\|$ on the right side of the margin if:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1$$

- The soft margin approach relaxes these constraints:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 - \zeta_i \text{ where } \zeta_i \geq 0$$

- How can we interpret ζ_i?
  - If $\zeta_i = 0$, then the original distance constraint is satisfied.
  - If $\zeta_i \in (0, 1)$, then the point is on the correct side of the decision boundary, but not as far as it should be.
  - If $\zeta_i = 1$, then the point is on the decision boundary.
  - If $\zeta_i > 1$ then the point is on the wrong side of the decision boundary.

Recall: Soft margin SVMs

- Optimization problem:

$$\begin{array}{rl} \text{min} & \|\mathbf{w}\|^2 + C\sum_i \zeta_i \\ \text{w.r.t.} & \mathbf{w}, w_0, \zeta_i \\ \text{s.t.} & y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0) \geq (1-\zeta_i) \\ & \zeta_i \geq 0 \end{array}$$

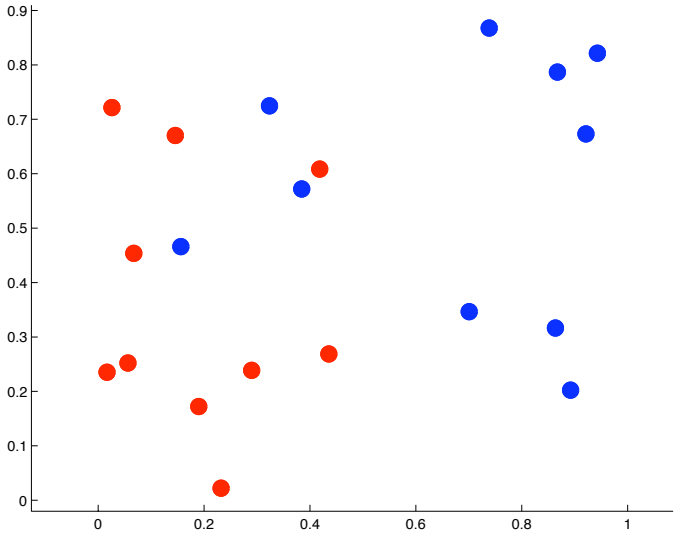
  where the first term is the margin, and the second term penalizes constraint violations

- $C > 0$ is a user-chosen cost associated with constraint violation, and help to control overfitting

- As in the separable case, the solution is of the form:

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i(\mathbf{x}_i\cdot\mathbf{x}) + w_0\right)$$

Non-linearly separable data



- A linear boundary might be too simple to capture the class structure.
- One way of getting a nonlinear decision boundary in the input space is to find a linear decision boundary in an expanded space (e.g., for polynomial regression.)
- Thus, $\mathbf{x}_i$ is replaced by $\phi(\mathbf{x}_i)$, where $\phi : \mathbb{R}^n \mapsto \mathbb{R}^N$ is called a *feature mapping*

Margin optimization in feature space

- Replacing $\mathbf{x}_i$ with $\phi(\mathbf{x}_i)$, the optimization problem to find $\mathbf{w}$ and w_0 becomes:

$$\begin{array}{rl} \text{min} & \|\mathbf{w}\|^2 + C\sum_i \zeta_i \\ \text{w.r.t.} & \mathbf{w}, w_0, \zeta_i \\ \text{s.t.} & y_i(\mathbf{w}\cdot\phi(\mathbf{x}_i) + w_0) \geq (1-\zeta_i) \\ & \zeta_i \geq 0 \end{array}$$

- Dual form:

$$\begin{array}{rl} \text{max} & \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j \phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j) \\ \text{w.r.t.} & \alpha_i \\ \text{s.t.} & 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^m \alpha_i y_i = 0 \end{array}$$

Feature space solution

- The optimal weights, in the expanded feature space, are $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x}_i)$.

- Classification of an input $\mathbf{x}$ is given by:

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) + w_0\right)$$

$\Rightarrow$ Note that to solve the SVM optimization problem in dual form and to make a prediction, we only ever need to compute *dot-products of feature vectors*.

Kernel functions

- Whenever a learning algorithm (such as SVMs) can be written in terms of dot-products, it can be generalized to kernels.
- A *kernel* is any function $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ which corresponds to a dot product for some feature mapping ϕ:

$$K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) \text{ for some } \phi.$$

- Conversely, by choosing feature mapping ϕ, we implicitly choose a kernel function
- Recall that $\phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) = \cos \angle(\mathbf{x}_1, \mathbf{x}_2)$ where $\angle$ denotes the angle between the vectors, so a kernel function can be thought of as a notion of *similarity*.

Example: Quadratic kernel

- Let $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$.
- Is this a kernel?

$$\begin{aligned} K(\mathbf{x}, \mathbf{z}) &= \left(\sum_{i=1}^{n} x_i z_i \right) \left(\sum_{j=1}^{n} x_j z_j \right) = \sum_{i,j \in \{1 \ldots n\}} x_i z_i x_j z_j \\ &= \sum_{i,j \in \{1 \ldots n\}} (x_i x_j)(z_i z_j) \end{aligned}$$

- Hence, it is a kernel, with feature mapping:

$$\phi(\mathbf{x}) = \langle x_1^2,\ x_1 x_2,\ \ldots,\ x_1 x_n,\ x_2 x_1,\ x_2^2,\ \ldots,\ x_n^2 \rangle$$

  Feature vector includes all squares of elements and all cross terms.
- Note that computing ϕ takes $O(n^2)$ but *computing K takes only $O(n)$*!

Polynomial kernels

- More generally, $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^d$ is a kernel, for any positive integer d:

$$K(\mathbf{x}, \mathbf{z}) = \left(\sum_{i=1}^{n} x_i z_i \right)^d$$

- If we expanded the sum above in the obvious way, we get n^d terms (i.e. feature expansion)
- Terms are monomials (products of x_i) with total power equal to d.
- If we use the primal form of the SVM, each of these will have a weight associated with it!
- *Curse of dimensionality:* it is very expensive both to optimize and to predict with an SVM in primal form
- However, *evaluating the dot-product of any two feature vectors can be done using K in $O(n)$*!

The "kernel trick"

- If we work with the dual, we do not actually have to ever compute the feature mapping ϕ. We just have to compute the similarity K.
- That is, we can solve the dual for the α_i:

$$\begin{array}{rl} \text{max} & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{w.r.t.} & \alpha_i \\ \text{s.t.} & 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^{m} \alpha_i y_i = 0 \end{array}$$

- The class of a new input $\mathbf{x}$ is computed as:

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sgn}\left((\sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x}_i)) \cdot \phi(\mathbf{x}) + w_0 \right) = \text{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + w_0 \right)$$

- Often, $K(\cdot, \cdot)$ can be evaluated in $O(n)$ time—a big savings!

Some other (fairly generic) kernel functions

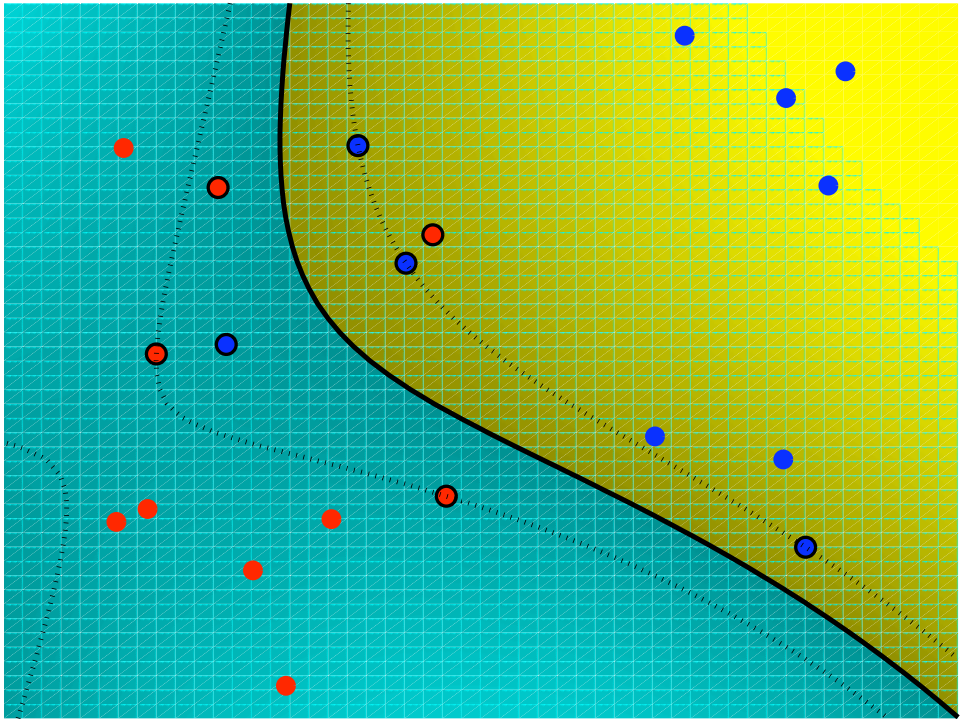
- $K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x} \cdot \mathbf{z})^d$ – feature expansion has all monomial terms of $\leq d$ total power.
- Radial basis/Gaussian kernel:

$$K(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2 / 2\sigma^2)$$

  The kernel has an infinite-dimensional feature expansion, but dot-products can still be computed in $O(n)$!
- Sigmoidal kernel:

$$K(\mathbf{x}, \mathbf{z}) = \tanh(c_1 \mathbf{x} \cdot \mathbf{z} + c_2)$$

Example: Gaussian kernel



Note the non-linear decision boundary

Application: Text classification (Joachims, 1998)

- Evaluated several methods, including SVMs, on a suite of text classification problems
- Words were stemmed (e.g. learn, learning, learned $\rightarrow$ learn)
- Nondiscriminative stopwords and words occurring < 3 times ignored
- For each remaining word, a binary feature indicates if the word is there or not
- 1000 features with greatest information gain retained, others discarded
- Each feature scaled by "inverse document frequency":

$$\log \frac{\text{\# docs}}{\text{\# docs with word } i}$$

Results

					SVM (poly) $d =$					SVM (rbf) $\gamma =$			
	Bayes	Rocchio	C4.5	k-NN	1	2	3	4	5	0.6	0.8	1.0	1.2
earn	95.9	96.1	96.1	97.3	98.2	98.4	**98.5**	98.4	98.3	**98.5**	98.5	98.4	98.3
acq	91.5	92.1	85.3	92.0	92.6	94.6	**95.2**	95.2	95.3	95.0	95.3	95.3	**95.4**
money-fx	62.9	67.6	69.4	78.2	66.9	72.5	75.4	74.9	**76.2**	74.0	75.4	**76.3**	75.9
grain	72.5	79.5	89.1	82.2	91.3	93.1	**92.4**	91.3	89.9	**93.1**	91.9	91.9	90.6
crude	81.0	81.5	75.5	85.7	86.0	87.3	88.6	**88.9**	87.8	**88.9**	89.0	88.9	88.2
trade	50.0	77.4	59.2	77.4	69.2	75.5	76.6	77.3	**77.1**	76.9	78.0	**77.8**	76.8
interest	58.0	72.5	49.1	74.0	69.8	63.3	67.9	73.1	**76.2**	74.4	75.0	**76.2**	76.1
ship	78.7	83.1	80.9	79.2	82.0	85.4	86.0	**86.5**	86.0	**85.4**	86.5	87.6	87.1
wheat	60.6	79.4	85.5	76.6	83.1	84.5	85.2	**85.9**	83.8	**85.2**	85.9	85.9	85.9
corn	47.3	62.2	87.7	77.9	86.0	86.5	85.3	**85.7**	83.9	**85.1**	85.7	85.7	84.5
microavg.	**72.0**	**79.9**	**79.4**	**82.3**	84.2	85.1	85.9	86.2	85.9	86.4	86.5	86.3	86.2
					combined: **86.0**					combined: **86.4**			

Figure 4: Precision/recall-breakeven point on the ten most frequent Reuters categories and microaveraged performance over all Reuters categories. k-NN, Rocchio, and C4.5 achieve highest performance at 1000 features (with $k = 30$ for k-NN and $\beta = 1.0$ for Rocchio). Naive Bayes performs best using all features.

SVMs are better than any other classifier

Getting SVMs to work in practice

- Two important choices:
  - Kernel (and kernel parameters)
  - Regularization parameter C
- The parameters may interact!

  E.g. for Gaussian kernel, the larger the width of the kernel, the more biased the classifier, so low C is better
- Together, these control overfitting: always do an internal parameter search, using a validation set!
- Overfitting symptoms:
  - Low margin
  - Large fraction of instances are support vectors

Interpretability

- More interpretable than neural nets if you look at the machine and the misclassifications
- E.g. Ovarian cancer data (Haussler) - 31 tissue samples of 3 classes, misclassified examples wronlgy labelled
- But no biological plausibility!
- Hard to interpret if the percentage of instances that are recruited as support vectors is high

Complexity

- Quadratic programming is expensive in the number of training examples
- Platt's SMO (sequential minimal optimization) algorithm is quite fast though, and other fancy optimization approaches are available
- Best packages can handle $20,000+$ instances, but not more than $100,000$
- On the other hand, number of attributes can be very high (strength compared to neural nets)
- Evaluating a SVM is *slow if there are a lot of support vectors*.
- Dictionary methods attempt to select a subset of the data on which to train.

Applications of SVMs

- The biggest strength of SVMs is dealing with large numbers of features (which relies on the kernel trick and the control of overfitting)
- Many successful applications in:
  - Text classification (e.g. Joachims, 1998)
  - Object detection (e.g. Osuna, Freund and Girosi, 1997)
  - Object recognition (e.g. Pontil and Verri, 1998)
  - Bioinformatics (e.g. Lee et al, 2002)
- SVMs are considered by many the state-of-the art approach to classification
- Experimentally, SVMs and neural nets are roughly tied based on evidence to date, each has its own preferred applications

Kernels beyond SVMs

- A lot of current research has to do with defining new kernels functions, suitable to particular tasks / kinds of input objects
- Many kernels are available:
  - Information diffusion kernels (Lafferty and Lebanon, 2002)
  - Diffusion kernels on graphs (Kondor and Jebara 2003)
  - String kernels for text classification (Lodhi et al, 2002)
  - String kernels for protein classification (e.g., Leslie et al, 2002)

  ... and others!

Example: String kernels

- Very important for DNA matching, text classification, ...
- Example: in DNA matching, we use a sliding window of length k over the two strings that we want to compare
- The window is of a given size, and inside we can do various things:
  - Count exact matches
  - Weigh mismatches based on how bad they are
  - Count certain markers, e.g. AGT
- The kernel is the sum of these similarities over the two sequences
- How do we prove this is a kernel?

Establishing "kernelhood"

- Suppose someone hands you a function K. How do you know that it is a kernel?
- More precisely, given a function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, under what conditions can $K(\mathbf{x}, \mathbf{z})$ be written as a dot product $\phi(\mathbf{x}) \cdot \phi(\mathbf{z})$ for some feature mapping ϕ?
- We want a general recipe, which does not require explicitly defining ϕ every time

Kernel matrix

- Suppose we have an arbitrary set of input vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_m$
- The *kernel matrix (or Gram matrix)* K corresponding to kernel function K is an $m \times m$ matrix such that $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ (notation is overloaded on purpose).
- What properties does the kernel matrix K have?
- Claims:
  1. K is symmetric
  2. K is positive semidefinite
- Note that these claims are consistent with the intuition that K is a "similarity" measure (and will be true regardless of the data)

Proving the first claim

If K is a valid kernel, then the kernel matrix is symmetric

$$K_{ij} = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) = \phi(\mathbf{x}_j) \cdot \phi(\mathbf{x}_i) = K_{ji}$$

Proving the second claim

If K is a valid kernel, then the kernel matrix is positive semidefinite

Proof: Consider an arbitrary vector $\mathbf{z} \in \mathbb{R}^m$

$$\begin{aligned}
\mathbf{z}^T K \mathbf{z} &= \sum_{i=1}^{m}\sum_{j=1}^{m} z_i K_{ij} z_j = \sum_{i=1}^{m}\sum_{j=1}^{m} z_i \left(\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)\right) z_j \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m} z_i \left(\sum_{k=1}^{N} \phi_k(\mathbf{x}_i)\phi_k(\mathbf{x}_j)\right) z_j \\
&= \sum_{k=1}^{N}\sum_{i=1}^{m}\sum_{j=1}^{m} z_i \phi_k(\mathbf{x}_i)\phi_k(\mathbf{x}_j) z_j \\
&= \sum_{k=1}^{N}\left(\sum_{i=1}^{m} z_i \phi_k(\mathbf{x}_i)\right)^2 \geq 0
\end{aligned}$$

Mercer's theorem

- We have shown that if K is a kernel function, then for any data set, the corresponding kernel matrix K defined such that $K_{ij} = K(\mathbf{x_i}, \mathbf{x_j})$ is symmetric and positive semidefinite
- Mercer's theorem states that the reverse is also true:
  Given a function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, *$K$ is a kernel if and only if, for any data set, the corresponding kernel matrix is symmetric and positive semidefinite*
- The reverse direction of the proof is much harder (see e.g. Vapnik's book for details)
- This result gives us a way to check if a given function is a kernel, by checking these two properties of its kernel matrix.
- Kernels can also be obtained by combining other kernels (see homework), or by learning from data
- Kernel learning may suffer from overfitting (kernel matrix close to diagonal)

Kernelizing other machine learning algorithms

- Many other machine learning algorithms have a “dual formulation”, in which dot-products of features can be replaced with kernels.
- Two examples now:
  - Logistic regression
  - Linear regression
- Later: kernel PCA

Linear regression with feature vectors

- Find the weight vector $\mathbf{w}$ which minimizes the (regularized) error function:

$$J(\mathbf{w}) = \frac{1}{2}(\mathbf{\Phi w} - \mathbf{y})^T(\mathbf{\Phi w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- The solution takes the form:

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{i=1}^{m}(\mathbf{w}^T\phi(\mathbf{x}_i) - y_i)\phi(\mathbf{x}_i) = \sum_{i=1}^{m} a_i\phi(\mathbf{x}_i) = \mathbf{\Phi}^T\mathbf{a}$$

  where $\mathbf{a}$ is a vector of size m (number of instances) with $a_i = -\frac{1}{\lambda}(\mathbf{w}^T\phi(\mathbf{x}_i) - y_i)$

- Main idea: *use $\mathbf{a}$ instead of $\mathbf{w}$ as parameter vector*
- Note that this is similar to re-formulating a weight vector in terms of a linear combination of instances, but we are not using the primal-dual mechanism in a literal sense

Re-writing the error function

- Instead of $J(\mathbf{w})$ we have $J(\mathbf{a})$:

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{a} - \mathbf{a}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{y} + \frac{1}{2}\mathbf{y}^T\mathbf{y} + \frac{\lambda}{2}\mathbf{a}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{a}$$

- Note that $\mathbf{\Phi}\mathbf{\Phi}^T = \mathbf{K}$, the kernel matrix!
- Hence, we can re-write this as:

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^T\mathbf{K}\mathbf{y} + \frac{1}{2}\mathbf{y}^T\mathbf{y} + \frac{\lambda}{2}\mathbf{a}^T\mathbf{K}\mathbf{a}$$

- By setting the gradient to 0 we get:

$$\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_m)^{-1}\mathbf{y}$$

Making predictions with dual-view regression

- For a new input $\mathbf{x}$, the prediction is:

$$h(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \mathbf{a}^T\mathbf{\Phi}\phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \lambda\mathbf{I}_m)^{-1}\mathbf{y}$$

  where $\mathbf{k}(\mathbf{x})$ is an m-dimensional vector, with the ith element equal to $K(\mathbf{x}, \mathbf{x}_i)$
- That is, the ith element has the similarity of the input to the ith instance
- Again, the *feature mapping is not needed* either to learn or to make predictions!
- This approach is useful if the feature space is very large.

Logistic regression

- The output of a logistic regression predictor is:

$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{\mathbf{w}^T \phi(\mathbf{x}) + w_0}}$$

- Again, we can define the weights in terms of support vectors: $\mathbf{w} = \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i)$
- The prediction can now be computed as:

$$h(\mathbf{x}) = \frac{1}{1 + e^{\sum_{i=1}^{m} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + w_0}}$$

- α_i are the new parameters (one per instance) and can be derived using gradient descent

Kernels in Bayesian regression

- The kernel view can be applied to Bayesian regression too
- Recall that in the Bayesian view, we have a prior over the parameters, $\mathbf{w}$
- The data induces a posterior distribution
- At any point, we can sample a parameter vector (i.e., a function) from the distribution
- Advantage: we get information about the variability of the prediction, in addition to the mean value

Example: Linear regression with features and prior

- Suppose that the weight vector $\mathbf{w}$ has a normal prior of mean zero:

$$P(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$

  where α is the "inverse variance" of the distribution
- What is the probability distribution of the vector of predictions $\mathbf{h} = \mathbf{\Phi w}$?
- Because $\mathbf{h}$ is a linear combination of normally distributed variables, it is also normal, so it is enough to compute the mean and covariance:

$$E(\mathbf{h}) = E(\mathbf{\Phi w}) = \mathbf{\Phi} E(\mathbf{w}) = \mathbf{0}$$

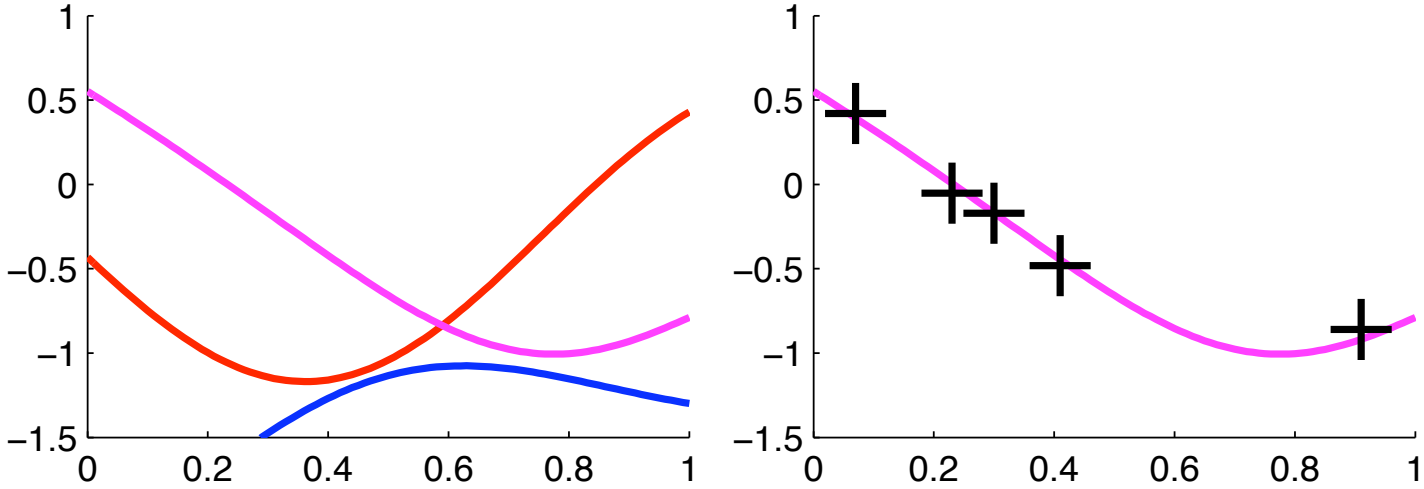
$$E(\mathbf{h}\mathbf{h}^T) = E(\mathbf{\Phi w w}^T \mathbf{\Phi}^T) = \mathbf{\Phi} E(\mathbf{w}\mathbf{w}^T)\mathbf{\Phi}^T = \frac{1}{\alpha}\mathbf{\Phi}\Phi^T = \mathbf{K}$$

  where $\mathbf{K}$ is a kernel matrix
- This is an example of a *Gaussian process*

Gaussian processes

- In general, a Gaussian process is a *probability distribution over functions* h such that the set of values $h(\mathbf{x}_i)$ evaluated at any arbitrary set of points $\mathbf{x}_i$ have a jointly Gaussian distribution
- The key property of the Gaussian process is that the mean and covariance are sufficient to specify the distribution
- Gaussian processes are increasingly used in regression as well as other parts of machine learning

Gaussian process for regression



$$f \sim \mathcal{GP}$$

$$\mathbf{f} \sim \mathcal{N}(0, K), \quad K_{ij} = k(x_i, x_j)$$

where $f_i = f(x_i)$

Noisy observations:

$$y_i | f_i \sim \mathcal{N}(f_i, \sigma_n^2)$$

Gaussian Process posterior

- The prior over observations and targets is Gaussian
- Hence, the posterior for any output point will also be Gaussian
- The posterior over functions is a Gaussian Process.
- Let β be the precision of the target noise
- To find the conditional distribution of the output $h(\mathbf{x})$ given the data, we partition the kernel matrix of the point and the data as:

$$\begin{pmatrix} \mathbf{K} & \mathbf{k}(\mathbf{x}) \\ \mathbf{k}(\mathbf{x})^T & K(\mathbf{x},\mathbf{x}) + \frac{1}{\beta} \end{pmatrix}$$

- The mean and variance of the predictions are, respectively

$$\mathbf{k}(\mathbf{x})^T\mathbf{K}^{-1}\mathbf{y} \text{ and } K(\mathbf{x},\mathbf{x}) + \frac{1}{\beta} - \mathbf{k}(\mathbf{x})^T\mathbf{K}^{-1}\mathbf{k}(\mathbf{x})$$

Binary classification: The golden goal

Given:

- The set of all possible instances X
- A target function (or concept) $f : X \rightarrow \{0, 1\}$
- A set of hypotheses H
- A set of training examples D (containing positive and negative examples of the target function)

$$\langle \mathbf{x}_1, f(\mathbf{x}_1) \rangle, \ldots \langle \mathbf{x}_m, f(\mathbf{x}_m) \rangle$$

Determine:

A hypothesis $h \in H$ such that $h(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in X$.

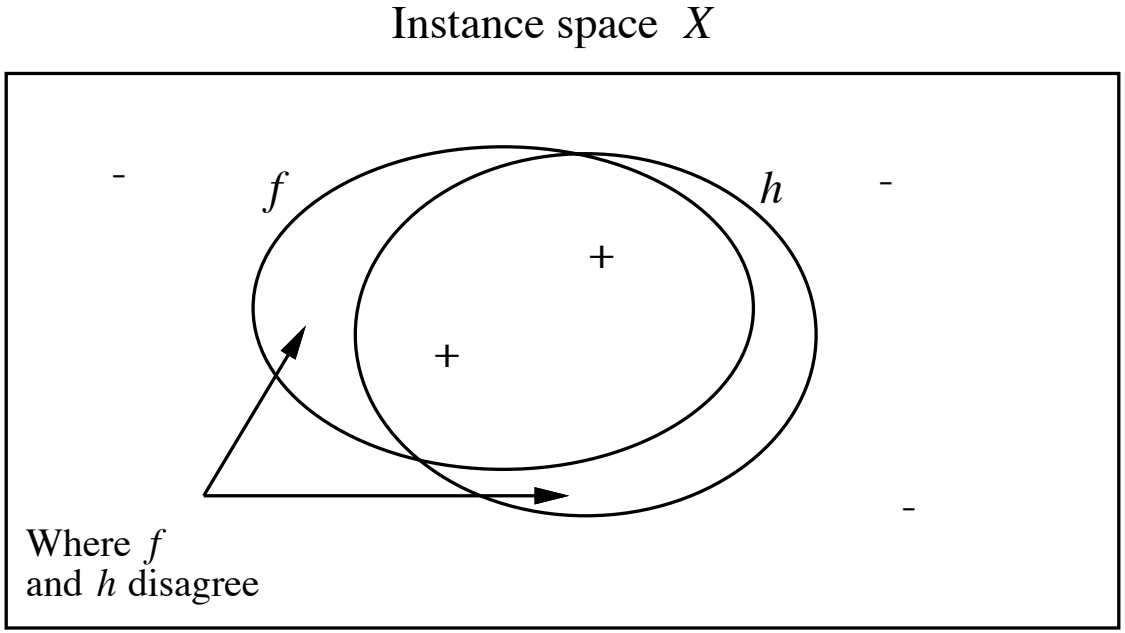
Approximate Concept Learning

- Requiring a learner to acquire the right concept is too strict
- Instead, we will allow the learner to produce a *good approximation* to the actual concept
- For any instance space, there is a non-uniform likelihood of seeing different instances
- We assume that there is a *fixed probability distribution* P on the space of instances X
- The learner is trained and tested on examples whose inputs are drawn *independently and randomly* according to P.

Recall: Two Notions of Error

- The *training error* of hypothesis h with respect to target concept f estimates how often $h(\mathbf{x}) \neq f(\mathbf{x})$ over the training instances
- The *true error* of hypothesis h with respect to target concept f estimates how often $h(\mathbf{x}) \neq f(\mathbf{x})$ over future, unseen instances (but drawn according to P)
- Questions:
  - Can we *bound the true error* of a hypothesis given only its training error?
  - How many examples are needed for a good approximation? This is called the *sample complexity* of the problem

True Error of a Hypothesis



True Error Definition

- The set of instances on which the target concept and the hypothesis disagree is denoted: $S = \{\mathbf{x}|h(\mathbf{x}) \neq f(\mathbf{x})\}$
- The *true error* of h with respect to f is:

$$\sum_{\mathbf{x} \in S} P(\mathbf{x})$$

  This is the probability of making an error on an instance randomly drawn from X according to P
- Let $\epsilon \in (0, 1)$ be an *error tolerance* parameter. We say that h is a *good approximation* of f (to within ϵ) if and only if the true error of h is less than ϵ.

Example: Rote Learner

- Let $X = \{0, 1\}^n$. Let P be the uniform distribution over X.
- Let the concept f be generated by randomly assigning a label to every instance in X.
- Let $D \subset X$ be a set of training instances.

  The hypothesis h is generated by memorizing D and giving a random answer otherwise.

- What is the training error of h?
- What is the true error of h?

Empirical risk minimization

- Suppose we are given a hypothesis class H
- We have a magical learning machine that can sift through H and output the hypothesis with the smallest training error, h_{emp}
- This is process is called *empirical risk minimization*
- Is this a good idea?
- What can we say about the error of the other hypotheses in h?

First tool: The union bound

- Let $E_1 \ldots E_k$ be k different events (not necessarily independent). Then:

$$P(E_1 \cup \cdots \cup E_k) \leq P(E_1) + \cdots + P(E_k)$$

- Note that this is usually loose, as events may be correlated

Second tool: Hoeffding bound

- Let $Z_1 \ldots Z_m$ be m independent identically distributed (iid) binary variables, drawn from a Bernoulli (binomial) distribution:

$$P(Z_i = 1) = \phi \text{ and } P(Z_i = 0) = 1 - \phi$$

- Let $\hat{\phi}$ be the mean of these variables:$\hat{\phi} = \frac{1}{m} \sum_{i=1}^{m} Z_i$
- Let ϵ be a fixed error tolerance parameter. Then:

$$P(|\phi - \hat{\phi}| > \epsilon) \leq 2e^{-2\epsilon^2 m}$$

- In other words, if you have lots of examples, the empirical mean is a good estimator of the true probability.
- Note: other similar concentration inequalities can be used (e.g. Chernoff, Bernstein, etc.)

Finite hypothesis space

- Suppose we are considering a finite hypothesis class $H = \{h_1, \ldots h_k\}$ (e.g. conjunctions, decision trees,...)
- Take an arbitrary hypothesis $h_i \in H$
- Suppose we sample data according to our distribution and let $Z_j = 1$ iff $h_i(\mathbf{x}_j) \neq y_j$
- So $e(h_i) = P(h_i(\mathbf{x}_j) \neq y_j)$ (the true error of h_i) is the expected value of Z_j
- Let $\hat{e}(h_i) = \frac{1}{m}\sum_{j=1}^{m} Z_j$ (this is the empirical training error of h_i on the data set we have)
- Using the Hoeffding bound (see next slide), we have:

$$P(|e(h_i) - \hat{e}(h_i)| > \epsilon) \leq 2e^{-2\epsilon^2 m}$$

- So, if we have *lots of data*, the *training error of a hypothesis h_i will be close to its true error* with high probability.

Hoeffding bound

- Let $X_1, \ldots, X_m$ be independent random variables (not necessarily identically distributed) and assume X_i are almost surely bounded, i.e.,

$$P(X_i \in [a_i, b_i]) = 1, i = 1, \ldots, m.$$

Let $\bar{X} = \frac{1}{m} \sum_{i=1}^{m} X_i$.

Then

$$P\left(\bar{X} - E\bar{X} \geq \epsilon\right) \leq \exp\left(\frac{-2\epsilon^2 m}{\frac{1}{m}\sum_{i=1}^{m}(b_i - a_i)^2}\right)$$

$$P\left(\left|\bar{X} - E\bar{X}\right| \geq \epsilon\right) \leq 2\exp\left(\frac{-2\epsilon^2 m}{\frac{1}{m}\sum_{i=1}^{m}(b_i - a_i)^2}\right)$$

What about all hypotheses?

- We showed that the empirical error is "close" to the true error for one hypothesis.
- Let E_i denote the event $|e(h_i) - \hat{e}(h_i)| > \epsilon$
- Can we guarantee this is true for all hypothesis?

$$
\begin{aligned}
P(\exists h_i \in H, |e(h_i) - \hat{e}(h_i)| > \epsilon) &= P(E_1 \cup \ldots E_k) \\
&\leq \sum_{i=1}^{k} P(E_i) \text{ (union bound)} \\
&\leq \sum_{i=1}^{k} 2e^{-2\epsilon^2 m} \text{ (shown before)} \\
&= 2ke^{-2\epsilon^2 m}
\end{aligned}
$$

A uniform convergence bound

- We showed that:

$$P(\exists h_i \in H, |e(h_i) - \hat{e}(h_i)| > \epsilon) \leq 2ke^{-2\epsilon^2 m}$$

- So we have:

$$1 - P(\exists h_i \in H, |e(h_i) - \hat{e}(h_i)| > \epsilon) \geq 1 - 2ke^{-2\epsilon^2 m}$$

  or, in other words:

$$P(\forall h_i \in H, |e(h_i) - \hat{e}(h_i)| < \epsilon) \geq 1 - 2ke^{-2\epsilon^2 m}$$

- This is called a *uniform convergence* result because the bound holds for all hypotheses
- What is this good for?

Sample complexity

- Suppose we want to guarantee that with probability at least $1 - \delta$, the sample (training) error is within ϵ of the true error.
- From our bound, we can set $\delta \geq 2ke^{-2\epsilon^2 m}$
- Solving for m, we get that the number of samples should be:

$$m \geq \frac{1}{2\epsilon^2} \log \frac{2k}{\delta} = \frac{1}{2\epsilon^2} \log \frac{2|H|}{\delta}$$

- So the *number of samples needed is logarithmic* in the size of the hypothesis space

Example: Conjunctions of Boolean Literals

- Let H be the space of all pure conjunctive formulae over n Boolean attributes. Then $|H| = 3^n$ (why?)
- From the previous result, we get:

$$m \geq \frac{1}{2\epsilon^2} \log \frac{2|H|}{\delta} \geq n \frac{1}{2\epsilon^2} \log \frac{2}{\delta}$$

- This is linear in n!
- Hence, conjunctions are "easy to learn"

Example: Arbitrary Boolean functions

- Let H be the space of all Boolean formulae over n Boolean attributes. Then $|H| = 2^{3^n}$ (why?)
- From the previous result, we get:

$$m \geq \frac{1}{2\epsilon^2} \log \frac{2|H|}{\delta} \geq 3^n \frac{1}{2\epsilon^2} \log \frac{2}{\delta}$$

- This is exponential in n!
- Hence, arbitrary Boolean functions are "hard to learn"
- A similar argument can be applied to show that even restricted classes of Boolean functions, like parity and XOR, are hard to learn

Another application: Bounding the true error

- Our inequality revisited:

$$P(\forall h_i \in H, |e(h_i) - \hat{e}(h_i)| < \epsilon) \geq 1 - 2ke^{-2\epsilon^2 m} = 1 - \delta$$

- Suppose we hold m and δ fixed, and we solve for ϵ. Then we get:

$$|e(h_i) - \hat{e}(h_i)| \leq \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}$$

  inside the probability term.
- Can we now prove anything about the generalization power of the empirical risk minimization algorithm?

Empirical risk minimization

Let h^* be the best hypothesis in our class (in terms of true error). Based on our uniform convergence assumption, we can bound the true error of h_{emp} as follows:

$$
\begin{aligned}
e(h_{emp}) &\leq \hat{e}(h_{emp}) + \epsilon \\
&\leq \hat{e}(h^*) + \epsilon \text{ (because } h_{emp} \text{ has better training error} \\
&\quad \text{than any other hypothesis)} \\
&\leq e(h^*) + 2\epsilon \text{ (by using the result on } h^*) \\
&\leq e(h^*) + 2\sqrt{\frac{1}{2m}\log\frac{2|H|}{\delta}} \text{ (from previous slide)}
\end{aligned}
$$

This bounds how much worse h_{emp} is, wrt the best hypothesis we can hope for!

Bias and variance revisited

- We showed that, given m examples, with probability at least $1 - \delta$,

$$e(h_{emp}) \leq \left(\min_{h \in H} e(h) \right) + 2\sqrt{\frac{1}{2m} \log \frac{2|H|}{\delta}}$$

- Suppose now that we are considering two hypothesis classes $H \subseteq H'$
  - The first term would be smaller for H' (we have a larger hypothesis class, hence less "bias")
  - The second term would be larger (the "variance" is increasing)
- Note, though, that if H is infinite, this result is not very useful...

Example: Learning an interval on the real line

- "Treatment plant is ok iff Temperature $\leq a$" for some unknown $a \in [0, 100]$
- Consider the hypothesis set:

$$H = \{[0, a] | a \in [0, 100]\}$$

- Simple learning algorithm: Observe m samples, and return $[0, b]$, where b is the largest positive example seen
- Clearly the processing time per example is polynomial. But how many examples do we need to find a good approximation of the true hypothesis?
- Our previous result is useless, since the hypothesis class is infinite.

Sample complexity of learning an interval

- Let a correspond to the true concept and let $c < a$ be a real value s.t. $[c, a]$ has probability ϵ.
- If we see an example in $[c, a]$, then our algorithm succeeds in having true error smaller than ϵ
- What is the probability of seeing m iid examples *outside* of $[c, a]$?

$$P(\text{failure}) = (1 - \epsilon)^m$$

- If we want

$$P(\text{failure}) < \delta \Longrightarrow (1 - \epsilon)^m < \delta$$

Example continued

- Fact:
$$(1-\epsilon)^m \leq e^{-\epsilon m} \text{ (you can check that this is true)}$$
- Hence, it is sufficient to have
$$(1-\epsilon)^m \leq e^{-\epsilon m} < \delta$$
- Using this fact, we get:
$$m \geq \frac{1}{\epsilon} \log \frac{1}{\delta}$$
- You can check empirically that this is a fairly tight bound.

Why do we need so few samples?

- Our hypothesis space is simple - there is only one parameter to estimate!
- In other words, there is one “degree of freedom”
- As a result, every data sample gives information about LOTS of hypothesis!
- What if there are more “degrees of freedom”?

Example: Learning two-sided intervals

- Suppose the target concept and hypothesis class are positive inside $[a, b]$.
- Our guess interval is $[\min_{(x,+)} x, \max_{(x,+)} x]$
- We can make errors on either side of the interval, if we get no example within ϵ of the true value.
- The probability of an example outside of an ϵ-size interval is $1 - \epsilon$
- The probability of m examples outside of it is $(1 - \epsilon)^m$
- The probability this happens on either side is $\leq 2(1 - \epsilon)^m \leq 2e^{-\epsilon m}$, and we want this to be $< \delta$

Example continued

- If we extract the number of samples we get:

$$m \geq \frac{1}{\epsilon} \ln \frac{2}{\delta}$$

- Compare this with the bound in the finite case:

$$m \geq \frac{1}{2\epsilon^2} \log \frac{2|H|}{\delta}$$

- But for us, $|H| = \infty$!
- We need a way to characterize the "complexity" of infinite-dimensional classes of hypotheses

Shattering a set of instances

- A *dichotomy* of a set S is a partition of S into two disjoint subsets.
- A set of instances D is *shattered* by hypothesis space H if and only if for every dichotomy of D there exists some hypothesis in H consistent with this dichotomy.

Shatter coefficient

- Formal definition.
- Let $\mathcal{A}$ be a class of subsets of $\mathcal{R}^d$ and let $n \in \mathcal{N}$.
  For $z_1, \ldots, z_n \in \mathcal{R}^d$ define

$$s\left(\mathcal{A}, \{z_1, \ldots, z_n\}\right) = \left|\{A \cap \{z_1, \ldots, z_n\} \ : \ A \in \mathcal{A}\}\right|,$$

  that is, $s(\mathcal{A}, \{z_1, \ldots, z_n\})$ is the number of different subsets of $\{z_1, \ldots, z_n\}$ of the form $A \cap \{z_1, \ldots, z_n\}$, $A \in \mathcal{A}$.

Shatter coefficient continued

- Let G be a subset of $\mathcal{R}^d$ of size n. One says that $\mathcal{A}$ **shatters** G if $s(\mathcal{A}, G) = 2^n$, i.e., if each subset of G can be represented in the form $A \cap G$ for some $A \in \mathcal{A}$.

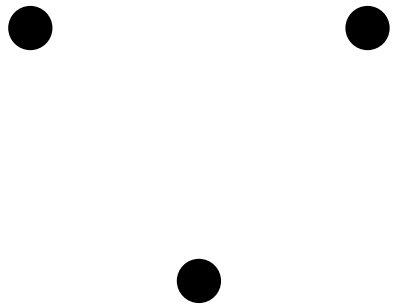
- The nth **shatter coefficient** of $\mathcal{A}$ is

$$S(\mathcal{A}, n) = \max_{\{z_1,\ldots,z_n\} \subseteq \mathcal{R}^d} s\left(\mathcal{A}, \{z_1, \ldots, z_n\}\right).$$

  That is, the shatter coefficient is the maximal number of different subsets of n points that can be picked out by sets from $\mathcal{A}$.

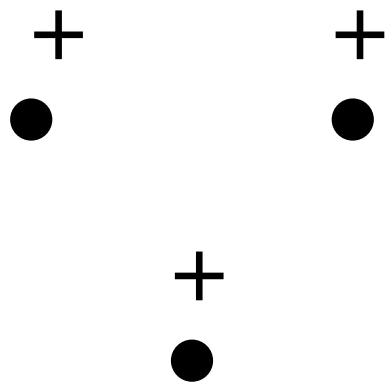
Example: Three instances

Can these three points be shattered by the hypothesis space consisting of a set of circular disks?



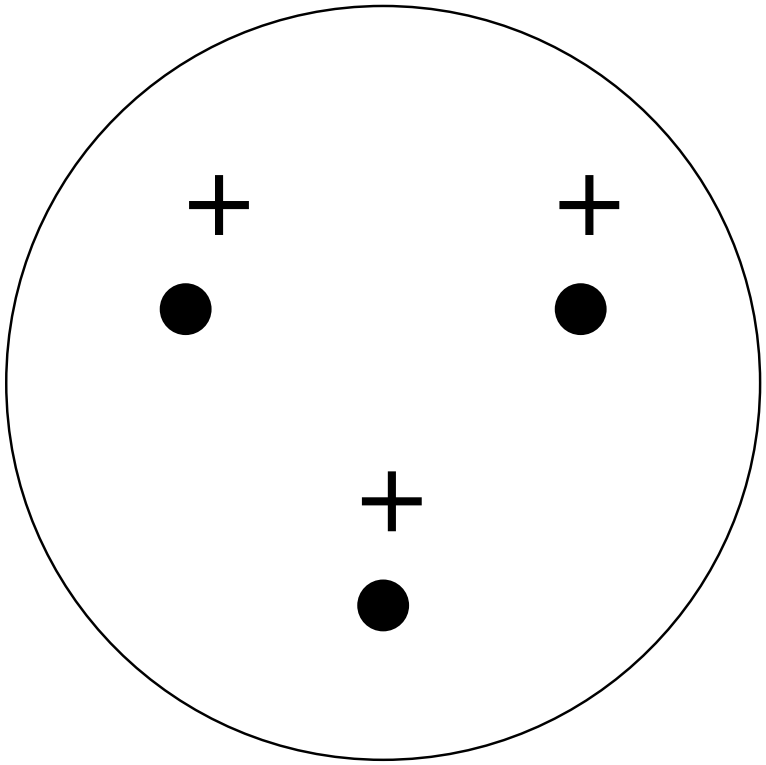
Example: Three instances

Can these three points be shattered by the hypothesis space consisting of a set of circular disks?



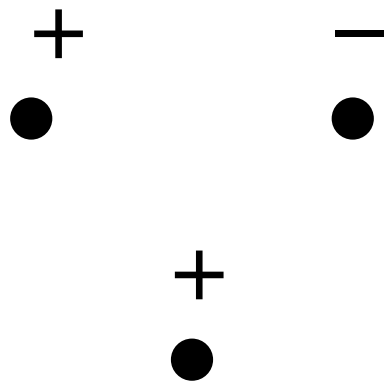
Example: Three instances

Can these three points be shattered by the hypothesis space consisting of a set of circular disks?



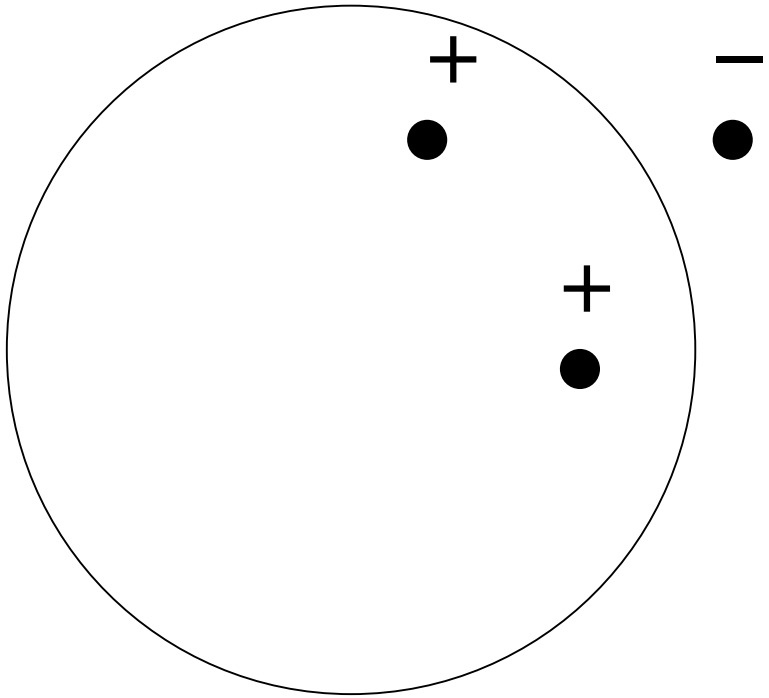
Example: Three instances

Can these three points be shattered by the hypothesis space consisting of a set of circular disks?



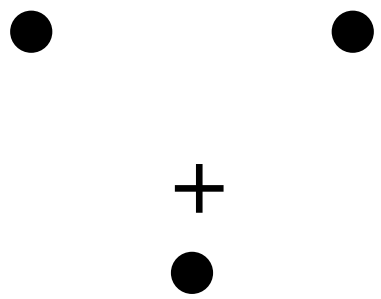
Example: Three instances

Can these three points be shattered by the hypothesis space consisting of a set of circular disks?



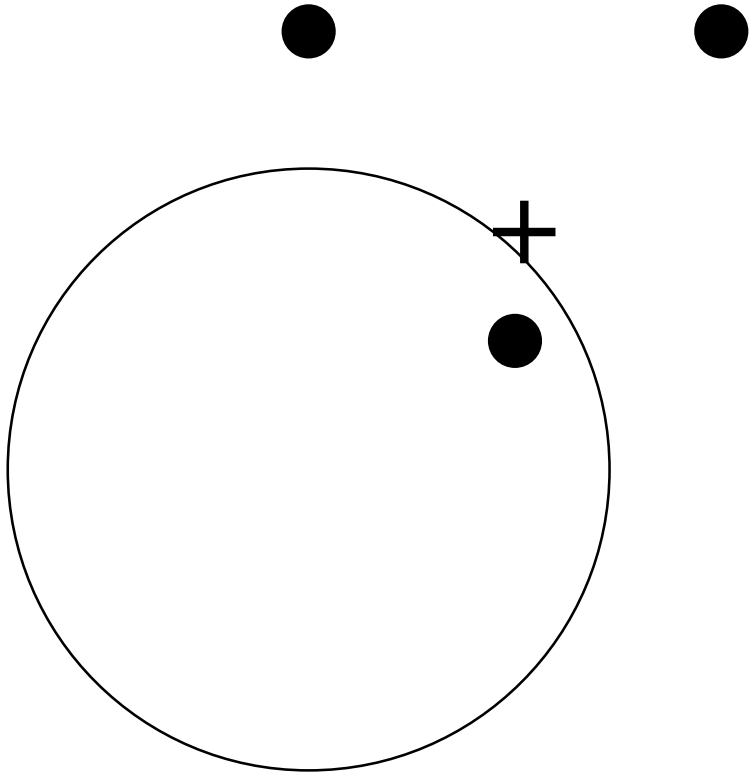
Example: Three instances

Can three points be shattered by the hypothesis space consisting of a set of circular disks?



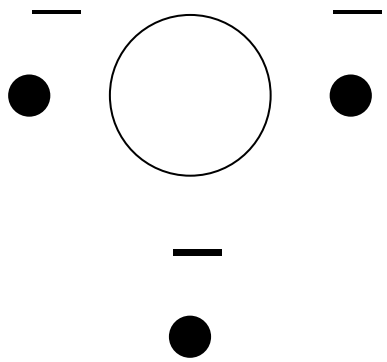
Example: Three instances

Can three points be shattered by the hypothesis space consisting of a set of circular disks?



Example: Three instances

Can three points be shattered by the hypothesis space consisting of a set of circular disks?



What about 4 points?

Example: Four instances

- These cannot be shattered, because we can label the farther 2 points as $+$, and the circular disk that contains them will necessarily contain the other points
- So circular disks can shatter one data set of three points (the one we've been analyzing), but there is no set of four points that can be shattered by circular disks (check this by yourself!)
- Note that not all sets of size 3 can be shattered!
- We say that the *VC dimension of circular disks is 3*

The Vapnik-Chervonenkis (VC) Dimension

- 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 $VC(H) \equiv \infty$.
- Formal definition.

  Let $\mathcal{A}$ be a class of subsets of $\mathcal{R}^d$ with $\mathcal{A} \neq \emptyset$. The **VC dimension** (or Vapnik–Chervonenkis dimension) $V_{\mathcal{A}}$ of $\mathcal{A}$ is defined by

$$V_{\mathcal{A}} = \sup \{n \in \mathcal{N} \ : \ S(\mathcal{A}, n) = 2^n\},$$

  i.e., the VC dimension $V_{\mathcal{A}}$ is the largest integer n such that there exists a set of n points in $\mathcal{R}^d$ which can be shattered by $\mathcal{A}$.

The Vapnik-Chervonenkis (VC) Dimension

- In other words, the VC dimension is the maximum number of points for which H is unbiased.
- VC dimension measures how many distinctions the hypotheses from H are able to make
- This is, in some sense, the number of "effective degrees of freedom"
- Introduced by Vapnik and Chervonenkis in a seminal paper:

  Vladimir N. Vapnik and A. Ya. Chervonenkis "On the uniform convergence of relative frequencies of events to their probabilities." Theory of Probability & Its Applications, vol. 16, no. 2, 1971, pp. 264-280.

Establishing the VC dimension

- Play the following game with the enemy:
  - You are allowed to *choose k points*. This actually gives you a lot of freedom!
  - The enemy then labels these points any way it wants
  - You now have to produce a hypothesis, out of your hypothesis class, which correctly produces these labels.

  If you are able to succeed at this game, the *VC dimension is at least k*.
- To show that it is *no greater than k*, you have to show that for any set of $k+1$ points, the enemy can find a labeling that you cannot correctly reproduce with any of your hypotheses.

Example revisited: VC dimension of intervals

- Can we shatter 2 points on a line with an interval?
- Can we shatter 3 points on a line with one interval?
- What is the VC dimension of intervals?

Example revisited: VC dimension of intervals

- Can we shatter 2 points on a line with an interval?

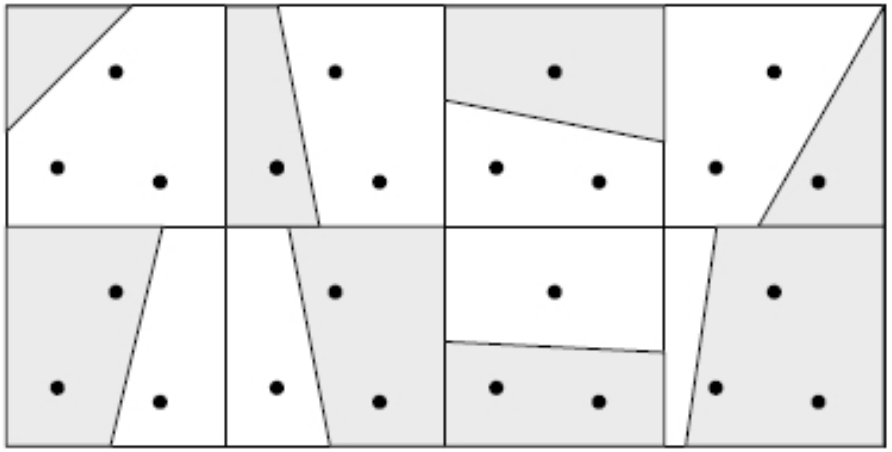
  Yes!
- Can we shatter 3 points on a line with one interval?

  No! The enemy can label the most distant points $+$ and the middle one $-$
- What is the VC dimension of intervals?

  VC dimension is 2

VC dimension of linear decision surfaces

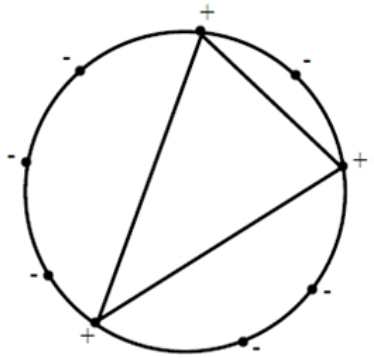
- Consider a linear threshold unit in the plane.
- First, show there exists a set of 3 points that can be shattered by half-spaces $\Longrightarrow$ VC dimension of half-spaces in the plane is at least 3.



- To show it is at most 3, show that NO set of 4 points can be shattered.
- For an n-dimensional space, VC dimension of linear estimators is $n + 1$.

VC dimension of convex polygons

- Consider a set of convex polygons on the plane (boundary and inside) with arbitrary number of vertices. Points inside the convex polygon are positive and outside are negative.
- Pick d points on a circle and label them arbitrarily. For every labeling of d points on the circle perimeter, there exists a convex polygon that is consistent with that labeling. This polygon includes all the positive examples and none of the negative.



- Hence d points in general position can be shuttered for any positive d and thus VC dimension of convex polygons in the plane is infinite.

Applying VC theory to feed-forward networks

- Let H_G be the class of functions that can be computed by feed-forward networks of perceptrons (also known as multi-layer perceptrons) defined on a fixed underlying graph G with E edges and $N \geq 2$ linear threshold nodes.
- Then it can be shown that $VC(H_G) \leq 2(E+N)\log(eN)$.

And the bad news...

- Sigmoid-like functions can have infinite VC dimension! E.g.

$$\frac{1}{1+e^{-x}} + cx^3 e^{-x^2} \sin x$$

  (see Macintyre and Sontag, 1993).
- However: the usual sigmoid function, as well as the hyperbolic tangent, have finite VC dimension! :-)
- But: it is doubly exponential... :-(
- However, in practice, neural networks seem to approximate well even with a lot fewer examples (sometimes fewer than the number of weights).
- Alternative analysis (see, e.g. Bartlett, 1996) suggest that the error may be related to the *magnitude of the weights*, rather than the number of weights, if the nodes are kept in their linear regions.

Error bounds using VC dimension

- Recall our error bound in the finite case:

$$e(h_{emp}) \leq \left(\min_{h \in H} e(h)\right) + 2\sqrt{\frac{1}{2m} \log \frac{2|H|}{\delta}}$$

- Vapnik showed a similar result, but using VC dimension instead of the size of the hypothesis space:
- For a hypothesis class H with VC dimension $VC(H)$, given m examples, with probability at least $1 - \delta$, we have:

$$e(h_{emp}) \leq \left(\min_{h \in H} e(h)\right) + O\left(\sqrt{\frac{VC(H)}{m} \log \frac{m}{VC(H)} + \frac{1}{m} \log \frac{1}{\delta}}\right)$$

Remarks on VC dimension

- The previous bound is tight up to log factors. In other words, for hypotheses classes with large VC dimension, we can show that there exists some data distribution which will produce a bad approximation.
- For many reasonable hypothesis classes (e.g. linear approximators) the VC dimension is linear in the number of "parameters" of the hypothesis.
- This shows that to learn "well", we need a number of examples that is linear in the VC dimension (so linear in the number of parameters, in this case).
- An important property: if $H_1 \subseteq H_2$ then $VC(H_1) \leq VC(H_2)$.

Structural risk minimization

$$e(h_{emp}) \leq \left(\min_{h \in H} e(h)\right) + O\left(\sqrt{\frac{VC(H)}{m} \log \frac{m}{VC(H)} + \frac{1}{m} \log \frac{1}{\delta}}\right)$$

- We have used this bound to measure the true error of the hypothesis with the smallest training error
- Why not use the bound directly to get the best hypothesis?
- We can measure the training error, and add to that the quantity suggested by the rightmost term
- We pick the hypothesis that is best in terms of this sum!
- This approach is called structural risk minimization, and can be used instead of crossvalidation or MDL to pick the best hypothesis class

Probably Approximately Correct (PAC) Learning

Let F be a concept (target function) class defined over a set of instances X in which each instance has length n. An algorithm L, using hypothesis class H is a *PAC learning algorithm* for F if:

- for any concept $f \in F$
- for any probability distribution P over X
- for any parameters $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$

the learner L will, with probability at least $(1 - \delta)$, output a hypothesis with true error at most ϵ.

A class of concepts F is *PAC-learnable* if there exists a PAC learning algorithm for F.

Computational vs Sample Complexity

- A class of concepts is *polynomial-sample PAC-learnable* if it is PAC learnable using a number of examples at most polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$ and n.
- A class of concepts is *polynomial-time PAC-learnable* if it is PAC learnable in time at most polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$ and n.
- Sample complexity is often easier to bound than time complexity!
- Sometimes there is a trade-off between the two (if there are more samples, less work is required to process each one and vice versa)

Bird Eye View of Computational Learning Theory

1. How hard is it to learn (in terms of the computation required)?

   Difficult to answer in general, but results have been established for some problems
2. How many examples are required for a good approximation?

   A lot of results here, regarding sample complexity bounds for different algorithms
3. What problems can be solved by a given algorithm?

   Little work done here so far.

Different Models of Learning

- Examples come randomly from some fixed distribution (the case usually considered in supervised learning)
- The learner is allowed to ask questions to the teacher (active learning) - we will look at this next lecture
- Examples are given by an opponent (on-line learning, mistake-bound model)

Most of the time the results assume that the examples are noise-free. However, results do exist for particular kinds of noise (e.g. noise in the target value).

Summary

- The complexity results for binary classification show trade-offs between the desired degree of precision ϵ, the number of samples m and the complexity of the hypothesis space H
- The complexity of H can be measured by the VC dimension
- For a fixed hypothesis space, minimizing the training set error is well justified (empirical risk minimization)
- We have not talked about
  - Relationship between margin and VC dimension (better bounds than the results discussed)
  - Lower bounds
  - ...