Part 3: Learning Methods

We now turn to a very general class of problems whose core ideas are relevant to pretty much everything we do in computational biology. Informally, we are given a set of objects, on which we have some measurements, and we would like to predict a property (or a set of properties) that cannot be directly measured. For example, the objects might be coding sequences, the measurements might be things like GC content, amino-acid hydrophobicity, etc, and the property of interest might be whether the protein will catalyze a phosphorylation reaction; or, perhaps the property of interest might be "belonging to gene family X, Y, or Z."

For the above problem a critical element is the measurements we take on the objects. For example, given a coding sequences we can "measure" (or compute) things like GC content or hydrophobicity; but we might also consider doing things like matching the sequence to known 3D sequences and then measuring some geometrical aspect of the 3D structure. The main concern is that these properties that we measure should be something that is naturally related to the property of interest. Unfortunately, most of the time we don't have a good way of knowing what is and what is not naturally related to the property we want to predict. We will call a measurement that we use to predict the property of interest, a "feature" following the common terminology from the machine-learning field. The set of measurements, or the features, are the input data. Without real loss of generality, we will assume that the features are all numerical. That is, the set of features measured on a given object generates a vector of numbers. We can pretty much always find a numerical representation for most measurement of interest, e.g., we can represent a particular sequence motif "AAT" with 1/0 for presence/absence. With some abuse of terminology, we will call the set of features the "feature space". Each object is associated with a vector of numbers, one number for each feature, which comprise the coordinates of the objects in the feature space. Recall our discussion of geometry in Unit 14. The features that we measure comprises the coordinates and geometry of the embedded space. As we previously discussed, we generally want to characterize the natural biological geometry within this embedding—but, we first have to make sure that the feature space actually contains the biological geometry. That is, we want the features to be relevant to the biological modeling problem. The main take home message for the feature space is that there is no principled manner to determine what is the right set of features to comprise the feature space; it depends on the problem—that is, the predicted property of interest. We must use biological intuition as well as examine the results to assess the utility of one set of features versus another. In more advanced topics, one may try to come up with an (necessarily ad hoc) algorithm for "feature selection." We will not discuss such algorithms here.

One somewhat simplified view of learning problems is to think of it as a big regression problem—something that most of you already know. In a regression problem, there is a variable of interest, which we call "the dependent variable" and often use "y" or "z" to denote it. This variable carries the information that we would like to "learn". For example, y might represent "risk of cancer". We would like to model the dependent variable as a function of a set of "independent variables", often denoted by "x_i". Theses x's comprise our feature space. The "learning machine" tries to find the best mathematical association between the x's and the information of interest, the y variable. In the standard regression problem, we make some assumptions about various kinds of noises in the system and use the least-squares estimator to directly solve for the best fitting regression model. In the more general learning setting, we might have different ideas about the type of noise or how the observations are sampled and therefore use some other algorithm to solve for the best fitting model—often times such algorithms are incremental in that they optimize for each piece of new observation.

The various learning situations then have the same problem structure as the regression problem but with technical variation on the types of variables for the "dependent" and the "independent" variables parts.

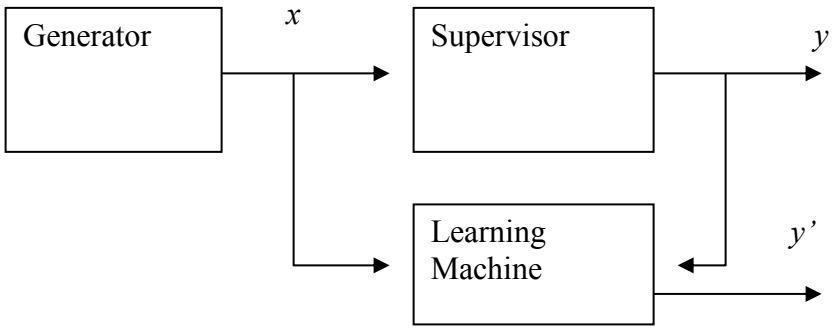
Here are some variations based on the type of the variables and what we call the analysis:

Dependent variable	Independent variable	Typical Name for the Analysis

Metric	Nominal	ANOVA
Metric Vector	Nominal	MANOVA
Nominal	Metric	Classification or Discrimination
Graph	Metric	Clustering
Metric Vector	Metric Vector	Partial Least Squares, Canonical Correlation

Finally, there are cases when we don't know the values of the dependent variable but would like to organize the observations in the independent variables (feature space) into nominal groups or model some patterns within this space. Below, we formally describe these problems as "supervised" and "unsupervised" learning. The canonical learning problem is classification problem where the feature space is metric variables and the dependent variables have two states—"class I" and "class II". For example, "class I" might represent GPCR proteins and "class II" might represent all others. For rest of the topic we will discuss classification problems, but it is always good to have the "big regression problem" as a unifying framework.

Unit 15: Structure of the learning problem



We now state more formally the learning problem. We assume that we have a blackbox that produces observed data—that is, objects with an associated feature vector that we will denote $\boldsymbol{x}$. (Note, in the sections below, I will generally denote vectors by BOLD font or accents like $\vec{x}$, equivalently.) This black box (= Nature) that we call "Generator" produces the observed data with an unknown probability distribution that we will denote as $G(\boldsymbol{x})$. Next, we have a hypothetical entity called the "Supervisor" that produces information that we will denote by y, conditional on receiving the input $\boldsymbol{x}$. The (random) variable y represents the property of interest like "does or does not catalyze phosphorylation", "is a brain cell", and so on. That is, y represents the knowledge we would like to infer from the observed data. The variable y may be a vector of numbers, but to simplify our discussion we will assume it is a single number. We will say that the supervisor is represented by the conditional distribution $S(y|\boldsymbol{x})$. We will use this notation to also represent the deterministic cases where the supervisor always produces the same y value for the same $\mathbf{x}$ value; i.e., $y = S(\boldsymbol{x})$. This probabilistic scheme allows us to generalize to the case where the supervisor might produce y values with some error. The supervisor may or may not exist—if it does not exist we call the problem "unsupervised learning" problem. If it exists, it may operate for a finite number of observations—maybe because the experiment to produce the y information is too hard.

The last element is the "Learning Machine". The "machine" part of the learning machine is a metaphor and typically what we are talking about is a mathematical construct that operates over the feature space. The "learning" part of the learning machine is also a metaphor as we will see below. Unfortunately, this terminology has stuck in the literature and we will follow the convention. The learning machine is a device that sees random samples of $\boldsymbol{x}$ and y, that is random samples from the joint distribution $P(\boldsymbol{x},y) = S(y|\boldsymbol{x})G(\boldsymbol{x})$. Given samples from

this joint distribution the learning machine tries to either estimate the conditional distribution of the supervisor, $S(y|\boldsymbol{x})$, or emulate its behavior. Estimating the exact form of $S(y|\boldsymbol{x})$, *i.e.,* recapitulating the supervisor, from finite samples is a poorly determined problem and cannot be solved except under very restrictive cases. Therefore, we will only consider the emulations. To emulate the supervisor's behavior, the learning machine has a set of tunable parameters. Typically, a given learning machine has a limited repertoire defined by its parameters. As a silly example, suppose a supervisor produces two types of y values +1 and −1 corresponding to whether an input protein is or is not a kinase. Let's also assume that the feature space is a vector $\boldsymbol{x}$ of k different features based on the amino-acids comprising the protein. Now suppose the learning machine only looks at the first element of the vector $\boldsymbol{x}$ and $x_1 > \alpha$ is assigned $y = +1$ and $x_1 \leq \alpha$ is assigned $y = -1$. Thus, the behavior of the learning machine depends on the value of α. Then given some samples, what we call training data, the learning machine will try to learn the best value of α to emulate $S(y|\boldsymbol{x})$. The repertoire of this learning machine is all the different ways it splits x_1 as a function of α. A machine that only looks at the first coordinate is not that useful and a real algorithm will have many parameters to tweak (learn) to emulate $S(y|\boldsymbol{x})$. Thus, to solve a problem we need to first choose a learning machine, each with its set of repertoire and then let it optimize on training data over the parameters of the learning machine. There may be many different parameters for a complicated learning machine but in the following we will generally represent the set of such parameters by a symbolic notation like α.

Example: Fisher's Discriminant function

Fisher's discriminant function is a classic example of a mathematical construct that was proposed to assign objects to several possible groups using measurements taken for each observation. We can have many possible groups but for simplification we will only discuss the case where we have only two possible groups. The procedure operates on a set of training dataset, finds an optimal discriminating function over the training dataset by considering some parametric probability models and then the discriminant function can be used for novel observations. To reiterate, the idea of Fisher's discriminant function is to obtain a mathematical function of the measured p-dimensional features, $x_1 \cdots x_p$, such that we can assign an unclassified observation by computing the values of the function. We would like a function $d(x)$ such that if $d(x) \geq c$, then we will call the object class I and if $d(x) < c$ then we will call the object class II. We can derive such a function from two (ultimately) equivalent viewpoints. The first is from likelihood point of view (Fisher's favorite view).

Let's suppose that all observations are either from class I or class II and the measured variables on each of them are samples from a multivariate normal distribution with a common variance-covariance structure. So, the distribution function for observations from each class is:

$$f_i(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^p \det(\Sigma)}} exp\left[\frac{-1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)'\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right] \quad i = 1,2 \qquad (1)$$

Note that $f_i(x)$is the likelihood function for the object with measurements $\boldsymbol{x}$ if it is from the ith class. Assuming that we know everything about the parameters μ_i, Σ of (1), it seems reasonable to write a ratio of the likelihoods and decide to put the observation in class I if this ratio is greater than some critical value:

$$\frac{f_1(x)}{f_2(x)} \geq c$$

Taking logs to make life easier and with a bit of algebra, we have the log likelihood ratio:

$$(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)'\Sigma^{-1}\boldsymbol{x} + \frac{1}{2}(\boldsymbol{\mu}_1+\boldsymbol{\mu}_2)\Sigma^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2) \geq \log c \qquad (1')$$

Only the first term of (1') depends on the data, x, so we can separate the two parts into what we will call the discriminant:

$$d(\boldsymbol{x}) = (\boldsymbol{\mu_1} - \boldsymbol{\mu_2})'\Sigma^{-1}\boldsymbol{x} \qquad (2)$$

what we will call the critical value

$$crit = \log c - \frac{1}{2}(\mu_1 + \mu_2)'\Sigma^{-1}(\mu_1 - \mu_2) \qquad (3)$$

If we examine $d(\boldsymbol{x}) = (\boldsymbol{\mu_1} - \boldsymbol{\mu_2})'\Sigma^{-1}\boldsymbol{x}$, although the matrix equations look complicated, it will turn out to be just some set of weights for each coordinate of $\boldsymbol{x}$ and thus the discriminant function will look like: $\sum_{i=1}^{p} w_i x_i$ for a *p*-dimensional feature space; that is, we will have a linear discriminant function. What remains is to determine μ_i, Σ from either prior knowledge or training data and also to decide on a good value of *c*, which is part of (3). An unbiased estimator of μ_i is $\bar{x}_i$, i.e., the sample mean of the training data set for each group class. An unbiased estimator of Σ is

$$\frac{1}{(n-2)}\sum_{i=1}^{2}\sum_{j=1}^{q_i}(x_{ij} - \bar{x}_i)(x_{ij} - \bar{x}_i)' \quad n = q_1 + q_2$$

the so-called pooled sample variance-covariance. The value for *c* is derived by considering relative error rates of misclassification.

Suppose if $d(x_0) \geq crit$, we assign this observation to first class (with μ_1) and assign it to the second class otherwise. Now, if x_0 is from class I, the probability of misclassification is

$$\alpha_1 = P_{\mu_1}((\mu_1 - \mu_2)'\Sigma^{-1}x_0 < crit)$$
$$= \Phi\left(\frac{crit - (\mu_1 - \mu_2)'\Sigma^{-1}\mu_1}{d}\right)$$
$$where \quad d = \sqrt{(\mu_1 - \mu_2)'\Sigma^{-1}(\mu_1 - \mu_2)}$$

and Φ is the unit cumulative normal distribution. The complex formula inside the Φ is simply normalizing the random variable $(\mu_1 - \mu_2)'\Sigma^{-1}x_0$ for mean (μ_1 multiplied by $(\mu_1 - \mu_2)'\Sigma^{-1}$) and standard deviation (*d*).

Similarly, if x_0 is from class II, the probability of misclassification is

$$\alpha_2 = P_2((\mu_1 - \mu_2)'\Sigma^{-1}x_0 > crit)$$
$$= \Phi\left(\frac{(\mu_1 - \mu_2)'\Sigma^{-1}\mu_2 - crit}{d}\right)$$

By assuming that the error rate for misclassifying observations from class I and class II should be equal and assuming a prior probability of an observation being from class I and class II is also equal, we can find the critical value (with some additional algebra):

$$crit = \frac{1}{2}(\mu_1 - \mu_2)'\Sigma^{-1}(\mu_1 + \mu_2) \qquad (4)$$

Note that while (4) looks a lot like (3), they are not the same. Also, note that while (4) looks complicated, if we know μ_i, Σ, it is simply a number.

As an example, let's consider the simple univariate case where x is one dimensional. Suppose, that our training data looks like this:

Class I: (-1.2, -1, -0.8, -1.1, -0.9, -1.08, -0.92)
Class II: (1.2, 1, 0.8, 1.1, 0.9, 1.08, 0.92)

Then sample mean of class I is −1 and that of class II is +1. Pooled sample variance is:

$$\frac{1}{14-2}\left[\left\{(-1.2-(-1))^2 + (-1-(-1))^2 + ... + (-0.92-(-1))^2\right\} + \left\{(1.2-1)^2 + (1.1-1)^2 + ... + (0.92-1)^2\right\}\right]$$
$$= 0.0188$$

Plugging these numbers into (4) we get

$$crit = \frac{1}{2}(-1-1)\frac{1}{0.0188}(-1+1) = 0$$

and from (2) the discriminant function is

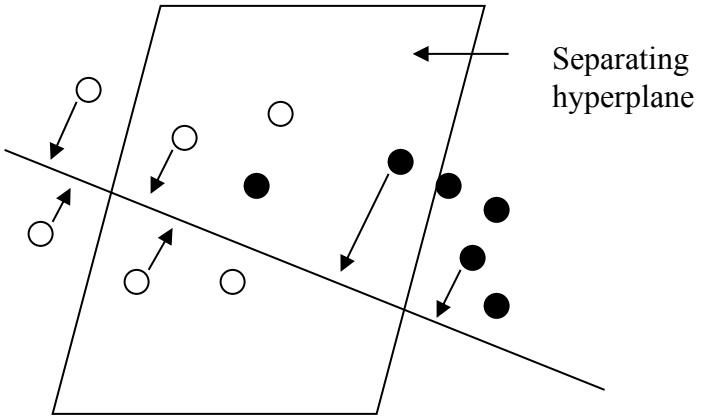
$$f(x) = (-1-1)\frac{1}{0.0188}x = -106.38x \qquad (*)$$

So, for novel data we will be plugging in the x variable into (*) and if it is greater than 0, we assign it to class I and if it is less or equal to zero we assign it to class II.

Another way of thinking about Fisher's discriminant function is to remember that the observations are points in p dimensional space. We wish to find a line where if we project the observations on the line and look at the result, we have maximized the following quantity

$$\frac{Between\ Class\ Sum\ of\ Square}{Within\ Class\ Sum\ of\ Square}$$

where the sum of squares is computed on the scalar values obtained from the projections. We can then select a point in between the two means, $\bar{x}_1$ and $\bar{x}_2$, say the midpoint, and draw an orthogonal plane that separates the observations into the two classes.



An important point to note is that a separating hyperplane in p-dimensional space can be defined by a p-dimensional vector, say $\vec{w}$. Then the p-1 dimensional separating hyperplane is defined as all points perpendicular to $\vec{w}$, or, $H = \{\vec{x} \subset R^p \mid \vec{x} \cdot \vec{w} = 0\}$. Thus, the task of finding the separating hyperplane will be the same as finding that vector $\vec{w}$. We now turn to one of the first "learning" algorithms.

Unit 16: Perceptron (F. Rosenblatt) and separating hyperplanes

The perceptron algorithm was first inspired as a model of how a biological neuron might function. The main idea was that a neuron gets electrical input from various other neurons and decides to fire (carry current) or not. The firing state is modeled as +1/-1 and the input signals are modeled as some linear function of input quantities x_1-x_p for p neurons feeding into the focal neuron. That is, the neuron acts like $sign(\sum_{i=1}^{p} w_i x_i)$ where x_i are the electric states of the input neurons and w_i are the weights that the focal neuron must "learn". Then a sequential algorithm was proposed to find the right weights for the linear function such that the neuron fires or not for the right kind of response. The most significant part of this is the sequential algorithm. Because the algorithm continually adjusts itself to new training examples, it had the flavor of "learning" as opposed to standard optimization like the Fisher's discriminant function. Of course, we can also convert Fisher's discriminant function to a sequential algorithm in that we can have it "learn" the training sample means and covariance sequentially. Thus, "learning" is a kind of a metaphor for sequential optimization. But regardless of the metaphoric license, this algorithm generated a great deal of new approaches in computer science, statistics, and cognitive psychology.

Instead of the learning metaphor we will go back to the geometrical interpretation. The signed function $sign(\sum_{i=1}^{p} w_i x_i)$ involves taking the dot product of a p-dimensional measurement $\boldsymbol{x}$ with a weight vector $\boldsymbol{w}$, and finding the sign of the resulting number. Note that $\vec{w} \cdot \vec{x}$ is positive when the two vectors are at an angle less than 90 degree from each other and it is negative when the vectors are greater than 90 degrees. It is zero when they are exactly perpendicular to each other. The weight vector $\boldsymbol{w}$ defines a $(p-1)$ dimensional hyperplane in p-dimensional space separating the input vectors $\boldsymbol{x}$ into all those on one side of the hyperplane versus the other. Thus, the Perceptron criterion is also an algorithm for finding a separating hyperplane. More concisely then,

Perceptron problem: Given some training data set separable by a hyperplane with *p*-dimensional measurements $\vec{x}_i = (x_i^1, \cdots x_i^p)$for the ith observation and the known supervisor function $S(\vec{x}_i) = y_i$ where y_i is +1 or -1, find an indicator function $f(\vec{x}, \vec{w}) = sign\{\sum_i^n w_i x_i\}$ such that $f(\vec{x}_i, \vec{w}) = y_i$ *for all i*.

If the training data are separable then the following algorithm is guaranteed to find the right vector ***w***.

Perceptron Algorithm

1. Set $\vec{w}(0) = (0,\ldots,0)$
2. Let

$$\vec{w}(t) = \begin{cases} \vec{w}(t-1) & if\ y_t(\vec{w}(t-1)\vec{x}_t) > 0, \\ \vec{w}(t-1) + y_t\vec{x}_t & if\ y_t(\vec{w}(t-1)\vec{x}_t) \le 0 \end{cases}$$

Note very carefully what the above algorithm states. Start with the zero vector for $\vec{w}(0)$ and then do a dot product with the first training observation $\vec{x}_1$, then multiply the result by the +1/-1 label value of y_1. If this quantity is less or equal to zero, then update the $\vec{w}$ vector with the addition of the signed vector $\vec{x}_1$…and continue this process iteratively until the end of the training data.

Novikoff's theorem: Suppose there exists some w_0 such that

$\min_{(y,x)} \frac{y(w_0 x)}{|w_0|} \ge \rho$ (separating plane with at least ρ margin) and $|x| < D$. Then the Perceptron algorithm correctly constructs a separating hyperplane after at most

$M = \left[\frac{D^2}{\rho^2}\right]$ corrections.

□

Proof.

The idea is to bound w(t) in terms of k from above and below and derive an inequality for k. We now index w by k, that is by the number of corrections to w. Then if at t-th observation, we make the kth correction we have,

$|w(k)|^2 = |w(k-1)|^2 + 2y_t(x_t \cdot w(k-1)) + |x_t|^2$. Given $|x| < D$,

$|w(k)|^2 \le |w(k-1)|^2 + D^2$

Then since w(0) = 0, we have $|w(k)|^2 \le kD^2$. This is the first inequality.

Now we consider the normalized vector $w_0/|w_0|$ and its inner product with w(k).

$$\frac{w(k)\cdot w_0}{|w_0|} = \frac{w(k-1)\cdot w_0}{|w_0|} + \frac{y_t x_t \cdot w_0}{|w_0|} \geq \frac{w(k-1)\cdot w_0}{|w_0|} + \rho$$

Therefore, we have the lower bound, $\frac{w(k)\cdot w_0}{|w_0|} \geq k\rho$ and then using Cauchy-Schwartz inequality, $x \cdot y \leq |x||y|$, we get $|w(k)| \geq k\rho$.

Combining the two bounds we obtain

$$k \leq \frac{D^2}{\rho^2}$$

□

Another theorem about the Perceptron. Suppose we have an infinite number of training sequence and we decide to stop the training after some number of corrections. In particular, it would be reasonable to say that we will stop if next m trials are correctly identified. We now want to generate a stopping rule such that we are assured of certain performance. So, suppose we've made k corrections and we want to stop training after next m(k) correct assignments. Then we have

With probability 1 - η the stopping rule has a risk of at most ε if we stop at

$$m(k) = \left\lfloor \frac{2\ln k - \ln\eta + \ln\frac{\pi}{6}}{-\ln(1-\varepsilon)} \right\rfloor + 1$$

□

For a proof see Vapnik, "Statistical Learning Theory", pg 379.

The above algorithm assumed that the two classes of points were separable by a hyperplane passing through the origin. Of course, in real data this may not be the case. A hyperplane that is off the origin can be defined by the equation $H_b = \{\vec{x} \subset R^p \mid \vec{x}\cdot\vec{w} = b\}$. This motivates the affine (linear subspaces displaced from the origin) algorithm for the perceptron. Unfortunately, this time we require multiple passes through the training dataset, adjusting both the vector and the unknown displacement quantity *b* at the same time.

Perceptron affine algorithm

1. Set w0 = 0, b0 = 0
2. $D = \max(|x|)$
3. Repeat

   For i = 1 to q

   If $y_i(w_k x_i + b_k) \leq 0$ then

   $w_{k+1} = w_k + \eta y_i x_i$

   $b_{k+1} = b_k + \eta y_i D^2$

   $k = k + 1$

endif
end for
until no mistakes are made

What if there isn't a separating plane?

One might want to minimize the number of training errors. But, this problem is known to be NP-complete. So, we try to find a local optimum to the so-called empirical risk function

$$R(w) = \frac{1}{q}\sum_{i}^{q}(y_i - sign(x_i \cdot w))^2$$

This might be done by some gradient algorithm

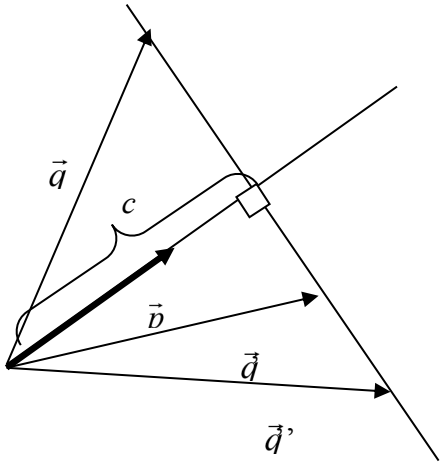
Unit 17: Support Vector Machines

In thinking about the separating hyperplane for the Perceptron algorithm, it becomes obvious that in general we will have lots of potential hyperplanes that will separate the data just as well. This means that we need another criterion beside just separation to uniquely determine a "good" separating hyperplane. In the case of Fisher's discriminant function, we found such unique hyperplane by assuming that points are distributed as a multivariate normal distribution and then minimizing the within versus between distance of the points—or equivalently equalizing the conditional misclassification error. Here we introduce a different idea of finding the separating hyperplane that has the largest "margin" between the points in class I versus class II along the border formed by the separating hyperplane. A margin can be imagined by assuming that the separating hyperplane has depth so that we have a "fat" hyperplane (see picture below). Then, we would like to find the fattest hyperplane that still separates the training data. The main idea is that such a hyperplane not only separates the training dataset well, but it is also likely to work well for unseen novel data because it creates a large "moat" between points that belong to one class and points that belong to another class. We call such a hyperplane, a large-margin hyperplane, and this idea will lead to learning algorithms called Support Vector Machines (SVM).

3.1 Large-margin optimal hyperplane

Suppose we have a finite set of training data $(y_1, \vec{x}_1),...,(y_q, \vec{x}_q)$ where y = -1 or 1 for each class I and II the x's are p-dimensional vector of features. We say this training set is separable by some unit vector ϕ and the associated (affine) hyperplane $H = \{h \mid h \cdot \phi = c\}$ if

$$\begin{aligned} x_i \cdot \phi > c \quad if\ x_i \in I \\ x_i \cdot \phi < c \quad if\ x_i \in II \end{aligned}. \tag{3.1}$$



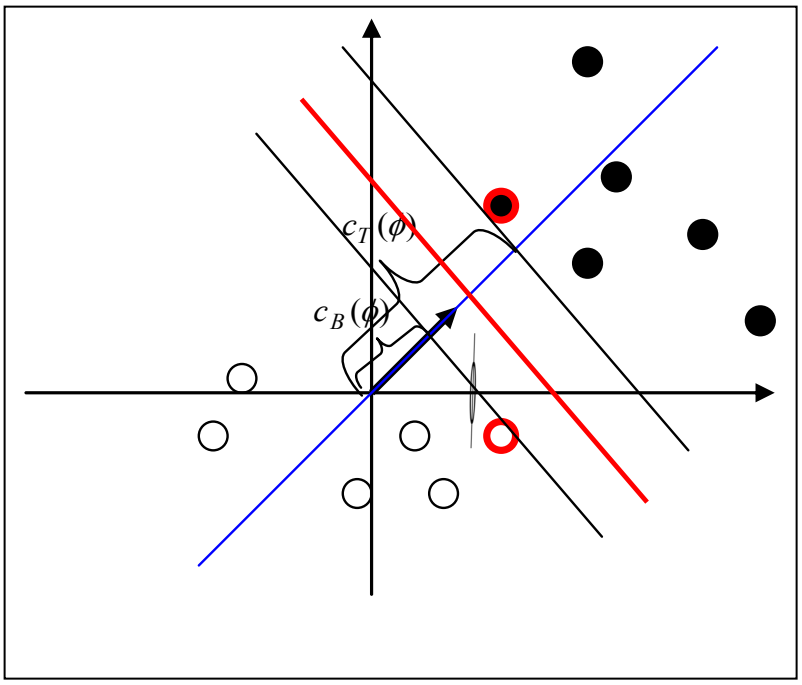
To interprete (3.1) we recall the fact that if we have a fixed vector $\vec{p}$ and we consider all vectors $\vec{q}$ that satisfy the equation $\vec{p} \cdot \vec{q} = c$, we get a collection of vectors whose end points form a line at an offset c distance from the origin. Each of the end points along the line projects to the vector direction $\vec{p}$ with length c as shown in the picture on the left. Thus

the vector ϕ and the constant c, defines a hyperplane that is perpendicular to ϕ and length c from the origin. That is called an ***affine linear subspace***.

Now define two extreme values

$$c_T(\phi) = \min_{\vec{x} \in I}(\vec{x} \cdot \phi)$$
$$c_B(\phi) = \max_{\vec{x} \in II}(\vec{x} \cdot \phi)$$

That is, considering all of the input points $\vec{x}$ in class I and class II respectively, $c_T(\phi)$ is the smallest value on one side of the separating hyperplane and $c_B(\phi)$ is the largest value on the other side of the separating hyperplane. The geometry is shown in the figure below where the filled circles are objects from class I and the empty circles are objects from class II:



The observation marked by the red outline and filled circle defines the larger value $c_T(\phi)$ (the least value among the filled circles) and the observation marked by the red outline and empty circle defines the smaller value $c_B(\phi)$ (the maximal value among the empty circles). These values and the corresponding observation points define the largest "margin" of separation between the two classes along the vector direction ϕ. The separating hyperplane is half way between the lines defining the margin (shown in red above) and the margin is a kind of "margin of error" for classification. The idea is that larger this margin of error for the training data, the more likely that the separating hyperplane will also correctly classify novel data. We quantify the separation margin as:

$$\rho(\phi) = \frac{c_T(\phi) - c_B(\phi)}{2}.$$

A vector ϕ_0 and associated hyperplane that maximizes the margin and satisfies (3.1), i.e., separates the classes, is called the maximal margin hyperplane or the optimal large-margin hyperplane. Furthermore, when we examine the figure above, we see that only a few of the observation points are relevant for establishing the separating hyperplane and the margin, namely the red outlined points in the diagram. The other points do not affect the position of the margin. Those observation vectors that determine the separating hyperplane and the margin are called Support Vectors. Thus, this class of learning method is called the Support Vector Machine (SVM).

Computing the optimal large-margin hyperplane

We now consider an equivalent form for the optimal hyper plane: Find a vector ψ_0 and a number b_0 such that they satisfy

$$x_i \cdot \psi_0 + b_0 \geq 1 \quad if \ y_i = 1$$
$$x_i \cdot \psi_0 + b_0 \leq 1 \quad if \ y_i = -1$$
and $|\psi|^2 = \psi \cdot \psi$ is minimum.

Then the vector ψ_0 is related the optimal hyperplane vector ϕ_0 by $\phi_0 = \frac{\psi_0}{|\psi_0|}$ and

$$\rho(\phi_0) = \max_{\phi} \frac{1}{2}(\min_{x \in I}(x \cdot \phi_0) - \max_{x \in II} x(x \cdot \phi_0)) = \frac{1}{|\psi_0|}.$$

In other words, we can find the optimal hyperplane by finding the vector ψ_0 such that $|\psi|^2 = \psi \cdot \psi$ is minimized and satisfies the constraints $y_i(x_i \cdot \psi_0 + b) \geq 1$ for all *i*.

Here we take a break for optimization

3.2 Lagrange multipliers

Suppose we want to minimize some function $f_0(x)$ under a set of constraints $f_i(x) = 0$, for $i = 1 \ldots m$. Then assuming $f_i(x)$ is differentiable and all partial derivatives exist we have the Lagrange function

$L(x, \lambda, \lambda_0) = \sum_{i=0}^{m} \lambda_i f_i(x)$ such that at a local extremum we have

$$\frac{\partial L}{\partial x_i}(x^*, \lambda^*, \lambda^*_0) = 0$$

for some set of numbers λ_i^* not all equal to zero.

3.3 Kuhn-Tucker theorem for convex optimizations,

Suppose we want to minimize some function $f_0(x)$ under a set of inequalities $f_i(x) \leq 0$ where $f_i(x)$ are convex functions and we are only considering some convex domain $x \in A$. Then we again consider the Lagrange function

$L(x, \lambda, \lambda_0) = \sum_{i=0}^{m} \lambda_i f_i(x)$ then the following three conditions hold true at the minimum x*:

(1) $\min_{x \in A} L(x, \lambda^*, \lambda_0^*) = L(x^*, \lambda^*, \lambda_0^*)$

(2) $\lambda_i^* \geq 0 \quad i = 0,1,\ldots m$

(3) $\lambda_i^* f_i(x^*) = 0, \quad i = 1,\ldots, m$

If certain interior point conditions are satisfied, we can set $\lambda_0 = 1$ then the Kuhn Tucker theorem conditions are equivalent to the existence of a saddle point (x*, λ_i^*) of the Lagrange function. That is,

$$\min_{x \in A} L(x, \lambda^*, 1) = L(x^*, \lambda^*, 1) = \max_{\lambda > 0} L(x^*, \lambda, 1)$$

Now back to solving the optimal margin problem

3.4 Finding optimal margin vector

We go back to the problem of finding the vector ψ_0 such that $|\psi|^2 = \psi \cdot \psi$ is minimized and satisfies the constraints $y_i(x_i \cdot \psi_0 + b) \geq 1$ for all *i*. We now write the Lagrange function

$$L(\psi, b \mid \alpha) = \frac{1}{2}\psi \cdot \psi - \sum_{i=1}^{q} \alpha_i (y_i((x_i \cdot \psi) + b) - 1) \qquad (3.4.1)$$

According the Kuhn-Tucker theorem, we want the minimize L over ψ and b and maximize it over the nonnegative α_i, Lagrange multipliers. To find the minimum we differentiate and obtain

$$\frac{\partial L}{\partial \psi} = \psi - \sum_{i=1}^{q} y_i \alpha_i x_i = 0$$
$$\frac{\partial L}{\partial b} = \sum_{i=1}^{q} y_i \alpha_i = 0 \qquad (3.4.2)$$

Substituting the equalities in (3.4.2) into (3.4.1) we have

$$w(\alpha) = \sum_{i=1}^{q} \alpha_i - \frac{1}{2} \sum_{j=1}^{q} \sum_{i=1}^{q} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) \qquad (3.4.3)$$

We now need to maximize (3.4.3) in the nonnegative domain for α_i's which is another quadratic optimization problem. But, we can note several important features of (3.4.3). First, it is now defined only in terms of the inner products of x's rather than their coordinates. Therefore, the dimensionality of x's do not come into play. Second, suppose the optimal values of α_i's were given by α_i^0's. Then we have $\psi_0 = \sum_{i=1}^{q} y_i \alpha_i^0 x_i$ and from Kuhn-Tucker conditions

$$\alpha_i^0 (y_i (x_i \cdot \psi_0 + b_0) - 1) = 0 \quad i = 1, \ldots, q$$

which means α_i^0 is non-zero only for those x's such that $x_i \cdot \psi_0 + b_0 = 1$. As noted above, such vectors are called *support vectors*.

A note on computing the optimal hyperplane.

If we look at functionals like (3.4.3), they are quadratic functions of the α_i's and there are as many α's as there are input training observations. This means that if we have very large data sets, say gene expression data, we may end up with an optimizing problem with thousands of variables. One way to solve this numerical problem is to notice that most of the α's are zero at the optimum. Only the support vectors have none-zero α's. If we solve the problem for a small number of observations and find some of the α's to be zero; they will still be zero for the full problem since they cannot be support vectors. This suggests the incremental algorithm on the right

Incremental SVM optimization algorithm:

Let k be the incremental problem size.

```
Repeat
    For i=1 to q by k
        Optimize the (3.4.3)
functional (or equivalent functionals)
for observations, xi  to  xk+i.
        If αi = 0, discard xi
    End for
Until no more x's are discarded
```

Δ-margin separating hyperplane

A hyperplane, $(\psi^* \cdot x) - b = 0, \quad |\psi^*| = 1$ is called the Δ-margin separating hyperplane if it classifies vectors as

$$y = \begin{cases} 1 & if\ (\psi^* \cdot x) - b \geq \Delta \\ -1 & if\ (\psi^* \cdot x) - b \leq \Delta \end{cases}$$

Now we want to consider classifying non-separable cases with the Δ-margin separating hyperplane. For a problem with non-separable cases, we cannot optimize for the margin size and we cannot solve for smallest error hyperplane without running into serious computational problems (NP-hard). So, we first decide on a desirable margin and then minimize some cost function which measures how different each observation is from respecting that margin. We consider the linear cost function

$$F(\xi) = \sum \xi_i \qquad (3.5.1)$$

and try to minimize (3.5.1) subject to the separating hyperplane constraints

$$y_i(\psi \cdot x_i - b) \geq 1 - \xi_i \text{ and } \psi \cdot \psi \leq \frac{1}{\Delta^2}. \qquad (3.5.2)$$

The terms of ξ are called slack variables and basically measures how much we have to adjust the points away from the hyperplane to make things separable.

After the usual Lagrange-Kuhn-Tucker conditions described in the above box we arrive at the problem of maximizing:

$$W(\alpha, \gamma) = \sum \alpha_i - \frac{1}{2\gamma} \sum\sum \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) - \frac{\gamma A^2}{2} \qquad (3.5.3)$$

with

$$\sum y_i \alpha_i = 0, \quad 0 \le \alpha_i \le 1, \quad \gamma \ge 0$$

There are both α and γ to optimize so one must solve an iterative quadratic optimization problem.

Soft margin separating hyperplane

A different approach which has better solutions to the optimization problem is to construct a penalized cost function on the norm of the separating vector, so we consider

$$\Phi(\psi, \xi) = \frac{1}{2}\psi \cdot \psi + C\sum_{i=1}^{q} \zeta_i$$

Under this objective function, after the LKT (Lagrange-Kuhn-Tucker) computations we obtain the same quadratic form as the separable case except for a slightly different set of constraints. So, we have maximize,

$$w(\alpha) = \sum_{i=1}^{q} \alpha_i - \frac{1}{2}\sum_{j=1}^{q}\sum_{i=1}^{q} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j)$$

with constraints

$$0 \le \alpha_i \le C$$
$$\sum_{i=1}^{q} \alpha_i y_i = 0 \qquad (3.6.1)$$

Compare (3.6.1) with the case we discussed for (3.4.3).

Minimum SV separating hyperplane

One other approach we might consider is to minimize the number of support vectors. There are statistical properties of SVM (to be discussed) that suggests this might be a reasonable thing to do. In this case one might consider minimizing the functional

$R = \sum \alpha_i^p$ for some value p. If we set p = 1, then we minimize a linear quantity making things a bit simpler.

Now for the nonseparable case, we put together this functional with the functional for the slack variables and try to minimize

$$L = \sum \alpha_i + C\sum \xi_i$$

subject to the constraints in (3.5.2)

Unit 18: Polynomial coordinate transforms and kernel distances

Up to now we discussed SVM in the basic setting where we have input observations with associated measurement vectors in p-dimensional space and we try to find a separating hyperplane. If this was all there was,

SVMs would not be very useful since many real examples are not likely to have separable classes, even with soft margins. The main power of SVM comes when we transform the observation points to an another coordinate system or another space such that what was inseparable becomes separable in the new representation. The idea of transforming the points to carry out better data analysis is not new. For example, gene expression data is often analyzed with a log-transformation, which has the effect of stretching points near zero and compressing points far away from zero. However, SVMs are especially good for exploring such transformations because much of the computations involved in using the transformations are simple. The basic observation is that the optimal separating hyperplane is computed from inner products of observations—therefore, one can directly manipulate the inner products using something called a kernel function. We will examine kernel functions in more detail but here we will give a simple example and then discuss coordinate transformations and general transformations in detail for a more advanced treatment.

Suppose we have a two dimensional input space (x,y). Now consider the following set of coordinate maps

$$\begin{aligned} z_1 &\leftarrow x \\ z_2 &\leftarrow y \\ z_3 &\leftarrow x^2 \\ z_4 &\leftarrow y^2 \\ z_5 &\leftarrow xy \end{aligned} \qquad (3.8.1)$$

The above means create a new 5-dimensional space with the coordinates z_1 to z_5 by the above (3.8.1) functions of the original two dimensional space coordinates.

Consider a separating hyper plane $z \cdot \phi$ = b, in Z-space and see what it means in (x,y) space. We obtain

$$\vec{z} \cdot \vec{\phi} = b \rightarrow \sum z_i \phi_i = b$$
$$\rightarrow \phi_1 x + \phi_2 y + \phi_3 x^2 + \phi_4 y^2 + \phi_5 xy = b$$

In other words, the linear separating plane in Z-space is a quadratic separating curve in (x,y) space with the coefficients $\phi_1,...,\phi_5$. In general, if we start with an *n*-dimensional space, we can send it to a $\frac{n(n+3)}{2}$ dimensional space, find a separating plane that will be equivalent to a quadratic separating curve in the original space. If we wanted to have p-degree polynomial separating curve, we need to send to $O(n^p)$ dimensional space. Therefore, it is easy to see that this kind of coordinate-wise computation would be impractical when the degree of polynomial is large.

The key solution to this is to use an idea from functional analysis. First, as we mentioned, all of the optimization functions such as (3.4.3) and (3.5.3) use data from the input space only through the inner product of the vectors. Given some set of coordinate functions, or what is called the orthonormal basis vectors, the standard inner product that we are familiar with is

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i y_i \,. \qquad (3.8.2)$$

Sometimes coordinates where the inner products are given in this form are called Euclidean co-ordinates. (In the more advanced section below, we discuss somewhat more precisely the relationship between vectors and

coordinates.) Suppose we now have a new space constructed by a set of coordinates obtained from the old Euclidean co-ordinates:

$$\begin{aligned} z_1 &= z_1(x_1,...,x_n) \\ &\vdots \\ z_m &= z_m(x_1,...,x_n) \end{aligned} \qquad (3.8.3)$$

where each $z_i(x_1,...,x_n)$ is a differentiable function of x's. According to theorems from functional analysis the inner product in the new space has a representation:

$$\vec{z}^1 \cdot \vec{z}^2 = \sum a_r z_r^1(x_1) \cdot z_r^2(x_2) = K(\vec{x}^1, \vec{x}^2) \qquad (3.8.4)$$

where the last function *K* is a symmetric and so-called positive definite function that we will call a kernel function. First, in (3.8.4) the left most quantity denotes the inner product of two vectors in z-space, and I have used the superscripts to index the two vectors. The second quantity states that the inner product of these two vectors can be written as the sum of the product of the coordinates of each of the vectors, $\vec{z}^1$ and $\vec{z}^2$, multiplied by some coefficient a_r. Here, I am using the subscripts to index the coordinates of the vectors. Each of the coordinates in z-space are functions of the coordinates $(x_1...x_n)$ as given in (3.8.3). Thus, the second quantity is actually a function of the corresponding x vectors, $\vec{x}^1$ and $\vec{x}^2$. In fact, it is a very particular function called a kernel function as we just mentioned.

The implication of (3.8.3) and (3.8.4) is critical to understanding SVM and kernel methods. As mentioned before, we start out with observations as vectors in a p-dimensional space; let's call this the x-space. We find that our training data is difficult to classify in x-space so we consider mapping to another space such as the polynomial transformations discussed in (3.8.1); let's call this the z-space. We can do this explicitly as in (3.8.1) and then compute a separating hyperplane in z-space and maybe find a good performing classifier that way. But, when we think carefully about what it means to take inner products in z-space (to compute the separating hyperplane), it involves taking two vectors and producing a number. In particular, if the vectors are represented by a set of numbers that are their coordinates we get inner products by computing sums of products of the form shown in (3.8.4), which in turn are functions of the coordinates in x-space. Thus, we can actually compute inner products in z-space by computing particular functions in x-space. All we have to insure is that the particular function in x-space has some properties that make it equivalent to computing inner products in z-space. It turns out that the particular properties are "symmetric" and "positive definite" (discussed in more detail in the advanced material). What do we gain by this? The main thing we gain is computational efficiency. As noted before, transforms like (3.8.3) can be complicated. In fact, we may even want to pretend that we are transforming our x-space to an infinite dimensional z-space (say we want to consider polynomials up to infinite degree)—which is obviously not computable by the explicit construction. However, if we do the computations by a kernel function in x-space, then we can (generally) easily compute the z-space inner product.

Since notations can get messy, from now on we will use <x,y> to indicate inner products. An easy example of a kernel function is $< \vec{x}_1, \vec{x}_2 >^2$ (*). (Note, in denoting vectors, it is sometimes confusing how to index two different vectors versus different coordinates of a given vector. In the notation (*) I've now used subscripts to index two different vectors rather than two different coordinates. When the meaning is clear from the context, I will not particularly point out the indexing schemes.) Suppose that $\vec{x}_i \in (x, y)$, vectors in 2-dimensional space. Then working out the inner product, we have

$< \vec{x}_1, \vec{x}_2 >^2 = (x_1x_2 + y_1y_2)^2 = (x_1x_2)^2 + 2x_1x_2y_1y_2 + (y_1y_2)^2$. This inner product looks a lot like what we would have obtained from the Z-space above (try writing out the inner product in Z-space). So, we might guessed such inner products, and kernel functions, correspond approximately to polynomial separating hyperplanes. In general, a kernel function of the form $< x, y >^p$ is called (p-degree) polynomial kernel and roughly corresponds to constructing a p-degree classification function. Computation of such kernel functions are much easier than explicitly considering a high dimension coordinate system.

SVM generalizations: Kernels and inner products

We noted in the previous section that the advantage of using the LKTM form of optimization problem is that it makes it obvious that the separating hyperplane depends only on the inner product of the input vectors. This is also reasonable if we think of the geometry of the problem. The hyperplane depends on the geometric configuration of the points and not on the particular coordinate system. Thus, the solution should only depend on the geometric (coordinate invariant) quantities of the points, namely the inner product.

So here we take a slight detour to discuss inner products of vector spaces

Vector space and Euclidean geometry

The idea of a vector is initially an algebraic idea. That is, we define a vector space as a set that is closed under vector additions and by multiplications by a "scalar" quantity. Up to this point, geometric notions like angles and length are not defined. Notion of the length of a vector can be defined without necessarily defining angles. Such a notion of length is called the norm of a vector and a vector space with a definition of a norm is called a normed vector space. A norm is a real valued function from the vector space denoted $\|x\|$ such that

$\|x\| = 0 \rightarrow x = 0$, $\|xa\| = |a|\|x\|$, and $\|x + y\| \leq \|x\| + \|y\|$.

An inner product is a concept that can be used to simultaneously define angles and length. Recall that in introductory algebra the inner product is *defined* as,

For $x = (x_1, \cdots x_n), y = (y_1 \cdots y_n), \quad < x, y >= \sum_{i=1}^{n} x_i y_i$. (4.1.1)

And we are given the formula $\cos\alpha = \frac{< x, y >}{| x || y |}$ for angles and $| x |= \sqrt{x_1^2 + \cdots + x_n^2}$ (4.1.2) for length. The angle formula is often given by considering Pythagorean geometry.

The problem here is that both (4.1.1) and (4.1.2) are dependent on the presence of a coordinate system; that is, the sequence of numbers that represent the vectors. Furthermore, we have no guarantee that the quantities (4.1.1) and (4.1.2) do not depend on the particular coordinate system. Since length and angles are geometric concepts, we would not want them to depend on the coordinate system—or necessarily the presence of a predefined coordinate system; after all, coordinates are Descartes' invention and geometry existed before his time. So the approach we take is to define the inner product without reference to coordinates.

Let V be a vector space. Then the inner product <x,y> is a function from VxV to some scalar field (usually real or complex numbers; but for all of the following discussion we will assume real numbers), such that:

1. (non-degenerate) $< x, y >= 0 \quad for\ all\ y \Rightarrow x = 0$
2. (symmetric) $< x, y >=< y, x >$
3. (bi-linear) $< x + y, z >= (x, z > + < y, z > \quad and \quad < \alpha x, y >= \alpha < x, y >$ for scalar α.
4. (positive definite) $< x, x > \ > 0 \quad if\ x \neq 0$

A more primitive object without condition 4 is sometimes called a metric tensor. Notice that none of this definition involves a coordinate system--yet. We now use the inner product to define length-like objects. An inner product norm is given by $\|x\| = \sqrt{< x, x >}$ (notice we need the positive definite property for this to exist; that is, we don't want the quantity inside the square root to be negative). Now instead of appealing to Pythagorean geometry, we directly define angles as $\cos\alpha = \frac{< x, y >}{\|x\|\|y\|}$. Thus the abstract inner product simultaneous defines the concept of length and angles—so far without referring to any coordinate system. This now allows us to define the crucial notion of orthogonality by saying vectors x and y are orthogonal iif <x, y> = 0.

However, to do any concrete calculations we need to use coordinate systems. There are many ways to set this up which often involves rather confusing statements like "a tangent vector is a set of numbers that transforms…" (at least confusing to me). I am going to try to set this up a little bit differently. First, remember that a basis of a vector space is a set of vectors $\{x_1, \cdots, x_n, \cdots\}$ such that any vector $y = \sum_{i=1}^{\infty} \alpha_i x_i$. (Notice here that we are allowing for possible infinite sums. Assuring such sums can be taken, etc. requires us to impose some conditions on the kind of vector space we are talking about. The nice kind—the kind that acts just like a finite dimensional Euclidean space are called Hilbert spaces. A basis set is called orthonormal if $< x_i, x_j > = 0 \ for\ i \neq j$ and $\|x_i\| = 1$. This is more compactly stated as $< x_i, x_j >= \delta_{ij}$ where $\delta_{ij} = 0\ if\ i \neq j\ and\ \delta_{ij} = 1\ if\ i = j$ (Kroenecker delta function). We take as given a theorem that says that any vector space has an orthonormal basis set. So far no need for coordinates yet…

Now we state the

Fourier Theorem: if $\{x_\sigma\}$ is an orthonormal basis set then any vector can be represented as

$y =< y, x_1 > x_1 + < y, x_2 > x_2 + \cdots < y, x_n > x_n + \cdots$ (4.1.3; Fourier series expansion)
□

(For sticklers of formalities, the statement of $\{x_\sigma\}$ as an orthonormal basis set and the existence of the Fourier expansion are equivalent.)

(4.1.3) does not require any coordinates. In fact, it allows us to define coordinates where we now say a system of coordinates with respect to an orthonormal basis set is

$y = \{< y, x_1 >, < y, x_2 >, \cdots, < y, x_n >, \cdots\}.$ (4.1.4)

Such a system of coordinates is called Euclidean coordinates. If we have Euclidean coordinates for vectors $p = \{p_1,..., p_n,...\}$ and $q = \{q_1,..., q_n,...\}$ then the inner product can be computed as $\sum_{i=1}^{\infty} p_i q_i$. I leave it as an exercise to see that if we use 4.1.4 definition of Euclidean coordinates, $\sum_{i=1}^{\infty} < p, x_i >< q, x_i > =< p, q >$.

Having set up a definition of a coordinate system, we can now create different coordinate systems simply by constructing functions from old coordinates to new coordinates. Restricting ourselves back to finite dimensions, we can try to see how we should compute the inner product with respect to the new coordinates. Suppose the old Euclidean coordinates in *p*'s and the new coordinates in *z*'s are related by:

$$p_i = \sum a_i^j z_j$$

Then $< p^1, p^2 >= \sum p_i^1 p_i^2 = \sum (\sum a_i^j z_j^1)(\sum a_i^j z_j^2)$ Lot's of indices, but eventually

$< p^1, p^2 >= z'Gz$ where we've now written z in vector form and G is a matrix with $g_{jk} = \sum_i a_j^i a_k^i$. This G is called a quadratic form on the vector space and connect coordinate systems to "bilinear form inner products". If the coordinate system is Euclidean, then G = I, the identity matrix. Otherwise, it is some other symmetric positive definite matrix (a matrix is positive definite if for all vectors $x \neq 0, x'Gx > 0$). While the following is not necessary, to be complete we now describe how G changes with general coordinate transforms. Suppose we transform the coordinates through a general differentiable transform function,

$$p_i = p_i(z_1,...z_n)$$

Let C be the Jacobian matrix evaluated at some point P, then $G_z = C'G_pC$.

Finally, we consider one more way of computing the inner product, let $\{x_\sigma\}$ be an orthogonal basis set, not necessarily unit normal. That is, $< x_i, x_j >= 0 \; if \; i \neq j \; and < x_i, x_i >= \frac{1}{\lambda_i}$ then $< p^1, p^2 >= \sum \lambda_i p_i^1 p_i^2$ with respect to $\{x_\sigma\}$ basis.

Kernel functions

The whole point of above discussion was to figure out how to compute an inner product and also to compute an inner product without explicitly doing the coordinate transform map. In the finite dimensional case, we can take any symmetric positive definite matrix and associate it with an inner product in some coordinate system. So, we can compute

$$K(x, y) = x'Gy$$

where *x* and *y* are input vectors and G is some symmetric positive definite matrix. We want to generalize this to infinite dimensional case. The following is a version of the Mercer's theorem found in SVM textbooks:

Theorem: Let *K(x,y)* be a continuous symmetric, positive definite function (= $\iint_C K(u,v)g(u)g(v) \geq 0$ for all *g(u)* in $L^2(C)$, where C is a compact domain). If ϕ_n, λ_n are the eigenfunction and eigenvalues of the integral operator *K(x,y)* then

$$K(x,y) = \sum_i^\infty \lambda_i \phi_i(x)\phi_i(y) \qquad (4.2.1)$$

which converges absolutely in $L^2(C)$

□

We need several comments for this theorem. First $L^2(C)$ is an infinite dimensional Hilbert space. As a short hand, one might think of it as a space of bounded functions on a closed interval. The eigenfunctions are then orthogonal basis set of this Hilbert space. Thus (4.2.1) says if we have a symmetric positive definite kernel function *K(x,y),* it is like computing an inner product in the Hilbert space.

This theorem is often used to justify the idea of using various kernel functions for SVM and also to construct new kernel functions. In practice, this construction is a little bit too overly formal. For example, we do not necessarily need to use a positive definite function to produce some kind of classification function. On the other hand, sticking to this construction allows us to geometrically use hyperplanes and assure certain statistical properties to be discussed later.

Now we look at some kernels:

4.3.1 Polynomial kernel

$K(x,y) = (< x,y > + c)^p$ where *p* is some integer

4.3.2 Fourier

$$K(x,y) = K(x-y) = a_0 + \sum a_n \sin(nx)\sin(ny) + \sum a_n \cos(nx)\cos(ny)$$

4.3.3 Radial basis kernel

$$K(x,y) = K(x-y) = \exp(-\|x-y\|^p / d)$$

4.3.4 Two-Layer Neural network kernel

$$K(x,y) = \frac{1}{1+\exp(v < x,y > -c)}$$

4.3.5 Discrete kernel for strings

Here we need a few definitions. Let a string s be a concatenation of letters from some finite set L. Let l(s) be the length of a string s and t = σ(s) indicate that t is a substring of s. Now we can define a "coordinate system" for the strings as:

$\phi_u(s) = a_u \lambda^{l(u)}$, where *u* is a substring, a_u is the number of times u is found in the string *s*, and *l(u)* is the length of *u*. Then

$K_n(s,t) = \sum_{u \in \Omega} < \phi_u(s), \phi_n(t) >$ where Ω is the set of all strings of size *n*.

4.3.6 Kernels from kernels

Kernels can be easily generated from other kernels because the property of positive definiteness is closed under many operations. Thus

(i) Products of kernels are kernels.
(ii) Sums of kernels are kernels
(iii) Kernels multiplied by a scalar is a kernel

Example of the effects of a kernel transform

Since we usually have a better feel for distances rather than inner products, as an exercise, we can try to see what happens to points when we map them to a different space using a kernel function. We will work with the radial kernel function

$$K(x,y) = K(x-y) = \exp(-\|x-y\|^p / d).$$

Suppose we have two vectors x and y, then the inner product distances between them are defined as $\|x-y\| = \sqrt{< x-y, x-y >} = \sqrt{< x,x > -2 < x,y > + < y,y >}$. Then with the radial basis kernel we can define distances as:

$$d(x,y) = \sqrt{K(x,x) - 2K(x,y) + K(y,y)} = \sqrt{2 - 2\exp(-\|x-y\|^p / d)}.$$

We can plug in some values from R^2 and see what happens. Let's try this for various triangles and p = 2 and two values of d.

1. A line of three points: A = (-1,0), B = (0,0), C = (1,0).
   d = 1: $d(A,B) = d(B,C) = \sqrt{2 - 2/e} = 1.124, \quad d(A,C) = \sqrt{2 - 2/e^4} = 1.401$
   d = 1/2: $d(A,B) = d(B,C) = \sqrt{2 - 2e^{-2}} = 1.315, \quad d(A,C) = \sqrt{2 - 2e^{-8}} = 1.401$
   d = 2: $d(A,B) = d(B,C) = \sqrt{2 - 2e^{-1/2}} = 0.887, \quad d(A,C) = \sqrt{2 - 2e^{-2}} = 1.315$

2. A right triangle: A = (-1,0), B = (-1,1), C = (0,0)

   d = 1: $d(A,B) = d(A,C) = \sqrt{2 - 2/e} = 1.124, \quad d(B,C) = \sqrt{2 - 2/e^2} = 1.315$
   d = 1/2: $d(A,B) = d(A,C) = \sqrt{2 - 2e^{-2}} = 1.315, \quad d(B,C) = \sqrt{2 - 2e^{-4}} = 1.401$
   d = 2: $d(A,B) = d(A,C) = \sqrt{2 - 2e^{-1/2}} = 0.887, \quad d(B,C) = \sqrt{2 - 2e^{-1}} = 1.124$

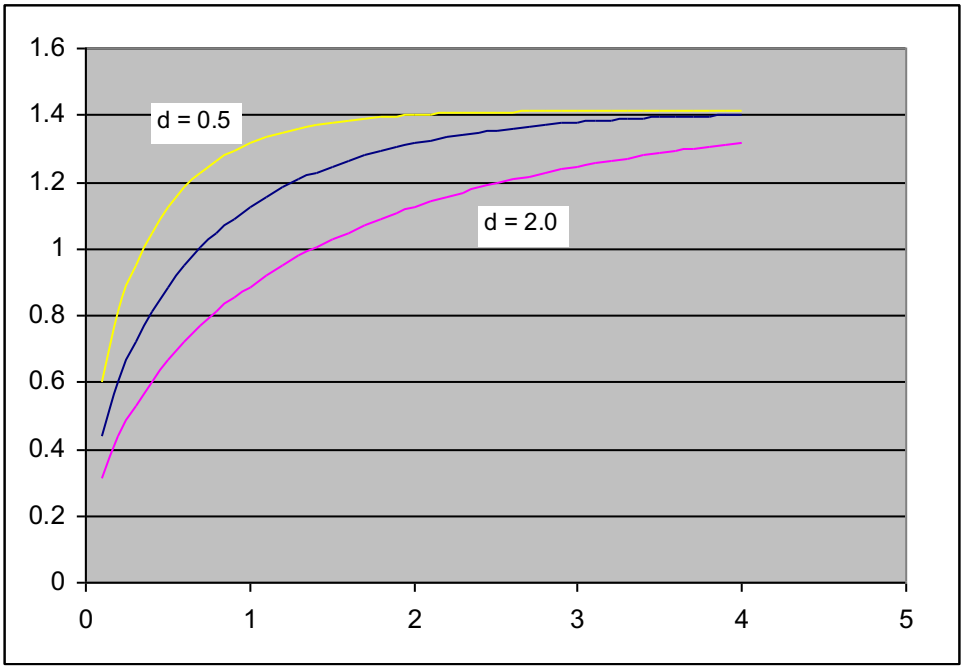
3. A equilateral triangle: A = (0, 0), B = (1, 0), C = (1/2, $\sqrt{3}/2$)

   d = 1: $d(A,B) = d(A,C) = d(B,C) = \sqrt{2 - 2/e} = 1.124$

   d = 1/2: $d(A,B) = d(A,C) = d(B,C) = \sqrt{2 - 2e^{-2}} = 1.315$

   d = 2: $d(A,B) = d(A,C) = d(B,C) = \sqrt{2 - 2e^{-1/2}} = 0.887$

   Graph of the distances in the radial basis space as a function of input distance



Convolution kernels and P-kernels for strings

In these sections, we would like to look at some concepts introduced by D. Haussler in "Convolution kernels and discrete structures", UCSC, Technical Report 99-10, which will form the basis for discussing a paper on protein classification. Haussler's paper discusses all of the below in the much more rigorous manner. Here we just want to get a flavor of the concepts.

Convolution kernels

First remember from the previous sections that kernels are closed under sums, products, and multiplication by a scalar. We start with the notion of a decomposition. A structure X is said to decompose into X_1, X_2, …, X_D, if there is a relation such that R((X_1…X_D), X). (Remember, a relation is like a function that allows many to many associations.) An example of a decomposition is a string S = "ACTTTAT", where the string is decomposed into 7 letters. Suppose on each of the decomposition set, X_i, there is a kernel function $K_i(X_i, X_i)$ on X_i x X_i. Then a convolution kernel on the whole set, S x S is defined as:

$$K(x,y) = \sum_{\text{decompositions of } x \text{ and } y} \prod_{i=1}^{D} K_i(x_i, y_i)$$

This is written as $K(x,y) = K_1 * K_2 * \ldots * K_D$

Suppose each of the X_i's are identical so the decomposition is defined over r copies of X , then we write $K^{(r)}$ to indicate rth convolution over X.

P-kernel

If the sum of the kernel function over all x and y equal one, that is, $\sum K(x,y) = 1$, then we call this a P-kernel.
P-kernels have intuitive interpretation of probabilities over paired configurations. Convex linear combinations of P-kernels are also P-kernels. For example, if K_1 and K_2 are P-kernels then $\alpha K_1 + (1 - \alpha)K_2$ is also a P-kernel. Using this property we can define a particular kind of infinite convex linear combination of kernels:

$$K_\alpha^* = (1-\gamma)\sum_{r=1}^{\infty} \gamma^{r-1} K^{(r)}$$

This is called γ-infinite iteration of K and denotes a geometric convex linear combination of $K^{(r)}$. The reason we make this construction is to associate a class of kernels to the *regular language family* of strings. Let A be some alphabet set such as A = {A,C,G, T} and let Ω be all strings comprised of the alphabet. A regular language is a subset of Ω that is the smallest subset containing empty string, each alphabet, concatenations, unions, and an iterated left concatenation called Kleene star. Given a kernel over the alphabet to each concatenation, we associate a convolution of the kernel of corresponding sub-kernels and we associate the γ-infinite iteration of K to the Kleene star. Then the smallest class of kernels generated this way is called string kernels.

Suppose we are now given the alphabet set A = {A, C, G, T}. We first define two kinds of kernels. First we start with kernel $K_A(x,y) = p(x \mid A)p(y \mid A)$ where p(x | A) is the probability of obtaining the letter x, given some "ancestral" letter A. Then we have the convolution kernel on pairs of letters:

$K_1(x,y) = \sum_{z \in A} p(z) K_z(x,y)$ where *p(z)* is the marginal probability of ancestral letter z. The kernel K_1 can be interpreted as the unconditional joint probability of the letters *x* and *y*.

Next, we define another kernel:

$K_2(X,Y) = g(X)g(Y)$ where X and Y are strings and g(X) is a marginal probability distribution of the string X. It can be any probability distribution as long as it is well defined.

Now for two strings S and T, we define the following γ-infinite iterated kernel on pairs of strings:

$$K_\gamma(S,T) = \gamma K_2 * (K_1 * K_2)_\gamma^* + (1-\gamma)K_2$$

When γ = 0, $K_\gamma(S,T) = K_2(S,T)$ corresponding to marginal insertion probability of strings S and T independent of each other.

If we define K_2 such that $K_2(S,T) = 1$ if both S and T are empty, otherwise $K_2 = 0$, then we do not allow any independent insertions. In this case, $K_\gamma(S,T) = 0$ if the length of S and T are not identical. If the lengths are both r, then $K_\gamma(S,T) = (1-\gamma)\gamma^r \prod_{i=1}^{r} K_1(s_i,t_i)$. Thus, modeling a joint distribution of the letters of the two strings with a geometrically distributed length.

These are special cases of $K_\gamma(S,T)$ and the general case models when the two strings have mixtures of joint distribution of letters and independent insertions.

4.5 Fisher Kernel and HMM

From the discussion so far, it is clear that the idea of a kernel is more interesting and fundamental than the idea of an SVM. From (3.4.2) we have $\psi_0 = \sum_{i=1}^{q} y_i \alpha_i^0 x_i$ with resulting classification function $f(z) = \sum y_i \alpha_i^0 < x_i, z > -b$. For general linear models we have $f(z) = \tau(< z, \theta > + c)$ where τ is the transfer function. For example, for logistic regression $\tau(x) = \frac{1}{1+e^{-x}}$ and *f(z)* denotes the probability of some class. In each of these we can imagine replacing the standard inner product with the more general kernel function. We now consider one kernel function that is obtained from parametric family of probability distributions called Fisher kernels (Jaakola and Haussler, 1998).

Suppose we have a parametric family of p-distribution, $P(X|\theta)$ that is sufficiently smooth. Then, we can define a so-called Fisher's information matrix:

$I_{\theta^*} = E_{\theta^*}(\nabla \log P(X|\theta)^t \nabla \log P(X|\theta))$, where the expectation is taken with respect to the model point θ*. The vector $\nabla \log P(X|\theta)$ is called the score function. Fisher's information matrix also defines a local metric for the manifold of the collection of the parametric family such that it defines a distance between $P(X|\theta^*)$ and nearby model $P(X|\theta^*+d)$ given by $d(\theta^*, \theta^*+d) = \frac{1}{2} d^t I_{\theta^*} d$. The idea is that the score function maps an example X into a feature vector that is a point in the gradient space of the parametric family manifold. The gradient given by the score function can be used to find the direction of steepest ascent along the curvature of the manifold by the direction, $I_{\theta^*}^{-1} \nabla \log P(X|\theta)$. This is called the natural mapping of the examples X. Thus, we obtain an inner product between these feature vectors with respect to local metric and obtain:

$$K(x,y) \approx (\nabla \log P(X|\theta) I_{\theta^*}^{-1})^t I_{\theta^*} (I_{\theta^*}^{-1} \nabla \log P(Y|\theta)) = \nabla \log P(X|\theta)^t I_{\theta^*}^{-1} \nabla \log P(Y|\theta)$$

This is called Fisher's kernel. As can be seen, we can define a Fisher's kernel for any parametric family of p-distributions, for example, HMM families.

A variation on the Fisher's kernel is the Gaussian kernel:

$$D^2(x,y) = (\nabla \log P(x \mid \theta) - \nabla \log P(y \mid \theta))^t I_{\theta^*}^{-1} (\nabla \log P(x \mid \theta) - \nabla \log P(y \mid \theta))$$
$$K(x,y) = \exp(-D^2(x,y))$$

In either case, for application to data, we need to be able to compute the score function. In a HMM, we have the following p-vale for a given sequence:

$$P(X \mid \theta, \gamma) = \sum \prod_i p(x_i \mid s_i, \theta) P(s_i \mid s_{i-1}, \tau)$$

where *x*'s are emission states, *s*'s are hidden states , θ are the emission probabilities, γ are the hidden state transition probabilities, and the outer summation is over all possible hidden states. In real data, we might simplify the problem by considering the score function only with respect to θ's, the emission probabilities, and ignoring the transition probabilities. In a profile HMM for sequences, this corresponds to ignoring the indels. So we need to be able to compute $\frac{\partial}{\partial \theta(x,s)} \log P(X \mid \theta, \gamma)$ where $\theta(x,s)$ is the emission probability of *x* state given *s* hidden state. Recalling that in a profile HMM, we have hidden states [I] = insert, [D] = delete, [M(i)] = match in the ith position, we are only concerned with M(i), since we are ignoring the indels. Then the score function is a vector of $|x| \times n$ elements, where $|x|$ is the cardinality of the output states (=20 for amino-acids) and *n* is the number of match states.

From Jaakola and Haussler we get:

$$\frac{\partial}{\partial \theta(x,s)} \log P(X \mid \theta, \gamma) = \frac{\pi(x,s)}{\theta(x,s)} - \pi(s), \quad \pi(s) = \sum_x \pi(x,s)$$

where $\pi(x,s)$ is the posterior expected frequency of visiting hidden state *s* and generating state *x*. This can be computed from the forward-backward algorithm described in earlier lectures.

Unit 18: Neural networks

see slides

Unit 19: Statistical properties of SVMs and learning machines

6.1 Risk

From section 1, the learning paradigm is that we have a Generator with distribution F(x),
a Supervisor with conditional distribution F(y|x), and the Learning Machine that is given the joint distribution F(x,y). Assume that the learning machine will try to emulate the supervisor's behavior using some set of functions {g(x)}. Furthermore, assume that these functions belong to some parametric family (e.g., SVM where the parameter is the hyperplane vector) with the index set α such that we can say that the available functions are $g_\alpha(x)$. We might quantify the mistake made by a particular emulating function by some positive function $L(y, g_\alpha(x))$. We will call this the Loss function. The idea is that if the true value for the input *x* was *y* and the learning machine's value was $g_\alpha(x)$ there must be some loss involved that can be quantified with $L(y, g_\alpha(x))$. "Error" is a kind of a loss, so loss is just a slightly more general way of discussing errors. From this we can define the expected loss as so-called ***risk***:

$$R(\alpha) = \int L(y, g_\alpha(x))dF(x) \qquad (6.1.1)$$

The quantity (6.1.1) is something called the ***true risk*** because we are integrating over the distribution *F(x)*, which is the distribution of the Generator. Thus, it includes all the possible observations. However, with real data, we only have a finite set of training data so we define the ***empirical risk***:

$$R_{emp}(\alpha) = \frac{1}{n}\sum_i L(y_i, g_\alpha(x_i)). \qquad (6.1.2)$$

That is, the expectation is taken as average of the finite observations. The risk is defined with respect to the variable α since we want to assess the cost of choosing some emulating function. (BTW, this use of the term risk is slightly different from the usual one in parameter estimation problems.) Empirical risk is something we know because it is computed over the observed data. In fact, many algorithms can be said to be "minimizing empirical risk". True risk is something we don't know because we can never directly know *F(x)*, nor can we really get an infinite sample. Then, the whole name of the game is whether we can say something about true risk (6.1.1) when all we know is empirical risk (6.1.2). If anything goes, then obviously we can say nothing. However, if we are allowed to make a reasonable assumption that the training data is like an unbiased sample from F(x), then we can say a lot of things. First, we enumerate the kinds of statements we would like to get between true risk and empirical risk, then in the advanced section we show how to derive such statements for SVMs. For the basic section, we only discuss one of the ideas called cross-validation.

So, from this point, we will be concerned with the behavior of the empirical risk in relation to risk. That is, suppose we choose a particular function, say $g_{\alpha_n^*}(x_n)$, for each n-sized dataset. For example, we might choose the function to minimize the empirical risk (5.1.1). What is the value of the true risk, $R(\alpha_n^*)$ and how does it differ from the empirical risk $R_{emp}(\alpha_n^*)$? Some of the relationships we are interested in are:

1. $\Pr ob(|R(\alpha_n^*) - R_{emp}(\alpha_n^*)| > \varepsilon),\ n \to \infty$. If this quantity goes to zero for every $\varepsilon > 0$, then we say our procedure for choosing the function is consistent. This is somewhat fancy way of saying that as the amount of empirical (training) data goes to infinity, the empirical risk will converge to true risk. It may seem strange that this is not always true, but it turns out that this kind of consistency property is not always met.

2. $\Pr ob(|R(\alpha_n^*) - R_{emp}(\alpha_n^*)| > \varepsilon) \le \eta(\varepsilon, n)$, the convergence bound of the absolute difference between the two risks as a function of *n* and difference ε. What this means is the following. Suppose for a finite sample size *n* the algorithm learns from the data and makes a particular choice of parameters; we are denoting this by α_n^*. Then for such a choice of α_n^* there will be a difference between true risk and empirical risk. If we repeat this for other samples of size *n*, we will get a distribution of the difference between true risk and empirical risk. We would like to be able to say something about this difference distribution, for example what is the probability that the difference is bigger than ε. Ideally, we would like to know the exact probability, but in many cases this is difficult to get so we would like to get at least an upper bound; i.e., "probability is not more than X". This upper bound will obviously depend on the sample size and the choice of ε and so we denote it as $\eta(\varepsilon, n)$.

3. $R(\alpha_n^*) \le R_{emp}(\alpha_n^*) + \phi(\alpha, n)$, bound on the true risk as a function of (known) empirical risk and some quantity that is dependent on the sample size *n* and confidence level α. This is very similar to item 2, but here

what we would like is not a distribution over many different samples of size n. But, rather for a particular finite sample and computed value of empirical risk what can we say about the upper bound on true risk?

4. Suppose $g_{\tilde{\alpha}}(x)$is the function that minimizes the true risk. $R(\alpha_n^*) - R(\tilde{\alpha}) \le \phi(\eta, n)$, bound on the difference in the true risk of a function obtained from some choice procedure on observed data and the true risk of a function that minimizes risk for all possible data. This is asking whether we can say something about the difference between the true risk when the algorithm has a finite amount of data to optimize and when it has infinite amount of data to optimize.

A warm up problem

Here we will consider a simple case where the loss functions, $L(y, g_\alpha(x))$, indexed by α, are indicator functions such that L = 0 on some subset of x and L = 1 elsewhere. We also suppose that our choice of possible functions, g_α, is finite indexed as g_{α_i} $i = 1,\ldots,N$.

First, a statement called Chernoff bound.

Chernoff inequality: Let f_n be the frequency of success for *n* independent Bernoulli trials with probability of success *p*. Then,

$$P\{p - f_n > \varepsilon\} < \exp(-2\varepsilon^2 n)$$
$$P\{f_n - p > \varepsilon\} < \exp(-2\varepsilon^2 n)$$

.

Now we consider the following inequalities using Chernoff bound.

$$P\{\sup_i(R(\alpha_i) - R_{emp}(\alpha_i)) > \varepsilon\} \le \sum_i^N P\{R(\alpha_i) - R_{emp}(\alpha_i) > \varepsilon\} \le N\exp(-2\varepsilon^2 n) \quad (6.2.1)$$

We can rewrite (6.2.1) by first setting $\eta = N\exp(-2\varepsilon^2 n)$, implying $\varepsilon = \sqrt{\frac{\ln N - \ln \eta}{2n}}$. Now we can say with probability 1 - η simultaneously for all g_α,

$$R(\alpha_i) - R_{emp}(\alpha_i) \le \sqrt{\frac{\ln N - \ln \eta}{2n}} \qquad (6.2.2)$$

The bound (6.2.2) is the convergence bound for any choice functions g_α.

Suppose g_{α^*} is the function that minimizes empirical risk and $g_{\tilde{\alpha}}$ is the function that minimizes real risk. Since (6.2.2) holds for all functions, we have,

$$R(\alpha^*) \le R_{emp}(\alpha^*) + \sqrt{\frac{\ln N - \ln \eta}{2n}} \qquad (6.2.3)$$

This bound, (6.2.3) says if we were to choose a function based on minimizing the empirical risk function, we can bound the real risk by the quantity on the right. Thus, it provides a bound on generalization of the choice—since real risk is risk over all possible input data (weighted by respective probability measure).

Next, Chernoff inequality holds for the function $g_{\tilde{\alpha}}$, so we have

$$P\{R_{emp}(\tilde{\alpha}) - R(\tilde{\alpha}) > \varepsilon\} \le \exp(-2\varepsilon^2 n)$$
$$\Rightarrow R(\tilde{\alpha}) \ge R_{emp}(\tilde{\alpha}) - \sqrt{\frac{-\ln\eta}{2n}} \qquad (6.2.4)$$

Since g_{α^*} minimizes empirical risk we also have $R_{emp}(\tilde{\alpha}) - R_{emp}(\alpha^*) \ge 0$. Taking this into consideration with (6.2.3) and (6.2.4), we have

$$R(\alpha^*) - R(\tilde{\alpha}) \le \sqrt{\frac{\ln N - \ln\eta}{2n}} + \sqrt{\frac{-\ln\eta}{2n}} + R_{emp}(\alpha^*) - R_{emp}(\tilde{\alpha}) \le \sqrt{\frac{\ln N - \ln\eta}{2n}} + \sqrt{\frac{-\ln\eta}{2n}} \qquad (6.2.5)$$

The bound (6.2.5) says that if we choose the function g_{α^*}, then the difference between the true risk for this function and the minimum true risk is bounded by the quantity on the right. In other word, it measures how the strategy of empirical risk minimization compares to the case if we had all the information.

Entropy, Annealed entropy, and Growth function

We now define some quantities useful for discussing bounds on risk:

1. ***Entropy of an indicator set function***. Let I(x, α) be a α-collection of indicator functions on the set x. Suppose we are given a training set $T = \{x_1,...x_n\}$ of n vectors. Now consider a binary vector of the result of applying some indicator function I_α to the training set, so we have $q(\alpha) = (I_\alpha(x_1),\cdots,I_\alpha(x_n))$. Suppose we vary α, that is, try out different indicator functions, then we will have different binary vectors q. Let $N^\alpha(x_1\cdots x_n)$ be the number of different binary vectors. Then, it is clear that $N^\alpha \le 2^n$. We now define, random entropy of a set of indicator functions as:

$H^\alpha(x_1\cdots x_n) = \log N^\alpha(x_1\cdots x_n)$ and entropy (or expected entropy) as $H^\alpha(n) = \int H^\alpha(x_1...x_n)dF(x_1...x_n)$.

2. ***Entropy of real-valued functions.*** We can now extend the above definition to real-valued functions. Suppose now we are working with a bounded family of functions $|g_\alpha(x)| < C$. We again consider some training set $T = \{x_1,...x_n\}$ and now the real-valued vector $q(\alpha) = (g_\alpha(x_1),\cdots,g_\alpha(x_n))$. Now these vectors are contained within some volume. We will try to "discretize" this volume of points by defining a finite ε-net as a finite collection of points, E, such that $d(E,q(\alpha)) < \varepsilon$ where d(.) is a defined norm. Now we define the minimal finite ε-net of the set $q(\alpha) = (g_\alpha(x_1),\cdots,g_\alpha(x_n))$ as the ε-net with the smallest cardinality (number of points in E). Then let $N^\alpha(x_1\cdots x_n)$ be the size of such a minimal finite ε-net, random entropy and entropy is defined exactly like the previous case.

3. ***Annealed entropy and Growth function.*** Once random entropy and entropy are defined we can define two more quantities:

Annealed entropy: $H^{\alpha}_{ann}(n) = \ln E(N^{\alpha}(x_1 \cdots x_n))$

Growth function: $G^{\alpha}(n) = \ln \sup_x N^{\varepsilon}(x_1 \cdots x_n)$

It is easy to see: $H^{\alpha}(n) \le H^{\alpha}_{ann}(n) \le G^{\alpha}(n)$

For indicator functions, we had $N^{\alpha} \le 2^n$. So the growth function is bounded by nlog2. In fact, it can be shown:

$$G^{\alpha}(n)\begin{cases} = n\ln 2 \quad if\ n \le h \\ \le \ln(\sum_{i}^{h} C(n,i)) \le \ln(\frac{en}{h})^{h} = h(1+\ln\frac{n}{h}) \quad if\ n > h \end{cases}$$

where h is the largest integer for which $G^{\alpha}(h) = h\ln 2$. This h is called the VC dimension of the set of indicator functions. The VC dimension of indicator functions corresponds to the largest number of training vectors that can be split into all possible 2-way classification. VC dimension is also said to indicate the maximum number of training vectors that can be shattered by the indicator functions. VC dimension may be infinite.

Bounds for infinite collection of indicator functions

Inequalities, (6.2.1)-(6.2.5) gave bounds for the case of finite number of learning function choices. We now present the bound for the case of infinite number of learning functions (e.g., SVM separating hyperplane).

Theorem:

$$P\{\sup_{\alpha} |\, R(\alpha) - R_{emp}(\alpha) \,| > \varepsilon\} < 4\exp\{(\frac{H^{\alpha}_{ann}(2n)}{n} - (\varepsilon - \frac{1}{n})^2)n\} \qquad (6.4.1)$$

□

Corollary:

If $\lim_{n\to\infty} \frac{H^{\alpha}_{ann}(n)}{n} = 0$ then a nontrivial exponential bound exists on uniform convergence.

□

(6.4.1) can be written in equivalent form as:

With probability at least $1-\eta$ simultaneously for all functions, g_{α}, the following holds:

$$R(\alpha) \le R_{emp}(\alpha) + \sqrt{\frac{H^{\alpha}_{ann}(2n) - \ln\eta/4}{n}} + \frac{1}{n} \qquad (6.4.2)$$

As before if g_{α^*} is the function that minimizes empirical risk and $g_{\tilde{\alpha}}$ is the function that minimizes real risk, we first have

$$R(\alpha^*) \le R_{emp}(\alpha^*) + \sqrt{\frac{H^{\alpha}_{ann}(2n) - \ln \eta/4}{n}} + \frac{1}{n} \qquad (6.4.3)$$

and using (5.2.4) again, we have

$$R(\alpha^*) - R(\tilde{\alpha}) \le \sqrt{\frac{H^{\alpha}_{ann}(2n) - \ln \eta/4}{n}} - \sqrt{\frac{-\ln \eta}{2n}} + \frac{1}{n} \qquad (6.4.4)$$

These inequalities generalize the previous results for the infinite function collection case and answers the bounds questions (1) – (4). However, in practice, H^{α}_{ann} is not a function we can easily compute since we do not know the distribution of the data generator (for taking the expectation). However, we can use $H^{\alpha}(n) \le H^{\alpha}_{ann}(n) \le G^{\alpha}(n)$ set of inequalities and replace H^{α}_{ann} by G^{α}. The growth function, G^{α}, is in turn upper bounded by a function of the VC dimension (by definition). Let h be the VC dimension of a learning machine, then,

$$G^{\alpha} < h(1 + \ln \frac{n}{h}) \qquad (6.4.5)$$

We can now write a distribution free version of (6.4.2) as:

$$R(\alpha) \le R_{emp}(\alpha) + \sqrt{\frac{h(1 + \ln \frac{n}{h}) - \ln \eta/4}{n}} + \frac{1}{n} \qquad (6.4.6)$$

and similar bounds can be written distribution free by using the VC dimension bound on the growth function.

We do not prove (6.4.1), but give an outline of the proof to see why the concept of entropy comes in. First we need one lemma that we state as part of a more general idea.

Cross Validation

In many cases, when we are presented with sufficient training data we can come up with a scheme for assessing something about true risk by reserving a part of the training data and optimizing on only part of the data. We imagine that we have *2n* observations and compute the risk for 1 to *n* and then *n*+1 to 2*n* separately. Our strategy is to compute empirical risk separately on the two halves of the risk and look at the difference. We define the maximum of the difference in the two risks:

$$\rho^{\alpha}(2n) = \sup_{\alpha} | R_{emp}(\alpha, 1:n) - R_{emp}(\alpha, n+1:2n) |$$

where $R_{emp}(\alpha, 1:n)$ means computing the empirical risk for function α over observations 1 to *n*. The lemma states,

Cross-Validation Lemma:

$$P\{\sup_{\alpha} | R(\alpha) - R_{emp}(\alpha, 1:n) | > \varepsilon\} < 2P\{\rho^{\alpha} > \varepsilon - \frac{1}{n}\} \quad (6.4.7)$$

□

This just tells us that we should study the difference in the empirical risk of two halves of an observation set to obtain information on the bound for the difference in true risk and empirical risk. The lemma is technical (so you don't have to understand it in detail) and basically tells us that by seeing how the results on two halves of a dataset matches, we can put a bound on how the current best model might perform on future—what we called true risk. However, in practice, people use cross-validation more heuristically. Here is the heuristic formulation of cross-validation

1. Split the input data into two random parts in x:y ratio. A typical ratio is 5:1 ratio, which is sometimes called 5-fold cross-validation.
2. Optimize the learning machine on part x. For example, fit a SVM class prediction model on the x part of the input data.
3. Use the optimized model to predict the left-out y part of the data. Compute the error rate. Call this the prediction error.
4. Repeat 1-3 a large number of times, say 100 or 1000, and find the average prediction error. Use the average prediction error as a heuristic for the true risk of the learning machine.

Suppose we were given a fixed sample of *2n* observations, then even if we have an infinite collection of indicator functions, g_{α}, we only have a finite number, in fact, $N^{\alpha}(x_1 \cdots x_{2n})$ number of different indicator functions on this input set. Thus similar to the sequence (6.2.1), we have

$$P\{\rho^{\alpha}(2n) > \varepsilon\} \le \sum_{i}^{\infty} P\{\rho^{\alpha}(2n) > \varepsilon\} \le N^{\alpha}(x_1 \cdots x_{2n}) P\{\rho^{\alpha}(2n) > \varepsilon\} \quad (6.4.8)$$

The last probability can be obtained from standard results as $P\{\rho^{\alpha}(2n) > \varepsilon\} < 2\exp(-\varepsilon^2 n)$ and we get

$$P\{\rho^{\alpha}(2n) = \sup_{\alpha} | R_{emp}(\alpha, 1-n) - R_{emp}(\alpha, n+1-2n) | > \varepsilon\} \le 2N^{\alpha}(x_1 \cdots x_{2n}) \exp(-\varepsilon^2 n) \quad (6.4.9)$$

(6.4.9) is for a fixed sample of size *2n*, if we were to consider a random sample, we only need to take expectations,

$$P\{\sup_{\alpha} | R_{emp}(\alpha, 1-n) - R_{emp}(\alpha, n+1-2n) | > \varepsilon\} \le 2EN^{\alpha}(x_1 \cdots x_{2n}) \exp(-\varepsilon^2 n)$$

$$= \exp\{(\frac{H_{ann}^{\alpha}(2n)}{n} - \varepsilon^2) n\}$$

Statistical properties of separating hyperplanes.

Suppose we now have a Δ-margin separating hyperplane and the input vectors are bounded by the radius R. Then the VC dimension of the separating hyper plane is bounded by

$$h \le \min\left(\left\lfloor \frac{R^2}{\Delta^2} \right\rfloor, n\right) + 1 \quad (6.5)$$

(6.5) tells us that the VC dimension gets smaller, the larger the margin Δ. (6.4.6) tells us that smaller VC dimension will results in tighter bounds on the real risk (error).