

Lecture notes #3: Support Vector Machines

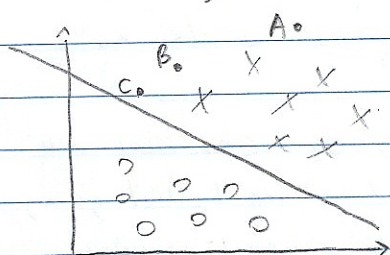
Part V: Support Vector Machines

- We now study the Support Vector Machines (SVM) learning algorithm
- SVMs are among the best "off-the-shelf" supervised learning algorithms

§1: Margins: Intuition

- Consider logistic regression where the probability $p(y=1|x;\theta)$ is modeled by $h_\theta(x) = g(\theta^T x)$
- We would then predict "1" iff $h_\theta(x) \geq 0.5$ iff $\theta^T x \geq 0$
- Consider a positive training example ($y=1$)
- The larger $\theta^T x$ is, the larger $h_\theta(x) = p(y=1|x;\theta)$ also is, and this also the higher our degree of "confidence" that the label is 1
- Thus, informally, we can think of our prediction as being a very confident one that $y=1$ if $\theta^T x \gg 0$ (conversely, that $y=0$ if $\theta^T x \ll 0$)
- Again informally, given a training set it seems that we have found a good fit to the training data if we can find θ so that $\theta^T x^{(i)} \gg 0$ whenever $y^{(i)} = 1$ and $\theta^T x^{(i)} \ll 0$ if $y^{(i)} = 0$
- This would reflect a very confident (and correct) set of classifications for all the training examples
- We will soon formalize this notion

- For a different type of intuition, consider the following figure, showing some training set



- A, B, C are three points
- x = positive training example
- o = negative training example
- The decision boundary, i.e. the line given by the equation $\theta^T x = 0$, and is also called the separating hyperplane

- Notice that A is very far from the decision boundary. Thus, if we were

asked to make a prediction for the value of y at A , it seems we would be very confident that $y=1$ there

- Conversely for C b/c it's close to decision boundary, just a small change to it would have caused us to predict $y=0$

- Thus, informally, it would be nice, if given a training set, we manage to find a decision boundary that is far from all training examples, so that we can make confident predictions about them

- We will also formalize this

- §2: Notation

- We now introduce new notation for talking about classification

- We will be considering a linear classifier for a binary classification problem with labels y and features x

- From now, we let $y \in \{-1, 1\}$ to denote class levels

- Also instead of parametrizing our linear classifier with the vector θ , we will use parameters w, b and write our classifier as

$$h_{w,b}(x) = g(w^T x + b)$$

where

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

- This " w, b " notation allows us to explicitly treat the intercept term b separately from the other parameters

- We also drop the convention of having the extra coordinate $x_0=1$

- Hence b replaces θ_0 and $w = [\theta_1 \dots \theta_n]^T$

- Note also that according to our definition of g , our classifier will directly output either -1 or 1 (cf. the perceptron algo) without first going through the intermediate step of estimate the prob. of $y=1$ (which is what LR did)

§3: Functional and Geometric Margins

- Given a training example $(x^{(1)}, y^{(1)})$, we define the functional margin of (w, b) w.r.t. the training example

$$\hat{\gamma}^{(1)} = y^{(1)}(w^T x + b)$$

- Note that if

• $y^{(1)} = 1$, then for the functional margin to be large (i.e. for our prediction to be correct and confident), we need that $w^T x + b$ to be a large positive number.

• $y^{(1)} = -1$, then for the functional margin to be large, we need that $w^T x + b$ be a large negative number.

- Furthermore, if $y^{(1)}(w^T x + b) > 0$, then our prediction on this example is correct.

- Hence, a large functional margin represents a confident and correct prediction!

- For a linear classifier with the choice of g given above, there's one undesirable property that make the functional margin a not very good measure of confidence.

- Given our choice of g , we note that if we replace (w, b) with $(2w, 2b)$, then $g(w^T x + b) = g(2w^T x + 2b)$.

• Thus $h_{w,b}(x)$ doesn't change at all in this case.

• This is b/c g , and hence $h_{w,b}(x)$, depends only on the sign and not on the magnitude of $w^T x + b$.

- However, replacing (w, b) with $(2w, 2b)$ actually doubles the functional margin.

- Thus, it seems that by exploiting our freedom to scale w and b , we can make the functional margin arbitrarily large without really changing anything meaningful.

- Hence, intuitively, it might make sense to impose some sort of normalization condition such as $\|w\|_2 = 1$.

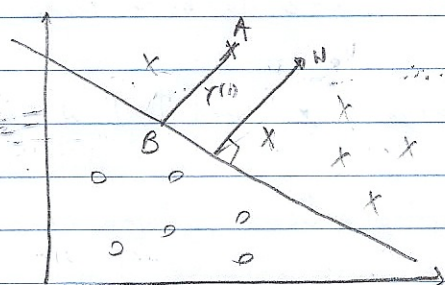
• Hence, we would replace (w, b) with $(w/\|w\|_2, b/\|w\|_2)$ and instead consider the function margin of $(w/\|w\|_2, b/\|w\|_2)$.

- We further discuss this later.

- Given a training set $S = \{(x^{(i)}, y^{(i)}) : i = 1, \dots, m\}$, we also define the functional margin of (w, b) w.r.t. S to be

$$\hat{\gamma} = \min_{i=1, \dots, m} \hat{\gamma}^{(i)}$$

- We now discuss geometric margins.



- The decision boundary corresponding to (w, b) , i.e. $w^T x + b = 0$, is shown

- Note that w is orthogonal to the separating hyperplane.

- Let x_1, x_2 be any points in the decision boundary.

- Then $w^T x_1 + b = 0$ and $w^T x_2 + b = 0$.

- Thus

$$(w^T x_1 + b) - (w^T x_2 + b) = \langle w, x_1 - x_2 \rangle = 0$$

where $x_1 - x_2$ is a vector that lies along the decision boundary

- Consider the point at A , which represents some training example $(x^{(i)}, y^{(i)})$ with $y^{(i)} = 1$

- Its distance to the decision boundary, $\gamma^{(i)}$, is given by the segment AB

- How can we find $\gamma^{(i)}$?

- $w/\|w\|$ is a unit-length vector pointing in the same direction as w

- Since A represents $x^{(i)}$, we therefore find that the point B is given by $x^{(i)} - \gamma^{(i)} w/\|w\|$

- But B lies in the decision boundary and so satisfies $w^T x + b = 0$. So,

$$w^T (x^{(i)} - \gamma^{(i)} w/\|w\|) + b = 0 \Rightarrow w^T x^{(i)} - \frac{\gamma^{(i)} \|w\|^2}{\|w\|} + b = 0$$

$$\Rightarrow \gamma^{(i)} = \frac{w^T x^{(i)} + b}{\|w\|} = \left(\frac{w}{\|w\|} \right)^T x^{(i)} + \frac{b}{\|w\|}$$

- We define the geometric margin of (w, b) w.r.t. a training example $(x^{(i)}, y^{(i)})$ to be

$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{w}{\|w\|} \right)^T x^{(i)} + \frac{b}{\|w\|} \right)$$

- Note that if $\|w\| = 1$, the geometric margin equals the fractional margin.
- Also the geometric margin is invariant to rescaling of the parameters (w, b) .

⇒ Hence, when trying to fit w and b to the training data, we can impose an arbitrary scaling constraint on w without changing anything important.

- Finally given a training set $S = \{(x^{(i)}, y^{(i)}) : i = 1, 2, \dots, m\}$, we define the geometric margin of (w, b) w.r.t. S as

$$\gamma = \min_{i=1, \dots, m} \gamma^{(i)}$$

§4: The optimal margin classifier

- Given a training set, we now try to find a decision boundary that maximizes the (geometric) margin, since this would reflect a very confident set of predictions on the training set and a good "fit" to the training data.
- This will result in a classifier that separates the positive and negative training examples with a gap.

- For now, we will assume that our training set is linearly separable, i.e. that it is possible to separate the + and - training examples using some separating hyperplane.

- How can we find the one that maximizes the geometric margin?
- We pose the following optimization problem

$$\begin{aligned} \max_{w, b} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)} (w^T x^{(i)} + b) \geq \gamma, \quad i = 1, 2, \dots, m \\ & \|w\| = 1 \end{aligned}$$

- The $\|w\|=1$ ensures that the functional and geometric margins are equal

- If we can solve the above, we are done.

- But the $\|w\|=1$ constraint is a nasty (non-convex) one

- Let us try to transform the problem into a nicer one. Consider

$$\begin{aligned} \max_{w, b} \quad & \frac{\hat{\gamma}}{\|w\|} \\ \text{s.t.} \quad & y^{(i)} (w^T x^{(i)} + b) \geq \hat{\gamma}, \quad i=1, \dots, m \end{aligned}$$

- Here, we are maximizing $\hat{\gamma}/\|w\|$, subject to the functional margin being at least $\hat{\gamma}$.

- Since the geometric and functional margins are related by $\gamma = \hat{\gamma}/\|w\|$, this will give us the answer we want.

- This gets us rid of the constraint $\|w\|=1$.

- But we now have a nasty (non-convex) objective function $\hat{\gamma}/\|w\|$.

- Recall that we can add an arbitrary scaling constraint on w and b without changing anything.

- We introduce the scaling constraint that the functional margin of w, b w.r.t. the training set must be 1:

$$\hat{\gamma} = 1$$

- Since multiplying w and b by some constant results in the functional margin being multiplied by the same constant, this is indeed a scaling constraint, and can be satisfied by rescaling w, b .

- Plugging this back, we now need to maximize $\hat{\gamma}/\|w\| = 1/\|w\|$ which is equivalent to minimizing $\|w\|^2$.

- So we now have the following optimization problem:

$$\begin{aligned} \min_{w, b} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad & y^{(i)} (w^T x^{(i)} + b) \geq 1, \quad i=1, \dots, m \end{aligned}$$

- We have now transformed the problem into one that can be efficiently solved.

- Its solution gives us the optimal margin classifier.

- We now study about Lagrange duality, which will lead us to our optimization problem's dual form.

§5: Lagrange duality

- We now talk about solving constrained optimization problems
- Consider a problem of the following form:

$$\min_w F(w)$$

$$\text{s.t. } h_i(w) = 0, \quad i = 1, \dots, l$$

- We can use the method of Lagrange multipliers to solve it.
- We define the Lagrangian to be

$$\mathcal{L}(w, \beta) = F(w) + \sum_{i=1}^l \beta_i h_i(w)$$

- The β_i 's are called the Lagrange multipliers

- We then would then find and set $\mathcal{L}$'s partial derivatives to zero:

$$\frac{\partial \mathcal{L}}{\partial w_i} = 0 \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial \beta_i} = 0$$

and solve for w and β

- In this section, we generalize this to constrained optimization problems in which we may have inequality as well as equality constraints.

- We consider the primal optimization problem

$$\min_w F(w)$$

$$\text{s.t. } g_i(w) \leq 0, \quad i = 1, \dots, K$$

$$h_i(w) = 0, \quad i = 1, \dots, l$$

- We defined the generalized Lagrangian

$$\mathcal{L}(w, \alpha, \beta) = F(w) + \sum_{i=1}^K \alpha_i g_i(w) + \sum_{i=1}^l \beta_i h_i(w)$$

- The α_i 's and β_i 's are the Lagrangian multipliers

- Consider the quantity

$$p^* = \max_{\alpha, \beta: \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta)$$

- Let some w be given

- If w violated any of the primal constraints, then it follows that

$$\theta_p(w) = \infty$$

- Conversely, if the primal constraints are satisfied for a particular value of w , then $\theta_p(w) = f(w)$

- Hence,

$$\theta_p(w) = \begin{cases} f(w) & \text{if } w \text{ satisfies primal constraints} \\ \infty & \text{otherwise} \end{cases}$$

- Hence, if we consider the minimization problem

$$\min_w \theta_p(w) = \min_w \max_{\alpha, \beta: \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta)$$

we see that it is the same problem, i.e. has the same solutions, as our original, primal problem.

- We define the value of the primal problem to be the optimal value of the objective: $p^* = \min_w \theta_p(w)$

- We now consider a slightly different problem.

- Define

$$\text{"dual"} \leftarrow \theta_D(\alpha, \beta) = \min_w \mathcal{L}(w, \alpha, \beta)$$

- We now pose the dual optimization problem

$$\max_{\alpha, \beta: \alpha_i \geq 0} \theta_D(\alpha, \beta) = \max_{\alpha, \beta: \alpha_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta)$$

- We also define the optimal value of the dual problem's objective to be $d^* = \max_{\alpha, \beta: \alpha_i \geq 0} \theta_D(w)$

- The dual and primal problems are related by

$$d^* = \max_{\alpha, \beta: \alpha_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta) \leq \min_w \max_{\alpha, \beta: \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta) = p^*$$

- However, under certain conditions, we will have that

$$d^* = p^*$$

so that we can solve the dual problem instead of the primal one
- Let's see what these conditions are:

- Suppose f and the g_i 's are convex and the h_i 's are affine, i.e. $h_i(w) = a_i^T w + b_i$ for some a_i, b_i
- Suppose also that the constraints g_i 's are (strictly) feasible, i.e. there exists w s.t. $g_i(w) < 0$ for all i

- Under our above assumptions, there must exist w^*, α^*, β^* so that

- w^* is the solution to the primal problem,
- α^*, β^* are the solutions to the dual problem and
- $p^* = d^* = \mathcal{L}(w^*, \alpha^*, \beta^*)$

- Furthermore, w^*, α^* and β^* satisfy the Karush-Kuhn-Tucker (KKT) conditions, which are:

- $\frac{\partial}{\partial w_i} \mathcal{L}(w^*, \alpha^*, \beta^*) = 0, \quad i = 1, \dots, n$

- $\frac{\partial}{\partial \beta_i} \mathcal{L}(w^*, \alpha^*, \beta^*) = 0, \quad i = 1, \dots, l$

- $\alpha_i^* g_i(w^*) = 0, \quad i = 1, \dots, K \quad \longrightarrow (*)$

- $g_i^*(w^*) \leq 0, \quad i = 1, \dots, K$

- $\alpha^* \geq 0, \quad i = 1, \dots, K$

- Moreover, if some α^*, β^*, w^* satisfy the KKT conditions, then it is also a solution to the primal and dual problems

- Equation (*) is called the KKT dual complementarity condition

- It implies that if $\alpha_i^* > 0$, then $g_i(w^*) = 0$

- This will later be important when discussing SVMs

§6: Optimal margin classifiers

- We have the following primal optimization problem

$$\min_{w, b} \frac{1}{2} \|w\|^2$$

s.t. $y^{(i)} (w^T x^{(i)} + b) \geq 1, i = 1, \dots, m$

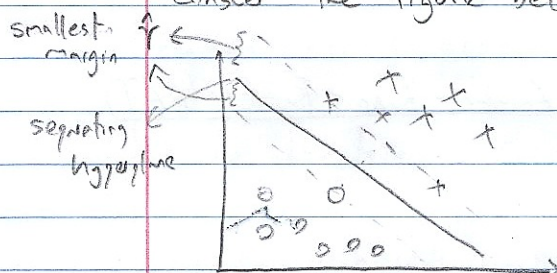
- We can write the constraints as

$$g_i(w) = -y^{(i)} (w^T x^{(i)} + b) + 1 \leq 0$$

- We have one such constraint for each training example.

- Note that from the KKT dual complementarity condition, we will have $\alpha_i > 0$ only for the training examples that have functional margin exactly equal to 1, i.e. the ones for which $g_i(w) = 0$

- Consider the figure below



- The points with the smallest margins are exactly the ones closest to the decision boundary (here we have three points)

- Furthermore, it is only for these three points that $g_i(w) = 0$ b/c we have by assumption

that the functional margin $\hat{\gamma} = 1$ and since these points attain the smallest margin, we have that $\hat{\gamma}^{(i)} = 1$ for each of these points $\Rightarrow g_i(w) = 0$

- Hence, only the three α_i 's corresponding to these points will > 0 at the optimal solution

• For any other point, $g_i(w) > 0 \Rightarrow \alpha_i = 0$

- These three points are called support vectors

• Notice how the no. of support vectors can be much smaller than the size of the training set; this will be useful.

- The Lagrangian for our optimization problem is:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^m \alpha_i [y^{(i)} (w^T x^{(i)} + b) - 1]$$

- We now find the dual form of the problem; we need to minimize $\mathcal{L}(w, b, \alpha)$ w.r.t. to w and b (for fixed α)

- Thus, we set the derivatives equal to zero:

$$\nabla_w \mathcal{L}(w, b, \alpha) = \nabla_w \left[\frac{1}{2} w w^T - \sum_{i=1}^m [\alpha_i y^{(i)} w^T x^{(i)} + \alpha_i y^{(i)} b - \alpha_i] \right]$$

• First, we have that

$$\nabla_W \frac{1}{2} W^T W = \frac{1}{2} \nabla_W \text{tr}(W^T W) = \frac{1}{2} \nabla_W \text{tr}(W W^T)$$

$$= \frac{1}{2} (W + W) \quad \text{by Prop 4 to 2.1 with } A=W, B=I, C=I$$

$$= W$$

• Also

$$\nabla_W \alpha_i y^{(i)} W^T X^{(i)} = \alpha_i y^{(i)} X^{(i)}$$

Hence

$$\nabla_W \mathcal{L}(W, b, \alpha) = W - \sum_{i=1}^n \alpha_i y^{(i)} X^{(i)} = 0$$

which implies that

$$W = \sum_{i=1}^n \alpha_i y^{(i)} X^{(i)}$$

• For the derivative w.r.t. b , we have that

$$\frac{\partial}{\partial b} \mathcal{L}(W, b, \alpha) = \sum_{i=1}^n \alpha_i y^{(i)} = 0$$

• Given the above definition of W , if we plug it into the Lagrangian,

$$\begin{aligned} \mathcal{L}(W, b, \alpha) &= \frac{1}{2} \left(\sum_{i=1}^n \alpha_i y^{(i)} X^{(i)} \right)^T \left(\sum_{j=1}^n \alpha_j y^{(j)} X^{(j)} \right) - \sum_{i=1}^n \left[\alpha_i y^{(i)} \left(\sum_{j=1}^n \alpha_j y^{(j)} X^{(j)} \right)^T X^{(i)} + \alpha_i y^{(i)} b \right] \\ &= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} (X^{(i)})^T X^{(j)} - \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} (X^{(i)})^T X^{(j)} - b \sum_{i=1}^n \alpha_i y^{(i)} \\ &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} (X^{(i)})^T X^{(j)} - \left(b \sum_{i=1}^n \alpha_i y^{(i)} \right) = 0 \quad \text{by } \frac{\partial}{\partial b} \mathcal{L} = 0 \\ &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} (X^{(i)})^T X^{(j)} \end{aligned}$$

• Putting all this together along with the constraints $\alpha_i \geq 0$ (that we always had), the dual optimization problem we obtain is

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

$$\text{s.t. } \alpha_i \geq 0, \quad i = 1, \dots, m$$

$$\sum_{i=1}^m \alpha_i y^{(i)} = 0$$

- It can also be shown that the conditions required for $p^* = d^*$ and the KKT conditions are all satisfied in our optimization problem.
- Hence we can solve the dual instead of the primal problem.

- If we can solve the dual problem and find the α 's that maximize $W(\alpha)$, we can then use the equation

$$W = \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \quad (*)$$

to find the optimal w 's

- Having found w^* , by considering the primal problem, we can also find the optimal value of the intercept term b as

$$b^* = - \frac{\max_{i: y^{(i)} = -1} W^{*T} x^{(i)} + \min_{i: y^{(i)} = 1} W^{*T} x^{(i)}}{2}$$

- Suppose we now fit our model's parameters to a training set and now wish to make a prediction at a new point input x .
- We would then calculate $w^T x + b$ and predict $y = 1$ iff this quantity > 0 .
- But using the equation $(*)$, we can rewrite:

$$\begin{aligned} w^T x + b &= \left(\sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \right)^T x + b \\ &= \sum_{i=1}^m \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b \end{aligned}$$

- Hence, the calculation only depends on $\langle x^{(i)}, x \rangle$
- Furthermore, the α 's will all be zero except for the support vectors so many of the terms will be zero.
- So we only need to calculate the inner products between x and the

support vectors.

- So our entire algorithm depends only on the inner products between input feature vectors.
- We exploit this by using kernels to our classification problem; the resulting algo. is known as support vector machines.

§7. Kernels

- Let x be an input. We considered performing regression using, for example, the features x, x^2 and x^3 to obtain a cubic function.
- To distinguish between these two sets of variables, we will call the "original" input value the input attributes of a problem (e.g. x in our case).
- When that is mapped to some quantities which are then passed to the learning algorithm, we will call these new quantities the input features.
- We also let ϕ denote the feature mapping which maps from the attributes to the features.

• For example, in our instance

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

- Rather than applying SVMs using the original input attributes x , we may instead want to learn using some features $\phi(x)$.
- To do so, we simply need to go over our previous algorithm and replace x everywhere with $\phi(x)$.
- Hence, we would replace all the inner products $\langle x, z \rangle$ with $\langle \phi(x), \phi(z) \rangle$.
- Specifically, given a feature mapping ϕ , we define the corresponding kernel to be

$$K(x, z) := \phi(x)^T \phi(z) = \langle \phi(x), \phi(z) \rangle$$

• Hence we would replace $\langle x, z \rangle$ with $K(x, z)$

- Now, given ϕ , we could easily compute $K(x, z)$ by finding $\phi(x), \phi(z)$.
- But, interestingly, $K(x, z)$ may be very inexpensive to calculate, even though $\phi(x)$ itself may be very expensive to calculate.
- In such settings, by using in our algo. an efficient way to calculate $K(x, z)$

we can get SVMs to learn in high dimensional feature space given by ϕ , but without ever having to explicitly find or represent vectors $\phi(x)$.

- For example, the kernel $K(x, z) = (x^T z + c)^d$ corresponds to a feature mapping to an $\binom{n+d}{d}$ feature space corresponding to all monomials of the form $x_1 x_2 \dots x_k$ that are up to order d .

- However, despite working in this $O(n^d)$ -dimensional space, computing $K(x, z)$ still takes only $O(n)$ time.

- Consider a slightly different view of kernels

- Intuitively (but this intuition is not 100% correct), if $\phi(x)$ and $\phi(z)$ are close together, then we might expect $K(x, z) = \phi(x)^T \phi(z)$ to be large.

- Conversely, if $\phi(x)$ and $\phi(z)$ are far apart, say nearly orthogonal to each other, then $K(x, z)$ will be small.

- So you can think of $K(x, z)$ as some measurement of how similar are $\phi(x)$ and $\phi(z)$, or of how similar are x and z .

- Given this intuition, consider the Gaussian Kernel:

$$K(x, z) = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right)$$

• If x and z are close, $\|x - z\|$ is small and so $K(x, z)$ is close to 1.

• If x and z are far apart, $\|x - z\|$ is large and so $K(x, z)$ is close to 0.

- This corresponds to an infinite dimensional feature mapping ϕ .

- But more broadly, given some function K , how can we tell if it's a valid kernel, i.e. if there exists some feature mapping ϕ s.t. $K(x, z) = \phi(x)^T \phi(z)$ for all x, z ?

- Suppose for now that K is indeed a valid kernel corresponding to some feature mapping ϕ .

- Now consider some finite set of m points (not necessarily the training set)

$$\{x^{(1)}, \dots, x^{(n)}\}$$

- Define a $n \times n$ matrix K by

$$K_{ij} = K(x^{(i)}, x^{(j)})$$

- This matrix is called the Kernel matrix

- Note that if K is a valid kernel, then

$$K_{ij} = K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)}) = (\phi(x^{(i)})^T \phi(x^{(j)})) \text{ b/c } \phi(x^{(i)})^T \phi(x^{(j)}) \text{ is a number}$$

$$= \phi(x^{(j)})^T \phi(x^{(i)}) = K_{ji}$$

- Thus K must be symmetric

- Moreover, letting $\phi_k(x)$ denote the k^{th} coordinate of the vector $\phi(x)$, we find that for any vector z

$$z^T K z = [z_1 \ z_2 \ \dots \ z_n] \begin{bmatrix} K_{11} & K_{12} & \dots & K_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ K_{n1} & \dots & \dots & K_{nn} \end{bmatrix} \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix}$$

$$= \left[\sum_i z_i K_{i1} \quad \sum_j z_j K_{j2} \quad \dots \quad \sum_l z_l K_{ln} \right] \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix}$$

$$= \sum_i \sum_j z_i K_{ij} z_j$$

$$= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j$$

$$= \sum_i \sum_j z_i \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j$$

$$= \sum_k \sum_i \sum_j z_i \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j$$

$$= \sum_k \left(\sum_i z_i \phi_k(x^{(i)}) \right)^2 \geq 0$$

- Since z was arbitrary, it follows that K is positive semi-definite, i.e. $K \geq 0$

- Thus, we have shown that

K is a valid kernel $\implies$ the Kernel matrix $K \in \mathbb{R}^{n \times n}$ is symmetric positive semi-definite

- This actually turns out to also be a sufficient condition for K to be a valid kernel, also known as a Mercer Kernel

Theorem (Mercer). Let $K: \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ be given. Then

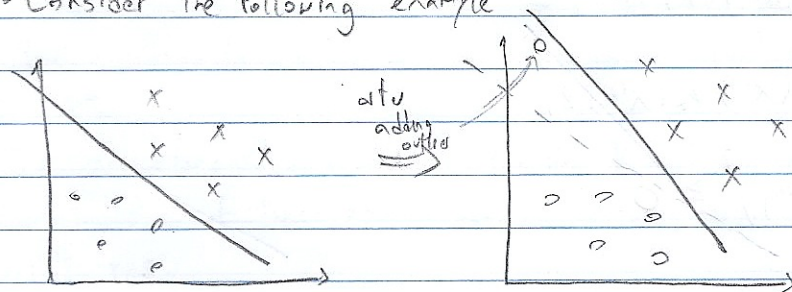
K is a valid (Mercer) kernel $\iff$ for any $\{x^{(1)}, \dots, x^{(m)}\}$ ($m < \infty$),
 K is symmetric positive semi-definite

- Thus, given a function K , we can use this theorem to check if it is a valid kernel

- In summary: using a kernel, we can "magically" allow our algorithm to work efficiently in the high dimensional space corresponding to K

§8: Regularization and the non-separable case

- The derivation of the SVM as presented so far assumes that the data is linearly separable
- While mapping data to a high dimensional feature space via ϕ does generally increase the likelihood that the data is separable, we can't guarantee this will always be so.
- Also, in some cases it is not clear that finding a separating hyperplane is exactly what we would want, since that might be susceptible to outliers:
- Consider the following example



- The left figure shows an optimal margin classifier
- When a single outlier is added (see right figure), it causes the decision boundary to make a dramatic swing, and the resulting classifier has a much smaller margin.

- To make the algo. work for non-linearly separable datasets as well as be less sensitive to outliers, we reformulate our optimization (using ℓ_1 regularization) as follows

$$\begin{aligned} \min_{w, b, \xi} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi_i \\ \text{s.t.} \quad & y^{(i)} (w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i=1, \dots, m \\ & \xi_i \geq 0, \quad i=1, \dots, m \end{aligned}$$

- Thus examples are now permitted to have (functional) margin less than 1
- But if an example has functional margin $1 - \xi_i$ (with $\xi_i > 0$), we would pay a cost of the objective function being increased by $C\xi_i$
- The parameter C controls the relative weighting between the two goals of:
 - making the $\|w\|^2$ small (which we saw earlier makes the margin large) and
 - ensuring that most examples have functional margin of at least 1

- As before, we can form the Lagrangian

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2} w^T w + C \sum_{i=1}^m \xi_i - \sum_{i=1}^m \alpha_i [y^{(i)} (w^T x^{(i)} + b) - 1 + \xi_i] - \sum_{i=1}^m r_i \xi_i$$

- Here, the α_i 's and r_i 's are our Lagrangian multipliers (constrained to ≥ 0)
- Deriving the dual in a similar way as we did before gives us:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i=1, \dots, m \\ & \sum_{i=1}^m \alpha_i y^{(i)} = 0 \end{aligned}$$

- As before, we have that

$$w^* = \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)}$$

so we can continue to use $w^T x + b = \sum_{i=1}^m \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b$ for our predictions

- Note that after the ℓ_1 regularization, the only changes to the dual problem are:
 - The constraint $0 \leq \alpha_i$ became $0 \leq \alpha_i \leq C$
 - The calculation for b^* has to be modified.

- Also, the KKT dual-complementarity conditions are

$$\bullet \alpha_i = 0 \Rightarrow y^{(i)} (w^T x^{(i)} + b) \geq 1 \quad (14)$$

$$\bullet \alpha_i = C \Rightarrow y^{(i)} (w^T x^{(i)} + b) \leq 1 \quad (15)$$

$$\bullet 0 < \alpha_i < C \Rightarrow y^{(i)} (w^T x^{(i)} + b) = 1 \quad (16)$$

- Now, all that remains is to give an algorithm for actually solving the dual problem, which we do next.

§9: The SMO algorithm

- The SMO (sequential minimal optimization) algorithm gives an efficient way of solving the dual problem arising from the derivation of the SVM
- We first discuss a bit

9.1: Coordinate ascent

- Consider trying to solve the unconstrained optimization problem

$$\max_{\alpha} W(\alpha_1, \alpha_2, \dots, \alpha_m)$$

- We have already seen two optimization algorithms: gradient ascent and Newton's method
- We now consider another algo, coordinate ascent:

Loop until convergence {

For $i = 1, \dots, m$, {

$$\alpha_i := \operatorname{argmax}_{\alpha_i} W(\alpha_1, \dots, \alpha_{i-1}, \alpha_i, \alpha_{i+1}, \dots, \alpha_m)$$

}

}

- In every iteration of the inner loop, we reoptimize W w.r.t. just the parameter α_i , while holding all other α 's fixed

- Our variable reoptimization follows the order: $\alpha_1, \dots, \alpha_m, \alpha_1, \dots$
- You may choose more sophisticated orderings.

- When the function W happens to be of such a form that the "arg max" in the inner loop can be performed efficiently, then coordinate ascent can be a fairly efficient algo.

9.2: SMO

- We now sketch the derivation of the SMO algorithm
- We have the following (dual) optimization problem we want to solve

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^m \alpha_i = \frac{1}{2} \sum_{i=2}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad i=2, \dots, m \quad (18)$$

$$\sum_{i=2}^m \alpha_i y^{(i)} = 0 \quad (19)$$

- Let's say we have a set of α_i 's that satisfy the two constraints (18-19)
- Now suppose we hold $\alpha_2, \dots, \alpha_m$ fixed and take a coordinate ascent step and reoptimize the objective w.r.t. α_1
- Can we make any progress? NO!
- This is b/c the constraint (19) ensures that

$$\alpha_1 y^{(1)} = - \sum_{i=2}^m \alpha_i y^{(i)} \Rightarrow \alpha_1 (y^{(1)})^2 = - y^{(1)} \sum_{i=2}^m \alpha_i y^{(i)}$$

$$\Rightarrow \alpha_1 = - y^{(1)} \sum_{i=2}^m \alpha_i y^{(i)} \quad \text{b/c } y^{(1)} \in \{-1, 1\} \text{ so } (y^{(1)})^2 = 1$$

- Hence, α_1 is exactly determined by the other α_i 's
- Hence, if we were to hold $\alpha_2, \dots, \alpha_m$ fixed, then we can't change α_1 without violating constraint (19).

- Thus, if we want to update some of the α_i 's, we must update at least two simultaneously in order to keep satisfying the constraints

- This motivates the SMO algorithm:

Repeat until convergence {

1. Select some pair α_i and α_j to update next (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum)

2. Reoptimize $W(\alpha)$ w.r.t. α_i and α_j , while holding all the other α_k 's ($k \neq i, j$) fixed.

}

- To test for convergence of this algo, we can check whether the KKT conditions (eqs. 14-16) are satisfied to within some tol
- tol is the convergence tolerance parameter and is typically $0.001 \leq \text{tol} \leq 0.01$

- The key reason that SMO is efficient is that the update to α_1, α_2 can be computed very efficiently.

- We now sketch the main ideas for deriving the efficient update

- Suppose we have some setting of the α_i 's that satisfy the constraints (18-19) and suppose we decide to hold $\alpha_3, \dots, \alpha_n$ fixed. We want to reoptimize $W(\alpha_1, \dots, \alpha_n)$ w.r.t. α_1, α_2 (subject to the constraints)

- From (19), we require that

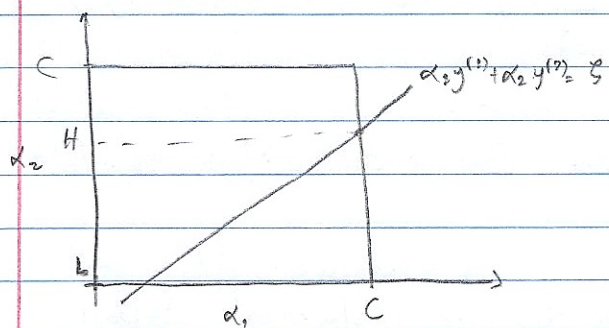
$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = - \sum_{i=3}^n \alpha_i y^{(i)} \quad (20)$$

- Since the RHS is fixed, we can just let it be denoted by some constant ξ

- Thus,

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \xi$$

- We can thus picture the constraints on α_1 and α_2 as follows:



- From constraint (18), we now that

$(\alpha_1, \alpha_2) \in [0, C] \times [0, C]$, i.e. lie

within the box

- Also plotted is the line $\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \xi$ on which α_1, α_2 must lie

$\rightarrow (L=0 \text{ in this case})$

- Note also that, from these constraints, we know $L \leq \alpha_2 \leq H$; otherwise (α_1, α_2) can't simultaneously satisfy both the box and the straight line constraint.

- We can rewrite equation (20) as follows

$$\begin{aligned} \alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \xi &\Rightarrow \alpha_1 y^{(1)} = \xi - \alpha_2 y^{(2)} \\ &\Rightarrow \alpha_1 (y^{(1)})^2 = y^{(1)} (\xi - \alpha_2 y^{(2)}) \end{aligned}$$

$$\Rightarrow \alpha_2 = (\xi - \alpha_2 y^{(2)}) y^{(2)} \quad \text{b/c } y^{(i)} \in \{-1, 1\} \text{ so } (y^{(i)})^2 = 1$$

- Thus, we can rewrite:

$$W(\alpha) = W((\xi - \alpha_2 y^{(2)}) y^{(2)}, \alpha_2, \dots, \alpha_n)$$

$$= (\xi - \alpha_2 y^{(2)}) y^{(2)} + \alpha_2 + \sum_{i=3}^n \alpha_i - \frac{1}{2} \left[(y^{(2)} y^{(2)} (\xi - \alpha_2 y^{(2)}) y^{(2)} \alpha_2 \langle x^{(2)}, x^{(2)} \rangle) + \sum_{\substack{j=1 \\ j \neq 2}}^n y^{(1)} y^{(j)} \alpha_j (\xi - \alpha_2 y^{(2)}) y^{(2)} \langle x^{(2)}, x^{(j)} \rangle + \sum_{\substack{j=2 \\ j \neq 3}}^n y^{(j)} y^{(2)} \alpha_j \alpha_2 \langle x^{(j)}, x^{(2)} \rangle + \sum_{\substack{i=3 \\ j=3}}^n y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \right]$$

- Treating $\alpha_3, \dots, \alpha_n$ as constants, we can see that this is just some quadratic function in α_2 , i.e. $W(\alpha) = a\alpha_2^2 + b\alpha_2 + c$ for some appropriate constants a, b and c .

- If we ignore the box constraints, we can easily maximize this quadratic function by setting its derivative to zero and solving.

- Let $\alpha_2^{\text{new, unclipped}}$ denote the resulting value of α_2 .

- Notice that we can maximize W w.r.t. to α_2 but subject to the box constraint by taking $\alpha_2^{\text{new, unclipped}}$ and "clipping" it to lie in the interval $[L, H]$.

- Thus,

$$\alpha_2^{\text{new}} = \begin{cases} H & \text{if } \alpha_2^{\text{new, unclipped}} > H \\ \alpha_2^{\text{new, unclipped}} & \text{if } L \leq \alpha_2^{\text{new, unclipped}} \leq H \\ L & \text{if } \alpha_2^{\text{new, unclipped}} < L \end{cases}$$

- Finally, having found the α_2^{new} , we can use Equation (29) to go back and find the optimal value of α_2^{new} .

Lecture notes #4: Learning Theory

Part VI: Learning Theory

§1: Bias / variance tradeoff

- The generalization error of an hypothesis is its expected error on examples not necessarily in the training set
- There are various ways of having large generalization errors:
- Suppose we attempt to model y as a linear function of x , i.e. $y = \theta_0 + \theta_1 x$, but x and y don't have a linear relationship.
- In this case, even if we were fitting this linear model to a very large amount of training, the linear model would still fail to accurately capture the structure in the data.
- Informally, we define the bias of a model to be the expected generalization error even if we were to fit to a very (say, infinitely) large training set.
- In the above example, we would say that it suffers from large bias and may underfit (i.e. fail to capture structure exhibited by) the data.
- There is a second component to the generalization error, consisting of the variance of a model fitting procedure
- For instance, suppose we now model y with a feature x by $y = \theta_0 + \theta_1 x + \dots + \theta_5 x^5$
- There is now a large risk that we are fitting patterns in the data that happened to be present in our small, finite training set, but that don't actually reflect the wider pattern of the relationship between x and y
- By fitting these "spurious" patterns in the training set, we might again obtain a model with large generalization error
- In this case we say the model has large variance.
- Often there is a tradeoff between bias and variance.

§2: Preliminaries

- We now begin discussing learning theory
- This will allow us to derive rules of thumb about how to best apply learning algorithms in different settings.
- We will also seek to answer some questions:
 - Can we make formal the bias/variance tradeoff?
 - * This will lead us to talk about model selection methods, which can, e.g., automatically decide what order polynomial to fit to a training set
 - In ML, what we really care about is the generalization error but most learning algos. fit their models to the training set.
 - * Why should doing well on the training set tell us anything about the generalization error?
 - * Can we relate error on the training set to generalization error?
 - * Why should doing well on the training set tell us anything about generalization error?
 - Are there conditions under which we can actually prove the learning algos. work well?

Lemma (The union bound). Let $A_1, \dots, A_K$ be K different events (that may not be independent). Then

$$P(A_1 \cup \dots \cup A_K) \leq P(A_1) + \dots + P(A_K)$$

Lemma (Hoeffding inequality) Let $Z_1, \dots, Z_m$ be m iid random variables drawn from a Bernoulli(ϕ) dist., i.e. $P(Z_i = 1) = \phi$ and $P(Z_i = 0) = 1 - \phi$. Let

$$\hat{\phi} = \frac{1}{m} \sum_{i=1}^m Z_i$$

be the mean of these random variables. Also, let $\gamma > 0$. Then

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

- This is also known as the Chernoff bound
- This tells us that $\hat{\phi}$ is a good estimate of ϕ when m is large

- For simplicity, we consider only binary classification in which the labels are $y \in \{0, 1\}$

• But everything discussed generalizes to other problems, like regression and multi-class classification.

- Let $S = \{(x^{(i)}, y^{(i)}) : i = 1, \dots, m\}$ be a training set where the training examples $(x^{(i)}, y^{(i)})$ are drawn iid from some prob. dist. $\mathcal{D}$

- For a hypothesis h , we define the training error (or empirical risk or empirical error) to be

$$\hat{E}_S(h) = \hat{E}(h) = \frac{1}{m} \sum_{i=1}^m \mathbb{1}\{h(x^{(i)}) \neq y^{(i)}\}$$

- We define the generalization error to be

$$E(h) = P_{(x,y) \sim \mathcal{D}}(h(x) \neq y)$$

• This is the prob. that, if we now draw a new example (x, y) from the distribution $\mathcal{D}$, h will misclassify it

- The PAC (probably approximately correct) assumptions is a framework and set of assumptions under which numerous results on learning theory were proved.

- Of these assumptions, the most important ones are:

(1) The training and testing data are all drawn from the same dist. $\mathcal{D}$

(2) The training examples are independently drawn.

- Consider the setting of linear classification and let

$$h_{\theta}(x) = \mathbb{1}\{\theta^T x \geq 0\}$$

- What is a reasonable way of fitting the parameters θ ?

- One approach is to minimize the training error

$$\hat{\theta} = \arg \min_{\theta} \hat{E}(\theta)$$

- This process is called empirical risk minimization (ERM)

This is
used
here

- The hypothesis output by this learning algo. is $\hat{h} = h_{\hat{\theta}}$
- We think of ERM as the most "basic" learning algo. and it is the one we now focus on.
- In our study of learning theory, it is useful to consider problems (such as the one above) more abstractly
- We define the hypothesis class H used by a learning algo to be the set of all classifiers considered by the algo.
 - For linear classification, $H = \{h_{\theta} : h_{\theta}(x) = \mathbb{1}\{\theta^T x \geq 0\}, \theta \in \mathbb{R}^{n+1}\}$ is $\mathcal{H}_3$
- The set of all classifiers X with linear decision boundary
- ERM can now be thought of as a minimization over the class of functions H in which the algo. picks the hypothesis:

$$\hat{h} = \underset{h \in H}{\operatorname{argmin}} \hat{\mathcal{E}}(h)$$

§3: The case of finite H

- Suppose we have a finite hypothesis class $H = \{h_1, \dots, h_K\}$
 - This, H is just a set of K functions from X to $\{0, 1\}$
- ERM selects $\hat{h}$ to be whichever one of these K functions minimizes the training error.
- We would like to give guarantees on the generalization error of $\hat{h}$
- To do so, we follow two steps:
 - (1) We show that $\hat{\mathcal{E}}(h)$ is a reliable estimate of $\mathcal{E}(h)$ for all h
 - (2) We show that this implies an upper bound on the generalization error $\mathcal{E}(\hat{h})$
- Take any one, fixed, $h_i \in H$
 - Consider a Bernoulli random variable Z whose dist. is defined as follows:
 - We are going to sample $(x, y) \sim \mathcal{D}$
 - Set $Z = \mathbb{1}\{h_i(x) \neq y\}$ i.e. We draw one example and let Z indicate if h_i misclassifies it.

• Similarly, define

$$Z_j = 1\{h_i(x^{(j)}) = y^{(j)}\}$$

• Since our training set was drawn iid from $\mathcal{D}$, Z and the Z_j 's have the same dist

- We see that the misclassification prob. on a randomly drawn example, i.e. $E(h)$, is exactly $E(Z)$ (and $E(Z_j)$)

- Moreover, the training error can be written

$$\hat{E}(h_i) = \frac{1}{M} \sum_{j=1}^M Z_j$$

- Thus, $\hat{E}(h_i)$ is exactly the mean of the M random variables Z_j that are drawn iid from a Bernoulli dist. with mean $E(h_i)$

- We can thus apply Hoeffding's inequality

$$P(|\hat{E}(h_i) - E(h_i)| > \gamma) \leq 2e^{-2\gamma^2 M}$$

- Thus the training error will be close to the generalization error when M is large for this particular hypothesis h_i

- But we want to show that this will be true for all $h \in H$

- Let A_i denote the event that $|E(h_i) - \hat{E}(h_i)| > \gamma$

- We have shown that for any A_i ,

$$P(A_i) \leq 2e^{-2\gamma^2 M}$$

- Thus, using the union bound:

$$P(\exists h \in H. |E(h) - \hat{E}(h)| > \gamma) = P(A_1 \cup \dots \cup A_K)$$

$$\leq \sum_{i=1}^K P(A_i)$$

$$= 2K e^{-2\gamma^2 M}$$

- Subtracting both sides from 1:

$$P(\neg \exists h \in H. |\varepsilon(h) - \hat{\varepsilon}(h)| > \gamma) = P(\forall h \in H. |\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma)$$

$$\geq 1 - 2K e^{-2\gamma^2 m}$$

- Thus, with prob. at least $1 - 2K e^{-2\gamma^2 m}$ we have that $\varepsilon(h)$ will be within γ of $\hat{\varepsilon}(h)$ for all $h \in H$

- This is called a uniform convergence test b/c this is a bound that holds simultaneously for all $h \in H$

- The two quantities of interest: m , γ and the prob. of error

- We can bound either one in terms of the other.

- For instance: given γ and some $\delta > 0$, how large must m be before we can guarantee that with prob. at least $1 - \delta$, the training error will be within γ of the generalization error?

- Setting $\delta = 2K e^{-2\gamma^2 m}$, we have that:

$$e^{-2\gamma^2 m} = \frac{\delta}{2K} \Rightarrow -2\gamma^2 m = \log \frac{\delta}{2K}$$

$$\Rightarrow m = \frac{1}{2\gamma^2} \log \frac{2K}{\delta}$$

$$\Rightarrow m = \frac{1}{2\gamma^2} \log \left(\frac{2K}{\delta} \right) = \frac{1}{2\gamma^2} \log \frac{2K}{\delta}$$

- We then have that for all $m \geq \frac{1}{2\gamma^2} \log \frac{2K}{\delta}$,

$$P(\forall h \in H. |\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma) \geq 1 - 2K e^{-2\gamma^2 m}$$

$$\geq 1 - 2K e^{-2\gamma^2 \left[\frac{1}{2\gamma^2} \log \frac{2K}{\delta} \right]}$$

$$= 1 - \delta$$

- Hence if $m \geq \frac{1}{2\gamma^2} \log \frac{2K}{\delta}$, then with prob. at least $1 - \delta$, we have that $|\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma$ for all $h \in H$

- The training set size m that a certain method or algo. requires in order to achieve a certain level of performance is called the algo's sample complexity

- The key property of the bound above is that the no. of training examples needed to make this guarantee is only logarithmic in K

- Similarly, hold m and δ fixed

• Again, we have that $\delta = 2K e^{-2\gamma^2 m}$ and solving for γ :

$$\delta = 2K e^{-2\gamma^2 m} \Rightarrow -2\gamma^2 m = \log \frac{\delta}{2K}$$

$$\Rightarrow \gamma = \sqrt{\frac{1}{2m} \log \frac{2K}{\delta}}$$

- We then have that

$$P(\forall h \in H, |\epsilon(h) - \hat{\epsilon}(h)| \leq \sqrt{\frac{1}{2m} \log \frac{2K}{\delta}})$$

$$\geq 1 - e^{-\left[\sqrt{\frac{1}{2m} \log \frac{2K}{\delta}}\right]^2 m}$$

$$= 1 - \delta$$

• Thus, we have with probability $1 - \delta$, that for all $h \in H$

$$|\epsilon(h) - \hat{\epsilon}(h)| \leq \sqrt{\frac{1}{2m} \log \frac{2K}{\delta}}$$

$$\text{... } |\epsilon(h) - \hat{\epsilon}(h)| \leq \gamma \quad \forall h \in H$$

- Now, assuming that the uniform convergence result holds, what can we prove about the generalization^{error} of our learning algo. that picked $\hat{h} = \operatorname{argmin}_{h \in H} \hat{\epsilon}(h)$

- Define $h^* = \operatorname{argmin}_{h \in H} \epsilon(h)$ to be the best possible hypothesis in H .

• Note that h^* is the best that we could possibly do given that we are using H , so it makes sense to compare our performance with h^*

- So,

$$\begin{aligned} \epsilon(\hat{h}) &\leq \hat{\epsilon}(\hat{h}) + \gamma && \text{b/c } |\epsilon(\hat{h}) - \hat{\epsilon}(\hat{h})| \leq \gamma \text{ by assumption} \\ &\leq \hat{\epsilon}(h^*) + \gamma && \text{b/c } \epsilon(\hat{h}) \leq \epsilon(h) \quad \forall h \in H \text{ by definition of } \hat{h} \\ &\leq \epsilon(h^*) + \gamma && \text{b/c } \end{aligned}$$

- This shows that: if uniform convergence occurs, the generalization error

of $\hat{h}$ is at most 2γ worse than the best possible hypothesis in H !
- In short,

Theorem. Let $|H| = K$ and let m, δ be fixed. Then with probability $1 - \delta$, we have that

$$E(\hat{h}) \leq \left(\min_{h \in H} E(h) \right) + 2 \sqrt{\frac{1}{2m} \log \frac{2K}{\delta}}$$

- This also quantifies the bias/variance tradeoff in model selection.

- Specifically, suppose we have some hypothesis class H and we are considering switching to some larger hypothesis class $H' \supseteq H$.

- If we switch to H' , then the term $\min_{h \in H} E(h)$ can only decrease.

- Hence by using a larger hypothesis class, our "bias" can only decrease.

- However, by switching to H' , K increases and so the term $2\sqrt{\cdot}$ also increases.

- This increase corresponds to our "variance" increasing when we use a larger hypothesis class.

- Finally, we can also obtain the following sample complexity bound.

Corollary. Let $|H| = K$ and let δ, γ be fixed. Then for

$$E(\hat{h}) \leq \min_{h \in H} E(h) + 2\gamma$$

to hold with prob. at least $1 - \delta$, it suffices that

$$m \geq \frac{1}{2\gamma^2} \log \frac{2K}{\delta} = O\left(\frac{1}{\gamma^2} \log \frac{K}{\delta}\right)$$

§4: The case of infinite H

- Many hypothesis classes, including any parametrized by real numbers (as in linear regression) actually contain an infinite number of functions.

- Can we prove similar results for this setting?

- We first consider an argument that is not the "right" one, but it is useful to gain intuition.

- Suppose we have an H that is parametrized by d real numbers

- Since we are using a computer to represent real numbers, and IEEE double-precision floating point uses 64 bits to represent a floating point number, this means that our learning algo. is parametrized by $64d$ bits (assuming we are using double precision floating point)

- Thus, our hypothesis class really consists of at most $V = 2^{64d}$ hypotheses

- From the Corollary at the end of the previous section, to guarantee that

$$\mathbb{E}(\hat{h}) \leq \mathbb{E}(h^*) + 2\gamma$$

holds with prob. at least $1-d$, it suffices that

$$M \geq O\left(\frac{1}{\gamma^2} \log \frac{2^{64d}}{d}\right) = O\left(\frac{1}{\gamma^2} \log \frac{1}{d}\right) = O_{\gamma,d}(d)$$

(where the γ, d subscripts indicate that the big- O is hiding constants that may depend on γ and d)

- Thus, the no. of training examples need is at most linear in the parameters of the model.

- The fact that we relied on 64-bit floating point makes this argument rather unsatisfying, but the conclusion is nonetheless roughly correct.

- The other part that is unsatisfying about our argument is that it relies on the parametrization of H .

- Intuitively, however, the no. of parameters shouldn't be a factor

• Consider the class of linear classifiers in n dimensions:

$$h_{\theta}(x) = \{ \sum \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n \geq 0 \}$$

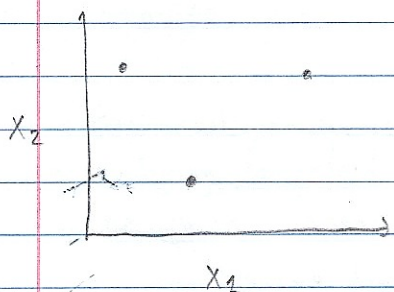
or

$$h_{w,r}(x) = \{ \sum (w_0^2 - r_0^2) + (w_1^2 - r_1^2) x_1 + \dots + (w_n^2 - r_n^2) x_n \geq 0 \}$$

• Both of these define the same hypothesis class, yet the first definition uses $n+1$ parameters while the second uses $2n+2$ parameters.

- To derive a more satisfying argument, we need some definitions.
- Given a set $S = \{x^{(1)}, \dots, x^{(d)}\}$ (no relation to the training set) of points $x^{(i)} \in X$, we say that H shatters S if H can realize any labeling on S , i.e. if for any set of labels $\{y^{(1)}, \dots, y^{(d)}\}$, there exists some $h \in H$ so that $h(x^{(i)}) = y^{(i)}$ for all $i = 1, \dots, d$.
- Given a hypothesis class H , we define its Vapnik-Chervonenkis dimension, written $VC(H)$, to be the size of the largest set that is shattered by H .
- If H can shatter arbitrarily large sets, then $VC(H) = \infty$.

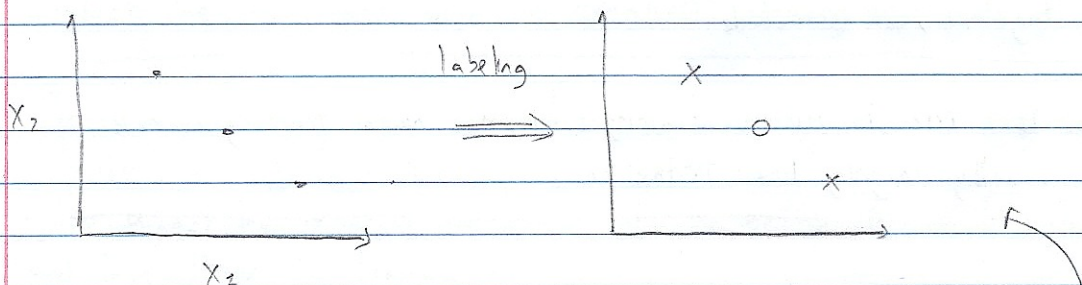
- For instance, consider the following set of three points



- Can the set H of linear classifiers in two dimensions ($h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0$) can shatter the set above?

- The answer is yes; for any of the eight possible labelings of these points, we can find a linear classifier that obtains "zero training error" (i.e. it classifies them exactly as given by the labels) on them.

- Moreover, it is possible to show that there is no set of 4 points that this hypothesis can shatter.
- Thus the largest set H can shatter is of size 3, and hence $VC(H) = 3$.
- Note that the dimension of H is 3 even though there may be sets of size 3 that it can't shatter.



- There is no way to find a linear separator for the labeling above.
- In other words, to show that $VC(H) \geq d$, we need to show only that there is

at least one set of size d that H can shatter.

- This is one of the most, or the most, important theorem in all of learning theory

Theorem. Let H be given and let $d = VC(H)$. Then with probability at least $1 - \delta$, we have that for all $h \in H$,

$$|\mathcal{E}(h) - \mathcal{E}(\hat{h})| \leq O\left(\sqrt{\frac{d}{M} \log \frac{M}{\delta} + \frac{1}{M} \log \frac{1}{\delta}}\right)$$

Thus, with probability at least $1 - \delta$, we also have that

$$\mathcal{E}(\hat{h}) \leq \mathcal{E}(h^*) + O\left(\sqrt{\frac{d}{M} \log \frac{M}{\delta} + \frac{1}{M} \log \frac{1}{\delta}}\right)$$

- In other words, if a hypothesis class has finite VC dimension, then uniform convergence occurs as M becomes large.

Corollary. For $|\mathcal{E}(h) - \hat{\mathcal{E}}(h)| \leq \gamma$ to hold for all $h \in H$ (and hence $\mathcal{E}(\hat{h}) \leq \mathcal{E}(h^*) + 2\gamma$) with probability at least $1 - \delta$, it suffices $M = O_{\gamma, \delta}(d)$

- In other words, the no. of training examples needed to learn "well" using H is linear in the VC dimension of H

- Also, it turns out that, for "most" hypothesis classes, and, assuming a "reasonable" parametrization, is also roughly linear in the no. of parameters

- Putting this together, we conclude:

(For an algo. that tries to minimize training error) the no. of training example needed is usually roughly linear in the no. of parameters of H .

Lecture notes #5: Regularization and Model Selection

Part VII: Regularization and Model Selection

- Suppose we are trying to select among several different models for a learning problem

• E.g. We might be using a polynomial linear regression model

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_K x^K$$

and wish to decide if K should be $0, 1, \dots, 10$

- How can we automatically select a model that represents a good tradeoff between bias and variance?

→ Other examples would be: choosing the bandwidth parameter τ for locally weighted regression, or the parameter C for our L_1 -regularized SVM.

- Suppose we have a finite set of models $M = \{M_1, \dots, M_d\}$ that we are trying to select from

• The generalization to infinite M is not hard.

§1: Cross validation

- Suppose we are given a training set S

- Given that we know about ERM, we could try using it to select the best model as follows:

(1) Train each model S_i to get some hypothesis h_i

(2) Pick the hypothesis with the smallest training error.

- This algo. does NOT work!

• Consider choosing the order of a polynomial

• The higher the order of the polynomial, the better it will fit the set S and thus lower the training error

• Hence this method will always select a high-degree, high-variance polynomial model, which would be a poor choice.

- The following algo, known as hold-out cross validation or simple cross-validation, works better:

(1) Randomly split S into S_{train} (say, 70% of the data) and S_{cv} (the remaining 30%). Here S_{cv} is called the hold-out cross validation set.

(2) Train each model M_i on S_{train} only, to get some hypothesis h_i .

(3) Select and output the hypothesis h_i that had the smallest error $\hat{E}_{S_{\text{cv}}}(h_i)$ for the empirical error of h_i on S_{cv} , on the hold-out cross validation set.

- By testing on a set of examples S_{cv} that the models were not trained on, we obtain a better estimate of each hypothesis h_i 's true generalization error.

- We then pick the hypothesis with the smallest estimated generalization error.

- Optionally, after picking h_i s.t. $\arg\min \hat{E}_{S_{\text{cv}}}(h_i)$, we can then retrain M_i on the entire training set S .

• This offers a good idea, except when we are using learning algorithms that are very sensitive to perturbations of initial conditions and/or data.

- The disadvantage of using hold-out cross validation is that it "wastes" about 30% of the data, since the models are only trained with 70% of it.

- Even if we use $\arg\min$, we would still be selecting among models who were only trained on $0.7 \cdot |S|$ training examples.

- Although this is fine if data is abundant, if it is scarce (say, $m = 20$), we would like to do better.

- Another method is K-fold cross validation.

(1) Randomly split S into K disjoint subsets of m/K training examples each, resulting in subsets $S_1, \dots, S_K$.

(2) For each model M_i , do as follows:

For $j = 1, \dots, K$

• Train the model M_i on $S_1 \cup S_2 \cup \dots \cup S_j \cup S_{j+2} \cup \dots \cup S_K$ (i.e. train on all data except S_j) to get some hypothesis h_{ij} .

- Test the hypothesis h_{ij} on S_j to get $\hat{E}_{S_j}(h_{ij})$
- The estimated generalization error of model M_i is then calculated as the avg. of the $\hat{E}_{S_j}(h_{ij})$ (averaged over j)
- (3) Pick the model M_i with the lowest estimated generalization error and retrain the model on the entire training set S . The resulting hypothesis is our answer.

- A typical choice is $K=10$
- If data is scarce, we will sometimes use $K=m$ in order to leave out as little data as possible
 - In this case, all but one training example is used to train the model
 - This special case is known as the leave-one-out cross validation
- Finally, notice we can use cross validation to simply evaluate a single model or algorithm
 - ~~For~~ we want to estimate how well a learning algorithm we implemented actually performs

§2: Feature Selection

- One special and important case of model selection is called feature selection
- Suppose you have a supervised learning problem with a ^{very} large number of features n (perhaps $n \gg M$), but you believe that only a small subset of them are "relevant" to the learning task.
- Even if we use a simple linear classifier over the n input features, the VC dimension of our hypothesis class would still be $O(n)$ and thus overfitting would be a problem unless the training set is fairly large.
- In such a setting, we can apply a feature selection algo. to reduce the no. of features
- Given n features, there are 2^n possible feature subsets, and thus feature selection can be considered as a model selection problem over 2^n possible models
- For n large, it is usually too expensive to explicitly enumerate and compare

all 2^n models

- Therefore, usually some heuristic search procedure is used to find a good feature subset

- The following search procedure is called forward search:

(1) Initialize $F = \emptyset$

(2) Repeat $\{$

(a) For $i = 1, \dots, n$ if $i \notin F$, let $F_i = F \cup \{i\}$

Use some version of cross-validation to evaluate features F_i (i.e. train your learning algo. using only the features in F_i and estimate its generalization error)

(b) Set F to be the best feature subset found on step (a)

$\}$

(3) Select and output the best feature subset that was evaluated during the entire search procedure

- The outer loop of the algo (i.e. (2)) can be terminated when $F = \{1, \dots, n\}$ is the set of all features, or when $|F|$ exceeds some pre-set threshold (corresponding to the max. no. of features you want the algo. to consider using)

- This algo. described above is one instantiation of wrapper model feature selection, since it is a procedure that "wraps" around your learning algorithm and repeatedly makes calls to the learning algo. to evaluate how well it does using different feature subsets.

- Besides forward search, we have backwards search which starts off as $F = \{1, \dots, n\}$ as the set of all features and repeatedly deletes one feature at a time until $F = \emptyset$

- Wrapper feature selection algos. often work quite well, but can be computationally expensive due to the many calls that are made to the learning algos.

- For instance, complete forward search, i.e. forward search with $F = \{1, \dots, n\}$, would take about $O(n^2)$ calls to the learning algo.

- Filter feature selection methods give heuristic, but computationally much cheaper, ways of choosing a feature subset.
- The idea is to compute some simple score $S(i)$ that measures how informative each feature X_i is about the class labels y .
- We then simply pick the K features with the highest scores $S(i)$.
- One possible choice of score would define $S(i)$ to be (the abs. value of) the correlation between X_i and y , as demonstrated by the training data.
- In practice, it is more common (especially for discrete-valued X_i) to choose $S(i)$ to be the mutual information $MI(X_i, y)$ between X_i and y .

$$MI(X_i, y) = \sum_{x_i \in \mathcal{X}_i} \sum_{y \in \mathcal{Y}} p(x_i, y) \log \frac{p(x_i, y)}{p(x_i)p(y)}$$

(more generally, the summations would be over the domains of the variables)

- The probabilities $p(x_i, y)$, $p(x_i)$, $p(y)$ can all be estimated according to their empirical dists. on the training set.
- To gain intuition about this score, note that the mutual information can also be expressed as a Kullback-Leibler (KL) divergence:

$$MI(X_i, y) = KL(p(x_i, y) \parallel p(x_i)p(y))$$

- Informally, this gives a measurement of how different the prob. dists. $p(x_i, y)$ and $p(x_i)p(y)$ are.
- If X_i, y are independent random variables, we would have $p(x_i, y) = p(x_i)p(y)$ and the KL-divergence between the two dists. would be zero.
 - This is consistent with the idea that if X_i and y are independent, then X_i is clearly very "non-informative" about y , and thus the score $S(i)$ should be small.
- Conversely, if X_i is very "informative" about y , then their mutual information $MI(X_i, y)$ would be large.

- One last detail: now that we have ranked the features according to their scores $S(i)$, how do we decide how many features K to choose?

- One standard way to do so is to use cross-validation to select among possible values of K

§3: Bayesian statistics and regularization

- We now discuss one more tool to prevent overfitting
- At the beginning of these notes, we discussed about parameter fitting using maximum likelihood (ML) and chose our parameters according to:

$$\theta_{ML} = \arg \max_{\theta} \prod_{i=1}^M p(y^{(i)} | x^{(i)}; \theta)$$

- Throughout our subsequent discussions, we viewed θ as a constant-valued but unknown parameter of the world
 - This is the view taken in frequentist statistics.
- In this view, θ is not random; it just happens to be unknown.
- It is our job to come up with statistical procedures (such as ML) to try to estimate this parameter
- Another approach to our parameter estimation problem is to take the Bayesian view of the world and think of θ as being a random variable whose value is unknown
- In this approach, we would specify a prior distribution $p(\theta)$ on θ that expresses our "prior beliefs" about the parameters.
- Given a training set $S = \{(x^{(i)}, y^{(i)}) : i = 1, \dots, m\}$, we can then compute the posterior distribution of the parameters:

$$p(\theta | S) = \frac{p(S | \theta) p(\theta)}{p(S)} = \frac{\left(\prod_{i=1}^M p(y^{(i)} | x^{(i)}; \theta) \right) p(\theta)}{\int_{\theta} \left(\prod_{i=1}^M p(y^{(i)} | x^{(i)}; \theta) \right) p(\theta) d\theta} \quad (1)$$

- In the equation above, $p(y^{(i)} | x^{(i)}; \theta)$ comes from whatever model you are using for your learning algo.

- E.g. if we are using Bayesian logistic regression, we might choose

$$p(y^{(i)} | x^{(i)}; \theta) = h_{\theta}(x^{(i)})^{y^{(i)}} (1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}} \text{ where } h_{\theta}(x^{(i)}) = \frac{1}{1 + e^{-\theta^T x}}$$

- When we are given a new test example x and asked to make a prediction on it, we can compute our posterior distribution on the class label using the posterior distribution on θ .

$$p(y|x, S) = \int_{\theta} p(y|x, \theta) p(\theta|S) d\theta \quad (2)$$

where $p(\theta|S)$ comes from Eq. (1)

- Thus, for example, if the goal is to predict the expected value of y given x , we would output

$$E[y|x, S] = \sum_y y p(y|x, S) \quad \text{for discrete-valued } y$$

$$E[y|x, S] = \int y p(y|x, S) dy \quad \text{for } \mathbb{R}\text{-valued } y$$

- Unfortunately, it is usually very difficult to compute the posterior dist. $p(\theta|S)$
 - This is b/c it requires taking integrals over the (usually high-dimensional) θ as in Eq. (1) and this typically can't be done in closed-form.
- Thus, in practice we will instead approximate the posterior distribution for θ .
- One common approx. is to replace our posterior dist. for θ (as in Eq. (2)) with a single point estimate.
- The MAP (maximum a posteriori) estimate for θ is given by

$$\theta_{\text{MAP}} = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^m p(y^{(i)}|x^{(i)}, \theta) p(\theta)$$

- Note that this is the same formula as for the ML estimate for θ , except for the prior $p(\theta)$ term at the end

- In practical applications, a common choice for the prior $p(\theta)$ is to assume that $\theta \sim N(0, \tau^2 I)$

- Using this choice of prior, the fitted parameters θ_{MAP} will have smaller norm than that selected by maximum likelihood
- In practice, this causes the Bayesian MAP estimate to be less susceptible to overfitting than the ML estimate of the parameters
 - E.g. Bayesian logistic regression turns out to be an effective algo. for text classification, even though in text classification we usually have $n \gg m$